Itsuki Noda
Noriaki Ando
Davide Brugali
James J. Kuffner (Eds.)

LNAI 7628

Simulation, Modeling, and Programming for Autonomous Robots

Third International Conference, SIMPAR 2012
Tsukuba, Japan, November 2012
Proceedings

Springer

Itsuki Noda   Noriaki Ando
Davide Brugali   James J. Kuffner (Eds.)

Simulation, Modeling, and Programming for Autonomous Robots

Third International Conference, SIMPAR 2012
Tsukuba, Japan, November 5-8, 2012
Proceedings

Springer

Volume Editors

Itsuki Noda
Noriaki Ando
National Institute of Advanced Industrial Science and Technology
AIST Tsukuba Central 2, Tsukuba, Ibaraki 305-8568, Japan
E-mail: {i.noda, n-ando}@aist.go.jp

Davide Brugali
University of Bergamo, School of Engineering
v.le Marconi 5, 24044 Dalmine, Italy
E-mail: brugali@unibg.it

James J. Kuffner
Google Inc., Google Research
1600 Amphitheatre Parkway, Mountain View, CA 94043, USA
E-mail: kuffner@google.com

ISSN 0302-9743 e-ISSN 1611-3349
ISBN 978-3-642-34326-1 e-ISBN 978-3-642-34327-8
DOI 10.1007/978-3-642-34327-8
Springer Heidelberg Dordrecht London New York

Library of Congress Control Number: 2012949408

CR Subject Classification (1998): I.2.9-11, I.2, I.6, H.3, F.1, D.2, C.2, H.4-5

LNCS Sublibrary: SL 7 – Artificial Intelligence

Typesetting: Camera-ready by author, data conversion by Scientific Publishing Services, Chennai, India

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

Preface

The application fields of autonomous robots are considered to be getting wider and wider. Living and office spaces will be the most promising domain for autonomous robotics, in which robots should complete complicated and intelligent tasks under uncertain environments including human behaviors. Disaster areas and deep space are also important application domains, in which robots are required to behave flexibly and (semi-)automatically against unexpected situations.

No-one doubts the importance of software for such robotics applications. Simulation is required to design complex behaviors of robots and to confirm the stability and safety of action plans. Modeling of robots and environments is a necessary part of developing autonomous robotic systems. Programming tools and libraries are the most active area in the development of robotics research. Many projects of autonomous robots have recently started from preparing such software platforms.

The series of International Conference on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR) is organized to foster research in the above topics. Gathering the most recent works in this field enhances re-usability of software for robotics and pushes research forward swiftly.

The third SIMPAR 2012 was held during November 5–8, at the National Institute of Advanced Industrial Science and Technology in Tsukuba, Japan. It followed the previous works of the first SIMPAR 2008 in Venice, Italy, and the second SIMPAR 2010 in Darmstadt, Germany, and provided a forum for free and concentrated discussions on the topics of interest.

This book collects 34 contributed papers, selected among a total 46 submissions. Ten papers describe the design of complex behaviors of autonomous robots, nine are on software layers, eight papers are related to modeling and learning, and six are simulation-related works. Each submitted paper received at least two reviews by the members of a carefully selected international Program Committee.

We also had three impressive invited talks presented by Yoshiyuki Sankai, Jean-Paul Laumond, and Michael Beetz, which discussed the subject field in relation to clinical application, cognitive science, and artificial intelligence.

We want to gratefully thank the Program Committee members and all other supporters, organizers, and volunteers who contributed for SIMPAR. Without their effort, it would be impossible to hold this important conference.

November 2012

Noriaki Ando
Davide Brugali
James Kuffner
Itsuki Noda

Organization

Executive Committee

General Chair

Itsuki Noda	AIST, Japan

International Program Co-chairs

America

James Kuffner	Google Inc., USA

Asia

Noriaki Ando	AIST, Japan

Europe

Davide Brugali	University of Bergamo, Italy

Publicity Co-chairs

Tetsuo Kotoku	AIST, Japan
Hyun Kim	ETRI, Korea

Publication Co-chairs

Oskar von Stryk	Technische Universität Darmstadt, Germany
Shuichi Nishio	ATR, Japan

Workshop and Tutorial Co-chairs

Emanuele Menegatti	University of Padua, Italy
Geoffrey Biggs	AIST, Japan

Financial Chair

Yasushi Nakauchi	University of Tsukuba, Japan

Exhibition Chair

Hiroyuki Okada	Tamagawa University, Japan

Web Co-chairs

Yosuke Matsusaka	MID Academic Promotions Inc., Japan
Kenji Suzuki	University of Tsukuba, Japan

Local Arrangements Co-chairs

Tetsuo Kotoku	AIST, Japan
Kenichi Ohara	Osaka University, Japan

Steering Committee

Tamio Arai	University of Tokyo, Japan
Herman Bruyninckx	Katholieke Universiteit Leuven, Belgium
Xiaoping Chen	University of Science and Technology of China, China
Maria Gini	University of Minnesota, USA
Enrico Pagello (Founding Chair)	University of Padua, Italy
Lynne Parker	University of Tennessee, USA
Oskar von Stryk	University of Darmstadt, Germany

Program Committee

Asia

Sang Chul Ahn	KIST, Korea
Joschka Boedecker	Osaka University, Japan
Kosei Demura	Kanazawa Institute of Technology, Japan
Toshio Hori	AIST, Japan
Seung-Woog Jung	ETRI, Korea
Takayuki Kanda	ATR, Japan
Hyun Kim	ETRI, Korea
Joo-Ho Lee	Ritsumeikan University, Japan
Bruce MacDonald	Auckland University, New Zealand
Takashi Minato	ATR, Japan
Kazuyuki Morioka	Meiji University, Japan
Mihoko Niitsuma	Chuo University, Japan
Kei Okada	University of Tokyo, Japan
Hiroyuki Okada	Tamagawa University, Japan
Jun Ota	University of Tokyo, Japan
HongSeong Park	Kangwon National University, Korea
Masayuki Shimizu	Shizuoka University, Japan
Kai-Tai Song	National Chiao Tung University, Taiwan
Yuki Suga	Waseda University, Japan
Masaki Takahashi	Keio University, Japan
Sasaki Takeshi	Shibaura Institute of Technology, Japan
Kazuyoshi Wada	Tokyo Metropolitan University, Japan
Hongxing Wei	Beyhang University, China
Mary-Anne Williams	University of Technology, Sydney, Australia
Hiroaki Yaguchi	University of Tokyo, Japan

Europe

Levent Akin	Boğaziçi University, Turkey
Rachid Alami	LAAS/CNRS, France
Berthold Baeuml	DLR/Institute of Robotics and Mechatronics, Germany
Jan Bender	Technische Universität Darmstadt, Germany
Mirko Bordignon	University of Southern Denmark, Denmark
Andreu Corominas Murtra	Beta Robotics, Spain
Alessandro Farinelli	Università di Verona, Italy
Alexander Ferrein	RWTH Aachen University, Germany
Holger Giese	Hasso Plattner Institute, Germany
Giuseppina Gini	Politecnico di Milano, Italy
Martin Huelse	University of Wales, UK
Luca Iocchi	University of Rome “La Sapienza”, Italy
Alberto Jardon	University of Carlos III of Madrid, Spain
Daniel Kubus	Technische Universität Braunschweig, Germany
Konrad Kułakowski	Institute of Automatics, AGH UST, Poland
Reinhard Lafrenz	Technische Universität München, Germany
Jacques Malenfant	Université Pierre et Marie Curie, France
Luis Manso	University of Extremadura, Spain
Pavel Petrovic	Comenius University, Slovakia
Piotr Trojanek	Warsaw University of Technology, Poland

America

Stephen Balakirsky	NIST, USA
Stefano Carpin	University of California Merced, USA
John Hsu	Open Source Robotics Foundation and Willow Garage, USA
Michael Quinlan	Clover Network Inc., USA
Javier Ruiz del Solar	Universidad de Chile, Chile
Mohan Sridharan	Texas Tech University, USA

Table of Contents

A Geometric Perspective of Anthropomorphic Embodied Actions

Jean-Paul Laumond

LAAS-CNRS, Toulouse, France

Abstract. Starting from a mechanics point of view, the human (or humanoid) body is both a redundant system and an underactuated one. It is redundant because the number of degrees of freedom is much greater than the dimension of the tasks to be performed: around 640 muscles for humans and 30 motors for humanoid robots. It is underactuated because there is no direct actuator allowing the body to move from one place to another place: to do so human and humanoid robots should use their internal degrees of freedom and actuate all their limbs following a periodic process (named bipedal locomotion!).

By considering first that motions are continuous functions from time to space (i.e. trajectories), and second that actions are compositions of motions, actions appear as sequences of trajectories. The images of the trajectories in spaces are named paths. Paths represent geometric traces left by the motions in spaces. The reasoning holds for the real space, the configuration and the control space. Therefore actions appear as continuous simple paths in high dimensional spaces.

A simple path embodies the entire action. It integrates into a single data structure all the complexity of the action. The decomposition of the action into sub-actions (e.g., walk to, grasp, give) appears as the decomposition of the path into sub-paths. Each elementary sub-path is selected among an infinite number of possibilities within some sub-manifolds (e.g., grasp fast or slowly, grasp while bending the legs or not).

All complex cognitive and motor control processes that give rise to an action in the real world are reflected by the structure of paths in the body control space. In this framework, symbols may be defined as sub-manifolds that partition the control space. Such a partition decomposes paths into sub-paths. From this perspective the questions are:

- Motion Segmentation: what are the invariant sub-manifolds that define the structure of a given action?
- Motion Generation: among all the solution paths within a given sub-manifold (i.e. among all the possibilities to solve a given sub-task) what is the underlying law that converges to the selection of a particular motion?

The talk overviews recent results obtained in this framework (including whole body manipulation, locomotion trajectory generation, action recognition) and illustrated from the HRP2-14 humanoid platform.

Cybernics: Fusion of Human, Machine and Information
Robot Suit for the Future

Yoshiyuki Sankai

Center for Cybernics Research (CCR), Univ. of Tsukuba, Japan
Dept. of System & Information Engineering, Univ. of Tsukuba, Japan
FIRST Program on Cybernics Research, JSPS, Cabinet Office, Japan
CYBERDYNE Inc. Japan

Abstract. Cybernics is a frontier science centered on cybernetics, mechatronics and informatics, and is a new domain of interdisciplinary academic field of human-assistive technology to support, enhance and expand human's physical/cognitive functions, which challenges to integrate and harmonize humans and robots (RT: robotics technology) with the basis of information technology (IT) in a functional, organic, and social manner, based on several areas of science and technology such as neuroscience, physiology, robotics, computer science, medicine, behavioral science, ethics, safety engineering, psychology, cognitive science and social science. A pioneering achievement is Robot Suit HAL (Hybrid Assistive Limbs), which is the world's first cyborg type robot that supports, enhances and strengthens the physical motion of human with the interaction between human and robot by detecting the weak bioelectrical signal through the body from the brain, which generates the nerve signal to control the musculoskeletal system. In this talk, I will deliver the outline of Cybernics and mention about clinical applications for stroke patients, SCI patients, and severe incurable disease such as Neuro-Muscular disease. And I will introduce some remarkable works including new applications of HAL and Vital Sensing System based on Cybernics technologies.

Keywords: Cybernics, Robot Suit, HAL, Interactive HAL, Medicare.

Robot Suit for the Future
Challenges for the Future !
Leading Edge of Robot Suit HAL and Clinical Applications."
Robotics that suit you!
筑波大学
Center for Cybernics Research

If Abstraction Is the Answer, What Is the Question? — Reasoning for Everyday Manipulation Tasks

Michael Beetz

Artificial Intelligence
Faculty of Computer Science and
Center for Computing and Communication Technologies (TZI)
University of Bremen, Germany

Abstract. In recent years we have seen tremendous advances in the mechatronic, sensing and computational infrastructure of robots, enabling them to act faster, stronger and more accurately than humans do. Yet, when it comes to accomplishing manipulation tasks in everyday settings, robots are still far away from reaching the sophistication and performance of humans. A key component of the "action intelligence" needed for reaching such a level of sophistication and performance are robot control systems that can take vague action descriptions and automatically infer how they are appropriately executed in a given task and environment context.

Artificial Intelligence (AI) is the research discipline that has studied such reasoning problems for more than fifty years. Researchers in AI have investigated naive physics reasoning, temporal projection, reasoning about action and change, action planning, spatial reasoning, to name only a few. Unfortunately, the proposed methods have not yet achieved their desired impact on autonomous robot control. We believe that one of the reasons is that most AI researchers consider perception and action to be the mere input and output of symbolic reasoning. In contrast, some researchers in cognitive psychology suggest a "simulation model" for reasoning through actions. In their view predicting the consequences of actions is very similar to executing the actions without causing physical effects — perception and action get simulated at a very fine-grained feedback loop.

In this talk I will present reasoning techniques of an autonomous robot control system that are inspired by the "simulation model" for reasoning through actions. These techniques use perception and motor control mechanisms and simulations thereof not only as input and output but more importantly also as resources for symbolic reasoning. I will show, using an autonomous robot making pancakes as an example, that such techniques reason about actions more realistically and thereby enable the robot to improve its performance.

Towards Partners Profiling in Human Robot Interaction Contexts

Salvatore M. Anzalone, Yuichiro Yoshikawa, Hiroshi Ishiguro[1], Emanuele Menegatti, Enrico Pagello[2], and Rosario Sorbello[3]

[1] Intelligent Robotics Laboratory, Dept. of Systems Innovation, Graduate School of Engineering Science, Osaka University
[2] Intelligent Autonomous Systems Laboratory, Dept. of Information Engineering, Faculty of Engineering, University of Padua
[3] Dept. of Chemical, Management, Computer and Mechanical Engineering, Faculty of Engineering, University of Palermo

Abstract. Individuality is one of the most important qualities of humans. Social robots should be able to model the individuality of the human partners and to modify their behaviours accordingly.This paper proposes a profiling system for social robots to be able to learn the individuality of human partners in social contexts. Profiles are expressed in terms of of identities and preferences bound together. In particular, people's identity is captured by the use of facial features, while preferences are extracted from the discussion between the partners. Both are bound using an Hebb network. Experiments show the feasibility and the performances of the approach presented.

Keywords: profiling, personal robots, human robot interaction.

1 Introduction

Human beings are social animals. People are individuals but are also members of a group. Our behaviours, our actions are not only highly influenced by our society, but are also capable of influencing the society itself, creating a strictly, deep connection between each single individual and the others. Moreover, social capabilities have an extremely significant role in our evolution, in our development and education. Sociability influences our deep human qualities, like identity, friendship, empathy and also emotion. It is not possible to understand human nature without considering our social capabilities. Classic robotics has been focused on making mobile robots completely autonomous, capable of exploring, moving and accomplishing missions in a safe way, inside real environments. According to this approach, robots are something like a tool or a sort of "intelligent instrument" employed to achieve missions that are too hazardous for humans to carry out, dangerous tasks in a remote, unreachable environment. This use of robots does not catch the sense of a real partnership between robots and humans because they always identify a master-slave relationship between them [1]. Despite of this, robots can collaborate in a strict way with humans as a social, cooperative and capable partner: not only as systems able to perceiving and acting in order to extend human productivity, but also as interactive and communicative subjects, reliable working partners [2]. However, going social is not enough [3]. Humans consider themselves individuals with an unique personality, with personal preferences, and with a social role, so they expect to be treated likewise. Robots should interact with people in a "personal" way. To be perceived as active and effective partners, truly accepted by humans, social robots need to learn how to relate with the individuality of single persons: sociable robots must be able to identify and represent human partners according to their physical features, their preferences and their social relations to adapt their behaviours, vocabulary, and social rules according to the individuals that are involved in the interaction.

2 System Overview

The system proposed in this paper tries to model the profiles of human partners. As depicted in figure 1, a robot is involved in the interactions between two human users: while people discuss, the robot is able to capture the information related to their identity and the data about the current discussion. Profile of each partner is modelled by bounding together this information. A person's profile can be described as a conjugation of several kinds of different features. A characterization of a face can be used to depict the identity of a human partner, but this can be improved using other related descriptors such as the voice. Using different kind of features is possible to obtain a more reliable model of the appearance of the partners, according to their physical cues. Through this characterization the robot is able to recognize the identity of its human partners. In the work presented on this paper only facial cues are considered. Then, the profile is completed by joining to this physical characterization the preferences of the partners, by analysing the conversations between them and by finding their topics: preferences can be seen as the most recurrent topics discussed by each partner. Finally, the robot can be enabled to adapt its behaviour according to this information perceived. Several software modules have been built in order to accomplish the main task, as shown in figure 2. Data from a camera is collected by a face recognition system and processed to extract faces, if any, and, from them, facial features. Then, such kind of features are classified together in order to obtain identity claims of the partners that share the environment with

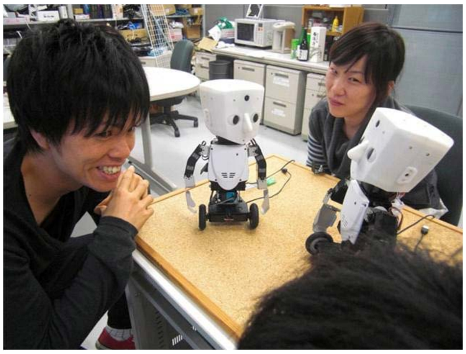


Fig. 1. A typical setup of the system

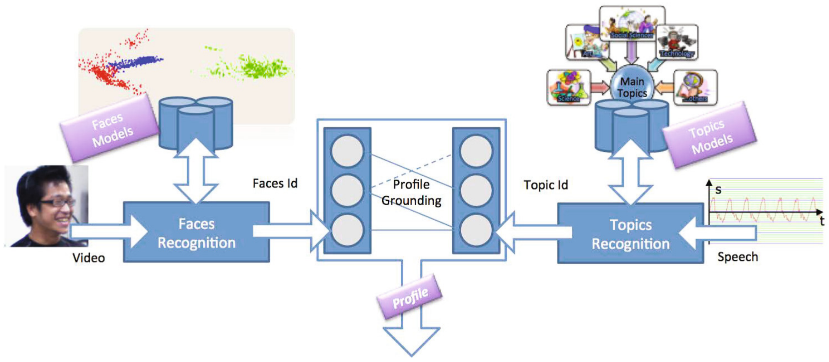


Fig. 2. An overview of the system

the robot. On the other side, through a speech recognition system, sound is processed to extract the utterances of the conversation [4]. Then, the topic recognition system tries to deduce the topic of the current conversation through a statistical characterization of these sentences. Finally, a profiling system will bound identities and topics together and will store this information as a model of the profile of each person.

3 People Identification

Identification of partners in a human-robot interaction context can be achieved using different ways, such as voice identification, face recognition, and so on [5]. While this can be an easy task in controlled environments, it becomes a challenging problem in daily life applications in unstructured environments. Focusing on vision based systems, features retrieved by a camera are usually very noisy, can change if the person in front of the camera changes pose, and accordingly to the light conditions of the environment. Such circumstances will strongly affect the recognition results. Furthermore, recognition results are also affected by non verbal communication, such as showing emotional states or social behaviours. But this is not enough; in long term interactions these problems become bigger because people can alter their appearance. tis possible to think about a robot that should interact with the same woman with and without make up, or with the same man with and without a beard. The approach here presented achieves partner identification using visual information. As shown in figure 3, data from the environment is perceived through the camera of the robot. This raw information, is analysed to detect humans features that will be extracted and collected. Facial features are retrieved through the use of Eigenfaces applied to the visual information. The features collected represent a biological signature of each person, so they are opportunely classified in a supervised way to obtain the identification claims. n detail, the face features extraction process is subdivided in two main steps: in a first phase the camera data is processed to find faces. The Viola-Jones classifier is an efficient detection system that tries to find features, called Haar-like features, that encode the existence of oriented contrast in different regions of the image [6]. A set of this kind of features has been chosen using different pictures of faces taken under the same lighting conditions and normalized to line up the eyes and mouths: these features encoded the contrasts and the special relationships showed by human faces. The Haar classifier is trained to detect and localize faces inside the images taken in the same conditions of the training set. Faces found by the Viola-Jones detector are processed in order to find a model capable of describing the identities of people, by extracting the most relevant information contained in them. The Principal Component Analysis offers theory concepts to achieve this, in particular using the technique of the Eigenfaces [7]. According to this approach, a small set of pictures is used to calculate some vectors, called Eigenfaces, that define a space that best encodes the variations of between faces. This space is called Eigenspace. In this project, the eigenspace has been built using a standard database, YaleFaces, to represent a generic space capable of describing the features of many kinds of faces [8]. The features of the faces seen are the eigenvalues calculated from this space by projecting on it the images found by the Viola-Jones detector. As a final step, vectors of faces are classified by a SVM properly trained to classify the frontal images of human users faces giving them a face claim identifier [9]. A statistical refinement of the output can be performed in this stage, forcing the system to return as a result of the current face identity the most recurrent one of the last 5 identities claimed.

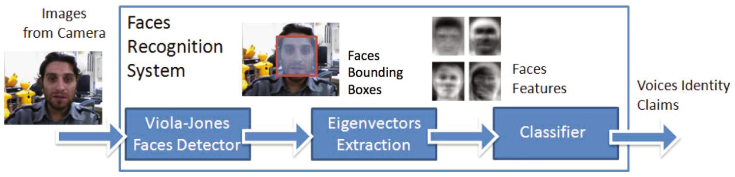


Fig. 3. The faces recognition system

4 Topics Identification

The human speech is a natural and intuitive way of communication with a robot [10]. However, the usage of the auditory channel in a human robot interaction context becomes very difficult due to its huge and noisy informative content, and due to the incompleteness and ambiguity of the natural languages. On one side, the auditive flow can encode one or more overlapped speeches, with echoes and environmental noise; on the other side, the natural language seems unable to describe them only in terms of syntax, semantics or phonetics rules, making its deep understanding a very hard task [11]. In order to avoid such problems, the approach used in this work tries to overcome the low recognition rate on the accuracy of the speech recognition system by grounding conversation between people to their topic, using only some relevant words. According to this approach, each word is weighed using a modified version of the classical "Term Frequency - Inverse Document Frequency" ranking function, a statistical measure often used in text mining and information retrieval [12]. Given a corpus of documents, the TF-IDF evaluates the importance of a word in a document. In the work here presented the same idea has been applied to evaluate the relevance of a word in a given topic,

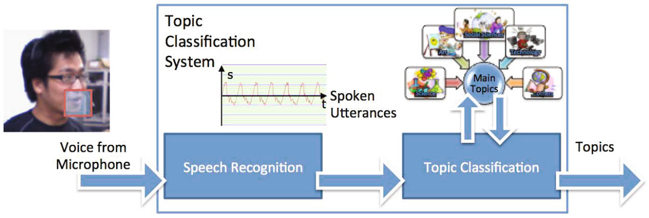


Fig. 4. The topics recognition system

performing a “Term Frequency - Inverse Topic Frequency” ranking function [13]. The TF-ITF approach in particular gives more weight to the terms often used in few topics and gives a low weight to the terms used in all the topics considered. In this way, it is possible to discard by thresholding all the negligible terms of the vocabulary, such as verbs, conjunctions, adjectives, by considering only the meaningful terms for each topic. The algorithm relies only on words frequencies, then any syntactical and semantical considerations are not taken in account: because of this, the system will not understand the details of the sentences, and, in particular, it will not be able to distinguishing affirmations and negations, likes and dislikes, and so on.

However, in order to approach free context conversations the system should be able to deal with thousands of topics. It is not possible to directly apply TF-ITF to such a huge amount of categories. A convenient way to approach this problem is by using a hierarchical categorization of the topics. Wikipedia, one of the biggest existing encyclopedias,can be seen as a huge repository of sentences categorized by thousand of topics. Moreover, each Wikipedia topic is itself categorized according to one or more parent topics [14]. From this point of view, Wikipedia offers an unique set of knowledge base that can be used to process natural language, categorized in a hierarchical graph, from the most abstract topics to the most detailed ones. Top level nodes, all children of a main root node, are the most general nodes, such as “Science”, “Art”, “Social Sciences”, “Technology” or “Society”; descending deeper to the bottom, topics start to become more and more concrete, arriving to the leaf topic nodes that represents topics related to very specific categories. In this hierarchy TF-ITF is calculated between nodes with a common parent: it is possible to evaluate for each node which is the most relevant topic between its children. Then, starting from the root of the hierarchy, it is possible to categorize words and sentences by finding a branch of the tree inside the Wikipedia based hierarchy that is coherent to its topic. In particular, Tf-Itf for each word of the sentence is normalized among the children nodes, then words with a high level of entropy are discarded. Using the remaining words, a probability of the sentence to belong to each child node is calculated. The child node with the highest probability is chosen and its topic is selected as being coherent with the sentence. A reference of the coherence of the topic with the sentence has been also retrieved by calculating the entropy of the sentence among the different childs. Through the recursive use of this algorithm, it is possible to explore the whole topic tree in order to find a path of the most related topics to a given sentence, from the abstract categories to the most detailed ones. This recursive algorithm can stop in the case of the reaching of a leaf, or in several other

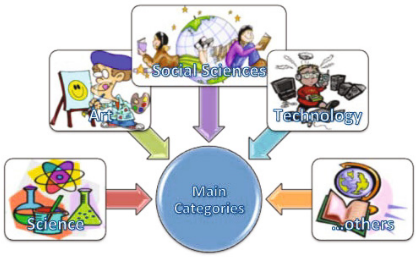


Fig. 5. Some of the top level categories in the hierarchy used by the topic recognition system

situations, such as the complete discarding of all the words in the sentence, due to high entropy. In the categorization of conversations for application in the real world, there are several aspects to be considered. First of all, according to the applications, the classification of the topic should be executed in realtime, or in several seconds, or in several minutes or, also, offline. A significant amount of data may involve a lot of calculations, and this may decrease the performance of the system. Moreover, the quality of these clusters should be settled in an accurate way. Given a specific number of clusters, a high degree of separation between them implies the use of highly-reliable, abstract topics, but which are not capable of informing about the details of the conversations. On the opposite side, a low degree of separation implies the use of more detailed, but less-reliables, topics. The choice of the amount of details for the topic classification should be carefully chosen according to the application, taking in account their reliability performances.

5 People Profile Grounding

Claims of faces coming from the SVM classifier are referred to clusters that better encode the similarities between features. However, they do not inform in an explicit way about the persons profile. The system should learn about the preferences by linking in some way the information about the identity with the information about the topic. This can be seen as a “symbol grounding” problem, in which features of the same person extracted from different modalities will converge to the same high level mental concept [15]. From this point of view, the symbol, or the model, of the human identity will “emerge” by binding between them all with the information about each partner, that is, his face, his preferences. An anchoring system, capable of linking in a correct way the features from each modality to their high level symbols representing the model of each human identity is needed. The solution here presented follows a statistical approach: while humans interact in front of the robot, claims of identities and topics recognized will be perceived at the same time. Then, the system will join them by learning this binding. According to this approach, identities and topics are bound together to form multi modal clusters that represent an unique complete model of each identity. This idea has been implemented through a Hebb network capable of representing the connection between the clusters from each modality [16]. In an Hebb network, if two connected neurons are repeatedly activated at the same time, they will strengthen their connection, tending to become more and more associated. According to this, while users converse in front of the robot, the system will learn and improve its bindings between the topics recognized and identities perceived. The system updates the weights of the most active links using the formula:

$$\triangle w_{i^* j^*} = \eta ({}^l d_{i^* j^*} a_{i^*} \cdot {}^r d_{i^* j^*} \cdot a_{j^*} - w_{i^* j^*}) \tag{1}$$

in which: w_{ij} are the weights; i^* and j^* are the indexes of the most active link; a_i and a_j are the activation level of input and output; η is a coefficient about the speed of the learning and ${}^l d_{i^* j^*}$ and ${}^r d_{i^* j^*}$ are coefficients calculated as:

$${}^l d_{ij} = \exp^{-\frac{\sum\limits_{k,k\neq j} w_{ik}}{\sigma^2}} \tag{2}$$

$${}^r d_{ij} = \exp^{-\frac{\sum\limits_{k,k\neq i} w_{kj}}{\sigma^2}} \tag{3}$$

where σ is a variance weight. According to the mutual exclusivity, the other links are inhibited:

$$w_{ij^*}(t+1) = w_{ij^*}(t) - \eta_l \cdot (1 - {}^l d_{ij^*} \triangle w_{i^* j^*}) w_{ij^*}(t) \tag{4}$$

$$w_{i^* j}(t+1) = w_{i^* j}(t) - \eta_r \cdot (1 - {}^r d_{i^* j} \triangle w_{i^* j^*}) w_{i^* j}(t) \tag{5}$$

where η_l and η_r are coefficients regarding the speed of the lateral inhibition. Following this approach, the system learns the relationship between the most active couples and it is also capable ofrecovering when something wrong has been learned. The lateral inhibition grants the system the possibility of forgeting and learning the correct connection. The dimension of the η coefficients, η, η_l and η_r, are important parameters: if the value of η is big the system will learn quickly, and if the value of η_l and η_r are low it will forget slowly. During the development, a careful evaluation of these parameters is needed. While people are discussing in front of the robot, face claims and topics from the conversation are calculated. The activation level of each of the two sides of the Hebb network will rely on this information. In particular, the probability of the current face to belong to the set of trained identities, calculated by the faces recognition system, will be used as activation level on one side of the network. The topics belonging to the branch of the tree calculated using the current discussion data, by the topic recognition system, are used for the second side of the network. In particular, the topics selected will be all together active using an activation level that will be dependent to their respective degree of detail, to their significance and to their entropy associated with the conversation considered. In this way, more abstract levels, with less informative content will give a small contribute on the identity modelling, than the others that will bring more informative content. According to the presented approach, as shown in figure 6, during different conversations, the Hebb network will store the models of all the human partners of the robot in terms of strong connections between its nodes. Favourite topics for each partners will be the most discussed and detailed topics. In this case, the network will model strong connections between the identity claims and their respective favourite topics and will be able to store the models of all the human partners of the robot.

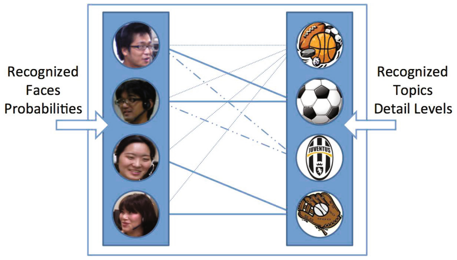


Fig. 6. The Hebb network models the profiles of the human partners as connections between identities and conversation topics

6 Experimental Results

The system has been tested in different scenarios in order to evaluate the performances of each single component. Then, an evaluation of the profiling system is performed.

6.1 People Identification Evaluation

To evaluate the physical identification of humans, the system was tested 10 times using a population of 5 people. In each experiment, a person reads, for about 15 seconds, a text in front of the robot. As shown in figure 7, the system has been tested by incrementing the number of the people in the training set. With a small number of identities in the training sets, performances of the system are very good, but while incrementing them discriminating identities in real environment, it becomes more and more difficult. To preserver the performances of the partners recognition a biggest number of features can be used, in order to rely on a more detailed description. The figure 7 shows also a comparison of the results obtained using a different number of the features. There are several reasons for the failures. The most relevant are the changes in the light conditions

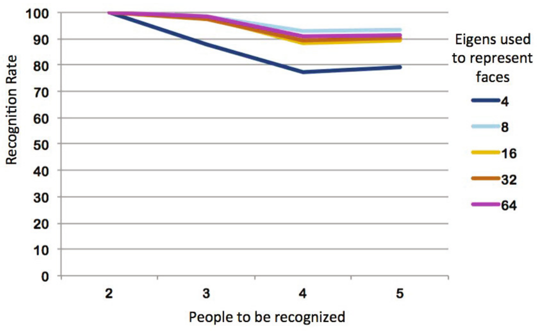


Fig. 7. Performances of the people identification by varying the number of faces to be recognized, and among different lengths of the features vector

of the environment and the change of the gaze direction of the persons, that can also bring a variation on the shadows on the faces. These conditions introduce important differences in respect to the training set, so the system is not able to identify human users in a correct way. Also, emotions can be the reason for a wrong identification: a big smile can alter considerably not only the features of the face, but also the gaze direction and, consequently, its shadows. Recognition problems will be overcome using more adaptive techniques of clustering instead of SVM or by using a more reliable multi modal approach, as using voices claims.

6.2 Topics Identification Evaluation

The Wikipedia based hierarchical topic recognition system has been evaluated using a set of 45 sentences coherent to 15 common topics (3 sentences per topic) such as "soccer", "travel", "recipes", "manga", "music" an other everyday topics of conversation. From the whole test set of conversations, the system was not able to recognize in 40% of the topics, because they were not included in the Wikipedia based training set. Despite of this, the 78% of the remaining 60% of the conversations considered, was successfully classified by the system into coherent topics. In order to judge the quality of the topics recognized, more tests have been performed to understand to what extent the results were acceptable for humans. We asked one volunteer cooperator to score the acceptability of them using points from 1 ("very unrelated") to 5 ("very coherent"), comparing original sentences and recognition results. The collected data was normalized among the deepness of the tree explored, depicting with this their detail degree, and then statistically interpolated, in order to find a trend. As shown in Figure 8 by the blue line that represents the trend extracted from the coherence results, the most abstract topics are felt as less unrelated. Going through the most detailed nodes, the coherence grows up until a maximum, then it decades. This can be explained by the difficulty that the system finds on separating strongly tighten nodes, due to their low amount of information differences. It is interesting to explain something more about this curve: the two maximum points are both located to high coherent topics with a different level of detail, such as "sport" and "soccer", according to the figure. This can be explained by the search in the graph, that explores among different levels of abstraction of the topics. Finally, the trend of the information entropy associated to the sentences, amongst the detail level of the tree explored, was calculated. A statistical interpolation of this data has been conduced in order to find the trend depicted in the Figure 8 by the red line. In this case, the closer it is to zero the entropy, the higher is the informative content of the detail level. As it is possible to see, the coherence results are highly correlated to the entropy, showing that this can be used as a criteria to judge the quality of the solution given. Despite of this important result achieved, several are the limitations of this system. The use of a huge amount of data elaborated during the search in the hierarchy of the topics is reflected on the long processing time. At this stage, the hierarchical topic recognition system cannot perform in real-time. Furthermore, the lack of information of the training set depicted in the results shows that, though the Japanese Wikipedia provides a vast and expansive set of topics, it is not able of including all the possible topics that can be discussed during daily life conversations. Lastly, the system at this stage is

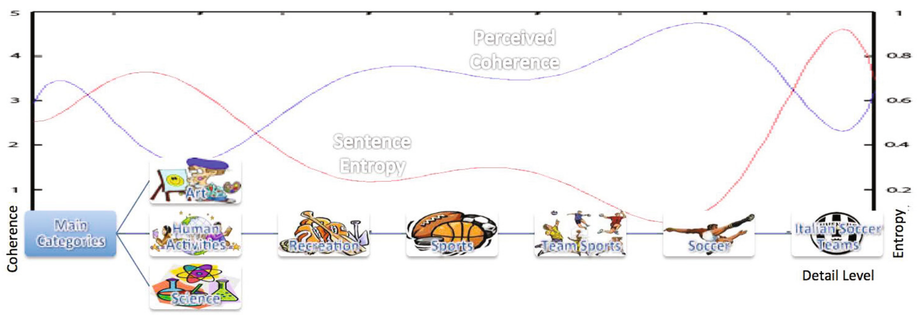


Fig. 8. Coherence trend of proposed topics, in blue, among their detail level, and their entropy

not able of infering any kind of information, other than what is explicitly described by the tree, but that can be obvious to an human listener.

6.3 Profiling Evaluation

Three couples of people were considered for the experiments. For each experiment, a person in the couple is chosen randomly to watch a video related to some sport. Then, as shown in figure 9, the couple is encouraged to discuss about it in front of a robot able of nodding and pointing to the current speaker. The experiment is performed three times for each couple, proposing three different videos related to two different categories. During each experiment, facial information and speech data is recorded and used offline to extract people profiles, according to the algorithm described. In particular, according to the previously obtained results, a set of 16 features was used for the face recognition system. Moreover, topics activation levels for the grounding system have been weighed according to their detail level using the trend function previously found. Analysis of the connections learnt by the profile grounding system revealed connections between people and topics coherent to the conversation in question. In

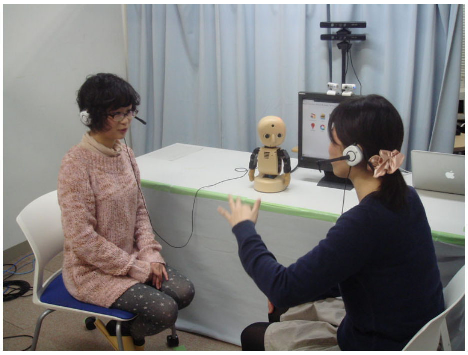


Fig. 9. People conversates in front of the robot during the experiment

particular, the strongest connections were found between people and the most coherent topics of the conversations in which they where involved, in the 77% of experiments. Moreover, the strongest connection between people and topics occurred with the most frequent topics discussed, coherently with the topic proposed in the conversations achieved, in 66% of experiments. Errors come mainly from the topic recognition system that is not always able to recognize coherent topics in all the different conversations in which human partners can be involved in. Moreover, despite the system being able to find coherent topics, it is not able to classify within identical topic conversations, that humans will easily recognize as belonging to the same topic, due to the ambiguity hiden in the natural language itself.

7 Conclusions

A profiling system for robots involved in human conversations was presented. Identities of human partners and topics discussed during the conversations were bound together in order to model their own profiles. Identities are recognized in a closed set by the classification of faces using eigenfaces technique, while topic recognition was achieved using an hierarchical approach based on "Term Frequency - Inverse Topic Frequency" ranking function. Profile modelling was exploited using a Hebb network. Experiments performed show the potential and the deficiencies of the system. Despite these problems that showed how the system is very far from its use in real, long-term, daily life contexts, the results obtained encourage us to pursuit its development and experimentation. In particular, more efforts should be focused on real time capabilities of the system and on obtaining stronger recognition identities, by relying on more stable features of the faces and by introducing a multi modal characterization, using other channels, such as the voices. Moreover, topic recognition system should be improved to have a better understanding of the conversation. In particular, ontologies [17] can be used in combination with the presented hierarchy of topics in order to improve the recognition itself, as a way to infer other connections between conversation topics. Furthermore, the system should be able to cope with dynamic situations by recognizing and incorporating in its set of known acquaintances new, unknown, people. Lastly, future experiments will focus on the effect of customized behaviours according to the profiling results during human robot interactions. Many are the real world applications that can take advantage from the idea of profiling. However, it is important to underline that here only a first approach using faces and conversation topics has been presented. It is possible to imagine more complex profiles, able to rely on more features, that can be used in several applications such as robotic companions, elderly assistants, human-robot gaming systems, and all the applications that need to rely in a strong characterization of the behaviours related to the individuality of the human partners.

Acknowledgment. Authors would like to thank Prof. M. Chetouani, Dr. S. Livieri, Dr. F. Dalla Libera, Prof. A. Chella, Prof. G. Vassallo for their support during the development of this project. This work has been supported by a Grant-in-Aid for scientific research fellowships from Japan Society for the Promotion of Science (JSPS). This project was partially founded by Regione Veneto Cod. progetto: 2105/201/5/2215/2009 approved with DGR n. 2215, 21/07/2009.

References

1. Breazeal, C.: Toward sociable robots. Robotics and Autonomous Systems 42(3-4) (2003)
2. Breazeal, C.L.: Designing sociable robots. The MIT Press (2004)
3. Anzalone, S., Nuzzo, A., Patti, N., Sorbello, R., Chella, A.: Emo-dramatic robotic stewards. Social Robotics, 382–391 (2010)
4. Lee, A., Kawahara, T., Shikano, K.: Julius—an open source real-time large vocabulary recognition engine. In: Seventh European Conference on Speech Communication and Technology (2001)
5. Anzalone, S.M., Menegatti, E., Pagello, E., Yoshikawa, Y., Ishiguro, H., Chella, A.: Audio-video people recognition system for an intelligent environment. In: 2011 4th International Conference on Human System Interactions (HSI), pp. 237–244. IEEE (2011)
6. Viola, P., Jones, M.: Robust real-time object detection. International Journal of Computer Vision 57(2), 137–154 (2002)
7. Turk, M., Pentland, A.: Face recognition using eigenfaces. In: Proc. IEEE Conf. on Computer Vision and Pattern Recognition, vol. 591, pp. 586–591 (1991)
8. Hurkens, C., Van Iersel, L., Keijsper, J., Kelk, S., Stougie, L., Tromp, J., Dolech, D., Eindhoven, A.: Face Image Database, publicly available for non-commercial use (2008), `http://cvc.yale.edu/projects/yalefaces/yalefaces.html`
9. Steinwart, I., Christmann, A.: Support vector machines. Springer (2008)
10. Kraft, F., Kilgour, K., Saam, R., Stuker, S., Wolfel, M., Asfour, T., Waibel, A.: Towards social integration of humanoid robots by conversational concept learning. In: 2010 10th IEEE-RAS International Conference on Humanoid Robots (Humanoids), pp. 352–357. IEEE (2010)
11. Anzalone, S.M., Cinquegrani, F., Sorbello, R., Chella, A.: An emotional humanoid partner. Linguistic and Cognitive Approaches To Dialog Agents (LaCATODA 2010) At AISB (2010)
12. Jackson, P., Moulinier, I.: Natural language processing for online applications: Text retrieval, extraction and categorization, vol. 5. John Benjamins Pub. Co. (2007)
13. Anzalone, S.M., Yoshikawa, Y., Menegatti, E., Pagello, E., Sorbello, R., Ishiguro, H.: A topic recognition system for real world human-robot conversations. In: IAS 2012, 12th International Conference on Intelligent Autonomous Systems (2012)
14. Denoyer, L., Gallinari, P.: The Wikipedia XML Corpus. SIGIR Forum (2006)
15. Coradeschi, S., Saffiotti, A.: An introduction to the anchoring problem. Robotics and Autonomous Systems 43(2-3), 85–96 (2003)
16. Yoshikawa, Y., Hosoda, K., Asada, M.: Unique association between self-occlusion and double-touching towards binding vision and touch. Neurocomputing 70(13-15), 2234–2244 (2007)
17. Kobayashi, S., Tamagawa, S., Morita, T., Yamaguchi, T.: Intelligent humanoid robot with japanese wikipedia ontology and robot action ontology. In: Proceedings of the 6th International Conference on Human-Robot Interaction, pp. 417–424. ACM (2011)

Motivation-Based Autonomous Behavior Control of Robotic Computer

Blagovest Vladimirov, Hyun Kim, and Namshik Park

Electronics and Telecommunications Research Institute,
138 Gajeongno, Yuseong-gu, 305-700 Daejeon, Korea
{vladimirov,hyunkim,nspark}@etri.re.kr

Abstract. Successful development of a robotic computer as a mediator in smart environments requires providing a certain level of behavior autonomy to the robot and a capability to adapt its behavior in long-term interaction with the users. We attempt to identify core autonomy-related functionalities and describe the design and implementation of an autonomous behavior control subsystem that provides them. The Motivation Module is essential for providing a balance between the robot's autonomy and our ability to influence its behavior development in a long term. We present the results of two test scenarios illustrating basic use of the newly provided functionality.

Keywords: autonomous behavior, motivation, robotic mediator.

1 Introduction

The volume of available digital information and the variety of related services in our everyday lives are increasing. Access to these services is not constrained to the traditional computers anymore, as witnessed by the proliferation of such appliances as smart phones, tablets, smart TVs, etc., gradually bringing us toward the realization of smart environments.

In our work on the Future Robotic Computer (FRC) project [1], we explore the possibility of using a robotic computer as a mediator in smart environments. By building a robotic computer we aim at weaving relevant digital information in a flexible manner into the objects that surround us to support multi-modal, context-aware interactions. As an extension to the FRC's Software Framework, which supports creating of applications on the platform, we developed an Autonomous Behavior control Subsystem (ABS). In this paper we describe the Motivation Module (MM) of ABS and its role for supporting behavior autonomy.

2 Behavior Autonomy for FRC

The concept of autonomy has been analyzed in various contexts. Froese et al. [2] considered different aspects of autonomy in biological and artificial systems in relation to research on artificial life. They made a distinction between constitutive and behavioral autonomy. The former, focuses on the internal organization and the system's capacity for self-production and self-sustaining of organizational identity in respect to its environment. The latter is concerned with external behavior and is related to the stability and flexibility of the system's interactions with the environment. Many factors that affect the robot's autonomy are summarized in the ALFUS project [3], which aimed to devise a comprehensive approach for evaluation of the autonomy level of unmanned systems. In addition to the independence from human intervention, it also takes into account such factors as mission complexity and environment difficulty when comparing the levels of autonomy observed in the performance of different systems.

The issue of balancing the autonomy of a system and our ability to control it from outside to perform desired tasks has been discussed in [4]. On one hand, exercising too much control, limits the system's autonomy and shifts toward us the burden of taking care of mundane details. On the other hand, too much autonomy makes it more difficult to obtain specific, useful behavior from the system. The author suggested guidelines for designing cognitive architecture that is autonomous and inherently trainable, drawing on essential ideas from the field of Developmental Robotics [5,6].

For our purposes, we are interested primarily in the role of robot's autonomy in supporting useful behaviors while relieving us from the need to specify all details in advance. In respect to the works mentioned above, this means that we focus on behavior autonomy with relative independence from human intervention, while still retaining the ability to guide the behavior adaptation process. Therefore, we consider the following aspects of behavior autonomy: autonomous behavior execution; autonomous behavior selection; autonomous adaptation of existing behaviors; and autonomous initiation of behaviors.

The autonomous behavior selection and execution form the core of behavior autonomy. The abilities to adapt and generate new behaviors further increase the behavior autonomy through modifying the set of available behaviors in response to environmental changes. Finally, the ability to initiate behaviors not only in direct response to external events (e.g., user's command) but also based on internal motivation allows for implementation of proactive services.

Thus, our aim with implementing ABS is to provide capabilities for selection and execution of appropriate behaviors without explicit request from the user and for adaptation of behaviors in the process of interaction with the users and the environment.

3 Autonomous Behavior Subsystem Architecture

Cognition is necessary for behavior autonomy because it provides adaptive mechanisms for action selection based not only on past and present events but also on possible future consequences of the selected actions [6]. In a survey of artificial cognitive systems, Vernon et al. [7] distinguish three general groups: Cognitivist, Emergent, and Hybrid, depending on the underlying approaches. Cognitivist approaches are based on symbolic information representation and processing systems, while Emergent approaches employ connectionist, dynamical, and enactive systems. The Hybrid approaches attempt to combine the strengths of the other two groups so that we can retain the ability to supply the system with relatively advanced initial knowledge and rely on the system's capabilities for adaptation and self-development in the process of interaction with the environment for further tuning of the desired behavior.

In ABS, we follow the general ideas of Hybrid approaches [8], using a dual-level architecture with both connectionist modules and symbolic rules. The interactions among these two levels, including top-down learning and bottom-up rule extraction, support the desired ability to specify some initial behaviors and later to adapt or refine them through appropriate interaction with the robot.

Depending on their goals and underlying approaches, various cognitive architectures emphasize different cognitive functions: reasoning, planning, memory, learning, etc. KnowRob [9] has a rich knowledge model supporting robot activities in a realistic household environment. It can also learn/adapt the action models and use them to plan the robot's behavior. On the other hand, the iCub Cognitive Architecture [6] emphasizes on starting with a core set of fundamental capabilities and developing the desired behaviors and functionality in the long-term interaction with the environment.

Compared to KnowRob, the ABS favors the learning of appropriate behavior selection instead of planning. Since with ABS, we aim more at a practical implementation rather than at a comprehensive solution of autonomous behavior control, unlike the iCub Cognitive Architecture [6], we adopt the hybrid, dual-level behavior selection based on CLARION's approach [8] that allows us to start with more complex initial behaviors. Finally, while CLARION has been used to model a diverse set of cognitive capabilities trying to approximate data from human performance on standardized cognitive tasks, with ABS we concentrate on issues specific to the robot's autonomous behavior.

ABS includes the following modules: Behavior Selection Module (BSM), Task Selection Module (TSM), Motivation Module, User Model (UM), Working Memory (WM), and Controller. BSM is at the core of ABS and its main purpose is to select appropriate behaviors. The other subsystems help to improve this selection process in various ways. The WM maintains task-relevant context which includes relevant history of previous events. The UM learns user preferred services. The MM contributes to the robot's autonomy by modeling internal drives so that the robot's behavior can be modulated indirectly, based on internal state.

The ABS structure, the internal communication among its modules and the external communication with DAS are shown in Fig. 1. In one behavior selection step, initially, ABS receives a sensory event from DAS. The sensory event is augmented with information maintained in the WM to form the current input context. The current input context is sent to the UM and if there is a user preferred service in the current situation, it is added back to the input context. Based on the input context, MM updates the internal drives' and goals' activations and returns the new values. Next, TSM selects a task that is appropriate for the input context. Finally, BSM selects the appropriate behavior for execution. The action specified by the selected behavior is sent to DAS, while

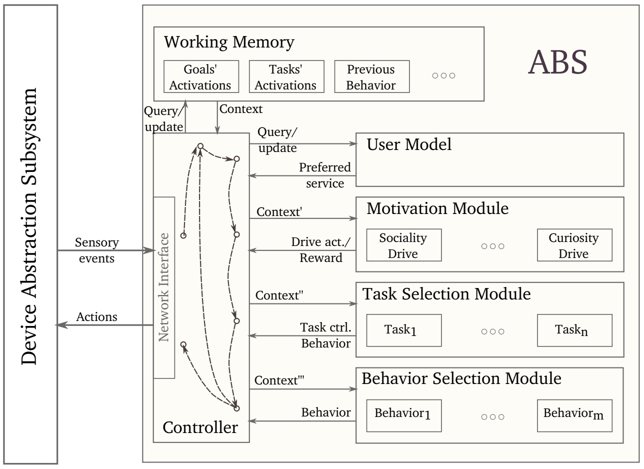


Fig. 1. ABS structure, internal communication among its modules and external communication with the Device Abstraction Subsystem

simultaneously the WM is updated with information that is deemed necessary for the system's functionality in the future.

Starting with the core functionality, below we provide detailed description of the ABS's modules.

3.1 Behavior and Task Selection

The primary purpose of the BSM is to select an appropriate behavior in a given context. In addition, BSM supports learning of behavior selection rules and adaptation of rules specified in advance based on reward signal computed by MM. Thus, the BSM partially implements the required ABS functionality to provide capacity for behavior selection and behavior adaptation. After explaining the common information representation used in ABS, we will describe rules and behavior representations, and finally the behavior selection and adaptation mechanisms.

A sensory event consisting of a set of features with their values is represented by a chunk containing a set of dimensions. A dimension d is a named, ordered set of tuples (v_i^n, v_i^a) that represents a given feature, where v_i^n is the name and v_i^a is the activation of the i^{th} value. For example, a chunk with one dimension *recognized_object* and tuples *((“book”, 0.8), (“magazine”, 0.2), (“pie_box”, 0.0))* could represent an object recognition event with the values' names denoting the possible objects and the values' activations showing the probabilities assigned by the recognition algorithm.

We represent explicit symbolic rules as a combination of a condition chunk and an output chunk. The condition chunk is used as a prototype against which the context is matched [10], while the output chunk represents an action with its parameters. The degree of activation A_R of rule R, in a given input context represented by an input chunk I, is computed using a distance metric between the dimensions specified by the rule's condition chunk and the dimensions with the same names that are present in the input context description, as shown in (1).

$$A_R = \sum_{d \in D_R^C} W^C \Bigg(\sum_{v \in V_d^C} U_{dv}^C W_d^C \Big(1 - \| A_{dv}^C - A_{dv}^I \| \Big) \Bigg) \tag{1}$$

Here, D_R^C is the set of dimensions of the condition chunk of rule R; V_d is the set of values that belong to the current dimension d; A_{dv}^C and A_{dv}^I are the activations of the current value v of the current dimension d of the condition and input-context chunks correspondingly; the dimension weight W^C is 1 if we have a disjunction of dimensions in the condition chunk or $\frac{1}{|D_R^C|}$ if we have a conjunction; similarly, the value weights W_d^C are 1 for disjunction or $\frac{1}{|V_d|}$ for conjunction of values in the dimension d of the condition chunk; and finally, the parameters U_{dv}^C are set to 0 for the values v of dimension d that we want to ignore, and to 1 otherwise.

In ABS, we model robot's behaviors (external or internal) with internal control structures representing situated actions, where actions are basic primitives specified and implemented in DAS. Consequently, as shown in Fig. 2, behaviors are represented by an output chunk that specifies the action with its parameters and a dual-level structure specifying the conditions for executing that action. The top level of that structure consists of an initial rule and an optional collection of candidate rules extracted from the bottom level (currently, all rules share the behavior's output chunk). The bottom level consists of a multi-layer neural network trained with Q-Learning [11]. The structure of the neural network is defined at initialization time. It has one unit in the output layer to represent the estimated Q-Value. The neural network weights are updated according to (2).

$$\Delta w_t = \eta \Big[r_t + \gamma Q_{t+1} - Q_t \Big] \sum_{k=0}^{t} (\gamma\lambda)^{t-k} \nabla_w Q_k, \tag{2}$$

where η is the learning rate, γ is the future reward discount factor, λ is the eligibility traces decay parameter, r_t is the reward, and Q_t is the Q-value of the selected behavior's action in the given input context at time t.

For a given behavior and a given specific input context represented as an input chunk, we compute the rules' activations at the top level. We present the same input context as an input pattern to the bottom-level neural network and obtain the estimated Q-value from the output. Then, the behavior's activation is computed as a weighted, linear combination of the maximum rule activation from the top-level and the estimated Q-value. The behavior selection is performed by comparing the behaviors' activations and selecting the maximum activated behavior following an ϵ-Greedy policy.

Behavior adaptation functionality is provided by optional learning processes in the top level, through maintaining performance statistics for each rule, and in the bottom level, through Q-learning based on reward signals from the MM.

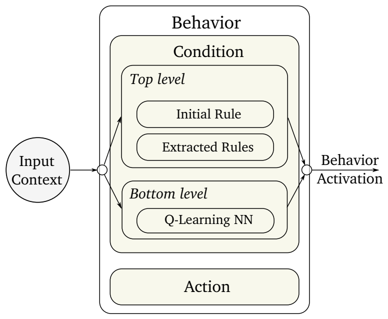


Fig. 2. Dual-level Behavior representation in ABS

In addition, top-down learning can be performed by using only the top-level rules to compute the behavior's activation and at the next step using the obtained reward to train the bottom-level network. In the current implementation, the following simple approach is used for bottom-up rule extraction. When the reward is above a pre-specified threshold, the input context used to compute the behavior's activation is transformed into a new candidate rule condition. This new rule is added to the collection of candidate rules if no similar rule exists already. In the consequent interaction, the rules' performance statistics are used to remove rules that fail to meet a pre-specified performance threshold.

The purpose of the TSM is to select an appropriate task for the current context. In ABS, a task is related to a set of behaviors that are used in combination to achieve some desired result. The TSM supplements BSM in implementing the required ABS functionality to provide behavior selection and adaptation capabilities.

The TSM has the same structure as the BSM. It consists of a set of *task-control behaviors*. For each task there are task-control behaviors with fixed actions for starting, suspending, resuming, and stopping the task, which modify the corresponding task-activation state maintained in the WM. This task-activation state is used in the behaviors' conditions in BSM to distinguish, when necessary, the behaviors that belong to the currently active task from the rest of the behaviors. Thus, the task selection functionality is provided by appropriate selection of task-control behaviors.

3.2 Motivation Module

The MM plays a role in the implementation of autonomous behavior initiation and in behavior adaptation functionality. The MM provides internal state that can be used in the behavior conditions to trigger or to inhibit behaviors. It also guides the behavior adaptation by computing reward signals used in the learning process.

This is achieved by implementing internal drives' models and a mechanism for computing reward signals from drives' activations and activation changes.

The activation state of the internal drives and goals is included into the input context used in TSM and BSM to compute the behaviors' activations. The reward signals are used in TSM and BSM to modify the network weights and the rules performance statistics of the relevant behaviors' conditions.

Currently, in ABS we have implemented two main internal drives: a *Sociality drive* and a *Curiosity drive*. The aim with implementing a Sociality drive is to balance behavior autonomy and external behavior control. It gives the users a mechanism for influencing the behavior adaptation process effectively, while providing for a certain level of autonomy. On the other hand, the Curiosity drive makes exploration-based behavior adaptation possible even without user intervention.

The model of the Sociality drive is shown in Fig. 3. The Sociality drive is configured with a collection of conditions that specify the effect of certain sensory events on the drive's activation level. When an input context is presented to the MM, the Sociality drive's activation d^s is computed as a linear combination of its conditions' activations a_i and a time component $b(t)$ as follows: $d^s = w_t b(t) + \sum_{i=1}^{n} w_i a_i$. The time component is included to allow increasing of the Sociality drive's activation in the absence of social interaction. A reward signal is computed from the Sociality drive's activation change, interpreting a decrease in the activation as a positive reward.

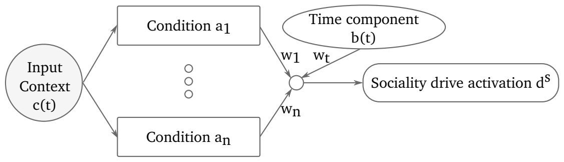


Fig. 3. Sociality drive model in MM

The model of the Curiosity drive is shown in Fig. 4. Due to the chosen implementation model of the Curiosity drive, in the process of exploration the robot develops a predictive model for the consequences of performing a given behavior. This model can be used to support behavior planning in further development of the ABS.

As shown in Fig. 4, the Curiosity drive model is based on the ability to predict relevant subset of the next input context. Currently, we use a neural network, trained on-line to learn the mapping from current context and currently selected behavior to the input context at the next step. The prediction performance is used to compute two components of the curiosity. The first component d_s^c is related to the 'surprise' or the unexpectedness of the obtained context and is computed from the prediction error as shown in (3)

$$d_s^c(t) = g d_s^c(t-1) + e(t), \tag{3}$$

where $e(t)$ is the normalized prediction error at the current time step and $0 \leq g < 1.0$ is a coefficient controlling the rate of decrease. The second component d_k^c

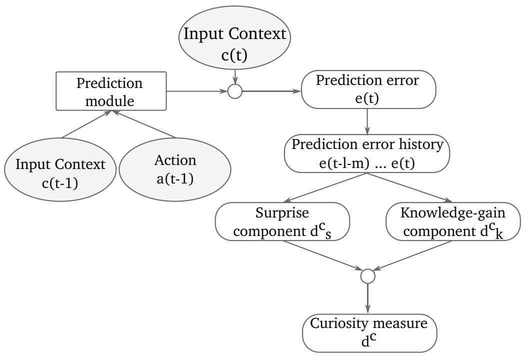


Fig. 4. Curiosity drive model in MM

is related to the knowledge gain and is computed as a decrease of the prediction error according to (4)

$$d^c_k(t) = \frac{1}{n}\sum_{i=0}^{n-1} e(t-i) - \frac{1}{m}\sum_{j=0}^{m-1} e(t-l-j), \tag{4}$$

where l is the span between the reference time points used in comparing the error, while n and m are smoothing parameters. Separate error history traces are kept for each behavior, thus the knowledge gain reflects the prediction performance for the specific behavior that was selected for execution.

The Curiosity drive activation is set to the curiosity measure d^c, which is a linear combination of the two components d^c_s and d^c_k. The reward signal from the Curiosity drive is proportional to the current drive activation.

The MM uses the computed drives' activations in predefined, linear combination dependencies to set the goals' activations. Also, a common reward signal $r = c_s r^s + c_c r^c$ is computed, where r^s and r^c are the reward signals from the Sociality drive and the Curiosity drive correspondingly, and c_s and c_c are coefficients balancing the contribution into the common reward signal.

3.3 User Model, Working Memory and Controller

The UM learns users' preferences from interaction. When a user requests some service (internally represented by a task), the UM associates the requested service with the perceived current context using probability-based associative memory. With time, the salient associations are used to extract explicit rules. A detailed description of the UM is given in [12].

The WM uses a collection of chunks to maintain relevant information necessary for setting drives' and goals' activations, suggesting services, and selecting tasks and behaviors.

The purpose of the Controller is to coordinate the interaction among the ABS modules and the external interaction with DAS. Through the interaction with DAS, it provides the capabilities for collecting sensory information and for behavior execution.

4 System Implementation

The current implementation of FRC includes an Agent Unit shown in Fig. 5 and a server. The main components of the Agent Unit are two projector/camera pairs with five degrees of freedom, 3-channel microphone array, a stereo speaker set, and an embedded PC with wireless networking.

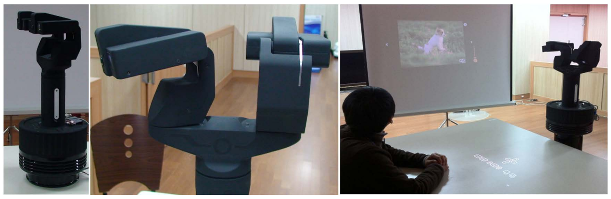


Fig. 5. The FRC's Agent Unit

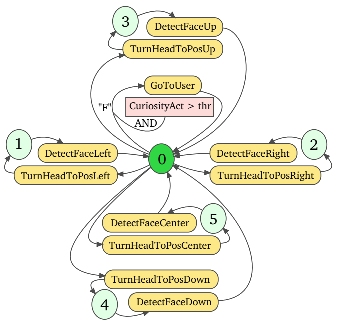
DAS is a part of the FRC's Software Framework implementation ICARS (integrated control architecture for robotic mediator in smart environments) is described in [13]. ICARS consists of three layers that provide: a flexible communication/device model; an adaptive service model for the integrated robot control architecture; and a behavior-based high-level collaboration model. The ABS is implemented in C++ and the Controller communicates with ICARS over a TCP connection.

5 Test Scenarios

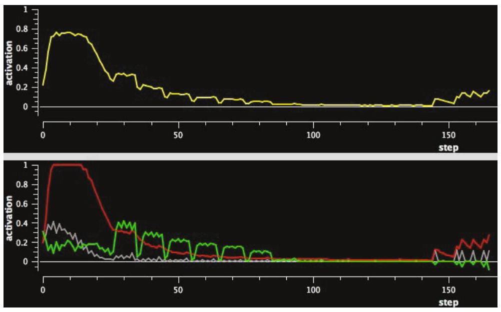
We present two test scenarios showing the role of the Curiosity drive in behavior initiation and inhibition.

5.1 Curiosity Driven Behavior Task 1

In this task, the Agent Unit observes the room around itself trying to detect user presence. The prediction module of the Curiosity drive has user presence from the previous time step and the last two performed behaviors as input and current user presence as target. If there is a user and if the Curiosity drive's activation is above certain level, then the Agent Unit approaches the user. In this scenario we have one task and eleven behaviors shown in Fig. 6 a). The Sociality drive and the UM are not used. The top plot in Fig. 6 b) shows the Curiosity drive's activation. The predictor starts from a random state and the initial hundred steps it learns that there is nobody in the room. The arrival a)



b)



c)

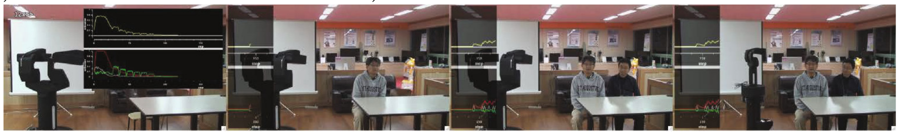


Fig. 6. Simplified task diagram a) and results for the curiosity test scenario b) and c). The rounded rectangles represent behaviors. The abbreviation “F” stands for face-detected sensory event. In b) the top plot shows the Curiosity drive activation while the bottom plot shows the ’surprise’ (red) and knowledge-gain (green) components and the prediction error (gray). The snapshots in c) correspond approximately to the following time steps in b): 118, 145, 165, and 166 (not on the plot).

of the first user violates the expectation of *no-user-presence* and leads to an increase of the curiosity but it is still below the threshold. The increase of the curiosity from the arrival of the second user is sufficient to trigger the approaching behavior *GoToUser*.

5.2 Curiosity Driven Behavior Task 2

In this task, the user puts objects in front of the Agent Unit, which reacts by saying something related to the objects. Fig. 7 a) shows a simplified diagram of the five behaviors and their conditions. The Curiosity drive is related to predicting the appearance of unknown objects in result of *WatchTableTop* behavior. The plot in Fig. 7 c) shows the Curiosity drive’s activation while known *Pie box*, initially unknown *Time magazine*, known *Wimpy Kid book* and already seen *Time magazine* are presented. Fig. 7 b) includes annotated snapshots from the Agent Unit’s camera related to the events in c).

As can be seen from diagram a), the User Agent will ask a question if an unknown object is presented and the Curiosity drive’s activation is above a specified threshold. Fig. 7 d) is a plot of the Curiosity drive’s activation from a simulated run where unknown objects are presented in a sequence to illustrate a behavior inhibition effect. After a while, the predictor learns to expect unknown objects, the curiosity drive’s activation falls below the threshold (0.1) and the Agent Unit stops asking questions.

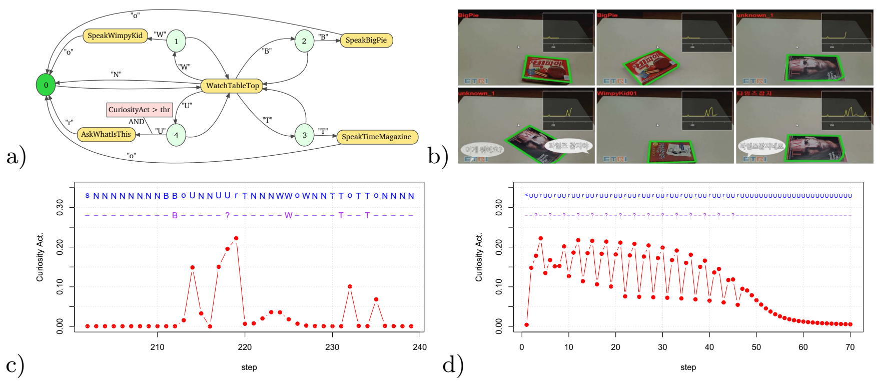


Fig. 7. Simplified task diagram a) and results for the curiosity test scenario b), c) and d). The rounded rectangles represent behaviors with the labels on incoming arcs showing the conditions for execution and the labels on the outgoing arcs show the resulting sensory event. The abbreviations stand for "N" no object, "W" 'Wimpy Kid' book, "T" Time magazine, "B" 'Big Pie' box, "U" unknown object, "o" task finished, "r" response from the user. In c) and d) the top-row labels show sensory input and the bottom-row ones show the behavior, where "-" is WatchTableTop, "?" is AskWhatIsThis.

6 Conclusions

The development of ABS aims at augmenting the FRC's Software Framework by providing behavior control with autonomous behavior selection, initiation and adaptation functionality. An important role in this process is played by the MM which, in a certain sense, shifts the balance of control over the robot's behavior toward the robot itself. In future work on the FRC, we will develop more application-oriented scenarios with long-term, user interactions and curiosity-based behavior adaptation. A promising direction of development is to use FRC to help the user to modify its own attitudes and habits as envisioned by Fogg [14]. One example could be to devise internal drives that guide the Agent Unit's behaviors to provide adaptive, timely, context-dependent opportunities and cues to aid the user in acquiring healthier lifestyle habits.

Acknowledgment. This research has been supported by Korean Ministry of Knowledge Economy and Korea Research Council for Industrial Science & Technology [Grant No. 2010-ZC1140, Development of Future Robotic Computer].

References

1. Kim, H., Suh, Y.-H., Lee, K., Vladimirov, B.: Introduction to system architecture for a robotic computer. In: Int. Conf. Ubiquitous Robots and Ambient Intelligence (URAI), pp. 607–611 (November 2011)

2. Froese, T., Virgo, N., Izquierdo, E.: Autonomy: a review and a reappraisal. In: Proc. of 9th European Conference on Artificial Life, Berlin, Germany (2007)
3. Huang, H., Pavek, K., Novak, B., Albus, J., Messin, E.: A framework for autonomy levels for unmanned systems (ALFUS). In: Proc. of AUVSI's Unmanned Systems North America, Baltimore, Maryland (2005)
4. Vernon, D.: Reconciling autonomy with utility: A roadmap and architecture for cognitive development. In: Proc. of Int. Conf. on Biologically-Inspired Cognitive Architectures, pp. 412–418. IOS Press (2011)
5. Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., Yoshida, C.: Cognitive developmental robotics: A survey. IEEE Trans. Autonomous Mental Development 1(1), 12–34 (2009)
6. Vernon, D., von Hofsten, C., Fadiga, L.: A Roadmap for Cognitive Development in Humanoid Robots. Cognitive Systems Monographs (COSMOS), vol. 11. Springer (2011)
7. Vernon, D., Metta, G., Sandini, G.: A survey of artificial cognitive systems: Implications for the autonomous development of mental capabilities in computational agents. IEEE Trans. Evolutionary Computation 11(2), 151–180 (2007)
8. Sun, R.: The importance of cognitive architectures: An analysis based on CLARION. J. Experimental and Theoretical Artificial Intelligence 19(2), 159–193 (2007)
9. Tenorth, M., Beetz, M.: KnowRob – Knowledge Processing for Autonomous Personal Robots. In: IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 4261–4266 (2009)
10. Duch, W., Setiono, R., Zurada, J.: Computational intelligence methods for rule-based data understanding. Proc. IEEE 92, 771–805 (2004)
11. Rummery, G.A., Niranjan, M.: On-line Q-learning using connectionist systems. Technical Report CUED/F-INFENG/TR 166, University of Cambridge, Cambridge, England (1994)
12. Koo, S.-Y., Park, K., Kwon, D.-S.: A dual-layer user model based cognitive system for user-adaptive service robots. In: 20th IEEE International Symposium on Robot and Human Interactive Communication (Ro-Man 2011), pp. 59–64 (2011)
13. Suh, Y.-H., Lee, K.-W., Lee, M., Kin, H., Cho, E.-S.: ICARS: Integrated Control Architecture for the Robotic mediator in Smart environments A Software Framework for the Robotic Mediator collaborating with Smart Environments. In: 9th IEEE International Conference on Embedded Software and Systems (ICESS 2012), pp. 25–27 (2012)
14. Fogg, B.J.: Persuasive Technology: Using Computers to Change What We Think and Do. Morgan Kaufmann, San Francisco (2003)

An Evaluation Method for Smart Variable Space in Living Space

Kazuyoshi Wada[1,*], Keisuke Takayama[1], Yusuke Suganuma[1], and Toshihiko Suzuki[2]

[1] Graduate School of System Design, Tokyo Metropolitan University, Hino, Japan
{k_wada,takayama-keisuke,suganuma-yuusuke}@sd.tmu.ac.jp
[2] Faculty of Engineering, Kogakuin University, Shinjuku, Japan
suzuki@atelier-opa.com

Abstract. The methods which improve space usage efficiency is important especially for city lives. Development of high-rise buildings and underground is one of the methods. However, those developments just lay out the spaces which are expanded in plane into vertical. Physical and monetary limitations are the problem. Therefore, the spaces which can change its functions easily/automatically depending on the situations are necessary instead of stacking of single function spaces. So far, we have proposed Smart Variable Space which realizes various functional spaces by changing its Spatial Configuration Modules dynamically. In this research, the simulated environment for Smart Variable Space was developed by using Virtual Robot Experimentation Platform. In order to clarify the efficacy of Smart Variable Space, a new evaluation index was proposed, and then, the efficacy of Smart Variable Space in living space was assessed by comparison with the conventional housing models.

Keywords: Smart Variable Space, Intelligent Space, Architectural Furniture, Skelton Infill.

1 Introduction

Big cities, like Tokyo, have been attracting many people since olden days. The trend is no changes, and population inflow into the big cities is continuing, even Japan is going into depopulating society [1]. Cities, where various functions are concentrated in limited area, have been suffering from chronic space shortage. Therefore, high-rise buildings and underground have been developed to increase space usage efficiency. However, those developments just lay out the spaces which are expanded in plane into vertical. They require a lot of resources, high construction cost, long vertical movement, and spoil a view. Therefore, instead of stacking of single function spaces, another method is necessary to increase space usage efficiency.

We have proposed Smart Variable Space (SVS) which realizes various functional spaces by changing its Spatial Configuration Modules dynamically [2, 3]. Each module can automatically transform from wall/box into a functional space, such as, bedroom, office, according to the user's daily living cycle. So far, a bedroom type

[*] Corresponding author.

module (Figure.1) was developed as an example of the spatial configuration module. However, the efficacy of daily changes of living space is not verified. Even in the field of architecture, it remains in subjective evaluation by the users. In this paper, we proposed new index to clarify the efficiency of SVS and evaluate its effectiveness in living space by using the simulator. Section 2 explains related researches, section 3 explains evaluation index for SVS and section 4 describes housing and special configuration modules model in the simulator. Section 5 explains experimental method and results of the simulation. Finally, section 6 offers conclusions.

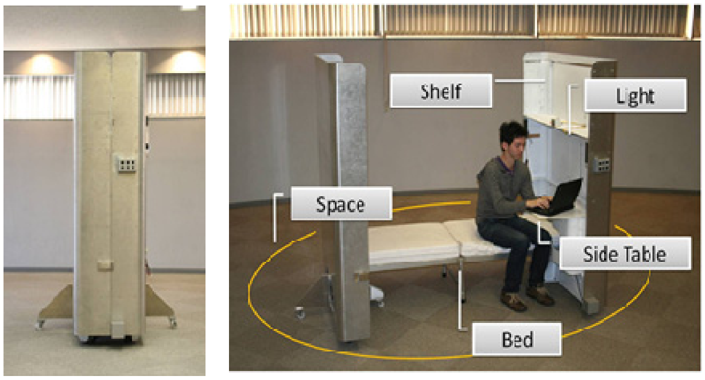


Fig. 1. Appearances of bedroom module for smart variable space

2 Related Research

2.1 Skeleton Infill

Recently, a concept of a method called skeleton-infill to extend building duration has received attention. This is the method to enhance architectural sustainability by dividing a building into two factors: the skeleton (an empty space without partitions and equipment) and infill (changeable equipment and partitions). Building duration is generally decided not on physical duration of structural skeleton but on when it cannot respond to its residents' changes of family structure or life style. In fact, duration of an architecture would be longer if it can be adaptable to such changes. The concept of the skeleton-infill which enables equipment having shorter duration than a building to be more renewable and inner room arrangement to be more changeable to respond to changes in lives has been widely accepted from the above point of view. However, infill which can change room arrangement is not easy to carry out and does not usually provide for enough changeability to respond to daily alternation of application thereof. Upon such problem consciousness, some case examples trying to solve those problems with "furniture combining architectural functions" which is like an intermediate between furniture and partitions or equipment have been carried out in order to deliver interior design of higher changeability [4-7].

2.2 Architectural Furniture

In the above-mentioned, Infill is not provided with quick-response changeability against everyday application alteration, since it is supposed to be altered in a time-span of around 15 to 30 years. Reflecting such situation, "furniture combining architectural functions" to play an intermediate role between infill and furniture have appeared.

For example, Baumhaus, Nobuaki Furuya and Studio NASCA proposed a mechanism, as a trial to enhance sustainability of rented apartments which are expected to respond to various life styles, wherein architectural part is to be an empty space without partition like a skeleton, and interior provides furniture functioning as partition as well as closet which residents can arrange interior layout by deciding how the furniture is placed [4].

Suzuki named such concept like an intermediate between infill and furniture as "furniture combining architectural functions - Architectural Furniture". Further, Architectural Furniture is defined as "what is able to segment space, provide functions in place, and to be moved and altered easily" [6, 7].

Figure.2 shows the examples of architectural furniture, mobile kitchen, foldaway guest room and foldaway office, designed by Suzuki. Each architectural furniture has casters to realize easy operation by humans. When those are folded in, its appearances look like suits case. When those are folded out, kitchen space, guest room, and study room appears. However, one or two operators are required due to the size and weight.

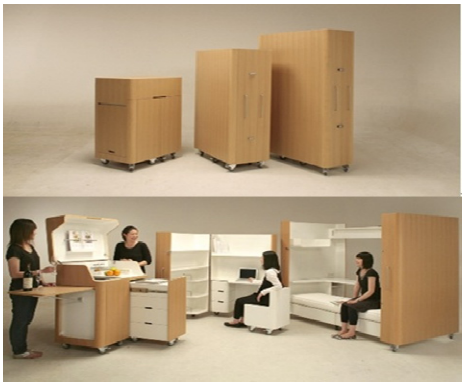


Fig. 2. Folding in and out of three types of Architectural Furniture

2.3 Robotized Structurization of Living Environment

On the other hand, there are many researches which apply information and robot technology into living environment to give intelligence and achieve various services. For example, Sato developed robotic room which supports daily activities of bedridden patients/student living alone in the room, using embedded sensors in furniture and robot arm [8]. Hashimoto proposed intelligent space and developed DIND (distributed intelligent network device) which had various sensors and communication device, and then studied the navigation of robots, wheelchair and people with visually impairment [9]. Ohara studied ubiquitous robot system which provides physical services by cooperation with distributed robot functions in environment [10]. Sugano developed WABOT-HOUSE. Robotic partition and movable kitchen were installed in the house to realize daily changes in a layout to meet various life styles of the residents. Automatic change of the layout and cooperative movement with mobile robot were achieved using sensors and RFID in the environment [11]. Tanikawa developed active caster with a built-in motor to physically assist people with disabilities. The object (table, chair, door, etc.) to be moved can be maneuvered easily and remotely by the casters attached to it [12]. However, the research which focused on efficiency improvement of the space is rare.

2.4 Smart Variable Space

Architectural furniture is suitable for the change in a layout that is more daily than Skeleton Infill. Additionally, the character of architectural furniture is effective for the efficiency improvement of the space. Even narrow space could have various functions by preparing various architectural furniture modules. However, architectural furniture requires one or two operators due to the size and weight. Against this problem, robot technology can offer a solution; even more can add some intelligence and autonomy. Smart Variable Space is the expanded concept of architectural furniture by combining with robot technology. The space is composed with robotic architectural furniture, such as bedroom, and office room, and can change its functions automatically according to the users demand or lifestyles.

3 Evaluation Index for SVS

Present floor plan of house is divided into living, kitchen, study, bedroom, and etc. according to the function. The resident moves the room according to the usage. At this moment, the room not used is a useless space in the viewpoint of the space efficiency improvement. Therefore, we introduce a concept of "available area; A_{avail}". A_{avail} is defined as the area excluding disused area from the evaluating area, $A_{eval.}$ A_{avail} changes according to the user's daily living activities. The each activity is expected to continue some period of time. So, we defined "available area rate; P_i" as follows;

$$P_i = \frac{A_{avail.i}}{A_{eval}} \tag{1}$$

Here, $A_{avail,i}$ is A_{avail} in the i^{th} time period. On the other hand, the time for changing spatial functions, $T_{trs.j}$, is much smaller than time for a user's daily living activity. $T_{trs.j}$ is j^{th} time period for changing space function; means the time for transposing of spatial configuration modules in SVS, and moving to other rooms in conventional housing. We defined "available area in transposing; $A_{trs.j}$", and the rate; $P_{trs.j}$ as follows;

$$P_{trs.j} = \frac{A_{trs.j}}{Aeval} \quad (2)$$

Then, the total average available area; P_{avg}, was defined as follows;

$$P_{avg} = \alpha \frac{\sum_{i=1}^{N} P_i T_i}{T_{all} - \sum_{j=1}^{M} T_{trs.j}} + \beta \frac{\sum_{j=1}^{M} P_{trs.j} T_{trs.j}}{\sum_{j=1}^{M} T_{trs.j}} \quad (3)$$

Here, α and β are weight parameters. Finally, we explain "evaluating area; A_{eval}". A_{eval} is defined as the area excluding plumbing equipments/product area and storage area from total internal dimension of the housing; because, those are difficult to rearrange or not necessary to change its function. Figure.3 shows the relationship between the available area rates and the time periods.

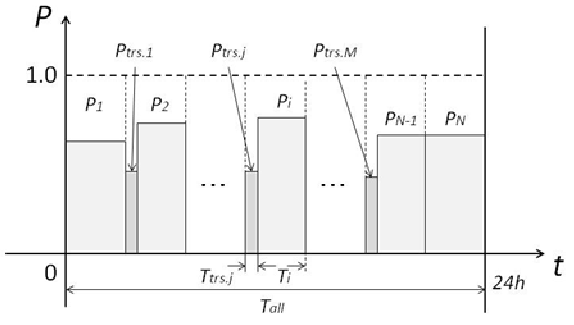


Fig. 3. Relationship between available area rate and time period

4 Simulator

In order to evaluate SVS in various situations, we made housing and spatial configuration module models by using the simulator, named V-REP [13]. We use this simulator to visualize the space and to obtain the information on the movements.

4.1 Housing models

SVS assumes big city, therefore, apartment is the target housing. The SVS housing model was made as a studio apartment which internal dimension was 50 square meter. The dimension came from "compact mansion", a new category of apartment, for singles and couples in Japan. Figure.4 shows its room arrangement. Plumbing products/equipments, such as toilet and bathroom were arranged in one side to create large working space for spatial configuration modules. On the other hand, a conventional housing model was made for comparison (Figure.5). The model was "nLDK" apartment which had been provided by the public corporation, and became the common arrangement of present apartment in Japan. The internal dimension was defined as 70 square meter which was average dimension in Tokyo area.

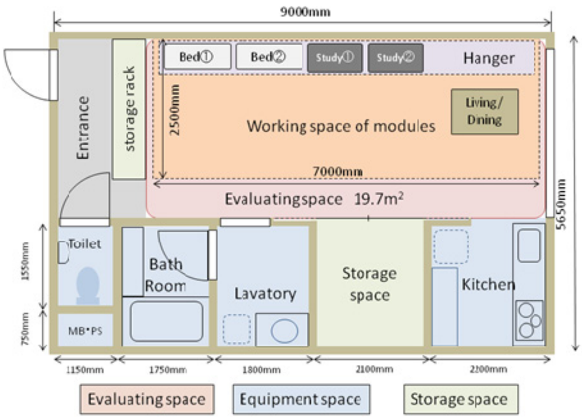


Fig. 4. Room arrangement of SVS housing model

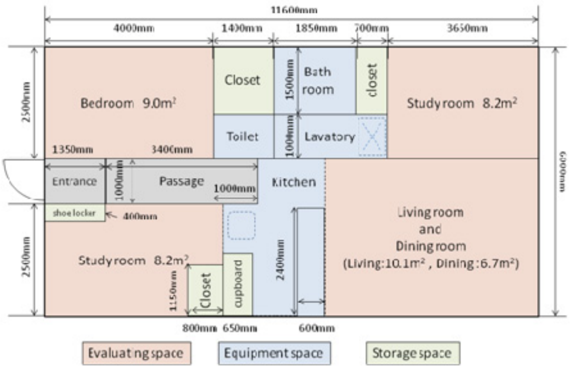


Fig. 5. Room arrangement of conventional housing model

4.2 Models of Spatial Configuration Module

From the view point of housing studies, we extracted the major spatial functions from the relationship between activities of daily living and kind of rooms. As the results, the functions were “bedroom”, ”study room”, “living room”, and “dining room”. We made the spatial configuration module virtual models which had these functions in the V-REP. These module models have autonomy.

Bedroom and study room module are single function type module. These modules have two modes, fold out and fold in. Bedroom module model is designed based on real bedroom module developed in previous research. Four omnidirectional wheel models are installed as the moving mechanism. And imaginary torque is applied to fold in/out at the each joint. The working speed of the module is also decided according to the real bedroom module. This speed is commonly used in other two module models because they are not realized in actual modules. Figure.6 and Table 1 show its appearance and specifications.

As for the study room module model, the appearance and dimensions are decided based on office room type architectural furniture. The moving mechanism is same as the bedroom module model. The study room module folds in/out like a large book. Figure.7 and Table 2 show its appearance and specifications.

As for the living and dining room, the module model is designed to have those two functions in one module because those are exclusively used in daily activities. The module has three modes, living room, dining room and fold in. the module assumes to have omnidirectional wheels as moving mechanism. Imaginary torque is applied at each joint for transformation. The details of deformation mechanics will be presented in the future. Figure.8 and Table 3 shows its appearance and specifications.

In this research, we didn't consider weight of all module models in order to focus on changing arrangement of the modules.

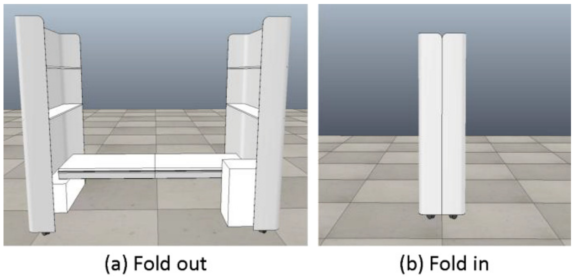


Fig. 6. Appearance of bedroom module model

Table 1. Specification of bedroom module model

	Fold out	Fold in
Width	2040 [mm]	440 [mm]
Depth	1200 [mm]	
Height	1770 [mm]	
Occupation Area	2.45 [m^2]	0.53 [m^2]
Moving speed	–	0.4 [km/h]
Swing speed	–	7.5 [deg/sec]

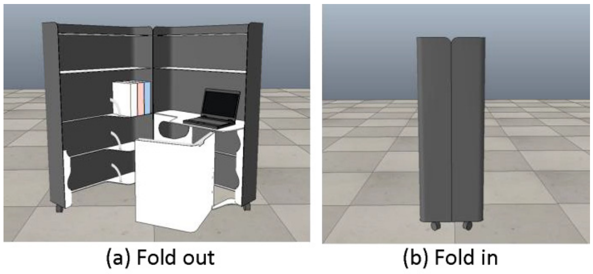


Fig. 7. Appearance of study room module model

Table 2. Specification of study room module model

	Fold out	Fold in
Width	1281 [mm]	520 [mm]
Depth	1261 [mm]	1000 [mm]
Height	1518 [mm]	
Occupation Area	1.62 [m^2]	0.52 [m^2]
Moving speed	—	0.4 [km/h]
Swing sped	—	7.5 [deg/sec]

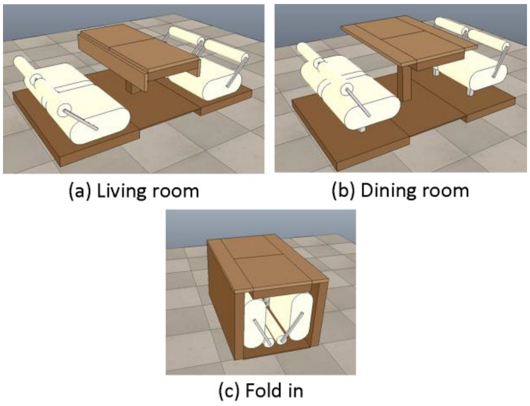


Fig. 8. Appearance of living-dining room module model

Table 3. Specification of living-dining room module model

	Living	Dining	Fold in
Width	1960 [mm]	1960 [mm]	800 [mm]
Depth	1200 [mm]	1200 [mm]	1200 [mm]
Height	780 [mm]	580 [mm]	780 [mm]
Occupation Area	2.35 [m^2]	2.35 [m^2]	0.96 [m^2]
Moving speed	—	—	0.4 [km/h]
Swing speed	—	—	7.5 [deg/sec]

4.3 Resident Model

In the conventional housing model, time for changing spatial functions corresponds to the resident's walking time from room to room. Moreover, how change the spatial functions depends on the household composition and resident's lifestyle. In this research, we targeted couple household increasing the number recently [14]. A young couple model was made based on the average physical data of Japanese male and female (Table 4) [15]. Meanwhile, their day-to-day timetable was defined using the statistical data published by the Ministry of Internal Affairs and Communications, Japan [16]. The timetable was divided into 48 time periods, and then, we extracted the rooms as spatial functions which corresponded to their activities of daily living at each time period (Table 5).

Table 4. Specification of residents model

Sex	Male	Female
Age	25 - 34	25 - 34
Average height [cm]	172.1	158.7
Walking speed [m/s]	1.5	1.21

Table 5. Day-to-day timetable of the residents

Time	Room		Scene
	Male	Female	
0:00 ǀ 6:00	Bedroom	Bedroom	Scene 1
6:30	Lavatory	Kitchen	Scene 2
7:00	Dining room	Dining room	
7:30	Go to work	Kitchen	
8:00 ǀ 18:00		Go to work	
18:30		Kitchen	
19:00			
19:30	Dining room	Dining room	
20:00	Living room	Kitchen	Scene 3
20:30		Living room	
21:00	Bath room		
21:30	Living room		
22:00			
22:30	Study room	Study room	Scene 4
23:00		Bath room	
23:30		Study room	

5 Experiment

5.1 Methods

Switching points of spatial functions were extracted from the timetable. 4 scenes, bedroom scene (Scene 1; 0:00-6:30), dining room scene (Scene 2; 6:30-20:00), living room scene (Scene 3; 20:00-22:30), study room scene (Scene 4; 22:30-0:00) were defined. In order to have same spatial functions in conventional housing model, two bedroom and study room modules, and one living-dining module were installed in the housing model for SVS. The layout of the modules in each scene are shown in Figure.9. We defined minimum path for each modules and residents models to change the scenes, and then, simulated their movement at each switching points of the scenes. The paths of residents models are described in Figure 10. The blue and red line means male's and female's path, respectively.

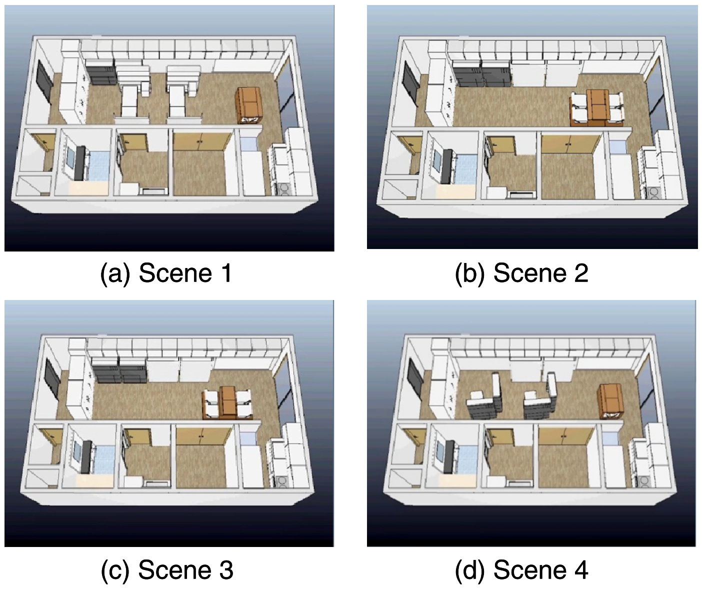


Fig. 9. Layout of the modules in each scene

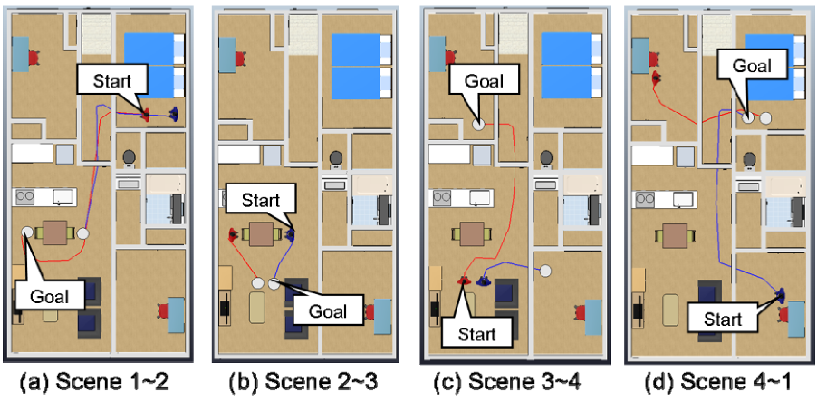


Fig. 10. Walking paths of resident model

5.2 Results

A_{eval} of each housing model were measured as 19.7 m^2 in SVS housing, and 42.2 m^2 in conventional housing. The results of required times for changing spatial functions in each housing model are described in Table 6. SVS required longer time in every switching point. The maximum required time was 70 sec. at the switching point from the scene 4 to 1. Then, the total average available area, P_{avg} was calculated by equation (3). In this research, weight parameters, α and β, were defined as follows:

$$\alpha = \frac{T_{all} - \sum_{j=1}^{M} T_{trs.j}}{T_{all}} \tag{4}$$

$$\beta = \frac{\sum_{j=1}^{M} T_{trs.j}}{T_{all}} \tag{5}$$

As for the conventional housing model, all of the evaluating area is not used during the residents walking from the room to room. Therefore, the value of equation 2 is always zero. Besides, the time for changing spatial functions, $T_{trs.j}$ was defined to use smaller one among the residents. Table 7 shows the results of both housing models. SVS housing required longer time for changing spatial functions in comparison with conventional housing. However, the time, 187 sec. is much smaller than 24 hours. On the other hand, SVS housing's P_{avg} is about 4 times higher than the value of conventional housing.

Table 6. Required times for changing spatial functions

	SVS [sec]	Male [sec]	Female [sec]
Scene 1-2	62.00	4.10	7.65
Scene 2-3	6.00	1.65	1.80
Scene 3-4	49.00	1.85	5.75
Scene 4-1	70.00	6.20	4.45

Table 7. Results of total avarage available area

	Conventional housing	SVS housing
P_{avg}	0.24	0.96
$\sum_{j=1}^{M} T_{trs.j}$	12 [sec]	187 [sec]

6 Conclusion

In this paper, we proposed an evaluation index for variable space which changes its spatial functions in daily. The new concept, "available area rate: P_i" and "available area rate in transposing: $P_{trs.j}$" were introduced to evaluate the space use efficiency in every time period. SVS and conventional housing model, and young couple household model were made, and then the efficacy of SVS was evaluated using simulator. The result showed SVS housing have high space use efficiency. In order to investigate more suitable target and necessary modules for SVS, we will further experiment to evaluate the efficacy in various housing, households, and lifestyle model.

References

1. National Institute of Population and Social Security Research: Population Projections for Japan: 2001-2050 (January 2002)
2. Wada, K., Suzuki, T., Takayama, K., Kubo, E.: Development of Bedroom Module for Smart Variable Space. In: Proc. of the 5th Int. Conf. on the Advanced Mechatronics, pp. 540–544 (2010)
3. Wada, K., Kubo, E., Nakashio, N., Takayama, K., Suzuki, T.: Path Planning for Bedroom type Robot Module. In: Proc. IEEE/SICE International Symposium on System Integration, pp. 288–292 (2011)
4. STUDIO NASCA, `http://www.studio-nasca.com/`
5. Vegesack, A.V., Schwartz-Clauss, M., Allie, M., Haub, B.: Living in Motion. Vitra Design Stiftung (2002)
6. Suzuki, T.: Architectural Furniture -furniture combining architectural functions. AIDIA Journal 7 (2007)
7. Suzuki, T., Honma, K.: Development and Validation of "Architectural Furniture" Research on Furniture Combining Architectural Functions ("Architectural Furniture"). AIDIA Journal 8 (2008)
8. Mori, T., Sato, T.: Human Support System: Robotic Room. Systems, Control and Information 44(3), 151–156 (2000)
9. Lee, J.H., Hashimoto, H.: Intelligent Space - Its concept and contents. Advanced Robotics Journal 16(4) (2002)
10. Ohara, K., Ohba, K., Kim, B.K., Tanikawa, T., Hirai, S.: Ubiquitous Robot with Ubiquitous Function Activate Module. In: INSS 2005, San Diego, USA (2005)
11. WABOT-HOUSE LABORATORY, `http://www.wabot-house.waseda.ac.jp/`
12. Home Environment Models for Comfortable and Independent Living of People with Disabilities, `http://www.aist.go.jp/aist_e/latest_research/2010/20100715/20100715.html`
13. V-REP: Virtual Robot Experimentation Platform, `http://www.hibot.co.jp/vrep.php`
14. Nishioka, H., Suzuki, T., Yamauchi, M., Suga, K.: Household Projections for Japan: 2005 – 2030 Outline of Results and Methods. The Japanese Journal of Population 9(1) (2011)
15. Yamasaki, M., Sato, H.: Human Walking: With Reference to Step Length, Cadence, Speed and Energy Expenditure. Journal of the Anthropological Society of Nippon 98(4), 385–401 (1990)
16. Ministry of Internal Affairs and Communications: 2006 Survey on Time Use and Leisure Activities (2007)

Modeling Robot Behavior with CCL

Konrad Kułakowski and Tomasz Szmuc

Department of Applied Computer Science,
AGH University of Science and Technology
Al. Mickiewicza 30,
30-059 Cracow, Poland
{konrad.kulakowski,tomasz.szmuc}@agh.edu.pl

Abstract. This paper presents the use of a *Concurrent Communicating Lists (CCL)* library in robot behavior modeling. *CCL* provides several software components, which allow the model to be built, simulated and formally verified. Due to the integration with the *Robust* library the *CCL* models can be deployed and executed on the actual hardware platforms. Besides the modeling robot behavior, the work also addresses the problem of modeling a robots environment.

The *CCL* models can be verified either formally or by simulation. Since the use of formal methods is always associated with the state explosion problem, the work provides practical guidelines on how to deal with this problem using *CCL*.

1 Introduction

In recent years, increased interest in the design and building of robots has been visible. Robots have become accessible to a wide audience. The ease and availability of even sophisticated robotics platforms encourages researchers to seek new, efficient methods of modeling of control software for such constructions. One of them can be *Concurrent Communicating List (CCL)* - the *Clojure* language library supporting executable modeling of concurrent and distributed systems. It allows users to write a control program in a special *lisp-like CCL* notation, run it step by step in a simulation mode, perform their formal verification or execute them like a regular computer program.

The first two sections of this paper contain a brief outline of *AI* robotic architectures and, on this basis, tries to draw a map of various approaches to the modeling of AI robot software. Section 3 summarizes the *CCL* library. Section 4 presents a simple control algorithm allowing the robot to move and sense. Section 5 discusses *CCL* in model simulation and formal analysis. Finally, Section 6 includes a work summary and presents the plan for future research and development.

2 Robotics Models and Architectures

The architecture design in mobile AI robotics tries to follow the three intelligent control architectural styles [2]:

– Hierarchical Planning and Control Architecture
– Reactive/Behavior Based Control Architecture
– Hybrid Architecture

One of the most influential representatives of the first approach is *"A Reference Model Architecture of Intelligent Control"* proposed by *J. S. Albus* and *A. M. Meystel* later on implemented as *4D/RCS* [1]. The Albus model provides several levels of control nodes, where each of them is able to sense the environment, judge the situation on a certain level of granularity and generate behavior. The nodes higher in the hierarchy take strategic decisions and perform actions on the higher level of abstraction, whilst the nodes lower in the hierarchy have the shorter time perspective and perform simpler actions. All the nodes maintain the data base (world model) storing important facts about the environment.

Another architectonic style is determined by the famous *Subsumption Architecture* proposed by Brooks [5]. Following the principle *"The world is its own best model"* it focuses on immediate data sensing and behavior generation rather than spending time on the possibly resource-consuming: sense, process the knowledge, and execute the plan processing loop. Due to the relative simplicity and intuitiveness of model creation, the reactive approach resulted in a number of works on the various frameworks and notations supporting behavior modeling and analysis [19,21,18,8,17].

The third, the hybrid approach tries to take benefits from both hierarchical planning and a reactive approach. Its supporters argue that the previous two approaches in fact do not exclude each other but rather try to perceive the same phenomenon from two different perspectives. They observe that sometimes intelligent constructions need to behave in a reactive manner, and at other times to perform careful knowledge-based hierarchical planning. Example of this approach is *AuRA* [14].

Among the papers that focus on modeling system behavior, there is an important group of works that use formal methods. Using formalisms allows the system behavior to be specified more clearly and efficiently, and opens the possibility of using formal techniques for validation and verification of the model. An example of such an approach is *Behavior Language* [6]. This is directly derived from the widely recognized *Subsumption Architecture* [5]. Its syntax is based on the *AFSM (Augmented Finite State Machines)* description language, which allows the model to be compiled and deployed on different hardware platforms such as *Motorola 68000* or *Hitachi 6301*. Other robot behavior specification languages using the state machine concept are *COLBERT* [10] and *XABSL* [19]. The first of them, supporting the *SAPHIRA* platform, is designed for modeling behavior of individual robots, whilst *XABSL* tries to address the problem of behavior specification for multi-robot systems. In addition, *process algebras* [4] or *behavior trees* are represented as formalisms for modeling robots behavior [9,20]. *Petri Net Plans (PNP)* [22] proposes *Petri nets* layer as an actual model specification language, and then offers possibility of formal model verification. Although formal methods are often used for modeling behavior, the reactive systems that use them also take benefits from hierarchical approach. An example of solution, which tries to use both reactive and hierarchical methods is *RS (Robot Schemas)* [15].

CCL notation is derived from process algebras and primarily focuses on behavior modeling. It defines operators and actions (as with the process algebras) which are used later to create more complex expressions forming a model specification. There are two types of communication, internal, between two different processes within the model, and external, between the model and the rest of the system (e.g. *world model*). Such a distinction provides modularization, since once module can be completely external to the other module. *CCL,* due to its close relationship with process algebras, gives the possibility to perform formal verification of the model. Some operations, such as *deadlock* finding or *bisimulation* checking, are supported directly by the *CCL* library. Others, such as temporal formula verification, are supported by exporting the model into the *CADP* tool [7]. *CCL* is executable, which means that all the models can be freely simulated and executed. For the purpose of simulation, the external environment can be modeled using the *CCL* simulation environment *CCL Sim* [12].

3 CCL Library at a Glance

3.1 CCL Notation

The *CCL* syntax is modeled on the process algebras, such as *CCS* [16], and the *Clojure* and *Lisp* language. For this reason all the *CCL* expressions are in the form of lists, and they are built up from the operations, which are close in meaning to what can be found in algebraic notations. The *CCL* model consists of lists denoting processes, communication channels between them, and primitives – *Clojure* functions, which are called by the processes during the execution of a model. The basic notion introduced by the *CCL* notation is the *nlist* expression denoting the sequence of operations to execute. The *nlist* expressions are executed within the *CCL* processes launched as the part of the concurrent composition clause. Processes can be anonymous or named. A brief *CCL* syntax summary can be found in *Table* 1[1]. The *CCL* processes communicate via blocking queues. The use of the synchronization queue mechanism is possible through the set of queue access methods, such as: q-get, q-put, q-peek, q-try-put, q-size and q-capacity. The q-get and q-put functions add and remove elements from the queue. These functions can be blocking or non-blocking depending on the adopted strategy and the number of elements in the queue. The next two functions q-peek and q-try-put behave like q-get and q-put but they do not wait. When they fail the *nil* value is returned. The last two functions do not change the state of the queue. They return the number of actual elements in the queue (method: q-size) and return the maximal possible size of the queue (method: q-capacity). Depending on the adopted policy and the length of the queue, the processes attempting to read from or write into the queue can be blocked for a while or returns immediately.

[1] The more comprehensive syntax reference with examples can be found at www.kulakowski.org/ccl

Table 1. *CCL* - Syntax summary

Construction	Description
(defn Foo [] body-expr) (reg-as-prim Foo)	Defines the *Clojure* function *Foo* and registers it as a *CCL* primitive.
def-nlist Boo (exp$_1$ exp$_2$ … exp$_N$))	Defines the *nlist Boo* executing its nlist-body i.e. the list of subsequent expressions: exp_1, $exp_2, \dots, exp_N$.
(def-nlist (Roo :y) ((exp$_1$:y) … (exp$_k$:y))	Defines the *nlist* named *Roo* with the initial parameter :y. The expressions in *Roo's* nlist-body can freely use the parameter :y.
((:x (Foo)) (Roo (+ :x :y)))	Defines the local *nlist* variable :x and initializes it to the value returned by *Foo*, then starts execution of *Roo* with the input value set to the sum of :x and :y.
(? (cond$_1$) (nlist$_1$) … (cond$_N$) (nlist$_N$))	The conditional choice operator allows the definition of the nlist-expression to be executed next depending on their condition expressions, i.e. if cond$_k$ is the first true expression on the left then the nlist$_k$ expression is to be executed.
(?? X$_1$ (nlist$_1$) X$_2$ (nlist$_2$) … X$_N$ (nlist$_N$))	Within the random choice statement, nlists are picked for further execution randomly. The chance of being selected for nlist$_k$ is given as: $x_k/\Sigma(x_1,\dots,x_k)$
(| Moo :moo Goo :goo)	As a result of execution of this expression two *CCL* processes have been launched, where the first process labeled *:moo* will execute the *nlist Moo*, whilst the second process *:goo* will execute the *nlist Goo*.

Due to the blocking property and the maximal number of elements in the queue (it is assumed that a queue can be zero-length or non zero-length) there are eight possible types of synchronization queue. All of them have been summarized in Table 2. There are five columns, where *type* means the type id of a synchronization queue, *cap.* comes from the maximal capacity of the queue, *read* and *write* determines whether the operations read and write are blocking and non-blocking. Since these parameters affect the meaning of queuing methods, the fifth table column contains a brief function semantics summary. Synchronization queues are used for modeling communication between different processes within the model. Communication between the external environment and the model is implemented by primitives call (Table 1). In such a case all the technicalities of a communication channel are hidden and it is assumed that the function call returns a correct result as soon as possible.

Table 2. Synchronization queues - functions meaning

Type	**Cap.**	**Read**	**Write**	**Functions meaning**
1	0	n-b	n-b	The 0-length queue is always empty. Thus, all the operations except `q-size` and `q-capacity`, are ineffective.
2	0	b	n-b	Since at the given point of time the queue is empty (there is no space to store the element for any non-zero period of time) the operations `q-peek` and `q-try-put` are ineffective. The function `q-get` always blocks and waits for the counterpart `q-put`. The function `q-put` always adds the element to the queue. If there is no waiting `q-get` on the other side the inserted element is lost.
3	0	n-b	b	As for type 2 operations, `q-peek` and `q-try-put` are ineffective. The function `q-put` always blocks and waits for the counterpart `q-get`. The function `q-get` removes the element from the queue. If there is no waiting `q-put` on the other side the returned element is *nil*.
4	0	b	b	As for type 2 operations, `q-peek` and `q-try-put` are ineffective. The function `q-put` always blocks and waits for the counterpart `q-get`, and reversely the function `q-get` always blocks and waits for the counterpart `q-put`. When both functions meet each other `q-get` returns the element inserted by `q-put`.
5	$k > 0$	n-b	n-b	The functions `q-peek` and `q-try-put` are ineffective, since they work as `q-put` and `q-get`. The function `q-get` is successful if the queue is non-empty, `q-put` when the queue is not full.
6	$k > 0$	b	n-b	The function `q-try-put` is ineffective. The function `q-get` blocks until the queue is empty. The function `q-put` fails immediately when the queue is full.
7	$k > 0$	n-b	b	The function `q-peek` is ineffective. The function `q-get` fails immediately when the queue is empty. The function `q-put` blocks as long as the queue is full.
8	$k > 0$	b	b	The function `q-get` blocks as long as the queue is empty, and similarly the function `q-put` blocks as long as the queue is full.

CCL notation provides an *externalization* mechanism which allows the synchronization queue to be wrapped within the primitives call, so that the explicit communication link between two processes becomes external to the model. This leads to a decrease in model complexity[2] as regards the number of inter-process synchronizations, and finally may result in splitting one model into several sub-models. Thus, the externalization mechanism introduces modularity, so that one

[2] Of course, at the expense of model accuracy.

model can be independently modeled and analyzed from the others. This property seems to be especially useful when different parts of the model are loosely coupled as, for example, a sub-model of a robot and the sub-model of its environment. The *CCL* library[3] provides a few *APIs* allowing the model of a system to be created, executed or simulated, and formal analysis of a model to be carried out. In addition, the *CCL* software bundle contains *CCL Sim* [12], an interactive model development environment facilitating step-by-step tracking of the model and building various mockups helping simulation of an external environment model.

3.2 CCL Software Setup

One of the key component of the *CCL* software setup is the *Robust* platform. This was originally conceived as a simple *Mindstorm NXT Java* library moving *CPU* intensive processing to a *PC* platform and providing a robust and efficient *PC-NXT* communication link. With time, *Robust* gained new components allowing the creation of control programs running on another robotic platform *Hexor II* [13], and the *cljRobust API* [11] interfacing *Robust* with the *Clojure* programming language. In this way *cljRobust API* functions can be declared as *CCL* primitives, then the models written in *CCL* notation can actually control the mobile robots supported by the *Robust* library. Such a tool-chain involves a few additional, not explicitly mentioned yet, software components. In the case of the *NXT* computer LeJOS - the embedded version of *Java* for *Mindstorms NXT* is required. Hexor II comes with its own operating system and proprietary control libraries. The *Robust* library as well as *CCL* are run under the control of a Java Virtual Machine. The same applies to the *Clojure* language library which binds the *Robust* platform and *CCL APIs* together. When working with *CCL* models, choosing one of a few professional *Clojure* developer environments[4] is worth considering.

4 Modeling Robot Behavior - Study Case

One of the basic *CCL* constructions is primitive. From the system modeling perspective a primitive is like an indivisible action, which can take some parameters from the model and return the computed value. Implementation details behind the primitive call are not important except for the fact that primitives should be interruptible, i.e. as functions executed within the JVM threads they should be able to safely break ongoing operations when the interruption request is raised. Since the primitive is able to transmit the values to and from the model, it can be used for implementing communication between the robot model and the robot model's environment. Due to the externalization mechanism, there is no

[3] The *CCL* library binaries, manual and examples are available at `www.kulakowski.org/ccl`

[4] There are, for instance: Eclipse with counterclockwise plugin and Net Beans with enclojure plugin.

need to fix the model boundaries at the very beginning, and the designer is able to decide later on where the robot model stops and where the model of the environment starts. Let us consider a simple reactive robot with one touch sensor (bumper) sending a short stimulus when the robot hits the obstacle. Implementing that with *CCL* and *Robust* requires definition of the synchronization queue touch-event-source (Listing: 1, line: 1) and definition of the *Clojure* function touch-handler (Listing: 1, line: 2) being an event listener hooked up in *Robust API*. When the *bumper* hits the obstacle, touch-handler puts an element into the touch-event-source queue (Listing: 1, line: 4).

```
 1 (def-queue touch-event-source :size 1 :rb)
 2 (defn touch-handler [value]
 3   (if (= value 1)
 4     (agh.ccl.nlists/q-put :touch-event-source 1)))
```

Listing 1: Communication between the robot model and its environment - executable version

In response to the appearance of an element in the queue the robot should retreat a little bit the same way it came and then choose the other direction. That simple behavior can be easily specified using standard *CCL* constructions.

```
 5 (def-nlist ExplorationRobot
 6     ( (rb-system-startup)
 7     (rb-touch-async-handler touch-handler)
 8     (rb-move-forward (rnd 200 400) 200)
 9     (| GoAhead :goAhead CollisionDetector :colDetector)))
10 (def-nlist GoAhead
11     ( (! AvoidObstacle)
12     (rb-move-forward (rnd 200 400) 200)
13     (rb-move-wait-for-new-move)
14     GoAhead))
15 (def-nlist AvoidObstacle
16     ((rb-move-forward 200 -100) ; move backward
17     (rb-move-inplace-turn (rnd -120 120) 100)
18     (rb-move-forward (rnd 200 400) 200)
19     GoAhead))
20 (def-nlist CollisionDetector
21     ( (q-get touch-event-source)
22     (rb-move-stop-now)
23     (-> :goAhead)
24     CollisionDetector ))
```

Listing 2: Random exploration. CCL/Robust executable behavior specification

The first *nlist* expression ExplorationRobot (Listing: 2, line: 5) calls the mandatory *Robust* initialization function rb-system-startup (line: 6), registers the touch-handler, puts on the execution queue the one *move forward* command, and then launches two threads :goAhead and :colDetector. They start executing correspondingly the GoAhead (line: 10) and CollisionDetector (line: 20) expressions. The first action of the GoAhead expression is to register (operator !) an interruption handler AvoidObstacle (Listing: 2, line: 11). Thus, when the interruption request has been raised, the :goAhead thread immediately starts processing the AvoidObstacle expression. Next GoAhead follows the processing loop: queues one straightforward move with a randomly chosen length between 200 and 400 millimeters and speed 200 (line: 12), waits until the currently executed move ends (line: 13), and starts execution from the beginning (line: 14). In the case of the robot's bumper hitting into an obstacle, the *CCL* process :goAhead is interrupted and the fallback procedure is executed. In such a case, the AvoidObstacle *nlist* expression is executed (Listing: 2, lines: 15 - 19), i.e. after withdrawal of the robot 100 units back (line: 16), the new random direction is chosen (line: 17), and the construction continues moving ahead (line: 18). `CollisionDetector` (Listing: 2, lines: 20-24) is the last expression in the *Random Exploration* example. It is designed as a collision listener, which in the case of collision immediately stops the whole construction (line: 22) and interrupts the :goAhead process execution (line: 23).

5 Model Simulation and Formal Verification

Although the model as presented on Listings 1 and 2 is fully executable[5] its simulation and formal analysis require the introduction of several additional enhancements. For the purpose of simulation, due to the lack of the *Robust* library, all the functions referring to the external environment provided by *Robust* need to be replaced by mockups or modeled in *CCL* as sub-models. The *CCL* library supports simulation experiments by providing the additional GUI application *CCL Sim* [12], together with a *ccl-sim-utils* API allowing for creation of primitives controlled remotely from within the *CCL Sim*. Hence, every action made by the model upon the external environment can be logged, and every sensor reading request can be manually handled in *CCL Sim*. An example of mockup implementation of (Listing: 3) first waits a random amount of time (no longer than 200 milliseconds, and not shorter than 100 milliseconds), then inserts a log entry, which shows up in the *CCL Sim* application's dashboard.

```
(defn rb-move-inplace-turn [x y]
  (do (wait 100 200)
      (ccl-sim-model-log-writer "rb-move-inplace-turn" x y)))
```

Listing 3: An example cljRobust API mockup implementation

Although *CCL Sim* can capture all the I/O communication between the model and its environment, it does not allow the external world to be modeled. Its functionality is limited to receiving data from the model and sending the manually chosen or automatically pre-specified values back to the model. Such a solution

[5] The whole model code, together with a short movie showing the model execution can be found at:
`http://www.kulakowski.org/ccl/`

works very well when the model need to be debugged, but it is less useful in the case of the long-term simulation runs. In the second case, the external environment needs to be modeled as a separate sub-model TouchHandler. In the considered example TouchHandler sends (Listing: 4) in a loop an interrupt request (Listing: 4, line: 29) then waits a random amount of time (line: 30) and starts its execution once again (line: 31).

```
28 (def-nlist TouchHandler (
29     (q-put touch-event-source 1)
30     (wait 300 700)
31     TouchHandler))
32 (def-nlist TouchWorld (| TouchHandler ExplorationRobot))
```

Listing 4: An example cljRobust API "dummy" implementation

Now the model (expression *TouchWorld*) is self-contained in the sense that there is no explicit synchronization queue leading outside the model. Thus, it is possible to generate a graph where nodes represent states of the model and arcs between them possible state transitions, and then perform their formal analysis. For formal reasons it is convenient to call such a graph as a labeled transition system *(LTS)* [3]. The *LTS* for such a simple model has 261 states and 707 transitions[6] - in this approach the state in this formalization is represented by the set of states of synchronization queues and the set of states of all the processes. In the adopted approach the values of variables are not taken into account during the *LTS* construction, thus the state of a synchronization queue is reduced to its length, and the every deterministic choice *?* is reduced to its non-deterministic counterpart *??* (Table 1). The change of state is determined both by the primitive call and the operator evaluation. Thus, the transitions can take the labels of both primitives and operators (Fig. 1).

Using *CCL* shell its easy to check that the *TouchWorld* model is deadlock free, compare the *LTS* with another *LTS* in terms of the weak and strong bisimulation [4] or export it into *CADP* [7] and check other temporal properties like *safety* [3]. Sometimes it is convenient to analyze only a part of the model. In such a case all the synchronization queues leading outside the sub-model of interest need to be wrapped into primitives. Thus, in the case of TouchWorld, to be able to to separately analyze ExplorationRobot and TouchHandler, the queue touch-event-source needs to be externalized. For this purpose the queue definition gets a new flag :ext (Listing: 5, line: 33), and both queue ends need to be accessed through the wrappers (lines: 34 - 35). After replacing the queue operations q-get and q-put by their wrapper counterparts the model is ready to be analyzed locally (Listing: 5).

The touch-event-source externalization reduces *LTS* almost ten times to 35 states and 62 transitions (Fig. 1). Of course, the reduced *LTS* loses the information related to states of the sub-model mimicking the robots environment, thus it is impossible to automatically prove that every state of the environment

[6] Obtained by calling the *CCL* shell command (gen-lts WorldModel).

```
(def-queue touch-event-source :size 1 :rb :ext)
(defn-queue-wrapper bumper-stat-out touch-event-source :get)
(defn-queue-wrapper bumper-stat-in touch-event-source :put)
(reg-as-prim bumper-stat-in bumper-stat-out)
(def-nlist TouchHandler ((bumper-stat-in :touch-event-source 1)
    (wait 300 700) TouchHandler))
(def-nlist CollisionDetector (
    (bumper-status-out)
    ...
```

Listing 5: An example cljRobust API "dummy" implementation

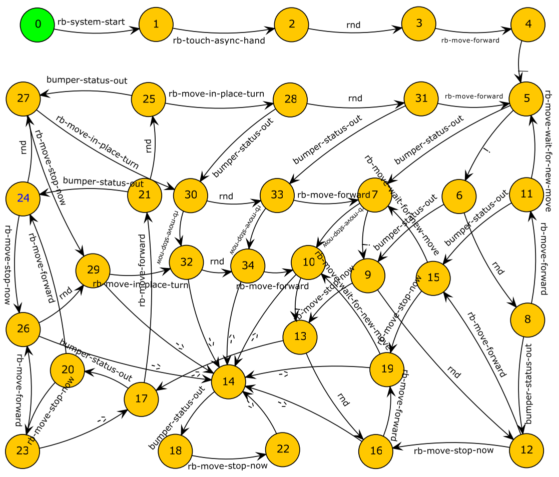


Fig. 1. LTS Graph for ExplorationRobot after touch-event-source externalization

is covered by the appropriate behavior of a robot, but still some important and sound model properties can be proved. That is because, when wrapping the q-get function into the bumper-stat-out primitive, the implicit assumption was made that the bumper-stat-out function behaves like any other function i.e. when called it returns some value in a finite amount of time. Since the robot moving straight forward all the time should sooner or later hit the obstacle, it seems to accurately reflect the actual behavior of the robot under control of the ExplorationRobot algorithm. It is also consistent with the behavior modeled by the TouchHandler expression. Thus, marking communication between TouchHandler and ExplorationRobot as external (thus not considering it during ExplorationRobot sub-model analysis) does not limit verifiability of other temporal model properties, such as deadlock freedom or liveness. Moreover, the reduced *LTS* is small enough to be browsed and analyzed manually. Of course, there is no one golden rule for deciding when the communication channel can be hidden by externalization. In general, it is assumed that the states of models on both ends of the synchronization queue are loosely coupled, hence the omission of one sub-model will have little effect on other sub-models. Of course, by excluding the synchronization queue out of the model a designer runs the risk of losing something important, so eventually he has to decide whether this will affect the property he wants to examine.

6 Summary and Future Work

In this paper the new *CCL (Communicating Concurrent Lists)* notation and its application to robot behavior modeling has been presented. The proposed notation is supported by the *CCL* library, which offers several software components allowing for model building, model execution, model simulation and debugging and formal analysis and verification of the model. The *CCL* library integrates well with *cljRobust* and the *Robust* library, thus all the models created and verified in the *CCL* notation can be easily executed on the actual hardware platforms.

The article also tackles the hard problem of modeling the boundary between the model of robot behavior and the external environment. The *CCL* library addresses the problem by providing the *CCL Sim* simulation environment, which can imitate the outer environment, and allowing users to write the model of the external world directly in the *CCL* notation. In the latter case, sometimes it makes sense to separately analyze the model of robot behavior and the model of the surroundings. *CCL* facilitates such analysis by providing externalization - an effective syntactic mechanism supporting sub-model separation.

Although the *CCL* library is ready to download and use, still a lot of problems need to be addressed. Since, initially, the *CCL* was designed as a set of *Clojure* macros rather than a regular modeling language, the syntax error information is difficult to understand for end-users. Thankfully, work on the new *CCL* parser is already underway. At the moment the *CCL* library supports only a limited number of predefined formal methods itself. Thus, the project will also try to provide methods which allow for easy construction of any temporal formula.

Acknowledgment. This research is partially supported by AGH University of Science and Technology, contract no.: 11.11.120.859.

References

1. Albus, J.S., et al.: 4D/RCS: A Reference Model Architecture For Unmanned Vehicle Systems Version 2.0. Technical report, NIST Interagency (2002)
2. Arkin, R.C.: Intelligent Control of Robot Mobility, ch. 16. Wiley (2007)
3. Baier, C., Katoen, J.: Principles of model checking. The MIT Press, Cambridge (2008)
4. Bergstra, J.A., Ponse, A., Smolka, S.A. (eds.): Handbook of Process Algebra. North-Holland (2001)
5. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE J. Robot. and Auto. 2(3), 14–23 (1986)

6. Brooks, R.A.: The behavior language; user's guide. Technical report, Massachusetts Institute of Technology, Artificial Intelligence Laboratory (1990)
7. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2010: A Toolbox for the Construction and Analysis of Distributed Processes. In: Abdulla, P.A., Leino, K.R.M. (eds.) TACAS 2011. LNCS, vol. 6605, pp. 372–387. Springer, Heidelberg (2011)
8. Groves, W., Collins, J., Gini, M.: Visualization and analysis methods for comparing agent behavior in TAC SCM. In: AAMAS 2009: The 8th International Conference on Autonomous Agents and Multiagent Systems (May 2009)
9. Hardey, K., Mattis, M., Goadrich, M., Corapcioglu, E., Jadud, M.: Exploring and Evolving Process-oriented Control for Real and Virtual Fire Fighting Robots. In: Proceedings of Genetic and Evolutionary Computation Conference (2012)
10. Konolige, K.: COLBERT: A Language for Reactive Control in Sapphira. In: Brewka, G., Habel, C., Nebel, B. (eds.) KI 1997. LNCS, vol. 1303, pp. 31–52. Springer, Heidelberg (1997)
11. Kułakowski, K.: cljRobust - Clojure Programming API for Lego Mindstorms NXT. In: Jędrzejowicz, P., Nguyen, N.T., Howlet, R.J., Jain, L.C. (eds.) KES-AMSTA 2010, Part II. LNCS, vol. 6071, pp. 52–61. Springer, Heidelberg (2010)
12. Kułakowski, K.: CCL Sim, the simulation environment for concurrent systems. In: Proceedings of Dependability and Complex Systems, DepCoS (2012)
13. Kułakowski, K., Matyasik, P.: RobustHX - The Robust Middleware Library for Hexor Robots. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 241–250. Springer, Heidelberg (2010)
14. Long, L., Hanford, S., Janrathitikarn, O.: A review of intelligent systems software for autonomous vehicles. In: IEEE Symposium on Computational Intelligence in Security and Defense Applications, CISDA (2007)
15. Lyons, D.M., Arbib, M.A.: A formal model of computation for sensory-based robotics. IEEE Transactions on Robotics and Automation 5(3), 280–293 (1989)
16. Milner, R.: Communication and Concurrency. Prentice-Hall (1989)
17. Nalepa, G.J., Biesiada, B.: Declarative Design of Control Logic for Mindstorms NXT with XTT2 Method. In: Jędrzejowicz, P., Nguyen, N.T., Hoang, K. (eds.) ICCCI 2011, Part II. LNCS, vol. 6923, pp. 150–159. Springer, Heidelberg (2011)
18. Rai, L., Kook, J., Hong, J.: Non-Deterministic Behavior Modeling Framework for Embedded Real-Time Systems Operating in Uncertain Environments. Journal of Information Science and Engineering 26(1), 83–96 (2010)
19. Risler, M., von Stryk, O.: Formal Behavior Specification of Multi-Robot Systems Using Hierarchical State Machines in XABSL. In: Workshop on Formal Models and Methods for Multi-Robot Systems, pp. 1–7 (August 2008)
20. Xiao, W., Liu, T., Baltes, J.: An intuitive and flexible architecture for intelligent mobile robots. In: The Second International Conference on Autonomous Robots and Agents (ICARA), Palmerston North, pp. 52–57 (2004)
21. Zhang, Q., Zhang, Y.-F., Qin, S.-Y.: Modeling and analysis for obstacle avoidance of a behavior-based robot with objected oriented methods. Journal of Computers 4(4), 295–302 (2009)
22. Ziparo, V.A., Locchi, L., Nardi, D., Palamara, P.F., Costelha, H.: Petri net plans: a formal model for representation and execution of multi-robot plans. In: AAMAS 2008: Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems (May 2008)

Visual-Trace Simulation of Concurrent Finite-State Machines for Validation and Model-Checking of Complex Behaviour

Robert Coleman, Vladimir Estivill-Castro, René Hexel, and Carl Lusty

Griffith University, Nathan 4111, QLD, Australia
`www.mipal.net.au`

Abstract. Simulation of models that specify behaviour of software in robots, embedded systems, and safety critical systems is crucial to ensure correctness. This is particularly important in conjunction with model-driven development, which is highly prevalent due to its numerous benefits. We use vectors of finite-state machines (FSMs) as our modelling tool. Our FSMs can have their transitions labeled by expressions of a common sense logic, and they are more expressive than other modelling approaches (such as Behavior Trees, Petri nets, or plain FSMs). We interpret the models using the same round-robin scheduler which is integrated into the simulator. Execution on a platform is exactly the same as in the simulator (where sensors and actuators are masqueraded by proxies) and coincides with the generator of the Kripke structure for formal model-checking. In three ubiquitous case studies we show that our simulation discovers issues where those models were incomplete, ambiguous, or incorrect. This further illustrates that simulation and monitoring need to complement formal verification.

Keywords: simulation, testing and validation of robot software, interpretation of models, model-checking, modeling framework for robots, software platform and middleware for robotics.

1 Introduction

While software development for autonomous robots commonly starts with a modelling phase, enough confidence in the correctness of behaviour can only be obtained through simulation in conjunction with formal model-checking.

Model-driven engineering minimises the programming phase as correct, and validated models can directly run on different platforms. Model-driven development (MDD) has been shown to be fast, as it provides higher level of abstraction than traditional programming languages. Models are automatically transformed into efficient, working software. In fact, models can more succinctly represent functionality. MDD is more cost-effective, due to a shorter time to deployment, but changes to models and traceability of functionality are more direct and transparent. This also leads to increased quality. MDD is less error-prone if we can perform testing, validation, and simulation of models. Therefore, MDD leads to meaningful validation, as it is the high-level model that is validated, because automatic translation to lower levels has perfectly defined semantics. Compare this with the translation process that human developers perform (where use-cases are translated into implementation by programmers). For robotics, MDD offers far more benefits. We certainly want to be sure the software is correct before we deploy expensive hardware in some environment and risk the physical integrity of the autonomous robot or others in the environment. We need software that is less sensitive to changes in requirements, and model-driven development provides this. We want to enable domain experts (environment experts) to participate more closely in defining the behaviour of autonomous robots. MDD empowers such domain experts as behavioral requirements can be captured in the models (in some cases, the models provide up-to-date documentation). Moreover, MDD provides platform independence and focus on behaviour issues instead of technological details. But then, simulation, validation, and formal verification of the model become far more important than with traditional development (e.g., in a waterfall cycle, the next stage may correct the previous representation of requirements of functionality as it is tested in the implementation).

Modelling the behaviour of a system using state machines has a long history in software development [11]. State machines are central to modelling object-oriented systems since OML [18], they are also used in system engineering languages such as OMG SysML^{TM} [17]. Executable models appeared in executable UML [15]. We have advocated models that specify behaviour through several finite-state machines (FSMs), whose transitions are labeled by predicates of a common-sense logic [4]. We have shown that these models are more succinct that plain FSMs, Petri nets and Behavior Trees [4]. They are easy to comprehend, as the components of FSMs are used in many systems for formal validation or descriptions of automatic devices and protocols. The reactive nature of finite-state machines can be compensated by a reasoning component and domain-knowledge in logic, when we label transitions by a query to an inference engine. Formal verification of behaviour models (model-checking) has traditionally been complicated when several components concurrently operate to define the behaviour of a system. The vector of finite-state machines that constitute such a system can be scheduled deterministically reducing the execution path search space [9]. With more succinct Kripke structures far more model-checking of classic software engineering case studies has been achieved.

Despite this, formal-model checking is costly, and it may not be clear which properties need to be verified. This is where simulation and validation come into play. Simulation allows to identify behaviour that was not considered at the time of requirement elicitation. Trough our simulation, we can discover scenarios where a behaviour specification was incomplete. In this paper we discuss our solution to construct a simulator (and monitor) of a vector of finite-state machines that control autonomous robots (or embedded systems). State machines are core elements in embedded systems and in popular commercial tools widely used in the industry, including *QP*^{TM}[19] *StateWORKS* [24]. While our simulator has characteristics similar of the integration of *MathWorks*^{R} *StateFlow* with *Symlink* we avoid the complex translations that this requires for model-checking [1].

2 Modelling

Designers of robotic system and of autonomous systems naturally model using states and transitions. Thus, a finite-state behaviour model is actually a table. This transition table consists of 3 columns: (`Source_State`, `Boolean_Expression` `Target_State`). Here, the projection of the transition table on each state is a list, i.e., transitions are considered in (deterministic) sequence. Also, each state of the FSM has 3 sections: **OnEntry**, **OnExit**, and an **Internal** section. One state is designated as the initial state. When a machine is running, it will execute a ringlet. A ringlet is the loop that starts by taking a snapshot (a copy) of all variables. Then the **OnEntry** section is executed (only for the first iteration in a new state, otherwise `OnEntry` is omitted), proceeding to evaluating the Boolean expressions in sequence. If the expression of a transition evaluates to `true`, the transition fires, meaning, the **OnExit** activity is performed and then the state is updated to the target state of the transition, concluding the ringlet in this case. However, if the list of Boolean expression is exhausted and none evaluates to `true`, then the **Internal** section is executed and the ringlet terminates.

Multiple FSMs are executed sequentially in round-robin fashion, in the order they appear in. The interpreter performs one ringlet of one machine before moving to the next one in sequence.

Our interpreter is extremely efficient, capable of performing several thousand transitions per second on-board of an Aldebaran Nao. This particular aspect makes it suitable for evaluating models that not only represent high level behaviours (e.g. the Game Controller of the SPL), but also fast control loops, e.g. the head tracking a ball in robotic soccer.

Our FSMs use simple `C` statements[1] for states, and expressions for transitions (including function calls and evoking expert system evaluation of non-monotonic logic). The communication medium between modules of the software is based on a whiteboard architecture and details have been provided elsewhere [3].

It is important to emphasise that our modelling tool enables three types of variables: local variables within one FSM and variables that are shared by the FSMs in the sequence. The third type are external variables whose value can be updated or modified outside the sequence of machines, but shared with FSMs (e.g. to communicate with sensors or actuators).

3 Illustration

We now present three ubiquitous embedded systems case studies. These have been widely discussed in the literature of software engineering and in particular formal methods. However, we show that simulation and validation discovers emergent behaviour that was not anticipated or constitutes an ambiguity in the specification of requirements. Simulation not only validates the models, but also helps determine the set of properties for formal verification. The first case study will be the *Mine Pump*. We show here that simulation enables to disambiguate

[1] We use the grammar `SimpleC.g` from ANTLR (`www.antlr.org`).

the need for 3-position switches, at least for the supervisor. The second case is the *Microwave*, and again, we show that simulation resolves the meaning of the requirement "opening the door stops the cooking." The third case is the *Industrial Press*. We show here that simulation is essential to ensure that the behaviour would be as expected. For these cases, formal model-checking has been performed in the literature, but not all properties were elicited, and thus, models were, in fact, inaccurate (which we discovered through simulation). We focus on model development and simulation (model-checking of these models is discussed elsewhere [9].) Moreover, we highlight pointers to videos of these models running directly on actual robotic platforms (in some cases the same model is being interpreted in two distinct platforms) as promised by model-driven development.

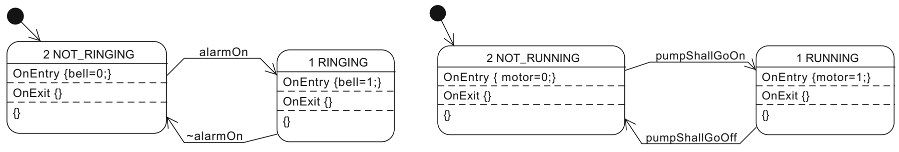
3.1 The Mine Pump

The mining pump is a case widely discussed in the literature [21,22,10,27] where software is controlling a safety-critical system. We follow Burns and Lister [6] where significant formality is provided. Here, we skim over the development of the model [9]. The requirements in natural language [6] are reproduced in Table 1. This study is again illustrative of the power of using FSMs with transitions labeled by a common-sense logic such as DPL. The pump is in one of two states, *running* or *not running*, while simultaneously the alarm is in a *ringing* or *not ringing* state. This is illustrated by the diagrams in Fig 1 (next page).

Table 1. Mine Shaft Pump Requirements

Req.	Description
R1	The pump extracts water from a mine shaft. When the water volume has been reduced below the low-water sensor, the pump is switched off. When the water raises above the high-water sensor it shall switch on.
R2	An human operator can switch the pump on and off provided the water level is between the high-water sensor and the low-water sensor.
R3	Another button accessed by a supervisor can switch the pump on and off independently of the water level.
R4	The pump will not turn on if the methane sensor detects a high reading.
R5	There are two other sensors, a carbon monoxide sensor and an air-flow sensor, and if carbon monoxide is high or air-flow is low, an alarm rings to indicate evacuation of the shaft.

Our techniques reveal several issues with the specification. First, one must inspect the pre-conditions and post-conditions [6], to confirm that the conditions that turn the pump on are not the negation of those that turn it off. In particular, the pump goes on when the water level is high, but remains on when the level drops (Requirement R1) until it is below the low sensor before stopping. But it does not re-start as soon as the water level is above the below sensor. Unfortunately, Requirement R3 is seriously more ambiguous and the language used suggests that the operator's interface as well as the supervisor's interface for controlling the pump are both switches "of the same class" [6] (an assumption also shared by the Behavior Tree approach [10,27]). We argue this first interpretation is inconsistent with the requirements and that simulation reveals that switches that are either on or off, holding only two exclusive states, make no sense. Under this first interpretation (2-state switches), it is possible to construct



(a) States for the alarm (b) States for the pump

Fig. 1. Two FSMs control the mine pump

```
name{MINEPUMP}.  input{lowWaterSensorOn}.   input{highWaterSensorOn}.  input{operatorButtonOn}.  input{supervisorButtonOn}.
input{methaneSensorHigh}.

N0: {} =>  ~pumpShallGoOff.                                     N1: lowWaterSensorOn =>  pumpShallGoOff.  N1>N0.
N2: {~lowWaterSensorOn,~highWaterSensorOn,~operatorButtonOn}=> pumpShallGoOff.                           N2>N0.
N3: ~supervisorButtonOn => pumpShallGoOff.                      N4: methaneSensorHigh => pumpShallGoOff.  N3>N0.  N4>N0.

P0: {} =>  ~pumpShallGoOn.                                      P1: highWaterSensorOn =>  pumpShallGoOn.   P1>P0.
P2: {~lowWaterSensorOn,~highWaterSensorOn,operatorButtonOn}=> pumpShallGoOn.                               P2>P0.
P3: supervisorButtonOn => pumpShallGoOn.                        P4: ~supervisorButtonOn => ~pumpShallGoOn. P3>P0. P4>P3.
P5: methaneSensorHigh => ~pumpShallGoOn.                                                                   P5>P3. P5>P2. P5>P1.

output{b pumpShallGoOn,"pumpShallGoOn"}. output{b pumpShallGoOff,"pumpShallGoOff"}.
```

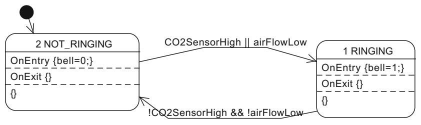
Fig. 2. DPL coding of the conditions for switching to the state `RUNNING` or to the state of `NOT_RUNNING`

a model and simulate it. We label the transition from `NOT_RUNNING` to `RUNNING` by the predicate `pumpShallGoOn` which we consider a request to an expert to indicate to us whether the pump shall be running on or not. The expert makes its judgment based on information about the low-water sensor, the high-water sensor, the operator and supervisor buttons, and the methane sensor. In DPL, having information on all of these inputs is not necessary (but desirable). We state this with the `input` section of the DPL code in Fig. 2.

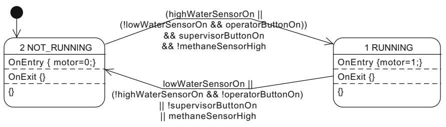
When shall the pump move from on to off? If we have no information, it shall remain on and not move (Rule `N0`). But if the low-water sensor in on, the pump switches off (this is Rule `N1`, which takes over Rule `N0`). Rule `N2` says that at a water level above *low* and below *high*, the operator can turn the pump off (`N2` takes precedence over `N0`). And the supervisor can turn the pump off, by Rule `N3` > `N0`. A high methane reading turns the pump off as well (Rule `N4`).

How to answer `pumpShallGoOn`? By default, the pump shall not go on; if asked this question when the pump is off, and in the absence of information, we should not recommend a change of state (this is Rule `P0`). Rule `P1` indicates that if the high-water sensor is on, then the pump goes on and this takes precedence over Rule `P0`. Rule `P2` indicates that the operator can turn the pump on if the water is between levels. Thus, `P2` overwrites `P0`. However, with `P3` and `P4` the supervisor can turn the pump on and off. Nevertheless, all these previous conditions are ruled out if methane is high (Rule `P5`). The reader may have noticed that we have stated Rule `P3` in contradiction with Requirement `R3`. This is because if we add the precedence `P3>P0` and write `P3:supervisorButtonOn=>pumpShallGoOn` enabling the switch of the supervisor to overrule the low-water sensor, compiling these rules obtains `pumpShallGoOff` $\equiv$ `!supervisorButtonOn || methaneSensorHigh`. That is, the water-levels do not matter at all in deciding if the pump shall be off (and therefore the operator is redundant as well). All depends on the methane level which, when high, overrules everything to switch the pump off (or on the supervisor switch that overrules everything else). Similarly, if the supervisor button overrules the operator button, the latter becomes redundant.

So here, one must accept that the low-water sensor takes precedence over the supervisor's switch, which makes sense if running the pump without water is dangerous. The logic theory as written in Fig. 2 provides this functionality. A fact that is verified with simulation in our methodology.

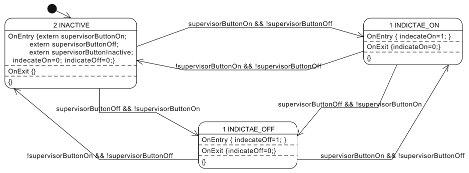


(a) Alarm state transitions

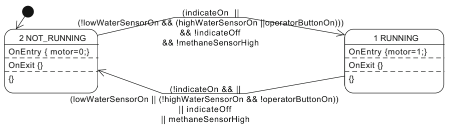


(b) Pump state transitions

Fig. 3. Two FSMs controlling the mine pump with two state-switches


2 INACTIVE
OnEntry {extern supervisorButtonOn;
extern supervisorButtonOff;
extern supervisorButtonInactive;
indecateOn=0; indicateOff=0;}
OnExit {}
{}
supervisorButtonOn && !supervisorButtonOff
1 INDICTAE_ON
OnEntry { indecateOn=1; }
OnExit {indicateOn=0;}
{}
!supervisorButtonOn && !supervisorButtonOff
supervisorButtonOff && !supervisorButtonOn
supervisorButtonOff && !supervisorButtonOn
1 INDICTAE_OFF
OnEntry { indecateOff=1; }
OnExit {indicateOff=0;}
{}
!supervisorButtonOn && !supervisorButtonOff
supervisorButtonOn && !supervisorButtonOff

(a) 3-state supervisor control (3SSC)



(b) Receiving signals from a 3SSC

Fig. 4. The model for the pump under 3-state supervisor control consists of these 2 FSMs and the FSM in Fig. 3a for the alarm

Compiling this DPL rule system using the `+c` option, we obtain equivalent `C` expressions for when the pump shall be on and for when it shall go off. Namely,

$$\texttt{pumpShallGoOff} \equiv \texttt{lowWaterSensorOn} \ \ || \ \ \texttt{!supervisorButtonOn}$$
$$|| \ \texttt{methaneSensorHigh} \ \ || \ \ (\texttt{!highWaterSensorOn \&\& !operatorButtonOn})$$

and $$\texttt{pumpShallGoOn} \equiv \texttt{!methaneSensorHigh} \ \ \texttt{\&\&} \ \ \texttt{supervisorButtonOn}$$
$$\texttt{\&\&} \ \ (\texttt{highWaterSensorOn} \ \ || \ \ \{\texttt{!lowWaterSensorOn \&\& operatorButtonOn}\}).$$

Notice the asymmetry between the supervisor and operator conditions for pump start! Combining this with the FSM for the alarm results in the model in Fig. 3. Nevertheless, we suggest that there is a second interpretation, i.e. the supervisor's interface is a three state control, that has an inactive state. In the inactive state, it does not overrule any of the other conditions that determine the running of the pump. But the supervisor can activate the switch to on, or off, switching the pump on and off regardless of water levels. That model for supervisor control is presented in Fig. 4 and the new logic theory appears in Fig. 5. Compiling the logic theory gives corresponding equivalent expressions:

$$\texttt{pumpShallGoOn} \equiv \texttt{!methaneSensorHigh \&\&!indicateOff\&\&}\left(\texttt{indicateOn.||}\left\{\texttt{!lowWaterSensorOn\&\&}\ [\texttt{highWaterSensorOn|| operatorButtonOn}]\right\}\right).$$
$$\texttt{pumpShallGoOff} \equiv \texttt{methaneSensorHigh} \ \ || \ \ \texttt{indicateOff} \ \ || \ (\texttt{!indicateOn}$$
$$\texttt{\&\&} \ \{\texttt{lowWaterSensorOn||} \ \ (\texttt{!highWaterSensorOn} \ \ \texttt{\&\& !} \ \texttt{operatorButtonOn})\}.$$

Replacing the transition labels in Fig. 1b, we obtain our model for the second interpretation of this case study. The Behavior Tree approach fails both interpretations of R3 and did not even verify any property regarding R3 [10,27].

Simulation uncovers these observations during development and helps define the properties to use for model-checking. This contrasts the properties under the two interpretations for the switches [9]. Finally, we can validate our models on actual platforms. We have enabled our interpreter to control a Lego NXT and built a model of the pump with touch-sensors and floating balls that, when the water level rises high enough, will trigger the sensor. Unfortunately the NXT only has 4 ports for sensors, thus we do not observe the behaviour of the alarm. Also, the 3-state supervisor button is simulated by cycling trough indicating on, off, and inactive for each momentary push of the corresponding touch sensor. Although this is a rather improvised artefact with unreliable sensors, we have a video that shows that the behaviour of the pump under the control of the model in Fig. 4 is correct. We trust the correctness derived from model checking as the utmost proof. A video of a pump executing the model appears on `youtu.be/y4muLP0jA8U`. We believe claims in the literature regarding the correctness of the model were incomplete because there was no simulation of the model that would have uncovered the above requirement issues.

```
name{MINEPUMP}.
input{lowWaterSensorOn}.  input{highWaterSensorOn}.  input{operatorButtonOn}.  input{methaneSensorHigh}.  input{indicateOn}.
input{indicateOff}.
P0: {} =>  ~pumpShallGoOn.            P1: highWaterSensorOn =>  pumpShallGoOn.                P1>P0.
P2: lowWaterSensorOn =>  ~pumpShallGoOn.                                                      P2>P1.
P3: {~lowWaterSensorOn,~highWaterSensorOn,operatorButtonOn}=> pumpShallGoOn.                  P3>P2. P3>P0.
P4: {~lowWaterSensorOn,~highWaterSensorOn,~operatorButtonOn}=> ~pumpShallGoOn.                P4>P3.
P5: indicateOn => pumpShallGoOn.      P6: indicateOff => ~pumpShallGoOn.                      P5>P2. P5>P4. P5>P0. P6>P5.
P7: methaneSensorHigh => ~pumpShallGoOn.                                                      P7>P5. P7>P3. P7>P1.
N0: {} =>  ~pumpShallGoOff.           N1: {~indicateOn,lowWaterSensorOn} =>  pumpShallGoOff.   N1>N0.
N2: {~indicateOn,~lowWaterSensorOn,~highWaterSensorOn,~operatorButtonOn}=> pumpShallGoOff.    N2>N0.
N3: indicateOff => pumpShallGoOff.    N4: methaneSensorHigh => pumpShallGoOff.                N3>N0.  N4>N0.
output{b pumpShallGoOn,"pumpShallGoOn"}. output{b pumpShallGoOff,"pumpShallGoOff"}.
```

Fig. 5. DPL coding of the conditions for switching to the state `RUNNING` using a 3-state supervisor control

3.2 The Microwave

A microwave is an example in software engineering [23] to illustrate modelling through states and transitions, but is also present in the context of model-checking [7, Page 39] as the safety feature of *disabling radiation when the door is open* is an analogous requirement to the case of faulty software on the Therac-25 radiation machine that caused harm to patients [2, Page 2]. The embedded software for the behaviour of a microwave oven is not only widely discussed in software modelling [14,20,24] but also as part of behaviour engineering [25,8,16].

Details of this example have been discussed previously [5]. Suffice it to say that one of the requirements is that opening the door shall stop the cooking. But this says nothing about what happens with the timer, and through modelling and simulation we discover that opening the door could *pause* the timer, or could *clear* the timer, or could *not affect* the timer at all, which continues to count down. Modelling by FSMs, where the labels for transitions can be statements in a logic that demand proof, has been contrasted with plain FSMs, Petri nets, and Behavior Trees (relevant behaviour modelling techniques in software engineering) using this very prominent example [4]. FSMs produce smaller models, and clarify requirements. This is because the ambiguity of the requirements is detected at the simulation of the model. The model-driven approach enables the automatic generation of implementations for diverse platforms and programming languages, e.g., the same models can generate code in Java for a Lego Mindstorm (`youtu.be/iEkCHqSfMco`) as well as C++ for a Nao (`youtu.be/Dm3SP3q9_VE`). Note, we can use a robot to simulate the behaviour of a microwave in the same way as our visual simulation tool.

3.3 The Industrial Press

The industrial metal press [13] has been studied in the literature of model checking for failure analysis [10,12,26]. Table 2 reproduces the requirements [26]. Once

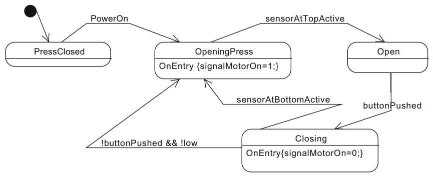
Table 2. Industrial Metal Press Requirements

Req.	Description
R1	The plunger is initially resting at the bottom with the motor off.
R2	When power is supplied, the controller shall turn the motor on, causing the plunger to rise.
R3	When at the top, the plunger shall be held there until the operator pushes and holds down the button. This shall cause the controller to turn the motor off and the plunger will begin to fall.
R4	If the operator releases the button while the plunger is falling slowly (above PONR), the controller shall turn the motor on again, causing the plunger to start rising again, without reaching the bottom.
R5	If the plunger is falling fast (below PONR) then the controller shall leave the motor off until the plunger reaches the bottom.
R6	When the plunger is at the bottom the controller shall turn the motor on: the plunger will rise again.

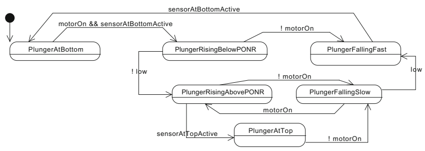
again, later papers [10,26] that cite the original source [13] capture requirements differently, or incompletely. For example, the requirement that once the plunger has come down, it shall stay down until the "human operator releases the button and inserts and removes sheets" [12]. In the original source [13] there is even an additional infrared-line sensor; once the plunger is down, the button must be released by the operator and the line cut for the plunger to move up again.

Because the On/Off button is not modelled correctly, and the infrared-link is not modelled at all, modelling as described in the Design Behavior Tree (DBT) [26, Figure 4 and HighResolution gif] results in a pathological cyclical behaviour of the system: while the human operator keeps the button pushed (and the power is on), the plunger rises to the top (with the motor on), reaches the top (the motor goes off) falls down, and rises up again.

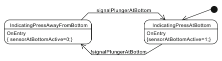
First, we have constructed a model that reproduces the DBT [26, Fig. 4] to focus on the model checking aspects. This model that mimics such a DBT appears in Fig. 6. We will not elaborate here on the formal verification of the model. We rather emphasise that the properties verified in the literature are not sufficient to completely validate the model. However, our analysis with a simulator does suggest a correction (shown in Fig. 7) that enables to remove the anomaly. See youtu.be/FpVUSrvLI0c for a video of the simulator operating concurrently on all the FSMs of the model for the corrected version. The video shows the progress of the FSMs, while sensor data is supplied trough the monitoring tool. The host computer acts as a robot through our whiteboard architecture and produces speech also from the FSMs. The plunger raises to the top and return back when falling slowly when the operator releases the button. However, once below the PONR, the release of the operator does not affect the fall. This again shows that our approach compares favourably with Behavior Trees. To further illustrate the effectiveness of modelling and simulation, we have deployed these models for execution on two platforms, a Nao robot and a NXT. A video (`youtu.be/blUpMdH14pM`) of the system emulating the behaviour as formalised from the Behavior Tree approach and its corrections is available. We emphasise that simulation is not a replacement for formal verification but it does enable elicitation of properties for a later stage of formal verification. Such model-checking is described elsewhere [9].



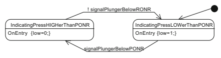
(a) States for the Controller



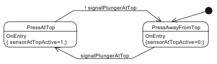
(b) States for the Plunger


signalPlungerAtBottom
IndicatingPressAwayFromBottom
OnEntry
{ sensorAtBottomActive=0;}
IndicatingPressAtBottom
OnEntry
{sensorAtBottomActive=1;}
!signalPlungerAtBottom

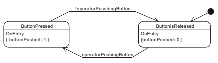
(c) Bottom Sensor



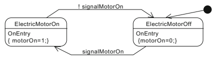
(d) PONR Sensor



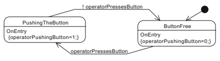
(e) Top Sensor



(f) Button



(g) Electric Motor



(h) States of the Operator

Fig. 6. Complete model of the industrial press that mimics the Design Behavior Tree [26, Fig. 4 and high-res gif]

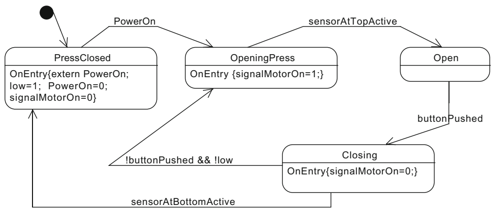


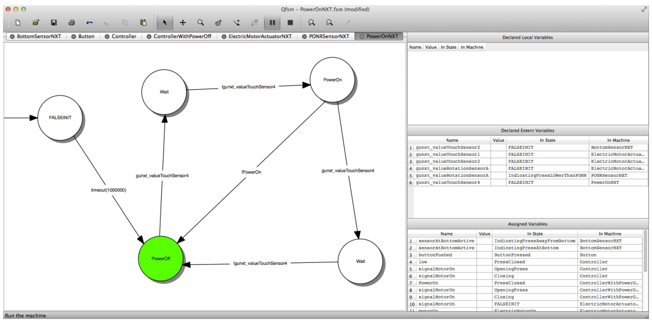
Fig. 7. Corrected model from Fig. 6a (based on DBT modelling from the literature). This model correctly pauses for a signal that restarts the system.

4 Simulation and Monitoring Tools

The execution of the models is performed by an interpreter based on data structures that represent a vector of FSMs in standard `C++`. The data structures for each FSM represent the states and the transitions, as well as the ANTLR expression tree for transition labelling. This is not only useful to obtain the Kripke structure of a vector of FSMs and perform model-checking, but also as an interpreter used for simulation. We have extended the publicly available `qfsm` (qfsm.sf.net) source code to implement the semantics of our FSMs. First, the states have the **OnEntry**, **OnExit** and **Internal** sections and the transitions can be Boolean expressions. However, the most important direction is that, while this tool was originally simply an editor of FSMs, we have converted it into a visual simulator of finite state machines.

Our tool enables the visualisation of the execution of a vector of FSMs highlighting the running state while keeping the interface responsive (this required additional Qt signals and slots). Since the model of a system is a vector of FSMs, we created a tabbed interface to enable the rapid visualisation for each FSM involved as a component of a larger model.

The simulator has a tightly coupled monitor that displays the values of the variables used in the FSMs (locals, globals shared, and external ones on the whiteboard. This enables the visualisation of the effects on the values on the variables as the simulation progresses. The update of the variables is performed asynchronously by calling code to update the variable table when the syntax checker has finished parsing a state. The integration of the interpreter demanded the sub-classing of some interpreter classes and since ANTLR generates `C` code, a `C++` wrapper was written. A video (youtu.be/rceNij4IkJE) demonstrates usage of the simulator and how a vector of FSMs is constructed, the syntax of the simple `C` language are verified and the simulator operates.



Declared Local Variables

Name	Value	In State	In Machine

Declared Extern Variables

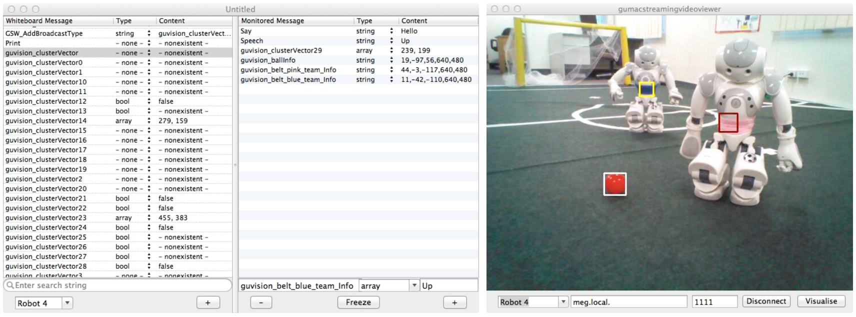
	Name	Value	In State	In Machine
1	gunxt_valueTouchSensor2		FALSEINIT	BottomSensorNXT
2	gunxt_valueTouchSensor1		FALSEINIT	ElectricMotorActua…
3	gunxt_valueTouchSensor2		FALSEINIT	ElectricMotorActua…
4	gunxt_valueRotationSensorA		FALSEINIT	ElectricMotorActua…
5	gunxt_valueRotationSensorA		IndicatingPressLOWerThanPONR	PONRSensorNXT
6	gunxt_valueTouchSensor4		FALSEINIT	PowerOnNXT

Assigned Variables

	Name	Value	In State	In Machine
1	sensorAtBottomActive		IndicatingPressAwayFromBottom	BottomSensorNXT
2	sensorAtBottomActive		IndicatingPressAtBottom	BottomSensorNXT
3	buttonPushed		ButtonPressed	Button
4	low		PressClosed	Controller
5	signalMotorOn		OpeningPress	Controller
6	signalMotorOn		Closing	Controller
7	PowerOn		PressClosed	ControllerWithPowerO…
8	signalMotorOn		OpeningPress	ControllerWithPowerO…
9	signalMotorOn		Closing	ControllerWithPowerO…
10	signalMotorOn		FALSEINIT	ElectricMotorActuato…
11	motorOn		ElectricMotorOn	ElectricMotorActuato…

Fig. 8. The interface of the simulator with the vector of FSMs for the an *Ambulatory Insulin Pump* and with the tab for the *Volume Sensor* active

The complimentary aspect to the simulator is a remote monitor that enables to visualise the execution of a vector of FSMs on an actual robot (Fig 9). This requires a distributed whiteboard that we have implemented using a time-triggered architecture and whose description will be presented elsewhere. With this idea, simulation is possible while tracing or monitoring the state of the execution, especially the values of the many variables involved in a complex model of several FSMs running in real-time. This monitor inspects the state of simulation by inspecting the message passing that happens with variables on our distributed whiteboard architecture. With this distributed whiteboard technology, is also possible to visualise in a model (a vector of FSMS) the switching states from our extended `qfsm`, although the vector of FSMs is executing remotely on a robotic platform. One important aspect of our monitor tool and our simulation tool is that the monitoring and the execution of the behaviour (a vector of FSMs) are the same, whether this is a simulation outside the robot or actual execution on the intended platform. The closest analogue to our ideas is Aldebaran's Coreographe. However, first, this is platform specific: the simulator is effective only for specific Nao robot versions. Aldebaran has chosen `SOAP` incurring large penalties to the relay of communication of its monitoring software. Because its wide open semantics, formal verification would result in exponential state explosion and is thus infeasible for formal model-checking approaches.



Whiteboard Message	Type	Content
GSW_AddBroadcastType	string	guvision_clusterVect...
Print	– none –	– nonexistent –
guvision_clusterVector	– none –	– nonexistent –
guvision_clusterVector0	– none –	– nonexistent –
guvision_clusterVector1	– none –	– nonexistent –
guvision_clusterVector10	– none –	– nonexistent –
guvision_clusterVector11	– none –	– nonexistent –
guvision_clusterVector12	bool	false
guvision_clusterVector13	bool	– nonexistent –
guvision_clusterVector14	array	279, 159
guvision_clusterVector15	– none –	– nonexistent –
guvision_clusterVector16	– none –	– nonexistent –
guvision_clusterVector17	– none –	– nonexistent –
guvision_clusterVector18	– none –	– nonexistent –
guvision_clusterVector19	– none –	– nonexistent –
guvision_clusterVector2	– none –	– nonexistent –
guvision_clusterVector20	– none –	– nonexistent –
guvision_clusterVector21	bool	false
guvision_clusterVector22	bool	false
guvision_clusterVector23	array	455, 383
guvision_clusterVector24	bool	false
guvision_clusterVector25	bool	– nonexistent –
guvision_clusterVector26	bool	– nonexistent –
guvision_clusterVector27	bool	– nonexistent –
guvision_clusterVector28	bool	false
guvision_clusterVector3	– none –	– nonexistent –

Monitored Message	Type	Content
Say	string	Hello
Speech	string	Up
guvision_clusterVector29	array	239, 199
guvision_ballInfo	string	19,-97,56,640,480
guvision_belt_pink_team_Info	string	44,-3,-117,640,480
guvision_belt_blue_team_Info	string	11,-42,-110,640,480

Fig. 9. Monitoring of variables with distributed whiteboard and monitoring behaviour in the robot (remote monitoring of vision analysis inside a Nao robot that is looking at other robots (ball and team colour analysis)

5 Conclusions

We have shown that simulation is a highly desirable tool to complement formal verification for system behaviour validation. We believe that earlier work on verification of correctness of models for the case studies discussed here has been incomplete because a faithful simulator has not been used. As a result, some properties are verified formally, but some emergent behaviour passes undetected (possibly until deployment).

References

1. Agrawal, A., Simon, G., Karsai, G.: Semantic translation of simulink/stateflow models to hybrid automata using graph transformations. Electr. Notes Theor. Comput. Sci. 109, 43–56 (2004)
2. Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press (2008)
3. Billington, D., Estivill-Castro, V., Hexel, R., Rock, A.: Architecture for Hybrid Robotic Behavior. In: Corchado, E., Wu, X., Oja, E., Herrero, Á., Baruque, B. (eds.) HAIS 2009. LNCS, vol. 5572, pp. 145–156. Springer, Heidelberg (2009)
4. Billington, D., Estivill-Castro, V., Hexel, R., Rock, A.: Non-monotonic reasoning for requirements engineering. In: Proc. 5th Int. Conf. on Evaluation of Novel Approaches to Software Engineering (ENASE), Athens, pp. 68–77. SciTePress (2010)
5. Billington, D., Estivill-Castro, V., Hexel, R., Rock, A.: Modelling Behaviour Requirements for Automatic Interpretation, Simulation and Deployment. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 204–216. Springer, Heidelberg (2010)
6. Burns, A., Lister, A.M.: A framework for building dependable systems. The Computer Journal 34(2), 173–181 (1991)
7. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT Press (2001)
8. Dromey, R.G., Powell, D.: Early requirements defect detection. TickIT Journal 4Q05, 3–13 (2005)
9. Estivill-Castro, V., Hexel, R., Rosenblueth, D.A.: Efficient model checking and fmea analysis with deterministic scheduling of transition-labeled finite-state machines. In: 3rd World Congress Software Engineering, China (to appear, 2012)
10. Grunske, L., Winter, K., Yatapanage, N., Zafar, S., Lindsay, P.A.: Experience with fault injection experiments for FMEA. Software, Practice and Experience 41(11), 1233–1258 (2011)
11. Harel, D., Politi, M.: Modeling Reactive Systems with Statecharts: The STATEMATE Approach. McGraw-Hill (1998)
12. Mahmood, T., Kazmierczak, E.: A knowledge-based approach for safety analysis using system interactions. In: 13th Asia Pacific Software Engineering Conf., APSEC 2006, pp. 445–452 (2006)
13. McDermid, J., Kelly, K.: Industrial press: Safety case. Technical report, High Integrity Systems Engineering Group, University of York (1996)
14. Mellor, S.J.: Embedded systems in UML. OMG White paper (2007) label: We can generate Systems Today, www.omg.org/news/whitepapers/
15. Mellor, S.J., Balcer, M.: Executable UML: A foundation for model-driven architecture. Addison-Wesley Publishing Co., Reading (2002)
16. Myers, T., Dromey, R.G.: From requirements to embedded software - formalising the key steps. In: 20th Australian Software Engineering Conf. (ASWEC), Gold Cost, Australia, pp. 23–33. IEEE Computer Society (2009)
17. OMG. OMG systems modeling language (OMG SysML^{TM}). Version 1.3 with change bars (June 2012)
18. Rumbaugh, J., Blaha, M.R., Lorensen, W., Eddy, F., Premerlani, W.: Object-Oriented Modelling and Design. Prentice-Hall, Inc., Englewood Cliffs (1991)
19. Samek, M.: Practical UML Statecharts in C/C++, 2nd edn: Event-Driven Programming for Embedded Systems, Newnes (2008)
20. Shlaer, S., Mellor, S.J.: Object lifecycles: modeling the world in states. Yourdon Press, Englewood Cliffs (1992)

21. Shrivastava, S.K., Mancini, L.V., Randell, B.: The duality of fault-tolerant system structures. Software — Practice and Experience 23(7), 773–798 (1993)
22. Sloman, M., Kramer, J.: Distributed systems and computer networks. Prentice Hall, UK (1987)
23. Sommerville, I.: Software engineering, 9th edn. Addison-Wesley, Boston (2010)
24. Wagner, F., Schmuki, R., Wagner, T., Wolstenholme, P.: Modeling Software with Finite State Machines: A Practical Approach. CRC Press, NY (2006)
25. Wen, L., Dromey, R.G.: From requirements change to design change: A formal path. In: 2nd Int. Conf. on Software Engineering and Formal Methods (SEFM 2004), pp. 104–113. IEEE Computer Society, Beijing (2004)
26. Winter, K., Yatapanage, N.: The metal press case study. Technical report, University of Queensland. Supplement in www.itee.uq.edu.au/~docs/FMEA
27. Winter, K., Yatapanage, N.: The mine pump case study. Technical report, University of Queensland. Supplement in www.itee.uq.edu.au/~docs/FMEA

Fast Dynamic Simulation of Highly Articulated Robots with Contact via $\Theta(n^2)$ Time Dense Generalized Inertia Matrix Inversion

Evan Drumwright

The George Washington University, Washington, D.C. USA
drum@gwu.edu

Abstract. The generalized inertia matrix and its inverse are used extensively in robotics applications. While construction of the inertia matrix requires $\Theta(n^2)$ time, inverting it traditionally employs algorithms running in time $O(n^3)$. We describe an algorithm that reduces the asymptotic time complexity of this operation to the theoretical minimum: $\Theta(n^2)$. We also present simple modifications that reduce the number of arithmetic operations (and thereby the running time). We compare our approach against fast Cholesky factorization both theoretically (using number of arithmetic operations) and empirically (using running times). We demonstrate our method to dynamically simulate a highly articulated robot undergoing contact, yielding an order of magnitude decrease in running time over existing methods.

1 Introduction

The inverse of the generalized inertia matrix is used in numerous robotics applications such as computing the *contact space inertia matrix* and the *kinetic energy matrix* [1]. When mechanism dynamics are computed in absolute coordinates (*i.e.*, mass matrices are of size $6m \times 6m$, where m is the number of links in the mechanism), the inertia matrix is banded and can be trivially inverted in time $\Theta(m)$ (though constraint forces must be computed, typically requiring operations exhibiting cubic time complexity). For n degree of freedom mechanisms without kinematic loops, the $n \times n$ generalized inertia matrix (formulated in independent coordinates) is dense, symmetric, and positive definite (PD). This matrix can be constructed in $\Theta(n^2)$ time using the *Composite Rigid Body Algorithm* [2,3] and is traditionally inverted using a combination of $O(n^3)$ Cholesky factorization and $\Theta(n^2)$ backsubstitution.

The key algorithms that make possible the results in this paper (the *Recursive Newton-Euler Algorithm* [4] and the *Articulated Body Algorithm* [5]) were developed over three decades ago; however, the community of robotics researchers is generally unaware of the straightforward implication of $\Theta(n^2)$ generalized inertia matrix inversion. This paper presents a modified version of the latter algorithm optimized toward that new purpose (efficient $\Theta(n^2)$ inversion of $n \times n$ generalized inertia matrices). We compare both operation counts and running times against existing methods for performing the factorization plus backsubstitution.

Before proceeding, observe that we use the concepts of matrix inversion and solving linear systems of equations interchangeably unless otherwise noted to simplify presentation. In the context of our presented algorithms, asymptotic time complexity for matrix inversion is $\Theta(n^2)$ and complexity for solving a system of linear equations with m right hand sides is $\Theta(mn)$.

2 Background

2.1 State of the Art in Robot Dynamics Computation

Robot dynamics equations are usually given in the form:

$$\mathbf{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\dot{\boldsymbol{q}}, \boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}) = \boldsymbol{\tau} \tag{1}$$

where $\mathbf{M}(\boldsymbol{q})$ is the generalized inertia matrix, $\boldsymbol{C}(\dot{\boldsymbol{q}}, \boldsymbol{q})\dot{\boldsymbol{q}}$ is the combined vector of Coriolis and centrifugal forces acting on the robot, $\boldsymbol{G}(\boldsymbol{q})$ is the vector of gravity forces acting on the robot, and $\boldsymbol{\tau}$ is the vector of actuator forces. When the robot base is "floating" (not affixed to the environment), the equation maintains the same structure, but additional terms are added both for the base acceleration and external forces applied to the base.

State of the art algorithms for computing the joint accelerations as a function of actuator forces are compiled in works by Featherstone [3,6] and Featherstone and Orin [7]. Featherstone groups these algorithms into two categories: $O(n^3)$ algorithms that directly solve the system of linear equations described by Equation 1 directly and an $\Theta(n)$ algorithm (the *Articulated Body Method*) that treats each link as a rigid body with a "handle". This latter algorithm is the key to achieving the $\Theta(n^2)$ inversion operation.

The former category is epitomized by the *Composite Rigid Body Algorithm*, which was studied extensively by Walker and Orin [2]. They claim $\Theta(n^2)$ running time for one of their algorithms (method 4); however, their analysis describes dense matrix-vector multiplication as an $O(n)$ operation (standard algorithms exhibit $O(n^2)$ complexity), and $O(n^3)$ is the true asymptotic behavior.[1] Featherstone [8] has reduced this $O(n^3)$ complexity to $O(n^2d)$, where d is the maximum depth of the kinematic "tree", by exploiting *branch induced sparsity*: mechanisms with branches (multiple child links emanating from a parent) induce sparsity in the generalized inertia matrix. Mechanisms composed of long chains will yield running times closer to $O(n^3)$ than $O(n^2)$, and it is even likely that Featherstone's specialized Choesky factorization and LDL^{T} algorithms [8] for exploiting branch induced sparsity are slower than specialized dense libraries like LAPACK for most cases (experiments described in Section 6 give credence to this hypothesis).

Finally, we note that Baraff also provides a linear time dynamics algorithm [9] that could be employed toward our purpose instead of Featherstone's algorithm. However, the constant factor for Baraff's algorithm (in absolute coordinates) is dependent upon $6n - r$, where n is the number of joints in the system and r is the

[1] Walker and Orin's algorithm is a minor adaptation of the iterative conjugate gradient method for solving symmetric PD systems and is known to exhibit $O(n^3)$ complexity.

number of joint degrees-of-freedom, while the constant factor for Featherstone's algorithm (in independent coordinates) is dependent only upon r; thus, for typical applications with single degree-of-freedom robot joints, Featherstone's algorithm should be significantly faster.

3 Overview of Spatial Algebra

This section presents an overview of Spatial Algebra, which permits dynamics algorithms to be described clearly and succinctly. Extensive tutorials of this subject are contained in [3,6]; this paper employs the system described in [3].

Spatial vectors are composed of two stacked three-dimensional vectors (a "line vector" and a "free vector"). For example, the spatial velocity of a rigid body is represented by the vector $\hat{\boldsymbol{v}} = \begin{bmatrix}\boldsymbol{\omega}\ \dot{\boldsymbol{x}}\end{bmatrix}^{\mathsf{T}}$. All Spatial Algebra operations can be performed using standard matrix and vector arithmetic except the spatial transpose operation. The spatial transpose operation is denoted using the superscript $^{\mathsf{S}}$ and yields $\hat{\boldsymbol{v}}^{\mathsf{S}} = \begin{bmatrix}\dot{\boldsymbol{x}}^{\mathsf{T}}\ \boldsymbol{\omega}^{\mathsf{T}}\end{bmatrix}$ when applied to the vector above.

3.1 Spatial Transformations

Spatial transformations take the form:

$$ {}_{j}\hat{\mathbf{X}}_{i} = \begin{bmatrix} \mathbf{E} & \mathbf{0} \\ -\tilde{\boldsymbol{r}}\mathbf{E} & \mathbf{E} \end{bmatrix} \tag{2}$$

where ${}_{j}\hat{\mathbf{X}}_{i}$ is the spatial transform from frame i (defined by rotation matrix $\mathbf{R}_i$ and offset $\boldsymbol{x}_i$) to frame j (defined by rotation matrix $\mathbf{R}_j$ and offset $\boldsymbol{x}_j$) and $\mathbf{E} = \mathbf{R}_j^{\mathsf{T}}\mathbf{R}_i$ and $\boldsymbol{r} = \mathbf{R}_j^{\mathsf{T}}(\boldsymbol{x}_j - \boldsymbol{x}_i)$. The skew-symmetric operator $\tilde{}$ is defined on vector $\boldsymbol{r} = \begin{bmatrix}r_x\ r_y\ r_z\end{bmatrix}^{\mathsf{T}}$ below:

$$\tilde{\boldsymbol{r}} = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}.$$

A spatial vector $\hat{\boldsymbol{v}}_i$ can be transformed from frame i to frame j by:

$$\hat{\boldsymbol{v}}_j = {}_{j}\hat{\mathbf{X}}_{i}\ \hat{\boldsymbol{v}}_i. \tag{3}$$

while a spatial inertia matrix (defined below) $\hat{\mathbf{I}}_i$ can be transformed from frame i to frame j via two matrix-matrix multiplications:

$$\hat{\mathbf{I}}_j = {}_{j}\hat{\mathbf{X}}_{i}\ \hat{\mathbf{I}}_i\ {}_{i}\hat{\mathbf{X}}_{j}. \tag{4}$$

3.2 Spatial Rigid Body and Articulated Body Inertias

The spatial inertia of a rigid body (in its local frame) is:

$$\hat{\mathbf{I}}' = \begin{bmatrix} \mathbf{0} & m\mathbf{1} \\ \mathbf{J} & \mathbf{0} \end{bmatrix} \tag{5}$$

where the mass of the rigid body is m ($\mathbf{1}$ is the identity matrix) and its 3×3 moment of inertia matrix is denoted $\mathbf{J}$. The spatial rigid body inertia transformed into the "global" reference frame (*i.e.*, via the spatial transformation in Section 3.1) is denoted $\hat{\mathbf{I}}$ (rather than $\hat{\mathbf{I}}'$). We call the sum of the spatial rigid body inertia of link i and the composite inertias of all of link i's children (successors in the kinematic chain) $\hat{\mathbf{I}}_{c_i}$, the *composite inertia*:

$$\hat{\mathbf{I}}_{c_i} = \hat{\mathbf{I}}_i + \sum_{j \in \text{children}(i)} \hat{\mathbf{I}}_{c_j}. \tag{6}$$

Featherstone's Articulated Body Algorithm uses a special inertia matrix, $\hat{\mathbf{I}}^A$—known as the *spatial articulated body inertia*—and defined below:

$$\hat{\mathbf{I}}_i^A = \hat{\mathbf{I}}_i + \sum_{j \in \text{children}(i)} \left[\hat{\mathbf{I}}_j^A - \frac{\hat{\mathbf{I}}_j^A \hat{\boldsymbol{s}}_j \hat{\boldsymbol{s}}_j^{\mathsf{S}} \hat{\mathbf{I}}_j^A}{\hat{\boldsymbol{s}}_j^{\mathsf{S}} \hat{\mathbf{I}}_j^A \hat{\boldsymbol{s}}_j}\right] \tag{7}$$

3.3 Spatial Axes

The spatial axis for a link transforms its inner joint velocity to the change in spatial velocity of that link. Thus, a collection of spatial axes are analogous to the Jacobian that transforms joint velocities to end effector velocity. Spatial axes for common single degree-of-freedom joints—for simplify of presentation and without loss of generality, only single degree-of-freedom joints are considered in this paper—are given below:

$$\hat{\boldsymbol{s}}_i' = \begin{cases} \begin{bmatrix} \boldsymbol{u}_i \\ \mathbf{0} \end{bmatrix} & \text{if } i \text{ revolute,} \\ \\ \begin{bmatrix} \mathbf{0} \\ \boldsymbol{u}_i \end{bmatrix} & \text{if } i \text{ prismatic.} \end{cases} \tag{8}$$

where $\boldsymbol{u}_i$ is the unit three-dimensional vector pointing along the joint axis and $\mathbf{0}$ is the three-dimensional zero vector. Note that the spatial axes above are computed in a frame with origin at the joint's Cartesian position (denoted by the "prime" applied to $\hat{\boldsymbol{s}}_i$).

4 Notation and Conventions

We adopt the following conventions/notation from [3,6]:

- $\hat{\boldsymbol{u}}$ indicates that $\boldsymbol{u}$ is a $6 \times m$ spatial vector or matrix ($m \leq 6$)
- $\hat{\boldsymbol{r}}^S$ indicates the spatial transpose operation is applied to $\hat{\boldsymbol{r}}$
- ${}_j\hat{\mathbf{X}}_k$ denotes the spatial transformation from frame k to frame j
- $\hat{\boldsymbol{J}}^A$ indicates that $\hat{\boldsymbol{J}}$ is an *articulated body* vector or matrix (*i.e.*, it is used in the context of Featherstone's Articulated Body Algorithm)
- $\lambda : \mathbb{N} \to \mathbb{N}$ maps the index of a link to the index of that link's parent

The joints and links of an n joint, n link (excluding the base link) mechanism are indexed in the following manner: (1) the base link (fixed or "floating") assumes index 0; (2) the inner joint for the link with index i assumes joint index i as well; (3) every link and joint assumes a unique index; and (4) no "ancestor" to a link can possess a link index greater than its descendant.

5 $\Theta(n^2)$ Inverse Inertia Matrix Computation

Our simplified version of Featherstone's Articulated Body Algorithm is based on the classical mechanics equation relating change in linear momentum to applied impulses (abstracted to generalized coordinates):

$$\mathbf{M}\Delta\boldsymbol{v} = \boldsymbol{j} \tag{9}$$

where $\mathbf{M}$ is a $n \times n$ generalized inertia matrix, $\boldsymbol{v}$ is an n-dimensional vector of generalized velocities, and $\boldsymbol{j}$ is an n-dimensional vector of generalized impulses. We wish to find the inverse of $\mathbf{M}$:

$$\Delta\boldsymbol{v} = \mathbf{M}^{-1}\boldsymbol{j}. \tag{10}$$

This equation indicates that $\mathbf{M}^{-1}$ is equal to the changes in velocity due to applying n unit-vector impulses (represented by the $n \times n$ identity matrix) to the system. Using impulses and velocity changes in place of forces and accelerations allows us to avoid determination of gravity, centrifugal, and Coriolis forces inherent in forward dynamics computation. Algorithm 1 implements this strategy with focus on practical (*i.e.*, numerically stable) implementation: the inverse is not constructed explicitly.

Algorithm 1. mMultInv(.) multiplies the inverse of the generalized inertia matrix for a fixed-base mechanism with n joints by an $n \times m$ matrix ($\mathbf{R}$) in $\Theta(nm)$ time

```
Require: articulated body inertia matrices in global frame ($\hat{\mathbf{I}}^A$), spatial axes in global
    frame ($\hat{\boldsymbol{s}}$)
 1: for $k = 1 \dots m$ do
 2:     for $i \leftarrow 1 \dots n$ do {Initialize articulated body impulses}
 3:         $\hat{\mathbf{Y}}_i^A \leftarrow \hat{\mathbf{0}}$
 4:     for $i = n \dots 1$ do {Propagate impulses}
 5:         $\hat{\mathbf{Y}}_{\lambda(i)}^A \leftarrow \hat{\mathbf{Y}}_{\lambda(i)}^A + \left[\frac{\hat{\mathbf{I}}_i^A \hat{\boldsymbol{s}}_i (R_{ik} - \hat{\boldsymbol{s}}_i^S \hat{\mathbf{Y}}_i^A)}{\hat{\boldsymbol{s}}_i^S \hat{\mathbf{I}}_i^A \hat{\boldsymbol{s}}_i}\right]$
 6:     $\Delta\hat{\boldsymbol{v}}_0 \leftarrow \hat{\mathbf{0}}$
 7:     for $i = 1 \dots n$ do {Compute velocity updates}
 8:         $\Delta\dot{q}_i \leftarrow \frac{\delta_{ik} - \hat{\boldsymbol{s}}_i^S \left[\hat{\mathbf{I}}_i^A \Delta\hat{\boldsymbol{v}}_{\lambda(i)} + \hat{\mathbf{Y}}_i^A\right]}{\hat{\boldsymbol{s}}^S \hat{\mathbf{I}}_i^A \hat{\boldsymbol{s}}_i}$
 9:         $\Delta\hat{\boldsymbol{v}}_i \leftarrow \Delta\hat{\boldsymbol{v}}_{\lambda(i)} + \hat{\boldsymbol{s}}_i \Delta\dot{q}_i$
10:         $M_{ik} \leftarrow \Delta\dot{q}_i$
```

All calculations are computed in the global frame rather than using link frames: the former is more efficient for our approach due to the $\Theta(nm)$ spatial transformations between link frames that would otherwise be required (Featherstone [7] shows that computation is more efficient in the link frame than the global frame for the unmodified Articulated Body Algorithm, in general). We also present Algorithm 2, which contains necessary modifications to handle mechanisms with floating bases.

Algorithm 2. mMultInvFB(.) multiplies the inverse of the generalized inertia matrix for a floating-base mechanism with n joints by an $n \times m$ matrix ($\mathbf{R}$) in $\Theta(nm)$ time

```
Require: articulated body inertia matrices in global frame ($\hat{\mathbf{I}}^A$), spatial axes in global
    frame ($\hat{\boldsymbol{s}}$)
 1: for $k = 1 \ldots m$ do
 2:     for $i \leftarrow 0 \ldots n$ do {Initialize articulated body impulses}
 3:         $\hat{\mathbf{Y}}_i^A \leftarrow \hat{\mathbf{0}}$
 4:     for $i = n \ldots 1$ do {Propagate impulses}
 5:         $\hat{\mathbf{Y}}_{\lambda(i)}^A \leftarrow \hat{\mathbf{Y}}_{\lambda(i)}^A + \left[\frac{\hat{\mathbf{I}}_i^A \hat{\boldsymbol{s}}_i (R_{ik} - \hat{\boldsymbol{s}}_i^S \hat{\mathbf{Y}}_i^A)}{\hat{\boldsymbol{s}}_i^S \hat{\mathbf{I}}_i^A \hat{\boldsymbol{s}}_i}\right]$
 6:     if $k \leq 6$ then {Compute floating base velocity change}
 7:         $\Delta\hat{\boldsymbol{v}}_0 \leftarrow -\hat{\mathbf{I}}_0^{A^{-1}}(\hat{\mathbf{Y}}_0^A + \boldsymbol{I}_k)$ {$\boldsymbol{I}_k$ is the $k^{\text{th}}$ column of the $6 \times 6$ identity matrix}
 8:     else
 9:         $\Delta\hat{\boldsymbol{v}}_0 \leftarrow -\hat{\mathbf{I}}_0^{A^{-1}}(\hat{\mathbf{Y}}_0^A)$
10:     $\boldsymbol{u}_k \leftarrow \Delta\hat{\boldsymbol{v}}_0$
11:     for $i = 1 \ldots n$ do {Compute velocity updates}
12:         $\Delta\dot{q}_i \leftarrow \frac{\delta_{ik} - \hat{\boldsymbol{s}}_i^S \left[\hat{\mathbf{I}}_i^A \Delta\hat{\boldsymbol{v}}_{\lambda(i)} + \hat{\mathbf{Y}}_i^A\right]}{\hat{\boldsymbol{s}}^S \hat{\mathbf{I}}_i^A \hat{\boldsymbol{s}}_i}$
13:         $\Delta\hat{\boldsymbol{v}}_i \leftarrow \Delta\hat{\boldsymbol{v}}_{\lambda(i)} + \hat{\boldsymbol{s}}_i \Delta\dot{q}_i$
14:         $A_{ik} \leftarrow \Delta\dot{q}_i$
15: return $\begin{bmatrix} \boldsymbol{u}_1 \ldots \boldsymbol{u}_n \\ \mathbf{A} \end{bmatrix}$
```

5.1 Complexity Analysis

The operations on lines 5, 8, and 9 of Algorithm 1 each require constant time ($\hat{\boldsymbol{Y}}^A$, $\hat{\mathbf{I}}^A$, $\hat{\boldsymbol{s}}$, and $\Delta\hat{\boldsymbol{v}}$ are of fixed size), so the nested **for** loops result in $\Theta(nm)$ time complexity.

5.2 Arithmetic Analysis

We can utilize a few optimizations not shown in Algorithm 1 to decrease the number of arithmetic operations. Vectors and quantities $\hat{\mathbf{I}}^A\hat{\boldsymbol{s}}$ and $\hat{\boldsymbol{s}}^{\mathrm{S}}\hat{\mathbf{I}}^A\hat{\boldsymbol{s}}$ are computed only once, rather than separately for each iteration ($\hat{\boldsymbol{s}}$ and $\hat{\mathbf{I}}^A$ are dependent upon only the coordinates of the mechanism and not its velocity). Additionally, the impulse propagation process (Lines 4–5 of Algorithm 1) needs

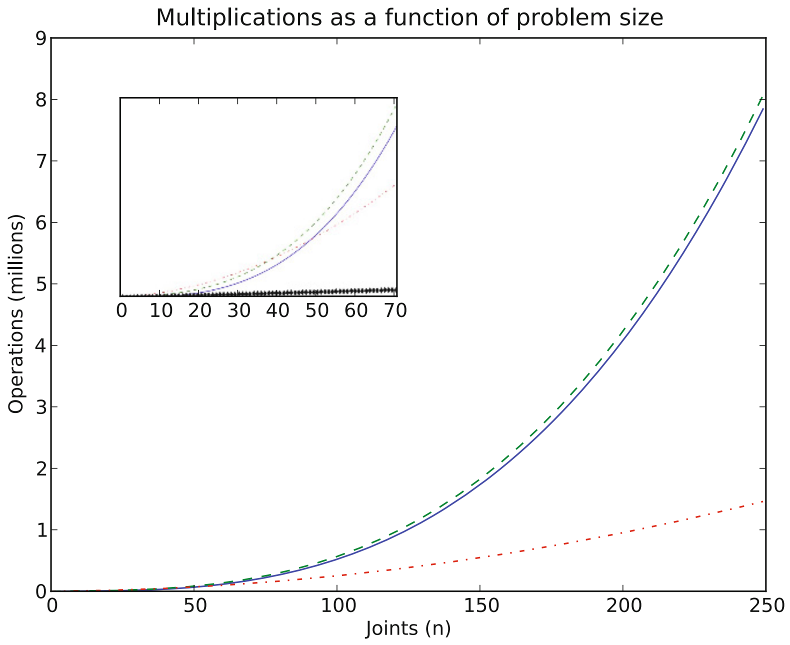


Fig. 1. Number of multiplication operations for generalized inertia inversion/factorization as a function of multibody joints for four methods: Cholesky factorization only (blue/solid), both formation and Cholesky factorization (green/dashed), the $\Theta(n^2)$ method presented in this paper (red/dash-dot), and the $\Theta(n^2)$ parallel method using n processors described in Section 5.2 (black/diamonds). Numbers of arithmetic operations are computed as described in Section 5.2.

Table 1. Arithmetic operation counts for $\Theta(n^2)$ inversion algorithm (Algorithm 1)

Operation	Multiplications	Additions
Computing spatial axes in global frame ($\hat{\boldsymbol{s}}$)	$24n$	$6n$
Computing isolated spatial inertias ($\hat{\mathbf{I}}$)	$84n$	$57n$
Computing articulated spatial inertias ($\hat{\mathbf{I}}^A$)	$108n$	$107n$
Impulse propagation ($\hat{\boldsymbol{Y}}$)	$13n^2$	$13n^2$
Link and joint velocity updates ($\Delta\hat{\boldsymbol{v}}, \Delta\dot{q}$)	$19n^2/2$	$26n^2/2$
Total	$45n^2/2 + 216n$	$26n^2 + 170n$

to start only at the single joint at which δ_{ik} is nonzero. Table 1 shows the multiplication and addition counts for the inversion algorithm.

One exciting potential of this work is its utilization in massively parallel computation. Although computing articulated spatial inertias must be done sequentially, the spatial axes ($\hat{\boldsymbol{s}}$) and the isolated spatial inertias ($\hat{\mathbf{I}}$) can be computed in parallel. Each column of $\mathbf{R}$ can be solved (or, equivalently, each column of $\mathbf{M}^{-1}$ can be determined) in parallel as well. Thus, if one employs an n multiprocessor

Table 2. Arithmetic operation counts for Composite Rigid Body Algorithm (n joint, fixed base mechanism, global frame)

Operation	Multiplications	Additions
Computing spatial axes in global frame ($\hat{\boldsymbol{s}}$)	$24n$	$6n$
Computing isolated spatial inertias ($\hat{\mathbf{I}}$)	$84n$	$57n$
Computing composite inertia matrices ($\mathbf{I}_c$)	0	$13n - 13$
Computing $\hat{\mathbf{I}}\hat{\boldsymbol{s}}$ vectors	$36n$	$24n$
Computing $\hat{\boldsymbol{s}}_i^{\mathrm{S}}\hat{\mathbf{I}}_j\hat{\boldsymbol{s}}_j$ (elements of $\mathbf{M}$)	$3n^2$	$5n^2/2$
Cholesky factorization (naive)	$n^3/2 + n^2/2 + n$	$n^3/2$
Total	$n^3/2 + 7n^2/2 + 145n$	$n^3/2 + 5n^2/2 + 100n - 13$

Table 3. Arithmetic operation counts per processor for $\Theta(n^2)$ inversion algorithm (n joint, fixed base mechanism) on n processors

Operation	Multiplications	Additions
Computing spatial axes in global frame ($\hat{\boldsymbol{s}}$)	24	6
Computing isolated spatial inertias ($\hat{\mathbf{I}}$)	84	57
Computing articulated spatial inertias ($\hat{\mathbf{I}}^A$)	$108n$	$107n$
Impulse propagation ($\hat{\boldsymbol{Y}}$)	$13n$	$13n$
Link and joint velocity updates ($\Delta\hat{\boldsymbol{v}}, \Delta\dot{q}$)	$19n/2$	$13n$
Total	$261n/2 + 108$	$133n + 63$

system for solving, only $261n/2 + 108$ multiplications and $133n + 63$ additions are required (specific operation counts are given in Table 3 for an n-processor system). Cholesky factorization benefits little from massively parallel or SIMD computation.

6 Experiments

Numerical experiments were conducted with the *Moby* robot dynamics library. Vector arithmetic utilized tuned BLAS libraries (ATLAS). Cholesky factorization was performed using LAPACK, our implementation of Featherstone's branch induced sparsity (BIS) factorization [8], or both. Experiments were conducted on a dual core 2.8GHz Intel Xeon W3530 processor (four virtual cores using HyperThreading$^{\mathrm{TM}}$) running Ubuntu Linux. for Cholesky factorization as well as our implementation of Featherstone's branch induced sparsity (BIS) Cholesky factorization.

6.1 Single-Threaded Inversion Experiments

The experiment described in this section uses a single-threaded version of the *Moby* library. The articulated bodies used in these experiments are fully serial (*i.e.*, the kinematic tree is of depth n) to obtain the most conservative timings for our method. However, we also wished to compare our solving algorithm against

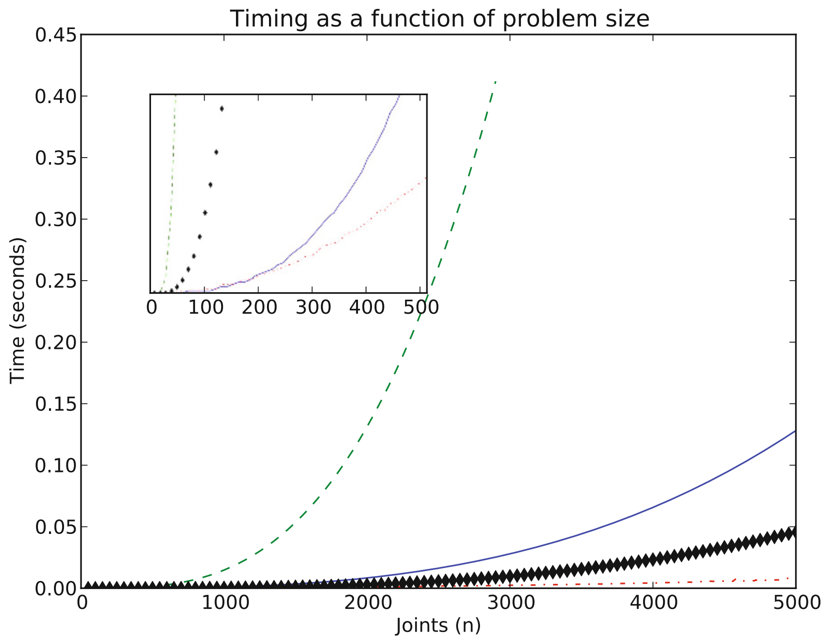


Fig. 2. Times required to compute the inverse of the generalized inertia matrix using the presented $\Theta(n^2)$ method (red/dash-dot) and the construction and Cholesky factorization of the generalized inertia matrix using (1) LAPACK (blue/solid) and the branch induced sparsity method for (2) a serial body of depth n (green/dashed) and for (3) a branched body of expected depth $n/2$ (black diamond).

BIS Cholesky factorization. As indicated by Figure 2, we tested branched bodies of expected depth $n/2$ *only* for the BIS method: with uniform probability 0.5, we added a link as a sibling to another rather than adding that link to the end of the chain (no link was permitted more than two children).

6.2 Multi-threaded Inversion Experiment

This experiment used OpenMP and a multi-threaded version of *Moby* to compute the inverse of the generalized inertia matrix; threads were limited to four (the Intel Xeon presents four virtual cores via HyperThreading$^{\text{TM}}$). Unlike the previous experiment, which timed only CPU operations, this experiment timed all operations (including I/O and time waiting for the OS's scheduler) in order to assess efficiency gains via parallelism.

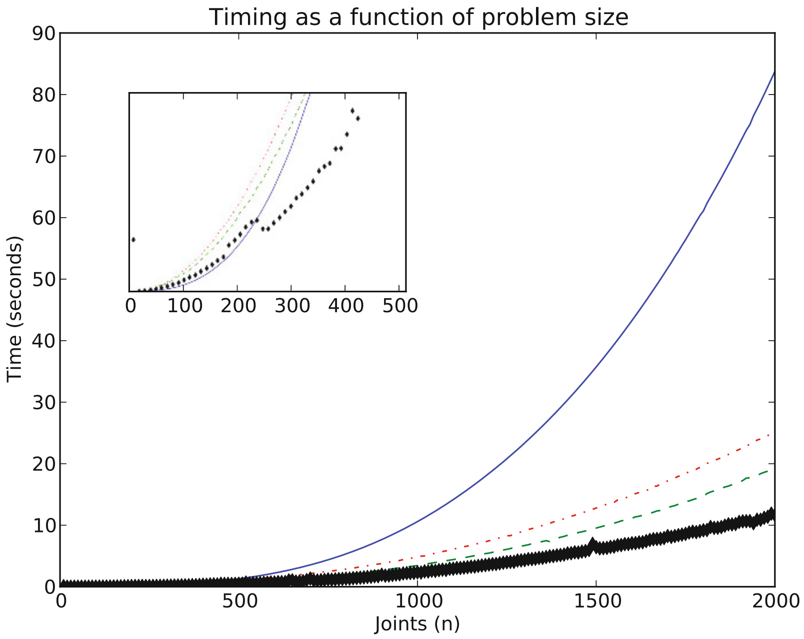


Fig. 3. Times required to compute the inverse of the generalized inertia matrix using LAPACK's Cholesky factorization (blue/solid) and the presented $\Theta(n^2)$ method with (1) one thread (red/dash-dot), (2) two threads (green/dashed), and (3) four threads (black/diamond).

6.3 Simulation Experiments

We tested the effectiveness of our method on a "real world" problem: dynamic simulation of a centipede walking on a planar surface. The centipede was simulated using increasing numbers of body segments (the maximum number of body segments was 250). Each body segment was connected to two upper legs via spherical joints; each upper leg was connected to a lower leg via a revolute joint. The simulation used explicit Euler integration with a step size of $1e^{-5}$ for one thousand steps. The software setup used in the previous experiment was employed in this experiment as well. Timings were conducted over all operations: dynamics computation, collision detection, contact resolution, *etc.*; however, only CPU timings were used (as in the first experiment). Contact resolution used the method described in [10], which requires computing the contact space inertia matrices $\mathbf{N}^\mathsf{T}\mathbf{M}^{-1}\mathbf{N}$, $\mathbf{N}^\mathsf{T}\mathbf{M}^{-1}\mathbf{D}$, and $\mathbf{D}^\mathsf{T}\mathbf{M}^{-1}\mathbf{D}$ ($\mathbf{N}$ and $\mathbf{D}$ are contact Jacobians for the normal and tangent directions, respectively); this was the only code that required the inverse inertia matrix and is solely responsible for the timing differences in Figure 4.

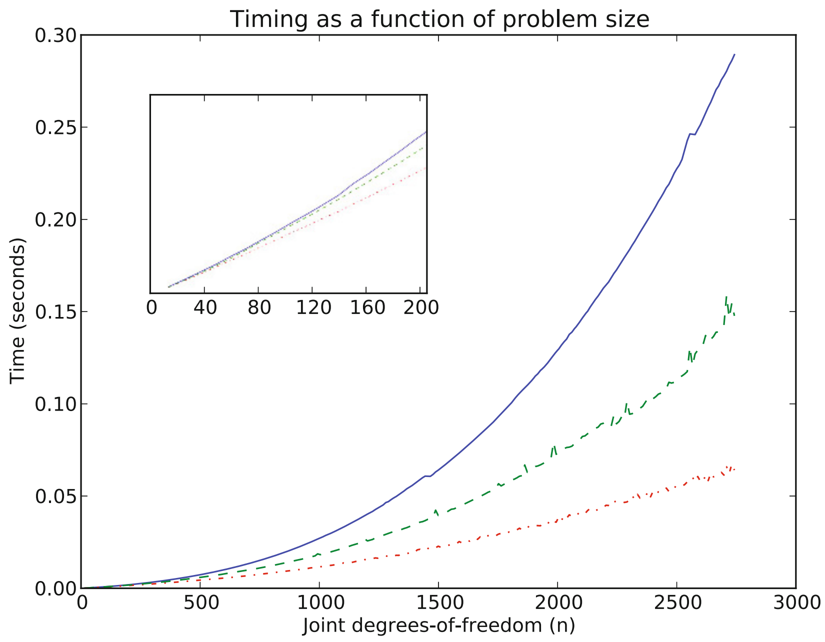


Fig. 4. Timings for dynamically simulating centipedes with varying numbers of legs walking on a planar surface. The contact space inertia matrix—$\mathbf{N}^\mathsf{T}\mathbf{M}^{-1}\mathbf{N}$—is computed using the presented $\Theta(n^2)$ method (red/dash-dot), Cholesky factorization and backsubstitution using LAPACK (blue/solid), and Cholesky factorization—using branch induced sparsity—plus backsubstitution (green/dashed).

7 Discussion

Our $\Theta(n^2)$ inversion method is amenable to both symbolic computation (which could significantly reduce the number of arithmetic computations) and parallelization (especially in the context of SIMD/GPU processing).

Figures 2, 3, and 4 from the experiments in the previous section show that our algorithm is competitive with BLAS/LAPACK even for bodies with relatively few degrees-of-freedom (fewer than 100); for bodies with greater degrees-of-freedom, the asymptotic advantage of our approach becomes evident quickly. The experiment using multi-threading in Section 6.2 illustrates the potential gains in performance from multiprocessing: using two threads increases performance over one thread by 34%, and using four threads increases performance by 58% and 117%, respectively, over two threads and one thread. Larger scale SIMD parallelism (via GPU processing, for example) should yield further large increases in performance.

The experiments in the previous section illustrate the power of tuned BLAS and LAPACK libraries for vector arithmetic and linear algebra: Cholesky factorization yielded superior performance for $n < 130$ in the first experiment (which used tuned libraries) and inferior performance in the third experiment. The second experiment illustrates the potential gains in performance from multiprocessing: using two threads increases performance over one thread by 34%, and using four threads increases performance by 58% and 117%, respectively, over two threads and one thread.

All experiments clearly show that for applications that require inverting the generalized inertia matrix of highly articulated robots, our method yields significant performance increases. Further optimizations should yield additional increases.

References

1. Khatib, O.: A unified approach to motion and force control of robot manipulators: The operational space formulation. IEEE Journal on Robotics and Automation 3(1), 43–53 (1987)
2. Walker, M.W., Orin, D.E.: Efficient dynamic computer simulation of robotic mechanisms. ASME J. Dynamic Systems, Measurement, and Control 104, 205–211 (1982)
3. Featherstone, R.: Robot Dynamics Algorithms. Kluwer (1987)
4. Hollerbach, J.M.: A recursive lagrangian formulation of manipulator dynamics and a comparative study of dynamics formulation complexity. IEEE Trans. Systems, Man, and Cybernetics SMC-10(11), 730–736 (1980)
5. Featherstone, R.: The calculation of robot dynamics using articulated body inertias. Intl. J. Robotics Research 2(1), 13–30 (1983)
6. Featherstone, R.: Rigid Body Dynamics Algorithms. Springer (2008)
7. Featherstone, R., Orin, D.: Robot dynamics: Equations and algorithms. In: Proc. of IEEE Intl. Conf. on Robotics and Automation, San Francisco, CA (April 2000)
8. Featherstone, R.: Efficient factorization of the joint space inertia matrix for branched kinematic trees. Intl. J. Robotics Research 24(6), 487–500 (2005)
9. Baraff, D.: Linear-time dynamics using lagrange multipliers. In: Proc. of Computer Graphics, New Orleans, LA (August 1996)
10. Drumwright, E., Shell, D.A.: Modeling contact friction and joint friction in dynamic robotic simulation using the principle of maximum dissipation. In: Proc. of Workshop on the Algorithmic Foundations of Robotics, WAFR (2010)

A Differential-Algebraic Multistate Friction Model

Xiaogang Xiong*, Ryo Kikuuwe, and Motoji Yamamoto

Department of Mechanical Engineering
Kyushu University
Fukuoka, 819-0395, Japan
{xiong@ctrl.,kikuuwe@,yama@}mech.kyushu-u.ac.jp

Abstract. Fidelity with friction properties and easiness of implementation are both important aspects for friction modeling. Some empirically motivated models can be implemented easily due to their simple expression and small number of parameters, but they cannot capture faithfully the main properties of friction. Some physically motivated models give close agreement with the friction properties, but they can be too complex for some applications. This paper proposes a differential-algebraic multistate friction model that possesses easiness of implementation and adjustment, a relatively small number of parameters and a compact formulation. Moreover, it captures all standard properties of well-established friction models.

Keywords: Stick-slip behavior, Nondrifting property, Hysteresis with nonlocal memory, Frictional lag, Rate-independent.

1 Introduction

Friction is a complex nonlinear phenomenon that arises from the interaction among asperities in two contact surfaces moving relatively to each other. Friction modeling is important to predict various frictional phenomena and behaviors for simulation and control in a variety of areas ranging from mechanical engineering, mechatronics, robotics, to geophysic [1]. For instance, precise simulation of the bipedal locomotion of a waking robot requires a valid friction model to produce friction force between the foot and ground.

The elaboration of friction model is improved gradually from Coulomb friction model, Dahl model [2], LuGre model [3, 4], elastoplastic model [5], and Leuven model [6], to some generic models at asperity level [7, 8]. Those generic models, as pointed out in [9], can provide good approximation to experimental results, but it can be too complex for some applications. Two basic criteria are proposed in [1] to judge a friction model: (i) the fidelity in predicting realistic friction phenomena, and (ii) the easiness of implementation. In [9], another criteria is

* Kyushu University, Motooka 744, Nishi-ku, Fukuoka 819-0395, Japan Phone: (+81) 92-802-3179, Fax: (+81) 92-802-3177.

added: (iii) the easiness of adjusting the model such that the predicted results fit to the realistic results. The third one is suitable for online adjustment.

The Dahl model, LuGre model and elastoplastic model have simple expressions and they are easy to be implemented. However, the three models do not satisfy the first criteria. Dahl model cannot capture Stribeck effect. LuGre model captures that effect but it can cause positional drift and cannot produce hysteresis with nonlocal memory[1]. The elastoplastic model exhibits nondrifting property, but it still cannot produce hysteresis with nonlocal memory. Leuven model exhibits hysteresis with nonlocal memory while it is difficult for implementation. Modified Leuven model [11] improves the easiness of implementation, but the frictional lag, nondrifting property and transition behavior exhibited by this model are not so faithful to empirical results [1, 12]. The generalized Maxwell-slip (GMS) model [1, 12] has a good fidelity, but it depends on ambiguous distinction between the presliding and kinetic friction states. Moreover, the results of transition behavior and frictional lag produced by this model are not so easy to be adjusted by varying its parameters. The generic model [7, 8], which is motivated by the physics, are capable of giving close agreement to friction force dynamics, satisfying the first criteria very well, but it is too complex for implementation in control systems involving friction [9].

This paper proposes a new multistate friction model, which has a similar structure to those of the GMS model and others [13–15], specifically, being composed of parallel connection of single-state elastoplastic elements. In the new model, each elastoplastic element is modeled as a modified version of the authors' previous model based on differential-algebraic inclusions [16]. Comparing to the GMS model, the presented model has clear conditions to distinguish the presliding and kinetic friction states and more compact formulation. In addition, one parameter of the new model permits the easiness of adjusting the behaviors of friction, especially the frictional lag and transition behavior. Those make the new friction model to satisfy the aforementioned three criteria better.

2 Mathematical Preliminaries

For the discussion throughout this paper, this section defines three functions and describes their properties. In the rest of this paper, $\mathbb{R}$ denotes the set of all real numbers and 0 is the zero vector with an appropriate dimension.

Let us define the signum function $\mathrm{sgn} : \mathbb{R}^n \to \mathbb{R}^n$, extended signum function $\mathrm{Sgn} : \mathbb{R}^n \to \mathbb{R}^n$ and the saturation function $\mathrm{sat} : \mathbb{R}_+ \times \mathbb{R}^n \to \mathbb{R}^n$ as follows:

$$\mathrm{sgn}(x) \triangleq \begin{cases} x/\|x\| & \text{if } \|x\| \neq 0 \\ 0 & \text{if } \|x\| = 0 \end{cases} \tag{1}$$

[1] Hysteresis with nonlocal memory is defined as an input-output map, of which the output at any time instant is dependent on the output at some time instant in the past, the input since then and the past extremum values of the input or the output [6, 10]. This is in contrast to the hysteresis with local memory, of which the output is only dependent on the output at some time instant in the past [6].

$$\mathrm{Sgn}(x) \triangleq \begin{cases} x/\|x\| & \text{if } \|x\| \neq 0 \\ \{z \in \mathbb{R}^n \mid \|z\| \leq 1\} & \text{if } \|x\| = 0 \end{cases} \tag{2}$$

$$\mathrm{sat}(Z, x) \triangleq \begin{cases} Zx/\|x\| & \text{if } \|x\| > Z \\ x & \text{if } \|x\| \leq Z \end{cases} \tag{3}$$

where $x \in \mathbb{R}^n$, $Z \in \mathbb{R}_+$ and $\|\cdot\|$ denotes the vector two-norm. If $n = 1$, the $\mathrm{Sgn}(x)$ and $\mathrm{sat}(Z, x)$ can be depicted as Fig. 1(a) and Fig. 1(b), respectively.

Theorem 1. *For $x, y \in \mathbb{R}^n$ and $Z \in \mathbb{R}_+$, the following relation holds true [16, 17]:*

$$y \in Z\mathrm{Sgn}(x - y) \Leftrightarrow y = \mathrm{sat}(Z, x). \tag{4}$$

Proof. : A proof can be given as follows:

$$\begin{aligned} & y \in Z\mathrm{Sgn}(x - y) \Leftrightarrow (y = Z(x - y)/\|x - y\| \wedge x \neq y) \vee (y = x \wedge \|y\| \leq Z) \\ & \Leftrightarrow (y = Zx/(Z + \|x - y\|) \wedge x \neq y \wedge \|y\| = Z) \vee (y = x \wedge \|x\| \leq Z) \\ & \Leftrightarrow (y = Zx/(Z + \|x - y\|) \wedge \|x\| = Z + \|x - y\| > Z) \vee (y = x \wedge \|x\| \leq Z) \\ & \Leftrightarrow (y = Zx/\|x\| \wedge \|x\| > Z) \vee (y = x \wedge \|x\| \leq Z) \Leftrightarrow y = \mathrm{sat}(Z, x). \end{aligned}$$ □

It must be noted that Theorem 1 is a special case of the following relation, which has been used in, e.g., [18, eq.(2.19)], [19, eq.(5)]:

$$x - y \in N_S(y) \Leftrightarrow y = \mathrm{prox}(S, x). \tag{5}$$

Here, $x \in \mathbb{R}^n$, $y \in S \subset \mathbb{R}^n$, S is a closed convex set, $N_S(y)$ is the normal cone to the set S at y, and $\mathrm{prox}(S, x)$ is the "proximal point" function defined as follows:

$$\mathrm{prox}(S, x) \triangleq \operatorname*{argmin}_{z \in S} \|z - x\|^2. \tag{6}$$

Theorem 1 can be obtained by using the relation (5) with $S = \{z \in \mathbb{R}^n \mid \|z\| \leq Z\}$.

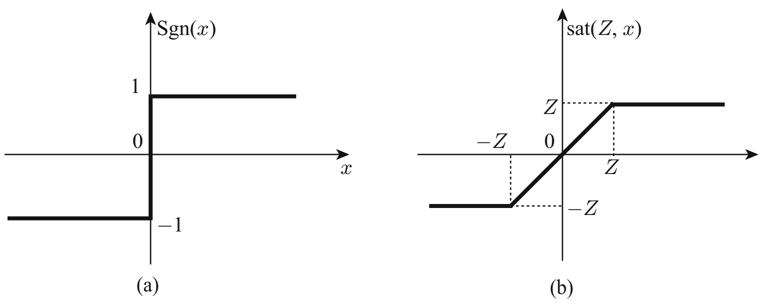


Fig. 1. The graphs of $\mathrm{Sgn}(x)$ and $\mathrm{sat}(Z, x)$

3 Related Work

3.1 LuGre Model, Elastoplastic Model and Leuven Model

LuGre model can be described as follows [3, 4]:

$$\dot{a} = v\left(1 - \mathrm{sgn}(v)\frac{\kappa a}{g(v)}\right) \tag{7a}$$

$$f = \kappa a + \sigma \dot{a} + b(v) \tag{7b}$$

where $a \in \mathbb{R}$ is a variable interpreted as the average deflection of asperities, $\kappa, \sigma, \in \mathbb{R}_+$ are the stiffness and micro-damping coefficient, respectively, f is the friction force, $v \overset{\Delta}{=} \mathrm{d}p/\mathrm{d}t$ where $p \in \mathbb{R}$ is the relative displacement between two surfaces in contact, $b(v)$ is a general term for modeling the memoryless velocity-dependent effect, $g(v)$ is a function that approximates the Stribeck effect and usually it is chosen as follows:

$$g(v) \overset{\Delta}{=} F_c + (F_s - F_c)e^{-\|v/v_s\|^{\alpha}} \tag{8}$$

where F_c and F_s are the magnitude of Coulomb friction force and static friction force, respectively, $\alpha \in \mathbb{R}_+$ is a constant, v_s is the Stribeck velocity. This model produces positional drift behavior and it lacks of nonlocal memory.

The elastoplastic model [5] partially resolves the positional drift problem. However, as pointed out in [1, 12], this model shares another problem with the LuGre model, i.e., lack of nonlocal memory. The Leuven model [6, 11] allows accurate modeling of frictional hysteresis behavior with nonlocal memory in the presliding state. As pointed out in [1, 12], the nondrifting property, frictional lag and transition behavior exhibited by this model is not so faithful to experimental results.

3.2 Generalized Maxwell-Slip Friction Model

The generalized Maxwell-slip (GMS) model [1] incorporates the LuGre-like model into each element of this model to replace the Coulomb law in the kinetic friction state. This model is expressed as follows:

$$f = \sum_{i=1}^{N}(\kappa_i a_i + \sigma_i \dot{a}_i) + b(v) \tag{9}$$

where $\kappa_i, \sigma_i \in \mathbb{R}_+$ is the stiffness and micro-damping of the ith element, respectively. The dynamic of each element is determined as follows:

(i) If the ith element is in the presliding state, the state equation is as follows:

$$\dot{a}_i = v \tag{10}$$

(ii) If the ith element is in the kinetic friction state, the state equation is as follows:

$$\dot{a}_i = \mathrm{sgn}(v) C_i \left(1 - \mathrm{sgn}(v) \frac{a_i}{\lambda_i g(v)}\right) \tag{11}$$

where $\sum_{i=1}^{N} \lambda_i = 1$ and C_i is the attraction parameter determining how fast a_i converges to $\lambda_i g(v)$.

This model is consisted of N parallel elements, as illustrated in Fig. 2, and each element is governed by (10)(11). This GMS model is consistent with experimental results in many frictional phenomena, such as hysteresis with nonlocal memory, frictional lag, stick-slip motion, nondrifting property [1, 12]. This model, however, provides ambiguous conditions to distinguish the presliding state and kinetic friction state. This makes implementation difficult and this model less attractive.

3.3 Differential-Algebraic Single-State Friction Model

In [16], the authors proposed a differential-algebraic friction model based on Kikuuwe et al.'s work [20] for the case where $F_s = F_c$ in (8). This model is derived from the following differential algebraic inclusion (DAI):

$$0 \in \kappa a + \sigma \dot{a} - F_c \mathrm{Sgn}(v - \dot{a}) \tag{12a}$$

$$f = \kappa a + \sigma \dot{a} \tag{12b}$$

The right hand side of (12a) relaxes $\mathrm{Sgn}(v - \dot{a})$ from $\mathrm{Sgn}(v)$. This means that the presliding state condition is $v = \dot{a}$ instead of $v = 0$. This relaxation is acceptable because in presliding state, the contact surfaces stick to each other macroscopically, but microscopically, due to shear loading of the external force, the asperities on the surfaces have relative displacement a and velocity $\dot{a}$ [18]. A physical interpretation of the approximation (12) can be illustrated as Fig. 3. A friction force described by $F_c \mathrm{Sgn}(v - \dot{a})$ acts on a massless object whose velocity is $v - \dot{a}$, and a viscoelastic element with the stiffness κ and the viscosity σ produces the force f in (12b), which exactly balances the friction force.

By the direct application of Theorem 1, (12) can be equivalently rewritten as follows:

$$\dot{a} = \frac{\mathrm{sat}(F_c, \kappa a + \sigma v) - \kappa a}{\sigma} \tag{13a}$$

$$f = \mathrm{sat}(F_c, \kappa a + \sigma v). \tag{13b}$$

As pointed out in [16], this model overcomes the drawback of positional drift exhibited by the LuGre model. One obvious limitation of (13) is that this model cannot capture the Strebick effect and this limitation will be removed in the next section.

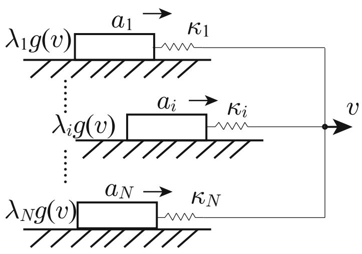


Fig. 2. Schematic representation of (9)

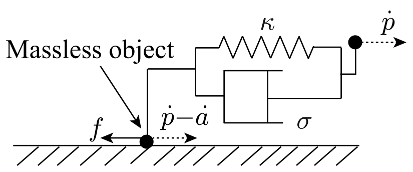


Fig. 3. Schematic explanation of (12)

4 Proposed Differential-Algebraic Multistate Friction Model

4.1 Modification of Previous Differential-Algebraic Single-State Friction Model

The motivation of modifying the previous friction model (13) to capture the Stribeck effect comes from the approximation function (8). The advantage of the previous friction model (13) can be inherited and the limitation can be remedied by modifying (12) as follows:

$$0 \in \kappa a + \sigma \dot{a} - g(v)\mathrm{Sgn}(v - \dot{a}) \tag{14a}$$

$$f = \kappa a + \sigma \dot{a} \tag{14b}$$

By the direct application of Theorem 1, (14) can be equivalently rewritten as follows:

$$\dot{a} = \frac{\mathrm{sat}(g(v), \kappa a + \sigma v) - \kappa a}{\sigma} \tag{15a}$$

$$f = \mathrm{sat}(g(v), \kappa a + \sigma v). \tag{15b}$$

When the friction force $\|\kappa a + \sigma v\| < g(v)$, the new model (15) is in the presliding state. According to the definition of saturation function (3), $\mathrm{sat}(g(v), \kappa a + \sigma v)$ in (15) reduces to $\kappa a + \sigma v$ and then (15) reduces to the following expression:

$$\dot{a} = v \tag{16a}$$

$$f = \kappa a + \sigma v. \tag{16b}$$

One can observe that (16a) is equal to the state equation (10) that governs a single element of the GMS model and (16b) is equivalent to the friction force produced by a single element of the GMS model (9). The equivalence implies that the modified model (15) possesses the same properties as a single element of the GMS model (9) in the presliding state.

When $\|\kappa a + \sigma v\| \geq g(v)$, the new model (15) is in the kinetic friction state. The function $f = \text{sat}(g(v), \kappa a + \sigma v)$ saturates the friction force, i.e., $f = \text{sgn}(\kappa a + \sigma v) g(v)$. Then, equation (15) reduces to the following equation:

$$\dot{a} = \frac{\text{sgn}(\kappa a + \sigma v) g(v) - \kappa a}{\sigma} \tag{17a}$$

$$f = \text{sgn}(\kappa a + \sigma v) g(v). \tag{17b}$$

Equation (17a) can be equivalently rewritten as follows:

$$\dot{a} = \frac{\text{sgn}(\kappa a + \sigma v) g(v)}{\sigma} \left(1 - \text{sgn}(\kappa a + \sigma v) \frac{\kappa a}{g(v)}\right) \tag{18}$$

One can see that the similar expressions among (18), (11) and (7a). For constant velocity $v \neq 0$ and in steady state i.e., $\dot{a} = 0$, one can derive $\kappa a = \text{sgn}(\kappa a + \sigma v) g(v)$ from (18), $a_i = \lambda_i g(v)$ from (11) and $\kappa a = \text{sgn}(v) g(v)$ from (7a). The corresponding friction force is $f = \kappa a = \text{sgn}(\kappa a + \sigma v) g(v)$ for the new model (15), $\text{sgn}(v) k_i \lambda_i g(v)$ for the ith element of the GMS model (9) and $\text{sgn}(v) g(v)$ for the LuGre model (7). This means that in the kinetic friction state, the friction force is described by functions describing Stribeck effect for the three models. In this sense, the three models are equivalent in the kinetic friction state.

4.2 Extension to Differential-Algebraic Multistate Friction Model

The modified model (15) is a single-state friction model. Motivated by the GMS model (9), it is natural to extend the model (15) to a new multistate friction model of which each single element is governed by (15). The result of extension of (15) is as follows:

$$\dot{z} = \frac{g(v) - z}{\tau_d} \tag{19a}$$

$$\dot{a}_i = \frac{\text{sat}(\lambda_i z, k_i a_i + \sigma_i v) - k_i a_i}{\sigma_i}, \ i = 1, \cdots, N \tag{19b}$$

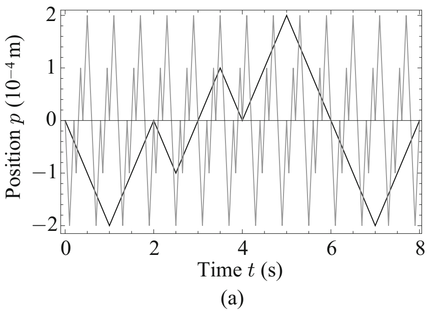
$$f = \sum_{i=1}^{N} \text{sat}(\lambda_i z, k_i a_i + \sigma_i v) + b(v) \tag{19c}$$

where z is a new state variable and τ_d is a new parameter that can be interpreted as the dwell time. The role of τ_d is similar to that of C_i in (11), which determines how fast z converges to $g(v)$. Again, here, $\sum_{i=1}^{N} \lambda_i = 1$. Equation (19a) is motivated by Gonthier et al.'s method [21]. In the same way with (11), here, the method to approximately describe the Stribeck effect for the ith element is given by a simple function $\lambda_i z$. The difference is that here z is used to replace $g(v)$ in (11).

Comparing to the GMS model (9), the new friction model (19) has a more compact expression that describes both the presliding and kinetic friction states into one equation (19b). It is not necessary to separately distinguish the two states during implementing the new friction model. The parameter C_i in (11) is eliminated in (19) and the new parameter τ_d can be used to adjust the transition behavior and frictional lag easily. Those make the model (19) easier than the GMS model to be implemented. Moreover, the model (19) can capture all the frictional phenomena that the model (9) does capture. This can be illustrated by simulation.

4.3 Properties and Simulations

In simulations throughout this section, (19) is applied with $N = 10$ and the timestep size is 0.001s. Fig. 4(a) shows the input triangle-like positional signal chosen so that the hysteresis with nonlocal memory becomes visible. The gray curve in Fig. 4(a) is 10 times the frequency of the black one. Fig. 4(b) clearly shows that the resulting friction force has closed internal loops whose top tips overlap the trajectory of outer loop. It also shows that the model (19) is rate-independent since the resulting friction force is independent from the frequency of position input. Fig. 5(a) shows two sinusoidal input velocity signals with the same magnitude but different frequencies. Fig. 5(b) shows that the extremum of friction force during the transition from the presliding state to kinetic friction state is influenced by both the frequency and the parameter τ_d. This means that the parameter τ_d can be used to adjust the extremum of friction force. This is an important difference from the GMS model. Fig. 6 shows the frictional lag in the case where an unidirectional velocity signal is applied. This velocity signal is obtained by offsetting a sinusoidal signal. Fig. 6(b) shows a higher frictional force for increasing velocity than that for decreasing velocity and $g(v)$ locating in the middle of them. In the same way as in Fig. 5(b), both the frequency and the parameter τ_d have effects on the shape of friction force-velocity curve in Fig. 6(b). Fig. 7(a) is the applied force proposed by Dupont et al. [5] to check



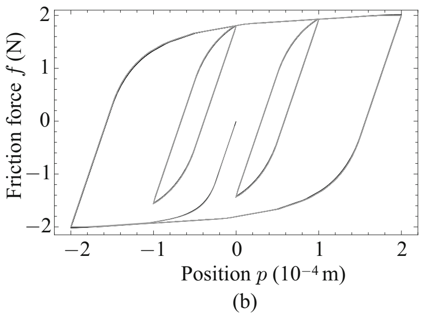


Fig. 4. Simulation result of hysteresis with nonlocal memory with the new model (19) in the presliding state: (a) the input position signal p as a function of time t (frequency changes by factor 10 between the black and gray curves). (b) the friction force f as a function of position p with the two different frequencies. The parameters are chosen as; $\alpha = 2$, $v_s = 9.8 \times 10^{-5}$m/s, $F_c = 2$N, $F_s = 3$N, $\kappa_i = (1 + 0.8(i - 1)) \times 10^3$N/m, $\sigma_i = 5$kg/s, $\lambda_i = 0.0955 + (i - 1) \times 10^{-3}$, $i \in \{1, 2, \cdots, 10\}$, $\tau_d = 0.02$s, $b(v) = 0$.

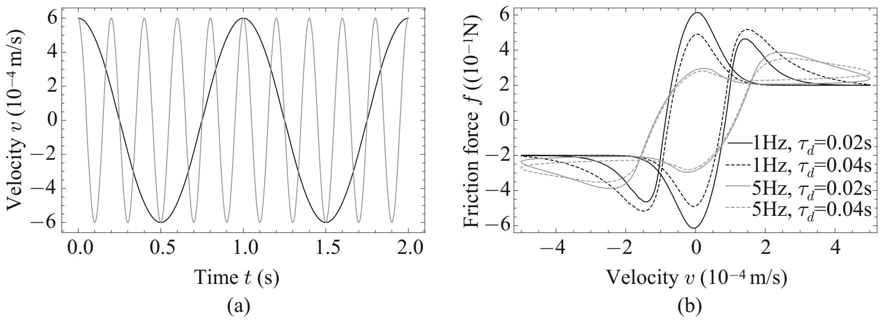


Fig. 5. Simulation result of transition behavior with the new model (19): (a) the input velocity signal v as a function of time t. (b) the friction force f as a function of position v. The parameters are chosen as; $\alpha = 2$, $v_s = 9.8 \times 10^{-5}$m/s, $F_c = 0.2$N, $F_s = 1$N, $\kappa_i = (1 + 0.8(i-1)) \times 10^4$N/m, $\sigma_i = 10^2$kg/s, $\lambda_i = 0.0955 + (i-1) \times 10^{-3}$, $i \in \{1, 2, \cdots, 10\}$, $b(v) = 0$.

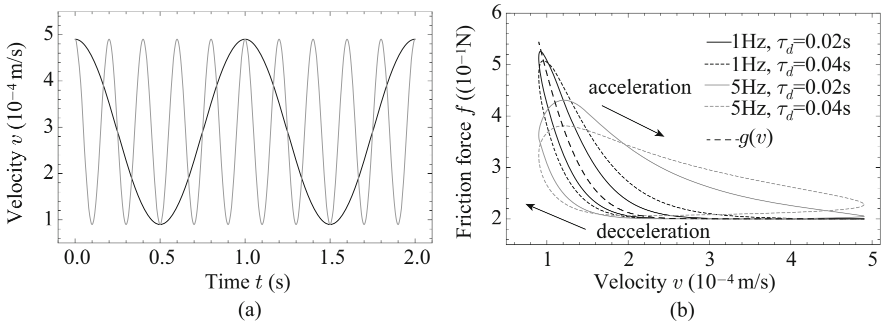


Fig. 6. Simulation result of frictional lag with the new model (19): (a) the input velocity signal v as a function of time t. (b) the friction force f as a function of position v. The parameters are chosen as; $\alpha = 2$, $v_s = 9.8 \times 10^{-5}$m/s, $F_c = 0.2$N, $F_s = 1$N, $\kappa_i = (1+0.8(i-1)) \times 10^3$N/m, $\sigma_i = 10^2$kg/s, $\lambda_i = 0.0955 + (i-1) \times 10^{-3}$, $i \in \{1, 2, \cdots, 10\}$, $b(v) = 0$.

the nondrifting property of their friction model. Fig. 7(b) shows that the position oscillates without shift.

Another set of simulations are performed with the scenario illustrated as Fig. 8 to investigate the stick-slip motion. A mass M, subjected to friction force f, is connected to a spring with stiffness κ. The free end of the spring is driven with constant velocity v_c. As pointed out in [1], although the LuGre model, elastoplastic model and Leuven model all can simulate stick-slip phenomenon, their simulation results are qualitatively different from the experimental results in [10] and from the simulation results of the generic friction models in [7, 8]. Fig. 9 illustrates the simulation result of the new model (19). Fig. 9(a) shows the step position and pulse velocity with oscillation. The dashing circles in Fig. 9(b) denote the spike-like force followed by high frequency oscillation after each stick

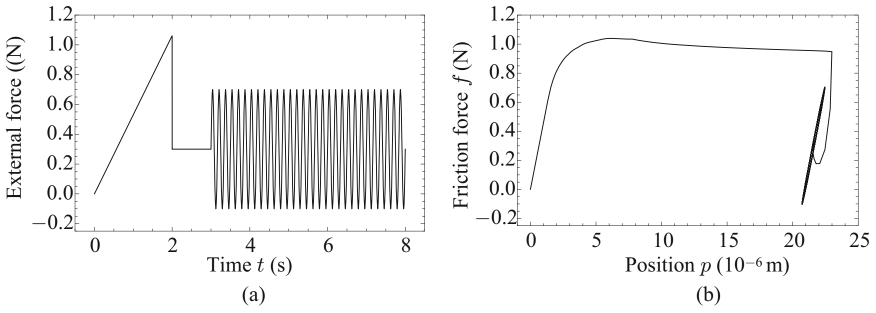


Fig. 7. Simulation result of nondrifting property with the new model ([19]): (a) the input external force as a function of time t. (b) the friction force f as a function of position p. The parameters are chosen as; $\alpha = 2$, $v_s = 9.8 \times 10^{-5}$m/s, $F_c = 0.2$N, $F_s = 1.1$N, $\kappa_i = (1 + 0.8(i-1)) \times 10^4$N/m, $\sigma_i = 10^2$kg/s, $\lambda_i = 0.0955 + (i-1) \times 10^{-3}$, $i \in \{1, 2, \cdots, 10\}$, $\tau_d = 0.2$s, $b(v) = 0$.

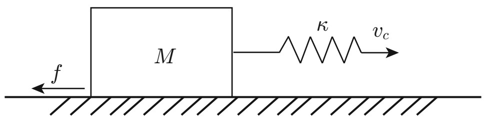


Fig. 8. The simulation scenario for investigating stick-slip phenomenon

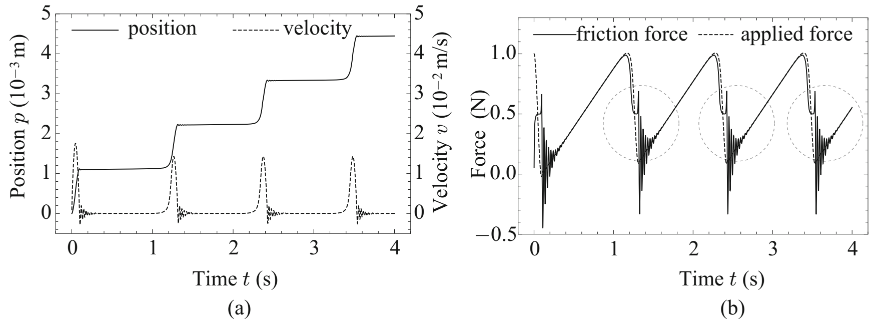


Fig. 9. Simulation result of stick-slip behavior with the new model ([19]): (a) the position and velocity of the mass M as a function of time t. (b) the applied force and friction force as a function of time t.. The parameters are chosen as; $M = 1$kg, $\kappa = 10^3$N/m, $v_c = 10^{-3}$m/s, $\alpha = 2$, $v_s = 4 \times 10^{-3}$m/s, $F_c = 0.5$N, $F_s = 1$N, $\kappa_i = (1 + 0.8(i - 1)) \times 10^3$N/m, $\sigma_i = (0.01 + 0.005(i - 1)) \times 10^2$kg/s, $\lambda_i = 0.0955 + (i - 1) \times 10^{-3}$, $i \in \{1, 2, \cdots, 10\}$, $\tau_d = 0.01$s, $b(v) = 0$.

period. Those phenomena are consistent with the GMS model ([9]), the generic model in [7, 8] and the empirical validation in [10] while the LuGre model, elastoplastic model and Leuven model cannot produce spike-like and oscillation force after each stick period.

5 Conclusion

The paper has introduced a new multistate friction model by extending the authors' differential-algebraic single-state friction model. This multistate friction model consists of multiple parallel elements. Each element is described by the differential-algebraic single-state friction model, which incorporates the presliding state and kinetic friction state into one equation (19b). For each element of this model, only three parameters κ_i, σ_i and λ_i are needed. For gross formulation of this model, the function $g(v)$ approximates the Stribeck effect and the parameter τ_d is used to effectively adjust the frictional lag and transition behavior. The effectiveness of this new model is illustrated by simulation results of hysteresis with nonlocal memory, frictional lag, transition behavior, nondrifting property and stick-slip behavior. Those simulation results are consistent with the results in the literature from experiments and simulations based on physical motivated models at asperity level.

Future work should investigate various properties of the new model such as effects of parameters and applications such as friction identification and compensation in various mechanical systems for precise control purposes. In addition, a further extension of the model to asperity level may be thought as a future topic of study. In this topic, maybe a model, which consists of multiple asperities governed by the single-state friction model, combined with material and geometrical information of the surfaces, can reveal more complex phenomena of friction observed from experimental study.

References

1. Al-Bender, F., Lampaert, V., Swevers, J.: The generalized maxwell-slip model: a novel model for friction simulation and compensation. IEEE Trans. on Automatic Control 50(11), 1883–1887 (2005)
2. Dahl, P.: A solid friction model. Tech. rep., Aerospace Corporation, El Segundo, CA (1968)
3. Canudas de Wit, C., Olsson, H., Åström, K.J., Lischinsky, P.: A new model for control of sytem with friction. IEEE Trans. on Automatic Control 40(3), 419–425 (1995)
4. Åström, K.J., Canudas-de-Wit, C.: Revisting the LuGre friction model. IEEE Control Systems Magazine 28(6), 101–114 (2008)
5. Dupont, P., Hayward, V., Armstrong, B., Altpeter, F.: Single state elastoplastic friction models. IEEE Trans. on Automatic Control 47(5), 787–792 (2002)
6. Swevers, J., Al-Bender, F., Ganseman, C.G., Prajogo, T.: An integrated friction model structure with improved presliding behavior for accurate friction compensation. IEEE Trans. on Automatic Control 45(4), 675–686 (2000)
7. Al-Bender, F., Lampaert, V., Swevers, J.: A novel generic model at asperity level for dry friction force dynamics. Tribology Letters 16(1-2), 81–93 (2004)
8. Moerlooze, K.D., Al-Bender, F., Brussel, H.V.: A generalized asperity-based friction model. Tribology Letters 40(1), 113–130 (2010)
9. Al-Bender, F., Swevers, J.: Characterization of friction force dynamics. IEEE Control Systems Magazine 28(6), 64–81 (2008)

10. Lampaert, V., Al-Bender, F., Swevers, J.: Experimental characterization of dry friction at low velocities on a developed tribometer setup for macrosopic measurements. Tribology Letters 16(1-2), 95–105 (2004)
11. Lampaert, V., Swevers, J., Al-Bender, F.: Modification of the leuven integrated friction model structure. IEEE Trans. on Automatic Control 47(4), 683–687 (2002)
12. Lampaert, V., Al-Bender, F., Swevers, J.: A generalized maxwell-slip friction model appropriate for control purposes. In: Proc. of IEEE International Conference on Physics and Control, St. Petersburg, Russia, pp. 1170–1177 (2003)
13. Iwan, W.D.: A distributed-element model for hysteresis and its steady-state dynamic response. Trans. ASME: J. of Applied Mechanics 33(4), 893–900 (1966)
14. Goldfarb, M., Celanovic, N.: A lumped parameter electromechnical model for describing the nonlinear behavior of piezoelectric actuators. Trans. ASME: J. of Dynamic Systems, Measurement, and Control 119, 478–485 (1997)
15. Lazan, B.J.: Damping of Materials and Members in Structural Mechanics. Pergamon Press, London (1968)
16. Xiong, X., Kikuuwe, R., Yamamoto, M.: A differential-algebraic method to approximate nonsmooth mechanical systems by ordinary differential equations. Submitted to Multibody System Dynamics
17. Kikuuwe, R., Yasukouchi, S., Fujimoto, H., Yamamoto, M.: Proxy-based sliding mode control: a safer extension of PID position control. IEEE Trans. on Robotics 26(4), 670–683 (2010)
18. Leine, R.I., Nijmeijer, H.: Dynamics and Bifurcations of Non-smooth Mechanical Systems. LNACM, vol. 18. Springer, Berlin (2004)
19. Brogliato, B., Daniilidis, A., Lemarechal, C., Acary, V.: On the equivalence between complementarity systems, projected systems and differential inclusions. Systems & Control Letters 55(1), 45–51 (2006)
20. Kikuuwe, R., Takesue, N., Sano, A., Mochiyama, H., Fujimoto, H.: Admittance and impedance representations of friction based on implicit Euler integration. IEEE Trans. on Robotics 22(6), 1176–1188 (2006)
21. Gonthier, Y., Mcphee, J., Lange, C., Piedbœuf, J.C.: A regularized contact model with asymmetric damping and dwell-time dependent friction. Multibody System Dynamics 11(3), 209–233 (2004)

Simulation of Flexible Objects in Robotics

Andreas Rune Fugl[1,2], Henrik Gordon Petersen[1], and Morten Willatzen[2]

[1] The Maersk Mc-Kinney Moller Institute, University of Southern Denmark
[2] The Mads Clausen Institute, University of Southern Denmark

Abstract. In this paper, we present what appears to be the first simulation model for grasping of flexible bodies based on the three-dimensional elastic constitutive relations and Newton's Second Law for solids known as the Navier-Cauchy equations. We give an overview of the most important equations for strain, stress, and elasticity tensors based on which we outline the format of the Navier-Cauchy equations of motion in the general anisotropic case. We then specifically study the equations for homogeneous isotropic bodies. We formulate a numerical scheme based on finite differences for solving the equations. Finally, we present preliminary experimental work and outline future directions.

Keywords: robotics, grasping, elasticity, finite difference.

1 Introduction

Realistic simulation of robotic grasping of objects has recently received substantial attention and there are now a variety of grasp simulators available. Each of the simulators uses a *simulation core*, that numerically implements a mathematical model of the physical processes affecting the grasping device and in particular the object to be grasped.

In addition, each of the simulators has various interfaces such as robot kinematic descriptions, libraries for collision checking etc. helping the user in applying the simulation tool to a practical application. It should be made clear, however, that it is the choice of simulation core that decides how close to reality the grasping is.

The most widely used dextrous grasp simulator is GraspIt [1] developed by Andrew Miller. The simulation core was based on implementing the grasping-object contact model as a linear complementary problem [2] and to solve it using Lemkes algorithm. Other grasp simulators have since then become available. The grasp simulators [3] and [4] rely on the simulation core Open Dynamics Engine whereas the recent grasp simulator dVC3d [5] uses the path solver in a modified version of the Bullet simulator. However, the simulation cores for these grasp simulators are only applicable for handling grasping of rigid bodies. There is thus a need to extend the simulation cores with the physical models and numerical methods necessary to simulate deformable objects.

Simulating grasping of deformable objects is a difficult task since several approximations have to be made. First, a mathematical model of the deformations as a function of external forces has to be established. This is done in the framework of the Navier-Cauchy equations, i.e., Newton's Second Law for an elastic solid. Hitherto, in the literature, a detailed description of elasticity in robotic applications is not available. We present here a formalism for handling elastic bodies under action of external surfaces (robotic grasping as an example) and body forces. The model presented accommodates, in principle, any underlying crystal structure but for the present purpose, to keep things simple, we assume the deformed body is isotropic, i.e., two independent stiffness constants suffice to describe local stress-strain relations. This is a good approximation in many engineering applications involving macroscopic solid deformations, similar to those described here. There exist numerous applications of the Navier-Cauchy equations in classical and nanoscale applications [6,7,8].

The second approximation is to transfer the mathematical model to a feasible numerical implementation. Here, two issues are important: It must be possible to compute the solution within a reasonable time and it must be possible to estimate the error in the numerical solution as compared to the theoretically correct solution for the given mathematical model. There are two widely used and popular approaches, namely Finite Element Methods and Finite Difference Methods. The advantage of Finite Element Methods is that it is somewhat easier to implement boundary conditions and that various well-proven packages are available, such as [9,10,11]. The disadvantage is that no reliable estimate of the error can be provided. The advantage of Finite Difference Methods introduced by [12] is that the error can be accurately estimated by Richardson extrapolation.

In [13] a simulator was developed for real-time simulation and visualization of cardiac surgery. The mass-spring topology, used in the real-time simulation was trained by a neural network using a Finite Element Method of linear elasticity as the evaluation function. However, the modeling and numerical errors were not evaluated as thoroughly as could be desired. Simulations were instead subjectively evaluated and tweaked requiring feedback from end-users.

In [14] two specific categories of flexible objects were explored: Cloth and string. Robot setups were constructed to perform various manipulation tasks. Although successful in their manipulation tasks, a method for simulating diverse objects was not developed.

To our knowledge no rigorous study has been reported within robot grasp simulation, taking the above proper physical and mathematical approaches to modelling and the numerical solving of object deformations into account. Robotic grasping involves mixed boundary conditions, since the surface of the object will be subject to Dirichlet boundary conditions (prescribed deformations) at the grasp points, and other exterior contacts and free boundary conditions (no stress) at all non-contact locations on the surface.

In this paper we provide a rigorous derivation of the algorithms for a simulation core for deformable objects. Starting with the constitutive laws and the field equations we will derive the partial differential equation governing the interior of the flexible object, and subsequently derive the equations for the necessary boundary conditions for the grasping of flexible objects. We do this for the simple useful case of isotropic homogeneous linear elastic objects, but our results can be straightforwardly modified to cases with varying degrees of anisotropy.

The paper is organized as follows: First we present the theory of elasticity, ending up with the equations of motions for an elastic material. We then derive the necessary boundary conditions for handling various grasping scenarios. Next we present a numerical method for solving the equations of motion, and last we show early experimental results comparing our method to real world grasping of deformable objects.

2 Linear Elasticity

In this section we summarize the notation and constitutive relations used in general linear elasticity as presented by [15] and [16].

We formulate the problem of determining the mechanical response, as the solution to the system of partial differential equations (PDEs) known as the Navier-Cauchy (NC) equations. We then derive the set of necessary boundary conditions needed for robot grasping scenarios.

2.1 Deformation Description

Let $\boldsymbol{x}$ be an arbitrary point in the material in the absence of deformation, and let $\boldsymbol{x'}$ be the position of the point when deformations are added. The *displacement vector* for a point is thus $\boldsymbol{u} = \boldsymbol{x'} - \boldsymbol{x}$, or in component form

$$u_i = x'_i - x_i$$

where $i = 1, 2, 3$ refers to the generalized coordinates.

Obviously, if the displacement vector u_i is known for every point inside the body, the deformations or strains are also known.

2.2 Strain Tensor

We use the following expression for the *infinitesimal strain tensor* u_{ij}:

$$u_{ij} = \frac{1}{2}\left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j}\right) \tag{1}$$

The general expression for strain includes second-order terms in $\frac{\partial u_i}{\partial x_j}$ but these are usually neglected as the strains are assumed to be small.

Notice that strains by definition are *relative* length changes. Small strains in a long thin object may therefore yield large deformations.

2.3 Stress Tensor

The motivation for having a stress tensor is to account for body and surface forces and relate them to strains in a compact and mathematically well-defined way. We define the *stress tensor* as σ_{ij}, where the first index i is the direction of the force component and the second index j is the direction of the surface normal of the area upon which the force acts.

2.4 The Tensor of Elasticity

The most basic assumption in relating stress to strain is *Hooke's Law*, i.e. *local stresses are proportional to the local strains.* Hooke's law can be formulated as

$$\sigma_{ij} = \sum_{k,l} C_{ijkl} u_{kl} \tag{2}$$

where i, j, k, l all take the values $1, 2, 3$. C_{ijkl} is thus a tensor of fourth rank known as the *tensor of elasticity* or *stiffness tensor.* This stiffness tensor is thus a multi-component analogue to the well known *spring constant* in Hooke's Law.

The stiffness parameters C_{ijkl} only depend on the elastic material. Thus, if the material is homogeneous, each of the stiffness coefficients is constant throughout the material volume. Even if the material is homogeneous, it seems that we have $3^4 = 81$ coefficients describing the elastic properties of a solid. However, it can be proven from free-energy considerations [15] that the tensor is symmetric in interchanging i with j, k with l and i, j with k, l. This leaves us with "only" 21 independent C_{ijkl} components in the general case. Typically, the solid has some symmetries that further reduces the set of independent C_{ijkl} components. A thorough analysis is given in [15]. Here, we only state the result for the most symmetric case, namely an isotropic solid which is described by only 2 independent stiffness coefficients. In the isotropic case, these parameters are often given as either the Lamé elastic constants λ, μ or as Young's modulus E and the Poisson ratio ν. The relations between the various choices of parameters are

$$\begin{aligned} C_{1111} &= 2\mu + \lambda = \frac{E}{1+\nu}(1 + \frac{\nu}{1-2\nu}) \\ C_{1122} &= \lambda = \frac{E}{1+\nu}(\frac{\nu}{1-2\nu}) \\ 2C_{1212} &= 2\mu = \frac{E}{1+\nu} \end{aligned} \tag{3}$$

Hence, for an isotropic solid, it follows that C_{1212} satisfies: $C_{1212} = \frac{C_{1111} - C_{1122}}{2}$.

We have chosen to formulate our method using the Lamé elastic constants. Equation (2) then reduces to

$$\sigma_{ij} = 2\mu u_{ij} + \lambda(u_{11} + u_{22} + u_{33})\delta_{ij} \tag{4}$$

where δ_{ij} is the unit tensor (δ_{ij} is one if $i = j$ and zero if $i \neq j$).

2.5 Navier-Cauchy Equations of Motion

The Navier-Cauchy equations of motion for a general elastic material are quite complicated partly due to the many non-zero stiffness tensor components. The equations of motion in the general case and each of the symmetry cases are all presented in [15]. In general, they can be written in the form

$$\rho \frac{\partial^2 \boldsymbol{u}}{\partial t^2} = \boldsymbol{A}(\boldsymbol{C})\boldsymbol{\partial^2 u} \tag{5}$$

where ρ is the mass density of the material and

$$\boldsymbol{\partial^2 u} = \{\frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial x \partial y}, \ldots, \frac{\partial^2 u}{\partial z^2}\} \tag{6}$$

containing the 6 second order partial derivatives and $\boldsymbol{A}(\boldsymbol{C})$ is a 3×6 coefficient matrix that only depends on the stiffness tensor.

For an isotropic material, the Navier Cauchy equations of motion are

$$\rho \frac{\partial^2 \boldsymbol{u}}{\partial t^2} = (\lambda + \mu)\nabla(\nabla \cdot \boldsymbol{u}) + \mu \nabla^2 \boldsymbol{u} + q(\boldsymbol{x}) \tag{7}$$

where the source term for the external forces $q(x)$ has been added. In the present work this consists of body force due to gravity $g\rho(x)$ where g is the gravitational acceleration and $\rho(x)$ is the mass density.

2.6 Boundary Conditions

For the case of robotics simulations involving the grasping of flexible objects, we need to consider two types of boundary conditions, both shown in Fig. 1.

The first situation is when material points on the boundary are forcibly moved to a set displacement, e.g. by a robot gripper. This also applies to points on the boundary that are resting on a surface. These points will be described below by "Dirichlet boundary conditions". The second situation applies to boundary locations, where the object boundary is not in contact with anything. These will be described below by "free boundary conditions". Both boundary conditions support polyhedron surfaces, such as shown on Fig. 2.

For *Dirichlet boundary conditions*, the value of the displacements are specified on the boundary. We can thus write these boundary conditions as

$$\boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{f}(\boldsymbol{x}, t) \quad \text{for all x on the Dirichlet boundary}$$

A *free boundary condition* corresponds to zero normal and shear stress components along the boundary normal. These boundary conditions are slightly more involved:

$$\sum_{j=1}^{3} \sigma_{ij} n_j = 0 \quad i = 1, 2, 3 \tag{8}$$

where σ_{ij} is again the stress tensor, and n_j is jth component of the boundary surface normal at the given location.

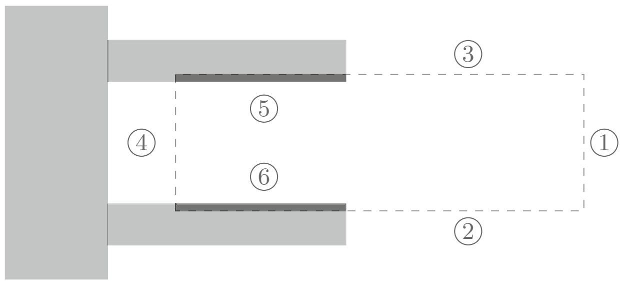


Fig. 1. A simplified view of a parallel gripper holding a flexible object. Surface regions 1 through 4 are modelled by free boundary conditions. Regions 5 and 6 are in direct contact with the gripper and are modelled by Dirichlet boundary conditions.

By the constitutive law for an isotropic, elastic solid (4) we can transform the BC (8) to an expression in terms of strains.

In the general case, these boundary conditions can be written in the form

$$\boldsymbol{B}(\boldsymbol{C})\boldsymbol{\partial u} = 0 \tag{9}$$

where

$$\boldsymbol{\partial u} = \{\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial u}{\partial z}\} \tag{10}$$

and $\boldsymbol{B}(\boldsymbol{C})$ is a 3×3 coefficient matrix that again only depends on the stiffness tensor.

Using Eq. (1) together with Eq. (7), we may obtain these boundary conditions for the isotropic case directly as

$$\sum_{j=1}^{3} \mu n_j \left(\frac{\partial u_j}{\partial x_i} + \frac{\partial u_i}{\partial x_j} \right) \tag{11}$$
$$+\lambda n_i \sum_{k=1}^{3} \frac{\partial u_k}{\partial x_k} = 0 \quad i = 1, 2, 3$$

3 Numerical Method by Finite Differences

Using the Finite Difference Method, the continuum representation of the Navier-Cauchy equations is approximated with a discrete set of algebraic equations. The equations are formed by approximating the partial derivatives of the Navier-Cauchy equations with finite differences on a rectangular spaced discrete set of points distributed over the original continuous domain. More specifically, we choose a coordinate system in which we place the undeformed object. We then define gridsizes H_x, H_y, H_z. The object is now represented by all the points $(X_i, Y_j, Z_k) \equiv (iH_x, jH_y, kH_z)$ residing inside or on the boundary of the object. Our discretization is then augmented with one complete layer of 'imaginary points' around the object (see Fig. 2). Each non-imaginary point thus has a neighbourhood of 18 nearest and next nearest points shown in Fig. 3. With these points it is easy to establish centered second order finite difference approximations for all first and second order derivatives.

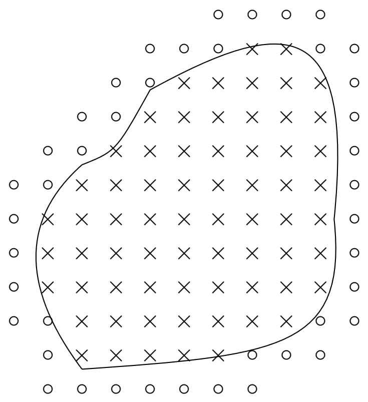


Fig. 2. Two-dimensional illustration of the discretization procedure for a continuum representation. The interior and boundary are discretized and subsequently augmented by imaginary points, marked with circles.

An example of a finite difference is

$$\frac{\partial u(X_i, Y_j, Z_k)}{\partial x} \simeq \frac{u(X_{i+1}, Y_j, Z_k) - u(X_{i-1}, Y_j, Z_k)}{2H_x}$$

where the approximation error is second order in H_x. Another example is

$$\begin{aligned} \frac{\partial^2 u(X_i, Y_j, Z_k)}{\partial x \partial y} &\simeq \frac{1}{4H_x H_y} \times \\ &[u(X_{i+1}, Y_{j+1}, Z_k) - u(X_{i-1}, Y_{j+1}, Z_k) \\ &-u(X_{i+1}, Y_{j-1}, Z_k) + u(X_{i-1}, Y_{j-1}, Z_k)] \end{aligned}$$

where the approximation error is proportional to $H_x H_y$.

Similarly second order finite differences can be found for all the other first and second order derivatives.

Discretization of the Navier-Cauchy Equations

To illustrate how the Navier-Cauchy equations can be discretized, we consider the f_x component in the isotropic case given in Eq. (7). We get

$$\rho \frac{\partial^2 u_x}{\partial t^2} = (\lambda + \mu) \frac{\partial}{\partial x} \left(\frac{\partial u_x}{\partial x} + \frac{\partial u_y}{\partial y} + \frac{\partial u_z}{\partial z} \right) + \mu \left(\frac{\partial^2 u_x}{\partial x^2} + \frac{\partial^2 u_x}{\partial y^2} + \frac{\partial^2 u_x}{\partial z^2} \right) \quad (12)$$

The derivatives on the right hand side is now approximated by second order accurate finite differences as discussed above. We thus obtain a set of second order linear ordinary differential equations. The grid points used in the finite difference approximations of the derivatives are illustrated in Fig. 3.

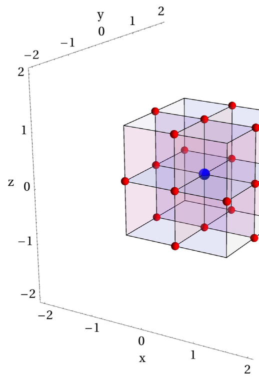


Fig. 3. Visualization of the NC PDE system, approximated by finite differences to second-order. Red points are grid points used in the computation.

Discretization of the Boundary Conditions

Grid points on the boundary of the object are subject to the boundary conditions described in section 2.6. In the discretization, a boundary point is a point inside the object that has at least one imaginary point as a nearest neighbour. The simplest case is to discretize the Dirichlet boundary conditions, defining a known displacement on the boundary point:

$$\boldsymbol{u}(X_i, Y_j, Z_k) = \boldsymbol{f}(X_i, Y_j, Z_k, t) \quad (13)$$

At points where the Dirichlet boundary condition apply, the Navier-Cauchy equation is omitted.

Consider now points on a free boundary. At such a point, both the Navier Cauchy equations and the boundary condition is applied. In order to implement the discretization, we first need to specify the direction of the normal vector. For boundary points having only one imaginary point as nearest neighbour (surface points), we choose the normal as the direction towards that point. If two or three imaginary points are nearest neighbours, we choose the average direction towards these as the surface normal direction. We then obtain surface normals as illustrated in Fig. 4.

The free boundary conditions are then again straightforward to discretize. Both for Dirichlet and free boundary conditions, the discretization thus yields a set of linear algebraic equations.

System of Equations

For each discretization point (X_i, Y_j, Z_k) including the imaginary points, we now define 3 equations for the three unknowns $u_1(X_i, Y_j, Z_k)$, $u_2(X_i, Y_j, Z_k)$, $u_3(X_i, Y_j, Z_k)$. For all non-imaginary points that are not subject to a Dirichlet boundary, we apply the discretized Navier-Cauchy equations. For all points that are subject to that boundary condition, we apply the given Dirichlet condition. Actually, these points could thus be removed from the system of equations. By this procedure, all points except the imaginary points have associated equations.

Consider now the imaginary points. For each imaginary point towards which a normal vector of a free boundary point is directed (see Fig. 4), the associated free boundary conditions are applied. All imaginary points that are not used in any equation are then removed, leaving a number of imaginary points which are used in one or several equations but have not yet received associated equations. An example are those with a filled circle in Fig. 4. For these points, we define the associated equations by second order interpolations. We thus obtain a set of coupled second order linear differential equations and linear algebraic equations for the displacements.

Having formed the linear system, we wish to solve for the unknown displacements. In the present work we do not exploit the sparse nature and structure of the system matrix, as we solve it by standard LU decomposition. It is obvious however, that iterative methods will be much faster and as such this is something we wish to exploit in the future.

4 Experiments and Results

Our derivations of the model and the corresponding finite difference discretization were finalized only recently. Hence, only preliminary experimental studies of the elastic model will be presented. We consider the situation illustrated in Fig. 5. A parallel gripper mounted on a robot grasps a flexible object of silicone rubber, and holds it horizontally. The silicone rubber deforms under its own weight and arrives at a resting position. The distribution of boundary conditions is close to the previously described situation, shown in Fig. 1.

In our setup the gripper closes to hold the object firmly, but not enough to cause any considerable deformation. Points on the boundary in contact with the gripper are assumed to be fixed but not deformed, i.e., we set their deformation value to zero through Dirichlet boundary conditions in order to clamp them in place. The rest of the points on the object surface interface to air, corresponding to no-stress boundary conditions.

Young's modulus for the silicone rubber used was measured experimentally by previous tensile tests to $E \simeq 1\text{MPa}$. The Poisson ratio is tabulated to $\nu = 0.49$

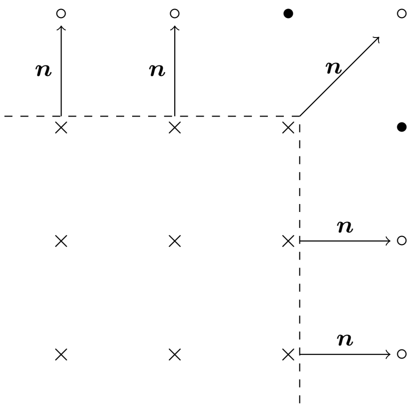


Fig. 4. Region of the computational grid for no-stress boundary conditions. Open circles are imaginary points, with corresponding equations calculated by the no-stress BC applied to the interior point. Filled circles are imaginary points, calculated by interpolation. The dashed lines show the boundary of the original continuum domain.

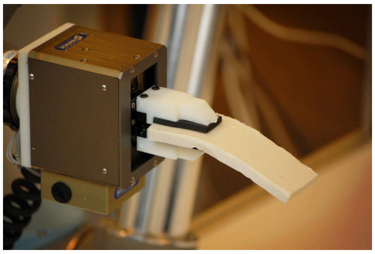


Fig. 5. The experimental setup. A flexible piece of silicone rubber grasped by a parallel gripper deforms under it's own weight.

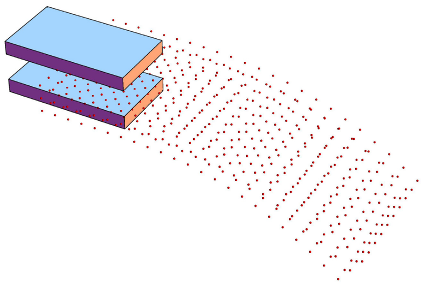


Fig. 6. The deformation of the setup, as calculated by our method

and the mass density is $\rho = 755\ \text{kg/m}^3$. The piece is cut as a cuboid with dimensions 11x55x140 mm. The contact surface of the gripper fingers measures 50x30 mm.

We use the above material parameters with our method. As the object is at rest, the left hand side in Navier-Cauchy's equations containing the accelerations is zero. The resulting object mesh is shown in Fig. 6.

5 Conclusion and Future Work

We have presented a model for elastic deformations of flexible objects grasped by a robot, based on the established three-dimensional linear elasticity equations. We have presented a Finite Difference scheme for a numerical implementation of the model and shown very preliminary experimental tests. In the near future, we will take two directions: a) We plan to perform a variety of tests with homogeneous objects of different shape and material in order to quantitatively study the accuracy of our method. b) In some applications, the linear model is insufficient and we will therefore also study simplified uni-directional non-linear models based on the work by [17].

Each of the models will be incorporated into the grasp simulator RobWorkSim [4] developed at our institute. We will then be able to make realistic simulations of object deformations under grasping, dynamic motion and manipulation.

Acknowledgment. This work was co-financed by the INTERREG 4 programme Syddanmark-Schleswig-K.E.R.N. by EU funds from the European Regional Development Fund.

References

1. Miller, A.T.: Graspit!: A versatile simulator for robotic grasping. PhD thesis, Citeseer (2001)
2. Program, A., Anitescu, M., Potra, F.A.: Formulating dynamic multi-rigid-body contact problems with friction as solvable linear complementarity problems. Nonlinear Dynamics, 231–247 (1997)
3. León, B., Ulbrich, S., Diankov, R., Puche, G., Przybylski, M., Morales, A., Asfour, T., Moisio, S., Bohg, J., Kuffner, J., Dillmann, R.: OpenGRASP: A Toolkit for Robot Grasping Simulation. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 109–120. Springer, Heidelberg (2010)
4. Jørgensen, J., Ellekilde, L., Petersen, H.: RobWorkSim - an Open Simulator for Sensor based Grasping. In: ISR/ROBOTIK 2010 (2010)
5. Nguyen, B., Trinkle, J.: dvc3D: a three dimensional physical simulation tool for rigid bodies with contacts and Coulomb friction. In: The 1st Joint International Conference on Multibody System Dynamics (2010)
6. Johns, J.: Rayleigh waves in a poroelastic half-space. Journal of Acoustical Society of America, 952–962 (1961)
7. Willatzen, M.: Exact power series solutions to axisymmetric vibrations of circular and annular membranes with continuously varying density in the general case. Journal of Sound and Vibration, 981–986 (2002)

8. Schliwa, A., Winkelnkemper, M., Bimberg, D.: Impact of size, shape, and composition on piezoelectric effects and electronic properties of In(Ga)AsGaAs quantum dots. Phys. Rev. B (2007)
9. COMSOL: COMSOL Multiphysics, Burlington, MA (2011)
10. Binder, J.B.: Algor finite element modeling tools aid aerospace. Aerospace America (1995)
11. ANSYS: ANSYS Structural, Canonsburg, PA (2011)
12. Richardson, L.F.: The approximate arithmetical solution by finite differences of physical problems involving differential equations, with an application to the stresses in a masonry dam. Philosophical Transactions of the Royal Society of London, 307–357 (1911)
13. Mosegaard, J.: Cardiac Surgery Simulation - Graphics Hardware meets Congenital Heart Disease. PhD thesis, Department of Computer Science, University of Aarhus, Denmark (October 2006)
14. Bell, M.: Flexible object manipulation. PhD thesis, Dartmouth College Hanover, New Hampshire (2010)
15. Landau, L.D., Pitaevskii, L.P., Lifshitz, E.M., Kosevich, A.M.: Theory of Elasticity, 3rd edn. Butterworth-Heinemann (1986)
16. Feynman, R.: The Feynman Lectures on Physics, vol. 2. Addison-Wesley, Boston (1963)
17. Russell, D.L., White, L.: An elementary nonlinear beam theory with finite buckling deformation properties. SIAM Journal of Applied Mathematics, 1394–1413 (2002)

Continuous Integration for Iterative Validation of Simulated Robot Models

Florian Lier[1], Simon Schulz[1], and Ingo Lütkebohle[2]

[1] Center of Excellence Cognitive Interaction Technology (CITEC), Universitätsstraße 21-23, 33615 Bielefeld, Germany
[2] CoR-Lab Research Institute for Cognition and Robotics, Universitätsstraße 25, 33615 Bielefeld, Germany

Abstract. Simulated environments often provide the first, and are usually the most frequent, test environment for robotic systems, primarily due to their cost and safety advantages. Unfortunately, changing aspects of both, the simulation and the real robot, as well as actuator control algorithms are often not taken into account when relying on simulation results. In this paper we present a continuous integration approach to verify simulated robot models in an integrated and frequent manner, comprising a simulated and a real robot for comparison. The central aspect of our concept is to iteratively assess the fidelity of simulated robot models. In an exemplary case study we distilled a first set of requirements and metrics, which can be used by developers to verify their algorithms and to automatically detect further system changes.

1 Introduction

Simulated environments often provide the first, and are usually the most frequent, test environment for robotic systems, primarily due to their cost and safety advantages. To be useful, such environments must be sufficiently close to the real world, which remains a challenging problem and requires detailed checking to confirm a reasonably close match, i.e., validation.

So far, research in validation has largely focused on the accuracy of the physical simulation. This is a natural choice, as it is the defining difference, but covers only a part of typical simulation engines. In contrast, we focus on the robot model, that is, that the actuation of the simulated robot matches that of the real robot. This requires that the simulator reflects not just the physical model accurately, but also the control algorithm. Depending on robot hardware, this can be challenging, e.g., when firmware source code is unavailable, or when the real robot uses control loops realized in dedicated hardware.

Furthermore, this match must be established not just once, but must be maintained over the course of development on the robot. Many rapidly changing aspects of the robot's actual configuration can affect this match, such as calibration, model updates, manufacturing variations, firmware updates, and so on. Last, but not least, on many research robots, control algorithms are in frequent development, which may obviously cause mismatches.

Thus, it is not surprising that several current simulator engines, particularly general-purpose ones, exhibit significant differences between the simulated and the real actuators (cf. Sec [2]). These differences concern not just physical fidelity, but also certain control modes not being fully implemented and the like. Furthermore, in some areas such as human-robot-interaction, the full dynamics of a movement are relevant, where there may also be differences that are not visible just by comparing end-points, as is often done.

Therefore, in this contribution, we propose a continuous integration (CI) approach towards validating the simulation model, comprised of both a simulated and a real robot. In continuous integration, the differences between the model and the real robot are tested on every change of control software or configuration, both through dedicated tests, and through testing of dependent software. As a result, the consequences of upstream changes can be detected immediately, and are available to all affected developers. For this, a standard CI server has been connected to a physical robot and provided with comprehensive internal and external data capture to support the necessary analysis and testing.

Furthermore, we define a purposefully limited experimental setup that makes validation efficient enough for constant and automated repetition, yet sufficiently insightful to detect relevant changes. As simulation is never exact, we report results from a case study to determine what level of fidelity can reasonably be expected, and which parameters influence it. This information is essential for developers to create appropriate tests for their platform. Some of the conclusions are likely general, but we also expect that some concrete values depend on the robot used, and thus also report on requirements for such testing which should facilitate repetition for other platforms.

2 Related Work

Simulator validation has not been widely studied in robotics so far, but one prominent exception is the "Unified System for Automation and Robot Simulation" (USARSim), a game engine based simulator [1,7]. In order to validate simulation results Pepper et al. define standardized tests which are manually performed (by domain experts) in the real world, e.g., "robot is climbing stairs", and are then compared to simulation results iteratively. This validation method also reveals differences between the actual robot and simulation, but requires a huge amount of manual testing and analysis. Because it only approximates the actual control algorithm and robot model, validation is an obvious concern, and significant differences have been found despite several iterations (e.g., [9,10] and [6]). In contrast to their work, we propose using the same control algorithm as on the real robot, to make configuration changes possible without having to repeat the iterative model optimization step. Furthermore, these validation methods are performed by a few experienced users, and are not intended to be executed frequently, changes to the robot or simulation engine remain undetected until the next test. While the practice of Continuous Integration [4] has been attempted also for robotics software development, the integration and frequent validation of a simulation environment in such an environment has, to the best of our knowledge, not been done so far. Regarding our goals for accuracy, it must be said that we are not only interested in the end state, but also in intermediate movements, so as to be able to validate the workings of the control algorithm. This can be a problem for general models, e.g., in [10, section 3.2, fig. 15], a large initial overshoot resulting from the interaction between control algorithm and the robot's inertia, is not replicated by the game engine-based model. Moreover, as [6] finds for other robot models on the same engine, there can be timing differences of several seconds between simulation and reality, and also small but significant differences in end state.

3 Concept Setup for Iterative Model Validation

"Continuous Integration is a software development practice where members of a team integrate their work frequently, usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible."— Martin Fowler[1]

While the concept of CI [2] is well established in common software development processes, e.g., in agile programming, CI practices have only received little attention in robotics so far. Mostly, a CI-enabled software integration cycle involves the following steps: A developer commits her/his recent changes to a source code repository. Shortly after the check-in a CI server registers the change and automatically performs a build of the new revision. Therefore, a CI server, or master, launches builds on multiple build slaves (not necessarily), e.g., on different Linux distributions including downstream builds. In each build, several tests are performed, based on test specific metrics. If the build has been successful, test results and artifacts such as, executable binaries, log files or documentation are transferred to the master and can be reviewed by the developer. If a build or test has not been successful, the server instantly notifies specified developers, at least the committer (e.g., via email), thus integration errors are quickly discovered and can be resolved. With respect to this paradigm we introduce to our concept setup, depicted in Fig. 1, for iterative testing and validation of simulated robot models including a real robot for comparison.

A CI server[2] provides a web front-end to create build jobs and inspect build results. A build job describes how a software build is performed, e.g, by executing CMake or shell scripts, and also tests if an application can be launched. In our setup, hierarchically structured build jobs (see Fig.2) are executed on a dedicated build slave. The build slave is connected to the master, as well as to the real robot. The simulation and a control server are steadily running on the slave. Moreover, three logging applications are installed on the build slave: a simulation logger, a control server logger and a video logger which can be triggered via build jobs. Last but not least, if started, a motion generation component sends movement

[1] http://martinfowler.com/articles/continuousIntegration.html

[2] http://jenkins-ci.org

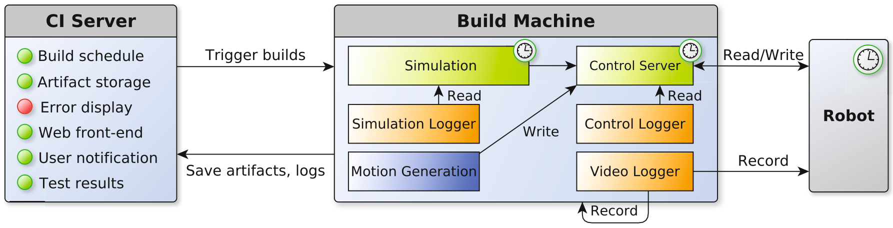


Fig. 1. Concept setup

commands to the control server which actuates the physical and virtual robot. Build jobs are created for each component running on the build slave as depicted in Fig. 2. In the following we will describe the build job hierarchy, and the function of each job.

3.1 Build Job Sequence

At the beginning of the build job sequence a series of checks are performed, such as are the simulation and control server still running. In case all checks finished successfully, four jobs are launched in parallel. The *Start Video Logger* job starts a recording software, therefore the simulation and the actual robot are recorded from an external point of view via web cam. If successfully finished the build artifact, a video file, is copied to the master node. Meanwhile, the *Start Motion Generation* job starts a motion replay application, thus movement commands are sent to the control server. Accordingly, the virtual and physical robot start moving. Both, the *Start Server Logger* and *Start Simulation Logger* job trigger logging applications. As soon as each logger receives a start event, they begin logging timestamped axis angle values for all actuated joints of the robot. The log files are also copied to the master and are archived. While build artifacts are only archived if a build has been successful, extensive build logs are saved for each job and run.

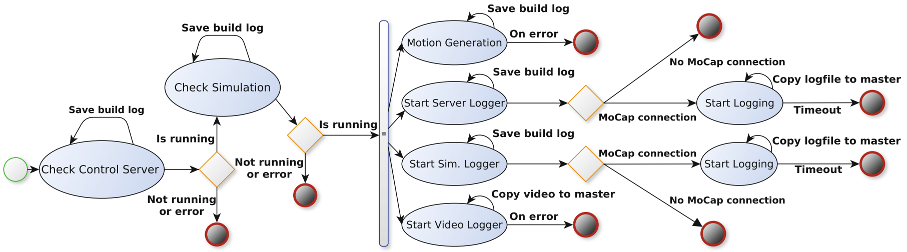


Fig. 2. Build job sequence

4 Concept Realization

In this section we will introduce our robot Flobi (Sec. 4.1), the implementation of its simulated correspondent, the applied simulation engine (Sec.4.2), and we will describe our shared control model (Sec. 4.3) for both, the actual, and the virtual robot. Our focus will be on the main requirements and design choices, e.g., accurate 3D modelling, a close software model, and data logging.

4.1 The Robot - Flobi

Flobi has been designed to provide an extensible head platform, which combines state-of-the-art sensing functionality with an exterior that elicits a sympathetic emotional response (see Fig.3). It features interchangeable external shells for the skin, eyelids and lips in order to study the influence of the outer appearance[5]. A total of 18 actuators allow individual, fine grained control of the neck, the eyes, eye-lids & -brows and the lips. The combination and simultaneous interaction of these actuators allow the expression of primary and secondary emotions which can facilitate and support communication situations in a human-robot interaction scenario.

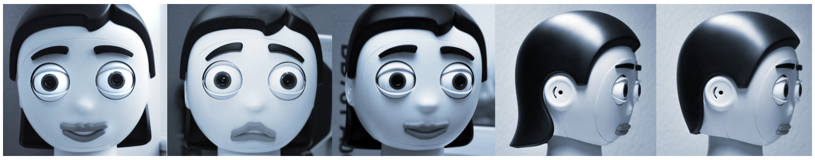


Fig. 3. The Bielefeld Anthropomorphic Robot Head "Flobi"

4.2 The Virtual Flobi - Modelling and Simulation Engine

Often, accurate modelling and rigging of a robot for 3D simulation, especially when its visual appearance is crucial (see Sec.4.1), is time consuming and requires experience in such techniques. Moreover, false or inconsistent dimensioning of robot parts lead to subsequent errors, e.g., when simulating collision avoidance. Hence, to provide an exact model of the robot, in terms of shape and proportions, it is beneficial to re-use existing technical drawings or ideally CAD files [8]. Since MORSE [3] is based on *Blender*[3], and therefore natively supports for import and also editing of diverse geometry definition file formats, we considered it as the most promising candidate (besides Webots™ and USARSim). For this reason we were able to re-use original Flobi CAD construction files to model an accurate virtual Flobi. Furthermore, we consider it particularly convenient that MORSE on the one hand, features a state-of-the-art simulation engine, and on the other hand also allows for the import, editing and creation of 3D objects

[3] http://www.blender.org/

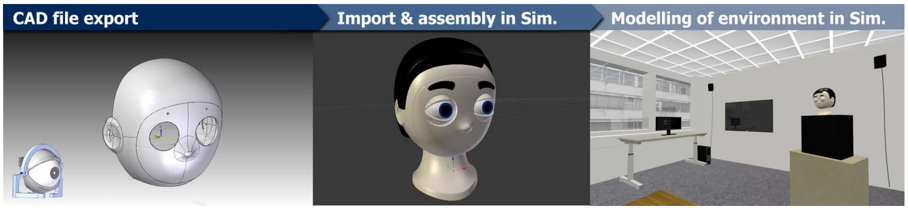


Fig. 4. Modelling pipeline of the virtual Flobi (without lips)

(and environments) in the same place, thus providing an efficient and consistent work flow (see Fig. 4). However, by this means we can provide an exact 3D model of our robot for the simulation environment, serving as a basis for further actuator and sensor implementations as introduced in the next section.

4.3 The Virtual Flobi - Control Model

In order to run and make use of a simulation, sensors and actuators have to be implemented for a virtual robot as, e.g., presented by Echeverria et al. [3]. We already mentioned false dimensioning as a possible risk in Sec. 4.2. We also perceive the implementation of virtual sensors and actuators as critical, such as wrong maximum speed/acceleration could work well in the simulation, but would damage the hardware, if applied on a physical robot. Additionally, as a basis for the integration into existing software systems and to expose sensor/actuator data to applications running outside the simulation (e.g., [8]), a simulation engine must provide middleware support. Since MORSE features abstract classes for both sensors and actuators, and also a middleware independent interface to expose this data, we were able to integrate the simulator into our existing software tool chain — in consideration of the issues mentioned above (see Fig. 5).

The virtual and physical Flobi can be controlled by using the same API implementing, e.g., *flobilib* and *flobixsc2 types*. Thus, from a developers point of view it is completely transparent whether the simulation or the real hardware is to be controlled.

To accomplish this functionality we have implemented a Flobi control server (XSC2) which acts as a gateway to physical or virtual Flobi motor control boards. The physical connection is established using a custom serial bus protocol, whereas for the virtual part, we wrap and execute the motor controller C code in a task based structure. For virtualization, the lower level PID speed control loop is assumed to be perfect and that any calculated target value is reached within one control cycle. There is no specific motor model taking motor power or friction into account. However, higher level control loops such as acceleration ramps and maximum velocities are taken into account as on the actual hardware.

The control server can be run in plain physical, pure virtual and even in a mixture of both modes (in contrast to, e.g., [11]). The latter is extensively being

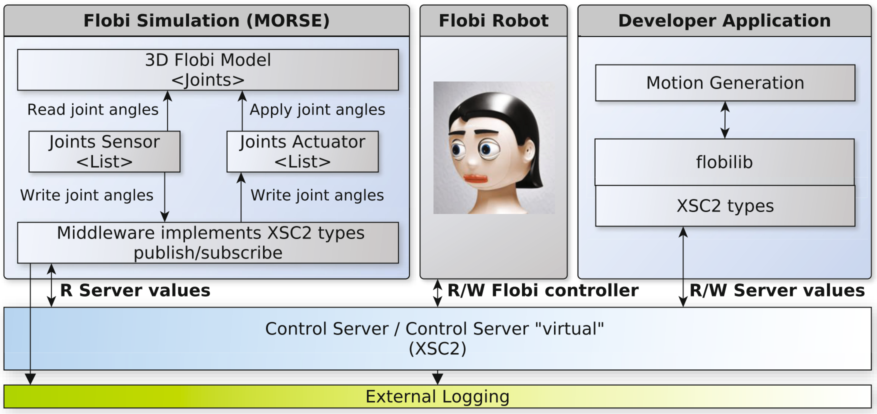


Fig. 5. Flobi control model

used during developing and testing of single parts by loading a configuration with only a new neck construction being physically connected to the server, while the residual motors are virtualized. Firstly, with this approach the simulation can be driven by either the real hardware, the virtualized hardware, or in mixed-mode, thus allowing for a flexible setup. Secondly, the virtual actuators/sensors are completely decoupled from any lower level communication by subscribing to sensor values such as angular position data in read only mode. Thirdly, we believe that this approach provides a close model of the simulation and target platform software, and therefore enables straight forward model validation as presented in Sec. 5.

4.4 Motion Generation and Data Logging

To detect differences between the model and the real robot, based on a realistic scenario, the nature of provided test data, and subsequently the precision of test results being logged are essential. Therefore, we have chosen a fixed, pre-recorded sequence of real human facial expressions[4] as a data provider for the motion generation component as depicted in Fig. 1. In our human-robot-interaction research, we are often interested not just in start and end-point precision, but also in the dynamics of the intermediate motion. Thus, the applied axis angles of i) the simulation and ii) the real robot are constantly logged over the complete duration of a test run. Axis angles in the simulation are logged with a frequency of 60Hz (native simulation cycle time), the robot axis angles are logged with 30Hz (due to current software limitations). Besides quantitative assessment, facilitated through the evaluation of log files archived in each build job (see Fig. 2), we are additionally recoding a video sequence of each test run for qualitative purposes.

[4] http://www.youtube.com/watch?v=PBs0c2LzMVM

5 Test Design and Case Study

For continuous comparison of simulated and real data, a repeatable real-world test setup is necessary. This test-setup must be sufficiently safe to be allowed to run unobserved, which rules out novel environments. Thus, a continuous integration test, even with real-world components, can not be a replacement for a full real trial. However, it should be sufficient to indicate that a real-world trial can be attempted given the current state of the software.

For safety, we propose to use a pre-generated safe movement which covers the relevant motion requirements in terms of positions and accelerations. In our case, we use motion-captured head movement data from real human head movements.

For indicativeness, we suggest that a test should exercise the same faculties of the robot, using the same control principles and the same configuration. For example, if the real scenario uses PID-based position control with some PID parameters, the test should use the same control mode and parameters. Some simulation setups may not be able to do this, but should attempt as close a match as possible. Nevertheless, test runs will never be completely the same. Therefore, we need to determine what can be considered a normal difference, and how to measure it. The procedure we used, and the differences we measured, will be described in the following case study.

5.1 Case Study Procedure

We assessed our robot model by i) measuring the overall joint angle drift of the actual and virtual robot, and ii) by deriving the offset between the actual and virtual robot within a set of test runs. To achieve this goal we ran the build job pipeline introduced in Section 3 (see Fig. 2) repeatedly for 12 hours. A complete run was automatically triggered every 15 minutes. In the following we will discuss the test results based on the archived log files in each run.

5.2 Results - Overall Axis Angle Drift

As a first assessment, we measured the overall axis angle drift in 49 test runs over 12 hours, as exemplarily depicted for the joint "Left Eye" in Fig. 6. There are two main results emphasize: firstly, the mean axis angle coordinates of the virtual and actual robot are closely aligned during the whole motion sequence (see Fig. 6, Virtual Mean, Physical Mean). Also, the standard deviation for both robot representations is almost similar (see Fig. 6, Virtual SD, Physical SD), but surprisingly high for "Left Eye" (3.73°) in contrast to, e.g, "Neck Roll' (0.45°, see Table 2). Apparently the standard deviation increases at high accelerations, e.g., in second 5,5 and 11, and reaches its maximum if the acceleration is abruptly changed as in second 15. To verify these assumptions we investigated the correlation between acceleration and standard deviation as exemplarily depicted in Fig. 7(a) for the "Left Eye" joint. Evidently, the standard deviation increases with the acceleration. While the majority of the plotted standard deviation values reside between 0° and 0.3°, as confirmed in Table 1, with an acceleration between

±1°/15ms. most of the outliers occur at high accelerations, e.g., at 4°/15 ms. However, the global maximum of 3.73° occurs at a relatively low acceleration of 0.95°/15m. As depicted in Fig. 6 (Max SD Virtual), the global maximum resides at an inflection point, hence the acceleration abruptly changed, causing the global maximum deviation. Overall, the measured standard deviation (mean) is conveniently low with a maximum of 0.31° and a minimum of 0.03°. Moreover, the virtual model and actual robot demonstrate almost the same behavior related to the axis angle drift as depicted in Table 2. The maximum difference between the virtual and actual robot amounts to 0.39° ("Left Eye"), which is within an acceptable range. Based on these results we obtained a first set of metrics, which can be used for further integration tests (see Table 2). As an example, a developer can utilze the setup presented in this study to verify her/his control code for each iteration, respectively for each commit, of his software by writing unit tests for instance. The unit tests may assert joint angle values at a given time, based on the standard deviation values presented in this section. It is up to the developer how strictly she/he tests the control code. In case of a pedantic test she/he may assert the target value of "Neck Roll" with an allowed deviation of 0.01°.

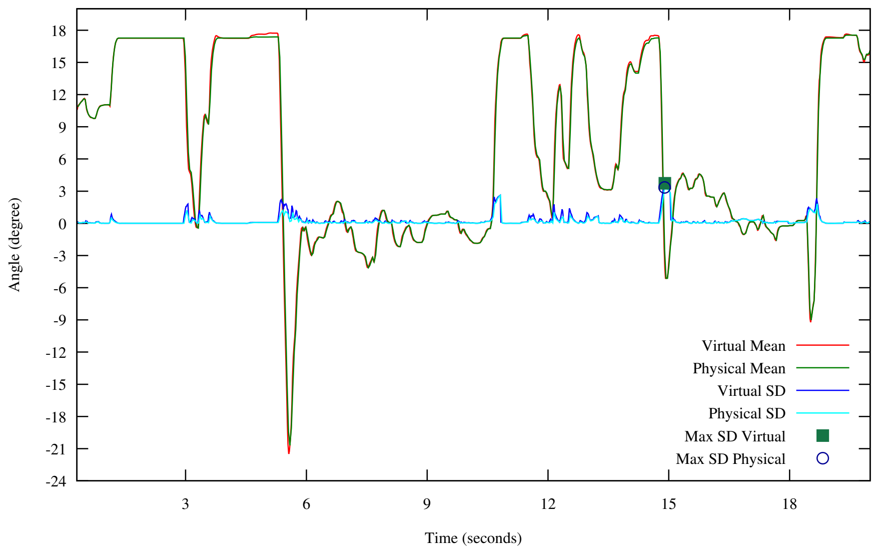


Fig. 6. Axis angle drift for joint "Left Eye", direction "left to right"

5.3 Results - Offset between the Virtual and Physical Robot

In the previous section we have shown that the simulated robot model and the actual robot behave the same way in terms of axis angle deviations. We also mentioned that the mean axis angle coordinates are closely aligned during the

Table 1. Physical and virtual (v) robot: max/min/mean joint angle standard deviation (σ) over 12 hours

Joints	Max. σ (v)	Min. σ (v)	Mean σ (v)	Max. σ	Min. σ	Mean σ
Neck Roll	0.45°	0.01°	0.05°	0.45°	0.01°	0.03°
Neck Tilt	0.48°	0.01°	0.05°	0.43°	0.01°	0.04°
Neck Pan	1.71°	0.01°	0.11°	1.80°	0.01°	0.10°
Left Eye LR	**3.73°**	**0.01°**	**0.25°**	**3.34°**	**0.01°**	**0.18°**
Right Eye LR	2.15°	0.01°	0.17°	2.02°	0.01°	0.12°
Left Eyelid Upper	3.41°	0.01°	0.30°	3.03°	0.01°	0.27°
Right Eyelid Upper	3.57°	0.08°	0.31°	3.79°	0.09°	0.30°
Both Eyes UD	1.80°	0.01°	0.12°	1.79°	0.01°	0.09°

whole motion sequence (see. Fig. 6, Virtual Mean, Physical Mean). If a developer needs to test his software in the simulation, because she/he doesn't have access to the physical robot for instance, it is also important to assure that the actual angle offset between the simulation and the actual robot is sufficiently small to obtain realistic test results. Therefore, we have measured the overall offset between both representations as shown in Table 2.

First of all, it is noticeable that the absolute mean offset between the virtual and the actual robot is acceptably small with a maximum of 0.99°. Secondly, similar to the standard deviation as presented in Table 1, the highest total offset was measured at joint "Left Eye". Subsequently, we investigated the offset distribution as shown in Fig. 7(b). Not surprisingly, maximum offsets also reside at high accelerations. Nevertheless, the findings of this section can be used as additional metrics when testing algorithms in the simulation. As an example, a developer can be assured that a scenario, that he as tested in the simulation, will perform nearly the same way on the actual robot by implying the given offsets.

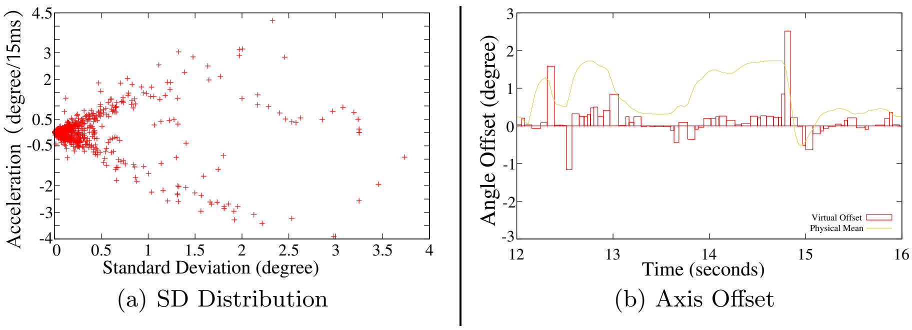


Fig. 7. Distribution of the standard deviation and axis angle offset

Table 2. Axis angle offset between virtual and physical robot

Joints (virtual)	Max. Positive Offset	Max Negative Offset	Abs. Mean Offset
Neck Roll	0.37	-0.44	0.04
Neck Tilt	0.45	-0.34	0.05
Neck Pan	2.40	-1.91	0.99
Left Eye LR	2.51	-1.16	0.13
Right Eye LR	0.87	-1.02	0.12
Left Eyelid Upper	1.73	-2.39	0.94
Right Eyelid Upper	1.48	-1.43	0.55
Both Eyes UD	1.31	-1.78	0.08

5.4 Results - Visual Comparison

With each test run we have recorded a video file showing the behavior of the simulation and the real robot. As presented in the last section, the axis angle differences are acceptably small. Therefore, we could not determine any difference based on the qualitative evaluation of all video logs. Nevertheless, such an evaluation mechanisms could be used by developers when assessing experimental control code, as the interpretation of a video file is more intuitive than the interpretation of "raw" axis angle data.

6 Conclusion

We have investigated continuous validation of robot models, to verify a sufficient match between the simulation and reality. To tackle this issue, we presented a continuous integration approach supporting iterative model validation (Section 3). In an experimental case study we have shown that the fidelity, in terms of end-positions and also the dynamics of intermediate motion, can be validated by measuring deviation and offset between axis angles of the real and the simulated model. The results indicate that a close software model leads to acceptable differences (see Sec. 5.2, Sec. 5.3). Based on the results of the case study, we could confirm that our simulated model basically matches the reality. Further changes to the simulation, or the actual robot can be automatically detected based on these findings. Our approach to continous robot model validation is based on a middle ground between testing isolated parts (which is usually safe but cannot detect interaction effects) and evaluating full scenarios (which would be too costly). For static robots, as in our current study, this has been fairly easy to define. An interesting target for future work would be to extend this to mobile robots, for which a suitable test setup would have to be defined, which exercises the mobility, without requiring a full test course. However, the metrics applied in the case study could be used by developers to verify their control code in an integrated and frequent manner, utilizing our system setup. Nevertheless, the measured divergence is higher than desired. We believe that this effect is caused by the applied logging frequency, therefore we will conduct yet another study with higher logging frequencies such as 100Hz.

Acknowledgments. This work has been partially supported by the German Aerospace Center (DLR) with funds from the Federal Ministry of Economics and Technology (BMBF) due to resolution 50RA1023 of the German Bundestag.

References

1. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: a robot simulator for research and education. In: Proceedings 2007 IEEE International Conference on Robotics and Automation, pp. 1400–1405. IEEE (April 2007)
2. Duvall, P., Steve, M., Glover, A.: Continuous integration: improving software quality and reducing risk, 1st edn. Addison-Wesley Professional (2007)
3. Echeverria, G., Lassabe, N., Degroote, A., Lemaignan, S.: Modular openrobots simulation engine: Morse. In: Proceedings of the IEEE ICRA (2011)
4. Holck, J., Jørgensen, N.: Continuous integration and quality assurance: a case study of two open source projects. Australasian Journal of Information Systems 11(1) (2007)
5. Lütkebohle, I., Hegel, F., Schulz, S., Hackel, M., Wrede, B., Wachsmuth, S., Sagerer, G.: The bielefeld anthropomorphic robot head flobi. In: 2010 IEEE International Conference on Robotics and Automation (ICRA), pp. 3384–3391 (May 2010)
6. Okamoto, S., Kurose, K., Saga, S., Ohno, K., Tadokoro, S.: Validation of Simulated Robots with Realistically Modeled Dimensions and Mass in USARSim. In: 2008 IEEE International Workshop on Safety, Security and Rescue Robotics, pp. 77–82. IEEE (October 2008)
7. Pepper, C., Balakirsky, S., Scrapper, C.: Robot simulation physics validation. In: Proceedings of the 2007 Workshop on Performance Metrics for Intelligent Systems - PerMIS 2007, pp. 97–104. ACM Press, New York (2007)
8. Roalter, L., Möller, A., Diewald, S., Kranz, M.: Developing intelligent environments. In: Seventh International Conference on Intelligent Environments (2011)
9. Taylor, B.K., Balakirsky, S., Messina, E., Quinn, R.D.: Design and validation of a Whegs robot in USARSim. In: 2007 Workshop on Performance Metrics for Intelligent Systems - PerMIS 2007, pp. 105–112. ACM Press, New York (2007)
10. Taylor, B.K., Balakirsky, S., Messina, E., Quinn, R.D.: Modeling, validation and analysis of a Whegs robot in the USARSim environment. In: Proceedings of SPIE, vol. 6962, pp. 69621B–69621B–12 (2008)
11. Tikhanoff, V., Cangelosi, A., Fitzpatrick, P., Metta, G., Natale, L., Nori, F.: An open-source simulator for cognitive robotics research: the prototype of the icub humanoid robot simulator. In: Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems, PerMIS 2008, pp. 57–61. ACM (2008)

Software Abstractions for Simulation and Control of a Continuum Robot

Arne Nordmann, Matthias Rolf, and Sebastian Wrede

Research Institute for Cognition and Robotics, Bielefeld University, Germany

Abstract. The Bionic Handling Assistant is a new continuum robot which is manufactured in a rapid-prototyping procedure out of elastic polyamide. Its mechanical flexibility and low weight provide an enormous potential for physical human robot interaction. Yet, the elasticity and parallel continuum actuation design challenge standard approaches to deal with a robot from a control, simulation, and software modeling perspective. We investigate how the software abstractions of the existing Robot Control Interface (RCI) and the Compliant Control Architecture (CCA) can deal with this platform from a software modeling and software architectural perspective. We focus on three different challenges: the first challenge is to enable reasonable and hierarchical *semantic abstractions* of the robot. The second challenge is to develop *hardware I/O abstractions* for the prototypical and heterogeneous technical setup. The third challenge is to realize this in a flexible and reusable manner. We evaluate our approaches to the above challenges in a practical scenario in which the robot is controlled either in simulation or on the real robot.

1 Introduction

Continuum robotic systems inspired by biological actuators like elephant trunks [1], octopus arms [2], or even squid tentacles [3] have gathered increasing interest in the last decade of robotics research. These systems move without traditional revolute or prismatic joints, but are based on continuous deformations in shape, and are typically driven by parallel hydraulic or pneumatic actuators. The focus of this paper is the Bionic Handling Assistant (BHA) [4] which is a new continuum platform inspired by elephant trunks and manufactured by *Festo* (see Fig. 1). The robot is pneumatically actuated and made almost completely out of polyamide which makes it very flexible and lightweight. The robot comprises three main segments, each with three parallel, pneumatic bellow actuators, a ball-joint as wrist, also actuated by three actuators, and a three finger gripper actuated by one bellow actuator. When the bellow actuators are supplied with pressure, they extend their length and can cause arc-like deformations as well as elongations.

The continuous arc-like deformations are not only challenging from a simulation and control point of view. On a semantic level, the multi-segment parallel actuation demands for software modeling approaches that allow an intrinsically hierarchical view on a robot. This is not the case for standard revolute joint robots which are appropriately modeled by a series of single-actuator abstractions. On a hardware level, the prototypical robot setup provides very heterogeneous I/O channels, including pressure sensing and control via a CAN bus, length sensing via an analog-digital converter PCI card, as

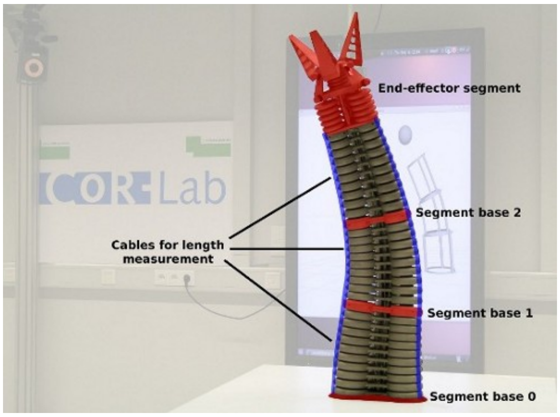


Fig. 1. The kinematic structure of the BHA comprises three main segments, each consisting of three parallel pneumatic bellow actuators. The length of these actuators can be determined with cable-potentiometers.

well as position sensing with Vicon [5] motion tracking system that communicates with a proprietary network protocol.

This paper investigates the use of the software concepts and abstractions of the existing software frameworks Robot Control Interface (RCI) and the Compliant Control Architecture (CCA) to master these challenges in a *flexible and reusable* manner. Our contribution is extracting software requirements imposed by the above challenges of continuum robots, elaborated in Section 2. We then discuss these requirements along implementation of a length control use-case with RCI and CCA. Section 3 introduces RCI and shows how we mapped the BHA to its semantic abstractions. In Section 4 we highlight certain design decisions of the technology mapping and Section 5 shows a practical use-case, introducing our open-source BHA simulator as well as length control on the real robot. In Section 6 we conclude and point out the key aspects we identified to cope with the challenges of continuum robots like Festo‘s Bionic Handling Assistant.

2 Challenges

With all its advantages and properties desirable for physical Human-Robot Interaction (*pHRI*), the continuum kinematics of the BHA impose a number of challenges both on a semantical and a technological level. On the lowest level, the robot is driven by 13 bellow actuators. Each of them allows to control and sense pressure. Although there are physical interactions between these actuators they can be seen as independent and conceptually identical from a software point of view – similar to different revolute joints on standard robots. The situation changes when postural sensing and control is considered. The pressure in the actuators is not a reliable means to define a robot posture, i.e. a geometric shape of the robot, since it only describes a force. The BHA comprises length sensors for all actuators except the gripper to circumvent this problem and provide geometric information. On the main segments these sensors are on the outside of each actuator. The length sensors of the wrist do not provide a "one to one" relation to the actuators since they are mounted in between two actuators. Although the main segments do have a "one to one" relation, the length of a single actuator is essentially meaningless: the parallel actuation design of three actuators in one segment only defines a shape when all three lengths come together. Hence, our approach is to use an entire segment as the elementary *semantic abstraction* that must be considered to describe the geometric configuration of the robot, which contradicts common ways of semantic and technical modeling of robotics platforms that focus on single actuator abstractions [6]. A software model must allow to work on top of this semantic abstraction as well as inside the segment abstraction in a hierarchical fashion: controlling the length is not done with standard PID control concepts and, in fact, the robot comes without a length controller. Solving the length control is active research and requires a software model that allows to plug-in control functionality into a segment abstraction that already provides sensing capabilities. Also the cartesian control of the end-effector is not available as standard method or component, although sensing is directly available.

A second challenge arises from the interfacing of the actual hardware I/O capabilities of the robot setup. The very heterogeneous hardware setup involves a number of different communication patterns, protocols and transports necessary in order to access and integrate all technical components of the robot. The different hardware interfaces use different data representations, have different timing and timing constraints, as well as completely different data volumes to process. The pressure control and sensing is interfaced with a custom binary protocol over a CAN-bus which is accessed with a Linux SocketCAN driver. This interface needs to be operated in (soft) real-time since it represents the instantaneous actuation of robot. The current measurements, commands, and possible error- or status messages need to be updated with a request-reply pattern at a rate of $50\,Hz$. Thereby the 13 actuators are controlled by two valve-units, which are connected to the CAN-bus as separate devices. Each valve-unit has eight valves, whereas the actuators of the lowest segment are each connected to two separate valves, both of which need to be actively controlled. The length measurement is interfaced with a proprietary driver accessing an analog-digital converter PCI card. The device allows for an almost instantaneous reading of the current lengths. Finally, the cartesian position of the end-effector can be sensed, but not controlled with a Vicon [5] motion tracking system. This system communicates with $200\,Hz$ via a proprietary network protocol. These heterogeneous channels need to be synchronized and leveraged in a coherent software framework which demands for *hardware I/O abstractions* that can capture the diversity of these devices.

The development of *flexible and reusable* abstractions for such components is a natural requirement for a robotics framework, although in particular the hierarchical modeling on a semantic level is not trivially achieved. The third challenge in our setup, however, goes beyond the reuse of segment- or whole-robot-abstractions: Current research on this very prototypical robot setup is mostly concerned with the development of controllers that mediate between high-level semantic abstractions and low-level hardware abstractions. This demands a high degree of flexibility and extendability even between reusable abstractions on several levels. Abstractions that only allow sensing in the first place must be extendable by control-semantics.

The use case investigated in this paper is the control of actuator lengths. Length control is fundamental to any other application with the robot, but requires a solution to all of the above challenges: It has to

1. rely on hierarchical abstractions of robot segments instead of single actuators.
2. incorporate heterogeneous sensor information and feed commands back in realtime.
3. integrate in a flexible, additive manner into other reusable abstractions.

Thereby the BHA particularly challenges the classical conceptualizations of control interfaces, often used in robotics frameworks. Length control on the BHA for example can not be expressed in the classical length control interface, providing length measurement and its control at the same place. Whereas length is *measured* in the nine chambers, *controlling* of the length only makes sense in the context of an entire segment due to the strong coupling inside a segment.

3 Software Abstractions and Programming Model

In order to access to the platform-specific features of a robot platform in a generic and coherent way, we developed a framework of software abstractions for compliant robots, called Robot Control Interface (RCI) [7]. RCI provides a set of domain-specific abstractions to represent common features of compliant robotics systems. The domain-specific abstractions of RCI are imposed through the domain of motion learning on soft and compliant robots, whereas a *domain* is a *"set of current and future applications which share a set of common capabilities and data"* [8]. The representations are available as part of the Robot Control Interface and comprise software solutions as well as interfaces, features and domain objects identified in a domain analysis process. To be able to describe the domain in a condensed manner, we performed an extensive feature-oriented domain analysis (FODA [8]), including currently existing applications, robot-features, and software frameworks in the domain. Examples of applications and frameworks included in the domain analysis are: The *KUKA Fast Research Interface* [9], the *Orocos-RSI* extensions of the Orocos Framework [10], the RobotCub software [11], especially the recently developed *iDyn* library, as well as the two ROS stacks *force-torque* by the Healthcare Robotics Lab at Georgia Tech [12] and *COB force-torque* for the the Care-O-bot platform [13]. In addition to analyzing features of frameworks and interfaces, we analyzed interfaces of compliant robot platforms like *COMAN*, the compliant successor of *iCub* and the compliant quadruped robot *Oncilla*.

The unifying Robot Control Interface (RCI) follows a *model-driven approach* and abstracts from a concrete hardware platform while allowing to integrate or generate the necessary platform-specific code. The description of the logical architecture and API views yields a first definition of the interfaces between the robot platform (simulated or hardware) and the user application. RCI focuses on the software interface for *proprioceptive sensors and actuators*, which includes velocities, accelerations and forces applied to a robot as well as sensorics for equilibrium or balance (accelerometer, gyroscope). This API defines a low-level robot programming interface specifically considering the requirements of compliant actuator control and proprioceptive sensing.

A first observation as result of the domain analysis is that in this domain a clear distinction of physical robot parts between either sensor or actuator is often infeasible and sometimes even cannot be made. Instead, actuator and sensor parts share a comprehensive set of controlling and sensing features (i.e. sensing of position, force). This is valid for example for almost every actuator with position control, being able to sense at least the current encoder value, often also the motor current et cetera. In the domain of compliant robots, e.g. including active compliant actuators, a single actuator often provides a rich subset of features from actuators and sensors, including measurement and control of force, torque and joint position.

Based on this observation RCI defines a set of possible features of physical robot parts, both controlling and sensing features. We then define the *ResourceNode* as a logical abstraction for sensors *and* actuators, which can have an arbitrary set of these features. A ResourceNode with just sensing features represents a pure sensing robot part, a ResourceNode with controlling and optionally sensing features represents an actuated robot parts. RCI defines a number of fine-grained interfaces of semantic data abstraction (e.g. joint angles, torques, forces, et cetera) and a separate control and sensing interface for each. A core aspect of this fine-grained assignment of features to robot parts is, that the control and sensing aspect of the same variable (e.g. joint angle) can even be split to several robot parts.

A second essential concept in RCI is the concept of a *Synchronizer* that solves the connection of the robot‘s software abstractions (ResourceNodes) with the actual robot – in simulation or hardware. The synchronizer reads commands sent to the ResourceNodes and passes them as commands to the robot. It also reads sensor values from the robot and passes them to the Resource Nodes.

The Robot Control Interface provides *platform-specific* software *abstractions* for modeling of the robot platform, in the form of the ResourceNode abstraction for sensors and actuators as well as a set of fine-grained controlling and sensing features. It also provides the *implementation-specific* abstraction of hardware I/O interfaces in the form of Synchronizers. This set of abstractions provides the *Programming Model* for implementations of a robot interface with RCI.

3.1 Modeling of the Bionic Handling Assistant

In order to check whether the Programming Model provided by the Robot Control Interface is capable of meeting the challenges depicted in Section 2, we modeled the BHA with RCI abstractions. We evaluate whether the concept of RCI ResourceNodes can provide reasonable *semantic abstractions* of the continuum kinematics and therefore provide a solution to the semantic challenges. Furthermore we evaluate whether the RCI concept of a Synchronizer is able to cope with the heterogeneous hardware interface setup of the BHA and provides meaningful *hardware I/O abstractions*.

Note, that for the sake of clarity in this example we focus on the first nine, main bellow actuators without wrist and gripper actuation.

Modeling of the platform-specific parts of the robot system, the semantic abstractions, is done in three kinds of different resource nodes: i) chambers, ii) segments and iii) the end-effector.

– A **chamber node** represents a bellow actuation unit of the robot, equipped with length sensing. Interfaces of this node are therefore `PressureSensing`, `PressureControlled` and `LengthSensing`.

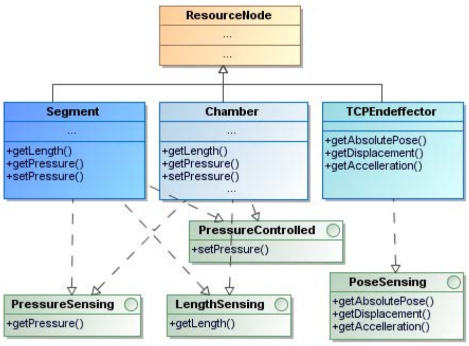


Fig. 2. RCI ResourceNodes for the BHA's three main segments, its chambers and the end-effector

– A **segment node** does not add any functionality at this point, but repeats the length sensed values and the pressures of its three chambers. The segments provide the three chamber length values in a semantically coherent way in order to provide a basis for later-on extensions with control capabilities.
– The **end-effector node** is the gripper, which cartesian position is sensed. Although the position is sensed by an external component, we model it as part of the robot system, since in our context (cf. use-case in Section 5) this is a relevant part of making the robot system usable.

The implementation-specific part of the robot system is modeled as set of synchronizers that have to deal with the diverse hardware interfaces for pressure control and sensing, length sensing and the Vicon system.

1. The **PressureSynchronizer** connects over the CAN bus to receive pressure values, writes them to the chamber nodes. It reads pressure commands from the chamber resource nodes and sends them via CAN to the responsible valves-unit.
2. The **LengthSynchronizer** accesses the driver of the analog-digital converter PCI card for reading values of the cable potentiometers at the outside of the bellow actuators, and writes them to the chamber resource nodes.
3. The **ViconSynchronizer** runs on a different workstation. It connects to the Vicon motion tracking server and reads the current end-effector position which are set as current sensor value in the end-effector node.

4 Technology Mapping

The following section highlights our design decisions for the technology mapping that we use in order to prove the above abstractions are suitable for applications on the BHA. The technology mapping includes the implementation of the abstractions in the Robot Control Interface, a component framework for extending the interface, applications running on the robot, as well as the middleware for integration and access to the hardware interfaces.

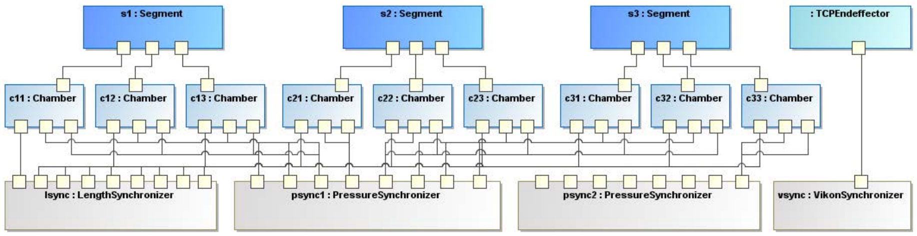


Fig. 3. Organization of BHA Resource Node instances (blue) and Synchronizer instances (gray). Note that there are two pressure synchronizers, one for each of the valve units. The first eight chambers connect to the first valve unit, the ninth chamber is connected to the second valve unit.

4.1 Robot Control Interface

Robot Control Interface is available as C++ library `librci`, providing interfaces and base implementations for the abstractions introduced in Section 3. The library includes a set of domain-specific base interfaces (various `...Controlled` and `...Sensing` interfaces, as well as generic implementations of its setters and getters. Features that allow to apply commands to a node, are expressed in the `...Controlled` interfaces, allowing to set a reference for a certain controller (position, length, pressure, etc.). The set of `Controlled` interfaces implemented collectively by all nodes of a robot defines a set of tasks that can be executed by the robot. Features that allow getting status information from a node, are expressed in the `...Sensing` interfaces. The set of `Sensing` interfaces implemented by nodes of a robot defines a set of status data that can be reported by the robot.

Additionally the RCI library provides a collection of domain-types serving as data-holders with domain-specific manipulation methods, setters and getters. The library will soon be open-source as part of the European AMARSi project.

4.2 Component Architecture

For implementation of functional components and integrating RCI entities into the applications, we leverage the Compliant Control Architecture (CCA). CCA is an event-based, middleware-agnostic component architecture for robotics research, focusing on (real-time) control of compliant hardware and enabling machine learning. The C++ library `libcca` serves as a technology mapping for platforms modeled in RCI and as component architecture for implementing user applications and the adaption of hardware and simulation interface as later-on described in Section 5.2. An application implemented using CCA is a graph of loosely coupled components which extend the CCA base component. CCA components possess ports to exchange data between each other via data-flow. Data in this data-flow graph is represented by domain-types, which specify the representation of data and provide domain-specific access methods. Data-flow in CCA can express acyclic as well as cyclic messaging graphs.

For management and coordinated processing of rich component graphs, CCA provides the concept of *ProcessingStrategies*. These strategies allow for a loose, input-driven coupling of nodes, but also for control-loops with tight timing constraints. CCA comes with an extendable set of basic strategies, like *TimedProcessing*, which processes a component based on a fixed timing, and *PortTriggered*, which processes a component triggered by incoming data at a specific input port. Components using timing-based processing strategies can also be executed in a real-time context, based on the Xenomai Real-Time API. By combining collections of real-time-based components, entire subgraphs can be executed in real-time context. To ease the construction of subgraphs, CCA provides a set of static connectivity skeletons (often called *algorithmic skeletons* in parallel computing literature), like pipelines, splitters and collectors. A pipeline is a set of components with input-driven processing, which processes the entire data as fast as possible through the sequence of connected components. A splitter takes a multi-dimensional data-items on its input port and splits it into sub-elements. A collector is collecting and merging incoming low-dimensional data-items to an outgoing multi-dimensional data item.

4.3 Middleware and Integration

In order to realize the data-flow between CCA components we facilitate the open-source robotics middleware *Robotics Service Bus* (RSB [14]). RSB is a lightweight, event-based, and highly customizable robotics middleware with native implementations in C++, Java, Python and Common Lisp.

Robotics applications often involve computationally complex modules, especially applications involving machine learning or complex kinematics modules (e.g. inverse kinematics for multiple limbs, continuum kinematics). RSB offers local and remote transports, that can transparently be switched during runtime, allowing CCA components to be distributed at runtime over several machines, for example the robot itself, workstations, as well as computation clusters. CCA components on the same machine communicate in a fast local inprocess transport, and only communicate with a network stack if they are distributed over different other workstations. Since type of communication (remote, local) is configured at runtime, a CCA component graph can be deployed in an arbitrary configuration across different computing devices.

In this sense RSB also provides an answer to the technical challenges. All involved hardware interfaces are connected to the system and therefore available for all software components through the middleware. End-effector positions are streamed from the VICON control PC to the functional components, pressure and length values are bridged to the CAN bus and IO card respectively.

5 Use Case

In this section we validate the above concepts, that were already proved working on robot platforms like iCub, Oncilla and KUKA LWR IV, on the continuum robot platform BHA. This provides a first measurement of the conformance of our suggested API and architecture to compliant robotics and successfully integrated our concepts on state-of-the-art hardware and simulated robots from the rigid body domain.

A first use-case for working with the BHA is length control. For a reliable positioning, it is not sufficient to control the pressure alone: Friction, hysteresis and non-stationarities can cause largely different postures on the BHA when supplying the same pressure several times. In particular during dynamic movements the pressure is not sufficient to determine the posture or position of the robot, since it only expresses a force on the actuators. Length control is therefore a central skill for any other task to be achieved on the robot.

5.1 Length Control on the Real Platform

A naive approach to the control of the actuator lengths would be to consider each actuator in isolation, and adjust the pressure with a standard PID feedback controller until the desired length is achieved. From a pure control perspective, this approach has two severe disadvantages:

1. Feedback control can only be done with very low gains on a pneumatic robot due to long temporal delays in the actuation chain, which corresponds to slow motion.
2. It does not consider the strong mechanical interplay of actuators in the same segment, which largely influences the pressure necessary to achieve a certain length.

In order to deal with both problems at the same time we use machine learning methods and estimate models that allow a feedforward control of an entire segment. A length command, generated by some application, is smoothed by a generic filtering mechanism. This three-dimensional length command is fed both to the learned model and a generic PID controller, which receives Kalman-filtered feedback about the current actual lengths. Both controllers output a pressure signal. These signals are added, filtered, and supplied to the segments as new target pressure.

This length control scenario is realized with a set of – generic as well as rather platform- and application-specific – CCA components as illustrated in Figure 4. The chamber nodes are domain-specific implementations for the BHA, while PID-controllers and various filters are generic components that are reused from other domains. Pressure and length data passed between the components are the domain-types defined by the Robot Control Interface. The inverse model is a clear platform- and application-specific component. The integration of this length control scheme uses the two essential properties of our modeling approach:

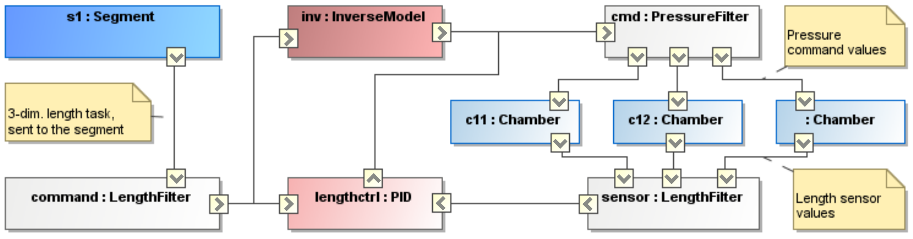


Fig. 4. Length control of the Bionic Handling Assistant. For the sake of clarity illustrated for the first of the three segments only. Resource nodes (segment, chambers) in blue, filters (length, pressure) in grey, and control components in red.

1. It uses the hierarchical modeling of segments and chambers and thereby utilizes the semantical grouping of sensory and command values in a segment abstraction.
2. It adds a `LengthControlled` interface to the segment, which was previously only a `LengthSensing` abstraction and an aggregator for the chamber values. This interface allows to set three-dimensional target values for the lengths.

Thereby the very fine-grained architectural design facilitates the reuse on component level which we extensively exploit by using filtering and feedback-control implementations that were developed in other scenarios.

5.2 Common Interface for Simulation and Hardware

The hierarchical aggregation of lengths into a segment abstraction is even more important when pure simulation is considered as alternative to the real robot scenario. We use an open source implementation [15] of a continuum kinematics model in order to model the kinematic structure of the BHA. This model assumes that bending and stretching movements of each robot segment behave like a torus section (see Fig. 5) which allows to infer the coordinate transformations for the forward kinematics. The model allows to predict the end-effector position of the BHA based on the actuator lengths with an accuracy of $1\,\%$ relative to the robot size [16].

The control of lengths in this scenario is trivial, because they can directly be "set". A substantial difference to the real-robot scenario is that single actuators do not exist. From a mathematical modeling point of view a single actuator is not a valid concept since it does not allow to determine any component of the six-dimensional pose-transformation involved in one segment. The entire simulation operates on aggregates of three lengths.

This simulation scenario can directly be described with the existing Robot Control Interface infrastructure and the abstractions developed for the real-robot as presented in the previous sections. The segment abstractions are entirely reused from the real-robot implementation, chambers and controllers are simply left out. The only difference in implementation is that a new `Synchronizer` is needed in order to connect segments

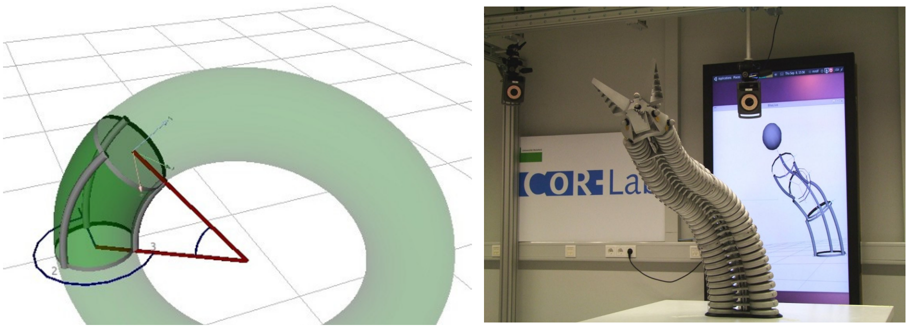


Fig. 5. Torus model to simulate BHA's kinematic structure (left) and on the basis the 3D visualization of the BHA (right)

and simulation software. Since the segment abstractions entirely decouple the backend-implementation (`Synchronizers`) from the application code, it is directly possible for an application to switch between real-robot and simulation, which is even possible during runtime.

6 Conclusion

In this paper we discussed the software challenges imposed by continuum robots, based on our exemplary platform, Festo‘s Bionic Handling Assistant. An analytical approach to find fine-grained software abstractions based on a feature-oriented domain analysis and a resulting programming model for robot interfaces were introduced. In the course of the paper we discussed these software abstractions and our technology mapping, the Robot Control Interface and the Compliant Control Architecture, along the practical use-case of length control on the real robot platform and simulation of the Bionic Handling Assistant. The contribution of this paper, the software abstractions, especially the clear separation of control and sensing aspects, and the hierarchical modeling of segments and chambers, showed to be helpful and necessary to master these challenges in a *flexible and reusable* manner.

The length-control use-case presented in this paper is fundamental to any application on the robot and has already served as basis for learning of reaching skills [17]. Similar to length-control, this learning adds a `Controllable` concept to a previously only sensible effector position, and has already been exploited in several applications.

Acknowledgements. The research leading to these results has received funding from the European Community's Seventh Framework Programme FP7/2007-2013 – Challenge 2 – Cognitive Systems, Interaction, Robotics – under grant agreement No 248311 - AMARSi.

References

1. Hannan, M.W., Walker, I.D.: Kinematics and the Implementation of an Elephant's Trunk Manipulator and Other Continuum Style Robots. Journal of Robotic Systems 20(2), 45–63 (2003)
2. Laschi, C., Mazzolai, B., Mattoli, V., Cianchetti, M., Dario, P.: Design of a Biomimetic Robotic Octopus Arm. Bioinspiration & Biomimetics 4(1) (2009)
3. Wilson, J.F., Li, D., Chen, Z., George, R.T.: Flexible Robot Manipulators and Grippers: Relatives of Elephant Trunks and Squid Tentacles. In: Robots and Biological Systems: Towards a New Bionics. NATO ASI, vol. 102, pp. 475–494 (1993)
4. Grzesiak, A., Becker, R., Verl, A.: The Bionic Handling Assistant: A Success Story of Additive Manufacturing. Assembly Automation 31(4), 329–333 (2011)
5. VICON. Motion Tracking Systems, `http://www.vicon.com`
6. Branson, D., Kang, R., Guglielmino, E., Caldwell, D.G.: Control Architecture for Robots with Continuum Arms Inspired by Octopus vulgaris Neurophysiology. In: International Conference on Robotics and Automation, pp. 5283–5288 (2012)

7. Nordmann, A., Wrede, S., Tsagarakis, N., Tuleu, A.: Software Interface for Proprioceptive Sensors and Actuators. Technical report, AMARSi (2010), http://www.amarsi-project.eu/system/files/AMARSI-D.7.1.pdf
8. Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: FODA: Feature-Oriented Domain Analysis (1990)
9. KUKA. Fast Research Interface - Preliminary Documentation. Technical report (2010)
10. The Orocos Project. Open Robot Control Software, http://www.orocos.org/
11. Fitzpatrick, P., Metta, G., Natale, L.: Towards Long-lived Robot Genes. Robotics and Autonomous Systems 56(1), 29–45 (2008)
12. Healthcare Robotics Lab Georgia Tech. force-torque Package, http://www.ros.org/wiki/force_torque
13. Bubeck, A.: Care-O-bot force-torque Package, http://www.ros.org/wiki/cob_forcetorque
14. Wienke, J., Wrede, S.: A Middleware for Collaborative Research in Experimental Robotics. In: International Symposium on System Integration, Kyoto, pp. 1183–1190 (2011)
15. Research Institute for Cognition and Robotics. Software: Continuum Kinematics Simulation, https://www.cor-lab.org/software-continuum-kinematics-simulation
16. Rolf, M., Steil, J.J.: Constant Curvature Continuum Kinematics as Fast Approximate Model for the Bionic Handling Assistant. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (2012)
17. Rolf, M., Steil, J.J.: Efficient Exploratory Learning of Inverse Kinematics on a Bionic Elephant Trunk. IEEE Transactions on Neural Networks and Learning Systems (submitted, 2012)

A Visual Modeling Language for RDIS and ROS Nodes Using AToM^3

Paul Kilgo, Eugene Syriani, and Monica Anderson

Computer Science Department, The University of Alabama, Tuscaloosa, AL 35487

Abstract. In robotics we are often faced with the problem of repeatedly writing robot drivers for the same devices, but different robot frameworks. In an effort to counter this, a domain specific language for generating robot drivers was developed. However, descriptions tend to get verbose fast and the adopted syntax was difficult for programmers. This paper outlines an attempt to shift away from a textual syntax and toward a visual syntax, instead relying on the textual syntax to communicate the model to other tools. In addition, a formalism for defining ROS nodes is presented and a model transformation for mapping RDIS messages to ROS messages and vice-versa is created.

1 Introduction

A common workflow for robotic software development is to write robot applications for specific robot frameworks, which handle the conversion of the messages used by the framework to the device-specific messages. This is a useful way to develop software, as it is possible to use an applications written for a specific framework and re-use them for many types of devices. The framework will manage the connection to the robot and may naming services or abstracted access to the hardware of the robot. Examples of such robot frameworks are Player [5] and Robot Operating System (ROS) [10]. As a prerequisite, such frameworks will need some sort of adapter to map the framework-specific messages to the device-specific messages, or a driver. Such drivers are specific to the particular permutation of framework and device.

For new frameworks or new robots, this means that one has to create drivers for popular existing robots or frameworks before many will consider the use of the platform or device as a possible target for their application. For those wishing to target a particular device and platform, they are reliant on there existing a driver already for that device-framework pair. As the number of devices and frameworks increase, then the work will increase quadratically as there must exist a driver for each possible pairing of device and framework.

The Robot Device Interface Specification (RDIS) is an effort to solve this problem. The general goal is to discover the shared abstract concepts which allow the framework and device to communicate. This goal lends itself to a large scope as robot software development encompasses a large set of tools, techniques, and ideas and it is the framework's goal to cover all of these areas and offer a rich API to the application developer. However, there is no one framework than roboticists decisively use, nor is there a standardized protocol for communicating with robots. Therefore something must be done to bridge this gap between framework and device.

Originally, RDIS was implemented as a domain specific language (DSL) using the parser generator ANTLR and its companion tool for code generation, StringTemplates [2]. This implementation generated code to support a few different robot-framework pairs. Later, the model was refined with the aim of supporting a wide variety of robots [1]. The syntax has taken a few different forms, initially taking form as a properties-like syntax and later converted into a JavaScript Object Notation (JSON) compatible syntax [4].

While achieving promising results, there are a couple of flaws associated with this approach. The first is the JSON syntax itself. The general goal in crafting a DSL is to accelerate the speed of development for the end user. However, one can quickly find that a JSON-compatible syntax is not the most intuitive language to craft by hand. However, there is merit in using JSON because of the ease of use in data exchange, particularly via the network, and the wealth of parsers which exist for it in a variety of languages.

The operational semantics of the internal RDIS model were also directly written into the templates, parameterized by the declarative description. While functional, it was generally thought that loading a description into memory interpreting the description at run-time was a more desirable solution. This is a better style for modularity and results in more maintainable components.

Finally, there was a huge amount of work involved in maintaining the validator itself. Extending the validator meant a modification to the grammar and possibly the feasibly many underlying templates. JSON was also a format ill-suited for supporting keywords as to maintain JSON-compatible syntax means to make the keywords indistinguishable from string type data.

RDIS has no notion of how a framework works. Instead, it demands that the framework itself pass standardized message types into the model and promises to return messages of a particular type via its output channels. In order to achieve the goal of interpreting RDIS descriptions in the framework, at some point an RDIS adapter must be produced for each existing framework. If a generalized formalism for describing frameworks existed, RDIS could be empowered to generate its own framework adapters.

However, frameworks vary wildly in their internal messaging systems and building a generalized formalism for modeling them is a large undertaking. Given this is not a well-understood task it is acceptable to limit the scope for the purpose of discovery.

This paper outlines how RDIS has been refined and implemented using model-driven techniques to create a domain-specific visual modeling language. The motivation is to remove barriers created by the adopted textual syntax, ground RDIS in a formal meta-model, and to explore options for code generation of both RDIS textual descriptions and framework adapters. The tool of choice for this article is A Tool for Multi-formalism Meta-Modeling (AToM^3).

The rest of the paper is laid out as follows. Section 2 presents a solution to the above outlined problems. Section 3 presents an evaluation of the solution. Section 4 presents some related work. Finally, Section 5 gives some conclusions for this project and future work.

2 Implementation

A solution to the above problems was implemented using the AToM^3 modeling tool [6] and Python. The implementation provides:

– A modeling formalism for RDIS
– A modeling formalism for ROS
– A model transformation for assigning the mappings between ROS and RDIS
– Code generators for both RDIS textual descriptions and an RDIS-ROS adapter
– An interpreter for RDIS descriptions

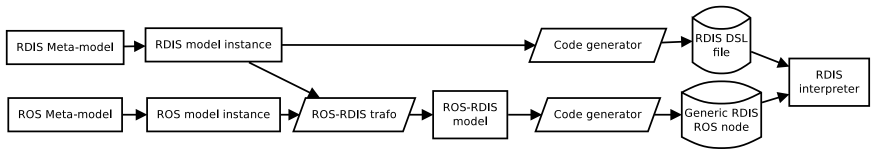


Fig. 1. Flowchart outlining the implementation

Figure 1 shows a flow chart for a general picture of the solution. There are two meta-models at work: a ROS meta-model and an RDIS meta-model. Valid instances of each of these meta-models may be fed as inputs into a model transformation which produces a composite ROS-RDIS model which contains the mappings between each of their message styles. The model produced from this transformation may then be compiled into a ROS node. Any abstract syntax graph containing a valid RDIS model may be serialized into an RDIS textual description.

The rest of this section is divided into subsections which explain how each component of the solution works in detail.

2.1 The RDIS Formalism

The RDIS formalism seeks to provide an easier way for developers to create robot descriptions conforming to the RDIS model. Each of the components of the RDIS model is assigned a visual concrete syntax. Valid model instances can then be serialized to their equivalent JSON form.

Many of the original model elements outlined in are present in the explained implementation. However, some of the model was changed during this implementation. This section gives a higher-level overview of generally how the RDIS model works and presents the modeling environment created for RDIS. Interested readers are referred to a previous paper [1] for more detailed discussion of the model itself.

The general goal of RDIS is to bridge the gap between framework and device in a way that is standardized and general for a large number of devices and frameworks. A specific subgoal of this is the specifically bridge the communication between the framework and device. At a high level, the solution is to standardize the framework-RDIS interface and the RDIS-device interface, and connect the two using intra-model message passing. Figure 2 illustrates this concept.

Given the complex network that can form inside the RDIS layer, this can be a difficult model to visualize in a text-only environment. A visual modeling environment can alleviate the work of creating the robot description by allowing the modeller to visually organize the description, and can offer more helpful feedback in the event of an error.

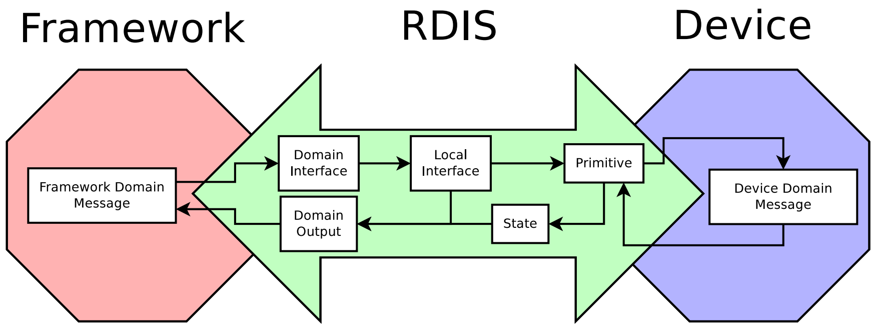


Fig. 2. An overview of the RDIS architecture

A formalism for AToM^3 was created. An example of the resulting concrete syntax is demonstrated in Figure 3. The modeling environment offers a wealth of incremental and compile-time constraints guiding the modeler to create valid model instances.

The significant advantage of a modeling approach is the ability to visualize the web of relations between the model components. Previously, in a textual syntax all of these relations would need to be specified by name. On top of placing the strain of remembering the names of the model elements on the modeler, this also introduces the potential for misspellings. A validator would need to check for broken name references to check if the model is well-formed. However, as the model will almost always contain circular references, this sort of name validation must be completed in multiple passes over the input which can be difficult to implement.

With a modeling approach broken name references are an impossibility because the correct name of the model element can always be drawn by following

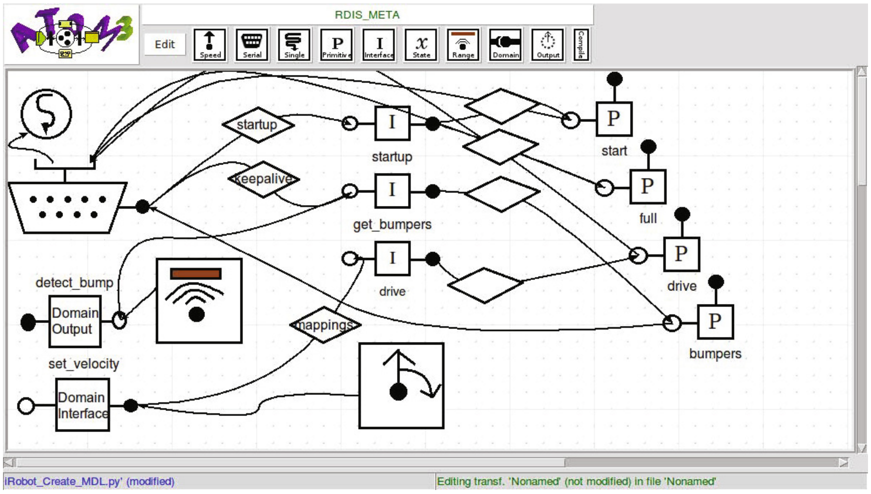


Fig. 3. RDIS modeling environment with concrete syntax for an iRobot Create

an edge in the ASG. If no edge exists, a constraint violation may be raised or that attribute may not be serialized during code generation. Small, incremental changes may be made quicker because a modeler may locate the component that needs to be changed based on human heuristics.

2.2 The ROS Formalism

With the robot description formalism created, the next step is to examine how one might create the adapters which become the framework-RDIS bridge. The problem becomes simpler when targeting just a specific framework. For the purpose of discovery, this paper focuses on ROS. The reason for the choice of ROS is because of its recent rise to popularity, growing community, and a communication model which is easy to grasp.

ROS at its core is a publish/subscribe communication broker. In the following sections basic terminology necessary for this solution is presented. A more complete overview of ROS is available in [10].

ROS Nodes. Nodes are the basic execution unit of ROS. For each instance of a node in the modeling environment, a node will be generated, sharing a filename with the model element name as well. A node publishes or subscribes to different topics. A subscription to a topic means the node will listen to that topic and react to messages which are inputted via that topic. Publishing to a topic generally happens periodically (e.g., reading a sensor), or in response to a message which travels in via one of the subscribed topics.

ROS Topics. Topics are a named data channel. Topics are strongly typed, and a valid ROS type must be specified for each topic. Topic types may be one of the common types shipped with ROS, or it may be a nonstandard, custom type which only exists for one particular package.

ROS Types. As mentioned previously, topics must have a type. ROS types are defined via a mini-language so that the ROS types may be generated for many different languages. A good approach to modeling types should reverse engineer this model so that any general ROS type could be constructed. This might be counter-productive as the goal of RDIS is to use standard types on either side of the framework-RDIS interface. Instead of fully modeling the ROS type system two particular types were selected. This presents a very constrained modeling environment. While it is expected that the chosen set is not complete, it is hoped that a finite set of types on the framework side (and on the RDIS side) as well such that an adequate number of devices and applications may be supported.

We should take a moment to become familiar with the types used in this project. Pose is a 6-dimensional data type which describes both position and orientation in space. Twist is the time derivative of Pose, and it is how ROS represents a change in position and orientation over time. The second ROS type modeled in this project is a Boolean, which is a wrapper type for booleans in the host language.

ROS Modeling Environment. Each of these model elements was assigned a concrete syntax for use in AToM^3. The result is a modeling formalism for generating skeletons of ROS nodes in that enough information is present to define the subscriptions and publications for a given node, but the actions that must be taken in response cannot be generated. The modeling environment is shown in Figure 4.

The formalism is not helpful until its relationship to the RDIS model is defined. This may be done using a model transformation.

2.3 ROS-RDIS Model Transformation

A model transformation which defines the bindings for RDIS domain interfaces and domain outputs to ROS publishers and subscribers has been created. It takes as input a ROS skeleton node model and a full, valid RDIS model and maps subscribed topics of the ROS type Twist and creates a mapping to domain interfaces which accept a Differential Speed. Differential Speed is just a special case of Twist where one linear component and one angular component are needed to describe motion.

Similarly, we can define a mapping between Range and Boolean. Range offers a one-dimensional (later in higher dimensions) view of how far ahead an obstacle is. For bumper type sensors this is a boolean type view, but for more advanced laser systems one can measure an actual distance to the object. We define when an object is colliding with the robot, the range on the operative sensor should be zero. If an application only needs to know if the robot has bumped into something, and it is subscribing to a Boolean-type topic with the expectation

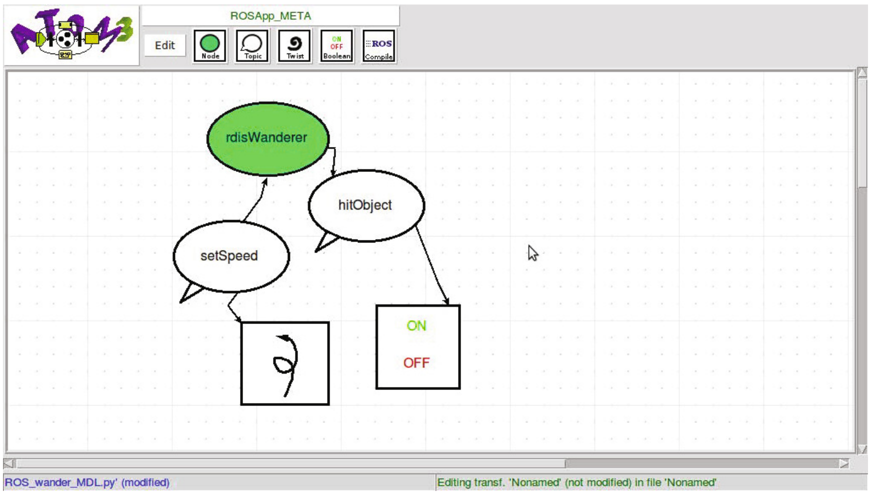


Fig. 4. ROS modeling formalism with an example skeleton node

that the message will be provided, we can map a Range on a domain output to that topic by a logical comparison of the Range value to zero.

The output model of this transformation then has enough information to fully generate a ROS node.

2.4 Code Generators

Code generation is the ultimate goal of the project. There are two artifacts to generate: a textual RDIS description and a ROS node. Two different strategies are adopted for each due to the nature of the artifact generated.

RDIS Code Generator. An RDIS description is a declarative description for a robot's communication model. Given that it is declarative, it can have a very natural correspondence to a data structure in computer memory. Because of this, the context object pattern of code generation is adopted.

Python has a built-in JSON serializer (the `json` module) which natively knows how to serialize dictionaries and lists to their equivalent JSON types. The strategy used for code generation in the solution is to make a pass over the input AToM^3 abstract syntax graph and build a context object using simple Python types. At the end of the pass, the context object is then serialized to an output file.

Listing 1.1 shows an example of the generated textual syntax for a Primitive from the iRobot Create robot. A couple of the differences between the abstract syntax and the textual syntax may be seen. The primitive-connection relation for instance is not represented by the containment of a connection by a primitive, but rather a by-name reference to the connection.

```
{
  "postActions": [
    "bumperStates = __out__[0]"
  ],
  "name": "bumpers",
  "formatArgs": [
    "<142>",
    "<7>"
  ],
  "connection": "btserial",
  "unpack": "B",
  "pack": "BB"
}
```

Listing 1.1. An example of a generated primitive

2.5 ROS Code Generator

A ROS node is also generated. A ROS node may currently be in C++ or Python; this solution generates ROS nodes in Python. A context object approach would then not be appropriate. A template-based code generation scheme works quite well. The benefit for this is that the templates can be easily changed without modification to the code generation module itself. However one then must maintain templates for many types of frameworks.

Listing 1.2 demonstrates a callback generated for a subscribed topic which receives a Twist object. The goal is to map the Twist object into its equivalent Differential Speed message. The RDIS interpreter accepts dictionaries containing the name-value pairs it expects to see in a Differential Speed as its argument. The code generator will pull the expressions from the Differential Speed adapter in the abstract syntax, overwrite the key in the dictionary which shares the name of each attribute with the result of the expression that attribute is tied to, and call the relevant domain interface.

```
def setSpeed_callback(data):
  global gModel

  angular=(data.angular.x, data.angular.y, data.angular.z)
  linear=(data.linear.x, data.linear.y, data.linear.z)

  env = dict()
  env["angular"] = angular
  env["linear"] = linear

  env["angular"] = rdis.safeEval("<angular[0]>", env)
  env["linear"] = rdis.safeEval("<linear[0]>", env)

  gModel.callDomainInterface("set_velocity", env)
```

Listing 1.2. Example of a generated callback for a subscribed topic

2.6 RDIS Interpreter

For this solution, an interpreter was written in Python. The interpreter is derived heavily from the meta-model in the design of the modeling environment with each major entity having a Python class describing its operational semantics with respect to its instance variables.

The interpreter was written because previously there were no existing tools for RDIS besides a parser and several accompanying templates. However, the syntax and model had changed greatly over the scope of the project, so this solution opted not to use the original parser. Instead, a custom parser was written specifically for constructing the model in Python.

Since model has been serialized to JSON, stock parsers may be used to load the model from its abstract syntax. Validation is not implemented in the interpreter. This is because the focus of the solution is to generate valid inputs for the interpreter rather then the interpreter verifying the input. The focus is more on the model than it is the textual syntax. As a sanity check, the interpreter can execute the model and the result may be observed for validity.

Interpretation is simply a series of intra-model message passing. On the side of the framework one has the option of which domain interface to call. The model will decide which primitives to call as a result. Similarly, after executing a local interface, the model will decide if it needs to invoke a domain output which eventually invokes a framework-side callback with an RDIS message on its payload.

Eventually, the model will need its own internal threads, but this is not implemented at present. The reason for this need is calling periodic interfaces and the keepalive interface.

3 Evaluation

This section presents an evaluation of the performance of the model-driven means of creating robot drivers as opposed to the traditional method. Modeling speedup, maintainability, and limitations will be addressed.

3.1 Modeling Speedup

The speed of modeling is the most significant gain in this method. A developer who is familiar with the model, modeling tool, and the device can generate an initial description in well less than an hour. If the generation target filename is set up correctly, incremental changes can be made quickly, redeployed, and tested at greater efficiency. The developer will spend most of his or her time refining the mappings between framework messages and RDIS messages, which is the desired outcome. Where the alternative is manually creating an RDIS description in a text editor with no feedback, the model editor is the clear winner.

Some drawbacks to this method is that the model used is not one which is universally understood by robot software developers. As well, most robot software developers will be more accustomed to hand-crafted code rather than code generation. Finally, communication models are not normally described declaratively, but rather in a general-purpose programming language. So, any model-based approach would seem very foreign to a robot software developer, and would take some time to be adopted by the community as a whole.

For generating ROS nodes the speedup may not be so noticeable. This may be due to the way that the transformation is defined. There is a very clear constructive mapping from an RDIS description to an equivalent ROS node. The current solution requires an explicit design of a ROS node. A better solution might generate the subscriptions by discovery at runtime of domain interfaces or domain outputs, and a static mapping of ROS types to RDIS types and vice-versa. Another possible solution is designing a transformation which creates ROS instance models from an RDIS description.

3.2 Maintenance and Refactoring

Assuming a static meta-model, long term maintenance should be achievable. If changes within the device specifications occur, then those changes should not often propagate beyond just one model component. As an example, if the Create's "read bumpers" primitive changed from returning one byte to two, then only that primitive's unpack string and post-actions would need to be modified.

Trouble might arise from changes in the meta-model. The addition of types of model elements could be supported without much trouble. A large-scale refactoring would invalidate all current model instances and a transformation would need to be defined to recover them. The main problem would be with the interpreter. The operational semantics of RDIS are currently not modeled, so keeping the interpreter synchronized would be an issue. As well the code generators are not modeled, so they would have to be changed by hand.

3.3 Limitations and Assumptions

RDIS is currently limited to describing disconnected devices which are controlled via some transport. Only robots which use fixed width messages can be modeled. Robot systems which are in part heterogeneous (e.g., an having an attachment) may work in its current state but remain untested. Threading is still an unsolved issue.

Some assumptions are made which make RDIS possible in concept. The first of which is that a finite set of framework-RDIS messages exists such that a good portion of robot applications can be supported. There is still research to be done in the categorization of messages which may travel between the robot and the framework.

4 Related Work

There is a small community within the modeling world who work specifically with modeling robots. It is first worth mentioning that the Object Management

Group has some standards [7–9] which their Robotics Domain Task Force has highlighted as relevant to the robotics domain. These standards are not in wide use to the author's knowledge.

The Eclipse Modeling Framework (EMF) has been used to generate code for several robotic frameworks [11]. Members of the same group have also created a meta-model for the analysis of robot systems [13]. In the former project, the primary difference in the approach that this solution takes is that the model is not interpreted; it is compiled.

Another researcher modeled a subsumption architecture and was able to generate code for robots as well [14]. Subsumption is an architecture type for robots dealing with the behavior of robots. RDIS does not deal at all with behavior, and assumes that the behavior unit is wholly separate of the communication model.

Outside of pure modeling, there are several domain-specific languages which are tailored for robot control [3,12]. These languages offer an enriched syntax for defining robot behavior as well. They do not solve the problem defined in this paper as they depend on an external module to actually handle the message passing.

5 Conclusion

The problem of creating drivers for pairs of robots and frameworks is significant and if left unchecked will leave the robotics community littered with incompatible combinations of devices and frameworks. RDIS models the communication between framework and device and vice-versa. When formalized it can generate code for a variety of platform-device pairs. Previously, RDIS was thought of as a domain specific language and adopted a JSON syntax. However, the tools were difficult to maintain, and there was no editor for the DSL; and the constrained syntax made it difficult to write descriptions for. This solution puts more emphasis on the model and treats the textual syntax as a compiled form of the model. The result is improved creation of RDIS textual description. The solution also demonstrates the possibility of compiling ROS nodes from an RDIS model.

There are still many flaws in the current implementation. Threading is largely ignored. Fixed-width messages can only be modeled. ASCII-encoded messages have not been tested. Periodic interfaces are not considered in the interpreters, and as well scheduling is not handled in the interpreter. Also, more attention should be given to the compile-time constraints so that no invalid RDIS textual descriptions may be generated.

Future work lies defining the set of domain interfaces and domain outputs. This is an essential assumption of RDIS so it is important these are defined and standardized. More work will be done in the coming months in the specification of the physical construction and kinematics of robots. Even then RDIS is not complete. More work needs to be done in the enhancement of robot state, exception handling, sensor and actuator error modeling, and improved modeling of threading in addition to that outlined above.

References

1. Anderson, M., Kilgo, P., Bowman, J.: RDIS: Generalizing domain concepts to specify device to framework mappings. In: International Conference on Robotics and Automation (May 2012)
2. Anderson, M., Kilgo, P., Crawford, C., Stanforth, M.: Work in progress: Enabling robot device discovery through robot device descriptions. In: 2nd International Workshop on Domain-Specific Languages and Models for ROBotic Systems (September 2011)
3. Baillie, J.C.: URBI: towards a universal robotic low-level programming language. In: IEEE International Conference on Robotics and Automation (2007)
4. Crockford, D.: The application/json Media Type for JavaScript Object Notation (JSON). RFC 4627 (Informational) (July 2006), http://www.ietf.org/rfc/rfc4627.txt
5. Gerkey, B.P., Vaughan, R.T., Howard, A.: The player/stage project: Tools for multi-robot and distributed sensor systems. In: Proceedings of the 11th International Conference on Advanced Robotics, pp. 317–323 (2003)
6. de Lara, J., Vangheluwe, H.: AToM^3: A Tool for Multi-formalism and Meta-modelling. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 174–188. Springer, Heidelberg (2002), http://dx.doi.org/10.1007/3-540-45923-5_12
7. OMG: Robotic Technology Component (RTC) 1.0. Tech. rep., Object Management Group (April 2008)
8. OMG: Super Distributed Object (SDO) 1.1. Tech. rep., Object Management Group (October 2008)
9. OMG: Robot Localization Service (RLS) 1.0. Tech. rep., Object Management Group (February 2010)
10. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source robot operating system. In: Proc. Open-Source Software workshop of the International Conference on Robotics and Automation, ICRA (2009)
11. Schlegel, C., Hassler, T., Lotz, A., Steck, A.: Robotic software systems: From code-driven to model-driven designs. In: International Conference on Advanced Robotics, ICAR 2009, pp. 1–8 (June 2009)
12. Schultz, U., Christensen, D., Stoy, K.: Automatic program generation for embedded systems. In: Proceedings International Conference on Advanced Robotics, ICAR 2009, pp. 28–36 (October 2007)
13. Steck, A., Schlegel, C.: Towards quality of service and resource aware robotic systems through model-driven software development. CoRR abs/1009.4877 (2010)
14. Trojanek, P.: Model-driven engineering approach to design and implementation of robot control system. In: 2nd International Workshop on Domain-Specific Languages and Models for ROBotic Systems (2011)

PRACSYS: An Extensible Architecture for Composing Motion Controllers and Planners

Andrew Kimmel, Andrew Dobson, Zakary Littlefield,
Athanasios Krontiris, James Marble, and Kostas E. Bekris*

Computer Science Department, Rutgers University, Piscataway, NJ, 08554, USA
kostas.bekris@cs.rutgers.edu

Abstract. This paper describes a software infrastructure for developing controllers and planners for robotic systems, referred here as PRACSYS. At the core of the software is the abstraction of a dynamical system, which, given a control, propagates its state forward in time. The platform simplifies the development of new controllers and planners and provides an extensible framework that allows complex interactions between one or many controllers, as well as motion planners. For instance, it is possible to compose many control layers over a physical system, to define multi-agent controllers that operate over many systems, to easily switch between different underlying controllers, and plan over controllers to achieve feedback-based planning. Such capabilities are especially useful for the control of hybrid and cyber-physical systems, which are important in many applications. The software is complementary and builds on top of many existing open-source contributions. It allows the use of different libraries as plugins for various modules, such as collision checking, physics-based simulation, visualization, and planning. This paper describes the overall architecture, explains important features and provides use-cases that evaluate aspects of the infrastructure.

1 Introduction

Developing and evaluating control or motion planning methods can be significantly assisted by the presence of an appropriate software infrastructure that provides basic functionality common among many solutions. At the same time, new algorithms should be thoroughly tested before applied on a real system. Physics-based simulation can assist in testing algorithms in a more realistic setup so as to reveal information about the methods helpful for real world application. These realizations have led into the development of various software packages for physics-based simulation, collision checking and motion planning for robotic and other physical systems, such as Player/Stage/Gazebo [1], OpenRAVE [2], OMPL [3], PQP [4], USARSim [5]. At the same time many researchers interested in developing and evaluating controllers, especially for systems with interesting dynamics, often utilize the extensive set of Matlab libraries.

There are numerous problems, however, which require the integration of multiple controllers or the integration of higher-level planners with control-based methods. For instance, controlling cyber-physical systems requires an integration of discrete and

* This work has been supported by NSF CNS 0932423. Any conclusions expressed here are of the authors and do not reflect the views of the sponsor.

continuous reasoning, as well as reasoning over different time horizons. Similarly, a problem that has attracted attention corresponds to the integration of task planners with motion planners so as to solve challenges that are more complex than the traditional Piano Mover's Problem. At the same time, interest is moving towards higher-dimensional and more complex robotic platforms, including humanoid systems and robots with complex dynamics.

This work builds on top of many existing contributions and provides an extensible control and planning framework that allows for complex interactions between different types of controllers and planners, while simplifying the development of new solutions. The focus is not on providing implementations of planners and controllers but defining an environment where new algorithms can be easily developed, integrated in an object-oriented way and evaluated. In particular, the proposed software platform, PRACSYS[1], offers the following benefits:

- **Composability:** PRACSYS provides an extensible, composable, object-oriented abstraction for developing new controllers and simulating physical systems, as well as achieving the integration of such solutions. The interface is kept to a minimum so as to simplify the process of learning the infrastructure.
- **Ease of Evaluation:** The platform simplifies the comparison of alternative methods with different characteristics on similar problems. For instance, it is possible to evaluate a reactive controller for collision avoidance against a replanning sampling-based or search-based approach.
- **Scalability:** The software is built so as to support lightweight, multi-robot simulation, where potentially thousands of systems are simulated simultaneously and where each one of them may execute a different controller or planner.
- **New Functionality:** PRACSYS builds on top of existing motion planning software. In particular, the OMPL [3] library focuses on single-shot planning but PRACSYS allows the use of OMPL algorithms on problems involving replanning, dynamic obstacles, as well as extending into feedback-based planning.
- ROS **Compatibility:** The proposed software architecture is integrated with the Robotics Operating System (ROS) [6]. Using ROS allows the platform to meet a standard that many developers in the robotic community already utilize. ROS also allows for inter-process communication, through the use of message passing, service calls, and topics, all of which PRACSYS takes advantage of.
- **Pluggability:** PRACSYS allows the replacement of many modules through a plugin support system. The following modules can be replaced: collision checking (e.g., PQP [4]), physics-based simulation (e.g., Open Dynamics Engine [7]), visualization (e.g., OpenSceneGraph [8]), as well as planners (e.g., through OMPL [3]) or controllers (e.g., Matlab implementations of controllers).

After reviewing related contributions, this paper outlines the software architecture and details the two main components of PRACSYS, simulation and planning. The paper also provides a set of use-cases that illustrate some of the features of the software infrastructure and gives examples of various algorithms that have been implemented with the assistance of PRACSYS.

[1] SourceForge package: http://sourceforge.net/projects/pracsys/

2 Related Work

The Robot Operating System (ROS) [6] is an architecture that provides libraries and tools to help software developers create robot applications. It provides hardware abstractions, drivers, visualizers, message-passing and package management. PRACSYS builds on top of ROS and utilizes its message-passing and package management. ROS was inspired by the Player/Stage combination of a robot device interface and multi-robot simulator [9]. Gazebo is focusing on 3D simulation of multiple systems with dynamics [1]. PRACSYS shares objectives with Gazebo but focuses mostly on a control and planning interface that is not provided by Gazebo.

There is a series of alternative simulators, such as USARSim [5], the Microsoft Robotics Developers studio [10], UrbiForge [11], the Carmen Navigation Toolkit [12], Delta3D [13] and the commercial package Webots [14]. Most of these systems focus on modeling complex systems and robots and not on defining a software infrastructure for composing and integrating controllers and planners for a variety of challenges.

Other software packages provide support for developing and testing planners. For instance, Graspit! [15] is a library for grasping research, while OpenRAVE [2] is an open-source plugin-based planning architecture that provides primitives for grasping and motion planning for mobile manipulators or full-body humanoid robots. The current project shares certain objectives with tools, such as OpenRAVE. Nevertheless, the definition of an extensible, object-oriented infrastructure for the integration of controllers, as well as the integration of planners with controllers to achieve feedback-based planning, are unique features of PRACSYS. Furthermore, multiple aspects of OpenRAVE, such as the work on kinematics, are complementary to the objectives of PRACSYS and could be integrated into the proposed architecture. The same is true for libraries focusing on providing prototypical implementations of sampling-based motion planners, such as the Motion Strategy Library (MSL) [16] and the Open Motion Planning Library (OMPL) [3]. In particular, OMPL has already been integrated with PRACSYS and is used to provide concrete implementations of motion planners. The proposed infrastructure, however, allows the definition of more complex problems than the typical single-shot motion planning challenge, including problems like replanning.

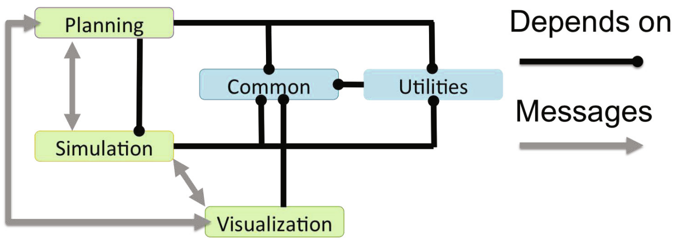


Fig. 1. Package interactions. ROS nodes communicate via message passing: *simulation*, *visualization*, and *planning*. The *common* and *utilities* packages are dependencies of the previous three.

3 General Architecture of PRACSYS

The proposed architecture is composed of several modules, following the architecture of the Robotic Operating System (ROS) [6]. ROS's architecture has separate nodes launched as executables which communicate via message passing and are organized into packages and stacks. A package is a collection of files, while a stack is a collection of such packages. PRACSYS is a stack and each node launched from PRACSYS is associated with a single package. PRACSYS also allows developers to integrate additional plugins into the architecture. There are three packages which run as nodes: the *simulation*, *planning*, and *visualization* packages. See Figure 1 for a visual representation of the interactions between different packages of PRACSYS. The advantage of having separate nodes is that it makes the jump to distributed computation such as on a computing grid easier.

The *common* package contains some useful data structures, as well as mathematical tools. The *utilities* package contains useful algorithms, such as graph search, as well as abstractions for planning. Both the *common* and *utilities* packages use the Boost[2] library to facilitate efficient implementations.

The higher-level packages include *simulation*, *visualization*, and *planning*. The *simulation* package has *common* and *utilities* as dependencies, while it is responsible for simulating the physical world in which the agents reside and contains integrators and collision checking. The same package also contains many controllers which operate over short time horizons. Controllers are part of the main pipeline and are not in the *planning* package because they only operate over a single simulation step. The *planning* package is primarily concerned with controlling agents over a longer horizon, using the *simulation* package internally. The *visualization* package provides an interface between the user and the *simulation*, such as selecting agents and providing manual control. The state of systems simulated in the *planning* package can be different than the state in the ground truth simulator, which is useful for applications such as planning under uncertainty.

PRACSYS makes use of a package for loading simulations from files YAML [17] format. There is also a set of dependencies to external software packages, such as the Approximate Nearest Neighbors library [18].

The following discussion details the capabilities of these packages, starting with the most fundamental of the three: the *simulation* package.

4 Description of the Packages

This section discusses the different packages of PRACSYS in further detail.

4.1 Ground-truth Simulation and Controller Architecture

The *simulation* package is the primary location for the development and testing of new controllers, and contains a set of features which are useful for developers. The following sections will go over each of these features individually.

[2] Boost is a set of libraries that extend the functionality of the C++ programming language.

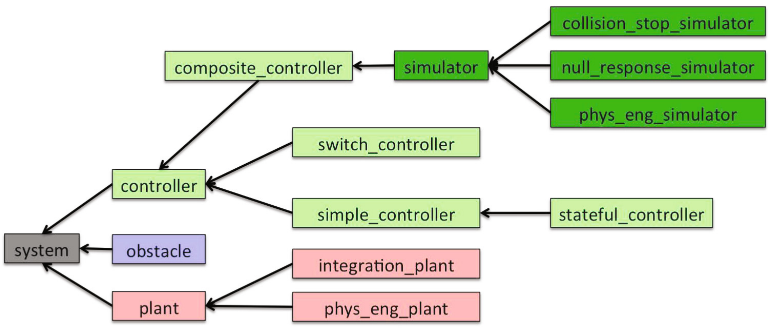


Fig. 2. A view of the class inheritance tree for the PRACSYS *system*. All classes are abstract, with the exception of the switch controller and the three concrete simulator classes.

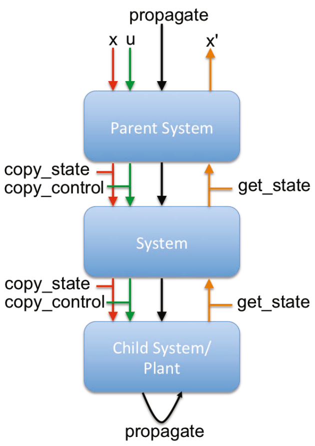


Fig. 3. The core interface of a system: x and x' are states, while u is a control

Composability. The simulator contains several classes which are interfaces used for the development of controllers. The fundamental abstraction is the *system* class - all controllers and plants are *systems* in PRACSYS, as shown in Figure 2. This functionality allows for other nodes, such as planning, to reason over one or more *systems* without knowing the specifics of the system. The interface is the same whether planning happens over a physical plant or a controlled system, which simplifies feedback-based planning.

The interaction between systems is governed by the pipeline shown in Figure 3, which ensures that every system properly updates its state and control. The functions in the pipeline are responsible for the following:

copy state: receive a state from a higher-level system, potentially manipulate this state, and pass it down to lower-level systems.

copy control: receive a control from a higher-level system, potentially manipulate this control, and pass it down to lower-level systems.

propagate: propagates a system according to its dynamics (if it is a plant), or sends a propagate signal down to lower-level systems (if it is a controller).

get state: receives the state from a lower-level system. This allows higher-level systems to query for the full state of the simulation.

The *system* class uses the *space* abstraction, which provides a way for users to define abstract spaces in a general way, allowing for scopes beyond a Euclidean space. A space is a box constrained region in n dimensions where each dimension can be Euclidean, rotational, or one component of a quaternion. The abstraction automatically provides a way to compute metrics between different points within the space. A *space point* stores the parameters of each dimension of the *space*, and can be used to represent states and controls in the *system* class.

Note that a *system* does not need to get controls from lower-level *systems*. The *system* class contains other functions, which primarily fall under the categories of initialization, set functions, and get functions. Systems are broken into: physical plants, obstacles and controllers. Physical plants are responsible for simulating the physical agents and how they move through the environment. They store geometries and have functionality to update their configurations based on states set through copy state. Furthermore, physical plants are governed by state-update equations of the form $\dot{x} = f(x,u)$ implemented by the propagate function.

Controllers are classified into four major types: simple, stateful, composite, and switch. These are classes that extend the abstract controller class. A **simple controller** contains a single subsystem, which is most often the plant itself, but can also be another controller. Simple controllers are useful for creating a chain of controllers, allowing for straightforward compositions. A controller which computes a motion vector along a potential field for a holonomic disk system is an example of a simple controller on top of a physical plant. In more complex compositions, controllers will also have the need to keep an internal state, separate from its subsystem. A **stateful controller** allows for an internal state, and one such example of a *stateful controller* is the consumer controller. The *consumer controller* simply supplies controls to its subsystem from a pre-computed plan, where its state is the point in time along the plan to extract the control. **Composite controllers**, which can have many subsystems, provide the necessary interface for controlling multiple subsystems. The simulator, for example, which is responsible for propagating systems, is a composite controller containing all controllers and plants. The final major type of controller is the **switch controller**, which behaves quite similar to a C/C++ switch statement. A switch controller operates over an internal controller state, called a mode, which determines which of its subsystems is active. For example, a switch controller could be used to change the dynamics being applied to a plant depending if it was in a normal environment or an icy environment, in which case an inactive slip-controller would be "switched" active. Developing controllers which utilize these archetypes allows for an easy way to create more complex interactions.

High-Level *simulation* **Abstractions.** In addition to the *system* abstraction, there are several high-level abstractions which give users additional control over the the simulation. One such abstraction is the **collision checking** abstraction, which consists of an actual collision checker and a collision list. The collision list simply describes which pairs of geometries should be interacting in the simulation, while the collision checker is actually responsible for performing the checks and reporting when geometries have come into contact. The **simulator** is an extension of the composite controller, but it has additional functionality and has the unique property of always being the highest-level *system* in the simulation. The abstraction which users are most likely to change is the **application**. The application class contains the simulator and is responsible for defining the type of problem that the user wants to solve.

Interaction. The *simulation* node can communicate with other nodes via ROS messages and service calls. For example, moving a robot on the visualization side involves a ROS service call. Similar to how a propagate signal is sent between systems, an update signal is used to change geometry configurations. Once all systems have appended to this update signal, a ROS message is constructed from it and sent to *visualization* in order to actually move the physical geometry. For non-physical geometries, such as additional information of a system (i.e., a robot could have a visualized vector indicating its direction), each system is responsible for making the appropriate ROS call. If a user needs additional functionality and interaction between the nodes, they only need to implement their function in the communication class and create the appropriate ROS files.

Plugin Support. Adding a new physics engine as a plugin simply involves extending the simulator, plant, obstacle, and collision checker classes. If a user does not require the use of physics-based simulation, they can simply disable this functionality by omitting it from the input configuration file. For collision checking, PRACSYS currently provides support for PQP, as well as the use of no-collision checking. Similarly, if a user would like to add a new collision checker, they only need to extend the collision checking class.

4.2 Planning

The *planning* package is responsible for determining sequences of controls for one or many agents over a longer horizon than a single simulation step. This excludes methods which use reactive control. Furthermore, *planning* also reasons over higher-level planning processes, such as task coordination. The *planning* package is divided among several modules in order to accomplish these tasks. The high-level task coordination is provided by *task planners*, which contain *motion planners* for generating sequences of controls given a *world model*. The *world model* is a system with a simulator as a subsystem and is responsible for providing information to the planners about the state of the simulator as well as providing additional functionality. The *motion planners* are the individual motion planning algorithms which compute controls. The current version of PRACSYS has certain sampling-based motion planners implemented, which use some basic modules to accomplish their specified tasks, including local planners and validity checkers. PRACSYS also integrates existing motion planning packages such as OMPL by providing an appropriate interface. OMPL is a software package developed for planning purposes [3]. The current focus of the *planning* package has been sampling-based methods; however, it is not limited to these types of planners and can easily support search-based or combinatorial approaches.

High-Level Abstractions. All of the abstractions described in this section interact according to Figure 4.

Task planners are responsible for coordinating the high-level planning efforts of the node. *Task planners* contain at least one instance of a motion planner and use planners to accomplish a given task. Ultimately, the goal of *planning* is to come up with valid trajectories for one or many systems, which bring them from some initial state to a goal state, but the *task planner* may be attempting to accomplish a higher-level task such as motion coordination. In this sense, the task planners are responsible for defining the objective of the planning process, while the motion planners actually generate the plan. One example is the single-shot task planner, which allows a planner to plan until it has computed a path to a specified goal. Then the single-shot task planner forwards the plan to the *simulation* node. Other tasks include single-shot planning, replanning, multi-target planning and velocity tuning or trajectory coordination among multiple agents.

A **World Model** represents the physical world as observed or known to the planner, and also extends the *system* abstraction. A *world model* contains a simulator as a subsystem, exposing the functionality of the simulator to the *motion planners*. World models can be used to hide dimensions of the state space from the motion planners, introduce and model the uncertainty an agent has about its environment, or removing certain agents from collision checking. Having the capability to remove dimensions from the state space is useful for planning purposes because the planning process has complexity which depends on the dimensionality of the space. This reduction will make the planning process more efficient, and is related to being able to remove some systems from collision checking. These two functions together allow a full simulation to be loaded, while allowing planning to plan for agents individually in a decoupled manner for greater efficiency.

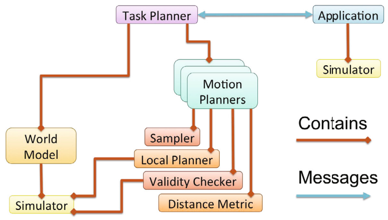


Fig. 4. The general structure of the *planning* modules. The *task planner* contains multiple *motion planners*. The *task planner* also contains a *world model* and communicates with the *simulation* node.

Motion planners are responsible for coming up with trajectories for individual or groups of agents. The flexibility of PRACSYS allows for planners to easily be changed from performing fully decoupled planning to any range of coupling, including fully coupled problems. *Motion planners* employ a set of black-box modules, which may have a wide variety of underlying implementations. Furthermore, because of the flexibility of the *system* class, planners can plan over controllers as well. In this case, planners are essentially able to create trajectories through parameter space of controllers. The sampling-based planners in PRACSYS make use of four modules: local planners, samplers, distance metrics, and validity checkers. The distance metric module and the sampler module are provided by the *utilities* package.

Sampling-Based Motion Modules **Local planners** propagate the agents according to their dynamics. PRACSYS offers two basic types of local planners, an approach local planner which uses a basic approach technique to extend trajectories toward desired states, and a kinematic local planner which connects two states exactly, but only works for agents which have a two-point boundary problem solver and kinematic agents.

Validity checkers provide a way to determine if a given state is valid. The most basic implementation of a validity checker, which is provided with PRACSYS simply takes a state, translates it into its geometrical configuration, and checks if there is a collision between the geometry of the agents and the environment.

Samplers are able to generate samples within the bounds of an abstract *space*. Different samplers will allow for different methods of sampling, such as uniform randomly, or on a grid.

Distance metrics are responsible for determining the distance of points in a *space*. These modules may use simple interpolating methods or may be extended to be more complex and take into account invalid areas of the space.

Interaction. The *planning* package communicates primarily with *simulation*. A *planning* node can send messages to the *simulation* such as computed plans for the agents. The *planning* package can further send trajectory and planning structure information to visualization so users can see the results of an algorithm. The *planning* node also receives control signals from the *simulation* node, such as when to start planning. Because of the plugin system of PRACSYS, a simple wrapper is provided around the existing OMPL implementation in order to utilize the OMPL planners within PRACSYS.

4.3 Visualization

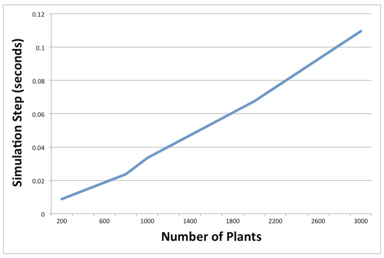
The *visualization* node is responsible for visualizing any aspect needed by the other nodes. Users interact with the simulation environment through the *visualization*. This includes, but is not limited to, camera interaction, screen shots and videos, and robot tracking. The *visualization* provides an interface to develop alternative implementations, in case users do not want to use the provided implementation based on Open Scene Graph (OSG) [8].

4.4 Other PRACSYS Packages

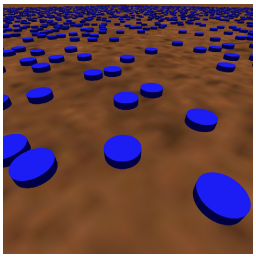
The remaining packages provide functionality useful across the infrastructure, such as geometric calculations, configuration information, and interpolation. An important concept is the idea of a *space* as provided by the *utilities* package, which was described earlier in Section 4.1. The *input* package is an optional package which includes sample input for use with PRACSYS. Configuration files are in YAML or ROS .launch format. PRACSYS also comes with an *external* package for carrying along external software packages, such as the Approximate Nearest Neighbors (ANN) package [18], which is useful for motion planning.

5 Use-Cases

This section provides specific examples of the features offered by PRACSYS.



(a)



(b)

Fig. 5. Figure on the left shows a plot of simulation steps vs number of plants, figure on the right shows 3000 plants running in the environment

5.1 Showing Scalability for Multiple Agents

Scalability is important for simulation environments in which multiple agents are to be simulated at once. The PRACSYS system structure was designed with multi-agent simulation in mind. Given a very simple controller, multiple physical plants were simulated and the time for a simulation step was tracked against number of agents, as shown in Figure 5. The trend shows a linear increase in simulation step duration with the number of agents, even with simulations of thousands of agents. The Velocity Obstacle (VO) Framework was introduced as a lightweight reactive obstacle avoidance technique [19]. The basic VO framework as well as several extensions have been implemented in PRACSYS and have also been used in large-scale experiments.

5.2 Planning Over Controllers Using LQR Trees

Through a C/C++ interface to Octave and its control package [20], PRACSYS can utilize the optimal control guarantees of linear quadratic regulators (LQR). Octave is an open-source Matlab clone. In this way, PRACSYS can run software developed in Matlab with little effort for the conversion. An implementation of LQR-Tree has been developed in PRACSYS [21]. This algorithm is a prototypical example of "planning over controllers" so as to provide feedback-based solutions. The created LQR-Tree is sent to the *simulation* node for execution. The process of incorporating the LQR code into PRACSYS took a matter of minutes.

The LQR-Tree is built incrementally similarly to the Rapidly-exploring Random Tree algorithm (RRT) and its variants. It computes an LQR that is based around the goal region, and then using sampling trajectories until new basins can be created using time-varying LQR over trajectories which enter existing basins. The technique has been shown to probabilistically cover the space, and stores a full description of the LQR controller used to create the basin of attraction at each tree node.

The *planning* node can send the controller information to the *simulation* node. This implementation illustrates the use of LQR controllers inside a planning structure. With this kind of framework, many more complex applications can be implemented and studied. This also shows the integration of a high-level language, primarily intended for numerical computations into PRACSYS.

5.3 Controller Composition in Physics-Based Simulation

PRACSYS offers a unique capability of composing systems. This scheme gives users flexibility by allowing the decomposition of individual steps into separate controllers, so that they can be reused and re-combined to create new functionality.

For example, the framework given in the SIMBICON project [22], in which controllers are created for controlling bipedal robots has been implemented in PRACSYS. The hierarchy of controllers for this implementation is shown in Figure 6. A breakdown of this hierarchy is as follows: **ODE Simulator:** A simulator built on top of the Open Dynamics Engine. **Finite State Machine (FSM):** Several FSMs, implemented with *switch controllers*, sit below the simulator. Each FSM corresponds to a particular bipedal gait, such as running or skipping. Changes in the state of the simulation eventually causes a switch in the used gait. **Bipedal PD Controller:** Several PD controllers sit underneath each FSM, and represents a specific part of a gait, such as being mid-stride or having both feet planted. **Bipedal Plant:** This is the physical representation of the robot, and contains the geometry and joint information, as well as the actual dynamics of the plant.

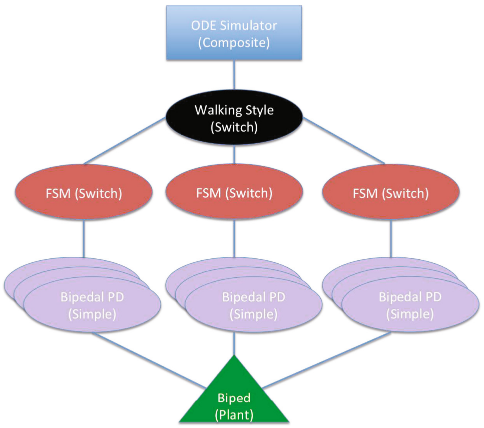


Fig. 6. A visual representation of the controller composition for controlling a bipedal robot

Because ODE focuses on quick simulation for real-time applications, interactive applications can be created while sacrificing as little realism as possible. This lightweight implementation allows users to interact with this physically simulated world in interesting ways, such as manually controlling a plant among many other plants being controlled through various means. An example as shown in the submission's video shows a toy car controlled by a user interacting with the bipedal *system*, described previously.

5.4 Integration with Octave, OMPL, and MoCap Data

PRACSYS has been integrated with several other software packages in order to extend its functionality, such as Octave, OMPL, and motion capture data from CMU. **Motion Capture** (MoCap) data is used to animate characters in a realistic manner. A controller which reads motion capture data has been utilized, and it reads and assigns the data to a control space point, where it is passed to the plant with copy control. The plant connected to this controller simulates a skeleton, which interpolates the configuration of each of its bones.

6 Discussion

PRACSYS is an extensible environment for developing and composing motion controllers and planners. It supports multi-agent simulations, physics-based tools, and incorporates Matlab code, the OMPL library [3] and MoCap data. There are multiple important future steps for PRACSYS. A current pursuit is the development of a communication node, which simulates communication protocol parameters and failures between agents by employing a discrete event network simulator, such as ns-3 [23]. This will allow the simulation of distributed planning involving communication on a computing cluster. Furthermore, a sensing node is developed for simulating sensor data in place of a physical sensor. This objective, as well as allowing algorithms coded on PRACSYS to run on physical systems, will be assisted by a tighter integration with the latest versions of Gazebo [1], OpenRAVE [2] and by utilizing existing ROS functionality [6].

References

[1] Koenig, N., Hsu, J., Dolha, M., Willow Garage, Gazebo, http://gazebosim.org/
[2] Diankov, R., Kuffner, J.J.: OpenRAVE: A Planning Architecture for Autonomous Robotics. Technical report, CMU-RI-TR-08-34, The Robotics Institute, CMU (2008)
[3] Kavraki Lab Group: The Open Motion Planning Library (OMPL), http://ompl.kavrakilab.org
[4] Gottschalk, S., Lin, M.C., Manocha, D.: OBBTree: A Hierarchical Structure for Rapid Interference Detection. In: SIGGRAPH, pp. 171–180 (1996), http://gamma.cs.unc.edu/SSV/
[5] Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: A Robot Simulator for Research and Education. In: IEEE ICRA, pp. 1400–1405 (2007)
[6] Willow Garage, Robot Operating System (ROS), http://www.ros.org/wiki/
[7] Smith, R.: The Open Dynamics Engine (ODE) (2007), http://ode-wiki.org/wiki/
[8] OpenSceneGraph, http://www.openscenegraph.org/
[9] Gerkey, B., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor and Systems. In: ICAR, pp. 317–323 (2003)
[10] Microsoft Robotics Developer Studio, http://www.microsoft.com/robotics/
[11] UrbiForge, http://www.urbiforge.org/
[12] Carmen Robot Navigation Toolk, http://carmen.sourceforge.net/home.html
[13] Delta3D (2006), http://www.delta3d.org/
[14] Michel, O.: Webots: Professional Mobile Robot Simulation. IJARS 1(1) (2004)
[15] Miller, A.: Graspit!: A Versatile Simulator for Robotic Grasping. PhD thesis, Columbia University (2001), http://www.cs.columbia.edu/~cmatei/graspit/
[16] LaValle, S.: Motion Strategy Library, http://msl.cs.uiuc.edu/msl/
[17] YAML Ain't Markup Language (YAML), http://yaml.org/
[18] Arya, S., Mount, D.M.: Approximate nearest neighbor searching. In: Proc. 4th Annual ACM-SIAM Symposium on Discrete Algorithms, pp. 271–280 (1993)
[19] Fiorini, P., Shiller, Z.: Motion Planning in Dynamic Environments Using Velocity Obstacles. International Journal of Robotics Research (IJRR) 17(7), 760–772 (1998)
[20] Eaton, J.W.: GNU Octave Manual. Network Theory Limited (2002)
[21] Reist, P., Tedrake, R.: Simulation-based LQR-Trees with input and state constraints. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 5504–5510 (2010)
[22] Yin, K., Loken, K., van den Panne, M.: SIMBICON: Simple Biped Locomotion Control. ACM Transactions on Graphics 26(3) (2007)
[23] NS3, http://www.nsnam.org/

RobotML, a Domain-Specific Language to Design, Simulate and Deploy Robotic Applications

Saadia Dhouib[1], Selma Kchir[2], Serge Stinckwich[3,4], Tewfik Ziadi[2], and Mikal Ziane[2]

[1] CEA, LIST, Laboratory of Model Driven Engineering for Embedded Systems Point Courrier 94, Gif-sur-Yvette, F-91191 France
[2] UMR CNRS 7606 LIP6-MoVe, Université Pierre et Marie Curie, Paris, France
[3] UMR CNRS 6072 GREYC, Université de Caen-Basse Normandie/ENSICAEN, Caen, France
[4] UMI IRD 209 UMMISCO
IFI/Vietnam National University, Hanoi, Vietnam
saadia.dhouib@cea.fr, serge.stinckwich@ird.fr, mikal.ziane@lip6.fr

Abstract. A large number of robotic software have been developed but cannot or can hardly interoperate with each other because of their dependencies on specific hardware or software platform is hard-wired into the code. Consequently, robotic software is hard and expensive to develop because there is little opportunity of reuse and because low-level details must be taken into account in early phases. Moreover, robotic experts can hardly develop their application without programming knowledge or the help of programming experts and robotic software is difficult to adapt to hardware or target-platform changes. In this paper we report on the development of RobotML, a Robotic Modeling Language that eases the design of robotic applications, their simulation and their deployment to multiple target execution platforms.

Keywords: Domain-Specific Languages, Generative Programming, MDE, robotic application.

1 Introduction

Large amounts of robotic software have been developed but cannot or can hardly interoperate with each other because their dependencies on specific hardware or software platform is hard-wired into the code. Thus, changing one or several of the used hardware components in the application implies several time consuming changes in the code. In addition, robotic software is difficult to adapt to target platform changes. Consequently, robotic software is hard and expensive to develop because there is little opportunity of reuse. Moreover, robotic experts can hardly develop their application without programming knowledge or the help of programming experts. This knowledge is not only related to algorithm programming but to the arcanes of the chosen simulation platform and the details of drivers of sensors and actuators.

Defining robotic applications using appropriate notations and abstractions and automatically generating executable code could be a solution to deal with variability problems and to hide the lower level programming details to robotics experts. Domain-specific languages (DSLs) have been introduced, based on thorough understanding of the application domain, to express solutions at the level of abstraction of the problem domain.This assumes that along with the DSL itself a whole toolchain is provided to automate as much as possible the lower-level task and especially code generation [4,10].

In this paper we propose the RobotML DSL and a toolchain to address these issues. In section 2, we outline the requirements for RobotML that is then presented in section 3. Afterwards, we describe the RobotML toolchain, from modeling to code generation in section 4. Then, a case study is outlined in section 5, validating the proposed DSL on a real robotic use case. Section 6 discusses how the proposed RobotML DSL fulfills the requirements presented in Section2. In Section 7, we outline previous work related to our approach and section 8 concludes the paper.

2 Requirements

In this section we specify the requirements of the ideal autonomous robotics DSL. Some of the requirements deal with the DSL itself while some are more related to its implementation and especially to how it will be compiled and run. In this paper we will report on which requirements were achieved and which ones were left for further study:

1. **Ease of use.** Using the DSL should be in the reach not only of programming experts but of robotics experts and ideally of mere robotics users.
2. **Specification of component-based robotic architectures.** Assuming that most robotics software is nowadays component-based, the DSL must allow the specification of component-based architectures of autonomous robotic systems.
3. **Neutrality regarding architectural styles.** The DSL must not impose a specific robotic architectural style (deliberative, reactive, hybrid, ...).
4. **Multiple, heterogeneous target platforms.** As long as the expected components are provided and conform to the architecture, the latter must be executable on robots or on a simulator. In addition, it must be possible to run some components of a given architecture on one platform and other components on another.
5. **Target-platform independence.** Even though target-platform independence is difficult to achieve, the DSL should be as independent as possible of the specificities of the execution platforms.
6. **Smooth evolution of the supported platforms.** Code generation should be as agile as possible so that supporting a new platform can reuse common transformations. Similarly, supporting a new version of a target platform should capitalize as much as possible on the previous implementation.

7. **Smooth evolution of the DSL.** Ideally it should be possible to change at least some superficial aspects of the DSL without having to build a completely new implementation.
8. **Reasoning.** It should be possible to reason on the architecture especially to check non-functional properties such as real-time constraints or energy consumption.

3 RobotML **Domain Specific Language**

This work aims at providing a domain specific language (and related tools like editors, model validation, code generators) suitable to specify missions, environments and robot behaviours that will be specified by robotics designers. Independently from the target platform, the DSL will help the robotic system designers to define:

– The system's architecture (i.e. its internal structure). In fact, the DSL will ease the definition of specific robotic architecture (reactive, deliberative, hybrid) and specific components that form the architecture (sensors, actuators, planners, mapping, etc.).
– The communication mechanisms between components (sending/receiving of event notifications and data).
– The behaviour of robotic components that form the system's architecture.

The main design entries of the DSL are the robotic ontology [3] developed in the frame of the French research project Proteus[1] and a state of the art study on languages and tools for robotic systems. The latter has helped to identify the requirements presented in the previous section. The former has helped to build the domain model. Once the domain model is built, we implemented the DSL as a UML profile (i.e. extension of the UML meta-model). Then we developed the graphical modeling tool as an extension of the Papyrus modeling tool[2]. In the following, we justify the choice of using a robotic ontology to build the domain model of the RobotML DSL, and then we present the domain model (i.e. meta-model) of the DSL.

3.1 Rationale of Using an Ontology during the Design of RobotML

Ontologies are mainly used in Artificial Intelligence and Semantic Web communities to represent knowledge, as a set of concepts and relationships of a domain. Several works have already used ontologies for their semantic or structural synergy with DSLs [7,11]. In our case, the main benefit from using the ontology in the design process of the DSL is *reusing a robotics experts knowledge base to enrich the DSL domain model* [6]. From our experience, using an ontology has benefits as well as drawbacks. In fact, reusing an ontology facilitates the DSL designers task by providing a set of concepts that are specific to the

[1] http://www.anr-proteus.fr/?q=node/111
[2] http://www.eclipse.org/modeling/mdt/?project=papyrus

robotic domain and that can be directly reused to define the DSL meta model. Examples of those concepts include `Robot`, `SensorSystem`, `ActuatorSystem`, `LocalizationSystem`. On the other hand, a lot of concepts are not useful for defining the DSL domain model. For example the concept of `Interaction` is not useful in the DSL, we have instead used the concepts of `Port` and `Connector` to capture the concept of an interaction. So having an ontology as a design entry implies that the DSL designer will filter the ontology concepts that are useful for the DSL domain model. This filtering task is not straightforward, specially when the DSL designer does not master the ontology language and editing tools.

3.2 Domain Model

The domain model of a given DSL defines the concepts of a language and their relationships. In this section, we present the domain model of the RobotML DSL based on the requirements defined in the previous section and the knowledge base provided by the ontology. We have used UML class diagrams to represent the domain model. The packages structuring the RobotML domain model and relationships between them are presented in Fig. 1.

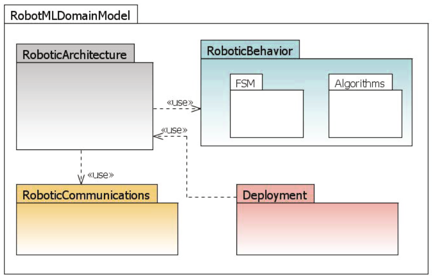


Fig. 1. RobotML Domain Model

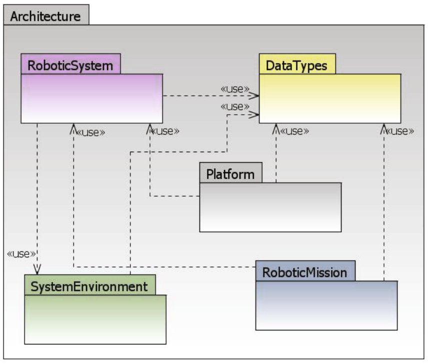


Fig. 2. RoboticArchitecture Package

Architecture. Fig. 2 shows a general view of the RobotML architecture domain model package. This package contains five sub-packages:

1. The *robotic system* package describes the concepts that help define and compose a robotic system. The `System` concept corresponds to the `Component` concept (as found in the component-based approach). The term *system* is more appropriate to describe a robotic component, this is due to the fact that `System` is more meaningful for a robotician than `Component`. Fig. 3 some of the concepts defined in this package. A `System` is composed of properties, ports and connectors. Properties could be either the system's subsystems or attributes (for storing configuration values for example). Ports and connectors allow systems to interact. This package includes also the robotic specific concepts such as sensors and actuators.

2. The *system environment* package defines the concepts composing the environment where robots evolve, since we not only model the robotic system but also its environment (for example for simulation purpose).
3. The *data types* package defines the types of data that will be exchanged between robotic components, between algorithms, etc.
4. The *robotic mission* package describes the concepts that are needed to define an operational mission and which are used by components of the architecture performing it.
5. The *platform* package defines the concept of platform which represents the execution environments for the robotic system (i.e. robotic middleware, robotic simulator).

Communications. Communications between robotic systems formalize data exchange and service calls between them. Communications are refined through the aspect of ports and connectors. A port formalizes an interaction point of a system. `Port` is an abstract concept that is refined through two concepts. On the one hand, we define the concept of `DataFlowPort` which is related to the publish/subscribe model of communication. `DataFlowPort` enables dataflow-oriented communication between systems, where messages that flow across ports represent data items. On the other hand, we define the concept of `ServicePort` that supports a request/reply communication paradigm, where messages that flow across ports represent operation calls.

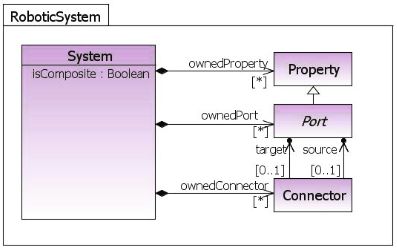


Fig. 3. RoboticSystem Package

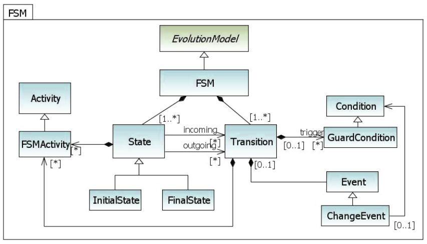


Fig. 4. Details of the FSM Package

Behaviour. The concept of `EvolutionModel` is used to describe the behaviour of the robotic system. An evolution model can be defined using algorithms or finite state machines. Fig. 4 shows the `FSM` concept inheriting from `EvolutionModel` and composed of states and transitions. `States` are composed of `FSMActivities`, meaning that activities can be executed during a `State`. `Transitions` are also composed of `FSMActivities`, meaning that if a transition is fired, an activity can be executed as an effect. An activity is a behaviour that is specified using an algorithm.

Deployment. This package specifies a set of constructs that can be used to define the assignement of a robotic system to a target robotic platform (a middleware or a simulator). The deployment is important because it feeds code generators with the information on which platform the system will be executed.

4 The RobotML Toolchain

RobotML provide a toolchain for robotic development from modeling to software simulation and deployment on real robots. This toolchain, illustrated in Fig. 5, is based on the Eclipse Modeling Project[3] that supports model-driven engineering approach. We have used Papyrus and Acceleo[4] plugins integrated to Eclipse for modeling and code generation.

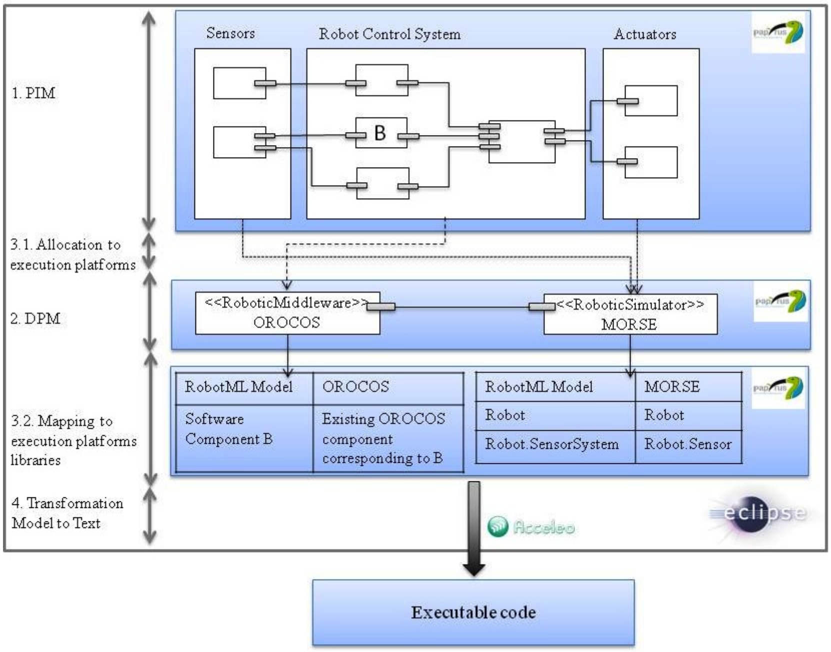

1. PIM
Sensors
Robot Control System
Actuators
B
3.1. Allocation to execution platforms
2. DPM
<<RoboticMiddleware>> OROCOS
<<RoboticSimulator>> MORSE
3.2. Mapping to execution platforms libraries
RobotML Model
OROCOS
Software Component B
Existing OROCOS component corresponding to B
RobotML Model
MORSE
Robot
Robot
Robot.SensorSystem
Robot.Sensor
4. Transformation Model to Text
Acceleo
eclipse
Executable code

Fig. 5. The RobotML toolchain

The RobotML user starts by designing a model (PIM, Platform Independent Model) of a specific scenario where sensors, actuators and the control system of the robots are represented. The next step is a definition of a deployment platform model (DPM), which consist of a set of robotic middlewares and/or simulators connected with each other to form an execution platform for the PIM. Then,

[3] http://www.eclipse.org/modeling/
[4] http://www.acceleo.org/

after the validation of the model, the user defines a deployment plan where he/she chooses to which robotic middleware (OROCOS-RTT[5], RTMaps[6], Urbi[7] or Arrocam[8]) and to which simulation engine (MORSE[9] or CycabTK[10]) code will be generated. The deployment plan is built in two steps:

– First, the designer allocates the components of the PIM to the modeled execution platform parts. For example, the control system of the robot can be allocated to a robotic middleware and sensors/actuators can be allocated to a robotic simulator. The designer can also specialize the robotic system components by setting some of their properties to match the specificities of runtime platforms.
– Second, the designer has the possibility of reusing existing components already deployed in the component library of the target robotic platform. In fact, a mapping to target platforms component libraries can be established by the user.

Finally, from the deployment plan information, the code is automatically generated. Code generators from the DSL to the aforementioned middlewares and simulators have been developed in the frame of the Proteus project.
Overall, the RobotML DSL provides a common ground for designing and implementing component-based robotic systems. We illustrate this in the next section with a case study.

5 Case Study: Urban Challenge

In order to validate the proposed DSL on industrial examples, several case studies (called challenges) have been designed by the Proteus project partners. In this section, we will focus on the Urban Open-Access Robotic Platform[11] challenge that deals with the problem of intelligent transportation systems (autonomous cabs). In the following, we present the urban challenge model in accordance with the four main parts of the proposed DSL: architecture, communication, behaviour and deployment then code generation.

5.1 Modeling

Using the RobotML DSL, we have represented the modules of the challenge in a component-based model.

[5] http://www.orocos.org/
[6] http://intempora.com/
[7] http://www.urbiforge.org/
[8] http://effistore.effidence.com/
[9] http://www.openrobots.org/wiki/morse/
[10] http://cycabtk.gforge.inria.fr/
[11] http://www.anr-proteus.fr/?q=node/64

Architecture. Fig. 6 illustrates the global architecture of the urban challenge model. The control system consists of controller, trajectory planning, localization and obstacle detection components. It defines the behaviour of the robot during its mission. The control system of the robot takes inputs from sensors (*LIDAR3D, Odometery, RTK_ GPS_ IMU* and *Front Camera*) and sends commands to actuators (*Brake, Steering* and *Motor*) through data flow communications. The `Localization` component calculates the position of the robot and returns the deviation of the robot with respect to the trajectory it must follow. `Trajectory Planning` computes the trajectory to follow from the current position of the robot in comparison with its goal (the goal position is initially specified by the user). `Obstacle Detection` is defined to deal with dynamic changes in the environment. It sends information to `Control` whether a new obstacle is detected. The `Control` component aims at transforming the received data from the components `Localization`, `Trajectory Planning` and `Obstacle Detection` into commands to `Actuators` components, which represent the actuators of the robot. A snapshot of the Papyrus-based modeling environment is shown in Fig. 7. At the right side of the component definition diagram, we can see the customized palette that contains RobotML concepts used for defining the aforementioned robotic components.

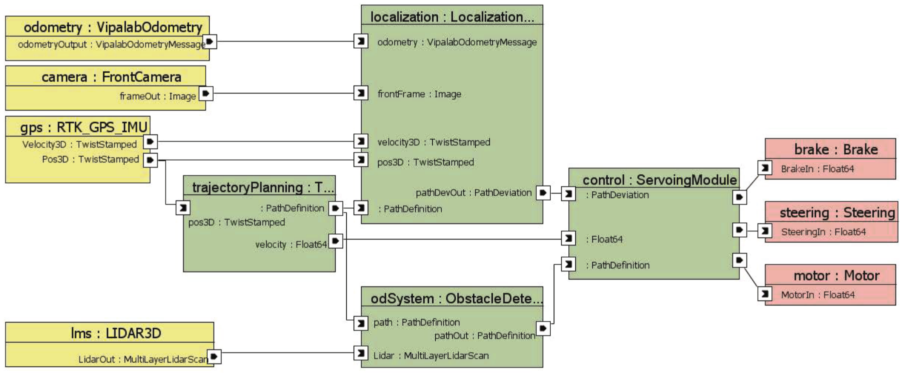
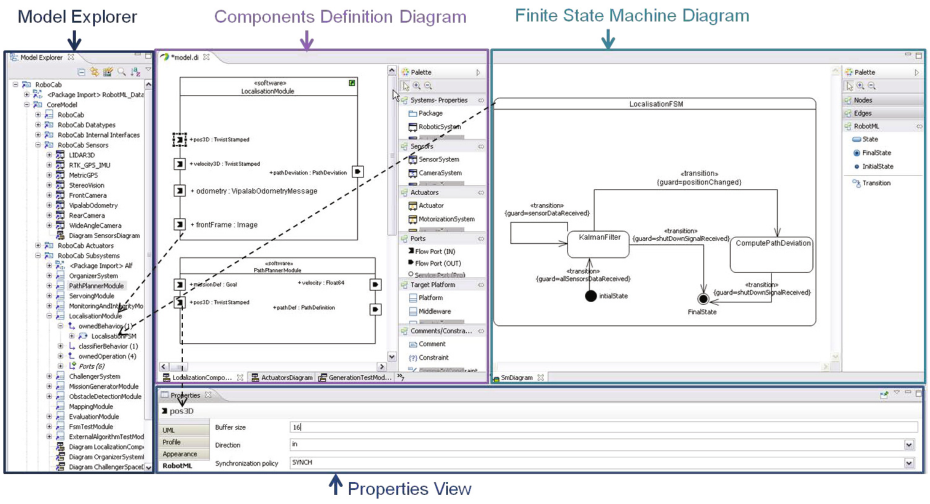


Fig. 6. Simplified urban challenge architecture diagram

Communications. In Fig. 6, components are connected through *Data Flow* ports. Let us take for example the components *Localization* and *RTK_ GPS_ IMU*. The synchronization policy for data exchange (synchronous/asynchronous mode) between these components is specified in addition to the buffer size for data storage. Those information are specified by setting ports attributes in the properties view of the modeling environment (Fig. 7).

Bevahiour. Components (except sensors and actuators) have a behaviour specified by a state machine or an algorithm (see section 3.2). At the right side of the modeling environment (Fig. 7), the state machine diagram of the component `Localization` is shown. The first state is the Kalman filter which handles data sent by sensors and returns an estimation of the pose of the robot. If the position of the robot changes (guard:positionChanged), the state *ComputePathDeviation* is activated and the computed deviation is returned to the controller component.

Deployment. In the case of the urban challenge, we have modeled an execution platform that contains a robotic middleware, namely OROCOS, communicating with a robotic simulator, namely Morse. The deployment plan contains allocations of sensors (yellow components in Fig 6), actuators (red components in Fig 6) to MORSE and allocations of the control system (green components in Fig 6) to OROCOS.



pos3D

Buffer size	16
Direction	in
Synchronization policy	SYNCH

Fig. 7. The RobotML modeling environment

5.2 Code Generation

The RobotML toolchain integrates several code generators defined by Proteus partners to several robotics middlewares and simulators engines (cf. section 4). The process of translating a model to code is to perform Model to Text (M2T) transformations from the DSL to text artifacts (source files, configuration files, etc.) needed to create an executable application. Starting from the model of the urban challenge and the deployment plan defined in the model, users can generate code for several platforms.

6 Discussion

RobotML The main objective of the RobotML DSL was to propose to the robotics community a domain specific language which facilitates the development of robotics applications. In this context a set of requirements was identified in Section 2. In this paper, the RobotML DSL presented an answer to achieve these requirements. In this section we report on which of these requirements were achieved by the RobotML DSL and which ones were left for further work.

1. **Ease of use** Thanks to RobotML, DSL users can model the components of their missions without mastering programming languages of robotics platforms. In fact, platforms details are hidden to the DSL users and concepts used in RobotML are based on robotics ontology.
2. **Specification of component-based robotic architectures** As presented in Section 3, the RobotML DSL includes the architecture part which gathers the concepts of *System*, *Port*, *Connector*. The latters allow the specification of component-based robotic architectures. For instance, Fig. (6) shows an example of such architecture.
3. **Neutrality regarding architectural styles** Most robotic systems software architectures are based on components, the only architecture taken into account in RobotML is component-based. Consequently, any robotic architecture (hybrid, deliberative, etc) can be specified using an approriate combination of components. For example, the used architecture in the scenario of Fig. 6 is hybrid.
4. **Multiple, heterogenous target platforms** Thanks to the use of ROS, components deployed into robotics platforms can easily communicate with components deployed into simulators or directly with real robots. Thus, the code generated from RobotML can be executable on both real and simulated robots.
5. **Target-platform independence** Using RobotML and thanks to the abstract concepts of the DSL, the architecture model of the robotic application is independent from the target platform (PIM).
6. **Reasoning** RobotMLenable roboticist to model some non-functional properties of the system notably timing properties for software components (period, deadline, WCET, priority). Such timing properties will feed schedulabilaty analysis tools e.g. Cheddar [9]. Another types of reasoning could be considered for robotic systems specified with RobotML given the extension of the language to integrate the adequate non functional properties modeling.

The requirements *smooth evolution of the supported platforms* and *smooth evolution of the DSL* were left for future work.

7 Related Work

At the moment, there are not that many proposals to use MDE (Model Driven Engineering) in the context of robotic systems. One of the first initiatives to promote this approach was Blanc et al. [2] who applied MDE to develop control ssoftware for an AIBO robot.

SmartSoft [8] combines a service-oriented component-based with a Model-Driven Software Development approaches. The SmartSoft[12] component model relies on a small and fixed set of communication patterns, e.g., request/response and publish/subscribe, that define the semantics of the interface (externally visible ports) of components. The component model is represented by a meta-model called

[12] http://smart-robotics.sf.net/

SmartMARS (Modeling and Analysis of Robotic Systems). Like RobotML, a concrete implementation of this meta-model has been done as a UML profile and the proposed Eclipse-based SmartMDSD model-driven software development toolchain is based on Papyrus as well. But unlike RobotML, the SmartSoft approach does not provide the possibility to describe components behaviour at the same abstraction level than components specification level.

BRICS (Best Practice in Robotics) is an ongoing FP7 EU funded project[13] that also promotes the model-driven engineering (MDE) approaches in order to solve robotic software engineering issues. Among all the activities done in the project, a features-based model toolchain was proposed in order to reflect variabilities in robotic systems [5] and a meta-model called BCM was defined for describing a minimal component model[14] suitable for code generation for multiple targets (ROS, OROCOS-RTT). Unlike RobotML, BRICS meta-model does not enable specification of composability, component's behaviour and robotic specific components.

The 3-View Component Meta-Model (V^3CMM) [1] relies on the use of the OMG Meta-Object Facility (MOF) instead of using UML. It aims to provide designers with an expressive yet simple platform-independent modeling language for component-based application design. V^3CMM is aimed at allowing designers (1) to model high-level reusable components, including both their structural and behavioural facets (modeling for reuse); (2) to build complex platform-independent designs up from the previous components (modeling by reuse); and (3) to automatically translate these high-level designs into lower level models or into different implementations, isolating functionality from platform details. Compare to RobotML, it is worth noting that although V^3CMM has been used mainly in robotics, it does not contain any specificities about this domain.

8 Conclusion

In this paper we have reported on the RobotML domain-specific language and toolchain. RobotML is easier to use than the targeted robotic execution or simulation platforms because low level details have been hidden behind easier to manage abstractions. Not all of these abstractions are directly related to robotics and some do relate to defining component-based architecture but a sizeable amount of low-level programming knowledge has been put into the code-generation transformations. Early usage reports suggest that even though the overall development time of a robotic application using RobotML has not significantly decreased, using RobotML induces the following advantages:

– more time is spent on design than on dealing with low level details,
– the architecture is made explicit,
– switching to a new target platform is much easier.

[13] http://www.best-of-robotics.org/

[14] http://www.best-of-robotics.org/pages/publications/BRICS_Deliverable_D4.1appendix.pdf

The RobotML DSL and toolchain have been designed in the context of the ANR Proteus projet. This project funded by the French research agency, gather 14 academic and industrial partners. The RobotML toolchain will be available with an open-source licence in the future.

References

1. Alonso, D., Vicente-Chicote, C., Ortiz, F., Pastor, J., Alvarez, B.: V^3CMM: a 3-View Component Meta-Model for Model-Driven Robotic Software Development. Journal of Software Engineering for Robotics 1(1), 3–17 (2010)
2. Blanc, X., Delatour, J., Ziadi, T.: Benefits of the MDE approach for the development of embedded and robotic systems. In: Proceedings of the 2nd National Workshop on "Control Architectures of Robots: from Models to Execution on Distributed Control Architectures", CAR 2007 (2007)
3. Dhouib, S., Du Lac, N., Farges, J.L., Gerard, S., Hemaissia-Jeannin, M., Lahera-Perez, J., Millet, S., Patin, B., Stinckwich, S.: Control architecture concepts and properties of an ontology devoted to exchanges in mobile robotics. In: 6th National Conference on Control Architectures of Robots (2011)
4. Gerard, S., Babau, J.P., Champeau, J.: Model Driven Engineering for Distributed Real-Time Embedded Systems. Wiley-IEEE Press (2005)
5. Gherardi, L., Brugali, D.: An Eclipse-based Feature Models Toolchain. In: Proc. of the 6th Workshop of the Italian Eclipse Community (Eclipse-IT 2011) (2011)
6. Lortal, G., Dhouib, S., Gérard, S.: Integrating Ontological Domain Knowledge into a Robotic DSL. In: Dingel, J., Solberg, A. (eds.) MODELS 2010. LNCS, vol. 6627, pp. 401–414. Springer, Heidelberg (2011)
7. Morin, B., Perrouin, G., Lahire, P., Barais, O., Vanwormhoudt, G., Jézéquel, J.-M.: Weaving Variability into Domain Metamodels. In: Schürr, A., Selic, B. (eds.) MODELS 2009. LNCS, vol. 5795, pp. 690–705. Springer, Heidelberg (2009)
8. Schlegel, C., Steck, A., Lotz, A.: Model-driven software development in robotics: Communication patterns as key for a robotics component model. Introduction to Modern Robotics (2012)
9. Singhoff, F., Legrand, J., Nana, L., Marcé, L.: Cheddar: a flexible real time scheduling framework. In: Proceedings of the 2004 Annual ACM SIGAda International Conference on Ada, SIGAda 2004, pp. 1–8. ACM, New York (2004)
10. Steck, A., Lotz, A., Schlegel, C.: Model-driven engineering and run-time model-usage in service robotics. In: Proceedings of the 10th ACM International Conference on Generative Programming and Component Engineering, GPCE 2011, pp. 73–82. ACM, New York (2011)
11. Walter, T., Ebert, J.: Combining DSLs and Ontologies Using Metamodel Integration. In: Taha, W.M. (ed.) DSL 2009. LNCS, vol. 5658, pp. 148–169. Springer, Heidelberg (2009)

A Java vs. C++ Performance Evaluation: A 3D Modeling Benchmark

Luca Gherardi, Davide Brugali, and Daniele Comotti

University of Bergamo, DIIMM, Italy
{luca.gherardi,brugali,daniele.comotti}@unibg.it

Abstract. Along the years robotics software and applications have been typically implemented in compiled languages, such as C and C++, rather than interpreted languages, like Java. This choice has been due to their well-known faster behaviors, which meet the high performance requirements of robotics. Nevertheless, several projects that implement robotics functionality in Java can be found in literature and different experiments conduced by computer scientists have proved that the difference between Java and C++ is not so evident.

In this paper we report our work on quantifying the difference of performance between Java and C++ and we offer a set of data in order to better understand whether the performance of Java allows to consider it a valid alternative for robotics applications or not. We report about the execution time of a Java implementation of an algorithm originally written in C++ and we compare this data with the performance of the original version. Results show that, using the appropriate optimizations, Java is from 1.09 to 1.51 times slower than C++ under Windows and from 1.21 to 1.91 times under Linux.

1 Introduction

Robot software systems are concurrent, distributed, embedded, real time, and data intensive. Computational performance is a major requirement, especially for autonomous robots, which process large volumes of sensory information and have to react to events occurring in the robotics operational environment.

In order to meet performance requirements, robotics algorithms have been typically implemented in C and C++. Robotics developers in fact have always considered C++ significantly faster than Java. Despite that, the idea of using it in robotics is not really new: it has been followed in several projects (see section 2) and recently Willow Garage and Google have started a project for developing a Java-based porting of ROS [4].

In this paper we report our work on the comparison of performance between Java and C++. Our goal is to quantify this difference and to offer a set of data in order to better understand whether the performance of Java allows to consider it a valid alternative to C++ or not. For this purpose we implemented in Java a well known algorithm originally written in C++ and we executed a comparison study. The chosen algorithm is the Delaunay triangulation and its implementation comes from the OSG library[2]. It was developed in the computer vision field but it is typically used also in robotics for reconstructing environment surfaces from a set of 3D points. The algorithm is well suited for the purpose of our study because it stresses several critical points of the programming languages performance such as: (a) the frequent access to the memory for operating on dynamic size array (massive use of the garbage collector) and (b) the frequent evaluation of logical conditions.

Although in the computer science domain many comparison studies has been proposed, we considered our test interesting because we implemented and executed the algorithm with a newer and improved version of the Java JDK. Indeed the current Java Virtual Machine (JVM) offers a new compiler, which greatly improves the performance of Java with respect to the older versions.

The paper is structured as follows. Section 2 reports about the Java related projects in robotics and presents a survey on the differences of performance of the two languages. Section 3 illustrates a performance comparison case study. We present a Java-based implementation of a mesh generation algorithm originally written in C++ and report several information about the execution time of both the Java and the C++ versions. Finally section 4 draws the relevant conclusion.

2 Java for Robotics

Java is an object oriented programming language and it was intended to serve as a new way to manage software complexity. It offers to its users a set of software libraries and specifications, which allow the designing and the deploying of cross-platform applications. Java is used in different application domains such as enterprise resource planning (ERP) and web servers (e.g. JSP). It is widely spread also on mobile phones and embedded devices. This section presents a set of robotics project developed with Java and a survey on several performance comparisons between Java and C++.

2.1 Robotics Java Projects

During the 2011 Google I/O the researcher of Willow Garage and Google presented a new project that aims to develop a pure Java implementation of ROS [4]. By means of this project Google and Willow Garage aim to boost the development of advanced Android applications for robotics and easiness the access to the cloud computing for reducing the cost of the robotics hardware.

In [11] the integration of Matlab in a distributed behavioral robotics architecture is presented. The architecture is completely implemented in Java and leverages on the Jini platform for distributed object registration, lookup and remote method invocation. The Matlab integration is realized by means of JMatLink and allows the invocation of Matlab scripts and the access to the Matlab workspace as a distributed object. The authors present as case study a multi-robot mines detection. In [17] a team from Lund University demonstrated that it is feasible to develop a motion control system entirely in Java. They designed an application that takes a picture of a person and controls a pick and place robot in order to draw on a paper the result of the shooting. The software and the motion controller guarantee the respect of the real time constraints by means of Java RTS. In [16] a real-time control for a remote manipulator over a local area network or over internet is presented. The developers implemented both the control system and the teleoperation of the robot in Java. In [9] an autonomous motion planning system completely developed in Java is developed. The application allows the user to set up the working environment though a graphical interface and offers the functionalities of collision detection, obstacle avoidance, free-paths generation and selection of the shortest path. Finally in [14] an application for controlling robots through the World Wide Web is implemented. The software is designed for dealing with low bandwidth and high latency and allows the operator to control the robot from any computer connected to the web.

2.2 Java versus C++

One of the main differences between Java and C++ is that the first was born as an interpreted language while the second as a compiled language. Compiled languages are translated into machine code trough a compiler. This process generates a file that can be directly executed by the CPU. Interpreted languages are compiled in a platform independent language (bytecode), which can be executed only by means of an interpreter (e.g. JVM). Hence, programs written in C++ (compiled language) are platform dependent and must be compiled for every computing platform before the first execution. Java programs instead are translated into bytecode only once and can be used on different platforms but have to be interpreted at every execution (the JVM is platform dependent). For this reason interpreted languages are in general more flexible and portable than compiled languages but at the same time slower.

In order to improve the performance, in 1998 Java 1.2 was released with a new feature called Just-In-Time compiler (JIT)[1]. JIT is integrated into the JVM and it is in charge of translating the Java byte code into binary code. Each method is translated only when it is called for the first time. Thanks to this improvement the execution time decreases and the code is again portable.

Many comparisons between C, C++ and Java were documented in literature. From this point we call "*Java*" the version optimized with JIT and "*interpreted Java*" the original version. In [19] the execution times of C++ and Java are compared. The authors tested the execution of four sorting algorithms, two of $O(n^2)$ complexity (bubble sort and insertion sort) and two of $O\left(n \cdot log\left(n\right)\right)$ (recursive quick sort and heap sort), on four integer data sets of different sizes. The results demonstrated that C++ was much faster than pure interpreted Java (from 11 to 20 times) and only from 1.45 to 2.91 times faster than Java (version 1.3). In [6] a set of polynomial multiplications was computed and executed using the three languages. The results showed that Java completed the operations faster than standard C (mean of 21%) but in average 2.61 times slower than C++. In [7] the executions of the Linkpack benchmark were compared for Java and standard C. This benchmark was introduced by Jack Dongara and measures how fast a computer solves a dense N-by-N system of linear equations. The

Table 1. Results summary (OS: W=Windows, L=Linux, S=Solaris, M=MacOs)

Paper	Test	OS	Java vs. C	Java vs. C++	JDK	C/C++ compiler
[19]	Sorting alg.	W	—	1.45 - 2.91	Sun 1.3	Borland v. 5.5
[6]	Polynomial mult.	S	0.79	2.61	Sun 1.2b5	Sun Workshop C 4.2
[7]	Linkpack bench.	W-S	2.25	—	Sun 1.2b4	—
[18]	Method call	W-S-L-M	—	1 clock slower	Please refer to the paper	
[12]	Int and float div.	W	—	~ 1	Sun 1.1.5	Visual C++ 5.0

results showed that for a 1000 x 1000 system Java was 2.25 times slower than C. In [18] Ruolo evaluated the Java method call performance. Different tests with a different numbers of parameters showed that Java was only one clock cycle slower than C++. The same tests also highlighted that the time needed for allocating user defined objects on the heap was roughly equivalent. However C++ also uses the stack for allocating temporary object and in this case it was from 10 to 12 times faster than Java, which uses only the heap. Interesting conclusions were reported by Mangione [12]. He tested the repetitive execution of simple operations like integers and float divisions and showed that Java was as fast as C++. As summarized in table 1 all the papers report that, since the introduction of the Just-In-Time compiler, Java is only 1.45-2.91 times slower than C++ (Column OS: operating system, columns 4-5 execution time ratios).

Since these studies demonstrated how the execution of simple operations in Java is more or less as fast as in C++, one factor that could influence the total execution time of a Java program is the Garbage Collector (GC). However [10] showed that Java GC is as fast as a *malloc/free* operation in C++. In fact when a program executes a *malloc* operation, the allocator looks for an empty slot of the right size and returns a pointer to a random place in the memory. In Java instead the allocator use the bits of memory adjacent to the last bit it used. Hence it doesn't need to spend time looking for memory. So the amount of time used for the garbage collector is comparable to the amount of time that the allocator uses in C++ for finding free memory slots.

Finally other interesting results are documented in [15]. The same program was implemented by 40 different programmers in different languages (24 in Java, 11 in C++ and 5 in C). The experiment compared not only the performance of the languages but also the differences between the implementations in the same language (interpersonal differences). The results demonstrated that Java was 2 times slower than C++ and that the interpersonal differences were much larger than the average difference between Java and C++. That means a well written Java program could be as efficient as an average C++ program.

3 A Performance Comparison Case Study: The Delaunay Triangulation

Visual sensors such as laser scanners acquire information on the environment geometry in form of a point cloud: a set of vertices in a 3D coordinate system. Each one of these vertices corresponds to a point on the surface of one of the objects present in the environment. In order to reconstruct the surface of these objects the vertices have to be connected. This problem is called mesh generation and one of the possible solutions consists of the Delaunay triangulation [8].

Delaunay's algorithm connects the set of points in such a way to build a series of triangles which respect the following property: for all the set of points there is no point which lies inside the circumcircle of any triangle. The triangulation result is unique except if more than three vertices stand on the same circumference. In this case more than one solution exist.

In this section a comparison between a C++ and a Java version of the Delaunay triangulation will be reported. We refactored in Java a C++ implementation coming from the OSG libraries[2]. The implementation of this triangulation algorithm is based on the Bowyer-Watson method, which works in the plane space. It iterates all the points of the cloud and for each one executes two main steps: identifying the triangles whose circumcircle contain the current analyzed point and then building a new set of triangles, which respect the Delaunay condition. This algorithm allows to process point clouds in 3D space but realizes only a triangulation in the plain space therefore the Z coordinate is ignored. It should be noted that the implemented algorithm does not provide a constrained Delaunay triangulation. For this reason, during the timing and the comparison of the computation time, we have excluded the constraints also in the OSG version.

Both the OSG and our implementations receive as input the point cloud in form of a collection of vertices. The OSG implementation defines a custom class, *Vec3Array*, which is a specialization of the class *MixinVector* (*MixinVector* allows inheritance to be used in order to easily emulate derivation from *std::vector* but without introducing undefined behaviour through violation of virtual destructor rules [3]). Hence, *Vec3Array* defines a vector of *Vec3* instances, which are triplets of float data types. Our implementation instead uses the Java *ArrayList*. We chose this collection because it is the fastest of all the collections provided by the Java framework for what regards the operations of inserting, iterating and sorting [20], and because its performance are comparable with the one of Java *Vector*. On the other side *ArrayList*, like C++ *std::vector*, is not as well efficient when it has to perform the operation of removing elements in random position. In order to better understand how much the overhead between Java and C++ is due to these data structures, we compared the performance of the *Vec3Array* and *ArrayList* collections. We executed a set of tests on the most used operations during the triangulation algorithm:

– Insertion. We executed 10000 and 100000 insertions of objects (instances of class that represent the 3D points) at the end of the 2 collections. We chose the values of 10000 and 100000 because they are the maximum orders of magnitude of the collection sizes used in the tests of the Delaunay algorithm.
– Removal. We executed the complete clearing of collections of 10000 and 100000 objects. We removed one element at time. In order to evaluate the performance in the worst case, the object at the head of the collection was chosen to be deleted during each iteration.

– Sorting. We invoked the sorting function on collections of 100 and 1000 points generated randomly. We chose these size values, which are lower with respect to the tests of the other operations, because in the Delaunay algorithm the sorting is always executed on little collections (see more details below).

Each test was executed 50 times and then the mean time was computed. We executed them on a 3.2 GHz Intel Pentium 4 processor with 1GB of RAM under Ubuntu 10.4 (OpenJDK Runtime Environment v. 1.6.0_20 and GCC v. 4.3.3). Results are reported in table 2 where times are expressed in milliseconds and regard the execution of all the n operations. *ArrayList* is faster than *Vec3Array* during the insert and the remove operations, whereas it takes much time to compute the sorting because of the used algorithm. Indeed the method for sorting Java collections uses a modified *merge-sort* algorithm [5], which offers guaranteed $O(n \cdot log(n))$ performance. The sorting algorithm provided by the C++ STL library instead uses the *introsort* algorithm whose worst case complexity is $O(n \cdot log(n))$.

Table 2. Times report - Collection comparison

	Insert		Remove		Sort	
N. of elements	10000	100000	10000	100000	100	1000
Java	1.30	6.33	46.67	5168.21	0.13	0.42
C++	1.51	11.27	275.82	27799	0.02	0.40
Java vs C++	0.86	0.56	0.17	0.19	6.5	1.05

We have also analyzed time required for the evaluation of logical conditions. Four tests were executed, taking into account the following logical conditions:

– Simple logical proposition ($var{=}{=}true$)
– Disequation ($a < b$). (Most evaluated condition in the case study, see eq. 1)
– Logical disjunction of two disequations ($(a < b)||(a > c)$)
– Logical conjunction of two disequations ($(a > b)\&\&(a < c)$)

Each evaluation was executed 10000 times and each test was repeated 50 times. Table 3 reports average times of the tests in milliseconds. We used a boolean variable (initialized false and its value was changed each execution ($var = !var$)) in the first test and float variables (initialized with a constant values) in the others. As can be seen, Java is always faster than C++, except for what regards the evaluation of simple logical proposition.

3.1 The Implementation Details

The two implementations compute the triangulation according to the same steps, which are described in the following list.

1. Initialization. The Initialization step consists of the setting up and the sorting of the input point cloud according to their coordinates. Then four new points are inserted in order to surround the plain point cloud. These four

Table 3. Times report - Logical conditions evaluation comparison

	Prop.	Diseq.	Disj.	Conj.
N. of elements	10000	10000	10000	10000
Java	0.187	0.084	0.103	0.093
C++	0.039	0.262	0.452	0.290
Java vs C++	4.79	0.32	0.23	0.32

points are used to build two main triangles (super-triangles), such that the plain point cloud lies inside their area. These triangles are stored in a collection, that we'll call *trianglesList*. The collection data structure was chosen accordingly to the operations that occur more often, indeed the *trianglesList* is subject to several iterations, insertions and removal. As shown in [20], *ArrayList* is the list of all the available lists in the Java framework that perform insertion, iteration and random access in the fastest way. Although removing objects from *ArrayList* requires a long time, insertions and iterations occur more often than remove operations; hence we decided to use *ArrayList* for implementing the *trianglesList*.

2. Iteration. During the iteration, each point is considered and is compared to the triangles contained in the *trianglesList*. First the condition 1 is checked ("*point*" stays for the current point and "*tri.circ* for the circumcircle of the current triangle).

$$point.X - tri.circ.X > tri.circ.radius \tag{1}$$

   – If it is true, the current triangle is removed from the *triangleList* and will not be more considered because the current point and also the following ones surely don't lie in the circumcircle of the current triangle (i.e. the triangle respects the Delaunay condition for all the points and it is part of the final mesh). This is guaranteed by the initial ordering of the points.
   – Otherwise, we must further investigate if the current point effectively lies in the circumcircle of the current triangle. In case it is true the Delaunay condition is not respected. Therefore the edges of the triangle are added to a specific *ArrayList* (called *edgeSet*) and the current triangle is deleted. Whereas, if the Delaunay condition is respected, the next triangle is considered. It has to be noted that the *edgeSet* collection has been implemented as an *ArrayList* because it is sorted many times during the triangulation algorithm. Hence, the usage of *Collections.sort* method and *ArrayList* is the Java solution that allows us to save time and increase performance in the best way. In our tests the maximum size of the *edgeSet* collection was never greater than 100.

When the whole *trianglesList* has been scanned, new triangles are constructed from the *edgeSet* collection and added to the *triangleList* (in our tests the maximum order of magnitude of this collection size is 10000). Note that if an edge is shared between two triangles that contain a point, then the edge is not considered. The iteration proceeds until all points have been analyzed, except for the four points created during the initialization.

3. Completion. The four points introduced during the initialization step and triangles having vertices in common with these four points are deleted. If there are degenerate triangles (circumcircle radius equals to 0) they are eliminated too. Finally a return result is built in form of a Mesh.

Since the order of magnitude of the size of the list on which we perform more insertion is 10000 whereas the one of the collection on which we perform the sorting is 100 the time gained in Java for populating the first collection is lost for sorting the second one (see table 2). This suggests that the time spent for managing collections will be more or less the same for both the Java and C++ implementations. The evaluation of logical conditions instead seems to be not important from a performance point of view. In fact the time spent for evaluating conditions on 10000 iterations is much lower than the time spent for populating collections in more or less 100 iterations.

3.2 The Java HotSpot Compilers

The current JVM offer a technology called HotSpot Compiler [13], which works better and faster than the pure JIT compiler. Rather than compiling each method at the first execution, the HotSpot runs the program using an interpreter for a while. During this time, in order to detect the most used and critical methods, the execution is analyzed. The collected information is then used to perform more intelligent optimizations and only the critical methods are actually compiled. This technique is called "*Adaptive Optimization*". It doesn't only produce better performance but it also reduces the overall compilation time. The adaptive optimization is continuously performed so that it adapts the performance to the users' needs.

The Java Platform Standard Edition offers a JVM that comes with two compilers: the Client and the Server versions[1]. The Client compiler is the default one and it has been specially tuned to reduce the start-up time. It is designed for client environment, in particular for applications where there is not the need of continuous computation, for example a GUI. The Server compiler instead is designed for long-running server applications, where the operating speed is more important than the start-up time. This compiler offers an advanced adaptive optimizer and supports many of the optimizations offered by the C++ compilers.

Subsections 3.3 and 3.4 report the tests executed using the client compiler whereas the server compiler is used in the experiments of subsection 3.5.

3.3 Performance Analysis

We executed the algorithm on five point clouds of different sizes: a semi-sphere, a floor, the Oxford Bunny and 2 terrains. Each point cloud was processed 50 times and the execution time was measured; then the average, the standard

[1] Users can specify the compiler by means of the options "-client" and "-server". The tests on the collections and logical conditions were executed with the client compiler.

deviation and the confidence intervals ($1-\alpha = 0.95$) were computed. In the first experiment each triangulation corresponds to a single program invocation, so we executed the program 50 times per point cloud.

Table 4 reports the results of our test on the same PC presented before running Windows XP (Sun Java 6 v. 1.6.0_23 and C++ programs compiled with MinGw v. 3.82 and GCC v. 4.5.0). Mean time, standard deviation and confidence intervals (c_1 and c_2) are expressed in milliseconds. Java vs. C++ is the ratio between the average execution time.

Table 4. Times report - Multiple invocation - Windows - Java Client

		Sphere	Floor	Bunny	Terrain 1	Terrain 2
Number of vertices		642	10000	35947	66049	263169
Java	Mean	70.62	1458.8	3102.2	20344	132926
	Std Dev.	7.89	12.90	52.96	403.44	986.18
	c_1	68.38	1455.1	3087.1	20229	132646
	c_2	72.86	1462.5	3117.3	20459	133206
C++	Mean	7.50	298.13	886.25	3305.9	22908
	Std Dev	7.89	25.61	63.77	144.83	227.06
	c_1	5.26	290.85	868.13	3264.8	22844
	c_2	9.74	305.40	904.37	3347.1	22973
Java vs. C++		9.42	4.89	3.50	6.15	5.80

Referring to the table 4, the triangulation execution time obtained with the first point cloud (sphere) is not very truthful. Indeed Java version is 9.42 times slower than C++ and this value doesn't fit the ratios obtained with the other point clouds. In this case the execution time is very small and so the time required for compiling the code greatly influences the result. Note that the costs required by the compiler have a fixed part, which is the same for each point clouds. Hence the smaller is the point cloud, the greater is the influence of the compilation overhead on the execution time.

3.4 Single Program Invocation

In order to avoid the compilation overhead we decided to change our measuring strategy. We set up a second experiment, where 51 triangulations were measured in a single program invocation. This kind of experiment corresponds to a long run execution, where only the first invocation of the algorithm pays the compiler costs. We discarded the execution time of the first invocation and computed our statistics on the other 50 samples. Table 5 reports the results obtained under Windows and Ubuntu Lucid (both with the same Java and C++ versions presented before). As expected, the average execution times decrease for Java and remain more or less the same for C++.

Under Windows, without the compiler overhead, Java performance are always better than the results reported in table 4, but remain worse than the results

Table 5. Times report - Single invocation - Java Client

		Windows					Linux				
		Sphere	Floor	Bunny	Terr. 1	Terr. 2	Sphere	Floor	Bunny	Terr. 1	Terr. 2
Num. of vertices		642	10000	35947	66049	263169	642	10000	35947	66049	263169
Java	Mean	15.58	1157.2	2487.2	18108	115977	6.66	869.20	2111.3	13478	92301
	Std Dev.	0.50	18.82	20.57	80.53	399.65	1.48	40.83	80.94	106.06	383.57
	c_1	15.44	1151.9	2481.3	18085	115863	6.24	857.60	2088.3	13448	92191
	c_2	15.72	1162.6	2493.0	18130	116091	7.08	880.80	2134.3	13508	92410
C++	Mean	6.56	288.75	914.06	3226	23168	3.23	224.01	750.09	3333.4	24277
	Std Dev	7.79	7.89	11.92	110.36	605.89	0.84	2.35	7.41	62.90	437.73
	c_1	4.35	286.51	910.68	3195	22996	2.99	223.35	747.99	3315.5	24152
	c_2	8.78	290.99	917.45	3257	23340	3.47	224.68	752.20	3351.2	24401
Java vs. C++		2.37	4.01	2.72	5.61	5.01	2.06	3.88	2.81	4.04	3.80

discussed in section 2. Indeed in our tests the execution time ratio between Java and C++ goes from 2.37 to 5.61 against the range 1.45-2.91 reported in table 1. One possible reason is that the triangulation process requires an intensive use of the memory and probably the C++ version leverages the possibility of store temporary objects on the stacks, which is much faster [18].

The table shows that under Ubuntu Java is more efficient than under Windows. Indeed the ratio range goes from 2.06 to 4.04. It should be noted that we executed the tests with both the OpenJDK and the Sun JDK. However in the paper we consider only the first one because the results are almost the same.

Another consideration can be done on the relation between the point clouds sizes and the execution time ratios. The value of the performance ratio doesn't show a linear trending, hence we can assert that for this algorithm there is no correlation between the performance ratio and the input size.

3.5 JVM Server Option

Table 6 report the results obtained using the Server compiler. We executed the tests on the same machine used previously with the same configurations. Of course, only the Java version was tested and the C++ rows report the results of the previous experiments. Despite we are aware of the existence of the optimizations provided by GCC, we didn't work on them. Indeed the default *release* configuration of the OSG libraries is tuned in order to offer the best performance.

The results shows that under Windows the server compiler considerably reduces the average execution time of complex operations. In particular the execution times decrease 3-4 times with respect to the client compiler. The first cloud represents the unique exception. Indeed, in that case the execution is too short and so most of the iterations are executed without optimization. This is the typical case in which the larger start-up time required by the server compiler is not compensated. Regarding the other point clouds, the ratio range goes from 1.09 to 1.51 and so Java is nearly equivalent to C++.

Table 6. Times report - Single invocation - Java Server

		Windows					Linux				
		Sphere	Floor	Bunny	Terr. 1	Terr. 2	Sphere	Floor	Bunny	Terr. 1	Terr. 2
Number of vertices		642	10000	35947	66049	263169	642	10000	35947	66049	263169
Java	Mean	24.96	356.26	999.02	4873.1	30320	20.40	428.5	1253.1	4778.9	29368
	Std Dev.	15.45	20.16	20.40	42.80	230.28	19.58	20.66	56.64	172.29	250.81
	c_1	20.57	350.53	993.22	4861.0	30255	14.84	422.63	1237.0	4729.9	29297
	c_2	29.35	361.99	1004.8	4885.3	30385	25.96	434.37	1269.2	4827.8	29440
C++	Mean	6.56	288.75	914.06	3225.9	23168	3.23	224.01	750.09	3333.4	24277
Java vs. C++		3.80	1.23	1.09	1.51	1.31	6.32	1.91	1.67	1.43	1.21

The results demonstrates that also under Ubuntu the server compiler significantly improves the performance. The same considerations reported above could be applied to the results obtained with the first cloud. Referring to the other clouds the ratio range goes from 1.21 to 1.91 and the average execution times are from 2 to 3 times better than the results obtained with the client compiler.

4 Conclusions and Future Works

In this paper we have described our work on the evaluation of the performance of Java with respect to C++ in robotics applications. The results obtained with the Client compiler, which works better for short-running applications, have shown that Java is from 2.72 to 5.61 times slower than C++. Using the Server compiler, which is best tuned for long-running applications, have instead demonstrated that Java is from 1.09 to 1.91 times slower. These results show that the performance of Java are now better with respect to the tests previously documented in literature and demonstrate that the use of the Server compiler for long run application greatly reduces the execution time.

In addition to the fact that now the performance are not so different with respect to C++, we also have to consider that Java offers a set of interesting features. *Portability*: Java is designed to be platform independent and so Java software is very portable. The low level data types such as integer and float are fully defined in Java specification and aren't platform dependent. *Reusability*: Java comes by default with a lot of common libraries for several purposes. It is also easy to deploy and reuse the developed libraries without sharing source or header files or requiring a specific compiler. *Maintainability*: Java is designed to forbid common bugs such dangling pointers, casting errors, out-of-bounds array, stack overflows, segmentation faults and uninitialized variables.

In conclusion, the results obtained with the server compiler and these important features suggest that Java can be considered a valid alternative to C++. We plan to executes new experiments in order to further confirm this thesis. The tests will regards the communication with external devices (USB, RS-232, ...) and the execution of multi-thread programs.

Acknowledgments. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics). The authors would like to thank all the partners of the project for their valuable comments.

References

1. Just in time compiler, http://en.wikipedia.org/wiki/JIT_compiler
2. Open Scene Graph, http://www.openscenegraph.org
3. Open Scene Graph API reference, http://www.openscenegraph.org/ documentation/OpenSceneGraphReferenceDocs
4. Rosjava - An implementation of ROS in pure Java with Android support, http://code.google.com/p/rosjava/
5. Specifications of java.util.collections, http://docs.oracle.com/javase/6/docs/api/java/util/Collections.html
6. Bernardin, L., Char, B., Kaltofen, E.: Symbolic computation in Java: an appraisement. In: Proceedings of the 1999 Int. Symposium on Symbolic and Algebraic Computation, pp. 237–244. ACM (1999)
7. Bull, J., Smith, L., Westhead, M., Henty, D., Davey, R.: A methodology for benchmarking Java Grande applications. In: Proceedings of the ACM 1999 Conference on Java Grande, pp. 81–88. ACM (1999)
8. Delaunay, B.: Sur la sphere vide. Izv. Akad. Nauk SSSR, Otdelenie Matematicheskii i Estestvennyka Nauk 7, 793–800 (1934)
9. Elnagar, A., Lulu, L.: A global path planning Java-based system for autonomous mobile robots. Science of Computer Programming 53(1), 107–122 (2004)
10. Lewis, J., Neumann, U.: Performance of Java versus C++. Computer Graphics and Immersive Technology Lab, University of Southern California (January 2003)
11. Long, M., Gage, A., Murphy, R., Valavanis, K.: Application of the distributed field robot architecture to a simulated demining task. In: Proceedings of the 2005 IEEE Int. Conference on Robotics and Automation, ICRA 2005. IEEE (2005)
12. Mangione, C.: Performance tests show java as fast as c++. JavaWorld (1998)
13. Meloan, S.: The Java HotSpot (tm) Perfomance Engine: An In-Depth Look. Article on Suns Java Developer Connection site (1999)
14. Monteiro, F., Rocha, P., Menezes, P., Silva, A., Dias, J.: Teleoperating a mobile robot. A solution based on JAVA language. In: Proceedings of the IEEE Int. Symposium on Industrial Electronics, ISIE 1997, vol. 1. IEEE (2002)
15. Prechelt, L., et al.: Comparing Java vs. C/C++ efficiency differences to interpersonal differences. Communications of the ACM 42(10), 109–112 (1999)
16. Raimondi, F., Ciancimino, L., Melluso, M.: Real-time remote control of a robot manipulator using java and client-server architecture. In: Proceedings of the 7th Int. Conference on Automatic Control, Modeling and Simulation (2005)
17. Robertz, S., Henriksson, R., Nilsson, K., Blomdell, A., Tarasov, I.: Using real-time Java for industrial robot control. In: Proceedings of the 5th Int. Workshop on Java Technologies for Real-Time and Embedded Systems, pp. 104–110. ACM (2007)
18. Roulo, M.: Accelerate your Java apps. Java World (1998)
19. Spw, S., Wentworth, S., Langan, D.: Performance evaluation: Java vs. c++. In: 39th Annual ACM Southeast Regional Conference, March 16-17. Citeseer (2001)
20. Wilson, S., Kesselman, J.: JavaTM Platform Performance - ch. 8. Sun Microsystems (2001), http://java.sun.com/docs/books/performance/1st_edition/html/JPAlgorithms.fm.html

A Comparison of Sampling Strategies for Parameter Estimation of a Robot Simulator

Gordon Klaus, Kyrre Glette, and Jim Tørresen

University of Oslo, Norway
Department of Informatics
{gordonk,kyrrehg,jimtoer}@ifi.uio.no

Abstract. Methods for dealing with the problem of the "reality gap" in evolutionary robotics are described. The focus is on simulator tuning, in which simulator parameters are adjusted in order to more accurately model reality. We investigate sample selection, which is the method of choosing the robot controllers, evaluated in reality, that guide simulator tuning. Six strategies for sample selection are compared on a robot locomotion task. It is found that strategies that select samples that show high fitness in simulation greatly outperform those that do not. One such strategy, which selects the sample that is the expected fittest as well as the most informative (in the sense of producing the most disagreement between potential simulators), results in the creation of a nearly optimal simulator in the first iteration of the simulator tuning algorithm.

Keywords: the reality gap, evolutionary robotics, simulation.

1 Introduction

Evolutionary robotics (ER) [1,2], the application of evolutionary algorithms (EAs) to robot design, has shown itself to be a powerful technique. The ability of EAs to find novel solutions in a large or unfamiliar search space has been demonstrated, e.g., with Sims' swimming robots [3] and in antenna design [4]. Using ER, the designer is free to explore otherwise daunting domains like the complex dynamics of tensegrity structures and soft materials [5,6,7] or the space of robot morphologies [7,8,9]. A simple demonstration of the advantage of ER over the hand-design of robots is given in [10].

While it is possible to do evolutionary robotics in the real world [11], this can be a very time consuming task. The evaluation of a single candidate solution can take on the order of tens of seconds, not including setup time, and an entire run of an EA typically involves thousands of such evaluations. The benefit of using a simulator, which, given sufficient computing power, can perform many evaluations in the time it would take to perform one in reality, is palpable.

This speedup, however, comes with a cost. Because a simulator is only an approximation of reality, it necessarily changes the problem being solved. An individual that behaves a certain way in simulation may not behave the same when transferred to reality. As a result, the fitness landscape will be different.

Most importantly, optima of the approximated fitness function may not be optimal in reality. To deal with this problem of the "reality gap" (as it has been called) several approaches have been made.

It was shown that, by simulating only those aspects of reality that are relevant to the problem at hand (i.e., building "minimal" simulations) and by using empirically determined amounts of noise to model noisy or poorly understood aspects of a system, the transferability to reality of robot controllers evolved in simulation can be improved [12,13,14]. For example, sensors and actuators do not behave in an idealized fashion; there is always some noise, and it can have a large effect on the functioning of a robot. Moreover, the nondeterminism of a noisy simulator helps to produce robust controllers that are less likely to take advantage of a simulator's idiosyncrasies; i.e., it discourages "cheating".

Rather than concerning oneself greatly with the quality of the simulator, it is alternatively possible to sidestep the problem by simply accepting that a simulator has inaccuracies. The transferability approach [15,16] uses multi-objective optimization to explicitly consider the trade-off between fitness and transferability. Other methods involve designing adaptive controllers that can cope with differences between simulation and reality. In [17], a mobile robot was able to perform a non-trivial task in reality after being very rapidly evolved in a simulator, using Hebbian rules to develop a neural controller on the fly. In [18], a control architecture was developed which enabled an evolved robot to adaptively anticipate errors between its expected and actual motions, allowing it to recover from perturbations not encountered in simulation.

Each of these methods is promising in its own way, but they are not the focus of this paper. Here, we deal with methods of improving the quality of simulation. A simulator is typically configured by some set of parameters. For example, a physics simulator has parameters related to friction and restitution that govern the interactions of different materials, and parameters for the dimensions and mass distributions of objects being simulated. Many values can be obtained by direct measurement. However, given that a simulator is a simplification of the real world, there may not be direct correspondences between real and simulated parameters – for example, when nonlinear mechanisms like robot actuators are modeled linearly or when detailed objects are represented by coarse shapes, as is often done.

It is thus necessary to adjust simulation parameters such that the simulator more accurately reflects the real world. Some improvement can be made by simply hand-tuning [19] but clearly more sophisticated methods are required. Even when suitable values can be determined by measurement, a method of automatic discovery or inference could be useful, for example in an autonomous adaptive robot that maintains internally a model of its environment.

The approaches taken in Back-to-Reality (BTR) [20,21], estimation-exploration (EE) [22,23], and using sequential surrogate optimization (SSO) [24] are largely similar to each other: Two coupled optimization algorithms are run in an interleaved fashion, one to search for solutions to a primary task such as simulated robot locomotion and another to improve the accuracy of the simulator. The product of each run of the primary search (a controller for a simulated robot) is evaluated in reality and then used in subsequent simulator optimizations; and the product of each simulator optimization is used in the following primary search. (A more detailed description is given later.) Ultimately, this should result in the convergence toward accurate simulators and, as a result, robot controllers that are fit in reality.

BTR actually interleaves *three* algorithms. In addition to the two just mentioned, it includes a learning-in-reality step after each simulator search, seeded by the prior primary search, and the results of which are used to seed the next primary search. The value of this extra step is a bit unclear. It is expensive to do such fine tuning in the real world, and any information gained is then lost upon return to simulation. Even if the real-world evaluations were used to inform the simulator search, it would be a large price to pay for some tightly clustered, and thus information poor, data points.

An interesting aspect of EE is its measure of simulator fitness. While the other two methods search for a simulator that minimizes the difference between the fitnesses of some individuals in simulation and reality, EE tries to minimize the difference between the robots' sensor readings in simulation and reality. It is a more complicated calculation to perform but it also provides a great deal more information.

Perhaps the most important difference between these methods is in their specific strategies for selecting individuals for evaluation in reality. Individuals evaluated in reality provide the data points, or *samples*, used for tuning the simulator. Because the goal is to do as few real-world evaluations as necessary, it is important that sample selection be done wisely.

BTR and EE operate in the same manner, selecting for evaluation in reality the individual that is fittest in the current simulator. EE is actually described to work differently, but is only used that way in one of its four applications, and not in evolutionary robotics; This other manner is to select the individual that maximizes the disagreement between a number of candidate simulators, i.e., the most informative individual. SSO uses a hybrid strategy, selecting the fittest individual, or, if that individual is within a threshold distance of a previously selected individual, the most informative individual (based on a fitness error estimate that is maintained as part of the surrogate fitness function).

These different selection strategies seem promising but it isn't clear which would give the best results. The goal of this paper, then, is to compare them (and others) on a common task. In the next section, we describe this task, the general algorithm, and each of the specific strategies to be compared. Then, we present and discuss the results. Finally, we conclude and consider avenues for future work.

2 Implementation

The benchmark task in this investigation is a common one: to design a controller for robot locomotion. Figure 1 depicts the robot, a quadruped with a total 12 degrees of freedom in its limbs. To produce motion, each of its actuators is periodically extended and contracted based on a simple pulse shaped function determined by two parameters; a total of 24 parameters thus define such a controller. The task is then to use an EA to search for controller parameters that cause the robot to walk at the highest speed. Details of the robot, its controller, and the EA are given in [10].

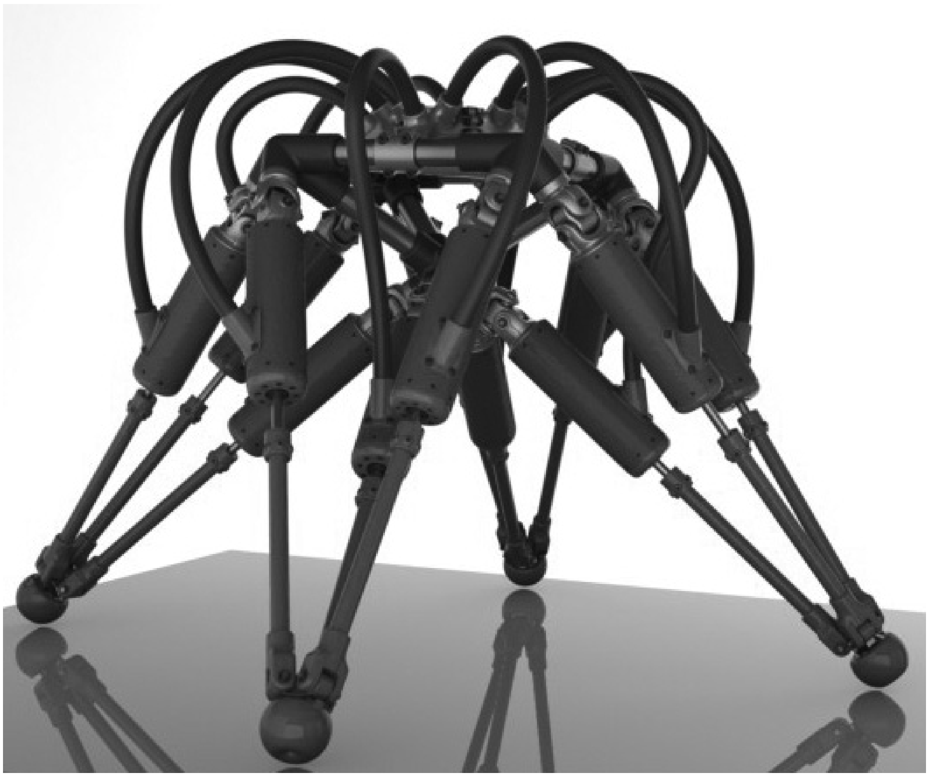


Fig. 1. A rendering of the robot used in this investigation

The ultimate goal is to be able to reliably perform this task for a robot situated in the real world, taking advantage of a simulator to perform most of the expensive robot evaluations required by the EA. In order to determine the expected relative performance of a number of different methods for achieving this goal, however, experiments need to be repeated many times due to the stochastic nature of EAs. This would require thousands of real-world robot evaluations. To expedite this investigation, we have substituted a simulation for the real world. That is, instead of doing evaluations in the real world, we do them in a simulator whose configuration is *unknown* (i.e., hidden from the algorithm). As long as the configuration of "reality" (simulated or otherwise) remains hidden from the process that generates approximations of it, we can consider this to be a "reality gap" problem. It should therefore be sufficient for the purpose of comparing selection strategies.

The PhysX software library was used to simulate robot movement, as in [10]. The simulated reality (henceforth just "reality") was configured such that the material constituting the robot and the ground plane had coefficients of static and dynamic friction 1.0 and 0.8, respectively, and restitution (bounciness) 0.0. The simulator (i.e., the approximating simulator, not "reality") was parameterized by these material properties, so that the process of simulator tuning was a search for values of these three coefficients.

The basic algorithm, illustrated in Figure 2, is essentially the same as in BTR, EE, and SSO. We iterate the following procedure: First, select a new sample (defined in the next paragraph) using one of the selection strategies described later; then, use all the samples collected thus far to optimize a new simulator. Finally, after performing some number of these iterations, the latest simulator is used to evolve a robot controller which, transferred to reality, is the final product of the algorithm. The algorithm is iterative because selection strategies typically use the tuned simulator for the subsequent sample selection; initial sample selection uses a random simulator.

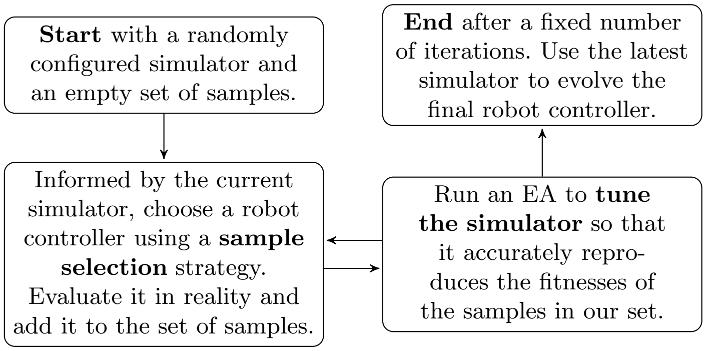

Start with a randomly configured simulator and an empty set of samples.
Informed by the current simulator, choose a robot controller using a **sample selection** strategy. Evaluate it in reality and add it to the set of samples.
Run an EA to **tune the simulator** so that it accurately reproduces the fitnesses of the samples in our set.
End after a fixed number of iterations. Use the latest simulator to evolve the final robot controller.

Fig. 2. The flow of the basic simulator tuning algorithm

A sample, as mentioned earlier, is a robot controller whose fitness (here, walking speed) we evaluate in reality. The real-world fitness values of the samples drive simulator tuning: We use an EA to find the simulator (specifically, the three parameters named earlier) that accurately reproduces the fitness values of the samples collected thus far. The fitness of a simulator *sim*, to be maximized, is

$$f(sim) = \frac{1}{\sum_{x \in samples} (f_{sim}(x) - f_{real}(x))^2}$$

where $f_{sim}(x)$ and $f_{real}(x)$ are the fitnesses in the simulator and in reality, respectively, of an individual x. We use the same EA parameters as in the robot optimization, but only 2000 evaluations.

Figure 3 illustrates the progression of the algorithm when it is run on a much simpler function optimization task. In this example, we can visualize the fitness landscape, sample selection, the successively more accurate simulators, and transferral from simulation to reality.

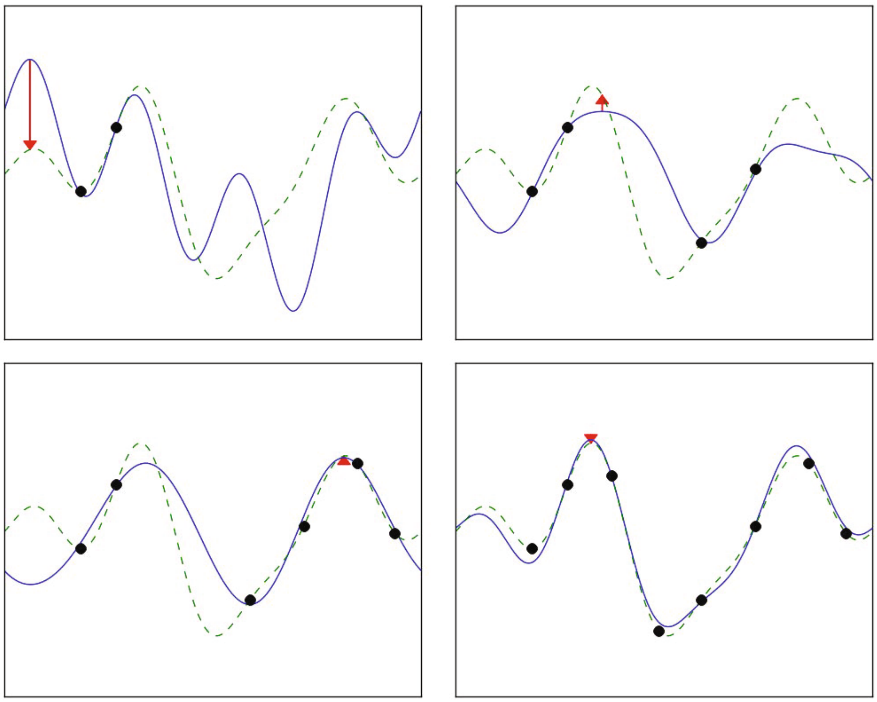


Fig. 3. A visualization of the algorithm being run on a function optimization task after 2, 4, 6, and 8 iterations. The dashed curve is the true function (reality) and the solid curves are approximating functions (simulations). Dots indicate samples. The arrow indicates the fittest in simulation and its transferral to reality. Notice that, as more samples are added, the simulations become more accurate and the fitness of the transferred individual increases until it is near the global maximum of the real function.

In the following paragraphs we describe the six sampling strategies that are the objects of comparison in this paper. The first four strategies fit in the iterative form of the algorithm described above and use the ideas from BTR, EE, and SSO, while the last two are non-iterative and much simpler.

Simulated Fit. This is the strategy used in BTR and EE and partially in SSO. The individual is selected for evaluation in reality that has the highest fitness in the latest tuned simulator (or, in the first iteration, a simulator with randomly chosen parameters). An EA is run to find this individual; this EA uses the same parameters as the one used to produce the final result of the algorithm. This strategy is motivated by the idea that our simulator needs to accurately model only individuals of high fitness; it thus tries to bias the samples towards what are estimated to be fit individuals.

Simulated Fit and Unique. This is the same as the previous strategy, but with a modification to the fitness function to maintain some distance between the samples. The modified fitness function is

$$f^*_{sim}(x) = f_{sim}(x) - \frac{1}{100N} \sum_{y \in samples} \frac{1}{d(x, y)}$$

where $f_{sim}(x)$ is the simulated fitness of x, $N = |samples|$, and $d(x, y)$ is the Euclidean distance between individuals x and y in terms of both genome and fitness. This strategy is motivated by the fact that we might expect the previous strategy to pick, after several iterations, very similar individuals near a local optimum. Several such tightly clustered samples would provide little more information than a single one.

Informative. This strategy was suggested in EE and used partially in SSO. The individual is selected that maximizes the disagreement between a number of simulators. A modification to the basic algorithm is required: Instead of evolving a single simulator, a diverse population of 20 simulators is evolved. To maintain diversity, the EA's method of replacement is changed: After a newly evaluated simulator is added to the population, instead of dropping the least fit, the pair of simulators whose genomes are most similar is found and the less fit of the two is removed from the population.

To find the individual that maximizes the disagreement between the simulators, an EA is used. Its fitness function is calculated as the weighted standard deviation of the fitness values the simulators produce for a given individual. The weights are the fitnesses of the simulators themselves, so that an inaccurate simulator contributes less to this measure than an accurate simulator.

Simulated Fit and Informative. This is a mixture of two strategies. It works the same as the *Informative* strategy, but in addition to the weighted standard deviation, its fitness function includes the weighted average of the fitness values produced by the diverse population of simulators.

Random. Individuals are picked with genomes drawn from a uniform distribution. As each selection is entirely independent of the previous ones, the algorithm collapses to a non-iterative form where all the samples are selected at once before a single simulator is finally tuned.

Known Fit. This strategy is like *Random* except the samples are generated as mutated versions of an individual known to have middling fitness. In this case, the hand-designed controller from [10] was used; it achieved a fitness of 0.73 in "reality". Values drawn randomly from a uniform distribution on $[0, 0.25)$ were added to each element of this controller's genome to produce each new sample. This strategy is motivated by the observation that many of the purely randomly generated samples had very low fitness. In SSO, a similar strategy was used to generate a small set of initial samples.

3 Experiments and Results

The aim is to achieve the highest real-world fitness while doing the fewest real-world evaluations. It is on this basis that we compare the performance of the different selection strategies. The creation of a generally accurate simulator may be an intermediate goal; however, a relatively poor simulator is perfectly acceptable if it enables us to find robot controllers that are very fit in reality – which may very well be the case for a simulator that captures only certain essential aspects of reality. For this reason, we make no explicit judgments of simulator accuracy.

For each of the strategies, the algorithm is iterated ten times; that is, ten real-world evaluations are performed. After each iteration of the algorithm, the best simulator is used to evolve a robot controller for walking speed. This controller is then evaluated in reality and its fitness (or the maximum fitness of the samples, if it is larger) is recorded. We do not count this real-world evaluation among those that are the basis for comparison, as it is not used as a sample for simulator tuning – it is only a view into the algorithm's performance. Figure 4 shows the recorded fitness values averaged over 40 repetitions of this process.

In addition to comparing the different strategies to each other, we should also consider them with regard to the expected performance of evolving directly in reality. Because we have substituted the real world with a simulator, it is a simple task to compute the fitness expected from evolving directly in reality. Just as in [10], robot controllers were evolved for 10000 evaluations in the simulator configured as "reality"; with 100 repetitions, the best fitness at the end of the evolutionary runs had an average of 1.33 (standard deviation 0.10).

All of the strategies showed real-world fitness increasing with the number of real-world evaluations. Before any iterations, all strategies of course produced roughly the same expected fitness, about 0.9, from evolving in a randomly configured simulator. With more iterations, fitnesses improved (diminishingly) to more or less 1.3.

Known fit performed only marginally better than *Random*. More improvement could probably have been achieved by using a fitter individual as the seed for generating the samples.

The three strategies that involve simulated fitness were significantly better than the others in terms of both average and standard deviation of real-world fitness. After only three real-world evaluations, these three strategies achieved real-world fitness of at least 1.3, and after ten iterations they reached 1.4. Standard deviations were in the range 0.10-0.15, about three times smaller than in the other strategies and comparable to that found when evolving directly in reality. This indicates that the inclusion of simulated fitness as a factor in sample selection is very effective, perhaps essential.

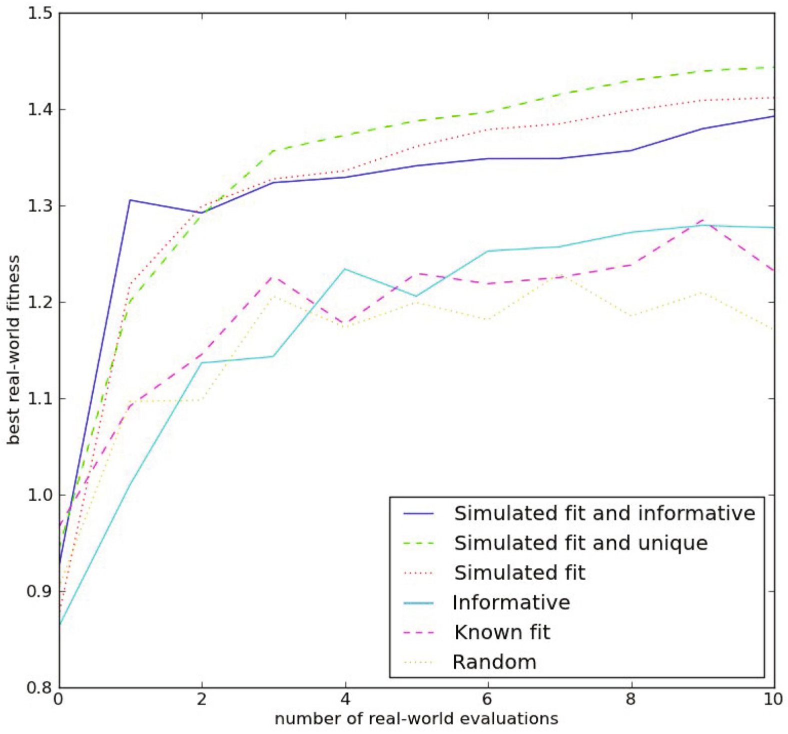


Fig. 4. Fitness in reality versus number of real-world evaluations for the six sample selection strategies. The statistical significance of the results (using Student's t-test) is as follows: With just one real-world evaluation, the *Simulated fit and informative* strategy was significantly different from the other strategies ($p < 0.01$). Combining the data for more than two real-world evaluations, each of the strategies involving simulated fitness was significantly different from the others ($p << 0.001$) and *Known fit* was significantly different from *Random* ($p \approx 0.017$). Considering more than five real-world evaluations, each of the three weakest strategies was significantly different from the others ($p < 0.05$).

The additional criterion of uniqueness appears to give a small improvement over plain *Simulated fit* after a couple of iterations, as expected. We might expect the same sort of improvement from *Informative*, as informative implies uniqueness (a repeated sample is totally uninformative), but we do not see it. This is because the strategies involving simulated fitness produced nearly perfectly tuned simulators after about 3 or 4 iterations, as they have attained the same fitness as evolution in reality using the same number of simulated evaluations. Beyond these 3 or 4 iterations, the algorithm is hardly tuning the simulator anymore, instead just searching for fitter samples; the increasing fitness is similar to what would be seen by simply extending the EAs beyond 10000 evaluations.

That the different strategies show different rates of improvement in later iterations is due to their specific construction: The strategy that promotes unique samples introduces a weak exploratory force in the algorithm, leading it to search locally and thus to improve more quickly, while the strategy biased towards informative samples introduces a stronger exploratory force (informative samples will typically be quite distant from each other) that may lead the algorithm away from regions of high fitness.

Most striking is the performance of the *Simulated fit and informative* strategy. While *Informative* alone was only slightly better than *Random* and *Known fit*, in combination with *Simulated fit* it produced a nearly perfect result in just a single iteration. The fact that the other two strategies outperform it in later iterations is due to the phenomenon described it the previous paragraph, and for this reason shouldn't be considered a weakness of this strategy. This is the most promising of the six selection strategies.

It is interesting to consider how poorly the *Informative* strategy performed, especially when its combination with *Simulated fit* was so successful. It seems likely that the diversity maintenance mechanism used in simulator evolution was not as effective as desired – it plays no role in the first iteration (in which randomly configured simulators are used), where *Simulated fit and informative* really shines.

4 Conclusion and Future Work

Our investigation has demonstrated the important role that sample selection strategies play in simulator tuning. Of the six strategies used here, the *Simulated fit and informative* strategy proved to be the most successful at closing the "reality gap" for this particular task of robot locomotion. In fact, seeing as the algorithm was able to nearly perfectly tune the simulator in just a single iteration using this strategy, it seems reasonable to say that this strategy has *solved* this particular instance of the task.

It remains to be seen how well this algorithm scales up to more complex scenarios; this will be the main focus of our future work. This instance of simulator tuning was rather simple, involving the estimation of only three parameters. Other parameters to be considered include motor forces, noise sources, and dimensions and mass distribution of the robot body. Ultimately, testing must be done on a physical robot outside simulation.

To succeed at more complex tasks, it may be necessary to improve upon the *Simulated fit and informative* strategy. The current implementation involves a simple sum of two factors (one from simulated fitness, the other informative) whereas this probably ought to be a more general weighted sum so that it is possible to adjust the influence of the two factors. It may also be beneficial to more tightly interleave the evolution of diverse simulators with the evolution of informative samples, as their respective diversity and informativeness are strongly coupled.

Finally, as sample selection strategies become more sophisticated, it could be practical to consider a more nuanced basis for comparison than just the number of real-world evaluations. If some strategies take much longer to run than others then the total running time would be a better measure. If a strategy takes a very long time to run then one might consider instead to do more real-world evaluations in that time; to capture this possibility, the time it takes to perform real-world evaluations must be considered. In this situation it would probably make sense to give greater weight to time spent doing real-world evaluations than time spent computing, as it demands more constant human attention.

References

1. Doncieux, S., Bredeche, N., Mouret, J.-B.: Exploring new horizons in evolutionary design of robots. In: Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE Press (2009)
2. Harvey, I., Husbands, P., Cliff, D., Thompson, A., Jakobi, N.: Evolutionary robotics: the sussex approach. Robotics and Autonomous Systems 20, 205–224 (1997)
3. Sims, K.: Evolving virtual creatures. In: SIGGRAPH 1994: Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques, pp. 15–22. ACM, New York (1994)
4. Linden, D., Hornby, G., Lohn, J., Globus, A., Krishunkumor, K.: Automated antenna design with evolutionary algorithms. American Institute of Aeronautics and Astronautics 5, 1–8 (2006)
5. Rieffel, J., Trimmer, B., Lipson, H.: Mechanism as mind: What tensegrities and caterpillars can teach us about soft robotics. In: Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems (2008)
6. Glette, K., Hovin, M.: Evolution of Artificial Muscle-Based Robotic Locomotion in PhysX. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2010)
7. Rieffel, J., Saunders, F., Nadimpalli, S., Zhou, H., Hassoun, S., Rife, J., Trimmer, B.: Evolving soft robotic locomotion in PhysX. In: GECCO 2009: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference, pp. 2499–2504. ACM, New York (2009)
8. Bongard, J.C.: Incremental Approaches to the Combined Evolution of a Robot's Body and Brain. PhD thesis, University of Zurich (2003)
9. Macinnes, I., Di Paolo, E.: Crawling out of the simulation: Evolving real robot morphologies using cheap reusable modules. In: Pollack, J., Bedau, M., Husbands, P., Ikegami, T., Watson, R. (eds.) Artificial Life IX: Proceedings of the Ninth Interational Conference on the Simulation and Synthesis of Life, pp. 94–99. MIT Press, Cambridge (2004)
10. Klaus, G., Glette, K., Høvin, M.: Evolving Locomotion for a Simulated 12-DOF Quadruped Robot. In: Lones, M.A., Smith, S.L., Teichmann, S., Naef, F., Walker, J.A., Trefzer, M.A. (eds.) IPCAT 2012. LNCS, vol. 7223, pp. 90–98. Springer, Heidelberg (2012)
11. Zykov, V., Bongard, J.C., Lipson, H.: Evolving dynamic gaits on a physical robot. In: Proceedings of Genetic and Evolutionary Computation Conference, Late Breaking Paper, GECCO (2004)
12. Jakobi, N.: Minimal Simulations for Evolutionary Robotics. PhD thesis, University of Sussex (1998)

13. Jakobi, N., Husbands, P., Harvey, I.: Noise and the Reality Gap: The Use of Simulation in Evolutionary Robotics. In: Morán, F., Merelo, J.J., Moreno, A., Chacon, P. (eds.) ECAL 1995. LNCS, vol. 929, pp. 704–720. Springer, Heidelberg (1995)
14. Miglino, O., Lund, H.H., Nolfi, S.: Evolving mobile robots in simulated and real environments. Artificial Life 2, 417–434 (1996)
15. Koos, S., Mouret, J.-B., Doncieux, S.: The transferability approach: Crossing the reality gap in evolutionary robotics. IEEE Transactions on Evolutionary Computation (2012)
16. Koos: The transferability approach- an answer to the problems of reality gap, generalization, and adaptation. PhD thesis, Institut des Systémes Intelligents et de Robotique Université Pierre et Marie CURIE (2011)
17. Floreano, D., Urzelai, J.: Evolution of Plastic Control Networks. Autonomous Robots 11(3), 311–317 (2001)
18. Hartland, C., Bredeche, N.: Evolutionary robotics, anticipation and the reality gap. In: IEEE International Conference on Robotics and Biomimetics, ROBIO 2006, pp. 1640–1645 (December 2006)
19. Glette, K., Klaus, G., Zagal, J.C., Tørresen, J.: Evolution of locomotion in a simulated quadruped robot and transferral to reality. In: Artificial Life and Robotics (2012)
20. Zagal, J.C., Ruiz-del-Solar, J., Vallejos, P.: Back to reality: Crossing the reality gap in evolutionary robotics. In: Proceedings of IAV 2004, the 5th IFAC Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal (2004)
21. Zagal, J.C., Ruiz-Del-Solar, J.: Combining simulation and reality in evolutionary robotics. J. Intell. Robotics Syst. 50, 19–39 (2007)
22. Bongard, J.C., Lipson, H.: Once more unto the breach: co-evolving a robot and its simulator. In: Proceedings of the Ninth International Conference on the Simulation and Synthesis of Living Systems (ALIFE9), pp. 57–62 (2004)
23. Bongard, J., Lipson, H.: Nonlinear system identification using coevolution of models and tests. IEEE Transactions on Evolutionary Computation 9, 361–384 (2005)
24. Hemker, T., Sakamoto, H., Stelzer, M., Stryk, O.V.: Hardware-in-the-loop optimization of the walking speed of a humanoid robot. In: CLAWAR 2006: 9th International Conference on Climbing and Walking Robots, pp. 614–623 (2006)

A Framework with a Pedestrian Simulator for Deploying Robots into a Real Environment

Masahiro Shiomi[1,2], Francesco Zanlungo[1,2], Kotaro Hayashi[1,2], and Takayuki Kanda[1,2]

[1] ATR Intelligent Robotics and Communications Labs
[2] Japan Science and Technology Agency, CREST
m-shiomi@atr.jp

Abstract. We describe a simulation framework aimed to develop and test robots before deploying them in a real environment crowded with pedestrians. In order to use mobile robots in the real world, it is necessary to test whether they are able to navigate well, i.e. without causing safety risks to humans. This task is particular difficult due to the complex behavior pedestrians have towards each other and also towards the robot, that can be perceived either as an obstacle to avoid or as an object of interest to approach for curiosity. To overcome this difficulty, our framework involves a pedestrian simulator, based on a collision avoidance model developed to describe low density conditions as those occurring in shopping malls, to test the robot's navigation capability among pedestrians. Furthermore, we analyzed the behavior of pedestrians towards a robot in a shopping mall to build a human-to-robot interaction model that was introduced in the simulator. Our simulator works as a tool to test the level of safety of robot navigation before deploying it in a real environment. We demonstrate our approach showing how we used the simulator, and how the robot finally navigated in a real environment.

Keywords: Pedestrian simulation, Safe navigation, Mobile robot, Field trial.

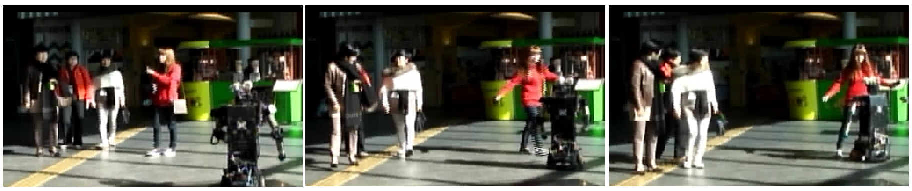
1 Introduction

Deploying a robot in a real environment with ordinary people is one of the major targets and challenges in robotics. Due to the improvement of robots' sensing abilities such as human tracking, robots can now assist people working in a real world environment, and lately research works have been performed to deploy robots in environments such as malls [1], science museums [2], or hospitals [3].

One of the most serious problems to face when using robots in a real environment is to ensure that they can be operated safely between humans. When operating our robots in the real world we have experienced difficulties in securing the robot's safety due to unexpected behaviors of pedestrians who may, for example, be strongly interested in the robot and surround it or even explicitly obstruct its motion by continuously standing on its front (Fig. 1-(a)). Moreover, the density in a real environment may change strongly with time (for example, the density in a shopping mall at lunch time is higher than in usual situations (Fig. 1-(b)), making difficult to develop and test in the laboratory a navigation system that may be safe in any setting.

Simulations have been already used to test robots before using them in real environments [4, 5], but these works dealt with a “static” environment, i.e. they needed only to simulate the robot behavior and not to cope with complex human behaviors and reactions. When the robot has to interact with humans, and even operate in a human crowd, the necessity to model the complex and diversified behavior of humans makes more difficult to close the gap between simulations and the real world [6].

In this paper, we report the effectiveness of our approach (Fig. 2) that uses a realistic pedestrian simulator in order to test the safety of robot navigation. To this end, we first gathered position data of pedestrians in a shopping mall by using a human tracking system [7], to analyze their behavior in that environment. Then, we used a pedestrian model explicitly developed to describe the relatively low-density situations occurring in shopping malls [8], to simulate the walking behavior of pedestrians and control robot locomotion. Finally, after testing the robot safety in the simulated environment, we conducted a real world field trial.


(a) An adult approaches and explicitly obstructs the robot

(b) Relatively high density situation in a mall

Fig. 1. Example of difficult situations for the deployment of a robot in real environments

2 Related Works

It is quite common to use simulators in robotics, in particular to reproduce physical phenomena. For example, León et al. developed a simulator to reproduce grasping behavior [4], while Tsai et al. used a simulator to investigate the safety of motion planning in a dual arm robot [9], and Boedecker et al. used a simulator to test the walking behavior of a two-legged humanoid robot [10]. A few research works consider also the behavior of robots around people [11,12]. For instance, Sisbot et al. used a simulator to investigate socially appropriate path planning around humans [12]. However, none of these studies addresses the influence of diverse, dynamical and reactive human behavior, focusing on static posture or fixed motion patterns.

On the other hand, pedestrian behavior has been relatively well studied in human science. Helbing proposed a “social force” model to simulate people's motion in escape situations [13], which has been used by Pelechano et al. to simulate high-density crowds [14]. These studies addressed pedestrian behavior in high-density situations, which is different from the one exhibited at lower densities of interest for normal social interactions. In our study, we use a pedestrian model specifically prepared for low-density settings as those occurring in shopping malls [8]. Also some previous works used pedestrian models to simulate robot navigation in a crowd [15, 16], but their approach was limited to simulation and did not address real world situations. Thus, we consider that the novelty of our study resides in addressing a method to use a pedestrian simulator in the deployment process of a robot toward real world use.

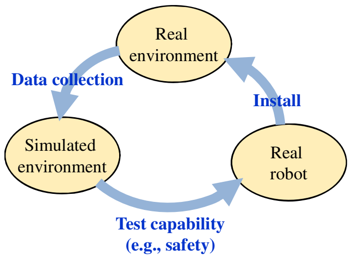


Fig. 2. The interaction of simulated and real environment enables testing robot capabilities

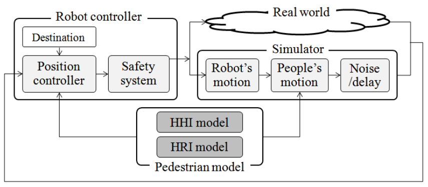


Fig. 3. Overview of our framework

3 System

Our framework (Fig. 3) consists of three main components: a pedestrian model, a robot controller, and a simulator. In the pedestrian model, the social forces among people were computed using the model as described in [8] (HHI model), while the force toward the robot was computed using specific parameters (HRI model), [17]. The robot controller navigates the robot by using the HRI model.

3.1 Robot and Sensors

We used a 120-cm-high 60-cm-wide humanoid robot, Robovie-II. Its mobile base is a Pioneer 3-DX (Active Media). We used it at a maximum velocity of 750 mm/sec and preferred velocity of 700 mm/sec. Its maximum acceleration is 600 mm/sec^2. It has bumpers and laser range finders (Hokuyo, UTM-30LX) for ensuring safety.

The pedestrian model needs information about people's positions far from the robot, which is not easy to collect using only on-board sensors. We thus used 8 environmental laser range sensors and a tracking system with shape matching at torso-level and a particle filter method [7], whose position error was in average 0.06 m.

3.2 HHI Model

Models of pedestrian collision-avoidance have been developed since the 50s to deepen understanding of crowd dynamics and design better facilities. The Social Force model (SFM) [13] is a popular pedestrian model that describes the behavior of pedestrians in a crowd through reaction forces inspired by physics.

However, the original social force model is designed to describe well high-density settings such as panic and escape situations [13], and it is not suitable to reproduce low-density many-people environments such as shopping malls. To solve this problem, we use a SFM specification which can reproduce such low-density settings [8]. This work proposes a new SFM specification called "collision prediction" (CP-SFM) in which relative velocity is used to compute the relative distance among pedestrians at the moment of maximum approach in future, a computation performed by assuming current velocities to remain constant. The acceleration of pedestrian i is given by

$$\boldsymbol{f}_{i,j} = A\frac{v_i}{t_i}e^{-d_{i,j}/B}\frac{\boldsymbol{d}'_{i,j}(t_i)}{d'_{i,j}(t_i)} \tag{1}$$

where $\boldsymbol{v}_i$ is the velocity of pedestrian i, t_i is the time of maximum approach and $\boldsymbol{d'}_{ij}$ is the (predicted) relative distance to pedestrian j at t_i. The parameters of the model, A= 1.13, B=0.71, were calibrated on real pedestrian trajectories (see [8] for details).

3.3 HRI Model

We extended the pedestrian model to also include, besides collision avoidance, the behavior around the robot. While some people only interact with the robot to avoid collisions, others slow down or stop to observe it, while some of them approach it to initiate a conversation. Modeling these people is important; if we rely on a pure collision avoidance model the robot may collide with people who behave differently.

3.3.1 Data Collection and Coding to Establish HRI-Behavior Categories

We used a field trial in which a robot provided information about shops to people that approached it and stopped to talk [18]. In the field trial, in which the robot roamed a 144 m^2 wide area in a shopping mall corridor, we recorded (using tracking system [7]) 266 pedestrians' trajectories during an hour of data collection. We have analyzed these trajectories based on how the pedestrians change their walking course in relation with the robot, and found four major patterns:

Approach to Stop: People approached the robot and stopped to talk at social distance zone (approx. 1.2m)

Stop to Observe: People stopped to observe the robot at a distance larger than the social zone one.

Slow Down: People did not change their walking course toward the robot but slowed down to observe it and passed by without stopping.

Collision Avoidance Only: People avoided the robot but did not change their walking course toward it nor slowed down their walking speed.

To confirm these patterns, we coded the data using a standard human science procedure. Two independent persons (coders) classified the trajectories using these four categories, and the coding process resulted in Cohen's kappa coefficient 0.709, showing a reasonable concordance between the coders. No trajectory was classified out of these four categories, while the number of trajectories in each category was: 70 approach to stop, 69 stop to observe, 11 slow down, and 116 collision avoidance only.

3.3.2 Development of the Models

We developed equations to model people's walking behavior around the robot according to the four patterns (HRI type models):

Approach to Stop: In the "approach to stop" category (Fig. 4), 90.1% of the people approached the robot from the front and their motion was straight toward the robot. We assumed that people in this category approach the robot only when it falls within their sight, and the following equation represents this idea. Note that it is used in combination with (1), so that the motion is also affected by the social force from other pedestrians.

$$\boldsymbol{v}_i^0 = \begin{cases} 0 & (d_{i,r} \le D_{stop}) \\ \boldsymbol{v}_i^{robot} & (D_{stop} < d_{i,r} \le D_{notice}\, and\, |\theta_{i,r}| < 90) \\ \boldsymbol{v}_i^{goal} & (otherwise) \end{cases} \tag{2}$$

Here v_i^0 is the preferred velocity of pedestrian i, v_i^{goal} is the preferred velocity directed to the goal, v_i^{robot} is the vector directed to the robot with the same scalar size as v_i^{goal} (i.e. the pedestrian aims to move toward the robot with her own preferred velocity), $d_{i,r}$ is the distance between the pedestrian and the robot, and $\theta_{i,r}$ is the angle between her frontal direction and the direction to the robot . We computed D_{stop} from observed trajectories (in average people in this category stopped at a distance of 0.893m, S.D. 0.229 m from the robot), and set D_{notice} to 10 m (the SFM visibility range).

Stop to Observe: As people in the "approach to stop" category (Fig. 5), people in this category come close to the robot as soon as it falls within their sight, but they stop at a larger distance. Thus, we approximate their behavior as:

$$\boldsymbol{v}_i^0 = \begin{cases} 0 & (d_{i,r} \le D_{observe}) \\ \boldsymbol{v}_i^{robot} & (D_{observe} < d_{i,r} \le D_{notice}\, and\, |\theta_{i,r}| < 90) \\ \boldsymbol{v}_i^{goal} & (otherwise) \end{cases} \tag{3}$$

where $D_{observe}$ is 2.38 m (average stopping distance from the robot, S.D. 1.19 m).

Slow Down: People's motion direction did not change toward the robot, but their speed decreased as they were close to it. We analyzed the change of the speed and found that their speed around the robot was 62% of the average in other areas. Thus, we modeled slow down behavior using the following equation:

$$\boldsymbol{v}_i^0 = \begin{cases} \alpha \boldsymbol{v}_i^{goal} & (d_{i,r} \le D_{slowdown}) \\ \boldsymbol{v}_i^{goal} & (otherwise) \end{cases} \tag{4}$$

where α=0.62. We set $D_{slowdown}$ =4 m on the basis of the observed pedestrian behavior.

Collision Avoidance Only: This behavior was modeled using eq. (1), as for inter-pedestrian interactions. However, since we expected a difference in the amount of force perceived from the robot (e.g. keep a larger distance), we re-calibrated the social force toward the robot (see Section 3.3.3).

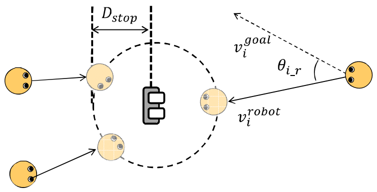


Fig. 4. Illustration of "approach to stop" trajectories

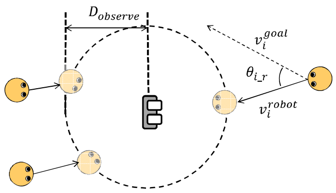


Fig. 5. Illustration of "stop to observe" trajectories

3.3.3 Calibration of the Social Force toward the Robot

We conducted a data collection to investigate how people behave when avoiding a robot. Fourteen Japanese people (six men and eight women whose average age was 25.1 years, S.D. 8.7) participated in this experiment. Each subject repeated the trial nine times. In each trial the robot moved straight toward a participant at 700 mm/sec, and the participants moved toward the robot, starting at a distance of 18 m. Subjects were instructed to walk freely toward a goal located behind the robot, and informed that the robot would not change its course to avoid collision.

For calibration we used a genetic algorithm to select the parameters maximizing similarity between simulated trajectories and real ones while minimizing collisions in simulation [8]. The algorithm provides parameters values for the interaction force (Ar=0.62, Br =1.07) that, from the point of view of collision-avoiding intensity, do not qualitatively differ from the inter-human values of [8] (Ah= 1.13, Bh =0.71). We expected the pedestrians to avoid the robot more strongly than they avoid other humans, but this behavior was not clearly observed in the experiment performed for data collection. We note that the parameters for the robot are eventually re-adjusted taking into account noise, delay, and robot motion capabilities (Section 4.1).

3.4 Robot Controller (Position Controller and Safety System)

We assume that while global path-planning providing the destination is conducted at an upper layer, this robot controller is responsible of local navigation, i.e. of safely avoiding collision with static and dynamic entities around the robot.

We could use our framework to test various navigation strategies to reveal the most appropriate navigation mechanism; in fact, we explored the navigation strategy of the robot using this same simulator, and the results of our analysis will be reported elsewhere [17]. In this paper we report only the most suitable strategy we found.

The idea underlying the navigation mechanism is to use a strategy similar to that used by pedestrians, to obtain human-like collision avoidance in the robot. To this end we used the social force model [8] with the human-robot parameter values. In concrete, given the local destination to obtain the preferred velocity, the system computes the robot's desired next position on x-y coordinates using the "collision prediction" (CP-SFM) social force model of eq. (1), and converts it into a polar coordinates velocity command (vp, ωp). However, we needed to consider the discrepancies between the "ideal" simulation world and the real one as, for example, slow acceleration, the inability of our differential drive robot to move aside, the noise in the human tracking system and the computation delays; discrepancies that cause the robot's trajectory to diverge from that determined by the "ideal" model. To compensate this difference, we further calibrated the pedestrian model parameters to fit the real world behavior (see section 4.1). The polar coordinates (vp, ωp) velocity command is finally examined through a safety-check mechanism, a time varying dynamic window method [19] using a 1.5 sec window time by considering maximum speed and acceleration of our robot, which is long enough to stop the robot.

3.5 Simulator

The simulator is used to test the robot navigation, reproducing people's walking behavior around the robot. The simulator has three sub-modules: pedestrian simulator, noise/delay simulator, and the robot's motion simulator.

The pedestrian simulator computes pedestrians' positions every 100ms, on the basis of the pedestrian model [8]. The noise/delay simulator simulates the noise in sensing, modeled as a Gaussian noise, and delay in observation and computation. The parameters of the noise simulator were decided on the basis of data collected in the target environment (section 4.1). The robot's motion simulator simulates the movements of the robot by using the robot controller taking also in account the dynamics of its two-wheeled mobile base.

Fig. 6 shows the trajectories of the robot (red ellipse) and of a pedestrian (black ellipse) that got closer and stopped around the robot before directing to his goal. As the pedestrian approached the robot, the latter deviated to the right and was able to avoid him, even if the pedestrian approached and stopped around the robot.

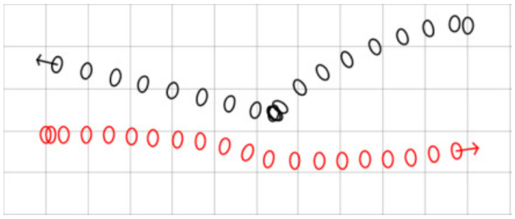


Fig. 6. Trajectories in simulation

4 Simulation

4.1 Overview

The simulation was conducted in a 10 x 20 m virtual corridor. The simulator sets people's initial positions and goals to opposite sides of the corridor, along with their arrival time to the environment and preferred velocity (average and S.D are 1.4m/sec and 1.33, based on the data collection in Section 3.3.1). The ratio of HRI type behaviors is set as the same as the one observed in data collection (section 3.3.1). We also measured the delay of the system in the laboratory, which resulted to be 350msec, and defined the noise of the sensing system as 0.06m, as reported in [7]. The initial position and goal of the robot are set as for the pedestrians.

By using delay and noise information, we further calibrated the values of the pedestrian model parameters to obtain in the real robot system trajectories as similar as possible to the "ideal" ones (i.e. obtained using the HRI model with no noise or delay). As a result, the parameters for the real robot were increased to Ar= 0.93, Br =1.61, showing that the collision-avoiding interaction has to be strengthened to cope with the robot's motion limitations.

4.2 Measurement

We propose two performance measures:

Ratio of Collision: we defined a collision initiated by the robot as a situation in which the distance between a center of person and the center of robot gets smaller than 30cm, and the ratio of collision was computed as the number of collisions per the number of people who entered within a 5m distance from the robot. In this evaluation, we did not count collisions caused by a pedestrian, defined as either a) a pedestrian collided with the robot while it was stopped, or b) a pedestrian collided with the robot from behind. Note that in the real world collisions might not happen even if this distance is attained, as humans may rotate their body to avoid the collision; nevertheless this is a valid measure of the safety of the robot's behavior.

Efficiency: defined as: "time to reach the goal" over "time to reach the goal going straight at preferred speed". Deviations due to collision avoiding reduce efficiency.

4.3 Results

To confirm the safety capability of robot navigation in various situations, we conducted simulations by increasing density from 0.01 to 0.05 people/m^2 with 0.01 intervals. In each density setting, we conducted 1000 simulations.

Fig. 7 shows *efficiency* and *ratio of collision* in each density setting. We had *ratio of collision* 0% until density 0.03, while the robot caused 0.01% and 0.02% collisions at density 0.04 and 0.05, respectively. The *efficiency* at density 0.01 was 79%, and it decreases with increased density (65% at density 0.05).

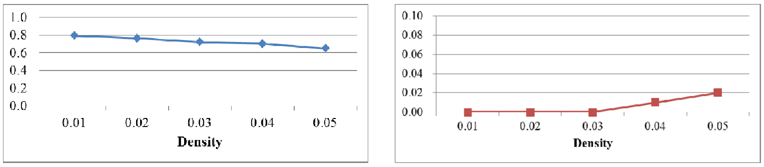


Fig. 7. Efficiency (left) and ratio of collision (right) in simulations

5 Field Trial

5.1 Overview

According to the results of simulations, our robot has enough safe capability in real environments provided that the density is not higher than 0.03. To confirm this prediction, we conducted a field trial in a real environment with characteristics similar to the simulated one. Fig. 8 shows the corridor of a shopping mall in which we performed the field trial, an area of size 10 x 20 m, in which people walk with an average speed of 1.32m/sec (s.d. 1.33) at a density up to 0.03 people/ m^2. The purpose of the field trial is to test (a) whether the robot safely navigates as predicted by simulations, and (b) whether the efficiency trend is reproduced as predicted.

The robot was fully autonomously operated, except for the start signal sent by an operator to trigger it to move. After receiving the signal, the robot moved from points A/B to B/A (we defined a single movement between these points as one trial).

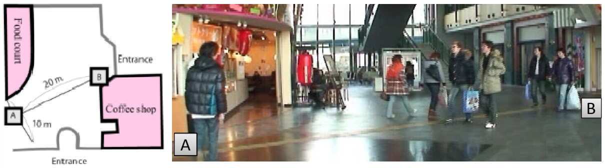


Fig. 8. Map and image of the field trial site

5.2 Measurements

To confirm whether the robot could navigate safely in a real environment, we measured *efficiency* and coded whether the robot's behavior caused any problems in terms of safety, i.e., for each person who walked within 5 m from the robot, we asked *coders* to determine whether the situation was safe, using the following criteria:

Unsafe: due to the presence or motion of the robot, the pedestrian had to make a quick change in his/her moving direction to avoid colliding with the robot.

Otherwise, the person's situation was coded as **safe**.

5.3 Results

In the field trial, we conducted a two-hour test consisting of 27 trials. Two coders classified the interactions between the robot and the 160 pedestrians that walked within a 5 m distance from it as safe or unsafe by observing the recorded videos.

Cohen’s kappa coefficient was 0.89, indicating high consistence. Moreover, for consistent analysis, the coders discussed and reached a consensus on all the observed situations.

Fig. 9 shows *efficiency* and *unsafe situation* in the field trial. As shown in the figure, no *unsafe situation* was found, confirming that our system safely navigates the robot in both simulated and real environments. The *efficiency* was 59%, 58% and 51 % for density 0.01, 0.02 and 0.03, respectively. These values are lower than the simulated ones, possibly due to the more complex behavior of actual pedestrians (the models reproduce only average pedestrian behaviors; introducing stochasticity in pedestrian decisions could thus reduce the gap between simulations and the real world). However, the results showed a trend similar to the simulated one (an increase in density caused a decrease of efficiency). These results suggest that our simulation system reproduces properly the interaction between the robot and a pedestrian crowd.

Fig. 10 shows a scene in which the robot successfully navigated in a many-people setting. The robot initially changed its moving direction to avoid a group of people, just to meet another incoming group (Fig. 10-a). As a result the robot slightly deviated to slip through the groups (Fig. 10-b). After avoiding the two groups, the robot tried to reach its goal, but another pedestrian was coming from the goal direction (Fig. 10-c). Therefore, the robot deviated again to avoid the pedestrian and eventually headed toward its goal (Fig. 10-d).

The robot was equally able to deal with pedestrians that tried to approach it, as predicted by our HRI type behaviors. In Fig. 11 we analyze one of these situations. While heading to its goal, the robot met a group of pedestrians coming from the opposite side (Fig. 11-a). After noticing the robot, a pedestrian deviated suddenly to approach it, and the robot changed its moving direction in order to avoid him (Fig. 11-b). The pedestrian continued to approach the robot despite this avoiding maneuver (Fig. 11-c), but the robot could safely cope with the pedestrian’s motion (Fig. 11-d). These examples illustrate that the robot is able to navigate safely in the real environment as well as in the simulated environment.

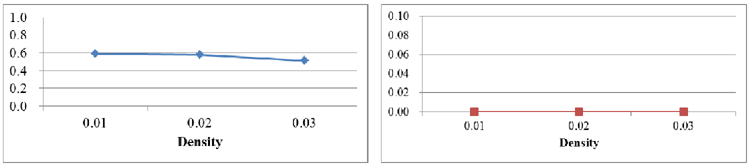
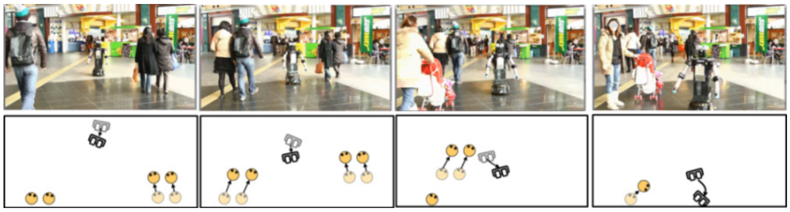


Fig. 9. Efficiency (left) and unsafe situations (right) in the field trial



(a) (b) (c) (d)

Fig. 10. The robot safely navigates through pedestrians in the mall

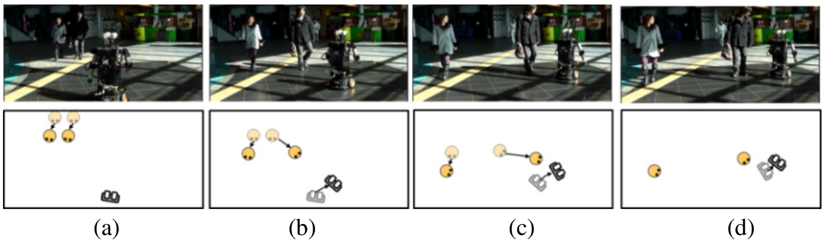


Fig. 11. The robot safely avoids an approaching pedestrian

6 Conclusion

We report our framework to deploy robots in a real shopping mall environment. We used a pedestrian simulator in order to develop and estimate the safety of the robot navigation system among a human crowd. In the simulator we employed a particular specification of the Social Force pedestrian model that has been developed to describe the relatively low-density settings occurring in shopping malls and the like [8]. We further addressed the diverse behavior of pedestrians toward the robot, i.e. we gathered data from a real environment and built a "HRI behavior model" for people slowing down to look at the robot, or approaching and stopping for curiosity, and included such a model in our simulator.

We first tested the developed robot, which is navigated using the same collision avoidance model used for simulated pedestrians, in a simulation to confirm its safety. The results showed that the robot safely navigated among people with reasonable efficiency. Given that the simulation yielded safe navigation for densities up to 0.03 people/m^2, we estimated that we could deploy it in a real world environment with a similar density. To confirm this estimation, we conducted a field trial in a real shopping mall, and the results of this trial demonstrated that the robot can navigate safely among people even when facing complex situations.

Acknowledgements. This work was supported by JST, CREST. We thank the staff of the Asia & Pacific Trade Center for their cooperation and ATR's Yoshifumi Nakagawa for his help.

References

1. Gross, H.-M., et al.: TOOMAS: interactive shopping guide robots in everyday use - final implementation and experiences from long-term field trials. In: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 2005–2012 (2009)
2. Burgard, W., et al.: The Interactive Museum Tour-Guide Robot. In: Proc. National Conference on Artificial Intelligence, pp. 11–18 (1998)
3. Mutlu, B., Forlizzi, J.: Robots in Organizations: Workflow, Social, and Environmental Factors in Human-Robot Interaction. In: Proc. of the 3rd ACM/IEEE Conference on Human-Robot Interaction, pp. 287–294 (2008)

4. León, B., et al.: OpenGRASP: A Toolkit for Robot Grasping Simulation. Springer, Heidelberg (2010)
5. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: a RobotSimulator for Research and Education. In: International Conference on Roboticsand Automation, pp. 1400–1405 (2007)
6. Xu, Y., Mellmann, H., Burkhard, H.-D.: An Approach to Close the Gap between Simulation and Real Robots. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 533–544. Springer, Heidelberg (2010)
7. Glas, D.F., Miyashita, T., Ishiguro, H., Hagita, N.: Laser-based tracking of human position and orientation using parametric shape modeling. Advanced Robotics 23(4), 405–428 (2009)
8. Zanlungo, F., Ikeda, T., Kanda, T.: Social force model with explicit collision prediction. Europhysics Letters 93, 68005 (2011)
9. Tsai, Y.-C., Huang, H.-P.: Motion Planning of a Dual-Arm Mobile Robot in theConfiguration-Time Space. In: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 2458–2463 (2009)
10. Boedecker, J., Asada, M.: SimSpark - Concepts and Application in the RoboCup 3D Soccer Simulation League. In: Proceedings of the SIMPAR 2008 (2008)
11. Sisbot, E.A., Marin-Urias, L.F., Alami, R., Simeon, T.: A Human Aware Mobile Robot Motion Planner. IEEE Transactions on Robotics 23(5), 874–883 (2007)
12. Mainprice, J., Sisbot, E.A., Simeon, T., Alami, R.: Planning Safe and Legible Hand-over Motions for Human-Robot Interaction. In: 2010 IARP Workshop on Technical Challenges for Dependable Robots in Human Environments, Toulouse, France (2010)
13. Helbing, D., Farkas, I., Vicsek, T.: Simulating dynamical features of escape panic. Nature 407, 487–490 (2000)
14. Pelechano, N., Allbeck, J., Badler, N.: Controlling Individual Agents in High-Density Crowd Simulation. In: Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation (2007)
15. Garrell, A., Sanfeliu, A., Moreno-Noguer, F.: Discrete Time Motion Model for Guiding People in Urban Areas using Multiple Robots. In: Proc. of IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, pp. 486–491 (2009)
16. Henry, P., Vollmer, C., Ferris, B., Fox, D.: Learning to navigate through crowded environments. In: Proc. of IEEE Int. Conf. on Robotics and Automation, pp. 981–986 (2010)
17. Shiomi, M., Zanlungo, F., Hayashi, K., Kanda, T.: Navigating Robots among Pedestrians Using a Pedestrian Model. Under Review
18. Kanda, T., Glas, D.F., Shiomi, M., Hagita, N.: Abstracting people's trajectories for social robots to proactively approach customers. IEEE Transactions on Robotics 25, 1382–1396 (2009)
19. Seder, M., Petrovic, I.: Dynamic window based approach to mobile robot motion control in the presence of moving obstacles. In: Proc. of IEEE Int. Conf. on Robotics and Automation, pp. 1986–1992 (2007)

Simulating Complex Robotic Scenarios with MORSE

Gilberto Echeverria[1], Séverin Lemaignan[1,2], Arnaud Degroote[1], Simon Lacroix[1], Michael Karg[2], Pierrick Koch[3], Charles Lesire[4], and Serge Stinckwich[3,5]

[1] CNRS, LAAS, 7 Avenue du Colonel Roche, F-31077 Toulouse, France / Université de Toulouse, UPS, INSA, INP, ISAE, LAAS, F-31077 Toulouse, France
[2] Institute for Advanced Studies, Technische Universität München, D-85748 Garching, Germany
[3] UMR 6072 GREYC Université de Caen-Basse Normandie/CNRS/ENSICAEN, France
[4] ONERA – The French Aerospace Lab, F-31055, Toulouse, France
[5] UMI 209 UMMISCO
IRD/IFI/Vietnam National University, Vietnam
{gechever,slemaign,adegroot,slacroix}@laas.fr, kargm@in.tum.de, pierrick.koch@unicaen.fr, charles.lesire@onera.fr, serge.stinckwich@ird.fr

Abstract. MORSE is a robotic simulation software developed by roboticists from several research laboratories. It is a framework to evaluate robotic algorithms and their integration in complex environments, modeled with the Blender 3D real-time engine which brings realistic rendering and physics simulation. The simulations can be specified at various levels of abstraction. This enables researchers to focus on their field of interest, that can range from processing low-level sensor data to the integration of a complete team of robots. After nearly three years of development, MORSE is a mature tool with a large collection of components, that provides many innovative features: software-in-the-loop connectivity, multiple middleware support, configurable components, varying levels of simulation abstraction, distributed implementation for large scale multi-robot simulations and a human avatar that can interact with robots in virtual environments. This paper presents the current state of MORSE, highlighting its unique features in use cases.

1 Introduction

Simulations are an essential component in robotics research, allowing the evaluation and validation of many sorts of developments before their integration on-board real robots. For such validations to be useful, the simulation must provide enough fidelity with respect to the real world, within the requirements of the considered robotics application. Creating a fully realistic simulation in every aspect remains nearly impossible and sometimes can also be seen as an inconvenience. Therefore numerous simulators are focused on a given specific aspect (*e.g.* terramechanics to study locomotion, contact forces to study manipulation, realistic photo-rendering to study vision algorithms, ...).

The MORSE simulator presented here is not targeted to the study of a specific domain of robotics, but is rather meant as a tool that can be used to test and evaluate *integrated robot software*, *i.e.* the suite of processes required to autonomously achieve complex missions. Using Blender to simulate photo-realistic 3D worlds and the associated physics engine, it brings enough realism to evaluate complete sets of software components within a wide range of application contexts.

The simulator is designed to be useful under varied environments, and provides features to adapt itself to existing robotic architectures, without imposing any changes on them to connect with MORSE. It has modular components that are versatile and configurable, to permit *software-in-the-loop* testing of robotics software. Input from several researchers has guided the evolution of this tool to satisfy the requirements of different labs, while keeping to its design principles. MORSE works on all major software platforms: win32, Linux and OS X. It is developed as an open–source project with a BSD 3-clause license.[1]

The outline of this paper is as follows: Section 2 reviews other comparable robotics simulators, and Section 3 introduces the main design principles of MORSE. Section 4 is the heart part of the article: it depicts the main features of MORSE, each being exemplified with the projects for which they were developed. Finally Section 5 gives the conclusion and future plans for the simulator.

2 Related Work

The Player/Stage/Gazebo[1] is a well known robotics suite. It is a full set of tools that include the Player communication layer and two integrated simulators: Stage [2] is basically a 2D simulator optimized for navigation on flat and closed environments. One of its advantages is the capability to handle very large numbers of simplified robots [3]. However, it is not ideal for more complex scenarios, specially in 3D space. The more recent Gazebo [4] was developed to cover these shortcomings. It has received an important boost in its development with its integration into the ROS platform, and has thus become the most commonly used robotics simulator. It integrates very well with ROS and Player, but connectivity with other middlewares requires additional programming.

The Unified System for Automation and Robot Simulation (USARSim)[5] shares many design concepts with MORSE. It was initially developed as a simulator for search and rescue operations, uses the Unreal Engine gaming platform and is built with the concept of modular components. It is widely used as an evaluation tool for the well known RoboCup competitions. However, it uses an adhoc interface to communicate with external software and does not support

[1] User documentation and additional information are available at `http://morse.openrobots.org`, and the source code can be downloaded from `http://github.com/laas/morse.git`.

some of the most common robotics middlewares. The Unreal Development Kit (UDK) used is a closed-source library and is not available on every platform.

Webots [6] is a popular commercial simulator. It provides a full programming environment to create customized robots and environments, but the interface to construct new components and robots is unintuitive and complex, requiring searching trough data trees to adjust physical and geometrical parameters. It also has an integrated code editor, but the control programs created in it must later be converted and transferred to the final robot platform.

Another commercial simulator is V-REP [7], with a large variety of components and sensors available. It allows scripting of the components using the LUA language. Being modular, it allows for copy-and-paste functionality of the components. However, it provides no native method for communication with middlewares; such functionality must be provided by the user in the form of add-ons.

3 MORSE Architecture Overview

The Modular Open Robots Simulation Engine (MORSE) is a library of Python scripts that run on the Blender Game Engine (BGE) to interface a 3D virtual environment with external robotics software. Its main design principles have been described in [8], they are quickly recalled here.

The BGE is used as the base platform for the graphical display of the simulated world and the physics simulation, using the Bullet physics library. The simulation is organized as a main core of control functionality that initializes and coordinates the events in the BGE, and a collection of components that can be used to assemble a robot with sensors and actuators. A variety of communication tools allow each of the MORSE components to connect with external robotics software through middlewares. MORSE has support for some of the most commonly used robotics middlewares nowadays, including ROS [9], Pocolibs [10] and YARP [11]. Other middlewares are regularly being added, mainly driven by the needs of new users. This process is simple, as existing middleware bindings can be used as templates for the new ones.

MORSE sensors and actuators are minimalistic in their functionality and completely middleware agnostic. They can provide similar data as their real world counterparts, but some of them have multiple variants that can work at different levels of realism and abstraction. The data employed by the components can be altered through the use of components called *modifiers*. These add noise or change the data as required to better match the data with real sensors/actuators (for instance, transforming the coordinate frame used by Blender into the Universal Transverse Mercator coordinate system, or adding noise to images captured by Blender cameras). When a simulation scene is created, the components are linked to middlewares as specified in a configuration script, and the necessary functionality is added to the components to be able to transmit/receive data through the middleware as data streams or synchronous/asynchronous requests. The whole data flow in MORSE can be seen in the diagram of Fig. 1.

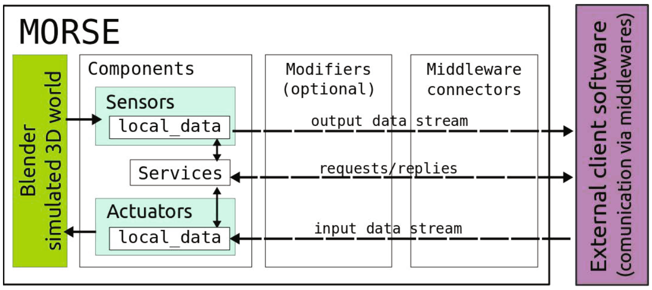


Fig. 1. Overview of the data flow in MORSE, between the simulated world in Blender and the external robotics software

MORSE can be easily extended by adding new components that inherit from the existing base classes. All components are programmed in Python, since this is the language that Blender uses for its scripts. However, when coding components that require a faster processing time, they can be implemented in C/C++ and interfaced with Python using Swig.

4 Morse Main Features

4.1 Blender Integration

Being based on the Blender 3D [12] modeling software, any object, robot or environment can be created and immediately be made ready to use in MORSE. Many file formats for 3D models can be imported directly into Blender, further increasing the range of elements that can be used. The high level of graphical detail is specially useful for realistic looking simulations, and very important when doing image processing from simulated video cameras. It is convenient to

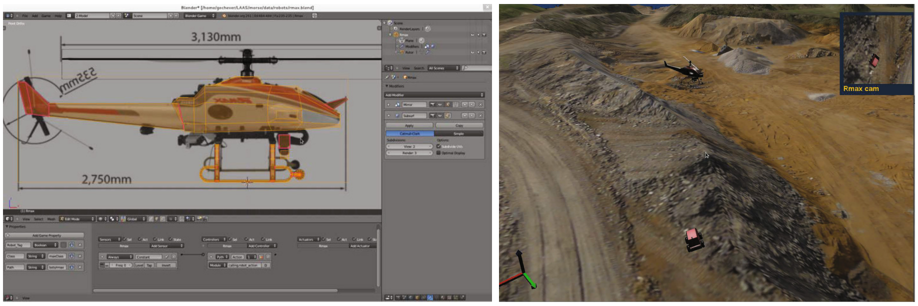


Fig. 2. Left: Modeling of an R-Max robotic helicopter. Right: Robots on a terrain imported from a Digital Elevation Map; the view from a camera mounted on a helicopter robot is shown in the top right frame.

have simulation environments recreating the terrain and buildings of real experimentation sites. Transforming the 2D map of a building into a 3D model is straightforward in Blender. However, outdoor terrains for experiments in *field robotics* are more difficult to create by hand. We have developed additional plug-ins for Blender that read Digital Elevation Map (DEM) files and create the appropriate 3D meshes. Adding normal maps and aerial photography for textures creates very realistic simulated environments. Figure 2 shows the modeling capabilities of Blender exploited in MORSE to create robots and environments from DEM data.

4.2 Component Library and Scene Construction

MORSE provides a collection of components that can be used to construct robots with various configurations. *Sensors* recover data from the simulated world and store it. This is done using the Blender predefined interfaces for the BGE and Python scripts. For example, GPS read the absolute coordinates of the sensor in the Blender world, laser scanners use ray tracing operations of z-buffers to generate range images and cameras use renders from the perspective of the Blender camera to produce images. Parameters that determine how the sensors gather data can be adjusted, such as range, resolution, image sizes, etc.

Actuators carry out actions within the simulated world. They provide motion to the robots through several methods and algorithms, or control moving parts such as robotic arms or pan-tilt units. In most cases, the motions are implemented using the functions in Blender to add a certain linear and angular speed to an object. Robotic arms consist of a series of articulated segments, and use the Blender armature system to determine the degrees of freedom at each joint. They can be operated either by direct control of the angles at each joint or by control of the end effector with Inverse Kinematics (using iTaSC [13], a constraints-based IK solver available for the BGE).

Robots are the platform upon which all sensors and actuators must be mounted in order to function. They currently have no behavior of their own, but define the physical shape and properties like mass, friction and collision bounds used by the Bullet physics library to compute the interactions with the environment.

The component library also includes other objects that can be used in the simulation, from large outdoor environments, to furniture and small items. All of these can be imported from individual Blender files into a simulation scenario.

Building and configuration of scenarios are done with a set of Python classes provided by MORSE, known as the *Builder API*. The Builder API provides an internal domain-specific language (DSL) that completely hides the somewhat complex interface of Blender from the user, so that those unfamiliar with Blender can directly configure MORSE using Python scripts. Scripts can then be tracked on a version control system to follow changes and apply patches. The API offers classes to add robots, sensors and actuators, to position, rename and configure any of the parameters used by these components, to include additional objects (furniture, obstacles, etc.) and to add middleware bindings, modifiers and services for each component.

4.3 Adjustable Levels of Realism

Many components come in several versions, which produce more or less realistic behavior, and can be chosen depending on the objective of the simulation and processing that will be applied to the data.

Ground robots can consist of rigid bodies for chassis and wheels using physical constraints to give realistic movement based on the speeds of the wheels. Alternatively, some robots are handled as a single rigid box sliding over the ground when the details of the motion of the robot are not important. For aerial and submarine robots the physics are limited to collision detection, but they are not affected by gravity, and there is no simulation of air or wind currents – but external dynamic models can be linked to MORSE for such purposes. The associated actuators offer the choice between controlling the speeds of wheels independently, controlling the motion of a robot as a whole with linear and angular velocities, giving direct waypoint coordinates for destination or "teleporting" a robot to a desired location.

Some sensors provide almost identical raw data as their real counterparts, which can then be processed to extract relevant information, and later used to make decisions. When the processing of data is not of interest to the experiment, alternative sensors can be used that provide higher level data, extracted directly from the BGE scenario, and avoid costly processing. The clearest example of this are the cameras: the regular video camera generates renders of the Blender scene, and outputs images to be processed by software. This generates much data, and can slow down the simulation when rendering for several cameras at once. A new sensor called the *semantic camera* uses Blender functions to determine the names and global 3D position of the objects within the view frustum of the camera. It directly outputs this data, making the simulation of object detection faster and avoiding the use of image processing algorithms.

As a practical example, researchers from ONERA use MORSE to evaluate online probabilistic decision-making on an autonomous UAV. In a first scenario [14], MDP-based policy optimization allows the UAV to find a suitable zone for an emergency landing. MORSE video camera sensors are used to feed the raw image data into the land mapping algorithm. On a second scenario [15], a POMDP model is used to optimize the probability to track and intercept an identified target among others. As the objective is to evaluate the optimized strategy, processing simulated images is not necessary. Hence, instead of using a video camera sensor, the *semantic camera* is used to identify the location of the target, adding some errors with modifiers to simulate both the sensor and the detection and identification processes.

4.4 Middleware Configurations

Research laboratories use different middlewares to connect the processes within their robots. In order to be usable with any architecture, MORSE enforces a clear separation between components and middlewares. When a simulation is started, the bindings between sensors/actuators and middlewares are read from the configuration script, and it is only then that components acquire the functions necessary to communicate with the outside world. This separation means that any middleware can be integrated with MORSE, by simply providing the scripts to marshal MORSE data in the expected format. Middlewares currently supported include ROS, MOOS, Pocolibs, YARP and raw TCP/IP sockets. Information exchange between the simulator and the robotics software can be done by a constant data stream, or via request/reply interfaces.

MORSE permits using different middlewares within a single simulation scenario with heterogeneous systems. This is exemplified in the *Action project* [16] developed by the LAAS and ONERA labs. It consists in the cooperation of ground and air robots to patrol a zone, locate and follow moving targets. Each lab provides an existing robotics platform with two different architectures: LAAS employs Segway RMP 400 land robots, using the G^{en}oM [17] architecture, while ONERA works with Yamaha R-Max helicopters based on OROCOS [18]. The simulation uses Pocolibs to communicate with G^{en}oM , and YARP to talk with OROCOS.

MORSE has been chosen as the simulator platform for a French research robotic project called *ANR-PROTEUS*, thanks to its middlewares versatility and the facility to integrate it with existing architectures. The PROTEUS project supports a model-driven engineering approach based on a domain-specific language and a robotic ontology [19] and aims at providing a toolchain for robotic development from modeling to software simulation and deployment on real robots. PROTEUS is based on open-source softwares like the Eclipse Modeling Project, the ROS middleware and MORSE simulator. Several robot models are currently under construction (Wifibot, ECA Camelon, and Thales R-Trooper).

4.5 Component Overlays for Specific Software Architectures

Besides the components and middlewares available, it is further possible to adapt these elements to better fit an existing architecture and allow true *software-in-the-loop* functionality. MORSE components provide dedicated I/O interfaces. In many cases, the interface methods will not be the same as those used in an actual device, although the component has the same functionality. For instance, an actuator may have two separate functions to modify its linear and angular velocities, while the corresponding MORSE component uses only one function with the two parameters.

To avoid creating additional components, MORSE implements the notion of *component overlays*. Following the well-known adapter pattern, they are implemented as special components that override (or wrap) the default behavior to make it match a different architecture. In the case of middlewares, it is possible to either use the default serialization methods provided by MORSE, or write additional serialization functions as extra scripts, specific for single components. Overlays are implemented as Python scripts that provide an interface between the real component being simulated and the equivalent sensor available in MORSE. These features have been used to connect MORSE with an existing Unmanned Aerial Vehicle (AUV) control architecture [20] without modifying the code in either the original architecture or MORSE.

4.6 Multi-node and Hybrid Simulation

The simulation of multiple robots in the complex environments permitted by MORSE is very demanding on computational and graphics resources. A scenario with several robots equipped with video cameras and other sensors, plus the robotics software all running on the same computer will slow down the system considerably. MORSE offers the possibility of running multiple instances of the same simulation scenario in separate computers, coordinated by a central server program, called the *multi-node manager*. This manager has been implemented using TCP/IP socket communication, and also using High Level Architecture (HLA) [21] for more strict time management. While every node shows all of the robots, each node will only be in charge of controlling a limited number of them. The movements of robots and specific objects in one node are sent to the multi-node manager, which in turn collects the updated positions across all nodes and redistributes the information, so that all nodes can immediately reflect the changes. The multi-node server also synchronizes the time and events across all nodes. This can be used to slow down the simulation in all nodes, by making them wait for the synchronization message. However, at the current time it is not possible to accelerate the simulation speed from the multi-node server.

This functionality was developed for applications that require a large amount of robots. Our use case is the rescue robotics project *Rosace* [22]. Its main objective is the coordination of robotic agents in search and rescue operations in the case of a disaster. In this scenario, terrestrial robots must be able to locate human victims, provide support for the victims and avoid dangerous areas. The robots in the team are to be equipped with different payloads, and take autonomous decisions on which of them should perform different tasks in the mission, such as searching, providing a communications relay, and helping victims directly. For this project, two or three robots can be handled by a single simulation node, and synchronized with those in other nodes. Special sensors in MORSE are used to determine the distance and visibility between robots, and this information is used to simulate loss of connectivity between them. The victims to be saved are internally considered as robots with scripted behavior, so that their status and position is also synchronized by the multi-node manager.

Additionally, the multi-node system permits the deployment of hybrid simulations. A 3D environment that closely represents the real experimentation site is used. Real robots report their updated positions to the multi-node server, which reflects the changes on a dummy robot in the simulation, so that other simulated robots can see it and interact with it. In the Action project, a real ground robot moves around while the simulated helicopter can follow it using video cameras.

The robots communicate in the same way in full simulation or in real life. The use of HLA makes it possible to synchronize the heterogeneous systems in both the simulated and the real world. Note however that the real robots are not able to "see" the simulated ones with their cameras – but this could be achieved using an augmented reality scheme that would modify the data gathered by the real robots.

4.7 The Human Avatar

For human-robot-interaction scenarios, we require a way to combine the reactive and sometimes unpredictable behavior of a human interacting with its environment with a simulated robot. Therefore the human avatar of the MORSE simulator has been equipped with an intuitive control that enables users to act upon the simulated environment. Inspired by modern 3D-computer games, the user takes a third-person perspective behind the human avatar to move around as shown in Fig. 3 (left). While moving around, the camera tries to avoid the objects and walls placed between the camera and the human avatar to prevent occlusions. All objects that can be interacted with can be displayed by pressing a key on the keyboard (also illustrated in Fig. 3). When the user decides to interact with an object, the camera switches to a first-person perspective and offers an interface showing possible actions the user can take when pointing to specific objects, as shown in Fig. 3 (right). Those actions at the moment include picking up and releasing objects, opening and closing drawers and cupboards and switching on and off specific objects like a light or an oven.

The motions of the avatar are animated using Blender armatures, inverse kinematics, and predefined movement loops. The avatar can be controlled much like a character in a videogame, using either the mouse and keyboard or a combination of the Microsoft Kinect and the Nintendo Wiimote. This enables users

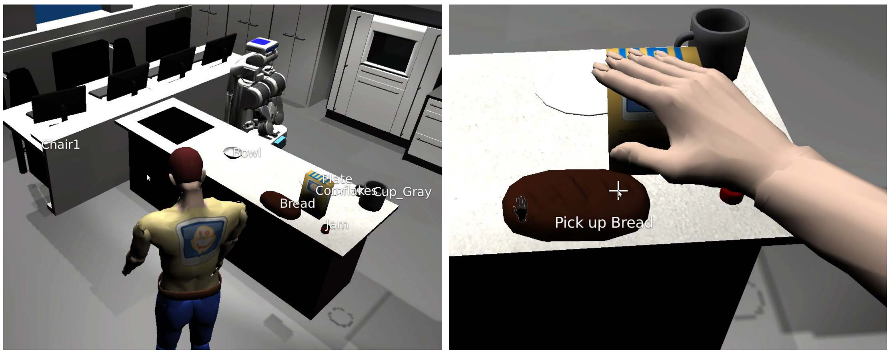


Fig. 3. Left: third-person view of the human component that is used to navigate in the environment, displaying the names of objects that the human can interact with. Right: first-person perspective of the human avatar that indicates a possible "pick-up-action" with the bread.

to perform pick and place actions in simulated worlds while at the same time the simulated robot(s) can be controlled through the supported robotic middlewares.

The human avatar is meant to be used in *personal robotics* scenarios. In these, complex robots are expected to collaborate with humans to carry out ordinary household tasks, such as cleaning, serving food or aiding humans to navigate an environment. Robots used in these experiments are equipped with one or two arms, and are capable of grasping objects. They are also expected to react to the actions of their human collaborator, using video cameras, motion detectors or telemetry to determine the location, pose and attitude of the human. This can be done at two different levels of abstraction. Using the video cameras to recognize the human and its pose can be done realistically, with the associated computational cost and uncertainties. Alternatively, the avatar can directly export the position of all of its joints, and feed them back to the robot, simulating a full motion capture system and avoiding the processing costs.

An example use case in this scenario is the testing and validation of human-aware navigation planners of service robots in human-centered environments at TU Munich. In this case, a simulated human tracking system provides the human pose to the robot while the robot navigates in the environment in a way that is safe and legible for the human [23]. In this project, MORSE has not only been used as a powerful tool for testing human-aware navigation strategies before carrying out experiments with real humans: it has also been used to evaluate them as Lichtenthäler et al. show in [24] by video-based user studies. After successful testing and evaluation, the human-aware navigation was applied using real robots and humans resulting in a safe and reliable behavior of the robot.

5 Summary and Future Work

We have presented the main features developed within the MORSE simulator, following the requirements of a variety of projects in robotics research. The design and architecture of MORSE has proved to be flexible and powerful enough to allow researchers to use it under various circumstances to test their robotics algorithms. Users can customize the existing components according to their needs, or develop new ones when necessary, by describing their behavior in Python scripts. All new developments done on top of MORSE are made available to the whole community, thanks to the open source license of the simulator.

MORSE allows for quick integration with an existing architecture (multiple middlewares, modular components and component overlays), heterogeneous robots (mixing components and middlewares), multiple robot simulation (multi-node synchronization) and human-robot interaction (high abstraction level sensors and human avatar).

Further work is planned to increase the usefulness of MORSE: for human-robot-interaction experiments, it is ideal to have a deeper immersion when using the human avatar. A planned feature is to provide stereo images of the simulation to the user wearing special goggles. This can trivially be done from Blender, with some separation of the images produced for the two eyes. Increased integration with motion reading devices, such as Microsoft Kinect can also make the experience more natural, by allowing a finer control over the human avatar. Another planned work is to be able to couple MORSE simulations with other physical simulation engines thanks to the HLA support. Further development is also necessary to give users more control over the speed of the simulation, by adjusting the base execution frequency of the BGE.

Acknowledgments. This work has been partially supported by the DGA funded Action project (http://action.onera.fr), the STAE foundation Rosace project (http://www.fondation-stae.net) and the ANR-PROTEUS project (http://anr-proteus.fr/).

References

1. Rusu, R.B., Maldonado, A., Beetz, M., Gerkey, B.P.: Extending Player/Stage/Gazebo towards cognitive robots acting in ubiquitous sensor-equipped environments. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA) Workshop for Network Robot Systems, Rome, Italy (2007)
2. Gerkey, B.P., Vaughan, R.T., Howard, A.: The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In: Proceedings of the 11th International Conference on Advanced Robotics, pp. 317–323 (2003)
3. Vaughan, R.: Massively Multi-robot Simulation in Stage. Swarm Intelligence 2, 189–208 (2008)
4. Koenig, N., Howard, A.: Design and Use Paradigms for Gazebo, An Open-Source Multi-Robot Simulator. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2149–2154 (2004)
5. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: a robot simulator for research and education. In: Proceedings of the 2007 IEEE Conference on Robotics and Automation, pp. 1400–1405 (April 2007)
6. Michel, O.: Webots: Professional mobile robot simulation. Journal of Advanced Robotics Systems 1(1), 39–42 (2004)
7. Freese, M., Singh, S., Ozaki, F., Matsuhira, N.: Virtual Robot Experimentation Platform V-REP: A Versatile 3D Robot Simulator. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 51–62. Springer, Heidelberg (2010)
8. Echeverria, G., Lassabe, N., Degroote, A., Lemaignan, S.: Modular open robots simulation engine: Morse. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 46–51 (May 2011)
9. Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
10. Pocolibs: Posix communication library, http://pocolibs.openrobots.org
11. Metta, G., Fitzpatrick, P., Natale, L.: YARP: yet another robot platform. International Journal of Advanced Robotic Systems 3(1), 43–48 (2006)
12. Blender 3D, http://www.blender.org/
13. Smits, R., De Laet, T., Claes, K., Bruyninckx, H., De Schutter, J.: iTASC: a tool for multi-sensor integration in robot manipulation. In: IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pp. 426–433 (August 2008)

14. Teichteil-Königsbuch, F., Lesire, C., Infantes, G.: A Generic Framework for Anytime Execution-driven Planning in Robotics. In: Proceedings of the 2010 IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, pp. 299–304 (2011)
15. Carvalho Chanel, C.P., Teichteil-Königsbuch, F., Lesire, C.: POMDP-based online target detection and recognition for autonomous UAVs. In: Proceedings of the 7th Conference on Prestigious Applications of Intelligent Systems (PAIS), Montpellier, France (2012)
16. Boumghar, R., Lacroix, S., Lefebvre, O.: An information-driven navigation strategy for autonomous navigation in unknown environments. In: 2011 IEEE International Symposium on Safety, Security, and Rescue Robotics (SSRR), pp. 172–177 (November 2011)
17. Mallet, A., Herrb, M.: Recent developments of the GenoM robotic component generator. In: 6th National Conference on Control Architectures of Robots, Grenoble, France. INRIA Grenoble Rhône-Alpes (May 2011)
18. Bruyninckx, H., Soetens, P., Koninckx, B.: The Real-Time Motion Control Core of the Orocos Project. In: Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), pp. 2766–2771 (September 2003)
19. Dhouib, S., du Lac, N., Farges, J.-L., Gerard, S., Hemaissia-Jeannin, M., Lahera-Perez, J., Millet, S., Patin, B., Stinckwich, S.: Control architecture concepts and properties of an ontology devoted to exchanges in mobile robotics. In: Proceedings of the 6th National Conference on "Control Architecture for Robots" (2011)
20. Barbier, M., Gabard, J.-F., Bertholom, A., Dupas, Y.: An Onboard Software Decisional Architecture for Rapid Environmental Assessment Missions. In: Proceedings of the 18th IFAC World Congress, Milano, Italy, pp. 11797–11802 (2011)
21. Kuhl, F., Weatherly, R., Dahmann, J.: Creating computer simulation systems: an introduction to the high level architecture. Prentice Hall PTR, Upper Saddle River (1999)
22. Lacouture, J., Gascueña, J., Gleizes, M.-P., Glize, P., Garijo, F., Fernández-Caballero, A.: Rosace: Agent-based Systems for Dynamic Task Allocation in Crisis Management. In: Demazeau, Y., Müller, J.P., Rodríguez, J.M.C., Pérez, J.B. (eds.) Advances on PAAMS. AISC, vol. 155, pp. 255–259. Springer, Heidelberg (2012)
23. Kruse, T., Kirsch, A., Sisbot, E.A., Alami, R.: Exploiting human cooperation in human-centered robot navigation. In: IEEE International Symposium in Robot and Human Interactive Communication, Ro-Man (2010)
24. Lichtenthaeler, C., Lorenz, T., Karg, M., Kirsch, A.: Increasing Perceived Value Between Human and Robots - Measuring Legibility in Human Aware Navigation. In: IEEE Workshop on Advanced Robotics and its Social Impacts (2012)

Masters' Skill Explained by Visualization of Whole-Body Muscle Activity

Yosuke Ikegami, Ko Ayusawa, and Yoshihiko Nakamura

The University of Tokyo, 7-3-1, Hongo, Bunkyo-ku, Tokyo, Japan
ikegami@ynl.t.u-tokyo.ac.jp

Abstract. In this paper, we discuss the computation of human motion dynamics and its analysis of experts' motion skills. The computation framework of the wire-driven multi-body dynamics previously developed by the authors is applied to the whole body human musculoskeletal model. While capturing time-series of motion data, somatosensory information measured by force plate and EMG sensors simultaneously is used for the dynamics computation. For the dynamics computation, we reduced the computation cost drastically by resolving to the non-linear optimization problem using decomposed gradient computation developed recently. As examples of analysis, we measured and analyzed experts' motion patterns, such as Tai Chi motion, tap dance and drum playing. In particular, we analyzed the characteristic behavior by the motion of the center of gravity (COG), condition of the ground contact, and muscle activity of the whole body.

Keywords: Musculoskeletal Model, Muscle Tension Estimation, Motion Visualization, Tai Chi, Tap Dance, Drum.

1 Introduction

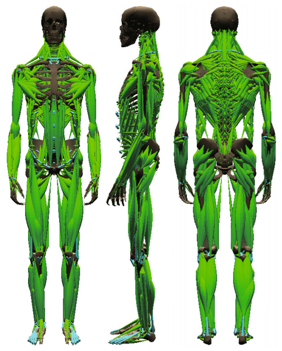
Since ancient times, physical skills acquired after a long period of time were transmitted to the next generation through trial and error. This is due to a problem that it is difficult to quantify the movement. In recent years, on the basis of development of large scale of the dynamics computation[1] and technique of the somatosensory measurement, quantification and visualization of motor skills are becoming available[2,3,4]. Muscle tension estimation and visualization system for physical activity during exercise has been developed using optical motion capture system, force plate and EMG sensors[5]. These techniques are applicable for the analysis of the mammal motion data and considered that its applicability is high [6].

There exists the technique that gives subject the real time visualization feedback of the muscle tension estimation using simplified human musculoskeletal model[7,8]. However, in the case of the detailed analysis for the complex model, computation cost was quite high due to the large degrees of freedom (DOF). The method for high speed inverse kinematics and inverse dynamics calculation reduce the calculation time[9][10], more motion data can be analyzed in recent days. Using these techniques, we analyzed the motion of expert that has been difficult to quantify and visualize so far. In this paper, we present an overview of the technology of motion analysis as a summary of previous technology. Three motions which are (1) Tai Chi, (2) Tap dance and (3) Drum playing are analyzed based on above technique and obtained findings are described respectively.

2 Musculoskeletal Model and Experimental Environment

2.1 Musculoskeletal Model

Figure 1(a) shows whole body musculoskeletal model developed by Nakamura et al[5]. Skeleton consists of a set of bones grouped into a suitable number of the body parts. Bones are treated as a set of rigid bodies with mass and inertia. Each Joint is approximated using spherical or rotational ones. Muscles, tendons, and ligaments are defined as wires without any mass placed on the skeletal. Wire is defined to vary only in the contractive direction in a general rule. The muscle elements are modeled as active constrictive wire actuators, and the others are modeled passive constrictive wires. The system can be treated as a virtual open tree-structure kinematic chain, assuming that the system is cut open if it has closed loops consist of multiple bones such as knee or ankle joints. The passive spring with zero nominal length is placed on the cut part, corresponding to cartilages that connect multiple bones and constrain their relative movement. The spring forces both contractive and expansive directions. If muscles, tendons, and ligaments have furcation, we place virtual links on the furcation. A virtual link is modeled as a small link without mass. With virtual links, each element can be separated into the several wires which have only one pair of origin and end


(a) Human Musculoskeletal Model

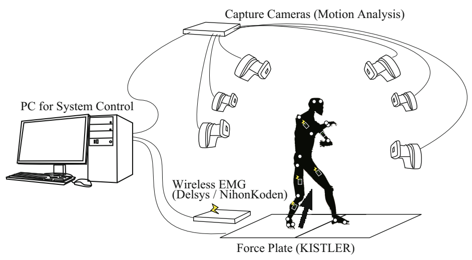

Capture Cameras (Motion Analysis)
PC for System Control
Wireless EMG (Delsys / NihonKoden)
Force Plate (KISTLER)
(b) Overview of the motion capture system

Fig. 1. (a) Human Musculoskeletal Model: Skeletal consists of 200 bones, 53 group of bones, 72 virtual links. 323 DOF in total(155 skeletal DOF).997 muscles, 50 tendons and 125 ligaments, and 34 cartilages. 1206 wires in total. (b)Overview of the motion capture system consist of 10 cameras (200fps), 16ch wireless EMG, and 3 (10 for drum playing motion capture) force plates.

points. The under actuated 6 DOF joint (or free joint) is located on each virtual link so that the sum of forces and moments acting on the virtual link are equal to zeros. For motion analysis, it is necessary to create a suitable individual model for each subject. The calculation of the individual kinematic parameters is proposed by authors using time series of motion data[11]. In this study, musculoskeletal model is scaled based on the comparison between measured marker length and modeled marker length for each grouped bone.

2.2 Experimental Environment

Figure 1(b) shows the overview of the motion capture system. By optical motion capture system (motion analysis), time series data of body marker trajectories (34 trajectories) are measured. Floor reactive force and surface EMG are measured simultaneously with motion capture system using faceplate (Japan Kistler) and wireless surface EMG sensors (Delsys, Nihon Kohden). For the consideration of external force onto the rigid body, the timing of the contact with external environment has to be known. In this study, the contact timing and joint are recorded by high speed cameras (300-600 fps) synchronized with motion capture system at the same time.

3 Optimization Problem for Muscle Tension Estimation

3.1 Overview of the Calculation

To estimate whole body muscle tension, the following three types of optimization problem is solved in order.

1. Inverse kinematics: Optimization calculation determines the generalized coordinates of the joint to reproduce the trajectory of the marker measurement while satisfying the wire (muscle and tendon) constraints.
2. Contact force estimation: Optimization calculation to determine the external forces on each contact joint to reproduce all external force calculated from the generalized coordinates and the known inertial parameters, and to reproduce the measured value of the total floor reaction force simultaneously.
3. Inverse dynamics: Optimization calculation determine the tensile strength (muscle, tendon, etc) of each wire to reproduce the joint torque calculated with the generalized coordinate, contact forces and known inertial parameters, and to reproduce the measured values from the EMG data.

The evaluation function and constraints of these optimization problems are formulated in the following subsection. Solution of the optimization problem itself are omitted here, however they can be solved applying general nonlinear programming method[9,10]. Solution of nonlinear programming can be significantly faster using fast kinematics and dynamics calculation for multi-body system developed in robotics field. See the reference for concrete method of the fast calculation[9,10].

3.2 Inverse Kinematics

The following optimization problem is calculated in each sample frame.

$$\min_{\boldsymbol{q}} \quad \frac{1}{2}\sum_{i=1}^{N_M} ||\widehat{\boldsymbol{p}}_i - \boldsymbol{p}_i(\boldsymbol{q})||^2 \tag{1}$$

$$\text{subject to} \quad l_j(\boldsymbol{q}) < \widehat{l}_j \quad (0 < j < N_T) \tag{2}$$

Where,

- N_J is total DOF of all joints,
- N_M is total number of markers,
- N_T is total number of wires(tendons etc) except muscles,
- $\boldsymbol{q} \in \mathbb{R}^{N_J}$ is generalized coordinate,
- $\boldsymbol{p}_i(\boldsymbol{q}) \in \mathbb{R}^3$ is position of each marker referred from generalized coordinate $\boldsymbol{q}$,
- $\widehat{\boldsymbol{p}}_i \in \mathbb{R}^3$ is each measured marker position,
- $l_i(\boldsymbol{q}) \in \mathbb{R}$ is each wire length referred from generalized coordinate $\boldsymbol{q}$,
- $\widehat{l}_i \in \mathbb{R}$ is each wire natural length.

$\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ is calculated by numerical differentiation.

3.3 Contact Force Estimation

The following optimization problem is calculated in each sample frame using $\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}$ obtained in inverse kinematics.

$$\begin{aligned}\min_{\boldsymbol{F}_c} \quad & \frac{k_{c1}}{2}||\boldsymbol{F}_0(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}) - \boldsymbol{K}_C(\boldsymbol{q})\boldsymbol{F}_c||^2 \\ & + \frac{k_{c2}}{2}||\boldsymbol{F}_{fp} - \boldsymbol{K}_{fp}(\boldsymbol{q})\boldsymbol{F}_c||^2 + \frac{k_{c3}}{2}||\boldsymbol{F}_c||^2 \end{aligned} \tag{3}$$

$$\text{subject to} \quad \boldsymbol{C}_z\boldsymbol{F}_c \geq \boldsymbol{0} \tag{4}$$

- N_C is the number of the grounded links at the frame,
- N_F is the number of the force plate,
- $\boldsymbol{F}_c \in \mathbb{R}^{6N_C}$ is vector representing the contact forces and moments applied to all grounded links,
- $\boldsymbol{F}_0 \in \mathbb{R}^6$ is the total external force and moment calculated from the inertial model and generalized coordinates (referred from the base link coordinate),
- $\boldsymbol{K}_C \in \mathbb{R}^{6\times 6N_C}$ is a projection matrix to all external force and moment from contact force and moment of each link (referred from the base link coordinate),
- $\boldsymbol{F}_{fp} \in \mathbb{R}^{6N_F}$ is the measured force and moment acting on the force plate,
- $\boldsymbol{K}_{fp} \in \mathbb{R}^{6\times 6N_C}$ is projection matrix from contact force and moment of each link to contact force and moment referred from the grounded force plate coordinate,

- $\boldsymbol{C}_z \in \mathbb{R}^{N_C \times 6N_C}$ is the matrix to extract normal force component at the contact surface from $\boldsymbol{F}_c$,
- $k_{c1}, k_{c2}, k_{c3} \in \mathbb{R}$ is the weight coefficient for optimization.

The first term in the formula describes cost term of all external forces and moment estimated from human inertial parameters and time-series of joint motion data, the second term describes evaluation value with force plate data, the third term describes evaluation value to determine the indefinite component omitting the influence of the friction force except contact direction, inequality constraint describes the contact force should not be negative along the normal direction on the contact surface. Calculating above quadratic programing problem, $\boldsymbol{F}_c$ is obtained.

3.4 Inverse Dynamics

The following optimization problem is calculated in each sample frame using obtained $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$, $\boldsymbol{F}_c$.

$$\begin{aligned}
\min_{\boldsymbol{f}} \quad & \frac{k_{f1}}{2}||\boldsymbol{\tau}(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}}, \boldsymbol{F}_C) - \boldsymbol{K}_f(\boldsymbol{q})\boldsymbol{f}||^2 \\
& + \frac{k_{f2}}{2}||\boldsymbol{f}_{emg} - \boldsymbol{C}_{emg}\boldsymbol{f}_c||^2 + \frac{k_{f3}}{2}||\boldsymbol{f}||^2 \qquad (5) \\
\text{subject to} \quad & \boldsymbol{f} \leq \boldsymbol{0} \qquad (6)
\end{aligned}$$

- N_W is total number of wires(muscles or tendons, and etc),
- N_{emg} is total number of EMG sensors,
- $\boldsymbol{f} \in \mathbb{R}^{N_W}$ is vector representing tensions of all wires,
- $\boldsymbol{\tau} \in \mathbb{R}^6$ is joint torque obtained from inertial parameters and generalized coordinate and contact forces,
- $\boldsymbol{K}_f \in \mathbb{R}^{N_J \times N_W}$ is projection matrix from wire tension to joint torque.
- $\boldsymbol{f}_{emg} \in \mathbb{R}^{N_{emg}}$ is physiological muscle tension measured with EMG sensors,
- $\boldsymbol{C}_{emg} \in \mathbb{R}^{N_{emg} \times N_W}$ is the matrix extracting the muscle tension measured in EMG sensors from force vector $\boldsymbol{f}$,
- $k_{f1}, k_{f2}, k_{f3} \in \mathbb{R}$ is weight coefficient for optimization.

The first term in the formula describes evaluation term with input joint torque estimated from human inertial model, joint trajectory, and contact forces, the second term describes evaluation term with muscle tension forces obtained from EMG measurement and physiological model. Hill-Strove model[12,13] is used to calculate $\boldsymbol{f}_{emg}$ from EMG signals as a physiological model. The third term describes evaluation term to determine indefinite force component while all muscle relaxing as possible. The constraint is the condition of the force contractive direction.

After the computation of whole muscle tensions, the muscle activity of each muscle that EMG is not attached can also be estimated from the obtained tension and the physiological model.

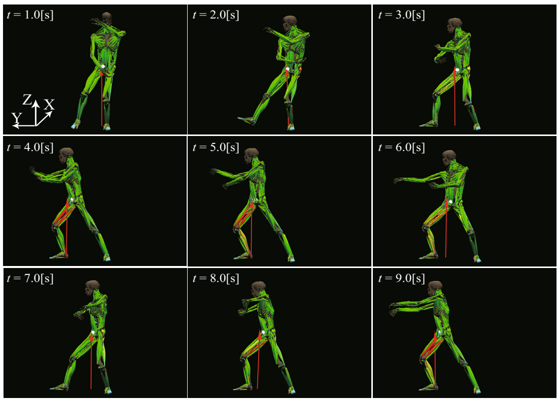
4 Experts' Motion Skills

4.1 Tai Chi Motion

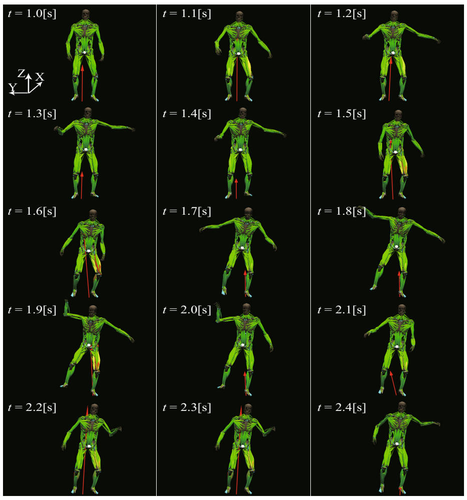
Tai Chi motion is characterized by a slow and flowing movement. It is said that to focus on muscle on the lower limbs and to keep a lower center of gravity (COG) is important. 40 seconds dance performance was analyzed. Orientation of the trunk is reversed before and after each 20 seconds and 35 seconds (right foot forward to left foot forward). Figure 2(a) shows the snapshots of the calculation result of muscle tension analysis. Figure 3(a) shows changes in the position of the COG and major muscle activities during movement. Graph shows (a) anterior deltoid (b) posterior deltoid (c) the long head of biceps (d) triceps lateral head (e) rectus femoris muscle (f) the long head of biceps femoris muscle (g) tibialis anterior muscle (h) lateral head of gastrocnemius muscle respectively.

The following considerations are described based on the analysis results.

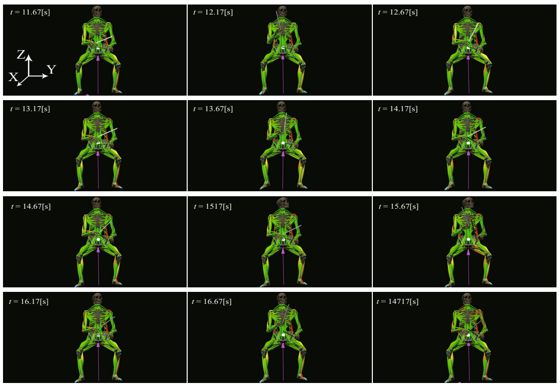
- Height of the COG is low and movement is stable. Height of the COG is low compared with standing posture, shown in figure 2(a). As we can see in figure 3(a), the maximum position of the Z-axis direction is 50mm at most during 0-30[s] that is substantially parallel to the XY plane. Considering the state of the lowering the height of COG, the movement is to be said very stable.
- Displacement in the axial direction Y of the COG is periodic. In each period, COG is moved from vertical position on the right foot to left foot that is similar to walking behavior in the steady state. In addition to this, periodical movement of the COG along with X-axis direction can be seen. That is because the subject orientation has been reversed after 20 seconds. Eight periodic cycle can be confirmed during 40 seconds motion. One period time is about 5 seconds, which is rather slow@compared to other movement.
- Muscles of the lower limbs are working actively. There is a relationship between the periodicity of the motion in the axial direction Y of the COG and the periodicity of the motion of the rectus femoris, tibialis anterior, and gastrocnemius. The movement of Y-axis positive direction corresponds to muscles of the right side of the body, negative movement corresponds to left side of the body. Biceps femoris works only around 20 and 35 seconds to turn the body.
- Muscles in upper body works less compare to the muscles in lower limb. Especially, triceps does not work well. Motion under the weakness condition is presumed. However deltoid and biceps works periodically, both arms are used to move COG. That is, the subject moves forward with both hands spread, move back with close. That Can be interpreted which forward movement represents attack movement, and back movement represents defense movement.



(a) Tai Chi motion

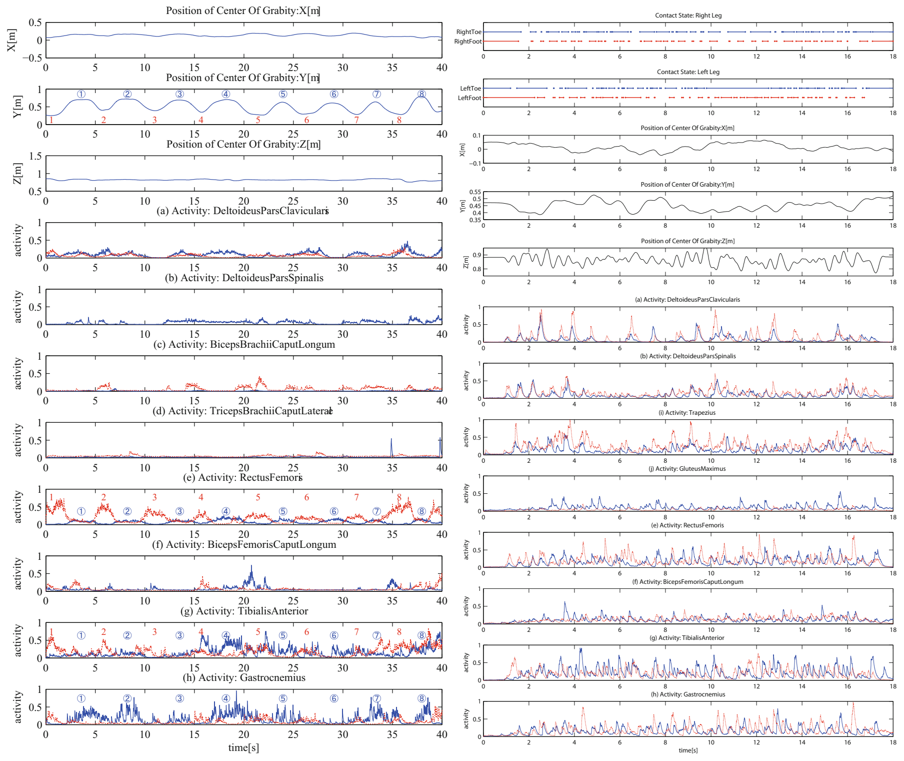


(b) Tap dance



(c) Drum play

Fig. 2. Snapshots of experts' motion



(a) TaiChi motion (b) Tap Dance motion

Fig. 3. (a) TaiChi motion: Position of the COG and activation of major muscles (Blue line shows right side of the body, red dot shows left side of the body (b) Tap Dance motion:Contact state (dots mean touch or leave events and line shows the "touch" state), position of COG and activation of major 16 muscles (Blue line shows right side of the body, red dot shows left side of the body)

4.2 Tap Dance

Tap dance is characterized by playing sound by contact with floor and special shoes which has metal on the toe and heel. The playing sound was produced by vertical motion. 20 seconds motion was analyzed. Figure 2(b) shows snapshots of the tap dancing with visualizing muscle activity. Figure 3(b) shows the contact state with foot and floor, the height of the COG, and 16 major muscle activities. In the contact state graph, each line represents touching state with foot and floor. Small different from Tai Chi motion, (i) trapezius muscle (j) gluteus maximus muscles are measured. The following considerations are described based on the analysis results.

– The COG has oscillatory motion to the vertical direction. The height of COG is lower than standing posture in average. Average value to the COG of the Z-axis direction was 0.883[m] during initial state in 0-0.8[s] and 0.861[m] (variance is 0.0011 [m ^{2}]) during 0.8-18[s] movement. This data implies that the subject does not try to hit the ground with jumping motion but also with his COG lower than initial state.

– Muscles of the whole body actively involved, in particularly upper limb muscles. It seemed to control the COG of Z-axis direction using inertial forces by swinging of the hands.
– Hip and knee joint does not move so hard from upright position in tap dancing. Activity of the biceps femoris and the gluteus maximus muscle are approximately 0.5 even in an instantaneous maximum value, it is not high compared to other muscle activity.
– Motion of the ankle joint has an important role to play for tap dance. This can be inferred by remarkable activity of Tibialis anterior and gastrocnemius.

4.3 Drum Playing

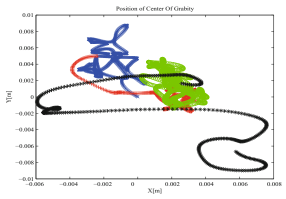
Based on the skilled drummer, the ideal motion during playing is stable condition of the trunk. There is no awareness that player is using muscles, and it is hard to feel the fatigue during long play. 20 seconds play was analyzed with three drums (snare drum, high tom and low tom). Contact force to snare head and timing of attack was calculated from the acceleration of the stick and the timing of the sound. Figure 2(c) shows the snapshots of the drum play. Figure 4(a) shows the score, height of the COG and major 16 muscle activities. As for the convenience, motion was separated into 4 phases.

– Phase 1: (12 - 17.5sec) Hi-Hat, snare drum, bass drum
– Phase 2: (17.5 - 21sec) Ride cymbal, snare drum, bass drum
– Phase 3: (21 - 27sec) Crush cymbal, ride cymbal, snare drum and bass drum
– Phase 4: (27 - 29sec) Snare drum, Hi-tom and Low-tom

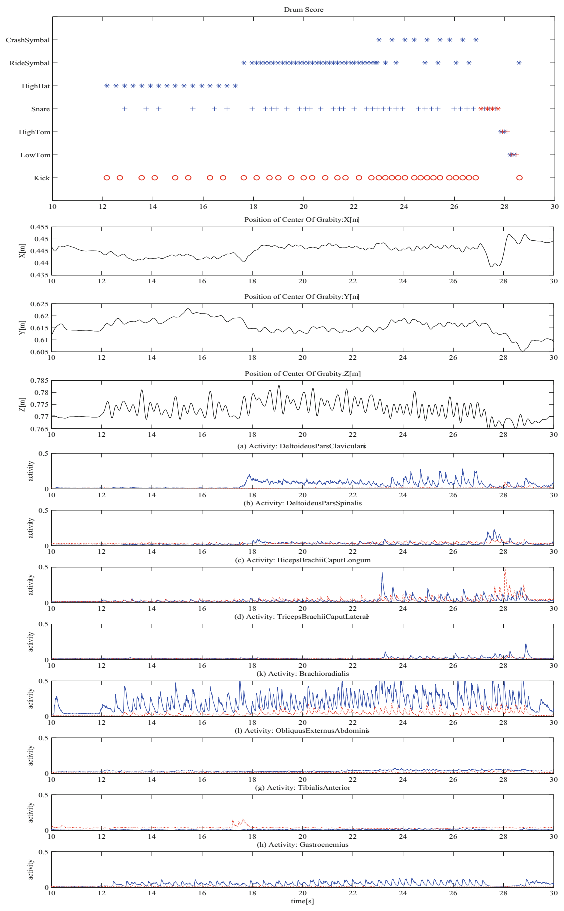
Crush cymbal, ride cymbal, and hi-hat is played by right hand, bass drum is by right foot, snare drum is by left hand basically, and hi and low tom was played by both hands. Measured muscle was partially changed to (k) brachioradialis (l) external oblique muscle. The following describes considerations based on the analysis results.

– The COG does not change much. 10-12 [s] average height of COG in the Z direction (initial static posture) is 0.4458 [m], average height in 12-30 [s] (during exercise) is 0.4453 [m] (variance is 7×10^{-6} [m ^{2}]).

  Figure 4(a) shows the change of the COG in XY plane (10-30[s]). The position of the COG is different from each phase respectively. The reason why COG swing large during 30 [seconds] was to try to mute the cymbal, not because of the play. Except that duration, COG moves 40mm (X-axis) and 15mm (Y-axis) at most, this amount is small.
– Muscle activity is dependent on the musical instruments. Radial arm muscle activity is particularly noticeable. Hi-hat: Right radial muscle, Bass drum: Right gastrocnemius, Ride cymbal: Right radial muscle and right anterior deltoid, Crush cymbal: Right radial muscle, anterior deltoid, and biceps,



(a) Position of COG in XY plane



(b) Tap dance

Fig. 4. (a) Position of COG in XY plane, Origin position was set at 12[s]. Blue x dots shows Phase(1) 12-17.5[s], red + dots shows Phase (2) 17.5-21[s], green o dots shows Phase (3) 21-27[s] and black * dots shows Phase (4) 27-29[s] (b) Drum playing:Contact state, position of COG and Activation of major 16 muscles. The motion is divided into 4 motion phase, Phase(1) 12-17.5[s], Phase (2) 17.5-21[s], Phase (3) 21-27[s] and Phase (4) 27-29[s].Score: * shows the timing of right hand contact, + shows left hand contact, o shows right foot contact. Muscle activation: Blue line shows right side of the body, Red dot shows left side of the body

Snare drum: Right and left radial muscle, right rear deltoid, and left biceps.
Tom-tom drums: Right and left radial muscle, left biceps, and right rear deltoid.
In particular, significant activity changes of the right deltoid muscle are observed in each phase. This is attributed to changes of the movement which is from the Z-axis direction to the X-axis direction during the changes from Phase 1 to Phase 2. In addition, triceps and biceps has worked remarkably well with respect from Phase 3 to 4. Because of the arm position that is close to body trunk at the phase 4, rear deltoid muscle works well.

– Z-axis direction movement of COG, it is correlated with the performance of the bass drum on the right foot and the snare drum in the left hand. Since the motion of the two above-mentioned behavior is striking in the vertical direction, and due to its effects. On the other hand, when playing cymbals in the right hand, which has influenced the motion of COG to the X and Y directions.

5 Conclusion

We discuss the computation of human motion dynamics and its analysis of the experts' motion skills. For the dynamic computation, to estimate whole body muscle tension, the following three types of optimization problem is solved in order. A) In inverse kinematics, generalized coordinates is obtained to reproduce the trajectory of the marker measurement while satisfying the wire constraints. B) In contact force estimation external forces to reproduce all external force calculated from the generalized coordinates and the known inertial model, and to reproduce the measured value of the total floor reaction force simultaneously. C) In inverse dynamics, the tensile strength of each wire to reproduce the joint torque calculated with the generalized coordinates, contact forces and known inertial model, and to reproduce the measured values from the EMG data. As examples of analysis, we measured and analyzed expertsf motion patterns, and obtained following considerations.

1. In Tai Chi dance performance, height of the COG is relatively low and movement is stable. Regardless of the orientation of the trunk during movement, the COG has periodicity of the Y-axis direction. Muscles of the lower limbs are working actively, and muscles in upper body works less compare to the muscles in lower limb.
2. For the tap dancing, the COG has oscillatory motion to the vertical direction. The height of COG is lower than standing posture in average. Muscles of the whole body actively involved, in particularly upper limb muscles. Hip and knee joint does not move so hard from upright position in tap dancing. Motion of the ankle joint has an important role to play for tap dance.
3. As for the drum playing, the COG does not change much. Muscle activity is dependent on the musical instruments. Z-axis direction movement of COG, it is correlated with the performance of the bass drum on the right foot and the snare drum in the left hand.

Acknowledgment. This research was supported by Japan Society for the Promotion of Science under Category (S) of Grant-in-Aid for Scientic Research (20220001), "Establishing Human-Machine Communication Through Kinesiology and Linguistics Integration" (PI: Y. Nakamura). Special thanks to the production staff in "Einstein's eye" in NHK BS premium. Drum playing was captured in a gymnasium in Tokyo Institute of Technology with the cooperation of Dr. Takeo Maruyama. The authors would like to thank Mr. Susumu Yo, Mr. HIDEBOH, and Mr. Akira Jimbo who kindly accepted to be the subject of these measurements. Special thanks to the laboratory members, Kanade Kubota, Akihiro Yoshimatsu, Shun-Chieh Hung, and Kensho Hirasawa who helped measurement and visualization.

References

1. Nakamura, Y., Yamane, K.: Dynamics computation of structure-varying kinematic chains and its application on human figures. IEEE Transaction on Robotics and Automation (2000)
2. Delp, S., Loan, J.: A computational framework for simulating and analyzing human and animal movement. IEEE Computing in Science and Engineering 2(5), 46–55 (2000)
3. Rasmussen, J., Damsgaard, M., Voigt, M.: Muscle recruitment by the min/max criterion - a comparative numerical study. Journal of Biomechanics 34(3), 409–415 (2001)
4. Forster, E., Simon, U., Augat, P., Claes, L.: Extension of a state-of-art optimization criterion to predict co-contraction. Journal of Biomechanics 37(4), 577–581 (2004)
5. Nakamura, Y., Yamane, K., Suzuki, I., Fujita, Y.: Somatosensory computation for man-machine interface from motion capture data and musculoskeletal human model. IEEE Transactions on Robotics (2005)
6. Nakamura, Y., Ikegami, Y., Yoshimatsu, A., Ayusawa, K., Imagawa, H., Oota, S.: Musculoskeletal morphing from human to mouse. In: IUTAM Symposium on Human Body Dynamics: From Multibody Systems to Biomechanics (2011)
7. Murai, A., Kurosaki, K., Yamane, K., Nakamura, Y.: Musculoskeletal-see-through mirror: Computational modeling and algorithm for whole-body muscle activity visualization in real time. Biophysics and Molecular Biology 3(103), 310–317 (2010)
8. Chadwick, E.K., Blana, D., van den Bogert, A.J., Kirsch, R.F.: A real-time, 3d musculoskeletal model for dynamic simulation of arm movements. IEEE Transactions on Biomedical Engineering 56(4), 941–948 (2009)
9. Ayusawa, K., Nakamura, Y.: Fast inverse kinematics algorithm for large dof system with decomposed computation of gradient and its application to musculoskeletal model. In: 17th Robotics Symposia (2B4) (2011) (in Japanese)
10. Ayusawa, K., Nakamura, Y.: Fast inverse dynamics algorithm with decomposed computation of gradient for wire-driven multi-body systems and its application to estimation of human muscle tensions. In: 2nd IFToMM International Symposium on Robotics and Mechatronics (11) (2011)
11. Ayusawa, K., Ikegami, Y., Nakamura, Y.: Simultaneous geometric parameters identification and inverse kinematics of time series motion by fast optimization using decomposed gradient computation. In: JSME Robotics and Mechatronics Conference 2012 (2P1-O10) (2012) (in Japanese)
12. Hill, A.: The heat of shortening and the dynamic constants of muscle. Proceedings of the Royal Society of London 126, 136–195 (1938)
13. Stroeve, S.: Impedance characteristic of a neuromusculoskeletal model of the human arm I. Posture control. Biological Cybernetics 81(5-6), 475–494 (1999)

Studying the Effect of Different Optimization Criteria on Humanoid Walking Motions

Kai Henning Koch[1], Katja Daniela Mombaur[1], and Philipp Souères[2]

[1] IWR - University of Heidelberg - Im Neuenheimer Feld 368
69120 Heidelberg - Germany
Henning.Koch@iwr.uni-heidelberg.de, kmombaur@uni-hd.de
[2] LAAS-CRNS - 7, av du Colonel Roche - 31077 Toulouse Cedex 04 - France
philippe.soueres@laas.fr

Abstract. The generation of stable, efficient and versatile walking motions for humanoid robots is still an open field of research. Several approaches have been implemented on humanoids in the past years, but so far none has led to a walking performance that is anywhere close to humans. This may be caused by limitations of the robotic hardware, but we claim that it is also due to the methods chosen for motion generation which do not fully exploit the capabilities of the hardware. Often, several characteristics of the gait, such as foot placement or step time, are fixed in advance in a suboptimal way for the robot. In this paper we discuss the potential of our optimal control techniques based on dynamical models of the humanoid robot for the generation of improved walking motions. We apply the method to a 3D dynamic model of the humanoid robot HRP-2 with 36 DOF and 30 actuators. Robot specific stability constraints (such as ZMP constraints) can be taken into account in the optimization. We present results for five different objective functions, and evaluate the influence of free foot placement and a relaxation of ZMP constraints.

Keywords: optimal control, humanoid robot, HRP-2, simulation, walking motion.

1 Introduction

Humanoid robots are highly redundant and underactuated multibody systems with many degrees of freedom. Generating walking motions for them which are at the same time efficient, stable and versatile, is a challenging task, and the motion capabilities of today's humanoids are still far behind those of humans. We claim that this problem is not only pertaining to the present robotic hardware, but that more effort should be put into choosing and developing appropriate software and control methods that can exploit all motion capabilities of the given hardware. In this paper, we explore the use of optimal control methods for the generation of walking motions for the humanoid robot HRP-2 [11]. The use of optimization approaches can be justified in two different ways:

– Optimization is used to mimic biology: It is a common assumption that movements of humans and animals are optimal due to evolution, individual development and training [15]. Optimization criteria depend on the particular situation. With optimal control techniques we can generate optimal motions for robot models, optimizing important gait characteristics such as stability, efficiency, effort or speed.

– Optimization is helpful for technical reasons: it serves on the one hand to find walking solutions that are feasible (among an infinite number of infeasible solutions), and on the other hand to select from the remaining motion abundance (i.e. the still large number of feasible gaits).

The efficient optimal control approach that we are presenting in this paper allows to determine position and velocity trajectories as well as actuator inputs simultaneously in an optimal way, and does not require to prescribe any of these quantities a priori. We compare the effect of five different objective functions (minimization of torque and of joint velocities, and a maximization of walking speed, of postural stability and of efficiency). In addition, we evaluate the effect of different constraints on the motion. First optimization results for HRP-2 have been presented in [12], but the present paper provides more detailed results for all objective functions and constraints as well as a more extensive discussion of the applicability of the different numerical results to the robot. We would like to emphasize that this paper treats the problem of offline motion generation and not the online control of walking. The fact that the computations described in this paper can not be performed in real time is therefore not an issue. Online control techniques such as real time optimization or NMPC (nonlinear model predictive control) methods later have to be applied to implement the computed trajectories on the real humanoid robot.

In many humanoid robots, walking is initially planned constraining the ZMP (zero moment point) [25] to lie within a desired region. To generate feasible reference walking trajectories many authors considered the linear inverse pendulum model [10]. The mass of the pendulum is usually set at the center of mass (CoM) of the robot and restricted to move horizontally. The method was extended in [8] to generate 3D walking. Model-based reference trajectory generation is the key point in the control of many humanoid robots such as ASIMO [23], WABIAN [6], or HRP-2. A well established approach is the technique of the pattern generators [19,24]. These methods may usually perform in real-time and are convenient to parametrize but do not choose gait characteristics in an optimal way. An alternative approach is the stack of task [14,20] to compute cascaded quadratic programs to minimize slacks with respect to a growing pile of constraint sets to represent descending priority-layers. Optimization approaches have also been used to generate walking motions. Investigations of cyclic walking motions for planar bipeds based on forward and inverse dynamic models have been published by [21,5,3,1] based on direct (e.g. collocation) and indirect optimization (Pontryagin Maximum Principle) methods with minimum energy consumption criteria. Based on stability optimization Mombaur et al. ([17,18,16]) published open-loop stable walking and somersault motions of bipedal walking mechanisms. In [22] optimization has been used to generate realistic running motions of 2D and 3D anthropomorphic models.

The remainder of this paper is organized as follows: In Section 2, we briefly present the dynamic models of walking motions of the humanoid robot HRP-2. In Section 3, we describe the formulation and solution of optimal control problems to generate walking motions. Section 4 presents extensive optimization results and compares different optimization criteria. We end the paper with a short summary and and outlook on the extensions of the presented research.

2 Mathematical Models of Walking Motions of the Humanoid Robot HRP-2

This section describes the 3D mathematical model of the humanoid robot HRP-2 in a form that is suitable for the use in optimal control problems. HRP-2 has 36 DOF and 30 torque actuators. We use the following assumptions for our robot model: the robot has rigid links and transmission units, the transmission ratios are sufficiently high that dynamic coupling effects of the motor inertias to the whole body structure are negligible and joint friction is not considered. The robot has flat, rubber coated feet and and an elastic 3 DOF ankle joint, but both elasticities are neglected for the present computations. HRP-2 is equipped with a stabilizer ([9]) that serves to prevent the robot from falling by compensating small modeling errors and small external perturbations. The stabilizer aims at keeping the ZMP in a stability region that is smaller than the actual support polygon. The base reference frame is fixed to the pelvis. As model coordinates we use the six coordinates of this base frame as well as the 30 internal joint angles, which would be minimal coordinates for a free-floating robot. In single and double support, the robot looses DOF, but we keep the same set of coordinates during all phases and describe their redundancy by additional algebraic constraints.

Walking motions are described as a series of alternating single and double support phases. In this study, we are only interested in symmetric and periodic gaits, and therefore we can reduce the mathematical problem formulation to one step of the gait and a subsequent mirroring of sides after which periodicity constraints are applied (see [22] for more details). The equations of motion of this multibody system result in nonlinear systems of differential algebraic equations for redundant coordinates:

$$\dot{q} = v \tag{1}$$

$$\dot{v} = a \tag{2}$$

$$\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} a \\ \lambda \end{pmatrix} = \begin{pmatrix} -N + F \\ \gamma \end{pmatrix}, \tag{3}$$

also satisfying the constraints on position and velocity level $g(q) = 0$ and $\frac{\mathrm{d}g(q(t))}{\mathrm{d}t} = G\dot{q} = 0$.

In these equations, M is the symmetric positive definite mass matrix and N is the term of nonlinear effects (combining Coriolis, centrifugal and gyroscopic forces): Matrix M and vector N for the humanoid robot HRP-2 have been computed with the automatic model generator HuMAnS [26]. In equation (3), F denotes the sum of all external forces acting on the multibody system, such as gravity force, joint torques, etc. The constraint Jacobian and constraint Hessian are $G = (\partial g / \partial q)$ and $\gamma = -\left(\left(\partial G / \partial q\right) \dot{q}\right) \dot{q}$, and λ the vector of Lagrange multipliers.

Switches from one motion phase to the next do not take place at given times but depend on the position variables which can be expressed by so-called switching functions

$$s(q, v, p) = 0. \tag{4}$$

Touchdown takes place, when a foot reaches zero height, and lift-off occurs, when the vertical contact force becomes zero. The contact forces are equivalent to the negative Lagrange multipliers in eqn. (3). There can be a discontinuity of velocities at touchdown of the swing foot to he ground. The velocity after impact v_+ is then computed using the same matrices M and G as above, and the velocity before impact v_-:

$$\begin{pmatrix} M & G^T \\ G & 0 \end{pmatrix} \begin{pmatrix} v_+ \\ \Lambda \end{pmatrix} = \begin{pmatrix} M v_- \\ 0 \end{pmatrix}. \tag{5}$$

It can be generally assumed that lift-off of the foot is smooth, i.e. there are no discontinuities. Models of the type described above - combining continuous motion phases as well as discrete "jump phases" - are often called hybrid dynamical systems.

3 Formulation and Solution of Optimal Control Problems for the Generation of Robot Walking Motions

The problem if generating optimal walking motions for HRP-2 can be formulated as a multiphase optimal control problem of the following form:

$$\min_{x(\cdot), u(\cdot) p, \bar{t}_i \in \mathcal{M}} \sum_{i=1}^{r} \int_{\bar{t}_{i-1}}^{\bar{t}_i} \Phi_i \left(x(t), u(t), p\right) dt + \Psi_i \left(\bar{t}_i, x\left(\bar{t}_i\right), p\right) \tag{6}$$

$$\text{subject to} \quad \dot{x}(t) - f_i\left(t, x(t), u(t), p\right) = 0 \quad i \in \mathcal{M} \tag{7}$$

$$x\left(\bar{t}_i^+\right) - h_i\left(x\left(\bar{t}_i^-\right)\right) = 0 \quad i \in \mathcal{M} \tag{8}$$

$$r_{eq}\left(x\left(\hat{t}_0\right), ..., x\left(\hat{t}_s\right), p\right) = 0 \tag{9}$$

$$r_{ineq}\left(x\left(\hat{t}_0\right), ..., x\left(\hat{t}_s\right), p\right) \geq 0 \tag{10}$$

$$g_i\left(x(t), u(t), p\right) \geq 0 \quad i \in \mathcal{M} \tag{11}$$

In these equations, x denote the state variables of the system, u the control variables - in this case the joint torques, p the model parameters, t the physical time. $\mathcal{M} = \{1, ..., r\}$ denotes the phase indices, with $r = 2$ for one step of a walking motion. $\bar{t}_{i-1}$ and $\bar{t}_i$ are the start and end times of phase i, respectively.

Without loss of generality it is assumed that $\bar{t}_0 = 0$ holds. With $x(\bar{t}_i^-)$ and $x(\bar{t}_i^+)$ we express a state x evaluated at time $\bar{t}_i$ just before or after a discontinuity.

The system dynamics during each phase is described by the corresponding set of DAEs (1) - (3) . The optimal control problem only "sees" the differential part of the variables and the equations which are considered as constraint of the optimal control problem , and the algebraic part is solved implicitly to determine a in the right hand side of (2). The objective function (6) can consist of continuous integral Lagrange-type functions Φ_i as well as end-value dependent Mayer-type functions Ψ_i. The purpose of this paper is to compare the effect of different objectives on the gait which will be further detailed below in section 4.1. Eqn. (8) describes state discontinuities between continuous motion phases: these include potential velocity discontinuities at impact, but also the discontinuity coming from the shift of sides after the end of the step. Eqn. (9) summarizes all pointwise coupled and decoupled equality constraints of the problem such as periodicity constraints, phase switching conditions, and the invariants resulting from index reduction of the DAE. (10) and (11) are pointwise as well as continuous inequality constraints, e.g. bounds on all optimization variables or foot clearance constraints or ZMP stability constraints, see below.

The multi-phase optimal control problem is solved with the powerful optimal control software MUSCOD II developed in Heidelberg. It is based on early works of [2] and was implemented by [13]. The approach applies the direct multiple shooting method to transform the infinite dimensional optimal control problem to a finite dimensional optimization problem. Controls are discretized by means of functions with local support (all computations in this paper are based on piecewise linear support functions). Multiple shooting is then used to parameterize the state variables of the system. Multiple shooting essentially transforms a boundary value problem into a set of initial value problems with continuity conditions. For structural reasons, multiple shooting and control grid are chosen identically. This finally results in a large but highly structured nonlinear programming problem that may efficiently be solved by a tailored sequential quadratic programming method. The integration of the system dynamics and the computation of the derivatives of the trajectories is performed by powerful integrators with sensitivity generation capabilities. As all optimization methods, also the direct multiple shooting approach needs initial values for all optimization variables (in this case for discretized controls, phase times and state variables at the multiple shooting nodes). The higher the quality of these values the better is generally the convergence of the algorithm. For the computations presented in this paper we have taken initial values from physically feasible and stable gaits from a preview control pattern generator (see [7]) since this motion was available to us. Note however that this motion is quite different from the motions that are finally produced by the optimization (see the next section). It is not necessary that the initial values are all feasible; also simpler ways of initialization (like an interpolation of state variables between initial and final point) might in principle be used.

4 Optimization Results

4.1 Overview of Computations

In this section, we compare optimization results for walking for the following objective functions:

– a minimization of joint torques squared (which penalizes actuator torques which are the control inputs for the optimal control problem - no matter if they are used for dynamic or static purpose. This objective generally produces smooth controls with little oscillations):

$$\min \Phi_{\text{torques}} = \int_0^T \sum_{j=1}^{N} (\omega_j u_j)^2 \, dt \tag{12}$$

– a maximization of average forward velocity (since humanoid robots still move very slowly compared to humans, we want to investigate the potential limits of HRP-2):

$$\max \Psi_{\text{forw Vel.}} = \frac{l_{\text{Step}}}{T} \tag{13}$$

– a maximization of postural stability (penalizing the deviation of the local center of pressure p_{COP_e} from a reference point under the sole of the foot p_{Centr_e}, see [3] for further discussion):

$$\min \Phi_{\text{post stab}} = \int_0^T \sum_{e=\{Lf,Rf\}} (p_{COP_e} - p_{Centr_e})^2 \, dt \tag{14}$$

– a maximization of efficiency of the gait which can also be expressed as a minimization of the cost of transport [4] or the mechanical power output over a step [3]:

$$\min \Phi_{\text{eff}} = \int_0^T \sum_{j=1}^{N} \frac{|\dot{q}_j u_j|}{l_{step}} dt \tag{15}$$

– a minimization of joint velocities (angular rates) squared which aims at reducing the angular motions as much as possible while still maintaining some form of gait):

$$\min \Phi_{\text{joint vel}} = \int_0^T \sum_{j=1}^{N} (\omega_j \dot{q}_j)^2 \, dt \tag{16}$$

For all five objective functions above, we have investigated the following variations of the constraints set:

– with and without constraint on the ZMP, restricting it to stay within a small circle below the center of the foot during single support and in a tube connecting the two foot centers during double support.
– for the ZMP constrained case: leaving the foot placement free, or constraining the step length and step width to the values of the initial walking solution

Overall, this results in 15 different objective function - constraint combinations.

4.2 Comparison of All Objectives and Constraint Combinations

Figure 1 shows bar plots of different gait characteristics for all optimization criteria and constraint combinations. The top left plot shows the optimal step length and step width for the different criteria. Trivially these quantities remain unchanged with respect to the initial value for the optimization runs with fixed foot position. However, they change - in some cases significantly - for the other five computations. Step width is in all cases chosen smaller than the original $0.144m$, for the maximization of efficiency it is even reduced to $0.016m$. Step length increases in three cases and is reduced in two. Compared to the original step length of $0.152m$, the longest step length occurred for a maximization of

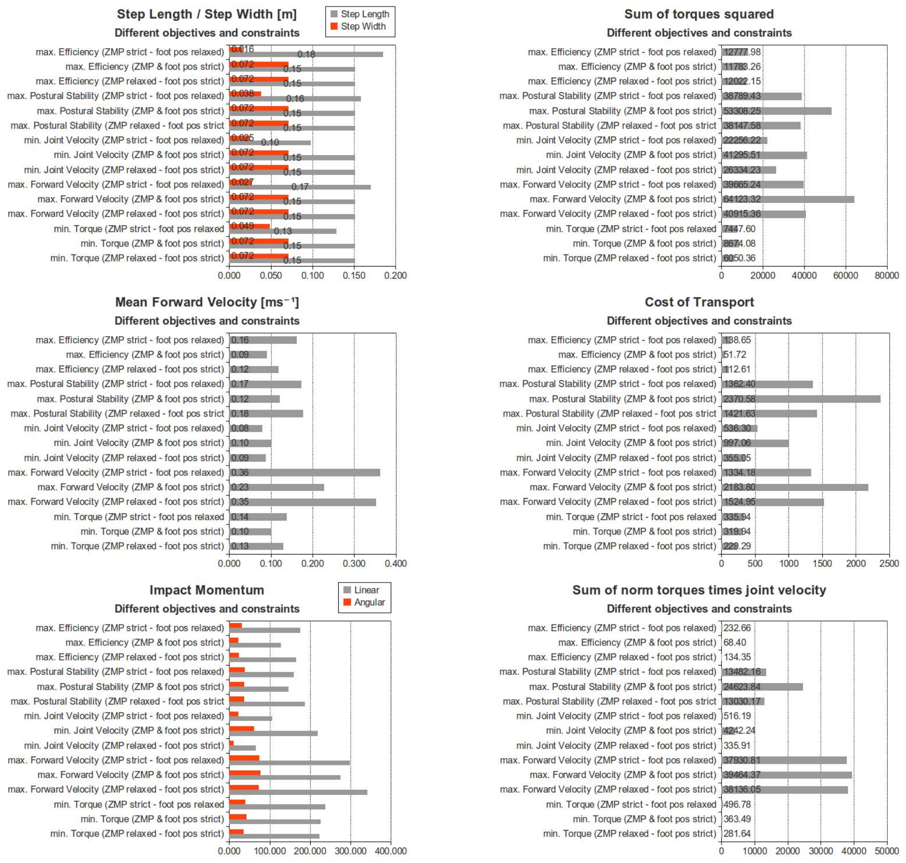


Fig. 1. Different gait characteristics compared for all optimization criteria and constraint combinations. Left column: step length/ width, average forward velocity and linear as well as angular impact momentum at touchdown. Right column shows different cost measures for these different solutions: sum of torques squared, cost of transport, and absolute mechanical energy (sum of torques along angles).

efficiency with $0.185m$ and the shortest for the minimization of joint velocities with $0.098m$. The middle left plot in figure 1 shows the average walking speed resulting from the different optimization criteria. The gait used for initialization had a walking speed of $0.178m/s$[1] As one could expect, the highest walking speeds result when precisely this quantity is maximized, i.e. when optimizing the second criterion listed in section 4.1. For this objective and free foot placement but constrained ZMP, the walking speed can be increased to $0.363m/s$, i.e. $1.31km/h$ which represents an increase by a factor of 2.03, even further than for an unconstrained ZMP but a fixed foot position ($0.352m/s$). With both constraints, the same criterion only leads to $0.228m/s$. For all objective functions, except for the minimum joint velocity criterion, the relaxation of any of the constraints leads to an increased average walking speed. The bottom left plot shows the linear and angular impact momentum for all 15 cases. A high impact momentum is undesirable since it results in a loss of energy and, in particular in the case of a humanoid robot, produces a high risk a destabilization. Both linear and angular impact momentum were particularly high for the maximum average velocity criterion, no matter if ZMP or foot position are constrained or relaxed. This makes the maximum velocity solution less interesting than it seemed above, and this criterion might only be useful for the real robot if additional constraints on the size of the impact are taken into account. The smallest impact momenta occur for the minimum joint velocity criterion and a relaxed ZMP, followed by the same criterion with relaxed foot position. Minimum torque, maximum efficiency and maximum postural stability result in medium size impacts which however also might have to be reduced for an implementation on the real robot. The right column of figure 1 presents different measures for the cost of the different walking motions, namely the sum of torques squared (which corresponds to the electric power consumed by the motors), the cost of transport as defined above, as well as the absolute mechanical energy (sum of absolute values of torques integrated over joint angles). Even though they have quantitatively different results, all three measures show the same tendencies: maximum velocity and maximum postural stability lead to quite costly solutions while minimum joint torques and maximization of efficiency lead to rather cheap solutions in terms of all three measures.

4.3 Further Analysis of Optimization Results for Constrained ZMP and Free Foot Placement

In the following, we will discuss the optimization results for free foot placement and constrained ZMP in more detail, since this combination of constraints seems to be the most interesting for HRP-2. Figure 2 shows snapshot sequences of the optimal walking cycles for all five criteria.

Figure 3 shows the trajectories of position (top row) and orientation (bottom row) variables of the pelvis for all five objective functions (see color code explained in the first plot) over a walking cycle of two steps. In all plots the

[1] For comparison purposes: the regular walking speed of humans is about $1.3-1.4m/s$.

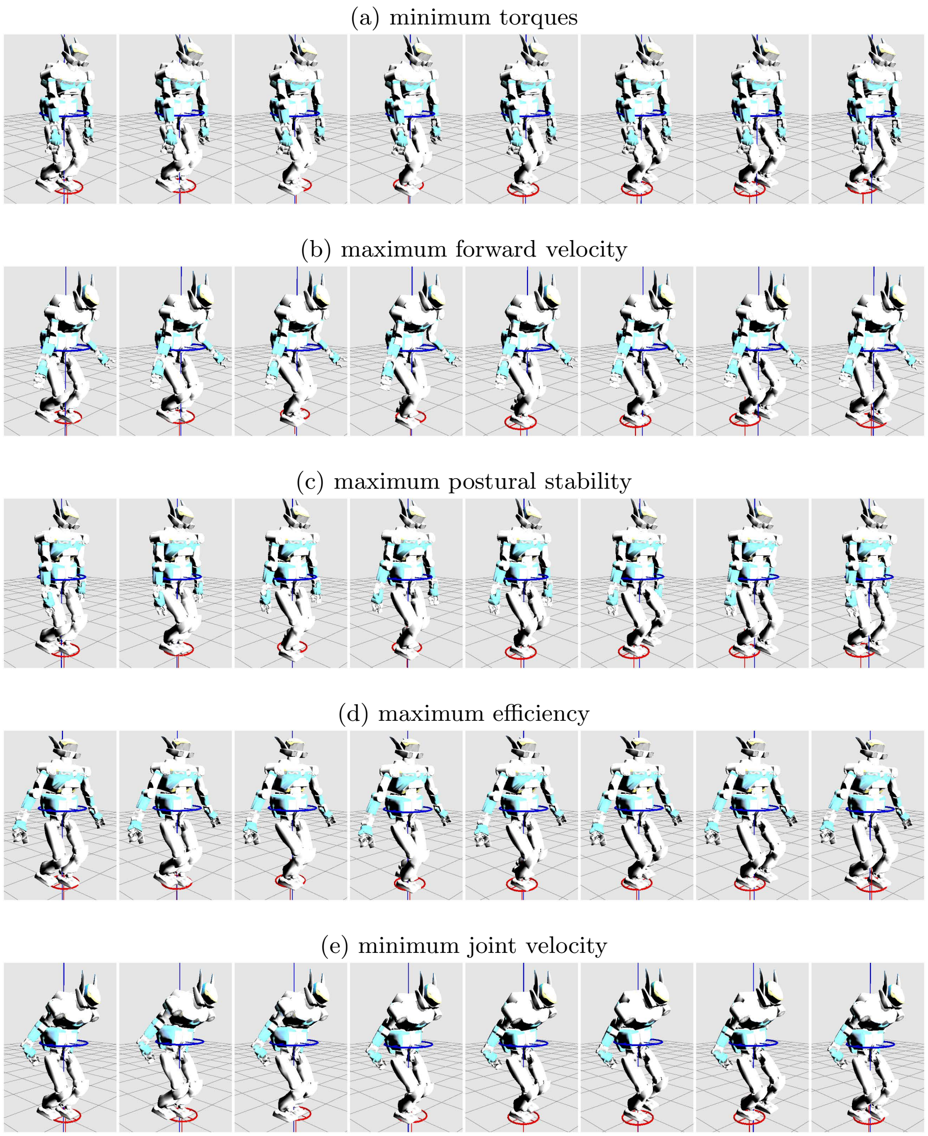

(a) minimum torques
(b) maximum forward velocity
(c) maximum postural stability
(d) maximum efficiency
(e) minimum joint velocity

Fig. 2. Walking sequences for HRP-2 with free foot placement and constrained ZMP for different objective functions (center of circle on the floor represents ZMP, center of circle near pelvis shows CoM)

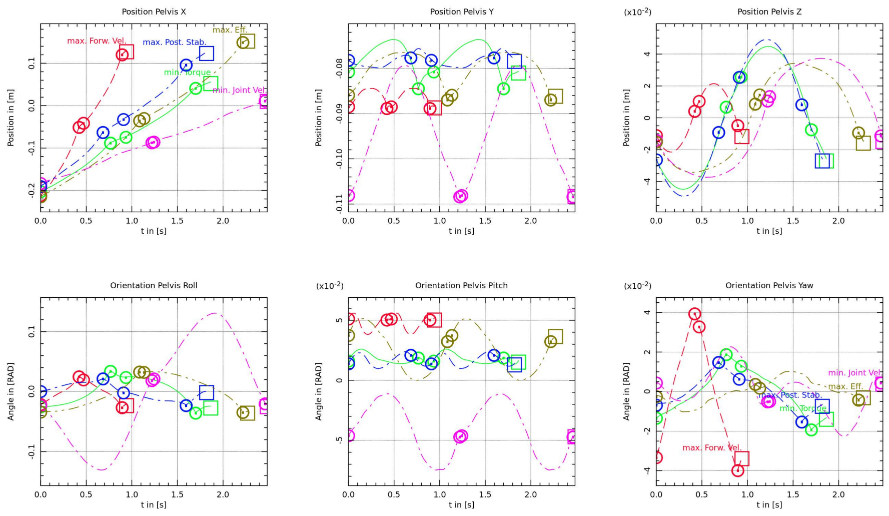


Fig. 3. Pelvis trajectories over a full gait cycle (two steps) for the five different objective functions with constrained ZMP and free foot placement. Top: pelvis position trajectories in forward, vertical and sideward direction. Bottom: pelvis roll, pitch and yaw angles. The gait cycle starts with the single support on the left leg, followed by double support, single support with the right leg and then double support phase (circles denote ends of phase, squares the end of the cycle).

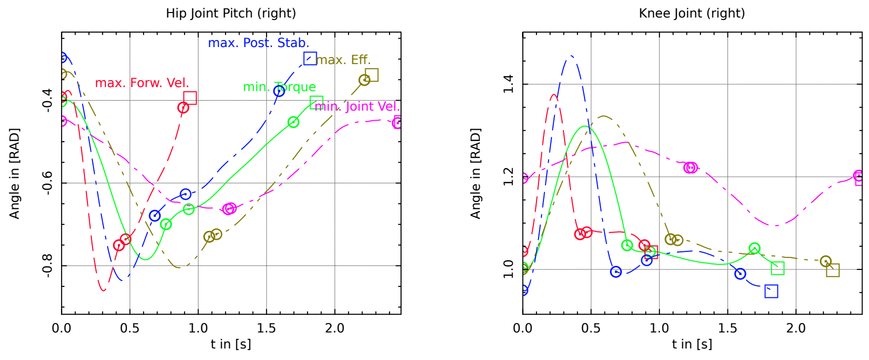


Fig. 4. Sagittal plane hip and knee angle trajectories of one leg over a full gait cycle (two steps) for the five different objective functions with constrained ZMP and free foot placement. The gait cycle for this leg starts with the swing phase, followed by double support, single support and then double support phase (circles denote ends of phase, squares the end of the cycle).

different step durations resulting from the different objectives become obvious. The top left plot describes the forward motion of the pelvis which differs significantly for the different objectives (also compare discussion about step lengths and average velocities above). The second plot in the top row shows the vertical motion of the pelvis, clearly depicting in all cases the two vertical oscillations over the two steps. The height variations of the pelvis are quite small (smaller than 1 cm) for all objective functions except for the minimum joint velocity criterion, where the variation is around 3 cm despite the small steps. This is due to the very small range of motion of the hip and the knee angles caused by this criterion which induces a high stiffness in the joints. The third plot shows the sideward motion of the pelvis. Note that this time there is only one oscillation since the periodic cycle for orthogonal gait oscillations is two steps and not one, as for the vertical motion. The variations in sideward directions lie between 4 cm - for the maximum speed criterion - and 10 cm - for the maximum postural stability criterion. The lower row in figure 3 presents the roll, pitch and yaw angles of the pelvis. Especially noteworthy is the high amplitude of the roll angle for the minimum joint velocity criterion which again is caused by the fact that the criterion stiffens the legs. This criterion also leads to a significantly reduced pitch angle, i.e. the pelvis is turned backwards. In all other cases, the pelvis is bent slightly forward. For the yaw angle, in particular the maximum speed velocity stands out with much larger amplitudes than the other criteria, which is caused by the large steps performed in this mode of motion.

Figure 4 shows the trajectories of the hip and knee angles in the sagittal plane for all five objective functions. The plot shows a whole walking cycle for one leg starting with the swing phase, followed by double support, then single support and again double support phase. Both angles are bent much more than in human walking motions and lead to the characteristic half-sitting position of humanoid robots. As we have shown in [12], this position is caused by the ZMP constraint, and a relaxation of this constraint results in a straightening of the legs and an increase of the pelvis height. As mentioned above, the oscillations linked to the minimum joint velocity criterion are very small for both knee and hip angle such that the leg angles are nearly constant over the full cycle. The shapes of the hip angle trajectories are very similar for the other four criteria, the only difference is the total duration of the cycle which results in more or less stretched angle trajectories. The same is true for the knee angle with slightly more pronounced differences in the amplitudes: the maximum postural stability criterion leads to the largest knee angle amplitude and the maximum efficiency criterion leads to the smallest one (but still much bigger than the minimum joint velocity knee amplitude).

We also found it interesting to analyze the different motions of the swing foot that are induced by the different optimization criteria. Figure 5 shows the sole center position trajectories as well as roll, pitch and yaw angles of the swing foot over one step. In addition to the foot step locations, the foot motion during swing is prescribed by some pattern generators, but in the optimal control approach, the foot trajectories can be freely determined along with the whole body motion. Appropriate constraints in the optimal control problem formulation avoid any penetration

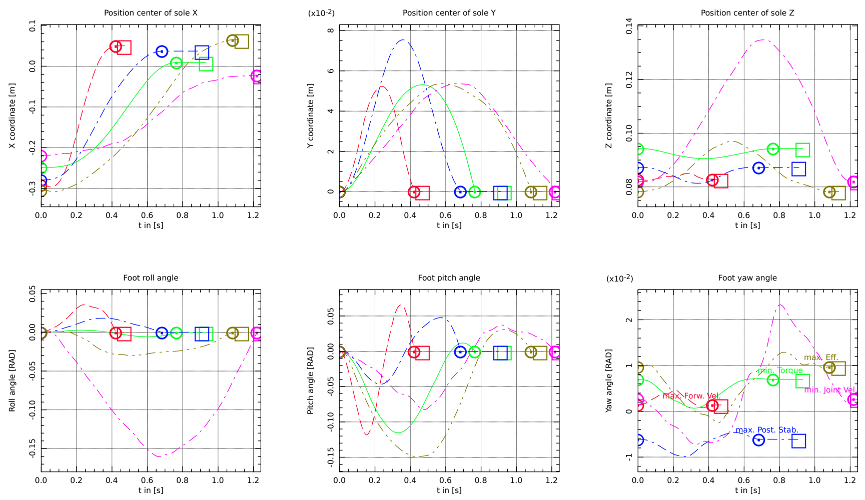


Fig. 5. Swing foot trajectories over one step (swing phase and double support phase) for the five different objective functions with constrained ZMP and free foot placement. Top: foot sole center position trajectories in forward, vertical and sideward direction. Bottom: foot roll, pitch and yaw angles (circles denote ends of phase, squares the end of the cycle. Same color code as in previous figures is used for the objective functions).

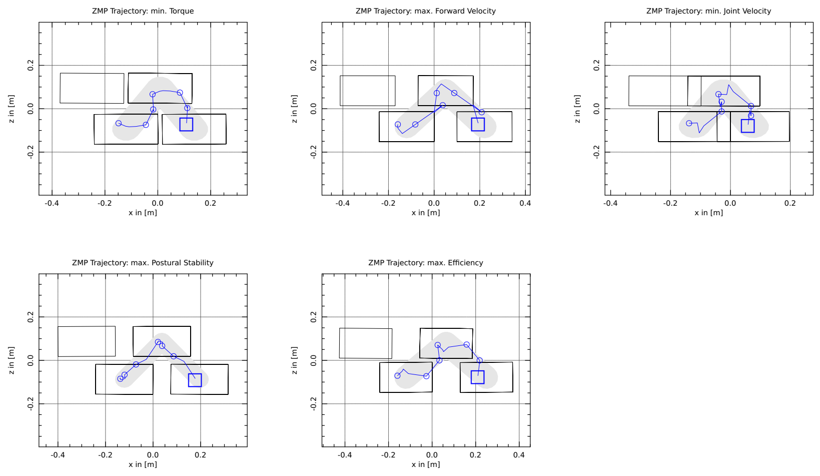


Fig. 6. ZMP paths over a full gait cycle (two steps) for the five different objective functions with constrained ZMP and free foot placement. Grey areas show the ZMP constraints.

of the foot into the ground and even prevent sliding contact before touchdown by guaranteeing a sufficient ground clearance. The maximum postural stability criterion leads to the swing foot trajectory with the highest lift (8cm), and the minimum joint velocity with the largest sideward variation of the foot (5cm). Changes of foot angles are generally not big over the swing phase with the highest roll and yaw angle variations again for the minimum joint velocity criterion, and the highest pitch angle variation for the maximum efficiency criterion.

Finally, we present the ZMP paths in the horizontal plane for all five criteria over a cycle of two steps. Grey tubes indicate the areas in which the ZMP is allowed to move for stable motions. In all five cases, suitable constraints force the ZMP to remain inside these areas during the optimal control problem solution. In four cases the ZMP moves to or along the boundaries of these stable areas. The constraints are quite expensive to satisfy, and the ZMP would move outside as soon as this constraint would be relaxed. The only exception is the postural stability criterion. This criterion punishes the ZMP moving away from the centers of these areas, and if this punishment is hard enough, it is not necessary to additionally formulate the corresponding constraint. The minimum torque criterion and the maximum postural stability criterion produce quite smooth ZMP paths while they are more "cracked" for the other criteria.

5 Conclusion and Perspectives

In this paper, we have presented solutions of optimal control problems for the generation of walking motions for the humanoid robot HRP-2. Five different objective functions have been evaluated as well as the effect of ZMP and foot placement constraints.

A free foot placement appears to be desirable in all cases investigated. Defining the foot placement a priori in some heuristic way reduces the gait variety considerably and decreases the optimization potential. The only reason to constrain foot positioning is an environment where only limited footholds are available such as walking on step stone bridges, but on even terrain with obstacles the foot positions should be chosen freely in an optimal way according to the chosen optimization criterion. Relaxing ZMP constraints has demonstrated some interesting perspectives - such as the possibility to walk in a more upright way than current humanoids do, but it is certainly not an option when generating motions for the real HRP-2 robot. It might become interesting again when applying this approach to a new robot generation or another humanoid model.

The minimization of joint velocities does not appear to be a useful criterion. Even though it may intuitively seem stabilizing to avoid unnecessary joint motions, the objective leads to very stiff, non-smooth and unnatural motions with high oscillations in the pelvis height and roll angle and the foot sidewards motion as well a backward inclination of the pelvis. We also do not consider the maximization of postural stability to be suitable for the generation of better humanoid walking motions. Postural stability in terms of the ZMP criterion is already considered in the constraints which could be made stricter if ever necessary. Maximizing postural stability results in extremely costly solutions - in particular compared to the slow walking velocities it exhibits, and we do not see the immediate advantage to be so far off the stability boundaries when shifting closer to the already very conservative boundaries would result in faster of more efficient motions.

The maximization of average walking velocity is certainly interesting if the limitations of a robot are to be evaluated. Coming closer to the role model of human walking also implies that humanoids have to become considerably faster that they are at the moment. The computed increase of walking speed by a factor of a little more than 2 with respect to the reference solution may not be reachable in reality since the associated impacts are too big for HRP-2. We therefore propose to add constraints reducing the impact to below the accepted threshold for all optimization runs maximizing walking velocity. This should then still lead to an increase of speed, but less significant than the one reported here.

The minimization of joint actuator torques and the maximization of walking efficiency also seem to be promising objective functions which are associated with low "energetic" costs (in all measures investigated). The minimization of torques leads to smoother motions while the maximization of efficiency leads to faster motions with higher speeds. These motions are characterized by smaller impacts than the maximum velocity solution, but it might still be too much for the real robot and appropriate constraints should be added.

Humans typically apply not a single optimization criterion but weighted combinations of several - in many cases contradicting - criteria which we would also recommend for robots. Interesting combinations that we will investigate in the near future are minimization of joint torques & maximization of efficiency, minimization of joint torques & minimization of time to target (i.e. maximization of walking speed), maximization of velocity & minimization of impacts, minimization of joint torques & minimization of impact for a given velocity, all combined with ZMP and impact constraints and with free foot placement.

We also would like to mention that one obvious difficulty of computed periodic motion is to reproduce the desired starting values of the periodic cycle for all position and velocity variables on the real robot. In order to tackle this difficulty we have developed a procedure that can generate - for every optimal periodic cycle that is of interest for the robot - a starting step (or more precisely 1.5 starting steps) that bring the robot from its regular half-sitting rest position onto the periodic cycle. In the same way, stopping motions are computed that take the robot out of the periodic cycle and bring it to its rest position.

Financial support by the Heidelberg Graduate School of Mathematical and Computational Methods for the Sciences and by the European FP7 project ECHORD (GOP) is gratefully acknowledged.

References

1. Bessonnet, G., Chessé, S., Sardain, P.: Optimal gait synthesis of a seven-link planar biped. The International Journal of Robotics Research 23, 1059–1073 (2004)
2. Bock, H., Plitt, K.: A multiple shooting algorithm for direct solution of optimal control problems. In: Proceedings of the 9th IFAC World Congress, Budapest, pp. 243–247. Pergamon Press (1984)

3. Buss, M., Hardt, M., Kiener, J., Sobotka, M., Stelzer, M., von Stryk, O., Wollherr, D.: Towards an autonomous, humanoid, and dynamically walking robot: Modeling, optimal trajectory plannung, hardware architecture and experiments. In: Proceedings of the 3rd International Conference on Humanoid Robots (2003)
4. Garcia, M., Chatterjee, A., Ruina, A.: Efficiency, speed, and scaling of 2d passive dynamic walking. Dynamics and Stability of Systems (1998)
5. Hardt, M., Kreutz-Delgado, K., Helton, J.W.: Optimal biped walking with a complete dynamical model. In: Proceedings of the 38th Conference on Decision & Control (1999)
6. Yamaguchi, J., Soga, E., Inoue, S., Takanishi, A.: Developement of a bipedal humanoid robot - control method of whole body cooperative dynamic biped walking. In: Proceedings of the 1999 IEEE International Conference on Robotics & Automation (1999)
7. Kajita, S., Kanehiro, F., Kaneko, K., Fujiwara, K., Harada, K., Yokoi, K., Hirukawa, H.: Biped walking pattern generation by using preview control of zero-moment point. In: Proceedings of the 2003 IEEE International Conference on Robotics & Automation (2003)
8. Kajita, S., Kanehiro, F., Kaneko, K., Yokoi, K., Hirukawa, H.: The 3D linear inverted pendulum mode: a simple modeling for a biped walking pattern generation. In: Proceedings of the 2001 IEEE/RSJ International Conference on Intelligent Robots and Systems (2001)
9. Kajita, S., Nagasaki, T., Kaneko, K., Hirukawa, H.: ZMP-based biped running control. IEEE Robotics and Automation Magazine 07, 63–73 (2007)
10. Kajita, S., Tani, K.: Study of dynamic biped locomotion on rugged terrain-theory and basic experiment. In: 'Robots in Unstructured Environments', 1991 ICAR, Fifth International Conference on Advanced Robotics, vol. 1, pp. 741–746 (1991)
11. Kaneko, K., Kanehiro, F., Kajita, S., Hirukawaa, H., Kawasaki, T., Hirata, M., Akachi, K., Isozumi, T.: Humanoid robot HRP-2. In: Proceedings of the IEEE International Conference on Robotics & Automation (2004)
12. Koch, K.H., Mombaur, K., Soueres, P.: Optimization-based walking generation for humanoid robot. Accepted for SYROCO, Dubrovnic, Croatia (2012)
13. Leineweber, D., Bauer, I., Bock, H., Schlöder, J.: An efficient multiple shooting based reduced SQP strategy for large-scale dynamic process optimization - Part I: theoretical aspects, pp. 157–166 (2003)
14. Saab, L., Ramos, O., Mansard, N., Souères, P., Fourquet, J.Y.: Generic dynamic motion generation with multiple unilateral constraints. In: IEEE International Conference on Intelligent Robots and Systems (2011)
15. McNeill, A.R.: Principles of Animal Locomotion. Princeton University Press (2006)
16. Mombaur, K.: Using optimization to create self-stable human-like running. Robotica 27, 321–330 (2009); published online June 2008
17. Mombaur, K.D., Bock, H.G., Schlöder, J.P., Longman, R.W.: Human-like actuated walking that is asymptotically stable without feedback. In: Proceedings of IEEE International Conference on Robotics and Automation, Seoul, Korea, pp. 4128–4133 (May 2001)
18. Mombaur, K.D., Bock, H.G., Schlöder, J.P., Longman, R.W.: Self-stabilizing somersaults. IEEE Transactions on Robotics 21(6) (December 2005)
19. Morisawa, M., Kajita, S., Kaneko, K., Harada, K., Kanehiro, F., Fujiwara, K., Hirukawa, H.: Pattern generation of biped walking constrained on parametric surface. In: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, Barcelona, Spain (2005)

20. Ramos, O., Saab, L., Hak, S., Mansard, N.: Dynamic motion capture and edition using a stack of tasks. In: IEEE Humanoids Proceedings, Bled, Slovenia (October 2011)
21. Roussel, L., de Wit, C.C., Goswami, A.: Generation of energy optimal complete gait cycles for biped robots. In: Proceedings IEEE International Conference on Robotics and Automation (1998)
22. Schultz, G., Mombaur, K.D.: Modeling and optimal control of human-like running. IEEE/ASME Transactions on Mechatronics 15, 783–792 (2010)
23. Takenaka, T.: The control system for the honda humanoid robot. Age and Ageing 35-S2, 24–26 (2006)
24. Takenaka, T., Matsumoto, T., Yoshiike, T.: Real time motion generation and control for biped robot - 1st report: Walking gait pattern generation. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems (2009)
25. Vukobratovic, M., Stephanenko, J.: On the stability of anthromomorphic systems. Mathematical Biosciences 15, 1–37 (1972)
26. Wieber, P.B., Billet, F., Boissieux, L., Pissard-Gibollet, R.: The HuMAnS toolbox, a homogenous framework for motion capture, analysis and simulation. In: Internal Symposium on the 3D Analysis of Human Movement (2006)

Modeling and Simulating Compliant Movements in a Musculoskeletal Bipedal Robot

Roberto Bortoletto[1], Massimo Sartori[2], Fuben He[3], and Enrico Pagello[1]

[1] Intelligent Autonomous Systems Laboratory
Department of Information Engineering (DEI)
University of Padua, Italy
{bortolet,epv}@dei.unipd.it

[2] Department of Neurorehabilitation Engineering
Bernstein Focus Neurotechnology Goettingen
Georg-August University
Von-Siebold-Str. 4, 37075 Goettingen, Germany
massimo.sartori@bccn.uni-goettingen.de

[3] School of Mechanical Engineering
Dalian University of Technology, Dalian, China
hefuben@mail.dlut.edu.cn

Abstract. This paper describes the modeling and the simulation of a novel Elastic Bipedal Robot based on Human Musculoskeletal modeling. The geometrical organization of the robot artificial muscles is based on the organization of human muscles. In this paper we study how the robot active and passive elastic actuation structures develop force during selected motor tasks, and how we can model the contact between feet and ground. We then compare the robot dynamics to that of the human during the same motor tasks. The motivation behind this study is to reduce the development time by using a simulation environment for the purpose of developing a bipedal robot that takes advantage of the mechanisms underlying the human musculoskeletal dynamics for the generation of natural movement.

Keywords: Flexible Robotic, Musculoskeletal Model, OpenSim Simulation, Compliant Movement, Bipedal Robot.

1 Introduction

Human body representations have been used for centuries to help in understanding and documenting the shape and function of its compounding parts [1]. The synthesis of human motion is a complex procedure that involves accurate reconstruction of movement sequences, modeling of musculoskeletal kinematics, dynamics and actuation, and characterization of reliable performance criteria. Many of these processes have much in common with the problems found in robotics research, with the advent of complex humanoid systems. In the past years studies about elastic bipedal robot (i.e. robot actuated by elastic actuators) produced many humanoid robot models based on compliant legs that utilize mono-articular and bi-articular arrangement of tension springs. The needs to develop robot devices that can move on uneven and rough terrain, but also the understanding of human and animal locomotion mechanics [2] have encouraged more studies on this field of robotics. Several experiments based on the use of simulation platform and also of real robots showed that these models can provide features that could not be explained by the previously developed simpler models such as the generally known as passive dynamic walking. In [3] it was shown that the compliant elements in the proposed robot leg structure made it possible to generate both walking and running gaits. In [4] a motion simulation model in the two-dimensional sagittal plane was based on the extended series actuation principle demonstrating a reduction in the energy requirements. In [5] a biologically inspired robot was presented showing to be capable of both energy-efficient and human-like walking and jogging gaits. Understanding the basis of human movement and reproducing it in robotic environments is a compelling challenge that has engaged a multidisciplinary audience. The understanding of how muscles are activated to actuate the human body will directly allow designing motion and balance controller to move humanoids in a more sophisticated way. To achieve this goal, a theoretical framework is needed, and in the last decade several researchers have turned their attention to the realization of simulation platforms that give a such framework.

This paper describes the design principles and the simulation of a three- dimensional novel bipedal robot based on the organization of the human musculoskeletal system. The description of the real robot prototype platform, illustrated in Fig. 8, that has been built at Dalian University of Technology, will be described in detail in a future paper. We want to make a robot that has passive structures which allow to reproduce the dynamics of force that characterize the passive structures of human muscle. This is because the human muscles are efficient structures. For this purpose we will compare the forces produced by the springs with the forces generated by passive components in human muscle. Since it is not possible to measure forces directly from human muscle, a simulation approach is needed. To evaluate the design concepts of this novel bipedal robot, based on musculoskeletal principles, we have made several simulations with the aim of evaluating its behaviors, by comparing the results obtained from our robot with those obtained in the simulation of a human being engaged in doing the same movements. Both robot and human were simulated using the open-source musculoskeletal simulation software OpenSim[1] [6]. The dynamics of the robot torque actuators and artificial muscles, represented by springs, was then compared to the muscle and joint dynamics of the human body. The aim of this work is a study with the final goal of providing a novel methodology to replicate the mechanisms underlying the human musculoskeletal dynamics into artificial anthropomorphic systems. This was motivated by the recent achievements obtained in the state of the art bipedal robots in which biologically inspired designed led to obtain better compliant movements. In Section 2, a brief description of musculoskeletal human model used and the structure of our robot is provided with

[1] Freely available from: `https://simtk.org/home/opensim`

a detailed description of methods and tools adopted during the modeling and simulation phases. In Section 3, the evaluation approach to validate our work is described. Section 4 provides the results obtained with the comparison between human and humanoid. Section 5 concludes the paper with discussion and plan for future work.

2 Methods

The proposed biologically inspired four-segment robot [7, 8] is characterized by a mass of about 2Kg and an height of about 40cm. It is a small humanoid robot, according to the dimensions suggested for participating into the kidsize class of the RoboCup Humanoid League. Every joint into the model connects a parent body to a child body. A joint defines the kinematic relationship between two frames each affixed to a rigid-body parameterized by joint coordinates. Every body has a moving reference frame in which its Center-Of-Mass (COM) and inertia are defined. Its elastic actuation structure is based on the musculoskeletal human model. In fact, we started our work from a computer model of the human musculoskeletal system that represents the kinematics of joints [9], the geometry of bones, the three-dimensional organization and force-generating properties of lower limb muscles [10–12]. Each leg in the model had seven Degrees Of Freedoms (DOF) including: hip internal-external rotation (HRO), hip adduction-abduction (HAA), hip flexion-extension (HFE), knee flexion-extension (KFE), ankle subtalar flexion-extension (ASA), and ankle plantar-dorsi flexion (AFE). Line segments approximated the muscle-tendon path from the origin to insertion of the 86 muscles of the lower limb included into the model.

On the other hand, the architecture of the actuator used into the robot has been inspired by the elastic characteristics of springs and by the antagonism in the muscle-tendon structure. Thanks to its suitable properties, this type of

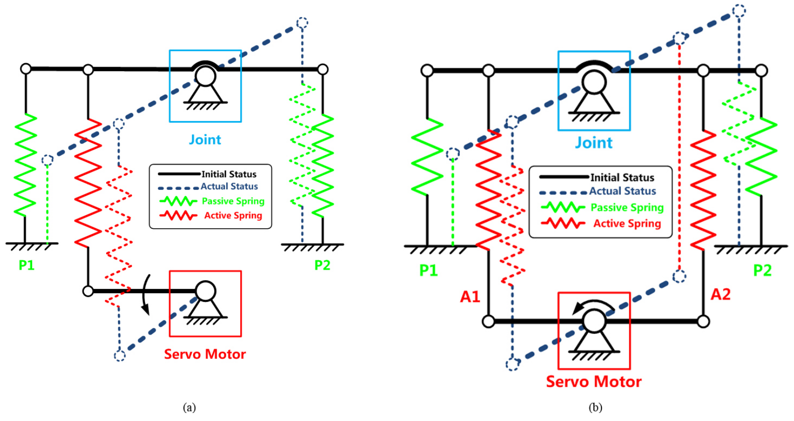


Fig. 1. (a) The schematic of elastic mono-articular joint; (b) The schematic of bi-articular joint

actuator can efficiently decrease the overload and backlash in the equilibrium position of the joints. Differently from the usual use of DC motors in classical robot joints, we apply servos to substitute the traditional actuation, because positioning sensors and feedback control modules are included in servos. Thus, the accuracy of actuators rotation angles can be guaranteed and it is easier to adjust and control the stiffness in the articular movements.

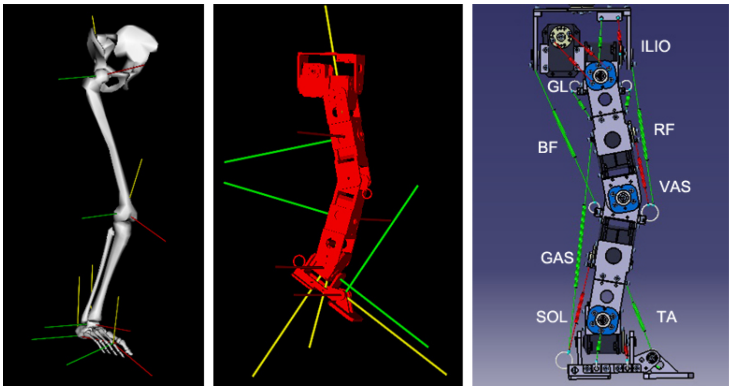


Fig. 2. Elastic Bipedal Robot Modeling. (a) Kinematic model of human lower limb; (b) Kinematic model of the robot lower limb; (c) The main muscle-tendon groups compartments considered in our modeling; green lines stand for passive springs, while the red ones are active springs.

The strong relationship existing between human musculoskeletal system and this robot is confirmed by the fact that the elastic actuation structure can be subdivided into: active and passive components which are represent in turn from unidirectional joint and bi-directional joint. In the first case, the joint is described as a single rotational servo motor of one DOF which produces the rotational motion in one direction only. A bi-directional joint is modeled as a servo motor that can rotate in two opposite directions. Both types of articulation are connected in series to active springs that are stretched or compressed as the servo motor rotates and produce passive resistive force to the rotational movement. Springs in the model can cross one (i.e. mono-articular) or two joints (i.e. bi-articular). Furthermore, the main human muscle groups are in direct correspondence with the robot elastic cables and springs as depicted in Fig. 2. The iliopsoas (ILIO) and gluteus (GL) drive the hip. The rectus femoris (RF) and the biceps femoris (BF) are to keep the equilibrium position of the thigh. The vastus (VAS) is active at the knee only while the gastrocnemius (GAS) also operates at the ankle. The tibialis anterior (TA) and the soleus (SOL) are antagonistic muscles acting at the ankle only. Among these muscles, VAS and SOL are mono-articular; the others act as bi-articular. This mapping between human and humanoid is principally based on the movement in which each muscle is involved.

2.1 Modeling of Contact between Feet and Ground

When aiming for realistic multibody simulations, an important aspect is modeling how a mechanical system interacts with the surrounding environment and how contact forces develop at its joints. Several types of surface contact models exist in literature, and each contact load may have various components, such as elastic (Hertz stiffness), viscous damping (Hunt and Crossley dissipation) and friction (Stribeck friction) [13].

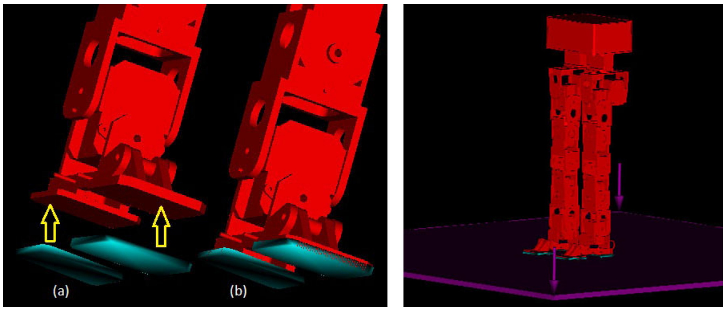


Fig. 3. Contact between feet and ground modeling

In this work, a foot-ground contact model allow modeling contact between objects that are defined by 3D polygonal meshes by utilizing an elastic foundation model [14]. It places a spring at the center of each face of each contact mesh it acts on. These springs interact with all objects the mesh comes in contact with. It is then possible to assign different physical properties to each contact mesh including: stiffness, dissipation and static, dynamic and viscous frictions.

Four triangular meshes were placed on the OpenSim model in the correspondence of heel and toe of each foot of the robot, and a fifth half-space was placed to model the floor contact, as illustrated in Fig. 3. This was done using the 3D position of the corresponding body parts, they were placed on the same plane and the floor contact mesh was consequently aligned. Finally, the dynamic contact parameters were assigned a meaningful physical value to define appropriate contact. In this work both floor location and contact parameter values were defined manually. In future implementations an optimization algorithm will be used to find optimal position of the floor and optimal contact parameters. Furthermore, more sophisticated meshes will be used. The contact model presented in this section was not used in the evaluation reported within this paper, expectant of a more precise future validation.

2.2 Modeling of Elastic Actuators

Based on the generalized coordinate system and according to Newton-Euler Equations, we can formulate the dynamics of the system as:

$$M(q)\ddot{q} + C(q,\dot{q})\dot{q} + G(q) = \tau \tag{1}$$

where q represents the angular positions of joints, $\dot{q}$ is the vector of angular velocities and $\ddot{q}$ is the vector of angular accelerations, $M(q)$ is the mass matrix, $C(q, \dot{q})$ stands for the Coriolis and centripetal forces, $G(q)$ is the gravity force, and τ is the torque of motor.

According to the spring characteristics and elasticity theory, the elasticity effects, $E(q, \dot{q})$, of springs can be represented as follow:

$$E(q, \dot{q}) = D(\dot{q}) + K(q) \tag{2}$$

where $D(\dot{q})$ is the coefficient matrix of damping, $K(q)$ is the coefficient matrix of stiffness. The elasticity effect of passive spring can be determined as follow:

$$M_p(q_p)\ddot{q}_p + E_p(q_p, \dot{q}_p)q_p = 0 \tag{3}$$

Similarly, the elasticity effect of active spring can be determined as follow:

$$M_A(q_A)\ddot{q}_A + E_A(q_A, \dot{q}_A)q_A = \tau_A \tag{4}$$

Then, the status of the elastic actuator can be described by combining (1), (3) and (4):

$$M(q)\ddot{q} + C(q, \dot{q})\dot{q} + E(q, \dot{q})q + G(q) = \tau \tag{5}$$

This dynamic equation is practical and integrated considering the influences of frictions and inertia items in robot locomotion. By varying values of stiffness and damping the dynamic situations can be modified and consequently the performances of the elastic actuators which have the effects of storing and releasing energy. Based on the principles of feed-forward controller we build up the expected control structure in which inputting the reference trajectory to the robot system, the joint positions, velocities and accelerations can be obtained by differential equations. To compensate the desired motions, a PID controller is introduced to deal with the values integrated with measurement data of sensors. Furthermore, a feed-forward controller is also used to reduce the errors produced by the characteristics of springs. Through the process of control, the robot can perform adjustable and compliant locomotion.

At this stage of work the control approach described above has been modeled through the implementation of a spring model in which we can control its force magnitude by varying only the value of stiffness as a function of a *control* value given in input to the system. With a typical linear actuator (i.e. a piston actuator) the magnitude of its force is calculated as the product:

$$F = f(control) = optimalForce \times control \tag{6}$$

and uses the convention that a positive force magnitude acts to increase the distance between the points where it is connected. Just like a linear actuator a spring acts between two points fixed on two different bodies, but its force magnitude can be calculated as:

$$F = k * (restLength - currentLength) \tag{7}$$

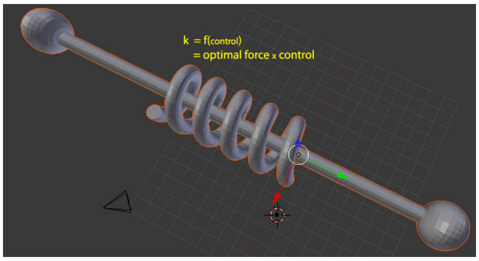


Fig. 4. Mesh representing a spring

where the spring stiffness is $k = (optimalForce \times control)$, $restLength$ is defined as the length at which the spring produces no force, and $currentLength$ is the actual length of the spring.

Each spring starts creating resistive force when the length exceeds the resting lengths: 3.2cm (ILIO), 3.5cm (GL), 13.8cm (RF), 14.6cm (BF), 6.4cm (VAS), 20cm (GAS), 8.6cm (SOL), and 7cm (TA). The control values were experimentally set in order to obtain stiffness coefficients between the range of 15-27 Newton x meters (Nm), based on the mechanical development of the robot [7, 8]. For accuracy, in the other pictures of robot the springs appear stylized through wires but have the same physical properties as defined by the model showed in Fig.4

2.3 Scaling

After the modeling phase, the first step of this study was to scale the human model to the anthropometry of a subject based on marker locations. Marker trajectories were collected during static trials from one healthy, male subject volunteered for this investigation (age: 28years, height: 183cm, mass: 67kg). Data were collected at the Gait Laboratory of the School of Sport Science Exercise and Health of the University of Western Australia. The subject was taken through the testing protocols and informed consent was obtained prior to data collection. To record the human body kinematics was used a 12 camera motion capture system (Vicon, Oxford, UK) with sampling frequency at 250Hz. Body kinematics was low-pass filtered with cut-off frequencies ranging from 2 to 8Hz depending on the trial. Starting from the original unscaled model (Weight 75Kg, Height 175cm) we obtained the new one referred to the real subject by placing a set of virtual markers on the model to match the locations of the experimental markers. Optimal fiber length and tendon slack length were scaled with the total muscle-tendon length so that they maintain the same ratio. The scaling process is an OpenSim built-in tool. In the case of robot, the scaling was performed in order to obtain a model of the robot in real size (Weight: 2.272 Kg; Height: 40 cm), starting from a generic model and preserving the distribution of mass across bodies.

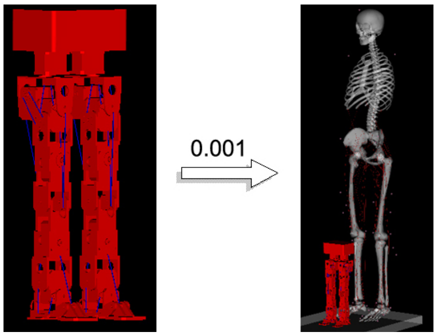


Fig. 5. Scaling procedure: from a generic model of robot, to the real-dimension biped

2.4 Computed Muscle Control

The Computed Muscle Control (CMC) [6] is a built-in tool of OpenSim. It was used, after scaling phase, to produce a dynamic simulation of muscle-tendon dynamics during walking and was applied both to the robot and human models. With regard to human model, we used the Thelen Muscle Model that is a slight modification of the Hill-type muscle model. A detailed description of the force-length, force-velocity and tendon force-strain relationships implementation can be found at [15]. We prescribed muscle-activation patterns and joint kinematics, and calculated the muscle forces and fiber lengths that satisfy these constraints.

On the other hand, in the robot we used elastic cables and extension springs instead of muscle contraction to generate the elastic forces. Servo motors are defined by parameters including: minimum and maximum allowed control values, an associated coordinate frame, and an optimal force value. In this case we prescribed the servo-motors and springs control values, and calculated the servo-motors and springs generated forces, respectively.

3 Evaluation

Experiments were performed using the real-size robot. The robot movement was first simulated using servo motors only. Then, we added the passive elastic springs. We developed some simulations in order to estimate our proposed robotic system capacities. We used the experimentally recorded human kinematics during one walking trial to directly drive our robot joints[2]. Three different type of analysis were conducted. In the first one we analyzed the kinematics of bodies that make up the models in order to obtain positions (center of mass position and orientation), velocities (linear and angular) and accelerations (linear and angular) of the COM of each segment. In the second analysis we compared the actuation force and power in joint and muscles between robot and human. The actuation power in Watt (W) was intended as the rate at which an actuator

[2] Video Available at: `http://youtu.be/-w4sEkckWY0`

produces work. Positive work means that the actuator is delivering energy to the model; negative work means that the actuator is absorbing energy from the model. In the third analysis, we performed a validation on the passive forces developed by the springs with respect to the passive forces generated by the human muscles during the same movements.

To summarize these analysis we report here the actuation torque, the actuation speed and the joint position error obtained for the right knee and ankle joints. In Fig. 6a-6d the robot develops a less torque than the human at knee and ankle joints in the same range. In the subsequent part of the cycle, from 12% to about 50% the actuation torque are very similar in all three joint: hip, knee and ankle. The main differences are observed during the swing phase of the cycle, from 62% to the next heel strike of the right foot. It shows a poor correlation between the actuation torques developed by the robot with those produced by the human. Discrepancies in the joint dynamics are mainly due to the different distribution of mass that characterizes the robot with respect to human. In Fig. 6b-6e is depicted the actuation speed obtained in each configuration, which shows an high similarity in both the knee and also the ankle joint. Small discrepancies is showed from 62% to 100% of the swing phase, for the ankle joint. Finally, Fig. 6c-6f show an estimation of the joint position error during the walking gait through one cycle, beginning and ending at heel strike of right foot. Fig.7 shows results on muscles and springs force dynamics during a walking gait through one cycle. The operating principle that governs a human muscle, characterized by active and passive forces, is not replicable in the springs which in the robot provide only passive forces. On the other hand, the active component in the actuation of the robot is represented by the engines. This justifies the comparison between passive forces developed by the human muscles and forces developed by the springs. We took as reference the values obtained from the human, depicted as a dotted line in the graphs, and compare them with those obtained with robot, simulated by using two different configuration also in this case: in the first one we moved the robot in which we introduced springs that reach a stiffness coefficient of 15Nm, meanwhile in the second one we took into account springs that reach a stiffness coefficient of 27Nm. Fig. 7a shows the force produced by the tibialis anterior muscle, Fig. 7b shows the force produced by the gastrocnemius, and Fig.7c is about the vastus muscle, Fig.7d depict the forces about soleus, in Fig.7e is referred to the passive forces developed by bicep femoris muscle group, and finally the Fig.7f in which there is the rectus femoris passive forces, both in the robot and in the human. A Pearson Correlation Coefficient (PCC) analysis shows values varying into the range 0.64 - 0.85. Forces and torques were scaled by the weight of the subject and the robot respectively in order to account for the different arrangements of the reference systems of the lower limb parts, together with the different position and orientation of the COM, and the resulting different distribution of mass and anthropometric characteristics of the human subject and robot. The activation timing and the shape of the curves are extremely similar despite the different dynamics observed at the joint level and the different operating principles between human muscle and spring.

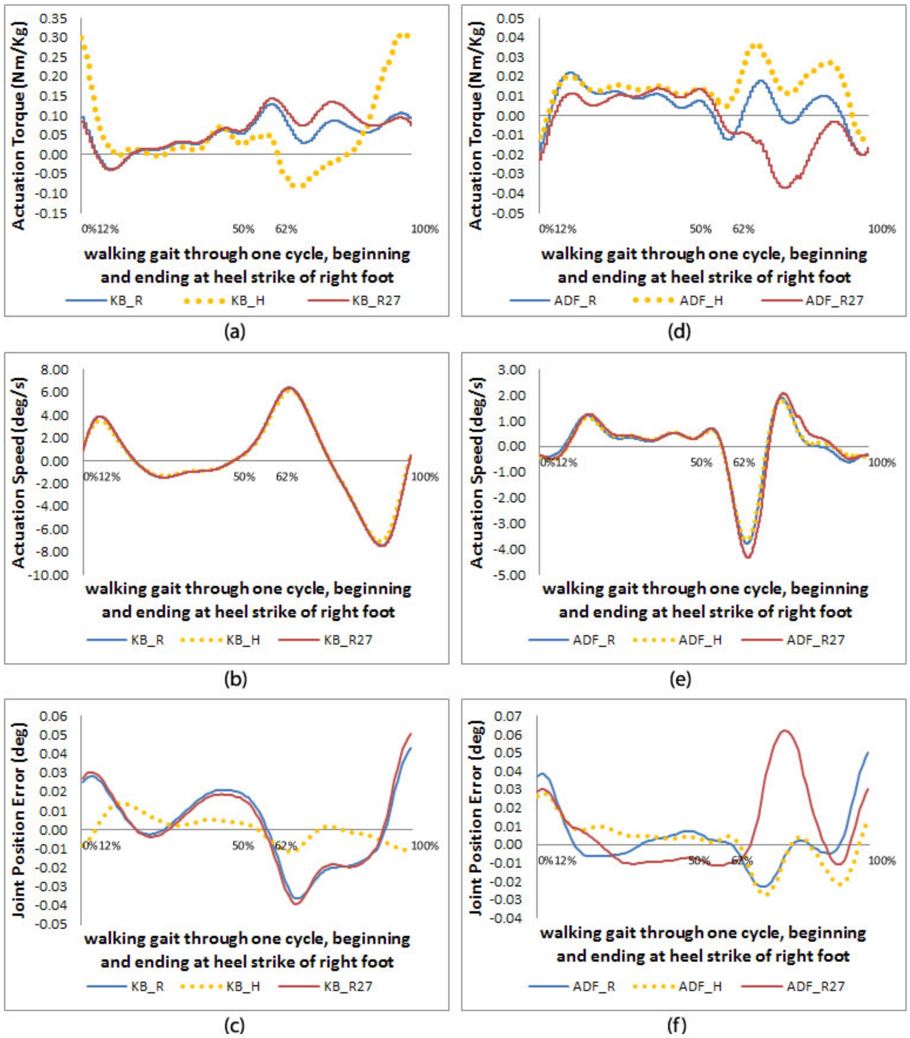


Fig. 6. (a) Actuation Torque of Knee joint; (b) Actuation Speed of the Knee joint; (c) Position Error of the Knee joint; (d) Actuation Torque of the Ankle joint; (e) Actuation Speed of the Ankle joint; (f) Position Error of the Ankle joint. In each graph two configuration of robot (without springs: KB_R, ADF_R; with springs characterized by a stiffness coefficient of 27Nm: KB_R27, ADF_R27) are plotted with the reference values of human model (KB_H, ADF_H).

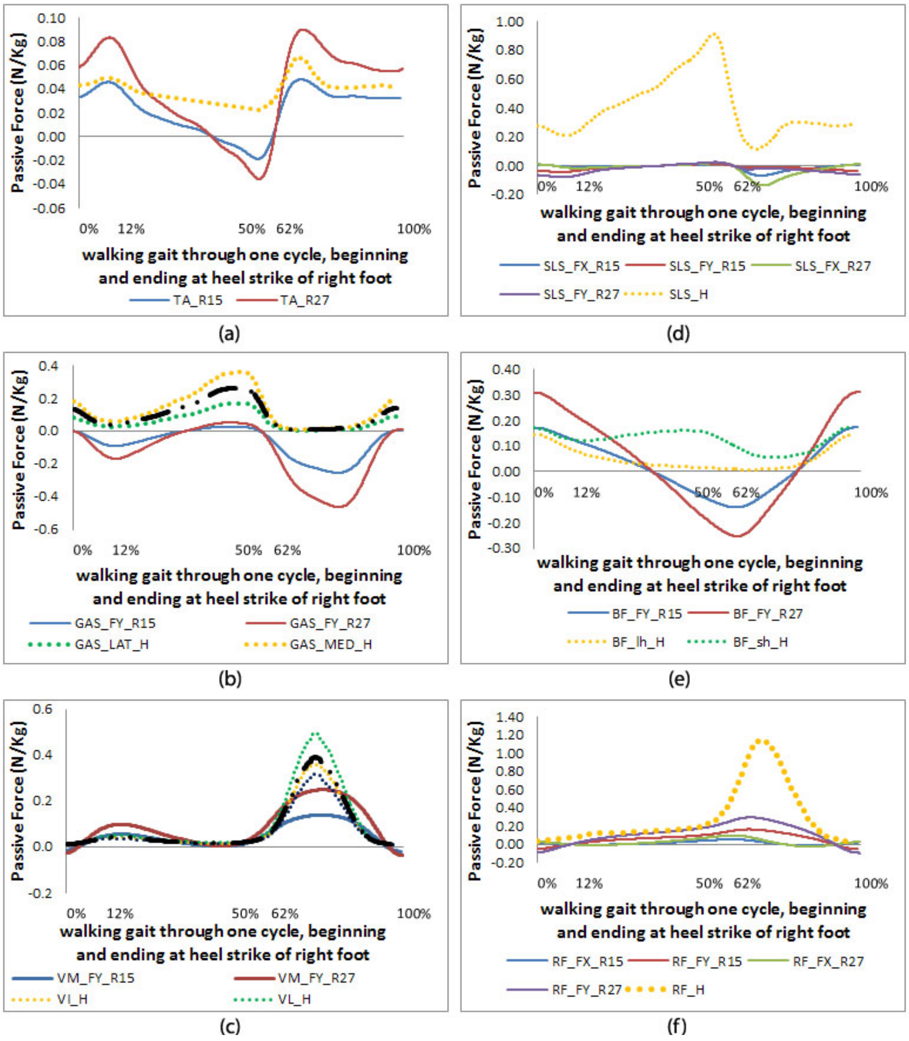


Fig. 7. Passive Force of Springs and Muscles [N/Kg]. (a)Tiabiali Anterior (TA); (b)Gastrocnemius (GAS); (c)Vastus Medialis (VM); (d)Soleus (SLS); (e)Bicep Femoris (BF); (f)Rectus Femoris (RF). For each muscle/spring two configuration of robot (with springs characterized by a stiffness coefficient of 15Nm: TA_R15, GAS_FY_R15, VM_FY_R15, SLS_FX_R15-_FY_R15, BF_FY_R15, RF_FX_R15-_FY_R15; with springs characterized by a stiffness coefficient of 27Nm: TA_R27, GAS_FY_R27, VM_FY_R27, SLS_FX_R27-_FY_R27, BF_FY_R27, RF_FX_R27-_FY_R27) are plotted with the reference values of human model (TA_H, GAS_LAT_H, GAS_MED_H, VI_H, VL_H, VM_H, SLS_H, BF_lh_H - BF_sh_H, RF_H).

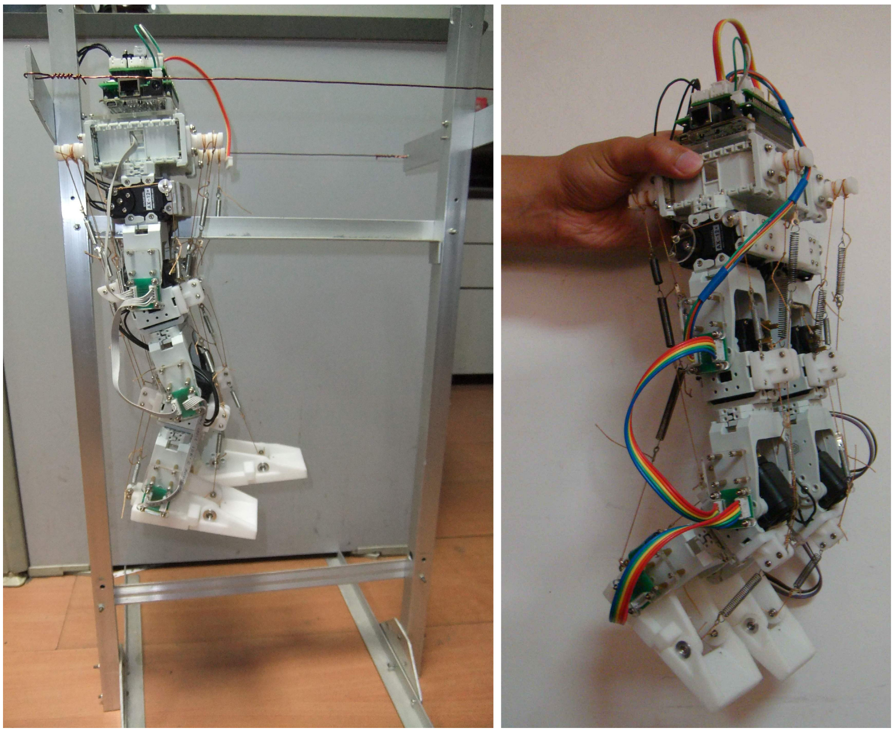


Fig. 8. Two pictures of the first prototype developed

4 Conclusion and Future Work

Starting from the studies developed in the biomechanical and biomimetic fields, this paper introduces the modeling and simulation of a novel bipedal robot equipped with a musculoskeletal-like mechanism. The description of the real robot prototype platform, illustrated in Fig. 8, that has been built at Dalian University of Technology, will be described in detail in a future paper.

We want to make a robot that has passive structures which allow to reproduce the dynamics of force that characterize the passive structures of human muscle. This is because the human muscles are efficient structures. For this purpose we will compare the forces produced by the springs with the forces generated by passive components in human muscle. Since it is not possible to measure forces directly from human muscle, a simulation approach is needed. The focus of this paper was motivated by the potential of making the motion of the humanoid robots more compliant and energy efficient. Furthermore, the bio-inspired arrangement of the artificial muscles allowed reproducing in the robot the mechanisms underlying the passive elasticity of human muscles (Fig. 7). We want now to further improve our study by defining other type of contact model between foot and ground in order to fully appreciate the compliant effects due to springs, but we want also to introduce a trunk in order to take into account that man is a vertebrate and its locomotion is clocked and driven by his trunk. The development of our proposed methodology will also provide insights into the design of robotics prosthetics and powered orthoses [16, 17]. This work is the starting point of a wide range of other possible future works: from the control structure completion and whole-body control application, from motion test on flat ground to motion test on rough ground with stable and compliant behaviors of walking, and obviously the transition from simulation to practice with the real novel elastic bipedal robot biologically-inspired.

References

1. Universite de Geneve, Project: 3D anatomical functional models for the human musculoskeletal system. Start:2006-10-01; End:2010-09-30, http://www.miralab.ch/
2. Geyer, H., Herr, H.: A Muscle-Reflex Model that Encodes Principles of Legged Mechanics Produces Human Walking Dynamics and Muscle Activities. IEEE Trans. on Neural Systems and Rehabilitation Engineering 18(3), 263–273 (2010)
3. Iida, F., Rummel, J., Seyfarth, A.: Bipedal walking and running with spring-like biarticular muscles. Journal of Biomechanics 41 (2008)
4. Radkhah, K., Lens, T., Seyfarth, A., von Stryk, O.: On the influence of elastic actuation and monoarticular structures in biologically inspired bipedal robots. In: Proc. of the 2010 IEEE Int. Conf. on Biomedical Robotics and Biomechatronics (2010)
5. Radkhah, K., Maus, M., Scholz, D., Seyfarth, A., von Stryk, O.: Toward Human-Like Bipedal Locomotion with Three-Segmented Elastic Legs. In: 41st Int. Symp. on Robotics/6th German Conf. on Robotics, pp. 696–703 (June 2010)
6. Delp, S.L., Anderson, F.C., Arnold, A.S., Loan, P., Habib, A., John, C.T., Guendelman, E., Thelen, D.G.: OpenSim: Open-Source Software to Create and Analyze Dynamic Simulations of Movement. IEEE Trans. on Biomedical Engineering 54(11) (November 2007)
7. Bortoletto, R.: Simulating a Flexible Robotic System based on Musculoskeletal Model. M.Sc Thesis - Department of Information Engineering, University of Padua (December 2011)
8. He, F., Liang, Y., Zhang, H., Pagello, E.: Modeling, Dynamics and Control of an Extended Elastic Actuator in Musculoskeletal Robot System. To appear in the Proc. of IAS 2012, Jeju (Korea), June 26-29 (2012)
9. Delp, S.L., Loan, J.P., Hoy, M.G., Zajac, F.E., Topp, E.L., Rosen, J.M.: An interactive graphics-based model of the lower extremity to study orthopaedic surgical procedures. IEEE Trans. on Biomedical Engineering (1990)
10. Anderson, F.C., Pandy, M.G.: A dynamic optimization solution for vertical jumping in three dimensions. Computer Methods in Biomechanics and Biomedical Engineering 2, 20131 (1999)
11. Anderson, F.C., Pandy, M.G.: Dynamic optimization of human walking. Journal of Biomechanical Engineering 123, 381–390 (2001)
12. Yamaguchi, G.T., Zajac, F.E.: A planar model of the knee joint to characterize the knee extensor mechanism. Journal of Biomechanics 22(1), 10 pages (1989)
13. Adams, G.G., Nosonovsky, M.: Contact modeling - forces. Tribology International 33, 431–442 (2000), www.elsevier.com/locate/triboint

14. Kalker, J.: Three-dimensional elastic bodies in rolling contact. In: Solid Mechanics and Its Application. K. A. Publishers (1990)
15. John, C.T.: Complete Description of the Thelen2003Muscle Model, `http://simtk-confluence.stanford.edu:8080`
16. Sartori, M., Reggiani, M., Lloyd, D.G., Pagello, E.: A neuromusculoskeletal model of the human lower extremity: Towards EMG-driven actuation of multiple joints in powered orthoses. In: Proceedings of IEEE Int. Conf. on Rehabilitation Robotics (ICORR 2011), Switzerland (June 2011)
17. Sartori, M., Lloyd, D.G., Reggiani, M., Pagello, E.: Fast Runtime Operation of Anatomical and Stiff Tendon Neuromuscular Models in EMG-driven Modeling. In: Proc. of IEEE Int. Conf. on Robotics and Automation (ICRA 2010), USA (May 2010)

Simulation and Experimental Evaluation of the Contribution of Biarticular Gastrocnemius Structure to Joint Synchronization in Human-Inspired Three-Segmented Elastic Legs

Dorian Scholz[1], Christophe Maufroy[2], Stefan Kurowski[1], Katayon Radkhah[1], Oskar von Stryk[1], and André Seyfarth[2]

[1] Fachgebiet Simulation, Systemoptimierung und Robotik
Technische Universität Darmstadt, Department of Computer Science
Hochschulstr. 10, D-64289 Darmstadt, Germany
{scholz,kurowski,radkhah,stryk}@sim.tu-darmstadt.de
http://www.sim.tu-darmstadt.de

[2] Lauflabor Locomotion Laboratory
Technische Universität Darmstadt, Institut für Sportwissenschaft
Magdalenenstr. 27, D-64289 Darmstadt, Germany
{cmaufroy,seyfarth}@sport.tu-darmstadt.de
http://www.lauflabor.de

Abstract. The humanoid robot BioBiped2 is powered by series elastic actuators (SEA) at the leg joints. As motivated by the human muscle architecture comprising monoarticular and biarticular muscles, the SEA at joint level are supported by elastic elements spanning two joints. In this study we demonstrate in simulation and in robot experiments, to what extend synchronous joint operation can be enhanced by introducing elastic biarticular structures in the leg, reducing the risk of over-extending individual joints.

1 Introduction

During bouncing gaits such as hopping and fore-foot running, the three-segmented human leg is loaded and unloaded during contact time while pivoting around the ball of the foot [1]. This observation contrasts with the current state-of-the-art running biped robots, which either run with their feet flat on the ground (such as ASIMO [2]) or have no foot at all and are equipped with pogo-stick [3] or two-segmented [4] legs. While this latter strategy is reasonable for running, it may be problematic for other gaits, such as walking and standing, where the role of the ankle-foot complex becomes important for posture control and energy injection (during the late stance push-off). This explains why many walking robots (such as [2][5]) have feet. Hence, the previous argument suggests that humanoid robots aiming at multimodal locomotion should be equipped with three-segmented legs. A better understanding of the dynamic operation of this type of leg structure, especially during bouncing gaits, would help in exploiting the benefits of this leg design.

In addition to segmentation, compliance is another key aspect of human leg operation. Elastic leg operation is widely observed in human locomotion [6] and biomechanical models suggest that this property may provide the mechanical basis of hopping, running and walking [7][8][9]. The elasticity of the leg as a whole is supported by elastic operation at the level of the individual joints, primarily ankle and knee [1][6]. These observations have fuelled the development of compliant actuators, including the series elastic actuators [10], and the progressive shift of compliance from the leg level (as in Raibert's hoppers [3]) to the joint level (as in M2V2 [11] or BioBiped1 [12]) in legged robots. However, the implementation of compliance at the joint level comes with the challenge of maintaining the leg's configurational stability. Using stability analysis in the static case, Seyfarth et al. [13] pointed out the risk of bifurcations from usual zigzag leg configuration to bow leg configuration, in which either the knee or ankle joints is overextended.

In the dynamic case, similar concerns related to the over-extension of one of the joints exist during the loading and unloading phases of the three-segmented leg in bouncing gaits. This risk could be mitigated by finding the properties of the elastic structures acting at the knee and ankle that can guarantee robust synchronous operation of these joints such as observed in human hopping [14]. This is precisely the goal of this paper. In addition to monoarticular structures, a biarticular elastic structure, mimicking the human gastrocnemius muscle, is considered. Our hypothesis, motivated by the known role of biarticular muscles in inter-joint coordination in humans [15][16], is that the use of biarticular structures like the gastrocnemius can increase the robustness of the behavior with respect to initial leg configurations and the spring stiffness ratio of the monoarticular structures. This is particularly important when it comes to real world application of the robot, where precise adjustments of these quantities are difficult to achieve, due to sensor and modeling inaccuracies as well as environmental factors, such as not perfectly flat ground, leading to variable foothold position.

2 Simulation and Experimental Setup

2.1 Experimental Framework

In this paper, passive rebound experiments were used as a simplified experimental framework to investigate the influence of the passive elastic structures in the segmented leg during hopping. The robot was dropped from a given height, landed with its foot tips vertically aligned with the hip joint and the subsequent rebound, resulting only from the action of the passive elastic leg structures, was observed. These experiments were performed on the BioBiped2 robot (see Fig. 1c) and in simulation using an approximate model of robot. The BioBiped2 is a revised version of BioBiped1 described in [12], which was improved compared to its predecessor in many electronic and mechanical design details. For the experiments presented here, the main improvement is the usage of ball bearings in the joints, drastically reducing the friction and allowing for an easier investigation of the elastic mechanism.

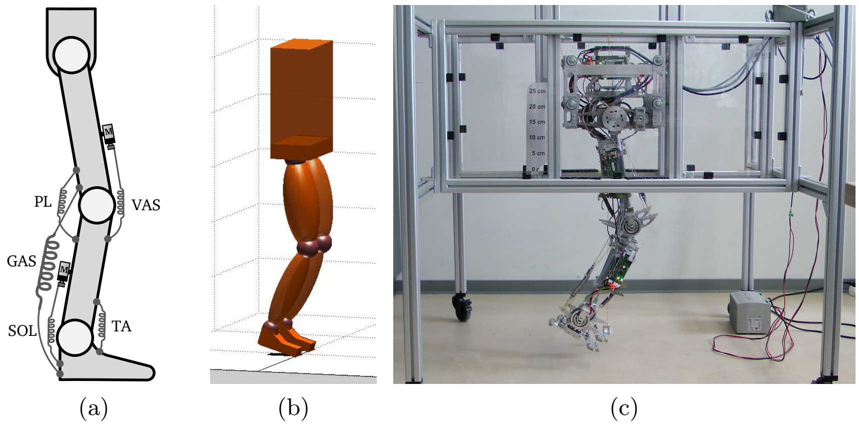


Fig. 1. (a) Elastic structures used during the passive rebound experiment (b) Snapshot of the BioBiped simulation model (c) BioBiped2 in the experimental setup for the robot experiments

For simplicity, the trunk motion of the robot was constrained to vertical motions and only elastic structures at the knee and ankle joint were considered (see Fig. 1a). At knee and ankle joints, the motor positions of the series elastic actuators, playing the role of the extensor muscles (VAS and SOL), were set to balance the joint torques generated by the passive flexor structures (PL and TA) in the initial leg configuration. Their position was subsequently held constant so that the generated torques at the knee and ankle were only the result of the passive elastic structures. To gain insight in the effect of the gastrocnemius structure (GAS), every experiment was performed twice: with and without a simplified GAS, implemented as a linear spring (see Table 1) connecting heel and thigh. It was mounted to be at its rest length in the initial leg configuration at touch-down.

The model of the robot, implemented and simulated using the framework presented in [17], is represented in Fig. 1b. The simulation model parameters are summarized in the Appendix, in Table 2. For simplicity, we consider that the joints are frictionless and that no energy dissipation occurs in the elastic structures.

The setup for the robot experiments is depicted in Fig. 1c. The constraint on the trunk motion is achieved using a frame that prevents all but the vertical motion, while rollers attached to the robot insure low friction along that direction.

2.2 Joint Synchronization Index

The synchronization of the knee and ankle movements was quantified using the phase difference $\Delta\phi$ between the flexion-extension motions of these two joints, as given by the following equation:

$$\Delta\phi = |\frac{t_K - t_A}{T}| = |\frac{\Delta t}{T}|$$

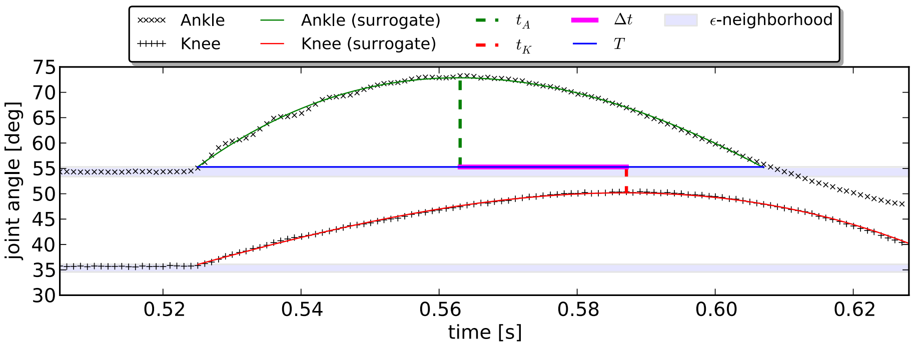


Fig. 2. Visualization of the joint trajectories during one of the robot experiments. Besides the actual measurement data, the graph shows the surrogate functions, the total time T between leaving the ϵ-neighborhood and reentering it and the time difference Δt between the trajectories' maxima (t_A, t_K).

where t_K and t_A are respectively the instants when the knee and the ankle are maximally flexed during contact, while T is the time, measured from landing, until either joint angle reaches its original landing value (see Fig. 2). For the estimation of t_K and t_A in the robot experiments, the data around the maximal flexion peak was approximated by a surrogate function (6th order, generated by regression using 40 data points) to reduce the influence of measurement noise. The measurement data together with the surrogate function for an example trajectory are displayed in Fig. 2. To enhance the robustness of phase-length detection, an ϵ-neighborhood was introduced around the value of the landing angle and T was defined for all experiments as the time from leaving this ϵ-neighborhood until the first trajectory reenters it. The value of ϵ was set to 5% of the difference between the landing angle and maximum flexion angle.

2.3 Parameter Space

The influence of the following two parameters on knee and ankle joint synchronization was investigated:

– the spring stiffness ratio $R = k_{\text{SOL}}/k_{\text{VAS}}$
– the joint angle difference at landing $\Delta\theta = \theta_{K,0} - \theta_{A,0}$

where k stands for the linear stiffness of the elastic structures and the indices K and A refer to the knee and ankle joints, respectively.

The other parameters were kept constant during all experiments. Although the initial leg configuration was variable and dependent on $\Delta\theta$, the joint angles were chosen to maintain the total initial leg length L_0 (distance from hip joint to foot tip) at a constant fraction of the maximum leg length L_{max} (sum of all leg segment lengths). The value of the initial leg length L_0 was set to the average value found in humans at preferred hopping frequency as described in [18].

Similarly, k_{SOL} was set constant, while k_{SOL} was computed based on the spring stiffness ratio R. The value of k_{VAS} was chosen to results in similar maximal leg compression as during human hopping (i.e. about 10% of L_{max}).

The values of the constant parameters are given in the upper part of Table 1. The lower part of Table 1 shows the values of the stiffness ratio R and the angle difference $\Delta\theta$ used in the robot experiments. Every combination of these configurations was tested on the robot with and without the gastrocnemius structure GAS. In simulation, many more configuration within the same parameter range were tested to produce more fine grained results.

Table 1. Constant and variable parameters used during the experiments: leg lengths L_{max} and L_0, spring stiffnesses k and pretensions F_0, spring stiffness ratios R and joint angles θ

Constant parameters								
L_{max}	[m]	0.727	k_{VAS}	[N/mm]	15.5	$F_{0_{\text{VAS}}}$	[N]	36.8
L_0	[m]	0.94 L_{max}	k_{GAS}	[N/mm]	7.9	$F_{0_{\text{GAS}}}$	[N]	27.6
			$k_{\text{PL/TA}}$	[N/mm]	4.1	$F_{0_{\text{PL/TA}}}$	[N]	13.8

Stiffness ratio R in robot experiments						
Experiment		A	B	C	D	E
R	[-]	0.265	0.432	0.510	0.839	1.155
k_{SOL}	[N/mm]	4.1	6.7	7.9	13.0	17.9
$F_{0_{\text{SOL}}}$	[N]	13.8	22.6	27.6	27.6	58.9

Angle difference $\Delta\theta$ in robot experiments						
Experiment		1	2	3	4	5
$\Delta\theta$	[deg]	-7	-0.5	6.6	14.8	24.7
knee $\theta_{K,0}$	[deg]	138	139.5	141.6	144.8	149.7
ankle $\theta_{A,0}$	[deg]	145	140	135	130	125

3 Results and Discussion

In this study the effect of GAS on synchronous joint operation in a three-segmented leg is studied in simulation and compared to robot experiments with BioBiped2.

The phase difference $\Delta\phi$, represented in Fig. 3 as a function of the stiffness ratio R and the initial leg configuration, is characterized by the angle difference at landing $\Delta\theta$. The results are shown without and with the GAS structure attached for simulation (Fig. 3a and 3b) and robot experiments (Fig. 3c and 3d).

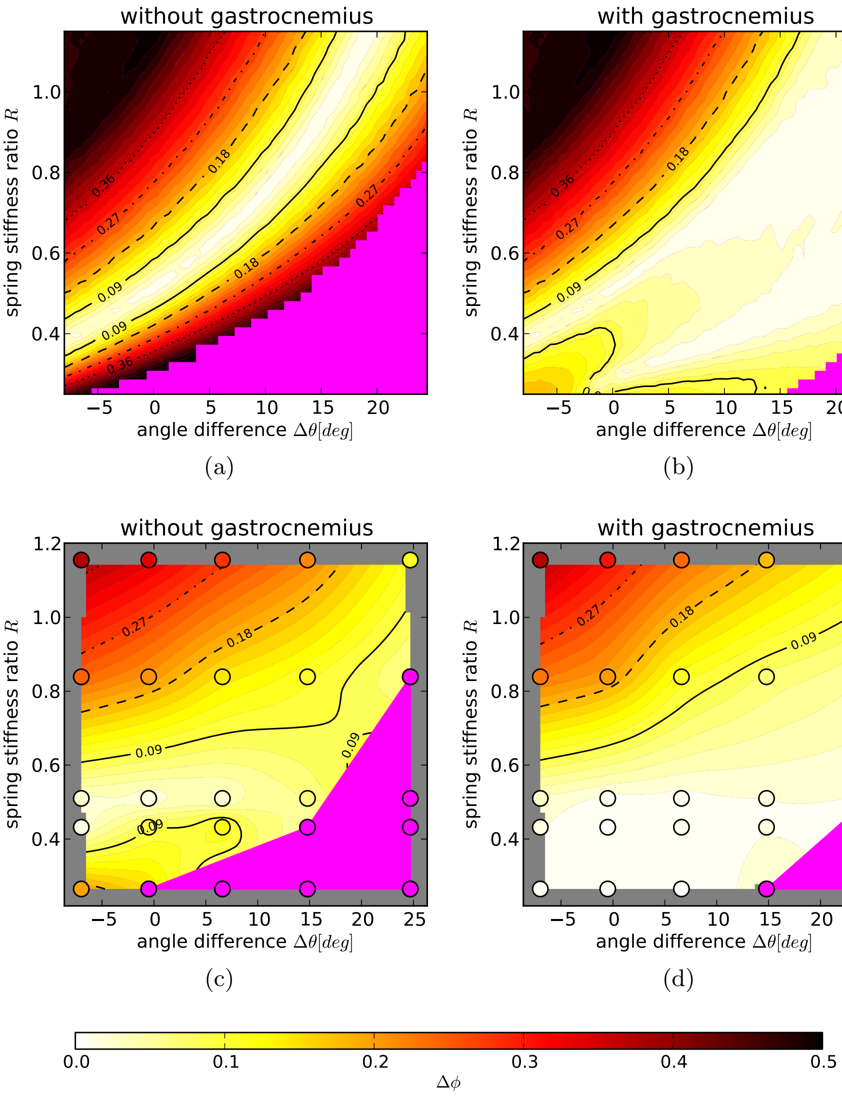


Fig. 3. Phase differences of knee and ankle joints in the simulation (3a, 3b) and in the robot experiments (3c, 3d) each without and with GAS. The trials where heel contact occurred during the stance phase are located in the lower right corner in both simulation and experiments and are marked in magenta. The configurations used for the 25 robot experiments (black circles) are shown in Table 1 and the angles' definitions in Fig. 4 (Appendix). As these configurations are not equidistant in the graph the $\Delta\phi$ values in-between the experiments have been linearly interpolated for easier comparison with the simulation results.

The simulation results show that, even without the GAS structure, synchronous operation of the knee and ankle joints is possible in most of the range considered for the angle difference $\Delta\theta$. However, this requires a fine adjustment of the stiffness ratio R to fall in the thin white region of Fig. 3a. This is particularly true for small values of $\Delta\theta$ (i.e. the landing configuration with congruent knee and ankle angles) where the sensitivity with respect to R appear to be the largest. On the other hand, the synchronous operation becomes less sensitive to variation of R as the angle difference $\Delta\theta$ increases. This situation corresponds to a landing configuration with extended knee and flexed ankle, which is favored by humans [18].

Adding the GAS structure has a considerable influence on the results (Fig. 3b). The parameter region where synchronous joint operation occurs with $\Delta\phi < 0.05$ is considerably enlarged. Hence, the sensitivity of the behavior with respect to the stiffness ratio R is greatly reduced, especially for large angle differences $\Delta\theta$. This allows the system to potentially operate with various overall leg stiffness, by varying the stiffness ratio R, while preserving the joint synchronization. In addition, the risk of heel strike leading to energy dissipation due to the impact with the ground is reduced (see magenta area in Fig. 3).

Some of the tendencies observed in simulations are found in the results of the robot experiments. Generally, synchronous operation is improved when the angle differences $\Delta\theta$ is positive. Additionally, good joint synchronization is possible (with phase differences $\Delta\phi < 0.10$), even without GAS structure (see Fig. 3c), but the synchronization is notably improved by the addition of the GAS structure. It also reduces the risk of heel strike.

Besides these common tendencies, the results for the robot experiments present specific features worth to discuss. First, the region of parameters resulting in low phase differences ($\Delta\phi < 0.20$) without the GAS structure is much more extended than in the simulations. As a result, the effect of the addition of GAS is not as pronounced as for the simulation results.

Another discrepancy between the robot experiments and the simulations is the spring model. In simulation a linear extension springs without pretension in used. In reality, the extension springs are not perfectly linear and have a significant pretension (see values for F_0 in Table 1). Hence, the apparent stiffness of the spring is altered and the ratio computed using the nominal spring stiffnesses may not reflect this change. This could potentially explain why low phase differences are observed in the robot experiments for much lower values of R than in the simulations.

4 Conclusions and Future Work

The results in Fig. 3 show that it is possible to achieve synchronized joint movements without gastrocnemius. But the corresponding parameter region is quite limited, in the simulation (Fig. 3a) as well as on the real robot (Fig. 3c). The in-phase operation of knee and ankle joints can be supported by an elastic biarticular structure (GAS) mimicking the function of the human gastrocnemius muscle. This was demonstrated in simulation and for the BioBiped2 robot as can be seen in Fig. 3b and Fig. 3d. More specifically the phase difference was reduced for every leg configuration tested on the robot, thus making it possible to get synchronized joint movements even without perfect touchdown conditions. Interestingly, the range of in-phase joint operation was even larger in the robot than predicted by the simulation model. This indicates, that other effects (e.g. joint damping) may further facilitate synchronous joint function. When looking at robots in real world scenarios variations in leg configurations are inevitable. The additional robustness gained through the biarticular structure against changes in leg configuration could help in solving the challenges of bipedal walking on rough terrain and unstructured environment.

Furthermore this additional robustness opens the possibility to reduce the effort in terms of sensory feedback and energy input on joint level while still achieving equally good overall leg performance. Another way of looking at this is the shift of parts of the control to the distribution of elastic structures and actuators in the segmented body.

In future work, the influence of the springs pretension on the results could be investigated by using overextended springs. Additionally, the evaluation, focused so far to the knee and ankle joints, will be extended to the hip joints. For that purpose, the constraints on the trunk will be relaxed and the elastic structures spanning this joint will be added. Yet another avenue of research will be to investigate how the benefits of the biarticular structures shown here in the passive case translate to an actively controlled motion. One interesting aspect would be the possibility to reduce the control effort and the energy consumption necessary for synchronous joint operation, i.e. in continuous hopping.

Acknowledgment. This research has been supported by the German Research Foundation (DFG) under grants no. SE 1042/6-1 and STR 533/7-1 and within the Research Training Group 1362 "Cooperative, adaptive and responsive monitoring in mixed mode environments".

References

1. Guenther, M., Blickhan, R.: Joint stiffness of the ankle and the knee in running. J. Biomech. 35, 1459–1474 (2002)
2. Asimo webpage, `http://world.honda.com/asimo/`
3. Raibert, M.H.: Legged Robots that Balance. MIT Press, Cambridge (1986)
4. Hurst, J.W.: The Role and Implementation of Compliance in Legged Locomotion. PhD thesis, The Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (2008)
5. Petman webpage, `http://www.bostondynamics.com/robot_petman.html`
6. Lipfert, S.W.: Kinematic and Dynamic Similarities between Walking and Running. Verlag Dr. Kovac, Hamburg (2010)
7. Blickhan, R.: The spring-mass model for running and hopping. Journal of Biomechanics 22, 1217–1227 (1989)

8. Seyfarth, A., Geyer, H., Guenther, M., Blickhan, R.: A movement criterion for running. J. Biomech.
9. Geyer, H., Seyfarth, A., Blickhan, R.: Compliant leg behaviour explains basic dynamics of walking and running. Proc. Royal Society B: Biological Sciences 273, 2861–2867 (2006)
10. Pratt, G.A., Williamson, M.M.: Series elastic actuators. In: Proc. IEEE International Workshop on Intelligent Robots and Systems, pp. 399–406 (1995)
11. Pratt, J., Krupp, B.: Design of a bipedal walking robot. In: Proceedings of the 2008 SPIE, vol. 6962 (2008)
12. Radkhah, K., Maufroy, C., Maus, M., Scholz, D., Seyfarth, A., von Stryk, O.: Concept and design of the biobiped1 robot for human-like walking and running. International Journal of Humanoid Robotics 8(3), 439–458 (2011)
13. Seyfarth, A., Guenther, M., Blickhan, R.: Stable operation of an elastic three-segmented leg. Biol. Cybern. 84, 365–382 (2001)
14. Rapoport, S., Mizrahi, J., Kimmel, E., Verbitsky, O., Isakov, E.: Constant and variable stiffness and damping of the leg joints in human hopping. J. Biomech. Eng. 125(4), 507–514 (2003)
15. Zajac, F.E.: Muscle coordination of movement: a perspective. J. Biomech. 26, 109–124 (1993)
16. van Ingen Schenau, G.J., Pratt, C.A., Macpherson, J.M.: Differential use and control of mono- and biarticular muscles. Human Movement Science 13(3-4), 495–517 (1994)
17. Lens, T., Radkhah, K., von Stryk, O.: Realistic contact forces for manipulators and legged robots with high joint elasticity. In: Proc. 15th International Conference on Advanced Robotics (ICAR), pp. 34–41 (2011)
18. Farley, C.T., Morgenroth, D.C.: Leg stiffness primarily depends on ankle stiffness during human hopping. J. Biomech. 32, 267–273 (1999)

5 Appendix

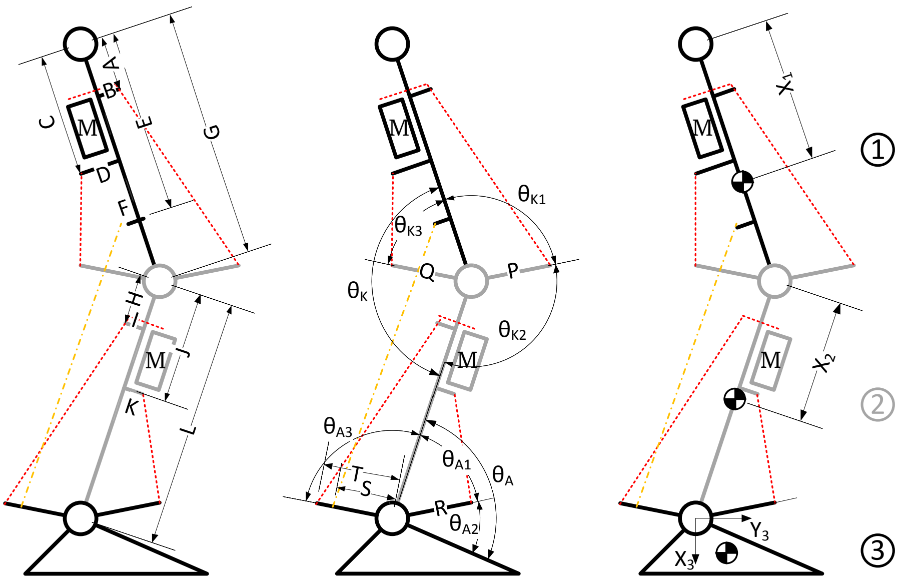


Fig. 4. Dimensions of the BioBiped2 leg. Corresponding values are listed in Table 2.

Measurement	Value	Unit	Measurement	Value	Unit
A	0.080	[m]	K	0.058	[m]
B	0.018	[m]	L	0.330	[m]
C	0.226	[m]	P	0.068	[m]
D	0.070	[m]	Q	0.038	[m]
E	0.226	[m]	R	0.023	[m]
F	0.070	[m]	S	0.053	[m]
G	0.330	[m]	T	0.061	[m]
H	0.076	[m]	θ_{K2}	155	[deg]
I	0.022	[m]	$\theta_{A1} + \theta_{A2} + \theta_{A3}$	213	[deg]
J	0.210	[m]			

Table 2. Values of the dimensions represented in Fig. 4. These values are also used in the simulation model.

Graph Optimization with Unstructured Covariance: Fast, Accurate, Linear Approximation

Luca Carlone[1], Jingchun Yin[1], Stefano Rosa[2], and Zehui Yuan[1]

[1] Dipartimento di Automatica e Informatica, Politecnico di Torino, Italy
{luca.carlone,jingchun.yin,zehui.yuan}@polito.it
[2] Italian Institute of Technology (IIT), Torino, Italy
stefano.rosa@iit.it

Abstract. This manuscript addresses the problem of optimization-based Simultaneous Localization and Mapping (SLAM), which is of concern when a robot, traveling in an unknown environment, has to build a world model, exploiting sensor measurements. Although the optimization problem underlying SLAM is nonlinear and nonconvex, related work showed that it is possible to compute an accurate linear approximation of the optimal solution for the case in which measurement covariance matrices have a block diagonal structure. In this paper we relax this hypothesis on the structure of measurement covariance and we propose a linear approximation that can deal with the general unstructured case. After presenting our theoretical derivation, we report an experimental evaluation of the proposed technique. The outcome confirms that the technique has remarkable advantages over state-of-the-art approaches and it is a promising solution for large-scale mapping.

Keywords: Pose graph optimization, Simultaneous Localization and Mapping, Mobile robotics.

1 Introduction

In several application scenarios (e.g., search and rescue, planetary exploration, disaster response) mobile robots are deployed in an unknown environment and are required to build a model (*map*) of the surroundings. The map is often used for planning human intervention or for enhancing situational awareness. Therefore, the mapping process is guided by three main requirements: (i) *accuracy*, since a misleading representation of the environment can seriously compromise the operation of human (or robotic) operators within the scenario, (ii) *efficiency*, since it is crucial to have time-sensitive information on the environment, (iii) *scalability*, since the robot may be in charge of mapping large areas.

Pose graph optimization has recently emerged as an effective problem formulation for SLAM. In a *pose graph*, each node represents a pose assumed by a mobile robot at a certain time, whereas an edge exists between two nodes if a relative measurement (inter-nodal constraint) is available between the corresponding poses. Inter-nodal constraints are usually obtained by means of proprioceptive sensors (*odometry*) or exteroceptive sensor-based techniques (*vector registration*, *scan matching*, etc.) [2]. Then, the objective of pose graph optimization is to estimate the nodes' poses (*pose graph configuration*) that maximize the likelihood of inter-nodal measurements. The utility of estimating the configuration of the pose graph stems from the fact that, from the estimated poses and from sensor measurements, it is then easy to construct a map of the environment. After the seminal paper [12], several authors put their efforts in devising sustainable and accurate solutions to pose graph optimization. Thrun and Montemerlo [17] enabled the estimation of large maps using a conjugate gradient-based scheme. Konolige [9] investigated a reduction scheme for reducing the number of nodes involved in the optimization. Frese et al. proposed a multilevel relaxation approach for full SLAM [7]. Olson et al. [14] proposed the use of incremental pose parametrization for improving efficiency and convergence. Grisetti et al. [8] extended such framework, taking advantage of the use of stochastic gradient descent in planar and three-dimensional scenarios. In [11] the authors presented a general framework for the optimization of graph-based nonlinear error functions. More recently, Sünderhauf et al. [16] stressed the topic of outliers rejection in pose graph optimization, proposing a strategy for discarding erroneous loop closure constraints. All the aforementioned techniques are iterative, in the sense that, at each iteration, they solve a local convex approximation of the original problem, and use such local solution to update the configuration [6]. This process is then repeated until the optimization variable converges to a local minimum of the cost function. As a consequence, all mentioned techniques require the availability of an initial guess for nonlinear optimization, which needs to be sufficiently accurate for the technique to converge to a global solution of the problem. A partial answer to these two problems (computational complexity and need of an accurate initial guess) came from the work [3]. In [3] the authors proposed a linear approximation for the pose graph configuration, assuming that the measurement covariance matrices have a block diagonal structure. The approach requires no initial guess and was shown to be accurate in practice.

In this article we extend and complement the previous work [3], proposing two contributions: (i) we relax the hypothesis of structured measurement covariances and we propose an approach that is able to deal with the full covariance case; (ii) we present an extensive evaluation of the performance of the proposed approach, compared with the approach of [3] and with other state-of-the-art techniques. The first contribution (Section 3) is more theoretical: we describe an algorithm for estimating the pose graph configuration and we then prove that it corresponds to a Gauss-Newton steps around a suitable suboptimal solution. The algorithm is an extended version of the approach proposed in [3]; also the proof proceeds on the same line, although it encompasses the more general case of full covariances. The second contribution (Section 4) is experimental. We test several state-of-the-art techniques on real and simulated datasets and we propose a performance evaluation in terms of accuracy, efficiency, and scalability.

2 Problem Formulation

The objective of pose graph optimization is to provide an estimate of the poses assumed by a mobile robot, namely $X = \{x_0, \ldots, x_n\}$. X is called a *configuration* of poses; the $n+1$ poses are in the form $x_i = [p_i^\top \ \theta_i]^\top \in \mathrm{SE}(2)$, where $p_i \in \mathbb{R}^2$ is the Cartesian position of the i-th pose, and θ_i is its orientation. The input for the estimation problem are m measurements of the relative pose between pairs of nodes. For instance a measurement $\bar{\xi}_{i,j}$ between nodes i and j is in the form

$$\bar{\xi}_{i,j} = \xi_{i,j} + \epsilon_{i,j} = \begin{bmatrix} R_i^\top (p_j - p_i) \\ \langle \theta_j - \theta_i \rangle_{2\pi} \end{bmatrix} + \begin{bmatrix} \epsilon_{i,j}^{\Delta} \\ \epsilon_{i,j}^{\delta} \end{bmatrix}, \tag{1}$$

where $\xi_{i,j}$ is the true (unknown) relative pose between node i and node j, $\epsilon_{i,j} \in \mathbb{R}^3$ is the *measurement noise*, $R_i \in \mathbb{R}^{2\times 2}$ is a planar rotation matrix of an angle θ_i, $\langle \cdot \rangle_{2\pi}$ is a modulo-(2π) operator that forces angular measurements in the manifold SO(2), and $\epsilon_{i,j}^{\Delta}$ and $\epsilon_{i,j}^{\delta}$ are the (possibly correlated) *Cartesian* and *orientation noise*. According to related literature, we assume $\epsilon_{i,j}$ to be zero mean Gaussian noise, i.e., $\epsilon_i^j \sim \mathcal{N}(\mathbf{0}_3, P_{i,j})$, being $P_{i,j}$ a 3 by 3 covariance matrix. $\xi_{i,j}$ describes the relative transformation that leads pose i to overlap with pose j. We can rewrite each measurement as $\bar{\xi}_{ij} = [(\bar{\Delta}_{i,j}^l)^\top \ \check{\delta}_{i,j}]^\top$, where $\bar{\Delta}_{i,j}^l \in \mathbb{R}^2$ denotes the *relative position* measurement, and $\check{\delta}_{i,j} \in \mathrm{SO}(2)$ denotes the *relative orientation* measurement. The superscript l in $\bar{\Delta}_{i,j}^l$ remarks that the relative position vector is expressed in a local frame. By convention, the pose of the first node is assumed to be the reference frame in which we want to estimate all the other poses, i.e., $x_0 = [0 \ \ 0 \ \ 0]^\top$.

In [3] the authors showed that the relative orientation measurements can be made linear by adding a suitable multiple of 2π, i.e, $\langle \theta_j - \theta_i \rangle_{2\pi} = \theta_j - \theta_i + 2k_{i,j}\pi$, with $\theta_i, \theta_j \in \mathbb{R}$, and $k_{i,j} \in \mathbb{Z}$ ($k_{i,j}$ is called *regularization term*). In this paper we assume that the regularization terms have been correctly computed, according to [3], and we call $\bar{\delta}_{i,j}$ the regularized measurements, i.e., we define $\bar{\delta}_{i,j} = \check{\delta}_{i,j} - 2k_{i,j}\pi$. Then the measurement model becomes:

$$\begin{bmatrix} \bar{\Delta}_{i,j}^l \\ \bar{\delta}_{i,j} \end{bmatrix} = \begin{bmatrix} R_i^\top (p_j - p_i) \\ \theta_j - \theta_i \end{bmatrix} + \begin{bmatrix} \epsilon_{i,j}^{\Delta} \\ \epsilon_{i,j}^{\delta} \end{bmatrix}. \tag{2}$$

Therefore, the maximum likelihood estimate of nodes configuration X attains the minimum of the following cost function (see [3] and the references therein):

$$f(X) = \sum_{(i,j)\in\mathcal{E}} \begin{bmatrix} R_i^\top (p_j - p_i) - \bar{\Delta}_{i,j}^l \\ \theta_j - \theta_i - \bar{\delta}_{i,j} \end{bmatrix}^\top \Omega_{i,j} \begin{bmatrix} R_i^\top (p_j - p_i) - \bar{\Delta}_{i,j}^l \\ \theta_j - \theta_i - \bar{\delta}_{i,j} \end{bmatrix} \tag{3}$$

where $\Omega_{i,j} = P_{i,j}^{-1}$ is the *information matrix* of measurement (i,j). Pose graph optimization reduces to find a global minimum of the weighted sum of the residual errors, i.e., $X^* = \arg\min f(X)$. In the following we use $\mathbf{I}_n$, $\mathbf{0}_n$, and $\otimes$ to denote an identity matrix, a vector of all zeros, and the Kronecker product.

3 A Linear Approximation

In this section we present the first contribution of this manuscript: a linear approximation for problem (3) that relaxes the assumption of previous work [3]. In [3] it was assumed that $P_{i,j}$ (and then $\Omega_{i,j}$) has the following structure:

$$P_{i,j} = \begin{bmatrix} P^{\Delta}_{i,j} & \mathbf{0}_2 \\ \mathbf{0}_2^{\top} & P^{\delta}_{i,j} \end{bmatrix}. \tag{4}$$

Roughly speaking, this essentially requires that the relative position and relative orientation measurements (that together give the relative pose measurement) are uncorrelated. In order to present the subsequent derivation we need to rewrite the cost function (3) in a more compact form. For this purpose we define the unknown *nodes' position* $p = [p_1^{\top} \; \dots \; p_n^{\top}]^{\top}$ and the unknown *nodes' orientation* $\theta = [\theta_1 \; \dots \; \theta_n]^{\top}$; therefore the to-be-computed network configuration may be written as $x = [p^{\top} \quad \theta^{\top}]^{\top}$ (note that we have excluded from x the pose x_0 that was assumed to be known). Then, we number the available measurements from 1 to m and we stack the relative position measurements in the vector $\bar{\Delta}^l = [(\bar{\Delta}^l_1)^{\top} \; (\bar{\Delta}^l_2)^{\top} \; \dots \; (\bar{\Delta}^l_m)^{\top}]^{\top}$, and the relative orientation measurements in the vector $\bar{\delta} = [\bar{\delta}_1 \; \bar{\delta}_2 \; \dots \; \bar{\delta}_m]^{\top}$. Accordingly, we reorganize the measurement information matrices $\Omega_{i,j}, (i,j) \in \mathcal{E}$, into a large matrix

$$\Omega \doteq \begin{bmatrix} \Omega_{\Delta} & \Omega_{\Delta\delta} \\ \Omega_{\delta\Delta} & \Omega_{\delta} \end{bmatrix}, \tag{5}$$

such that Ω is the information matrix of the vector of measurements $[(\bar{\Delta}^l)^{\top} \quad \bar{\delta}^{\top}]^{\top}$.

Then, the cost (3) can be written as:

$$f(x) = \begin{bmatrix} A_2^{\top} p - R\bar{\Delta}^l \\ A^{\top}\theta - \bar{\delta} \end{bmatrix}^{\top} \begin{bmatrix} R\Omega_{\Delta}R^{\top} & R\Omega_{\Delta\delta} \\ \Omega_{\delta\Delta}R^{\top} & \Omega_{\delta} \end{bmatrix} \begin{bmatrix} A_2^{\top} p - R\bar{\Delta}^l \\ A^{\top}\theta - \bar{\delta} \end{bmatrix} \tag{6}$$

where:

– A is the *reduced incidence matrix* of graph $\mathcal{G}$, see [3];
– $A_2 = A \otimes \mathbf{I}_2$ is an expanded version of A, see [1, 3];
– $R = R(\theta) \in \mathbb{R}^{2m,2m}$ is a block diagonal matrix, whose nonzero entries are in positions $(2k-1, 2k-1), (2k-1, 2k), (2k, 2k-1), (2k, 2k),\; k = 1, \dots, m$, such that, if the k-th measurement correspond to the relative pose between i and j, then the k-th diagonal block of R is a planar rotation matrix of an angle θ_i.

The residual errors in the cost function (6) are described by the following vector, whose entries represent the mismatch between the relative poses of a given configuration x and the actual relative measurements.

$$r(x) \doteq \begin{bmatrix} A_2^{\top} p - R\bar{\Delta}^l \\ A^{\top}\theta - \bar{\delta} \end{bmatrix} \tag{7}$$

Before presenting the proposed approach we anticipate the main intuition behind the algorithm. The cost function (6) is quite close to a quadratic function, since the last part of the residual errors in (7) is linear, and the overall cost function (6) becomes quadratic as soon as the rotation matrix R is known. Therefore, the basic idea is (i) to obtain an estimate of nodes orientations θ exploiting the linear part of the residual errors in (7), (ii) to use the estimated orientation to compute an estimate of R, and (iii) to solve the overall problem in the optimization variable x. This basic intuition is the same motivating [3], although here the derivation is made more complex by the presence of the correlation between position measurements $\bar{\varDelta}^l$ and orientation measurements $\bar{\delta}$.

We are now ready to present the proposed linear approximation for pose graph optimization, whose properties will be analyzed in Theorem 1.

Algorithm 1. *A linear approximation for the maximum likelihood pose graph configuration can be computed in three phases, given the relative measurements $\bar{\varDelta}^l$ and $\bar{\delta}$, the corresponding information matrix $\varOmega$, and the graph incidence matrices A and A_2:*

1. *Solve the following linear system in the unknown $z \doteq [(\varDelta^l)^\top \; \theta]^\top$:*
$$\varOmega_z \; z = b_z \tag{8}$$
with:
$$Z = \begin{bmatrix} \mathbf{I}_{2m} & \mathbf{0}_{2m,n} \\ \mathbf{0}_{m,2m} & A^\top \end{bmatrix}, \quad \text{and,} \quad \begin{array}{l} b_z = Z^\top \varOmega \left[(\bar{\varDelta}^l)^\top \; \bar{\delta}^\top\right]^\top \\ \varOmega_z = Z^\top \varOmega Z \end{array} \tag{9}$$
Call the solution of the linear system $\hat{z} \doteq [(\hat{\varDelta}^l)^\top \; \hat{\theta}]^\top$.
2. *Compute an estimate of the quantity $R\bar{\varDelta}^l$ in (7) from $\hat{z}$, preserving the correlation with the estimate $\hat{\theta}$:*
$$\hat{y} = T(\hat{z}) \doteq \begin{bmatrix} \hat{R} & \mathbf{0}_{2m\times n} \\ \mathbf{0}_{2m\times n}^\top & \mathbf{I}_n \end{bmatrix} \begin{bmatrix} \hat{\varDelta}^l \\ \hat{\theta} \end{bmatrix} = \begin{bmatrix} \tau_1(z) \\ \tau_2(z) \end{bmatrix}_{z=\hat{z}} \tag{10}$$
with $\hat{R} = R(\hat{\theta})$; compute the corresponding information matrix:
$$\varOmega_y = (\hat{T}\varOmega_z^{-1}\hat{T}^\top)^{-1} = (\hat{T}^{-1})^\top \varOmega_z (\hat{T}^{-1}), \tag{11}$$
where $\hat{T}$ is the Jacobian of the transformation $T(\cdot)$:
$$\hat{T} \doteq \begin{bmatrix} \frac{\partial \tau_1}{\partial \varDelta^l} & \frac{\partial \tau_1}{\partial \theta} \\ \frac{\partial \tau_2}{\partial \varDelta^l} & \frac{\partial \tau_2}{\partial \theta} \end{bmatrix} = \begin{bmatrix} \hat{R} & J \\ \mathbf{0}_{n\times 2m} & \mathbf{I}_n \end{bmatrix}. \tag{12}$$
3. *Solve the following linear system in the unknown $x = [p^\top \; \theta^\top]^\top$, given $\hat{y}$, see (10), and $\varOmega_y$, see (11):*
$$\varOmega_x \; x = b_x \tag{13}$$
with:
$$B = \begin{bmatrix} A_2^\top & \mathbf{0}_{2m\times n} \\ \mathbf{0}_{n\times 2n} & \mathbf{I}_n \end{bmatrix}, \quad \text{and,} \quad \begin{array}{l} b_x = B^\top \varOmega_y \hat{y} \\ \varOmega_x = B^\top \varOmega_y B \end{array} \tag{14}$$

The solution of the linear system (13) is the proposed linear approximation of the pose graph configuration: $x^* = [(p^*)^\top \;\; (\theta^*)^\top]^\top$. □

The effectiveness of the linear approximation computed using Algorithm 1 is assessed by the following result.

Theorem 1. *Given the inputs $\{\bar{\Delta}^l, \bar{\delta}, \Omega, A, A_2\}$, and assuming the information matrix Ω to be positive-definite, the following statements hold for the quantities computed in Algorithm 1:*

1. *$\Omega_z, \Omega_y, \Omega_x$ are full rank;*
2. *The combination of the three phases is equivalent to applying a Gauss-Newton step to the cost function (6), starting from the initial guess $\hat{x} = [\hat{p}^\top \ \ \hat{\theta}^\top]^\top$, with $\hat{\theta} = \left[A\Omega_\delta A^\top - A\Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}A^\top\right]^{-1} A\left(\Omega_\delta - \Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\right)\bar{\delta}$ and $\hat{p} = (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R}\Omega_\Delta\bar{\Delta}^l$.*

Proof. See Appendix. □

The first claim assures the uniqueness of the outcome of the proposed algorithm (no indetermination in the solution of the linear systems). The second claim assures that the proposed approximation improves over an initial guess $\hat{x}$, applying a Gauss-Newton step. It is worth noticing that $\hat{\theta}$ can be rewritten as: $\hat{\theta} = \left[AP_\delta^{-1}A^\top\right]^{-1} AP_\delta^{-1}\bar{\delta}$, where $P_\delta^{-1} = \left(\Omega_\delta - \Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\right)$ is the marginal information matrix of the orientation measurements $\bar{\delta}$. Therefore, the initial guess $\hat{\theta}$ is the BLUE (Best Linear Unbiased Estimator) for θ, given the sole orientation measurements, see [3]; moreover, $\hat{p}$ is the optimal estimate of nodes' positions, under the assumption that the actual orientations of the robot coincide with $\hat{\theta}$ [3]. The practical advantage of the algorithm is that $\hat{x}$ is already quite close to the optimal solution in practice, then the approximation is accurate in common problem instances. Moreover, the vector $\hat{p}$ is not computed explicitly by the approach, saving computation time.

4 Experimental Analysis

In this section we present the results of an extensive numerical evaluation on optimization-based SLAM. We compare the methodology proposed in this paper (Algorithm 1) with several state-of-the-art optimization approaches, namely a Gauss-Newton method [12], TORO [8], g^{2}o [11], and the linear approximation proposed in [3]. The Gauss-Newton approach is a standard implementation of the Gauss-Newton method for solving nonlinear least squares problems [13]. The halting condition for this approach is based on the norm of the local correction. Roughly speaking, if in two consecutive iterations the change in the configuration is smaller than a threshold the algorithm stops. In our tests the threshold on the norm of the local correction was set to 0.1. The results from TORO and g^{2}o are obtained using the C++ code available online [15]. For the tests we used default settings for both approaches. Our implementations of the linear approximation [3] and of Algorithm 1 are available online [4]. Also the implementation of the Gauss-Newton approach we used in the test campaign was released online [4].

The compared approaches are tested on publicly available datasets: Freiburg Indoor Building 079 (FR079), MIT CSAIL Building (CSAIL), Intel Research Lab (INTEL), Manhattan World M3500 (M3500), Manhattan World M10000 (M10000). The relative pose measurements of the datasets {FR079, CSAIL, M3500, M10000} are available online [10], while the measurements of the dataset INTEL were obtained through a scan matching procedure, from the raw sensor data, available at [10]. The INTEL dataset is the same studied in [3]. The relations available online [10] only describe the relative pose measurements, while we are interested to test the behavior of the approaches for different measurement covariance matrices. In particular, for each dataset we consider three variants, each one corresponding to a different choice of the covariance matrix. The first variant (e.g., FR079-I) uses identity matrices as measurement covariances, i.e., the noise of the relative pose measurement between node i and node j is $\epsilon_i^j \sim \mathcal{N}(\mathbf{0}_3, \mathbf{I}_3)$. The second variant (e.g., FR079-P_s) uses a structured covariance matrix, as in eq. (4). The third variant (e.g., FR079-P_f) uses full covariance matrices obtained as follows. According to the standard *odometry model* [18], we parametrize the relative pose between node i and j as a rotation γ_r^1, followed by a translation γ_t^1, and by a second rotation γ_r^2, see Section 5.4 in [18]. Then, fixing the uncertainty in the parameters, we can define the corresponding covariance matrix for the relative pose measurement. For our numerical experiments we set the standard deviations of γ_r^1, γ_t^1, γ_r^2, to 0.05 rad, 0.05 m, and 0.01 rad, respectively. For the sake of repeatability and for stimulating further comparisons with related approaches the datasets considered in this paper were released online [4].

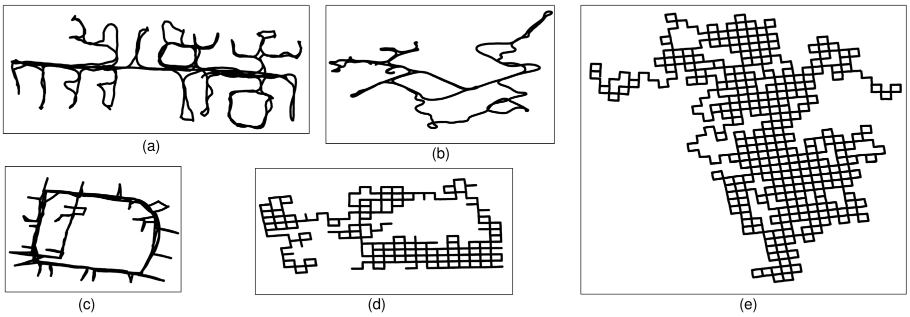


Fig. 1. Estimated trajectory for each of the considered datasets: (a) FR079, (b) CSAIL, (c) INTEL, (d) M3500, (e) M10000

Accuracy. In Figure 1 we show some qualitative results for the proposed approach on the considered datasets (for simplicity we only show the variant with the identity matrix as measurement covariance). For a quantitative evaluation of the accuracy of the approaches, we recorded the optimal value of the cost function (6), attained by each of the compared techniques. Since each approach is required to minimize the cost (6), the best solution is the one attaining the smallest value of the objective function. The results for the compared approaches and for each dataset are reported in Table 1. The Gauss-Newton method and $\mathtt{g^2o}$ attain the same solution (which is also the smallest observed cost function) in most cases. This comes at no surprise since they both solve the original optimization problem without any approximation involved. Only in two scenarios $\mathtt{g^2o}$ performs worse than the Gauss-Newton method: in the INTEL dataset (in which the initial guess is particularly bad), and in the M10000 dataset (which contains a large number of nodes). The difference is explained as follows: $\mathtt{g^2o}$ applies a fixed number of iterations, then in the two mentioned cases, the iterations are not sufficient to reach the optimal value. TORO shows the worst performance in all tests, due to the involved approximations, see [8]. The linear approximations of [3] (structured covariance) and the one proposed in this paper (unstructured covariance) produce intermediate results. They are practically optimal in the first variants (I) of each scenario (using identity matrices as covariances), and they are close to the Gauss-Newton solution in the second variants (P_s). Only in the scenario INTEL the cost function was remarkably larger than the Gauss-Newton approach, although being lower than TORO. It is worth noticing that in the first and in the second variants of each dataset, the linear approximation of [3] and the approach proposed in this paper attain the same objective. This is due to the fact that the approach proposed in this paper reduces to the linear approximation of [3] when the measurements covariance matrices of the input data are structured (as it happens in the variants I and P_s). The two approaches, instead, differ when measurement covariance is unstructured, as in the P_f variant reported in Table 1. In this case the linear approximation [3] simply neglects the correlation terms while the proposed approach can deal with the full covariance case. The numerical results show that in the third variant (P_f) of the datasets {INTEL, M10000} the proposed approximation remarkably improves the attained objective value. In the remaining datasets, the difference between the linear approximations is small. We conclude this paragraph by noticing that the proposed approximation and the approach in [3] are more accurate than $\mathtt{g^2o}$ on the large-scale dataset M10000.

Efficiency. The efficiency of the compared approaches is connected with the computational effort that each method requires for producing the estimate of node configuration. The average CPU time required by the compared approaches for each of the dataset is reported in Table 1. The reported statistics are averaged over 10 runs. The tests are conducted on a standard laptop, with an Intel Core i7 3.4 GHZ and 8 GB of RAM. The CPU times required by TORO and $\mathtt{g^2o}$ are the ones returned by the code available online. The Gauss-Newton method, the linear approximation [3], and the approach proposed in this paper are implemented in C++ and use the *CSparse* library [5]. Moreover, for the two linear approximations, the CPU time includes the computation of the *regularization terms* [3].

From the table it is possible to see that the Gauss-Newton method, although being very accurate, quickly becomes unsustainable for large datasets. TORO is slightly faster, but still remains not competitive w.r.t. the other techniques. $\mathtt{g^2o}$ is highly optimized and allows a remarkable speed-up w.r.t. the Gauss-Newton method. Table 1 highlights that the linear approximation [3] and the linear

Table 1. Objective function values and average computation time (in seconds) for the compared approaches

			Linear Approximation (structured covariance)	Linear Approximation (unstructured covariance)	Gauss-Newton	TORO	g^{2}o
FR079	I	Objective	7.20E-02	7.20E-02	7.20E-02	8.60E-02	7.19E-02
		Time (s)	5.80E-03	8.15E-03	1.99E-01	3.19E-01	1.05E-02
	Ps	Objective	3.94E+01	3.94E+01	3.88E+01	4.74E+02	3.89E+01
		Time (s)	5.76E-03	7.87E-03	2.00E-01	3.39E-01	1.07E-02
	Pf	Objective	2.76E+02	2.90E+02	1.47E+02	8.99E+03	1.47E+02
		Time (s)	5.81E-03	8.16E-03	2.00E-01	3.04E-01	1.06E-02
CSAIL	I	Objective	1.07E-01	1.07E-01	1.07E-01	1.18E-01	1.07E-01
		Time (s)	5.72E-03	7.55E-03	2.65E-01	2.89E-01	1.01E-02
	Ps	Objective	4.06E+01	4.06E+01	4.06E+01	2.41E+03	4.06E+01
		Time (s)	5.53E-03	7.46E-03	2.01E-01	2.90E-01	1.01E-02
	Pf	Objective	2.45E+02	2.33E+02	1.57E+02	4.57E+04	1.57E+02
		Time (s)	5.59E-03	7.50E-03	2.66E-01	2.88E-01	1.03E-02
INTEL	I	Objective	8.07E-01	8.07E-01	7.89E-01	1.17	7.89E-01
		Time (s)	7.10E-03	9.49E-03	5.87E-01	4.15E-01	1.32E-02
	Ps	Objective	1.45E+04	1.45E+04	2.15E+02	1.03E+05	2.15E+02
		Time (s)	7.01E-03	9.49E-03	4.90E-01	3.89E-01	1.31E-02
	Pf	Objective	1.51E+06	1.07E+05	3.95E+02	2.53E+07	1.08E+03
		Time (s)	6.98E-03	9.47E-03	5.91E-01	4.01E-01	1.31E-02
M3500	I	Objective	3.03	3.03	3.02	5.42	3.02
		Time (s)	3.26E-02	4.04E-02	5.81	1.57	7.07E-02
	Ps	Objective	3.73E+03	3.73E+03	3.55E+03	2.18E+06	3.55E+03
		Time (s)	3.25E-02	4.03E-02	4.84	1.61	7.06E-02
	Pf	Objective	1.15E+04	6.81E+03	2.09E+03	5.78E+08	2.09E+03
		Time (s)	3.26E-02	4.05E-02	5.82	1.61	7.14E-02
M10000	I	Objective	3.03E+02	3.03E+02	3.03E+02	3.29E+02	3.03E+02
		Time (s)	3.55E-01	4.86E-01	2.21E+02	1.73E+01	6.93E-01
	Ps	Objective	1.99E+05	1.99E+05	1.98E+05	7.65E+06	2.28E+05
		Time (s)	3.57E-01	4.89E-01	2.21E+02	1.83E+01	6.96E-01
	Pf	Objective	9.00E+05	9.61E+05	6.79E+05	2.07E+08	1.90E+07
		Time (s)	3.55E-01	4.86E-01	4.11E+02	1.77E+01	6.91E-01

approximation proposed in this paper outperform all state-of-the-art techniques in terms of computational time. In particular, they assure a reduction of the computational time of $30-50\%$ w.r.t. to g^{2}o and improve the computational time of orders of magnitude w.r.t. the other state-of-the-art techniques. We conclude this section observing that the proposed approach is able to compute an estimate of the configuration of a pose graph with 10000 nodes and 64311 edges in less than 0.5 seconds.

Finally a video showing an experimental test in which the edges of the graph are obtained online using a scan matching algorithm is available online [4].

5 Conclusion

The contribution of this article is twofold: a linear approximation for optimization-based SLAM and an extensive evaluation of the performance of state-of-the-art approaches on benchmarking datasets. The first contribution includes the presentation of an algorithm for estimating an approximation of pose graph configuration. The algorithm is an extended version of the approach proposed in [3] and can deal with the case of generic unstructured measurement covariance. The second contribution includes an experimental analysis of pose graph optimization approaches in terms of accuracy, efficiency, and scalability. As a results we demonstrate that the accuracy of the proposed linear approximation is comparable with the one of state-of-the-art techniques, although it requires a fraction of their computational effort.

Appendix

In this appendix we report the proof of Theorem 1. We omit for brevity the proof of the first claim, which is a straightforward extension of the results reported in [3]. We instead prove the second claim by direct calculation. We need to demonstrate that the outcome of the proposed algorithm is equivalent to a Gauss-Newton step from the initial guess $\hat{x} = [\hat{p}^\top \quad \hat{\theta}^\top]^\top$. The structure of the proof is the following: (i) we compute by direct calculation the solution of the proposed approach x^*, (ii) we compute the estimate x^{GN}, obtained from a Gauss-Newton step with initial guess $\hat{x}$, (iii) we show that $x^* = x^{GN}$. We start by computing $\Omega_z = Z^\top \Omega Z$:

$$\Omega_z = \begin{bmatrix} \Omega_\Delta & \Omega_{\Delta\delta} A^\top \\ A\Omega_{\delta\Delta} & A\Omega_\delta A^\top \end{bmatrix}$$

We can use blockwise inversion rule to compute the inverse of Ω_z. Notice that the explicit inverse needs not be computed in practice, since computationally effective methods can be used to solve the sparse linear system ([8]). For the sake of the proof, we instead evaluate:

$$P_z \doteq \Omega_z^{-1} = \begin{bmatrix} P_{z_{11}} & P_{z_{12}} \\ P_{z_{21}} & P_{z_{22}} \end{bmatrix}$$

with:

$$P_{z_{22}} = \left[A\Omega_\delta A^\top - A\Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}A^\top\right]^{-1},$$
$$P_{z_{11}} = \Omega_\Delta^{-1} + \Omega_\Delta^{-1}\Omega_{\Delta\delta}A^\top P_{z_{22}} A\Omega_{\delta\Delta}\Omega_\Delta^{-1}, \quad P_{z_{12}} = P_{z_{21}}^\top = -\Omega_\Delta^{-1}\Omega_{\Delta\delta}A^\top P_{z_{22}}.$$

Then we can compute $\hat{z} = \Omega_z^{-1} b_z$:

$$\hat{z} = \begin{bmatrix} \bar{\Delta}^l + \Omega_\Delta^{-1}\Omega_{\Delta\delta}\left[\bar{\delta} - A^\top P_{z_{22}} A\left(\Omega_\delta - \Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\right)\bar{\delta}\right] \\ P_{z_{22}} A\left(\Omega_\delta - \Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\right)\bar{\delta} \end{bmatrix}$$

If we call $\hat{\theta} = P_{z_{22}} A \left(\Omega_\delta - \Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\right)\bar{\delta}$, the vector $\hat{z}$ can be written in compact form as:

$$\hat{z} = \begin{bmatrix} \bar{\Delta}^l + \Omega_\Delta^{-1}\Omega_{\Delta\delta}\left(\bar{\delta} - A^\top\hat{\theta}\right) \\ \hat{\theta} \end{bmatrix} \tag{15}$$

We can then compute $\hat{y}$ and Ω_y according to (10) and (11):

$$\hat{y} = \begin{bmatrix} \hat{R}\bar{\Delta}^l + \hat{R}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\left(\bar{\delta} - A^\top\hat{\theta}\right) \\ \hat{\theta} \end{bmatrix}, \quad \Omega_y = (\hat{T}^{-1})^\top \Omega_z (\hat{T}^{-1}) \doteq \begin{bmatrix} \Omega_{y_{11}} & \Omega_{y_{12}} \\ \Omega_{y_{12}} & \Omega_{y_{22}} \end{bmatrix}$$

with:

$$\begin{aligned}
\hat{T}^{-1} &= \begin{bmatrix} \hat{R} & J \\ \mathbf{0}_{n\times 2m} & \mathbf{I}_n \end{bmatrix}^{-1} = \begin{bmatrix} \hat{R}^\top & -\hat{R}^\top J \\ \mathbf{0}_{n\times 2m} & \mathbf{I}_n \end{bmatrix}, \quad \Omega_{y_{11}} = \hat{R}\Omega_\Delta\hat{R}^\top \\
\Omega_{y_{12}} &= \Omega_{y_{12}}^\top = -\hat{R}\Omega_\Delta\hat{R}^\top J + \hat{R}\Omega_{\Delta\delta}A^\top \\
\Omega_{y_{22}} &= A\Omega_\delta A^\top + J^\top\hat{R}\Omega_\Delta\hat{R}^\top J - A\Omega_{\delta\Delta}\hat{R}^\top J - J^\top\hat{R}\Omega_{\Delta\delta}A^\top.
\end{aligned}$$

Now we compute Ω_x, its inverse Ω_x^{-1}, and b_x, from which it is easy to derive x^*. According to (3), Ω_x can be written explicitly as:

$$\Omega_x \doteq B^\top\Omega_y B = \begin{bmatrix} \Omega_{x_{11}} & \Omega_{x_{12}} \\ \Omega_{x_{21}} & \Omega_{x_{22}} \end{bmatrix}$$

with:

$$\begin{aligned}
\Omega_{x_{11}} &= A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top \\
\Omega_{x_{12}} &= \Omega_{x_{21}}^\top = -A_2\hat{R}\Omega_\Delta\hat{R}^\top J + A_2\hat{R}\Omega_{\Delta\delta}A^\top \\
\Omega_{x_{22}} &= A\Omega_\delta A^\top + J^\top\hat{R}\Omega_\Delta\hat{R}^\top J - A\Omega_{\delta\Delta}\hat{R}^\top J - J^\top\hat{R}\Omega_{\Delta\delta}A^\top.
\end{aligned}$$

After long and tedious calculations we obtain Ω_x^{-1} using standard blockwise-inversion rules:

$$P_x \doteq \Omega_x^{-1} = \begin{bmatrix} P_{x_{11}} & P_{x_{12}} \\ P_{x_{21}} & P_{x_{22}} \end{bmatrix}$$

with:

$$\begin{aligned}
&P_{x_{22}} = \left[A\Omega_\delta A^\top + J^\top\hat{R}\Omega_\Delta\hat{R}^\top J - J^\top\hat{R}\Omega_{\Delta\delta}A^\top - A\Omega_{\delta\Delta}\hat{R}^\top J + \right.\\
&\left.-(A\Omega_{\delta\Delta}\hat{R}^\top A_2^\top - J^\top\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)(A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}(A_2\hat{R}\Omega_{\Delta\delta}A^\top - A_2\hat{R}\Omega_\Delta\hat{R}^\top J)\right]^{-1} \\
&P_{x_{12}} = P_{x_{21}}^\top = -(A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}(A_2\hat{R}\Omega_{\Delta\delta}A^\top - A_2\hat{R}\Omega_\Delta\hat{R}^\top J)P_{x_{22}} \\
&P_{x_{11}} = (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1} + (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}(A_2\hat{R}\Omega_{\Delta\delta}A^\top - A_2\hat{R}\Omega_\Delta\hat{R}^\top J)\times \\
&P_{x_{22}}(A\Omega_{\delta\Delta}\hat{R}^\top A_2^\top - J^\top\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)(A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}.
\end{aligned} \tag{16}$$

From matrix-vector multiplication we also compute $b_x \doteq B^\top\Omega_y\hat{y} = [b_{x_1}^\top \;\; b_{x_2}^\top]^\top$, with:

$$\begin{aligned}
b_{x_1} &= A_2\hat{R}\left(\Omega_\Delta\bar{\Delta}^l - \Omega_\Delta\hat{R}^\top J\hat{\theta} + \Omega_{\Delta\delta}\bar{\delta}\right) \\
b_{x_2} &= A\Omega_\delta A^\top\hat{\theta} + J^\top\hat{R}\Omega_\Delta\hat{R}^\top J\hat{\theta} - A\Omega_{\delta\Delta}\hat{R}^\top J\hat{\theta} + A\Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\hat{\delta} + \\
&\quad -J^\top\hat{R}\Omega_{\Delta\delta}\bar{\delta} - A\Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}A^\top\hat{\theta} + A\Omega_{\delta\Delta}\bar{\Delta}^l - J^\top\hat{R}\Omega_\Delta\bar{\Delta}^l.
\end{aligned}$$

Finally we can calculate $x^* = P_x b_x \doteq [(p^*)^\top \;\; (\theta^*)^\top]^\top$, with:

$$\begin{aligned}\theta^* &= \hat{\theta} + P_{x_{22}}\big[(J^\top \hat{R}\Omega_\Delta \hat{R}^\top A_2^\top)(A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R} - (A\Omega_{\delta\Delta}\hat{R}^\top A_2^\top)\times \\ &\quad (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R} - J^\top\hat{R} + A\Omega_{\delta\Delta}\Omega_\Delta^{-1}\big]\big[\Omega_\Delta\bar{\Delta}^l + \Omega_{\Delta\delta}(\bar{\delta} - A^\top\hat{\theta})\big], \\ p^* &= \hat{p} + (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R}\left[\Omega_{\Delta\delta}(\bar{\delta} - A^\top\theta^*) + \Omega_\Delta\hat{R}^\top J(\theta^* - \hat{\theta})\right],\end{aligned}$$

where $\hat{\theta} = P_{z_{22}}A\left(\Omega_\delta - \Omega_{\delta\Delta}\Omega_\Delta^{-1}\Omega_{\Delta\delta}\right)\bar{\delta}$ and $\hat{p} = (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R}\Omega_\Delta\bar{\Delta}^l$. After obtaining x^* we have to compute the outcome of the Gauss-Newton step from $\hat{x} \doteq [\hat{p}^\top \;\; \hat{\theta}^\top]^\top$, since we claim that the result is the same. According to the standard Gauss-Newton approach, a single step from the guess $\hat{x}$ produces the estimate $x^{GN} = \hat{x} + \tilde{x}$, where $\tilde{x} \doteq [\tilde{p}^\top \;\; \tilde{\theta}^\top]^\top$ is the minimum of the quadratic cost obtained by linearizing the residual errors in (6) around $\hat{x}$. Let us start by linearizing the residual errors in the cost function around $\hat{x}$:

$$r(\hat{x}+\tilde{x}) \approx \begin{bmatrix} A_2^\top\hat{p} + A_2^\top\tilde{p} - \hat{R}\bar{\Delta}^l - J\tilde{\theta} \\ A^\top\hat{\theta} + A^\top\tilde{\theta} - \bar{\delta}\end{bmatrix} \doteq \tilde{r}(\tilde{x}) \tag{17}$$

Considering the linearized residue and evaluating the covariance matrix in $\theta = \hat{\theta}$ the cost function (6) becomes quadratic:

$$f(\tilde{x}) \approx \tilde{r}(\tilde{x})^\top \begin{bmatrix}\hat{R}\Omega_\Delta\hat{R}^\top & \hat{R}\Omega_{\Delta\delta} \\ \Omega_{\delta\Delta}\hat{R}^\top & \Omega_\delta\end{bmatrix}\tilde{r}(\tilde{x}) \doteq \tilde{f}(\tilde{x}) \tag{18}$$

The global minimum of the previous cost function can be computed by taking the gradient of the $\tilde{f}$ with respect to $\tilde{x} = [\tilde{p}^\top \;\; \tilde{\theta}^\top]^\top$ and imposing it to be zero. Let us compute the gradient with respect to the variable $\tilde{p}$ and $\tilde{\theta}$:

$$\begin{aligned}\nabla_{\tilde{p}}(\tilde{f}) &= 2A_2\hat{R}\Omega_\Delta\hat{R}^\top(A_2^\top\hat{p} - \hat{R}\bar{\Delta}^l + A_2^\top\tilde{p} - J\tilde{\theta}) + 2A_2\hat{R}\Omega_{\Delta\delta}(A^\top\hat{\theta} + A^\top\tilde{\theta} - \bar{\delta}) \\ \nabla_{\tilde{\theta}}(\tilde{f}) &= -2J^\top\hat{R}\Omega_\Delta\hat{R}^\top(A_2^\top\hat{p} - \hat{R}\bar{\Delta}^l + A_2^\top\tilde{p} - J\tilde{\theta}) - 2J^\top\hat{R}\Omega_{\Delta\delta}(A^\top\hat{\theta} + A^\top\tilde{\theta} - \bar{\delta}) + \\ &\quad +2A\Omega_{\delta\Delta}\hat{R}^\top(A_2^\top\hat{p} - \hat{R}\bar{\Delta}^l + A_2^\top\tilde{p} - J\tilde{\theta}) + 2A\Omega_\delta(A^\top\hat{\theta} + A^\top\tilde{\theta} - \bar{\delta})\end{aligned}$$

The global minimum has to satisfy the following linear system of equations:

$$\begin{cases}\nabla_{\tilde{p}}(\tilde{f}) = \mathbf{0}_{2n} \\ \nabla_{\tilde{\theta}}(\tilde{f}) \;= \mathbf{0}_n\end{cases} \tag{19}$$

It is possible to explicit the unknown $\tilde{p}$ from the first equation in (19), writing it in function of $\tilde{\theta}$:

$$\tilde{p} = (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R}\left[\Omega_{\Delta\delta}(\bar{\delta} - A^\top\hat{\theta} - A^\top\tilde{\theta}) + \Omega_\Delta\hat{R}^\top J\tilde{\theta}\right].$$

Now we can substitute $\tilde{p}$ in the second equation of (19), obtaining a linear equation containing only $\tilde{\theta}$. Solving such equation we obtain:

$$\begin{aligned}\tilde{\theta} = P_{x_{22}}\big[&(J^\top\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)(A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R} + \\ &-(A\Omega_{\delta\Delta}\hat{R}^\top A_2^\top)(A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R} + \\ &-J^\top\hat{R} + A\Omega_{\delta\Delta}\Omega_\Delta^{-1}\big]\big[\Omega_\Delta\bar{\Delta}^l + \Omega_{\Delta\delta}(\bar{\delta} - A^\top\hat{\theta})\big],\end{aligned}$$

where $P_{x_{22}}$ is defined as in (16). We can finally compute x^{GN}:

$$x^{\text{GN}} = \hat{x} + \tilde{x} = \begin{bmatrix}\hat{p} + \tilde{p} \\ \hat{\theta} + \tilde{\theta}\end{bmatrix} = \begin{bmatrix}p^{\text{GN}} \\ \theta^{\text{GN}}\end{bmatrix}$$

with:

$$\begin{aligned}\theta^{\mathrm{GN}} &= \hat{\theta} + P_{x_{22}}\big[(J^\top \hat{R}\Omega_\Delta \hat{R}^\top A_2^\top)(A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R} - (A\Omega_{\delta\Delta}\hat{R}^\top A_2^\top)\times \\ &\quad (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R} - J^\top\hat{R} + A\Omega_{\delta\Delta}\Omega_\Delta^{-1}\big]\big[\Omega_\Delta\bar{\Delta}^l + \Omega_{\Delta\delta}(\bar{\delta} - A^\top\hat{\theta})\big], \\ p^{\mathrm{GN}} &= \hat{p} + (A_2\hat{R}\Omega_\Delta\hat{R}^\top A_2^\top)^{-1}A_2\hat{R}\left[\Omega_{\Delta\delta}(\bar{\delta} - A^\top\theta^*) + \Omega_\Delta\hat{R}^\top J(\theta^* - \hat{\theta})\right].\end{aligned}$$

By inspection, it is easy to verify that $p^{\mathrm{GN}} = p^*$ and $\theta^{\mathrm{GN}} = \theta^*$, hence proving the second claim of Theorem 1.

References

[1] Barooah, P., Hespanha, J.P.: Estimation on graphs from relative measurements. IEEE Control Systems Magazine 27(4), 57–74 (2007)
[2] Carlone, L., Aragues, R., Castellanos, J.A., Bona, B.: A first-order solution to simultaneous localization and mapping with graphical models. In: Proc. of the IEEE lnternational Conf. on Robotics and Automation (2011)
[3] Carlone, L., Aragues, R., Castellanos, J.A., Bona, B.: A linear approximation for graph-based simultaneous localization and mapping. In: Proc. of Robotics: Science and Systems (2011)
[4] Carlone, L., Rosa, S., Yin, J.: Robotics research group: Resources – graph optimization with unstructured covariance (2012), www.polito.it/LabRob
[5] Davis, T.A.: Direct Methods for Sparse Linear Systems. Fundamentals of Algorithms, vol. 2. Society for Industrial and Applied Mathematics, Philadelphia (2006) ISBN 0898716136
[6] Dellaert, F., Carlson, J., Ila, V., Ni, K., Thorpe, C.: Subgraph-preconditioned conjugate gradients for large scale SLAM. In: Proc. of the IEEE-RSJ Int. Conf. on Intelligent Robots and Systems (2010)
[7] Frese, U., Larsson, P., Duckett, T.: A multilevel relaxation algorithm for simultaneous localization and mapping. IEEE Trans. on Robotics 21(2), 196–207 (2005)
[8] Grisetti, G., Stachniss, C., Burgard, W.: Non-linear constraint network optimization for efficient map learning. IEEE Trans. on Intelligent Transportation Systems 10(3), 428–439 (2009)
[9] Konolige, K.: Large-scale map-making. In: Proc. of the AAAI National Conf. on Artificial Intelligence (2004)
[10] Kümmerle, R., Steder, B., Dornhege, C., Ruhnke, M., Grisetti, G., Stachniss, C., Kleiner, A.: Slam benchmarking webpage (2009), http://ais.informatik.uni-freiburg.de/slamevaluation
[11] Kummerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W.: G2o: A general framework for graph optimization. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 3607–3613 (May 2011), doi:10.1109/ICRA.2011.5979949
[12] Lu, F., Milios, E.: Globally consistent range scan alignment for environment mapping. Autonomous Robots 4, 333–349 (1997)
[13] Nocedal, J., Wright, S.J.: Numerical Optimization. Springer (2006)
[14] Olson, E., Leonard, J.J., Teller, S.: Fast iterative optimization of pose graphs with poor initial estimates. In: Proc. of the IEEE Int. Conf. on Robotics and Automation, pp. 2262–2269 (2006)

[15] Stachniss, C., Frese, U., Grisetti, G.: Open SLAM webpage (2007), http://openslam.org/
[16] Sunderhauf, N., Protzel, N.: Towards a robust back-end for pose graph slam. In: Proc. of IEEE International Conference on Robotics and Automation, ICRA (2012)
[17] Thrun, S., Montemerlo, M.: The GraphSLAM algorithm with applications to large-scale mapping of urban structures. Int. J. Robot. Res. 25, 403–429 (2006)
[18] Thrun, S., Burgard, W., Fox, D.: Probabilistic robotics. MIT Press (2005)

Mobile Robot SLAM Interacting with Networked Small Intelligent Sensors Distributed in Indoor Environments

Fumitaka Hashikawa[1,*], Kazuyuki Morioka[1], and Noriaki Ando[2]

[1] Meiji University, Kanagawa, Japan
hashikawa@mork.mind.meiji.ac.jp
[2] AIST, Ibaraki, Japan
n-ando@aist.go.jp

Abstract. SLAM is a method of map building and self-position estimation for robot navigation. However, map building error is especially appeared in loop closing points when the mobile robot moves around loop trajectories. In this study, more accurate mobile robot SLAM is considered in intelligent space [1] where many sensors are distributed.

An intelligent space is constructed with various types of distributed sensors including networked laser range sensors. Laser sensor on a mobile robot and environment sensors share sensor information each other in intelligent space. Maps and self-positions of the mobile robot are estimated using geometrical relationships between the mobile robot and sensors in intelligent space. However, geometrical calibration of distributed sensors under the unified world coordinates is required for construction of the intelligent space. When many sensors are distributed in wide area, it generally becomes complicated tasks to calibrate all sensors. In order to solve these problems, we consider extend SLAM algorithm. In this study, a new method of SLAM, which uses distributed sensors fixed in the intelligent space, is introduced. This method aims to achieve precision SLAM and position estimation of networked laser range sensors in the intelligent space.

Keywords: Mobile Robot, SLAM, Intelligent Space.

1 Introduction

Moving functions are indispensable in order for a mobile robot to coexist with people in human living environments. And exact motion control is required. A wheel type mobile robot can estimate self-position by rotation of a wheel. However, position estimation errors are accumulated according to friction with a road surface, etc. In addition to this odometry, a laser range sensor carried in the mobile robot should be used in order to solve this estimation error problem by scanning the external object position. Position of the mobile robot is estimated by matching between the maps and the scan data. This method is called SLAM [2]. SLAM is one of the methods for building maps from sensor data observed mobile robot. In SLAM algorithm, the maps generated in the past are compared with a current scan data and an exact current position is estimated. However, map building errors are especially appeared around loop closing points when

[*] Corresponding author.

the mobile robot moves along loop trajectories [2]. In this study, more accurate mobile robot SLAM is considered in intelligent space where many sensors are distributed.

Recently the intelligent spaces have been developed as one of the environments where people and robots live together [1][3]. The intelligent spaces including networked laser range sensors, cameras and the other sensing device. In this study, the mobile robot shares sensor information with the intelligent space. Maps and the mobile robot's position are estimated using geometrical relationships between the mobile robot and distributed sensors in intelligent space. In order to build such intelligent spaces, it is required to know positions of distributed sensors in unified world coordinate system of intelligent space [4]. Then, calibration of distributed sensors must be achieved for development of intelligent spaces. However, it is generally complicated works. Especially, many distributed laser range sensors are widely used in intelligent spaces, geometrical calibration of many laser range sensors in unified world coordinate system is complicated. Easy calibration method is required for the intelligent space. This study also focuses on easy calibration of networked laser range sensors not only SLAM of mobile robots.

In order to solve above-mentioned two problems: accurate SLAM and calibration of distributed intelligent sensors, we consider extend FastSLAM [5]. The new method of SLAM, which uses distributed laser range sensors fixed in an environment as intelligent space is introduced as interactive SLAM. This method shares environment data between intelligent sensors and SLAM of a mobile robot. When mobile robot in a monitoring area of a distributed sensor, maps generated by distributed intelligent sensors and scanned data acquired by a mobile robot are shared for interactive map matching. Geometrical relationships between the mobile robot and the intelligent space are estimated based on the matching results. Then, positions of the distributed laser range sensors are estimated according to the matching results. Also, the map built by the mobile robot is improved accurately according to position estimation results of the distributed laser range sensors. This study aims to improve accuracy of SLAM and estimate the distributed sensor's position of intelligent space based on interactively communicating each other.

2 Interactive SLAM between Mobile Robot and Intelligent Space

2.1 Outline

In this study, SLAM process of mobile robot and distributed intelligent sensors (Intelligent Space) communicate each other.

In SLAM process of mobile robot, FastSLAM algorithm is mainly used[2]. FastSLAM based on particle filter can achieve SLAM with high speed and high accuracy. Especially, grid-based Fast SLAM is more promising. This method includes building a probabilistic grid map[2] based on environment data obtained by a laser range sensor on the mobile robot. Fig.1 shows the outline of this normal SLAM. Then, the grid map and new scanned results are compared every sampling time, and the grid map is updated. Mobile robot's self-position is also estimated based on this map [6][7].

In distributed intelligent sensors (Intelligent Space) process, grid map are constructed by the same map building method with the mobile robot process. In addition, estimation method of each distributed intelligent sensor's position is also similar with position estimation of the mobile robot based on the particle filter. In other words, that means position calibration of the distributed intelligent sensors is added in SLAM process of the mobile robot. We call this SLAM “interactive SLAM”. Fig.2 shows outline of this interactive SLAM.

In the initial state of this interactive SLAM, a mobile robot's initial position is set at the origin of world coordinate. And the initial position of each distributed intelligent sensor is unknown. That means all the state variables are set as zero. There is no map information on an initial state of the robot and sensors.This study considers extending FastSLAM of mobile robot to an interactive SLAM with the intelligent space. The interactive SLAM uses grid maps built by the distributed laser range sensors of the intelligent space in addition to grid maps built by mobile robot. When the mobile robot is in the distributed sensor’s monitoring area, grid maps built by the distributed

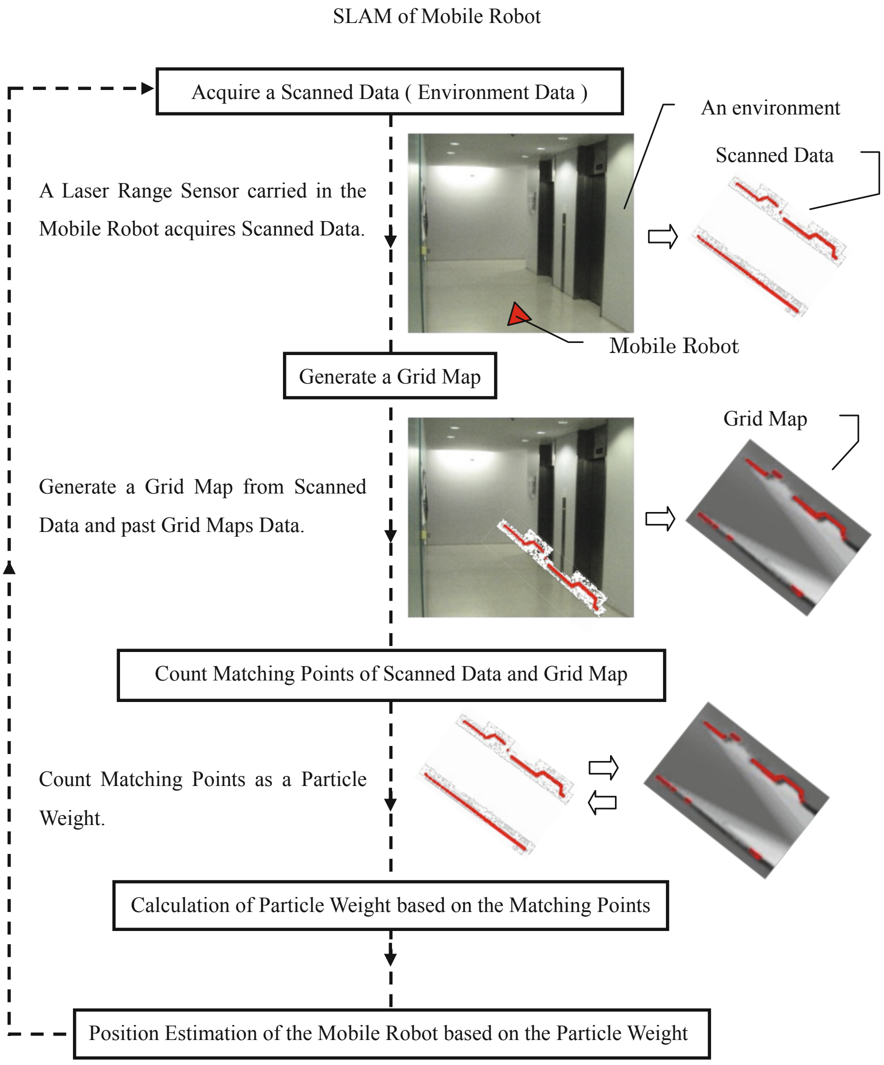

SLAM of Mobile Robot
Acquire a Scanned Data (Environment Data)
An environment
Scanned Data
A Laser Range Sensor carried in the Mobile Robot acquires Scanned Data.
Mobile Robot
Generate a Grid Map
Grid Map
Generate a Grid Map from Scanned Data and past Grid Maps Data.
Count Matching Points of Scanned Data and Grid Map
Count Matching Points as a Particle Weight.
Calculation of Particle Weight based on the Matching Points
Position Estimation of the Mobile Robot based on the Particle Weight

Fig. 1. SLAM process outline

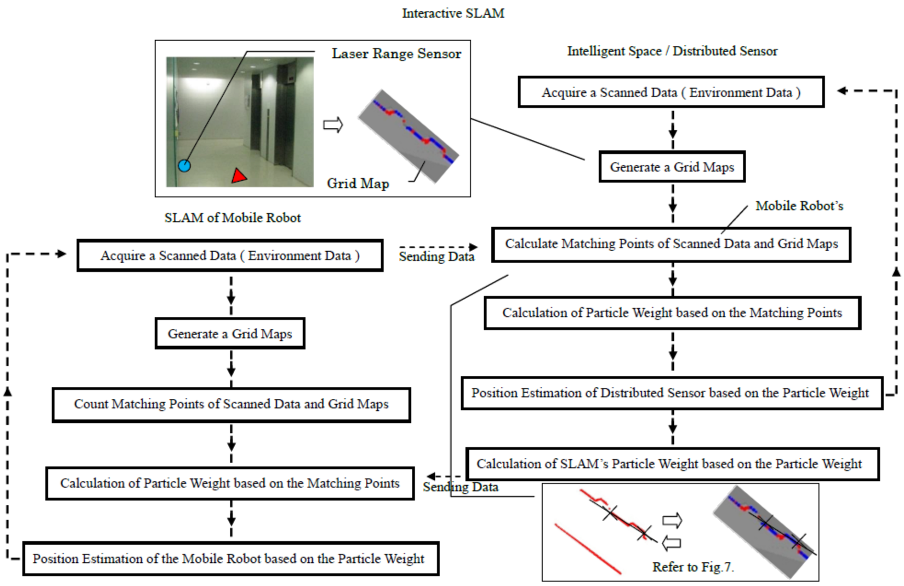

Interactive SLAM
Laser Range Sensor
Grid Map
Intelligent Space / Distributed Sensor
Acquire a Scanned Data (Environment Data)
Generate a Grid Maps
Mobile Robot's
Calculate Matching Points of Scanned Data and Grid Maps
Calculation of Particle Weight based on the Matching Points
Position Estimation of Distributed Sensor based on the Particle Weight
Calculation of SLAM's Particle Weight based on the Particle Weight
SLAM of Mobile Robot
Acquire a Scanned Data (Environment Data)
Sending Data
Generate a Grid Maps
Count Matching Points of Scanned Data and Grid Maps
Calculation of Particle Weight based on the Matching Points
Sending Data
Position Estimation of the Mobile Robot based on the Particle Weight
Refer to Fig.7.

Fig. 2. Outline of Interactive SLAM

sensors are shared with the mobile robot and used as the constraints of the mobile robot's SLAM. In that case, grid map by distributed sensor and sensing results of the mobile robot are compared in each particle in addition to matching with grid maps in SLAM process by mobile robot with laser sensing results. Particles that match both maps will be preserved as good particles in the FastSLAM. This is effective for improving accuracy of grid maps, because maps built by distributed sensors fixed in the environments are integrated to map built by mobile robot.

2.2 System Configuration

System configuration of interactive SLAM with networked intelligent sensors is shown in Fig.2. This system is configured with two processes: SLAM process of mobile robot and distributed intelligent sensors process.

In the SLAM process, a mobile robot acquires mobile robot's translation and angular displacement from odometry and scan data from laser range sensor on the mobile robot. Grid map is generated from above acquired data and maps generated before. Since this SLAM process adopts FastSLAM based on particle filter, a grid map per each particle is generated. After map generation, grid maps of all particles are compared with scan data of mobile robot. Number of matching points between each grid map and scan data is counted. This number of matching points is used as weight of each particle. A position of mobile robot is estimated by weighted mean of particle positions. This SLAM process is performed every 500 mm.

In the distributed sensor process in intelligent space, distributed sensor acquires scan data and generates grid map. This distributed sensor process also adopts particle filter for estimating distributed sensor's position. A grid map for each distributed sensor is generated in sensor coordinate system. The map is transformed to the map in world coordinate system according to each particle position of the distributed sensor. After map generation, grid maps of all particles are compared with scan data of mobile robot based on the highest likelihood particle position in SLAM process. A different comparison between maps and scan data is applied in the distributed sensor process.

In the SLAM process, map matching is performed in sequential robot movements. Then, many matching points are relatively obtained in several particles. As against to evaluation of matching points in the SLAM process, enough matching points can't be obtained because estimation of mobile robot and distributed sensors include slight angular differences in many cases. In the distributed sensor process, orientation difference between each grid map and scan data is calculated for weight of particles, not only matching points. A position of distributed sensor is estimated by weighted mean of particle positions. And new weights are updated in SLAM process according to the estimated position of the distributed sensor. Orientation difference, between scan data at each particle position of mobile robot and grid map based on the particle with the highest likelihood in the distributed sensor process, is calculated and considered as new weight of particles. The weight based on orientation difference is adjusted in order to balance to weight based on matching point. And the adjusted weight is added to the already calculated weight based on matching points. Maps, which match both of FastSLAM and intelligent space, can be generated by this interactive SLAM. Position and orientation estimation, which be suitable for both of robot and intelligent space, is also achieved. This interactive SLAM is performed when a mobile robot moves in a monitoring area of a distributed intelligent sensor.

2.3 Generation of Grid Map

In this study, grid map generated by a laser range sensor on a mobile robot is called local map. And grid map generated by a laser range sensor in an intelligent space is called area map. Detailed generation method of grid map, such as local map and area map, is shown in the literature [2]. Laser range sensors acquire scanned data every sampling time. Scanned data is accumulated and occupancy probability of each grid is calculated. Area maps generated from distributed sensors are based on sensor coordinate system because distributed sensors do not move. Local maps of mobile robot are generated from scan data and corresponding positions on robot coordinate system. These maps can be transformed on world coordinate system, using current position and orientation estimation. Current scan data by the laser range sensor on the robot and a local map generated in the past is compared in map estimation of usual Grid Based Fast-SLAM. Likelihood is calculated based on the number of scan points, which matched with the grids with high occupancy probabilities in the local map. This likelihood is calculated in each particle of particle filter. Robot position is estimated by the weighted mean based on likelihood of each particle. Fig.3 shows an example of experiment environment. Fig.4 shows an example of scan data by the laser sensor on a mobile robot in the environment. In the environment, moving humans and the other static objects exists. In this example, installation height of a laser range sensor on the mobile robot is about 340 mm from floor of Fig.3. Fig.5 shows a grid map generated by accumulating scan data like Fig.4. In Fig.5, red points represent grids that the local map and the scanned points are corresponding. This number of matching point is used as likelihood of each particle.

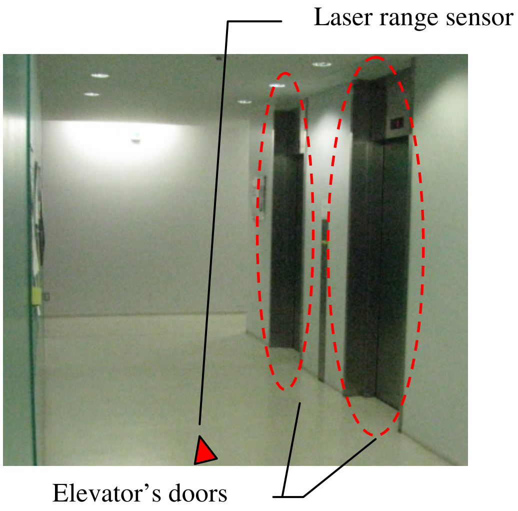
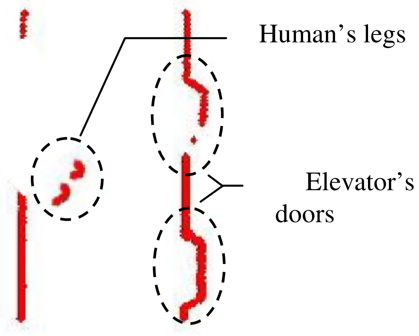


Fig. 3. An experiment environment



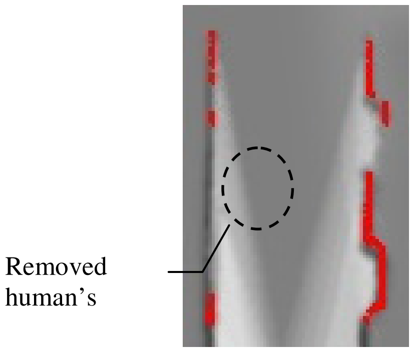


Fig. 4. Scanned data

Fig. 5. Local map

2.4 Matching with Maps by Distributed Laser Range Sensors

In the interactive FastSLAM using intelligent space, when a mobile robot moves into a monitoring area of a distributed laser range sensors, weight of each particle in distributed sensors is also computed from matching results between scanned data of a mobile robot and grid maps which the distributed sensors generated. These particle weights are calculated as a particle weight of FastSLAM in Fig.2. That is, when the mobile robot runs in the measuring range of a distributed sensor, it is aimed that accuracies of position and map estimations in SLAM process are improved by evaluating the matching point with the grid map from the distributed sensors. Fig.6 is an example situation of matching with the map from the sensor fixed in intelligent space. In this figure, red points are current scanned points by the mobile robot. Blue points show the grids, which the map by distributed sensor and the sensing points are corresponded. It is enough when many correspondent points are measured as shown in Fig.6. However, as shown in left figure of Fig.8, there are many cases that correspondent points between the map by the distributed sensor and scanned data are small because of orientation differences between maps and sensing data especially. It means that the number of correspondent points is not enough to evaluate matching with the maps of distributed sensors. So, in this study, orientation differences between maps of distributed sensors and scanned data are used for evaluation of matching. At first, two grids with the highest occupancy probability in the map of distributed sensor are selected as representative points. Next, the points nearest to selected two points are searched in the scanned data observed by laser sensor on the mobile robot. Finally, inclinations of straight lines between two points on world coordinate system are calculated by the 2 sets of two points. Differences of the inclinations are reflected to likelihood of each particle in the distributed sensor process. And this difference of the inclinations is adjusted into the similar value with the size of matching points in order to balance with particle weight calculated already in SLAM process. In this study, this weight based on the inclination is increased 10 times because it is smaller than the weight based on matching points. The adjusted inclination weight is added to weight based on the matching points, and the sum total is used as a final weight in SLAM process. Fig.7 shows an example of comparison of orientation. The maps of distributed sensors and the scanned data of mobile robot are evaluated by performing this process for every cycle of the SLAM process. Matching result is corrected on running of the mobile robot as shown in Fig.8. The number of matching points is shown by blue points increase in this figure. In the monitoring area of distributed sensors, the particles which match both of the past local map by SLAM and the maps by distributed sensors will be selected as this matching. And positions and orientations of distributed sensors are estimated suitably.

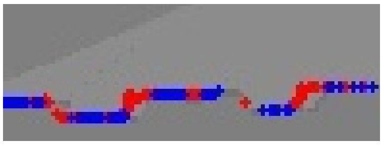


Fig. 6. Sensing data corresponding to map by a distributed sensor

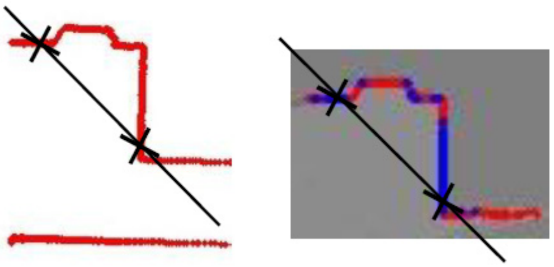


Fig. 7. Comparison of orientation

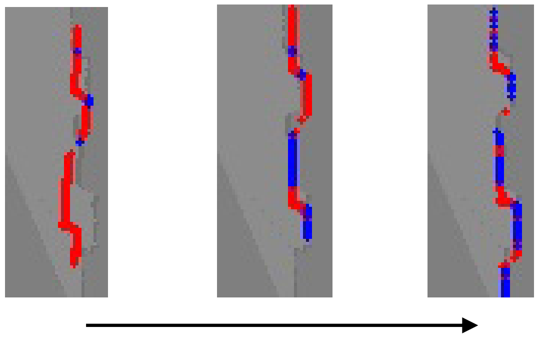


Fig. 8. Match correction

3 Experiment

Experiments for evaluating the proposed method are performed on the conditions which rough positions and orientations of distributed laser range sensors are given as initial values. The system of interactive Fast-SLAM is implemented and the experiment of self-position estimation and map generation was conducted. As experiment environment is showed in Fig.9, the mobile robot moved a squared course in one floor of building of our university. A length of the course is about 100 m. Four distributed sensors were installed at "x" marks in Fig.9. Each distributed sensor's monitoring area is a circle with radius 2500 mm. Mobile robot performs FastSLAM every 500 mm moving. Interactive SLAM with distributed sensors was performed at 5 areas, since the mobile robot can be monitored by a same distributed sensor around start and goal positions. In this experiment, the number of particles in a particle filter of SLAM is 50, and a grid size of maps is set to 50 mm x 50 mm. Pioneer3-DX was used as the mobile robot. Laser range sensor UTM30-LX was installed in the robot and URG04-LXs were distributed in the intelligent space. Although accurate initial positions and orientations of distributed sensors were unknown, rough initial positions and orientations of distributed sensors were given in this experiment. Fig.10 shows the map generated by usual FastSLAM only. On the other hand, Fig.11 is a map built by information sharing with four distributed sensors with the interactive SLAM. A part of a dotted square in Fig.10 has matching errors when the mobile robot revisited this area. However, an accurate map is obtained in a part of a dotted square in Fig.11,

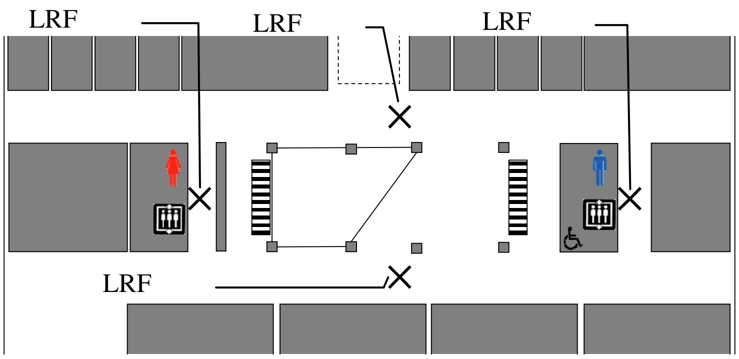


Fig. 9. An experiment looping course

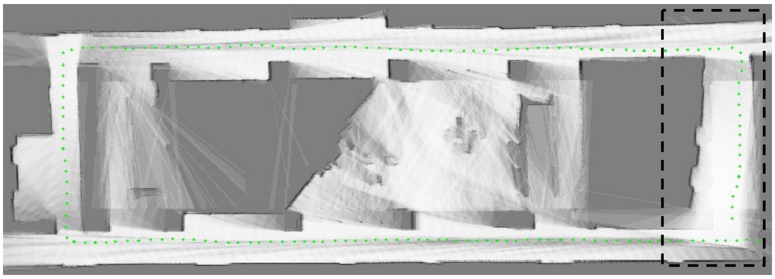


Fig. 10. FastSLAM

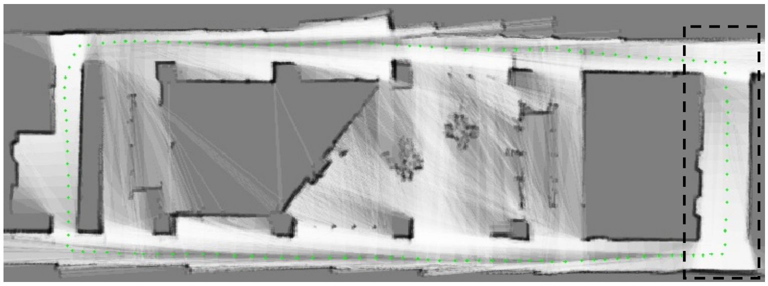


Fig. 11. Interactive SLAM

because the robot position and map are estimated by particles, which also matched with the maps from the intelligent space. Fig.12-15 is results of map matching in the measurement areas of four distributed sensors in the interactive SLAM. Left image in each figure is scan data obtained in the mobile robot. Middle image is a local map generated by mobile robot. Red points in middle image show matched points between scan data and local map. Right image is an area map generated by the distributed sensor in each “x” area. Blue points in right image show matched points between scan data of mobile robot and area map.

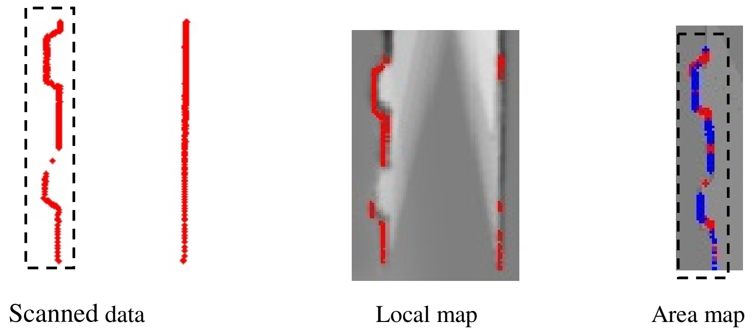


Fig. 12. Sensors matching in “LRF No.0” of Fig.11

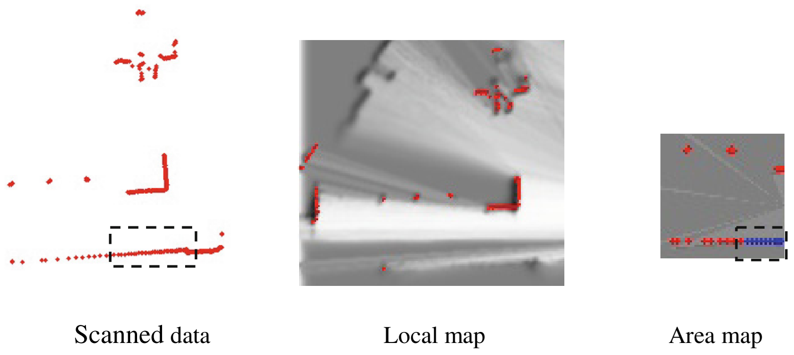


Fig. 13. Sensors matching in “LRF No.1” of Fig.11

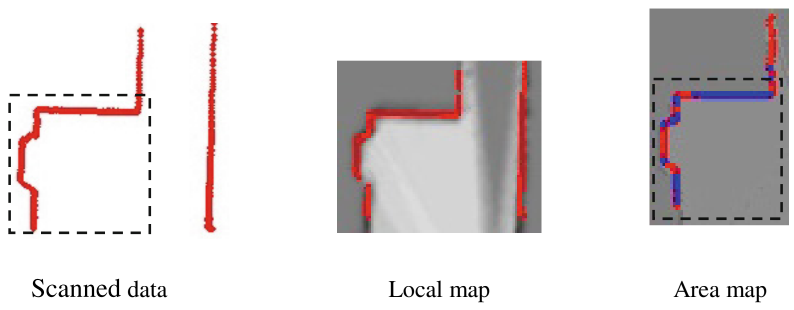


Fig. 14. Sensors matching in “LRF No.2” of Fig.11

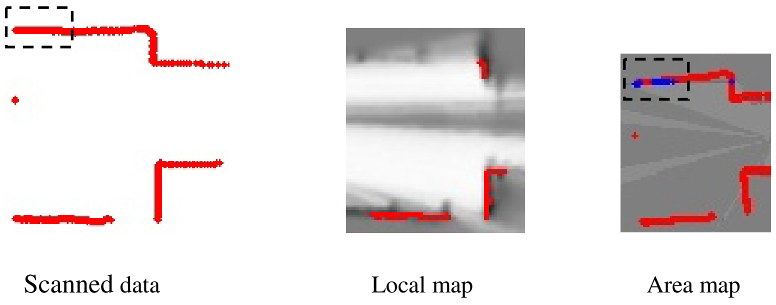


Fig. 15. Sensors matching in “LRF No.3” of Fig.11

Table 1. Each position parameter of distributed sensors

LRF No.0	x [mm]	y [mm]	θ [deg]
Measured Value	-210	-3490	0
Initial Value	0	-4000	0
Estimated Value	268.47	-3479.47	-0.61

LRF No.1	x [mm]	y [mm]	θ [deg]
Measured Value	16290	10	0
Initial Value	15000	0	0
Estimated Value	16408.86	-224.21	-2.05

LRF No.2	x [mm]	y [mm]	θ [deg]
Measured Value	32865	-4990	0
Initial Value	35000	-5000	0
Estimated Value	33279.47	-5109.58	0.51

LRF No.3	x [mm]	y [mm]	θ [deg]
Measured Value	16290	-9990	0
Initial Value	15000	-10000	0
Estimated Value	13813.53	-9487.64	0.96

Fig.12-15 shows maps and matching result of interactive SLAM. Table.1 shows position estimation results of distributed sensors in interactive SLAM. Measured Value is the position of distributed sensor measured manually on world coordinates as reference positions for evaluation of estimation results. Measured values are not necessarily accurate positions because it is not easy to measure sensor positions distributed widely on unified world coordinate system. Initial Value is the given rough initial value for each distributed sensor in interactive SLAM. Estimated Value is the final estimated value of each distributed sensor by interactive SLAM. In Fig.12 and Fig.14, many matching points between scan data and area map are obtained.

On the other hand, in Fig.13 and Fig.15, matching points are obtained slightly. Considering Fig.12-15 and Table1, this reason is that there are a few landmarks in the monitoring area of distributed sensor in Fig.13. In Fig15, the reason is that estimation of distributed sensor's orientation differed from scan data of mobile robot. Quantitative accuracy of Estimated Values cannot be explained enough in only Table.1. However, the generated map in Fig.11 show good loop closure in revisiting area. This accurate map generation means that Estimated Values are appropriate values on world coordinate system. These experimental results show that positions and orientations of the distributed sensors are updated to actual values according to robot movement simultaneously with accurate map building.

4 Conclusion

In this paper, it was shown that it is effective in improving the accuracy of SLAM by using information of distributed laser range sensors in intelligent space in the SLAM problem. This paper also described a possibility that the proposed SLAM is utilizable to estimating positions and orientations of distributed sensors in intelligent space.

When initial positions of distributed sensors are set out of their monitoring ranges, accurate positions of distributed sensors cannot be estimated by the current version of the proposed interactive SLAM algorithm. That means there are restrictions for initial position setting. As future works, the proposed algorithm should be improved for accepting flexible initial positions.

References

[1] Lee, J.-H., Hashimoto, H.: Intelligent Space - concept and contents. Advanced Robotics 16(3), 265–280 (2002)

[2] Thrun, S., et al.: Probabilistic Robotics, pp. 1–483. MIT Press (2007)

[3] Morioka, K., et al.: Human-following mobile robot in a distributed intelligent sensor network. IEEE Trans. on Industrial Electronics 51(1), 229–237 (2004)

[4] Kuroiwa, S., Morioka, K.: Development of Easy Camera Calibration Tool under Unified World Coordinate System Using Online Three-dimensional Reconstruction. In: Proc. of the 17th International Symposium on Artificial Life and Robotics, pp. 1179–1182 (2012)

[5] Haehnel, D., et al.: A highly efficient FastSLAM algorithm for generating cyclic maps of large-scale environments from raw laser range measurements. In: Proc. of IROS, pp. 1–6 (2003)

[6] Stachniss, C., Hähnel, D., Burgard, W.: Exploration with Active Loop-Closing for FastSLAM, pp. 1–6 (2004)

[7] Lu, F., Milios, E.: Robot Pose Estimation in Unknown Environments by matching 2D Range Scans. Journal of Intelligent and Robotic Systems 18, 249–275 (1997)

Computing 2D Robot Workspace in Parallel with CUDA

Paul Kilgo, Brandon Dixon, and Monica Anderson

Department of Computer Science, The University of Alabama, Tuscaloosa, AL 35487

Abstract. Workspace analysis is one of the most essential problems in robotics, but also has the possibility of being very tricky in complex cases. As the number of degrees of freedom increases, the complexity of the problem grows exponentially in some solutions. One possibility is to develop solutions which approximate the workspace for speedup, but this paper explores the possibility of using graphical processing units to parallelize and speed up a forward kinematics-based solution. Particular real-time applications are discussed. It presents a formal analysis of a simple 2D problem, a solution, and the results of an experiment using the solution.

1 Introduction

A common type of robot in industry is a manipulator robot, or more colloquially a robotic arm. This type of robot is secured to a base of some sort and has one or more segments, and eventually attaches to a tool. They are used in all kinds of applications, such as applying paint to a car, welding, and are a popular starting point for teaching robotics.

Manipulator robots are affixed to some **base**. The tool, gripper, or manipulator located at the end of the arm is referred to generally as an **end effector**. Therefore, the **workspace** of a robot is the set of all points the end effector may reach for a given base. Computing the workspace for a given robotic arm, abstractly referred to as a **kinematic chain**, is called **workspace analysis**. An example workspace is shown in Fig. 1. A more complete review of manipulator kinematics is available in most robotics textbooks, or the textbook by Craig [5].

Normally workspace analysis is a static type of analysis that needs to be computed once and remembered, therefore timing is not really an issue. However, there are some cases in which the kinematic chain is not static. For instance, the workspace will change anytime an characteristic of the kinematic chain changes. So in cases where the robot is being designed, it could be useful to the designer to quickly compute the workspace to verify incremental changes. As well, self-reconfigurable robots are robots whose kinematic chain is not static, either modifiable during run-time or in between run-times. In this case, the workspace would need to be computed at run-time (given that the robot can know or detect its kinematic chain). This could also present an interesting application in the case where the robot link is damaged. If a particular link of the robot is

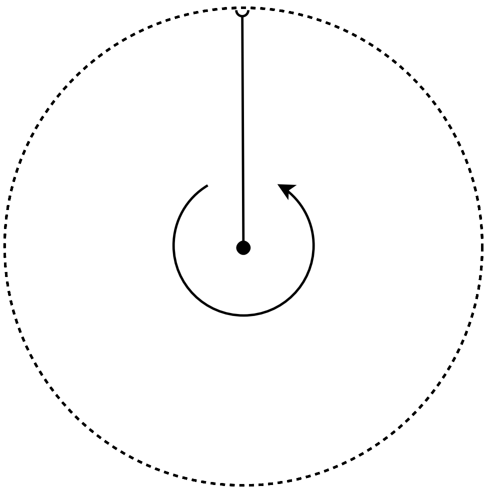


Fig. 1. Illustration of a simple workspace

damage, the damaged could be modeled by an adjustment to the link parameters and the workspace recomputed. This would complicated collision detection algorithms, and sensors for detecting and measuring such damages would need to exist.

1.1 Applications

There are many reasons to perform a workspace analysis. It can be used as an aid to place the robots in a factory floor to avoid collision. It is used in path planning to avoid any sensed obstacles. As well, it can be used to know if a provided target is reachable at all.

A first thought for specific application could be a robot CAD program. Such software is likely to have a workspace analysis feature present. If a designer makes a change to the robot's design, there is a good possibility such a change will alter the workspace. At that, changes in such applications are incremental and next future additions rely on some sort of feedback from the design application, such as the updated workspace. Therefore, the designer's productivity may be bottlenecked by the speed of the workspace analysis algorithm. Software for design work are typically deployed on systems with high end graphics cards which are more likely to be CUDA-enabled. Workspace analysis algorithms present in these CAD programs may benefit from parallelization at the payoff of increased user satisfaction for quicker results in incremental changes applied to a robot's design.

The researchers in [7] use workspace analysis as a step in their self-righting algorithm. This particular application is interesting since if a robot is not in a state which it can drive it could be due to some sort of damage from the environment. In this case, the workspace can not be assumed to be static may need to be recomputed. As well, this task could be time-sensitive either due to mission parameters or environmental danger. Therefore, this application might benefit from a speedup of workspace analysis.

2 Related Work

Workspace analysis is a fairly well-studied problem within the realm of robotics and particularly the literature focuses on new methods and algorithms. The researchers in [4] employ a probabilistic method to choose elements of the joint space and apply forward kinematics to compute the position of the end effector. They then employ methods of generating a boundary curve from the resulting point cloud. This is done for both 2D and 3D workspaces.

The method used in [3] instead relies on inverse kinematics. The algorithm described iterates over various test points and solves the inverse kinematics equations and ensures no constraints were violated. This method ensures that one will not waste time on computing the same point inside the workspace twice.

An analytical method of solving for the workspace boundary is presented in [6]. In particular, it deals with solving for the singularities associated with a particular robot, determining which of these singularities is a boundary singularity and incorporating them into the solution.

The researchers in [2] present a novel bottom-up method of solving the workspace problem which facilitates reuse of previous solutions. Reuse of solutions while helpful in serial implementations tends to be offsetting for parallel solutions. This particular solution may benefit from parallel solution for large values of ψ as parts of the algorithm could be done in a single parallel step rather than iteratively.

The particular method employed in solving for the robot workspace is not important, as one may be preferable over another for a given application. All may benefit from parallelization with similar speedup if possible.

Interestingly, there is little in the literature relating to CUDA kinematics solutions or even parallel solutions in general. The work in [1] presents a parallel, CUDA-based, genetic algorithm for solving the inverse kinematics problem with promising results. However workspace analysis remains particularly unstudied. Such an inverse kinematics implementation may aid in the speedup of workspace analysis.

3 Approach

The segments which make up the robotic arm are called **links**. Any two links may be connected together by a **joint**, which is some sort of topological constraint we impose between them. For 2D workspace analysis we define ℓ to be the length of the link and θ to be the angular deviation between a child and parent link. This is illustrated in Fig. 2.

A joint simply constrains one or more degrees of freedom. There are many theoretical possibilities for a joint, but the two most basic joints are **hinge**

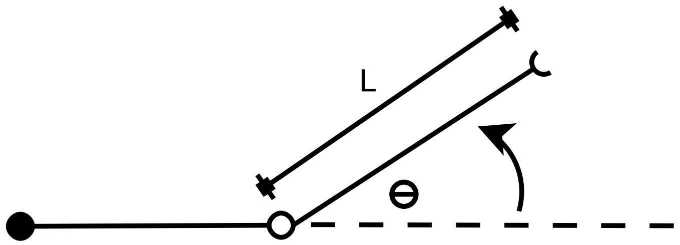


Fig. 2. Illustration of robotic arm parameters

joints, which allow θ to vary but hold ℓ fixed. Conversely, **prismatic joints** allow ℓ to vary but hold θ fixed. Each of these parameters is constrained by an upper and lower bound.

The base link is normally assumed to be the origin of the workspace. Therefore to compute the position of the end effector it is just a matter of computing a linear combination of trigonometric functions. A recurrence equation for the position of the tip of the n^{th} link is given by Eq. [1]:

$$\begin{aligned}(x_n, y_n) =& (x_{n-1} + \ell_n \cos\theta_n, \\ & y_{n-1} + \ell_n \sin\theta_n) \text{ for } n \geq 1 \\ (x_0, y_0) =& (0, 0)\end{aligned} \tag{1}$$

A point should not be considered a part of the workspace if no safe solution exists, where by **safe** it is meant there is no collision with any obstacles. We therefore need to check solutions for collisions with obstacles. Figure [3].

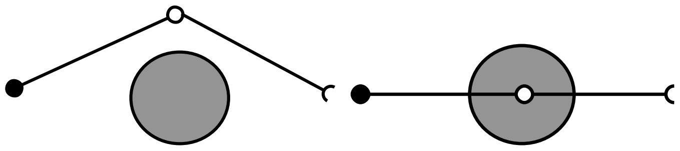


Fig. 3. Illustration of a safe and unsafe solution

Obstacles will be modeled as circles at some center (H, K) with radius R. For a geometric analogy, links may be thought of as a line segment. The circle-line intersection problem is well-known [9,11].

4 Solution

The problem is now formally defined, so a formal method to solve it may be given. The input to the solution is L links each with five parameters: type, static length (ℓ), static angular deviation (θ), upper bound, and lower bound. A second input is O objects with three parameters: center (H, K) and radius (R).

We then must define ℓ_n as a function of the upper bound (u_n), the lower bound (l_n), and the state identifier $0 < i < N - 1$, which we do by a simple linear interpolation. This is given in Eq. 2:

$$\ell_n = \begin{cases} l_n + (u_n - l_n)\frac{i}{N-1} & \text{if joint } n \text{ is prismatic,} \\ \ell & \text{otherwise} \end{cases} \tag{2}$$

We have a similar definition for θ_n:

$$\theta_n = \begin{cases} l_n + (u_n - l_n)\frac{i}{N-1} & \text{if joint } n \text{ is a hinge,} \\ \theta & \text{otherwise} \end{cases} \tag{3}$$

The general approach for calculating the workspace is as follows:

1. Start with L links and their parameters, O obstacles, and the number of discrete states to evaluate at each joint, N.
2. For each possible $0 < i < N$ at each possible joint $0 < j < L$, compute ℓ_j and θ_j.
3. Compute the location of the end effector for all possible combinations of joint states.
4. For each state, mark the position (x_{L-1}, y_{L-1}) as unsafe if there exists a $0 < k <= L$ such that the line segments defined by (x_k, y_k) and (x_{k-1}, y_{k-1}) intersect any of the O obstacles.

The complexity of the problem is $\theta(OLN^L)$.

The presented problem is highly parallelizable. The end effector's position at each possible state is wholly independent of all other computations. Therefore, it seems tractable to exhaustively compute each position by way of brute force. The above was implemented in CUDA C, with an output being a heat map.

A good standard benchmark for this problem is the three-link, three prismatic joint robot arm. It is an appropriate 2D analog to a common configuration in 3D robots, as well the workspace is well-known. An example of the solution output is shown in Fig. 4 as a heat map..

4.1 Optimizations

The array that contains the obstacle and link parameters are read multiple times between all the threads. Therefore, it is an obvious optimization that those parameters could be placed into shared memory. The array will be read sequentially by each thread in operation in the same order, so threads of the same warp will benefit in that the GPU can broadcast the memory at that location simultaneously to all of them. This optimization is implemented.

Another possible optimization is to stray away from operations which are computationally expensive. In particular, the code which detects collisions with obstacles involves solving the quadratic equation and therefore computing a

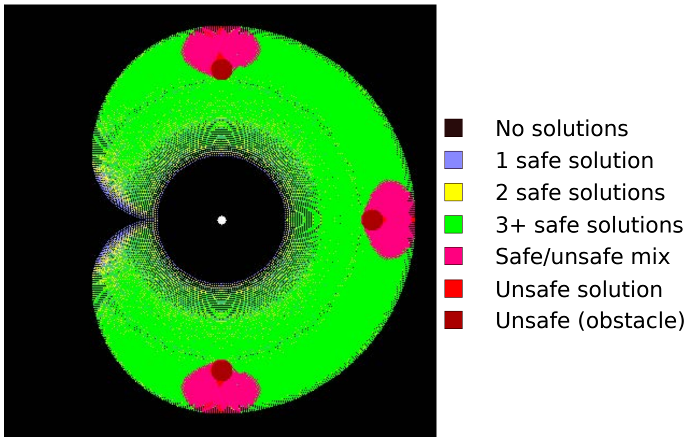


Fig. 4. Example solution output

square root. Unfortunately, there does not seem to be a way to avoid computing this square root as it carries in one form or another when rewriting the equation. Thus, it might be best to switch to a different algorithm for detecting collisions, or to approximate or assume something which allows us to avoid it. For instance, it might be better to iteratively check some N points along the line segment and see if they are contained in a circle as this can be done without computing square roots. This optimization is not implemented, and may not be worth pursuing as it only saves 50ms of compute time in the most longest-running test case.

Similarly, we may wish to avoid calls to `cos` and `sin` for the same reason (though a GPU ought to be well-optimized for these operations). Nonetheless, it is very safe to use approximations in workspace analysis because when there is no closed-form way of determining the solution it must be computed iteratively. A possible way to avoid the use of `sin` and `cos` might be to precompute for several values between 0 and π and store these in shared memory. We could then interpolate to an approximation with hopefully tolerable error. This could end up to be too much work for little or no payoff.

More work could be done in parallel. Between each segment on the kinematic chain, detecting a collision between a segment and an obstacle is independent from the results of any other segment in the chain. Therefore, this could be done in a parallel step.

Finally, the heat map itself could be computed in shared memory. This is similar to the histogram problem. Each thread needs to add some information to the heat map once. Currently, this is done in global memory. It may be possible to speed this up by computing the heat map in shared memory. However the speedup may not be worth it as it is rather unpredictable where the thread will need to write and will almost certainly lead to bank conflicts. This optimization was not implemented.

5 Experiment

To test the efficacy of the parallel solution against the single-threaded solution, an experiment was designed. Five different robotic configurations were created and the parallel and single-threaded solutions were timed using the CUDA timer library. Two separate parallel solutions were analyzed: one using shared memory and another which does not. The specifications of the hardware may be seen in Table 1. The data obtained is shown in Table 2. Each of the shown times is averaged over 30 trials. The shown speedup is the ratio of CPU time to GPU+SHM time.

Table 1. Specs of hardware involved in experiment

Property	GPU	CPU
Name	Quadro FX 570	Core 2 Duo E6550
Core clock MHz	460 MHz	2.33 GHz
Memory clock	400 MHz	667MHz
Memory size	256 MiB	2 GiB
Memory speed	12.8 GiB/s	N/A
Cores used	16	1

Table 2. Data obtained from the experiment

No.	Pr	Hi	L	N	O	N^L	GPU (ms)	GPU+SHM (ms)	CPU (ms)	Speedup
1	0	10	10	5	3	9765625	5807.39	1780.75	34196.00	19.20
2	1	1	2	2000	2	4000000	424.76	182.286	2519.52	13.82
3	1	1	2	1000	2	1000000	108.52	43.4624	632.89	14.56
4	0	3	3	100	3	1000000	220.56	75.6715	1143.11	15.10
5	1	0	1	1000	2	1000	0.09	0.0518283	1.04	20.06

Also of interest is how well the problem scales with respect to problem size. The CUDA solution will assign a thread to each possible entry in the discretized joint space. Therefore, the number of obstacles O should not have an effect on the number of threads launched. Therefore, N and L should be varied to examine the effect on average computation time. A similar experiment was designed as before. In one case (Fig. 5) L is varied with the constraint held that $N = 5$. In the other case (Fig. 6) N is varied with the constraint $L = 3$.

6 Analysis

The high parallelizability of the problem manifests in small and large test cases. The large speedup was unexpected with the addition of a shared memory. The 20X speedup could be due to some sort of memory bandwidth bottleneck in the

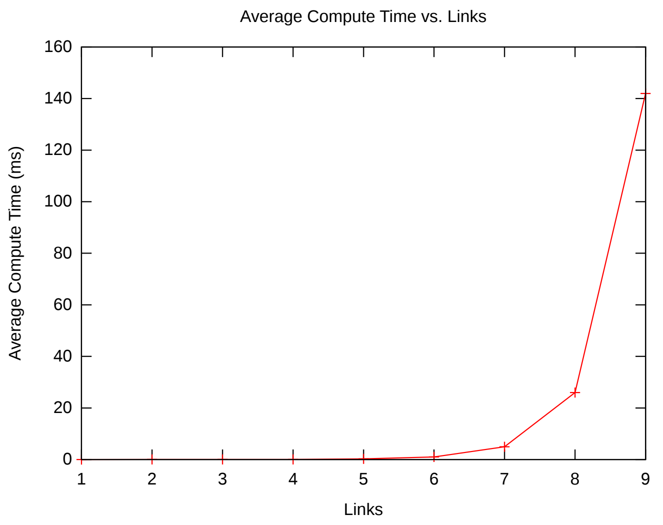


Fig. 5. The plotted relationship between average computation time and number of links (L)

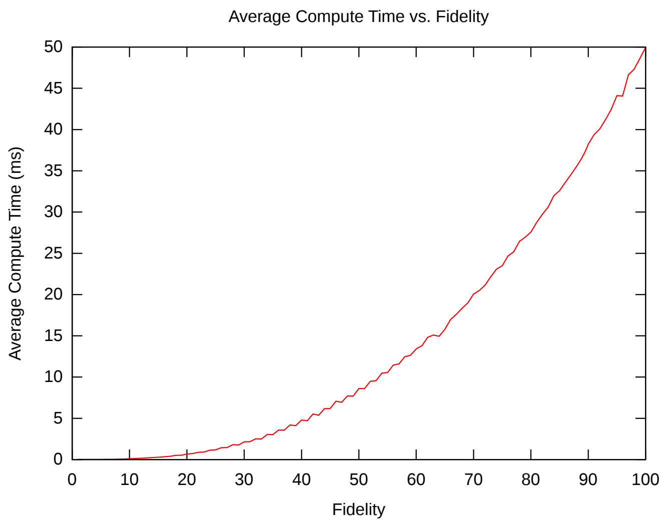


Fig. 6. The plotted relationship between average computation time and discretized joint space fidelity (N)

CPU solution, perhaps due to throttling from the operating system. Though it is more interesting that tests 2 and 3 fell below the normal 16X speedup. This could be due to a `__syncthreads()` call taking a more significant hit to the performance for shorter-running trials. It is clear that a parallel solution outperforms a single-threaded solution. This confirms the high paralellizability of the problem.

Putting aside the performance data, we should evaluate the quality of the solution given. For discussion we shall explore in particular Fig. 4. With a closer inspection of the heat map one can easily see there are irregularities in perceived contiguous regions. For instance, black pixels litter the interior of the workspace as well as other colors in the green regions. This sort of problem is inherent to the approach: discretizing the joint space leads to such impurities in the result. This solution does not employ any smoothing techniques or border calculation in attempt to counteract this problem.

More errantly there are some red pixels within regions occupied by an obstacle. For clarity, the color red is reserved for unsafe regions which are not occupied by an obstacle. While still technically correct, it does raise into question the accuracy of the solution, though it is likely attributed to a small coding error.

However, the challenge in this problem lies in the scalability of the solution. Workspace computation increases in complexity fastest as the number of degrees of freedom, or in this particular case the number of links, increases. Therefore, complex kinematic chains are still a problem for a naive solution such as this. The exponential portion of the complexity, L^N, easily inflates the number of joint configurations to check. Future methods for dealing with the exponential scaling of the problem are discussed in the conclusion.

7 Conclusion

As evident in Fig. 5, computation time takes a sharp turn as the number of links increases past even a modest number. This is still due to the limited number of cores available on the GPU. As only 16 cores were present in this experiment, only 16 threads may be scheduled at a time thereby limiting our speedup to about a constant factor of 16.

In typical systems, the number of cores available is likely to stay at this number. A cheaper solution would be to use a rougher approximation of the workspace in a tradeoff to lower the computation time. This will have the most impact when lowering either L or N. However, a modification to L corresponds to a fundamental change in the robot configuration and is therefore not useful. We are then forced to approximate the workspace by choosing smaller values of N.

Disappointingly, Fig. 6 does not show quite as sharp of a reclamation of computation time as N decreases. As well, decreasing the fidelity of the generated workspace can have unwanted consequences as the N uniformly divides the joint space into even sections. The unlucky division of these sections could lead to a missed unsafe configuration as Fig. 7 illustrates.

One possible solution is to generate a fixed number C of joint states to select at random. This allows the problem to scale to larger values of L and arbitrary values of N and the complexity changes to $\theta(OLC)$. This can lead to similar

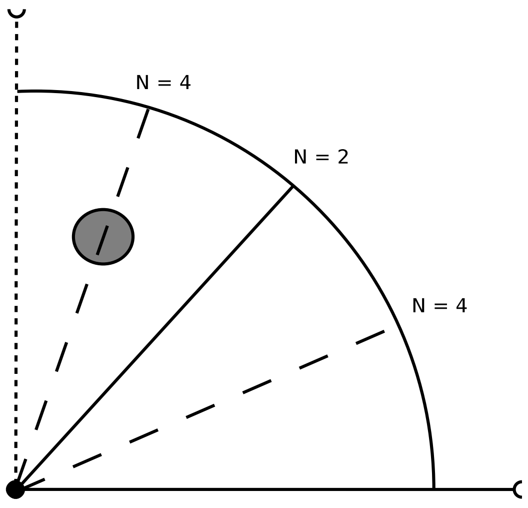


Fig. 7. Illustration of how an improper choice of N could lead to missed unsafe configurations. $N = 2$ is not granular enough to detect the obstruction, however $N = 4$ is.

situations as addressed before, but C can be finely adjusted to suit the particular application or repeated iteratively as necessary, making this a more flexible option.

However, random number generation in parallel algorithms has its own set of problems. As pseudo-random number generators (PRNG's) are inherently iterative, they can pose a threat to the performance of the application as all the threads must read and update the seed. Thus, specialized PRNG's must be developed for parallel applications. The interested reader may refer to [10] for information about how this problem was solved for Mersenne Twisters.

Implementations for the uniform distribution in CUDA do exist [8,10] however it may not be the best choice for selecting points. The particular kinematics of a robot may cause test points to cluster in a small part of the actual workspace, giving a warped description of the workspace. The researchers in [4] present a study on this and suggest the adoption of the Beta distribution for their particular goal of calculating the border of the robot workspace. Their approach is reliant on a sufficient number of border points being chosen and they had good results parameterizing the Beta distribution as a function of the joint limits. Since approximations must be made, it could be a more worthy goal to calculate the exterior of the workspace instead.

The possibility of implementing an on-line PRNG for the Beta distribution is therefore interesting. However, the since uniform distribution well-studied and has fast implementations for CUDA it is of most interest. One can use inverse transform sampling to transform a uniformly distributed value into arbitrary distributions by inverting the cumulative density function (CDF) of interest. However, the inverse of the Beta function's CDF does not have an analytical form, so an iterative method must be employed. This could end up having a worse performance improvement than desired in an approximation, but there is a converging Taylor series [12] for the inverse regularized beta function which may yield promising results.

Workspace analysis lends itself very well to computation on the GPU. This is due to the independent nature of the solutions obtained and perhaps due to some floating point operation optimizations present on the GPU itself. Some possible applications which could benefit from on-line workspace analysis are presented.

A formal analysis of a simple 2D forward kinematics-based workspace analysis approach is presented and a solution is described. A solution for the problem of workspace analysis has been implemented in CUDA C and in some cases was able to exceed the normal 16X speedup typical of highly parallelizable CUDA solutions. Some other optimizations may be possible and a few ideas are given.

There are some limitations with this solution. The solution does not check for self-interference in the links. Implementing this would have a noticeable effect on the running time, though it could be implemented in parallel if the link endpoints are cached. Also, links are modeled as simple line segments as opposed to rigid bodies. The geometry of the links is essential for collision detection, though robotic simulators will often use geometric approximations so that the collision detection may be performed analytically.

Future work for this project might be to see how the speedup benefits from randomly selecting a subset of joints states from the set of all possible joint states. This would likely result in an arbitrary speedup at the cost of a rougher approximation of the workspace. Additionally, rather than calculating the points in the workspace, the parallelization of calculating the workspace border could be explored. Methods for this are outlined in [4]. However, to take benefit from this method, approximating the Beta distribution in parallel must be explored to find the appropriate method for maximized performance.

As well, it would be interesting to apply parallelization to other types of workspace analysis methods to measure how parallelization affects different types of analysis algorithms, or whether or not such algorithms are parallelizable.

The orientation of the end effector could be useful in workspace analysis as well. The described solution could be modified to support this by providing a extra set of bounds for the kinematic boundaries of the end effector.

Finding singularities may also be an interesting problem to solve on the GPU, as there could be interior singularities in the workspace. An effective solution which computes boundary singularities could draw the boundaries of the robot workspace and potentially solve the problem faster.

References

1. Aguilar, O.A., Huegel, J.C.: Inverse Kinematics Solution for Robotic Manipulators Using a CUDA-Based Parallel Genetic Algorithm. In: Batyrshin, I., Sidorov, G. (eds.) MICAI 2011, Part I. LNCS, vol. 7094, pp. 490–503. Springer, Heidelberg (2011)

2. Anderson-Sprecher, P., Simmons, R.: Voxel-based motion bouunding and workspace estimation for robot manipulators. In: 2012 IEEE International Conference on Robotics and Automation (ICRA), pp. 2141–2146 (May 2012)
3. Bonev, I.A., Ryu, J.: A new approach to orientation workspace analysis of 6-dof parallel manipulators. Mechanism and Machine Theory 36(1), 15–28 (2001)
4. Cao, Y., Lu, K., Li, X., Zang, Y.: Accurate numerical methods for computing 2d and 3d robot workspace. International Journal of Advanced Robotic Systems 8(6) (2011)
5. Craig, J.J.: Manipulator kinematics. In: Introduction to Robotics: Mechanics and Control, 2nd edn., pp. 69–112. Addison Wesley Longman (1955)
6. Goyal, K., Sethi, D.: An analytical method to find workspace of a robotic manipulator. Journal of Mechanical Engineering 41(1) (2010)
7. Kessens, C., Smith, D., Osteen, P.: Autonomous self-righting of a generic robot on sloped planar surfaces. In: 2012 IEEE International Conference on Robotics and Automation (ICRA), pp. 4724–4729 (May 2012)
8. Langdon, W.B.: A fast high quality pseudo random number generator for nvidia cuda. In: Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers, GECCO 2009, pp. 2511–2514. ACM, New York (2009)
9. Middleditch, A.E., Stacey, T.W., Tor, S.B.: Intersection algorithms for lines and circles. ACM Trans. Graph. 8(1), 25–40 (1988)
10. Podlozhnyuk, V.: Parallel mersenne twister. Technical report, nVidia (June 2007)
11. Weisstein, E.: Circle-line intersection. From MathWorld–A Wolfram Web Resource, http://mathworld.wolfram.com/circle-lineintersection.html (accessed April 27, 2012)
12. Wise, M.E.: The incomplete beta function as a contour integral and a quickly converging series for its inverse. Biometrika 37(3/4), 208–218 (1950)

Acquisition of Object Pose from Barcode for Robot Manipulation

Yuexing Han, Yasushi Sumi, Yoshio Matsumoto, and Noriaki Ando

Intelligent Systems Research Institute,
National Institute of Advanced Industrial Science and Technology (AIST),
AIST Tsukuba Central 2, 305-8568, Japan
{yuexing.kan,y.sumi,yoshio.matsumoto,n-ando}@aist.go.jp
http://www.aist.go.jp/

Abstract. General, robots obtain poses of target objects by matching the images of the observed objects with data in database. However, the process of matching images costs so long time that robot's action become slow. In order to shorten response time for robots searching target objects, we propose a method for robots to obtain information of poses of observed objects by calculating corner points of barcodes on the objects. Since information in a barcode is less than the one in an image, the method can help robot rapidly obtain the information of its target objects. Furthermore, in order to reuse the method in other robot systems, we create a RT-Component(RTC) to realize the method.

Keywords: Object pose, Barcode, RT-Middleware, RT-Component.

1 Introduction

In daily life and manufacturers, there are a lot kinds of employed robots to assist persons and produce products automatically. Based on the environments of using robots, robots are divided into two kinds [1]: one is used in industry which is a well structured environment, and the kind of robots can continuously repeat same tasks with a predesigned system including limited reactions; the other is used in human life environments which can happen many undefined things, so the kind of robots need a more complicated system to response various surroundings.

For robots in human life environments to understand their surroundings, many technologies and sensors have been developed, such as using ultrasonic, infrared, GPS, cameras, accelerometers, encoders and a lot more based on tasks of robots. In these sensors, cameras as eyes of robots are often used. After taking images and dealing with the images, generally matching images, robots can do many predefined tasks. In fact, matching images or recognizing objects is one of the most important works in robotics technologies. For example, a robust visual perception system which includes cameras to obtain images from surroundings was proposed for service robotics applications [2]. Lang *et al.* developed an attention system including cameras for noticing currently speaking person as interest one [3]. In [4], developers described a technique of object recognition for a mobile robot to deploy in a multistory building. In order to move from a floor to another floor, a mobile robot needs to recognize various observed objects related to an elevator.

Most methods for object recognition can be used in robotics systems. For instance, Ozen *et.al* described shapes of objects with the turning angle representation and robots can match two sets of landmarks by calculating their least square distance, which is based on shapes of objects [7]. Belongie *et al.* described an approach of matching shapes and recognizing objects with shape contexts [8]. L. He *et al.* proposed a robust skeleton-based graph matching method to recognize objects [9], and the methods can be also used in robot technology. The scale invariant feature transform (SIFT) [5,6] is one of famous methods to match two images and is often applied in robotics technique for robots to find their targets.

Though these methods can be used to match two images, they may cost long time because of complicated images, and robots have to extend work time to achieve their tasks. For instance, vending robot searches products and matches every product with the needed one which is commanded from customer. Before the robot hands on the objects to guests, it needs to know the objects' information including positions, poses, species and shapes. If the above mentioned methods of matching two images are used to find target product, it will cost so long time for robots to wait until the matching of images is finished.

In order to shorten response time of robots searching target objects, we propose a method for robots to obtain information of poses of observed objects by calculating corner points of markers on the objects. There are a lot of kinds of markers can be selected, such as ARToolKit markers. Since our method can help robots to control objects, especially commodities, and most commodities are attached by barcodes which are one-dimensional code, we choice the kind of codes in the work. With a barcode on the observed object, robot can easily to obtain information of the object including its type and position, furthermore, its pose with our method (see Fig. 1). The method can help robots rapidly achieve their works in human living environments. We will firstly present the method that obtain pose of a barcode in Sec. 2.

The proposed method is developed on OpenRTM-aist. Thus, the method can easily be reused into many robot systems. Furthermore, the component of the method might combine other components of image processing to help robot realize more complicated works. For example, if robot searches objects without barcodes, other components, such as SIFT or SURF component can be used to find the objects; even, component for barcode provides the basic information of objects, such as type, position, pose of the observed object to other components, then the components can achieve matching objects in short time, since the component of barcode has reduced searching range (see Fig. 2). Combining these components of object recognition, a more perfect system might be built to recognize objects and obtain information from the in complicated surroundings. The RT-Component about the method will be described in Sec. 3.

Then, some examples of the method will be shown in Sec. 4. Finally, we will give a conclusion for the paper in Sec. 5.

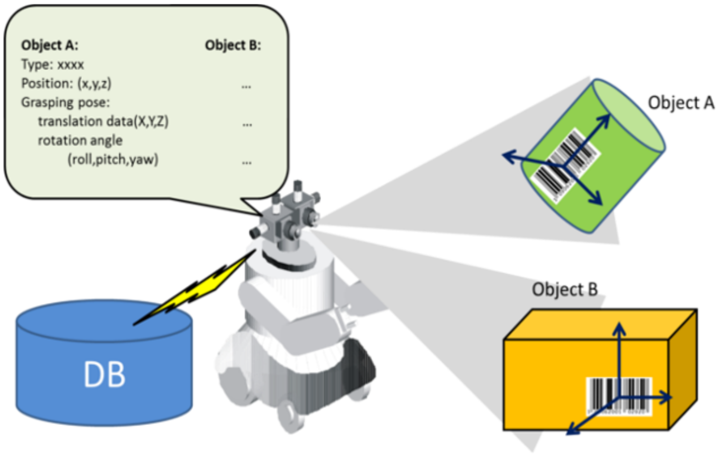


Fig. 1. Through observing barcodes, robot can obtain a lot of information including object's type, position and pose

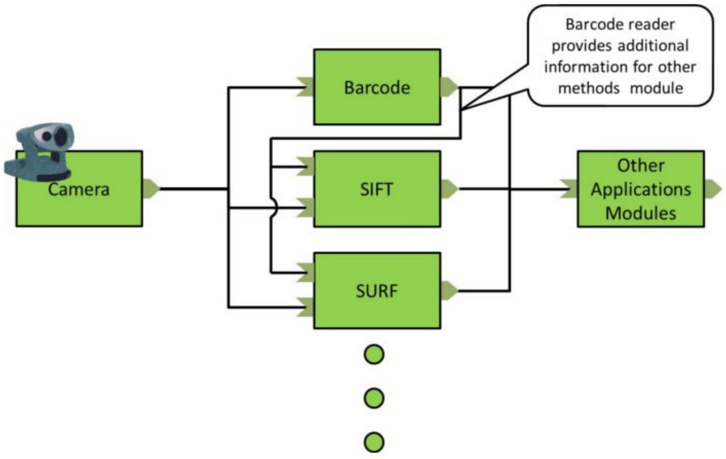

Barcode reader provides additional information for other methods module
Camera
Barcode
SIFT
SURF
Other Applications Modules

Fig. 2. Combining many components of object recognition, a more perfect system might be built to recognize objects and obtain information from the objects in complicated surroundings

2 Obtain Object Poses from a Barcode

A barcode is an representation of data which can be read by special optical scanners, i.e., barcode readers. It includes two sorts, i.e. 1-dimensional barcodes which are composed of parallel lines and 2-dimensional barcodes which include rectangles, dots, hexagons and other geometric shapes. In the paper, we mainly discuss 1-dimensional barcodes. Barcodes were firstly used to label railroad cars.

Since the beginning of the 1970s, barcodes have extensively used in supermarket checkout systems and postal systems.

We use an open source software of barcode reader, i.e. ZBar [10], to develop our technique. ZBar can scan many popular types of barcodes, such as, EAN-13/UPC-A, UPC-E, EAN-8, Code 128, Code 39, Interleaved 2 of 5 and QR Code. The library of ZBar has many programming interfaces for C++, Perl, Python.

Through these functions in the library and objects' database, we can obtain basic information including types, species and shapes of the observed objects. Simultaneously, positions of object can be calculated by measuring the positions of barcode. Then, poses of object will be obtained by calculating corner points of the barcode for robot to control the object.

First, a barcode is taken by a camera and then is input into a robot system, see (a) in Fig. 3. If there are many barcodes in an image simultaneously, the barcode reader can only recognize one with ZBar. In fact, if an observed object with barcode is scanned with ZBar, the barcode can be found by adjusting camera's focus and other parts are automatically ignored. The observed barcode is marked by words and numbers as same as the number under the barcode, and four columns of blue points nearby four lines of the barcode can be obtained. See (b) of Fig. 3.

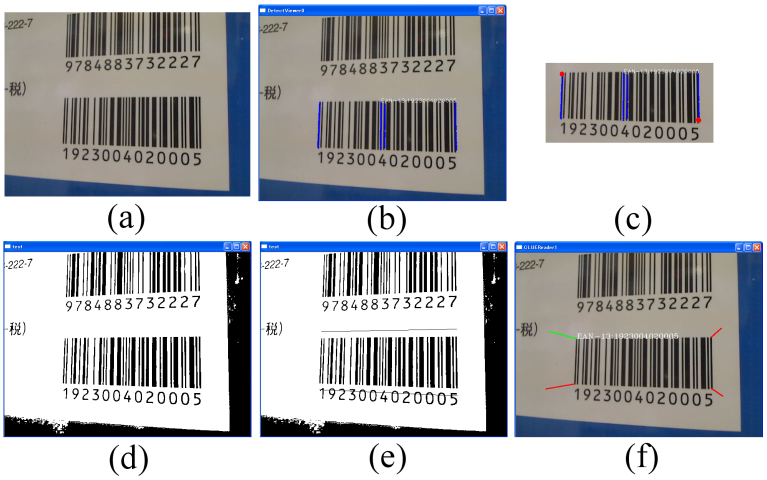


Fig. 3. Steps of obtaining ordering corners from a barcode

From the four sets of points on lines, two points whose distance is the longest are obtained as shown in (c) of Fig. 3, denoted as A and B. The two points are on the two outer blue lines, respectively, and all other points on the lines in blue color are not considered except for the two points. Though most blue points may not be located on the lines of the barcode as shown in Fig. 4, including A and B, they are very close to outer lines of the barcode. In order to find the corner points of the barcode, the two outer lines closing to A and B are ought to be obtained.

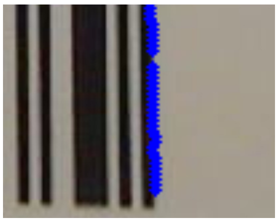


Fig. 4. Most blue points are not on the lines of the barcode

Then, the image including the barcode is transformed from color to black and white with an adjusted threshold. Since the image is taken in nature and the light cannot be controlled, there are many noises in the image and it is difficult to find a fixed threshold to differentiate the entire barcode in black and other points around lines of barcode in white. Thus, local thresholds for local parts in the image are calculated to reduce influence of surroundings. There are two thresholds corresponding with A and B, denoted as T_A and T_B, respectively. T_A is a mean value of points in the regions of A, and T_B is a mean value of points in the regions of B. Then, T is calculated with $T = (T_A + T_B)/2$ as a entire threshold to translate the image into black and white image, see Fig. 3 (d). With the method, most barcodes can be transformed into black color and other points into white color.

If A and B are located outside the lines of the barcode, colors of their neighbor points are all white; conversely, if they are inside the lines of the barcode, colors of their neighbor points are all black. If a point is on edge of the line, the colors of its neighbor points include both black and white. For any point p, its 8 nearest neighbor points is denoted as 1-neighbor points; then, points whose part neighbor points are 1-neighbor points of p are denoted as 2-neighbor of p; following the idea, we can denote points whose part neighbor points are $(n-1)$-neighbor points of p but not $(n-2)$-neighbor points of p as n-neighbor of p. Searching points from 1-neighbor to n-neighbor of A and B, it is easy to find two points on the edges of lines close to A and B, respectively, denoted as C and D. As shown in (a) in Fig. 5, A is located out of a line and B is located inner a line, which belong to the outer side lines of the barcode. C and D are their corresponding points on the contours of lines, respectively. Searching contour points from C and D, the top-left points of the two lines can be obtained, as shown in (b) of Fig. 5, which the left direction is more important than the top direction. Finally, two furthest points from C and D on the same lines of the barcode can be obtained by searching the contour points, respectively, as shown in (c) of Fig. 5. Thus, the four points as corner points of the barcode are obtained, designed as A, B, C and D in the following content and their indexes are 0, 1, 2, 3, respectivly. A and B are on one line, and C and D are on the other line.

The four corner points need to be arranged in clockwise direction order, i.e. 0, 1, 3, 2 in indexes. Indexes of two points on one line are 0 and 2, and the other two are 1 and 3, respectively. A method of ordering points is possibly

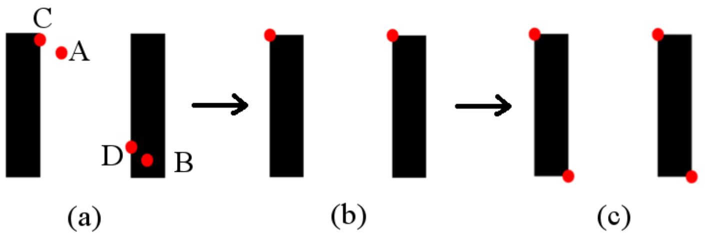


Fig. 5. Schematics illustration of obtaining corner points on the barcode

achieved by comparing two distances from point 0 to points 1 and 3, and the two points whose distance is the longest are 0 and 3. The method is correct when the barcode image is taken from perpendicularity direction. However, if the barcode is observed at an angle degree, it may be failed. In the work, we use cross product to obtain clockwise direction of corner points. As shown in Fig. 6, points $A(x_A, y_A)$ and $B(x_B, y_B)$ are on a line of the barcode; $C(x_C, y_C)$ and $D(x_D, y_D)$ are on another line of the barcode. It is supposed that index of A is 0 and index of B is 2. Then, the direction distance is given by

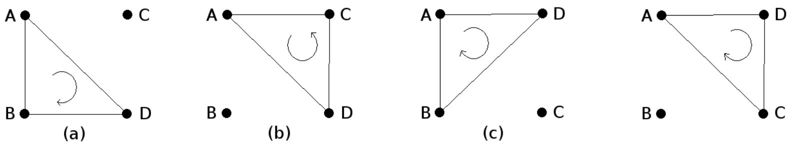


Fig. 6. Use cross product to obtain order of clockwise direction of corner points

$$D_{irection} = ((x_D - x_A)(y_B - y_A) - (y_D - y_A)(x_B - x_A)) \\ ((x_D - x_A)(y_C - y_A) - (y_D - y_A)(x_C - x_A)) \quad , \tag{1}$$

if $D_{irection} < 0$, the direction of the cross product of ADB is different from one of ADC, as shown (a) and (b) in Fig. 6, and index of D is 3 and index of C is 1; otherwise, if $D_{irection} > 0$, i.e., their directions are same, as shown as (c) and (d) in Fig. 6, index of C is 3 and index of D is 1. Thus, the order of the corner points is realigned, and new clockwise points are denoted as A', B', D' and C', which A' and C' is on one line of the barcode, and B' and D' is on another line of the barcode.

Then, we need to find the point which is left-top corner point and denote it as the first point. The left-top corner point can be confirmed by considering intersection among the numbers under barcode and two lines about A', C', B' and D'. The formulas are given by

$$P_1 = \xi P_{A'} P_{C'} \qquad P_2 = \xi P_{B'} P_{D'} \quad , \tag{2}$$

where $P_{A'}$, $P_{B'}$, $P_{C'}$ and $P_{D'}$ present points A', B', C' and D', respectively; ξ is a parameter to control detection of the extending lines and extending length, predefined as $0.075 < |\xi| < 0.15$ in the work; P_1 and P_2 are two extending points. If line P_1P_2 intersects the numbers and $\xi > 0$, A' is the left-top point; otherwise, if line P_1P_2 intersects the numbers and $\xi < 0$, D' is the left-top point, and its index is changed to 0 and indexes of other points are changed accordingly. An example is shown in (e) of Fig. 3. When P_1P_2 always intersects some black points whether ξ is greater than 0 or not, length between intersecting points is calculated to search the positions of the numbers. The line between two points with the longest length is considered to intersect the number. Thus, the corner points in order are obtained, shown in (f) of Fig. 3, which the first corner point is one end point of green line segment.

The proposed method can be used for a barcode on a plane surface. When a barcode is printed on a curved surface, there are two failed situations: first, if the curvature of the curve surface is observed too big and only part of the barcode is observed, it may be failed to find the four corner points with our method; second, P_1P_2 may be intersected by other shapes but not the numbers. Thus, in the work, the barcode is considered on a plane surface or a curve surface with small curvature.

After ordered the corner points, one-to-one relationship between two sets of corner points of two barcodes can be obtained, which one is recoded in image coordinates and the other is recoded in real-world coordinates. Then, extrinsic parameter of translation data (X, Y, Z) and rotation angle (*roll*, *pitch*, *yaw*) between image data and the data in the world coordinate can be calculated if camera calibration is obtained [11]. In this work, we use the image processing library OpenCV [12] to obtain the camera calibration and calculate the extrinsic parameter. Thus, all points on the observed objects can be obtained from relative points on images with

$$s \cdot m = A \cdot [R|t] \cdot M \qquad , \tag{3}$$

where s is scale factor; m is two-dimensional point; A is camera intrinsic parameter; $[R|t]$ is extrinsic parameter including translation data (X, Y, Z) and rotation angle (*roll*, *pitch*, *yaw*); M is three-dimensional point. Furthermore, robot can use the output data to control the observed objects with the barcodes.

3 RT-Component of Obtaining Pose of Barcode

3.1 RT-Middleware and OpenRTM-Aist

RT-Middleware (Robot Technology Middleware) developed by AIST (Advanced Industrial Science and Technology) is a common platform standard to operate robots with the distributed object technology [13,14]. Generally, robot technology is applied not only for robots in factory environments, but also in human environments. With RT-Middleware, a common platform can be established based on distributed object technology that supports the construction of various networked robotic systems with the integration of RT-Components which are various network enabled robotic elements. Modularization of RT-Components and RT-Middleware can raise the efficiency of robotic study and development, and furthermore, the RT target field can be expanded and a new robot market can be made. In any RT-Components, there are some parts including a main body which is activated as the main process unit and a few of input ports (InPort) and output ports (OutPort) which are data stream ports as interfaces.

OpenRTM-aist which have been developed and distributed by AIST is an implementation of the RT-Middleware interface specification and RT-Component object model. OpenRTM-aist is used to developed components of RT and run the components with a development frame work, manager and a set of tools. Since RT-Components made by OpenRTM-aist use same communication protocol, partial components can be replaced by new components without effecting other components working in the same system. Furthermore, the components can be reused in another robot system and so can reduce researching time.

3.2 Component of Obtaining Pose of Barcode

In this work, we use OpenRTM-aist to realize the components, including three parts, as shown in Fig. 7: one is the *CLUEReader* component which evaluates pose of a barcode with the mentioned method in above section; another is the *Imager* component which provides images from camera; the other is the *ObjectId-Provider* component that is used to obtain data from barcode database. Imager component and ObjectIdProvider were developed by other robot developers.

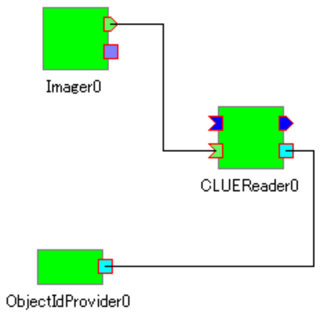


Fig. 7. Components in our reading barcode system

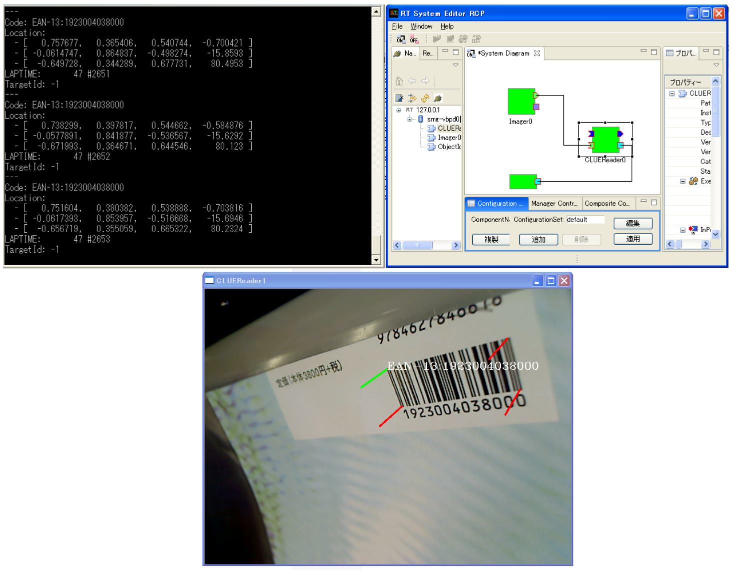
The CLUEReader component has four ports which include two input ports obtained different data from images, a service consumer which obtains information from database of marks that is barcode here, and an output port which provides information of barcodes. According to the CLUEReader component design, the data through output port, i.e. *OutPort*, is as follows: translation data (X, Y, Z) and rotation angle $(roll, pitch, yaw)$.

4 Evaluation Experiment

Several experiments are described in this section to show how the components work, with two species of barcodes at different poses. The camera to be calibrated is a 25-mm lens to obtain a clear image, the resolution of which is 640*480. With the OpenCV functions, the camera calibration and extrinsic parameter which is used to calculate the relative relation between the image coordinates and real-world coordinates can be obtained. Translation data and rotation angle data are calculated based upon four corner points of a barcode. Size of a used barcode is measured in advance in order to calculate right positions of corner points on screen, since different barcode has different size.

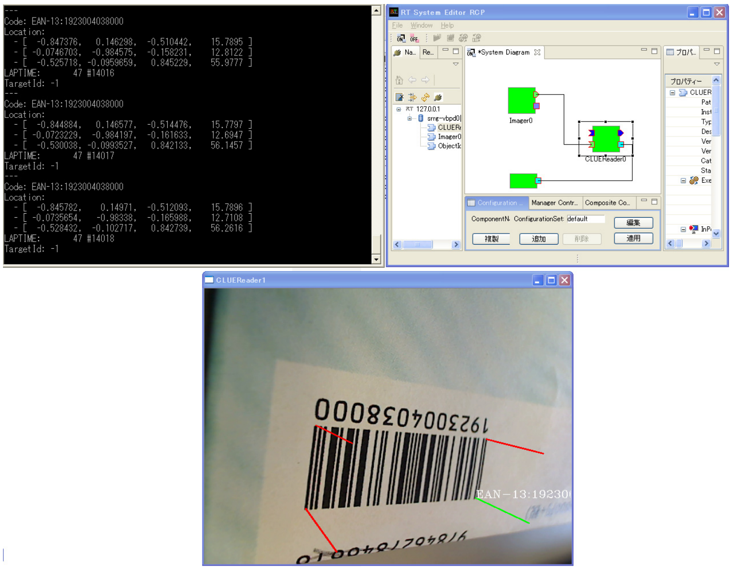
With the OpenRTM-aist software, there are some windows shown on screen, including the OpenRTM window, windows about components of Imager, ObjectIdProvider and CLUEReader. Some windows including our work are shown on screen, and other windows are hided. Information of the observed barcode is recoded under *Location* in the window of output data. The first three columns are rotation angle data (*roll*, *pitch*, *yaw*), and the last column is translation data (X, Y, Z).

The barcodes in Fig. 8 and Fig. 9 are same but observed at different angles. The green line is extended from the first corner point, and the three red lines are extended from the other three corner points. Their directions show the barcode's pose direction, and furthermore, present the object's pose direction. Fig. 10 is



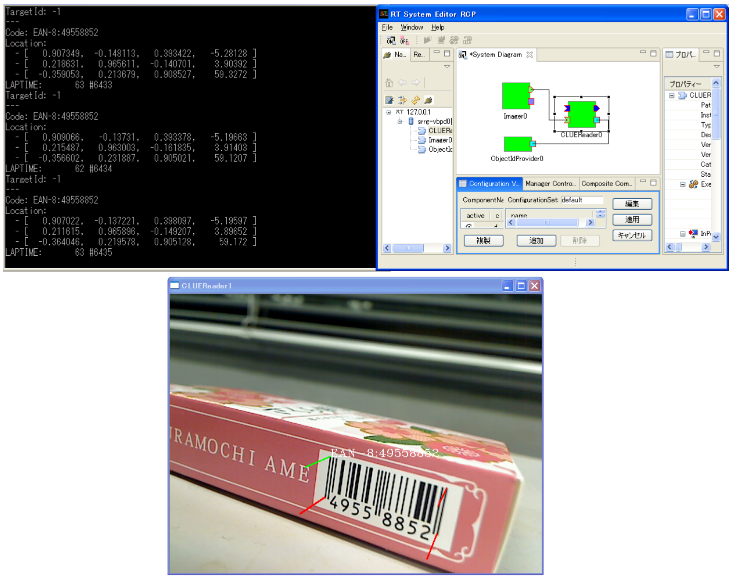
```
---
Code: EAN-13:1923004038000
Location:
  - [   0.757677,   0.365406,   0.540744,  -0.700421 ]
  - [ -0.0614747,   0.864837,  -0.498274,   -15.8593 ]
  - [  -0.649728,   0.344289,   0.677731,    80.4953 ]
LAPTIME:       47 #2651
TargetId: -1
---
Code: EAN-13:1923004038000
Location:
  - [   0.738299,   0.397817,   0.544662,  -0.584876 ]
  - [ -0.0577891,   0.841877,  -0.536567,   -15.6292 ]
  - [  -0.671993,   0.364671,   0.644546,     80.123 ]
LAPTIME:       47 #2652
TargetId: -1
---
Code: EAN-13:1923004038000
Location:
  - [   0.751604,   0.380382,   0.538888,  -0.703816 ]
  - [ -0.0617393,   0.853957,  -0.516668,   -15.6946 ]
  - [  -0.656719,   0.355059,   0.665322,    80.2324 ]
LAPTIME:       47 #2653
TargetId: -1
```

Fig. 8. The left-top window shows output data; the right-top window shows the system including the components; and in the bottom window, four lines are drawn to describe position, direction, and pose of the barcode



```
---
Code: EAN-13:1923004038000
Location:
  - [  -0.847376,   0.146298,  -0.510442,    15.7895 ]
  - [ -0.0746703,  -0.984575,  -0.158231,    12.8122 ]
  - [  -0.525718, -0.0959659,   0.845229,    55.9777 ]
LAPTIME:       47 #14016
TargetId: -1
---
Code: EAN-13:1923004038000
Location:
  - [  -0.844884,   0.146577,  -0.514476,    15.7797 ]
  - [ -0.0723229,  -0.984197,  -0.161633,    12.6947 ]
  - [  -0.530038, -0.0993527,   0.842133,    56.1457 ]
LAPTIME:       47 #14017
TargetId: -1
---
Code: EAN-13:1923004038000
Location:
  - [  -0.845782,    0.14971,  -0.512093,    15.7896 ]
  - [ -0.0735654,   -0.98338,  -0.165988,    12.7108 ]
  - [  -0.528432,  -0.102717,   0.842739,    56.2616 ]
LAPTIME:       47 #14018
TargetId: -1
```

Fig. 9. Barcode is as same as one in Fig. 8, which is different position, pose, and observed angle



```
TargetId: -1
---
Code: EAN-8:49558852
Location:
  - [   0.907349,  -0.148113,   0.393422,   -5.28128 ]
  - [   0.218631,   0.965611,  -0.140701,    3.90392 ]
  - [  -0.359053,   0.213679,   0.908527,    59.3272 ]
LAPTIME:       63 #6433
TargetId: -1
---
Code: EAN-8:49558852
Location:
  - [   0.909066,   -0.13731,   0.393378,   -5.19663 ]
  - [   0.215487,   0.963003,  -0.161835,    3.91403 ]
  - [  -0.356602,   0.231887,   0.905021,    59.1207 ]
LAPTIME:       62 #6434
TargetId: -1
---
Code: EAN-8:49558852
Location:
  - [   0.907022,  -0.137221,   0.398097,   -5.19597 ]
  - [   0.211615,   0.965896,  -0.149207,    3.89652 ]
  - [  -0.364046,   0.219578,   0.905128,     59.172 ]
LAPTIME:       63 #6435
```

Fig. 10. Another barcode is different from ones in Fig. 8 and Fig. 9

another kind of barcode, and the result is similar as Fig. 8 and Fig. 9. These examples are all real-time, and the average time of calculating a pose is about 0.0474 second.

5 Conclusion

In the paper, we proposed a method for robot to obtain poses of observed objects. The method is achieved by finding corner points of barcodes on the observed objects and calculating their order. Finally, we use OpenRTM-aist to create a corresponding component for the method. The method can help robot rapidly obtain the information of its target objects since the information in a barcode is less than the one in an image. The component system is easily applied to other existing robot systems based on RT-middleware implementation. Furthermore, developers can easily add new components into the system to achieve themselves robot systems since all RT-middleware components are independent from others.

Acknowledgment. This work is supported by NEDO (New Energy and Industrial Technology Development. Organization, Japan) Intelligent RT Software Project.

References

1. Ohara, K., Sugawara, T., Lee, J.H., Tomizawa, T., Do, H.M., Liang, X., Kim, Y.S., Kim, B.K., Sumi, Y., Tanikawa, T., Onda, H., Ohba, K.: Visual Mark for Robot Manipulation and Its RT-Middleware Component. Advanced Robotics 22(6-7), 633–655 (2008)
2. Grigorescu, S.M., Macesanu, G., Cocias, T.T., Puiu, D., Moldoveanu, F.: Visual Robust camera pose and scene structure analysis for service robotics. Robotics and Autonomous Systems 59, 899–909 (2011)
3. Lang, S., Kleinehagenbrock, M., Hohenner, S., Fritsch, J., Fink, G., Sagerer, G.: Providing the basis for human-robot-interaction: A multi-modal attention system for a mobile robot. In: Proceedings of IEEE International Conference on Multimodal Interfaces, pp. 28–35 (2003)
4. An, S.Y., Kang, J.G., Choi, W.S., Oh, S.Y.: A neural network based retrainable framework for robust object recognition with application to mobile robotics. Appl. Intell. 35, 190–210 (2011)
5. Lowe, D.G.: Object recognition from local scale-invariant features. In International Conference on Computer Vision (ICCV), Corfu, Greece, pp. 1150–1157 (1999)
6. Lowe, D.G.: Distinctive Image Features from Scale-Invariant Keypoints. International Journal of Computer Vision (IJCV) 60(2), 91–110 (2004)
7. Ozen, S., Bouganis, A., Shanahan, M.: A fast evaluation criterion for the recognition of occluded shapes. Robotics and Autonomous Systems 55, 741–749 (2007)
8. Belongie, S., Melik, J., Puzicha, J.: Shape Matching and Object Recognition Using Shape Contexts. IEEE Transactions on Pattern Analysis and Machine Intelligence 24(24), 509–522 (2002)
9. He, L., Han, C.Y., Everding, B., Wee, W.G.: Graph matching for object recognition and recovery. Pattern Recognition 37(7), 1557–1560 (2004)

10. ZBar bar code reader, http://zbar.sourceforge.net/
11. Zhang, Z.: A flexible new technique for camera calibration. IEEE Trans. Pattern Anal. Mach. Intell. 22, 1330–1334 (2000)
12. OpenCV, http://opencv.willowgarage.com
13. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Yoon, W.: RT-Middleware: Distributed Component Middleware for RT (Robot Technology). In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2005), pp. 3555–3560 (August 2005)
14. Ando, N., Suehiro, T., Kotoku, T.: A Software Platform for Component Based RT-System Development: OpenRTM-Aist. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 87–98. Springer, Heidelberg (2008)

WorkCellSimulator: A 3D Simulator for Intelligent Manufacturing

Stefano Tonello[1], Guido Piero Zanetti[1], Matteo Finotto[1], Roberto Bortoletto[2], Elisa Tosello[2], and Emanuele Menegatti[2]

[1] IT+Robotics srl
Contra' Valmerlara 21, Vicenza, Italy
{stefano.tonello,piero.zanetti,matteo.finotto}@it-robotics.it
[2] Intelligent Autonomous Systems Laboratory
Department of Information Engineering (DEI)
University of Padua, Italy
{bortolet,toselloe,emg}@dei.unipd.it

Abstract. This paper presents WorkCellSimulator, a software platform that allows to manage an environment for the simulation of robot tasks. It uses the most advanced artificial intelligence algorithms in order to define the production process, by controlling one or more robot manipulators and machineries present in the work cell. The main goal of this software is to assist the user in defining customized production processes which involve specific automated cells. It has been developed by IT+Robotics, a spin-off company of the University of Padua, founded in 2005 from the collaboration between young researchers in the field of Robotics and a group of professors from the Department of Information Engineering, University of Padua.

Keywords: Simulator, Manufacturing, Industrial Robotics, Off-line Robot Programming Tools.

1 Introduction

Simulation has a long-standing tradition in robotics by providing a useful tool for testing ideas on a virtual robot in a virtual setting before trying it on a real robot in a real environment. If a few years ago, simulation was mainly used in specific sub-communities like swarm robotics and robot learning, in the last decade, simulation in robotics is getting more attention again thanks to the computation power of computers which has been increasing significantly making now possible to run computationally intensive algorithms on Personal Computers (PC) instead of special purpose hardware. In this evolution, particular attention should be paid to the role that simulation tools play in the industrial robotics area. In the last twenty years these instruments are becoming more and more diffused thanks to their capability of improving manufacturing quality, accuracy and profitability. In fact, searching for the actions which optimize the process directly on the real robot implies loss of time and money: simulation does not degrade or damage the machinery (saving of money) and lets the robot to be used in other processes during the off-line programming (saving of time). Simulation results are optimal collision-free paths to be performed by the robot in the real world with consequent increase of productivity and decrease of maintence costs of the plant. A study conduced by The Electrical Power Research Institute (EPRI) estimates a payback of about three months. The saving are attribuited to reduced training costs, costs caused by environmental excursions, damage to equipment and improved plant availability [1]. An example can be the bending process of metal sheets: this process is often carried out automatically through work cells in which there are robot manipulators, pressbrakes, and other machines useful for this purpose. In order to obtain a metal sheet folded as desired is necessary to properly program the robots and the other machines. This is an expensive process both in terms of time and programming errors which can bring the machinery to collide with other system components, causing damage to them. In order to solve these problems, the authors propose a robot simulator software, called WorkCellSimulator, that allows to manage an environment for manufacturing tasks. Despite of most existing robot simulators, which functions are difficult to change to achieve the various requirements robot users have (as it is similary explained in [2]), the software presented was created with the primary aim of helping the user through the planning of bending process of metal sheets, but then, as we shall see later, it was extended and nowadays it is currently employed in many different fields: similarly to Robcad [3], it fully integrates core technologies and a powerful set of process-specific applications addressing a wide range of manufacturing processes including spot welding, arc welding, laser and water-jet cutting, painting and spraying, and material handling. At the same time, it supports the simulation of many types of robots. In this sense it differs from the major industrial robotics simulator, such that developed by KUKA, ABB or Fanuc [4–6], which are fully dedicated to a specific robotic manipulator. In detail, WorkCellSimulator has been developed by using the most advanced artificial intelligent algorithms, and motion and task planning approaches used to automatically generate the free obstacles trajectories, which optimize a given objective function (execution time, effort on the part, etc.) [7–10]. Moreover, its architecture allows the definition of algorithms for advanced task planning, such as for generating the optimal folding sequence in metal sheet bending process [11]. This features give the product easy-to-customize capabilities. WorkCellSimulator has been developed by IT+Robotics[1], a spin-off company of the University of Padua, founded in 2005 from the collaboration between young researchers in the field of Robotics and professors from the Department of Information Engineering, University of Padua. The mission of IT+Robotics is to increase the flexibility of industrial processes by transforming the latest results of the academic research into industrial solutions. IT+Robotics products aim at autonomy. After deploying IT+Robotics' products the need for human intervention is reduced and, in several cases, altogether eliminated. This is achieved exploiting: machine vision systems, robot motion planning algorithms and arti-

[1] http://www.it-robotics.it/

ficial intelligence techniques, which were an exclusive asset of academic research up to now.

This paper is structured as follows. In Section 2, some use cases are described. In Section 3, an introduction to the software architecture is provided with a detailed description of the functionalities. In Section 4, the main areas of application in which the WorkCellSimulator has been used are presented. Finally, in Section 5, a brief discussion about the possible future development closes the paper.

2 Use Cases

After an analysis of the requirements that a simulation-oriented application should have and the exam of the manufacturing processes, the main use cases has been extracted in order to design an appropriate software architecture. The studies performed have identified two major workflows: ***design of a work cell***, and ***simulation of the process***. During the work cell design, the user must be able to import 3D models of components or portions thereof, and to provide all the collateral information necessary for a complete definition of the component. The collection of this information depends on the type of object (e.g. for the robot manipulator will be must specify the configuration of the joints, their constraints and dynamic parameters). The result of this process is the creation of a library of components, which will be used for the design of the work cell. In fact, the objective of the design of the work cell is the positioning of the components in such a way that faithfully reflects the real plant. To this end a set of calibration tools has been made available to the user. After the creation of the work cell, the user can assess whether it fulfill final user requirements. E.g., the assessment can include verification of robot reachability of main components, suitability of robot grasping tecnology or evaluation of cycle time. The created work cell will be used for the simulation of the process. Within the simulation, the customer interacts with the software in order to define all stages of the process. Considering as example the metal sheet bending process, the first step is the definition of the product bend, using 2D or 3D CAD data. Once the process targed has been defined, the design of the bending process begins. The process of bending is composed of the following phases:

- **Loading the piece**. The robot manipulator, after checking the actual presence, takes the piece from the loader;
- **Alignment of the piece**. The robot places the part inside of the aligner, then takes it again. The aim of this step is to determine the position of the piece as correctly and repeatable as possible;
- **Execution of the fold**. For each bend to be made, the phases are the following:
  - ***Selection machine tooling***. The user, based on the fold to be made, selects the press brake tool station;
  - ***Overturning***. In the event that the product is not currently taken by the robot from the correct side for the next bend to be made, the robot carries the piece in a overturner, and then takes it in the opposite side;

  - ***Positioning in press brake***. The piece is brought into the press brake by the robot. In this phase, the piece is better aligned using feedback from the press brake's back gauges;
  - ***Folding***. The press brake does the fold of the piece. At this stage the robot can be inactive, or it can follows the movement of the piece during the bending;
  - ***Takes the parts***. The robot takes the part if it has been inactive in the previous step, and then leads the product in a safe position outside the area of the press;
- **Palletizing**. After being folded, the product is placed inside the pallet. The arrangement layout within the pallet can be configured by the user.

After the simulation, the results obtained are used by a post-processor with the goal of generating the robot program corresponding to the simulated process. The obtained program will be loaded into the robot for the real execution, possibly after having been manually modified by the user. Although the two streams of work are the most frequently sequential, they should be made iterative, e.g. in order to further configure the work cell during the simulation of the bending process.

From the previous discussion, the following actors have been identified:

- **Components Builder**. The user designs the components present in the simulation. The components are placed in libraries to be used within the work cells;
- **Workcell Builder**. The user uses components developed by the *Component Builder* to define the work cell layout;
- **Product designers**. The user defines the product.
- **Simulation User**. The user performs off-line programming of the work cell, defining in detail all the movements that the robot has to perform and the signals that are exchanged between the machineries. This actor has deepest knowledge of the problem domain, and therefore it is able to exactly determine the best necessary operations.
- **Robot Programmer**. The information given to the simulation for the process are used by the post-processor to generate the robot code.

The main use cases are identified:

- **3D Rendering**. It allows viewing of the scene by the end user;
- **Design of the components**. The *Component Builder* interacts with the software to define the various components used in the work cell;
- **Design of the work cell**. *Workcell Builder* uses components created by *Component Builder* to define the work cell;
- **Product design**. The product designer interacts with the software to define the product itself.
- **Performing simulations**. The *Simulation User* manages a simulation run within the work cell;

- **Design of the process**. The *Simulation User* interacts with the software to define the process. A scripting engine has been introduced in order to provide to the end-user a customization of the behavior of the simulator up to the minimum detail;
- **Robot code generation**. The software uses the simulation results in order to generate the robot code. The generated code may be changed manually by the robot programmer and finally sent to the robot for the real execution of the simulation.

The result of this study phase was a set of useful information for the next step: the design and development of a software architecture that was able to manage the use cases and the main actors described above, but that was also flexible and reusable in other industrial situations.

3 Architecture

The software architecture of WorkCellSimulator (WCS) can be divided in two macro blocks (as shown in figure 1):

- **WCS Core**. It contains the modules which implement the basic functionality for the simulation.
- **WCS Applications**. It contains the modules and the applications which compose the WorkcellSimulator suite.

While WCS Applications use the modules contained in WCS Core, WCS Core can be considered completely independent from WCS Applications itself.

In the following section the two macro blocks are separatelly described. Later, it will be shown how to verticalize application behavior through the development of plug-in modules.

3.1 WCS Core

WCS Core contains the following features:

- Management of 3D and 2D geometric models, using *Modelling Engine* based on OpenCasCade open-source library [12];
- Manages the rendering of the scene, using the *Rendering Engine* module. The rendering engine is based on OpenSceneGraph [13];
- Management of collisions, using internally developed high performance *Collision Engine* module;
- Verticalisation of the functionality provided by the rendering and collision modules for the simulated work cell, through the *World Manager* module;
- Management of the kinematic chains and their simulation using the *Numerical Kernel* module. This is the main component: it manages the kinematics, the trajectories and also contains the motion planning engine;
- Management of the graphical interface and communication between different components, using the *Application Framework* module.

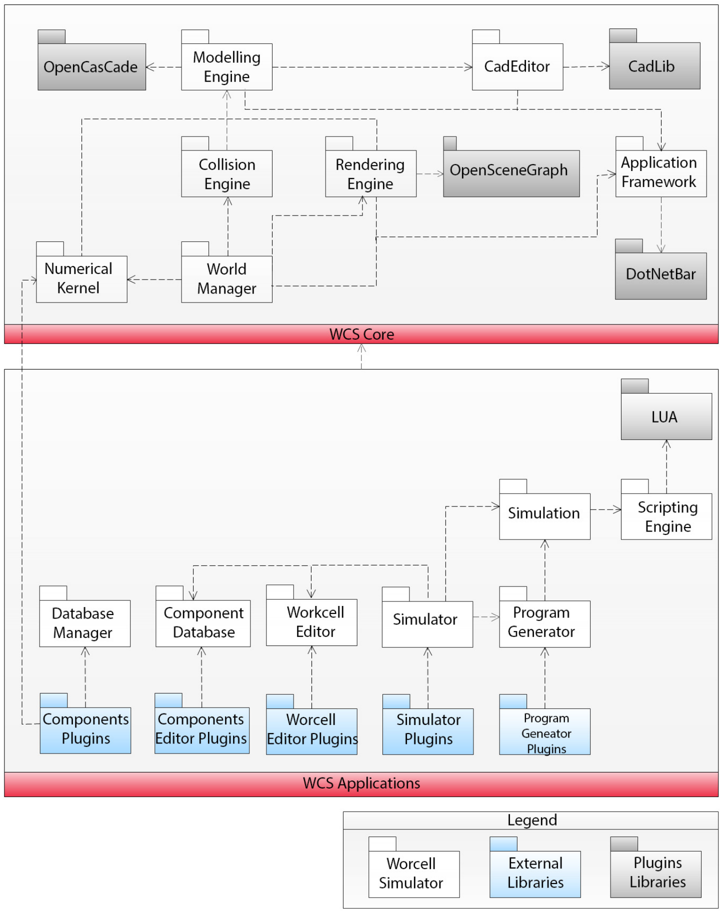


Fig. 1. A schematic view of the software architecture

The modules contained in WCS Core comprise three libraries:

– **C++ Library** for the management of low-level logical and numerical algorithms;
– **C++/CLI Library** for the management of the communication between the C++ native environment and the .NET one. The libraries contain simple wrapper to the functionality exposed by the C++ libraries.
– **C# Library** for the management of the high level application logic, persistence management, and graphical user interfaces.

It worth noting that this organization allows to unify the benefits of the two programming environments: the computational efficiency of C++, which allows you to make better use of optimization tools such as SSE or the parallelization through OpenMP [14], and the ease of programming for the definition of high-level logic and graphical user interfaces using C#. The exception is the Application Framework module, which, having to manage the graphical interfaces, is formed by a single C# library.

3.2 WCS Applications

The WorkCellSimulator suite consists of five main components:

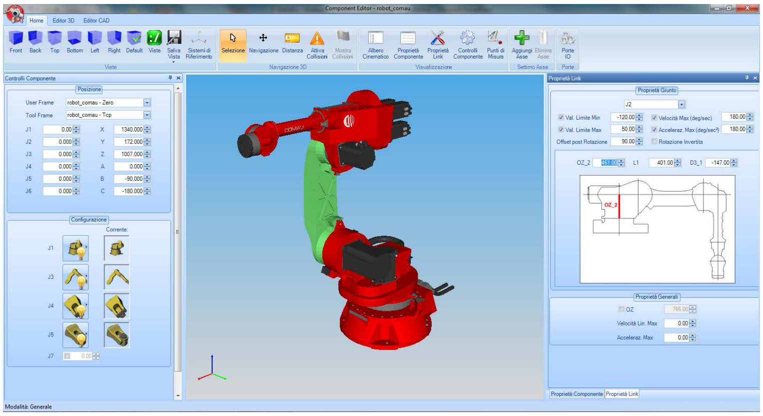
- **Database Manager**. Management of distributed database containing components, work cells, and simulation projects;
- **Component Editor**. Editor for simulation components, extensible throught plug-in modules. A general purpouse plug-in allows the definition of components as kinematic chains. Custom plug-ins can be developed for a specific work cell component using provided SDK, allowing to tailor user interfaces to ease usage (figure 2(a)).
- **Workcell Editor**. Editor for work cells. It allows the insertion and positioning of components into the work cell. Precise positioning can be archived using custom calibration procedures (figure 2(b)), defined in plug-in modules. Through the use of plug-in approach, editor behaviour configuration can be customized for a particular process or specific user needs;
- **Simulator**. This is the main application of the suite, used to perform actual off-line programming. This package provides the application user interface that allows to simulate the various components and their interactions. Plug-in modules can be defined using provided SDK to tailor user interfaces and software behaviour. Given the provided input from the user, process trajectories and signals are generated by means of an internal scripting engine based on LUA programming language. The use of scripting engine leads to two main benefits. First, process logic can be defined interacting with simulated 3D environment, using a command-line console or debugging previously defined scripts (figure 2(c)). Second, since process logic is defined in plain text code using a simply to use programming language, final simulation user can fully customize process behaviour to include product-specific optimizations or to handle a particular product requirement. As shown, the choice of a scripting engine allows customization of the software behaviour at every level and to the highest extent.
- **Program Generator**. Generates machine controller code given results from the simulation. This is a two step process. In the first stage, simulation data is read to generate an internal, vendor independent, rappresentation of the controller code. This step can be customized using provided SDK. This flexibility is required since, for simulation convenience, simulation execution flow can be different than real world execution flow. In the second stage, one of available post-processor are used to generate actual robot code based on current robot vendor and controller version. Program Generator manages also direct comunication with robot, using e.g. FTP, to upload created robot programs.

By leveraging the capabilities offered by the *Application Framework* module, developers can define the applications by integrating the various components in a single application of by creating an application for each one. Particularly, in the current setting, four separated applications was created. This choice provides the advantage of creating an application which allows a clear differentiation of users' type (e.g. the system integrator uses the *Component Editor* to define the components of the plant and the *Workcell Editor* to define the work cell, while the end-user uses only the *Simulator* to define the work process). On the other hand, it is possible to integrate the various components in a single application. This choice has the advantage of facilitating the workflow definition of the end user.

3.3 Plug-in Modules

As has been shown in the previous sections, WorkCellSimulator can be fully customized by mean of plug-in modules. It is possible to define:

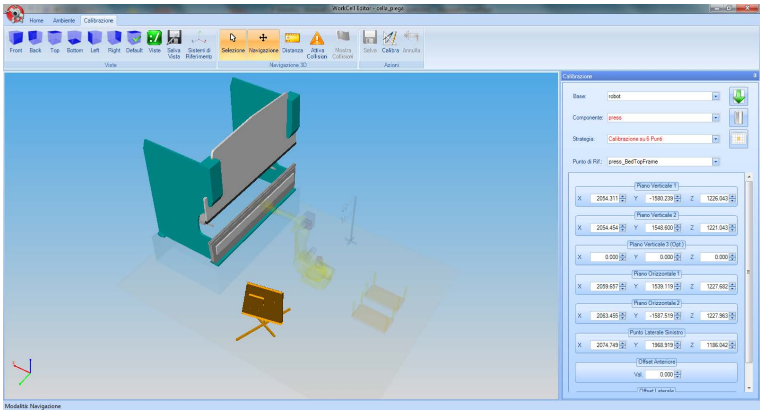
- C# Plug-in modules for the high-level logic associated with the component, e.g. to define a parameterization of the component or to define component specific behaviour.
- C++ Plug-in modules associated to a component which allow to customize:
    - The computation of the direct or inverse kinematics. Algorithms for computing direct and inverse kinematics given kinematic chain definition are already available in *Numerical Kernel* but a closed form is useful both to improve efficiency and to add new extended features. E.g., in the case of a manipulation robot, it is convenient to create a plug-in in order to customize the definition of the inverse kinematics. An inverse kinematics query for manipulator robots usually has 16 valid solutions. Using the plug-in, the user is able to constraint robot configuration or to choose one configuration over the others;
    - The calculation of the trajetory interpolation, in order to faithfully replicate the behavior of the robot controller during the simulation;
    - The calculation of the deformation of the 3d model during the simulation process;
    - The validation of the state of the kinematic chain. It can be usefull to define the special constraints in the positioning of the joints (standard axis limits are already defined by numerical kernel).
- C# Plug-in modules for the definition of a component editor;
- C# Plug-in modules for the definition of the user interface for the editor for a particular type of work cell and its calibration;
- C# Plug-in modules for the definition of the process logic. These modules are used to specialize system behavior during simulation of a particular process, allowing to customize components' behavior and interaction based on current process. The scripting engine allows a further customization of the process by the end user;
- C# Plug-in modules to create internal rappresentation of controller code.



User Frame: robot_comau - Zero
Tool Frame: robot_comau - Tcp
J1 0.00 X 1340.000
J2 0.000 Y 172.000
J3 0.000 Z 1007.000
J4 0.000 A 0.000
J5 0.000 B -90.000
J6 0.000 C -180.000

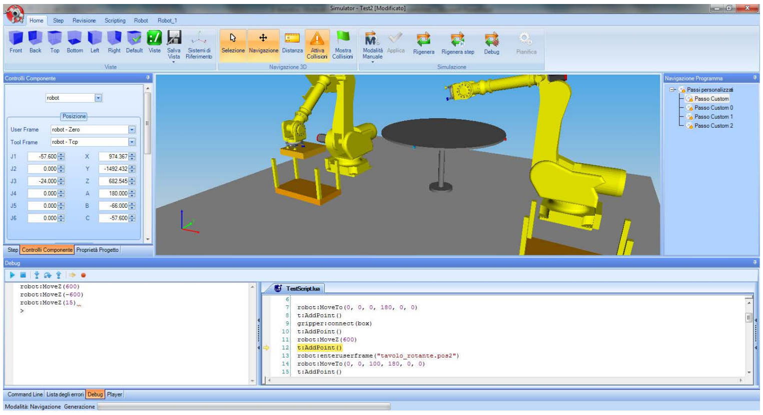
J2
Val. Limite Min -120.00 Velocità Max (deg/sec) 180.00
Val. Limite Max 50.00 Acceleraz. Max (deg/sec²) 180.00
Offset post Rotazione 90.00 Rotazione Invertita
OZ_2 461.00 L1 401.00 D3_1 -147.00
OZ 766.00
Velocità Lin. Max 0.00
Acceleraz. Max 0.00

(a)



Base: robot
Componente: press
Strategia: Calibrazione su 6 Punti
Punto di Rif.: press_BedTopFrame
Piano Verticale 1: X 2054.311 Y -1580.239 Z 1226.043
Piano Verticale 2: X 2054.454 Y 1548.600 Z 1221.043
Piano Verticale 3 (Opt.): X 0.000 Y 0.000 Z 0.000
Piano Orizzontale 1: X 2059.657 Y 1539.119 Z 1227.682
Piano Orizzontale 2: X 2063.455 Y -1587.519 Z 1227.963
Punto Laterale Sinistro: X 2074.749 Y 1968.919 Z 1186.042
Offset Anteriore: Val. 0.000
Offset Laterale

(b)



robot
User Frame: robot - Zero
Tool Frame: robot - Tcp
J1 -57.600 X 974.367
J2 0.000 Y -1492.432
J3 -24.000 Z 682.545
J4 0.000 A 180.000
J5 0.000 B -66.000
J6 0.000 C -57.600

```
robot:MoveZ(600)
robot:MoveZ(-600)
robot:MoveZ(15)
>
```

```

robot:MoveTo(0, 0, 0, 180, 0, 0)
t:AddPoint()
gripper:connect(box)
t:AddPoint()
robot:MoveZ(600)
t:AddPoint()
robot:enteruserframe("tavolo_rotante.pos2")
robot:MoveTo(0, 0, 100, 180, 0, 0)
t:AddPoint()
```

(c)

Fig. 2. Screen shots of WCS user interfaces

– C# Plug-in modules for creating post-processor program to generate the code for a given language and controller version.

In this case it worth noting that the WorkcellSimulator suite already contains default procedures for the management of components, work cells, the off-line programming and code generation. Plug-ins modules are used to tailor simulator to specific requirements.

4 Applications

WorkcellSimulator is currently used in companies typically belonging to the industrial automation sector, with application ranging from simple material handling to metal sheet bending. WCS Core is also applied to two products of IT+Robotics: Smart Pick 3D and SafePath.

Smart Pick 3D highlights the integration between robotics and vision: it is a software for the visual identification of the position of items within the production line, solving the well-known problem of bin-picking. Using standard cameras and sophisticated image processing algorithms, images are acquired and processed in order to determine product 3D position inside a bin and ensure the correct system operation under environmental light changes. The resulting positions are used to control the movements of the robot, so that it can pick the items to be processed from a bin, even when all the pieces are randomly placed. This application allows the robot to be completely autonomous: it handles the pieces as they come from previous processing phases, avoiding the use of ad-hoc loaders. The integration of Smart Pick 3D and WorkcellSimulator is two fold. WCS Core is used inside Smart Pick 3D to generate collision-free trajetories for the robot for product grasping. The other way round, a plug-in for the simulator has been developed to allow parametrization of loading process using vision based on Smart Pick 3D.

SafePath is designed for the off-line programming of numerical control machines (CNC): the software allows the definition of new processes through a 3D simulated environment created using WorkCellSimulator. SafePath allows the creation of programs from scratch or the modification of the programs generated by CAM (*Computer-Aided Manufacturing*) softwares. Through an off-line programming the machine can continue the old production while the new process is generated; moreover, on one hand SafePath verifies, in real time, the movement programmed for the machine, detecting possible collisions. The user is therefore able to identify and correct any programming errors. On the other hand the software verifies the accuracy of the processing, ensuring that the production is within the desired tollerance ranges.

Among others, with this work IT+Robotics is also involved in a European Project called ThermoBot. The aim of ThermoBot is the design of an autonomous robotic system for thermo-graphic detection of cracks[2]. WorkCell-Simulator is used to simulate the process of quality inspection using path and

[2] Available from: http://thermobot.eu/

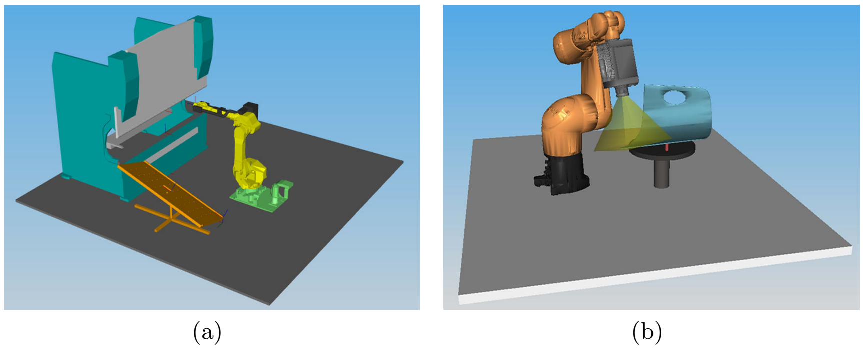


Fig. 3. Typical configurations of work cell environments

motion planning algorithms. The algorithms of path planning are able to determine, given the product, the set of points that the robot must reach in order to ensure the correct execution of the process. These points are then supplied to the algorithm of motion planning for generating a collision-free trajectory able to meet the timing constraints from the process of thermography. Moreover, during the execution, an online motion planning module, based on WorkcellSimulator, will be used to correct generated paths to optimize image contrast.

Figure 3 depicts two examples of WCS usage: in 3(a) the simulation of the bending process of metal sheets is represented, in 3(b), instead, a robot detecting cracks is shown according to the ThermoBot project.

5 Conclusion and Future Work

WorkCellSimulator confirms the mission of IT+Robotics: increasing the flexibility of industrial processes. The modular architecture allows the simulation of any kind of process and the scripting system lets the customization of the process itself to the highest extent. The software highlights the will of understanding the customers' needs of reliability and simplicity: it makes the programming easier from the most complex work cell to the single machine. WCS can be used with major robotic manipulation industrial platforms, and it's really easy to customize it to work with custom-made controllers. It allows to operate completely off-line with respect to the work cell, thus allowing to minimize work cell on-line programming phase. As previously described, once a work cell was modeled, it allows you to plan and simulate the entire production process, generating the corresponding robots' controller code.

For the future, It+Robotics aims to continue its research activity in order to improve the motion planning algorithms currently used: the integration of time constraints within the motion planning is one of its goals, usefull to improve its contribution inside the ThermoBot project; moreover, the company looks at the resolution of the problem of multi-robot motion planning in order to manage more complex work environments. In a long-term view, the integration between simulation and vision is eligible through the integration between WorkCellSimulation and ViDeE (Vision DEvelopment Environment), a software framework producted by IT+Robotics. Moreover, two projects are in progress in order to provide the optimality of the manufacture starting from simulation. The first one allows to use 3D models inside the simulation of bending process. It is Unfolding 3D: its goal is the development of an algorithm able to recognize the folds contained in the 3D input model, in order to convert the product into a 3D unfolded model usable, with the accessory information collected, as input to the simulator. The other aims, instead, to develop a search algorithm in order to find the optimal sequence of bends, to make programming of bending work cells fully automatic.

References

1. Hosseinpour, F., Hajihosseini, H.: Importance of Simulation in Manufacturing. World Accademy of Science, Engineering and Techonology (2009)
2. Tokunaga, H., Matsuki, N., Sawada, H., Okano, T., Furukawa, Y.: A robot simulator for manufacturing tasks on a component-based software development and execution framework. In: Proc. IEEE ISATP 2005 (2005)
3. http://www.plm.automation.siemens.com/en_us/products/tecnomatix/robotics_automation/robcad/index.shtml
4. http://www.kuka-robotics.com/en/products/software/kuka_sim/
5. http://www.abb.com/roboticssoftware
6. http://www.fanucrobotics.com/products/vision-software/ROBOGUIDE-simulation-software.aspx
7. Chirikjian, G., Amato, N., Kavraki, L. (eds.): Special issue: Robotics techniques applied to computational biology. International Journal of Robotics Research (2004) (to appear)
8. Kavraki, L.E., Svestka, P., Latombe, J.C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Transactions on Robotics and Automation 12(4), 566–580 (1996)
9. Latombe, J.C.: Motion planning: A journey of robots, molecules, digital actors, and other artifacts. The International Journal of Robotics Research - Special Issue on Robotics at the Millennium 18(11), 1119–1128 (1999)
10. Carpin, S., Pillonetto, G.: Motion Planning Using Adaptive Random Walks. IEEE Transactions on Robotics 21(1) (February 2005)
11. Duflou, J.R., Vancza, J., Aerens, R.: Computer aided process planning for sheet metal bending: A state of the art. Computers in Industry 56(7), 747–771 (2005) 01663615
12. http://www.opencascade.org
13. http://www.openscenegraph.org
14. http://openmp.org/wp/

A Meta-model and Toolchain for Improved Interoperability of Robotic Frameworks

Johannes Wienke, Arne Nordmann, and Sebastian Wrede

Research Institute for Cognition and Robotics, Bielefeld University, Germany

Abstract. The emerging availability of high-quality software repositories for robotics promises to speed up the construction process of robotic systems through systematic reuse of software components. However, to reuse components without modification, compatibility at the interface level needs to be created, which is particularly hard if components were implemented in different robotic frameworks. In this paper we propose an approach using model-based techniques for improving component reusability. We specifically address data type compatibility in a structured way through the development of a generic meta-model capable of representing data types from different frameworks and their relations. Based on this model a code generator emits serialization code which makes it possible to seamlessly reuse the existing data types of different frameworks. The application of this approach is exemplified by connecting the YARP-based iCub simulation with a component architecture using a current robotics middleware. Based on our experiences we describe requirements on robotics frameworks to further increase the level of interoperability between available components.

1 Introduction

In order to increase the usefulness of robots, they need to be equipped with a multitude of capabilities, which need to be reasonably combined into a working system. While many capabilities have already been implemented on different robots, their integration into a single system is still an open problem. A successful integration relies on an appropriate design of the functional architecture but often also more technical aspects slow down and complicate the development of integrated systems. As such, a major practical issue is the diversity of existing development frameworks like ROS [1], YARP [2], OROCOS [3] or OpenRTM-aist [4]. Even though most of the recent frameworks use a component-based approach and hence are composed of generally reusable building blocks, the created components can only be reused easily within the framework they have been developed for. This is caused by the lack of interoperation features in most frameworks. Recent development efforts tried to approach this problem, in particular by equipping frameworks with exchangeable transport layers, e.g. as done in OpenRTM-aist [5] with a ROS transport. Also YARP and OROCOS now have ROS transports. While this is a step towards interoperability, several issues still remain. Besides being able to use another framework's protocol, full interoperability on the transport level requires using foreign nameservices and introspection mechanisms. Otherwise, components imported from a foreign framework expose restrictions in their usability compared to native components and developers need to add specific exceptions for these components. On a more conceptual level, different transport semantics can prevent interoperability, e.g. if remote procedure call-based interfaces conflict with event-based communication. Another issue is the incompatibility of data types between different frameworks. Many frameworks developed type libraries with comparable semantics but using different approaches for Interface Definition Languages (IDLs), serialization schemes, and client APIs. This effectively prevents the reuse of components across frameworks. Even if transport-level communication is established, either (de-) serialization would fail or client components would be unable to deal with foreign data types. This commonly results in exceptions in the architecture like adapter and bridge components. Such solutions introduce inefficiencies and require manual work. Hence, a more structured approach to deal with data type incompatibility is required to fully make use the achieved transport-level interoperation support. Such an approach has to be easy maintainable and needs to be efficient to preserve the reactivity of the robot system.

In this paper we introduce an approach, termed Rosetta Stone, to deal with incompatible data types. It employs a generic meta-model for data representation and combines it with code generation. Section 2 starts with a discussion of how the interoperability issue has been addressed in other frameworks so far. Afterwards, we outline the conceptual ideas of our approach in Section 3. Based on a use case described in Section 4 our implementation will be explained in Section 5 and gained experiences and results of the application in the use case are detailed in Section 6. Finally, we conclude in Section 7 including a set of requirements which should be fulfilled by future frameworks to support data type compatibility.

2 Related Work

Interoperability between software components implemented in different robotics frameworks is usually achieved by applying classical software design patterns such as bridge, wrapper or protocol translator [6]. While it is reasonable to apply these techniques to specific interoperability problems, it imposes significant challenges for the software design of larger robotic systems if applied in the general case. For instance, using a dedicated bridge introduces performance penalties due to the additional reading, (de-) serialization and writing of data from and to connectors which are bound to the different frameworks. Furthermore, bridges are often specific to single data types and need to be kept up-to-date with the target types of both frameworks. As a consequence, bridges or wrappers are constantly out of date [5]. Instead, a more native level of interoperation between different frameworks is highly beneficial to achieve efficient solutions. In the following, we analyze different frameworks for features they provide towards native interoperability. As the representation of data is the key focus of this paper, we specifically examine these issues.

Recently, the Robot Operating System (ROS, [1]) has gained a lot of attention in the robotics community. It combines a middleware layer for communication with an extensive component collection, especially for mobile robots. Communication is statically typed and ROS comes with a collection of types based on a custom IDL. A compiler generates programming language types from these descriptions as well as framework-specific serialization code. While the transport layer in ROS can be exchanged and different implementations exist, the serialization code cannot be varied by the user, hence limiting interoperation with other frameworks.

A second widely used framework in robotics is OROCOS [3]. It has exchangeable transport implementations, e.g. CORBA, mqueue as well as a ROS transport. Additionally, different data representations can be used, called *typekits*, which are exchangeable through a plugin system. The traditional typekit is based on user-level C++ class definitions and a compiler builds serialization code for them by parsing the C++ code. Recently, support to interoperate with ROS has been added through the respective transport and a new typekit which has to be used by client components instead of the traditional one. This typekit is based on the ROS IDL to provide compatibility with ROS and the remaining OROCOS transport implementations cannot be used with these types. While transports and typekits are exchangeable, their choices are coupled and modifications to client components are required.

Another framework with exchangeable transports is OpenRTM-aist [5]. It provides features for hard real-time control and originally uses CORBA for data exchange between its components. Data types are defined through the CORBA IDL and in the case of the ROS transport, a duplicated definition of the respective type using the ROS IDL is required in order to provide the required serialization code to the ROS transport. The reuse of existing data types in both frameworks is not discussed so far and no mapping between them is explained in publications.

YARP [2] is a framework mainly used for the iCub robot. In contrast to the aforementioned frameworks, communication in YARP is dynamically typed, employing a custom data representation (*bottles*) and serialization approach. The framework has exchangeable transports and a partial ROS implementation is available. To combine the dynamic typing of YARP with the static ROS messages, the ROS transport comes with a specific compiler which generates YARP-serializable classes from the ROS IDL definitions, which effectively replaces the existing type system visible to the client components and hence requires modifications to them.

Summing up, while most recent frameworks provide ways to connect with other frameworks on a transport level, the way how data types are handled in these cases is not clearly structured. Often, special data types need to be chosen based on the transport decision because for most frameworks there is a strong connection between the transport, the applied serialization mechanism, and the exposed user-level data type API. This restricts the possible uses cases of the interoperation features and increases the development overhead when using such features. The issue of how to deal with semantically compatible but differently expressed data types, i.e. how to map between them, is completely neglected so far.

3 The Rosetta Stone Approach

In order to provide native data-level interoperability between different robotic frameworks we have developed a generic approach to mediate between the different technologies, which will be described on a conceptual level in the following paragraphs. The approach aims at a native integration inside the frameworks for high efficiency without the need for manual development of bridge components or a dependency on foreign frameworks. We assume that each of the potentially relevant robotic frameworks uses a structured and consistent approach of producing and consuming serializable data to be sent over a network connection. This means that a unified API for data holder classes and consistent serialization schemes for transmitted data exist, e.g. based on an IDL, so that at least inside each framework a level of generalization with respect to the data access is possible. Once this assumption is fulfilled, data communication in robotic frameworks fundamentally varies in four aspects: a) *the programming language used to produce or consume data*, b) *the API of the used data holders in this language*, c) *the (de-) serialization scheme applied to the data found in the data holders*, and d) *the transport mechanism used to communicate the serialized data*.

Our approach is based on a *meta-model* which describes data from the various robotic frameworks in an abstract and unified way, but including the aforementioned variables. Once the meta-model is populated with data types from different robotic frameworks it generates serialization code. The generated code uses the serialization scheme of one framework (A) on the network level and data types from another one (B). Assuming that framework B has exchangeable transports and serialization mechanisms, client applications written against this framework can connect with framework A by plugging in the generated serialization code and the transport for framework A. As a result, these applications can continue to use their native data types and no modifications to them are required. By using generated serialization code we can reach a high performance without requiring a dynamic use of the meta-model at runtime and without a dependency on the foreign framework's libraries (and hence increased compilation efforts) to import their serialization implementation. Moreover, no unnecessary serialization or conversion step is required as in the case of bridges. Finally, the mapping of different data types becomes a configuration aspect of the robotic system instead of being hard-coded.

3.1 Data Representation Features

A first question that arises for the proposed approach is whether it is possible to treat data from all recent frameworks in a common way, and if so, which features for data representation are required in the meta-model. For this purpose we analyzed the available representation features of common IDL-based serialization mechanisms (and hence statically typed) as well as the ones of 3 robotics-relevant dynamically typed systems[1]. The results of this analysis can be found in Table 1, indicating that besides some variations, a common and limited feature set available in most solutions can be identified, which is feasible to represent. On this basis we constructed a meta-model which represents the majority of the found features and is hence applicable for a broad variety of data types. In its essence, the meta-model represents data types as composed structures of named fields, comparable to the structure of most IDLs and programming languages.

[1] In our analysis we focused on features for the description of data types. Additional features found in several IDLs like the possibility to describe RPC interfaces are ignored.

Table 1. A matrix of features commonly found in different IDLs / dynamically typed data serialization solution. Features not applicable for dynamically typed solutions are marked in gray.

	Signed 64bit Integers	Unsigned 64bit Integers	Double Type	ByteBlob Type	Null Type	Enums	Constants	Variable Length Arrays	Multidimensional Arrays	Fixed Length Arrays	Maps	Optional Fields	Default Values	Unions	Message Nesting	Data Type Inheritance	Namespaces	Typedefs	Any Type
Static Typing																			
Protocol Buffers	x	x	x	x		x		x				x	x		x		x		
ROSmsg	x	x	x	x			x	x		x							x		
LCM		x	x	x			x	x	x	x							x		
Message Pack IDL	x	x	x	x		x		x			x	x	x			x			
Apache Thrift		x	x	x		x	x	x			x	x	x				x		
Apache Avro IDL		x	x	x	x	x		x	x	x	x		x	x			x		
Apache Etch		x	x	x		x	x	x	x		x					x			
OMG IDL	x	x	x	x		x		x	x	x				x			x	x	x
Dynamic Typing																			
YARP			x	x		x		x		▨	x	▨	▨	x	▨	▨	▨	▨	▨
ALValue			x	x				x		▨		▨	▨	x	▨	▨	▨	▨	▨
JSON	x	x	x		x			x		▨	x	▨	▨	x	▨	▨	▨	▨	▨

Currently supported data type features are: (signed/unsigned integers with different bit sizes, floats, strings, blobs, structs, arrays, unions.

3.2 Required Mapping Capabilities

Even though data types can be represented in a common meta-model, compatible types from different robotic systems still need to form separate entities in the meta-model, because for the code generation we need to be able to distinguish between them. Moreover, field names and representations usually do not exactly match between two semantically compatible types from different frameworks. E.g., a field might be called `angle` in one framework using radians but in the second framework it is called `phi` and based on degrees. For these reasons the meta-model also needs to contain a *mapping* between types which relates different fields and converts some representational differences. While the ability to map fields of different names is an essential requirement, the question arises which further capabilities are usually required. To find out an initial set of these capabilities that copes with the majority of cases, we analyzed the mapping of messages from the ROS `common_msgs` package to and from our own data types defined in the RST library [7]. For this purpose, we first decided which data types of one framework are semantically compatible with data types from the second, to define a set of realistic mapping candidates. For each candidate we then manually decided which operation categories are required to achieve a mapping. The results can be found in Table 2. Categories were chosen to reflect semantically related operations[2], namely: *Arithmetic*: basic mathematical operations (always selected when units were not clear from the type description, e.g. rad vs. degrees); *Array reorder*: shifting of entries of

[2] For brevity we excluded the obvious category of mapping varying field names.

Table 2. Operations likely to be required when converting between ROS and RST types

ROS	Arithmetic	Array Reorder	String Manipulation	Image Compression	Predicate Assignment	Subtype Loops	Rejection	Generators	Date Manipulation	RST
CompressedImage.msg				x	x			x?		Image.proto
Image.msg		x			x			x?		Image.proto
JointState.msg	x					x				JointPositionState.proto
JointState.msg	x									JointAngles.proto
JointState.msg	x									JointTorques.proto
JointState.msg	x					x				ProprioceptionState.proto
JointTrajectory.msg	x					x				Point2DTimeseries.proto
JointTrajectoryPoint.msg	x					x				Point2DTimestampPair.proto
KeyValue.msg			x				x			KeyValuePair.proto
Odometry.msg	x									Pose.proto
Path.msg	x					x		x		Point2DTimeseries.proto
Point.msg	x									Vec3DDouble.proto
Point.msg	x									Vec3DFloat.proto
Point.msg	x									Translation.proto
Point32.msg	x									Vec3DDouble.proto
Point32.msg	x									Vec3DFloat.proto
Point32.msg	x									Translation.proto
PointCloud.msg	x					x				PointCloud3DFloat.proto
Polygon.msg	x					x				PointCloud3DFloat.proto
Pose.msg	x									Pose.proto
Pose2D.msg	x					x				Pose.proto
Quaternion.msg	x									Rotation.proto
RegionOfInterest.msg	x									BoundingBox.proto
TimeReference.msg	x							x	x	Timestamp.proto
Transform.msg	x									Pose.proto
Vector3.msg	x									Vec3DDouble.proto
Vector3.msg	x									Vec3DFloat.proto
Vector3.msg	x									Translation.proto
Wrench.msg	x									Wrench.proto

any sequence-like container, e.g. to swap planes in image types; *String manipulation*: common string operations; *Image compression*: to decode compressed images; *Predicate assignment*: assignment to target type values based on logical predicates; *Subtype loops*: looping on and conversion of nested subtypes (composition); *Rejection*: abort translation in case of a dynamically detected incompatibility (e.g. unsupported image encoding); *Generators*: generation of values not found in the source type, e.g. constant value or loop variables; *Date manipulation*: conversion of date formats.

Starting with the total 65 message definitions of ROS and 65 message types in RST (by coincidence), 29 possible mapping candidates were found[3]. The results of this analysis are depicted in Table 2. For the majority of the mapping candidates arithmetic operations and the possibility to handle and convert collection-like contents (e.g. a list of floats to a list containing special `JointValue` objects) sufficed to achieve a mapping of all contents that can be represented in both messages. A notable but

[3] In ROS messages types often exist once without a header containing timestamps and if necessary a second time including this header. We did not consider the timestamped messages for the mapping if the comparable version without timestamp was already a valid candidate.

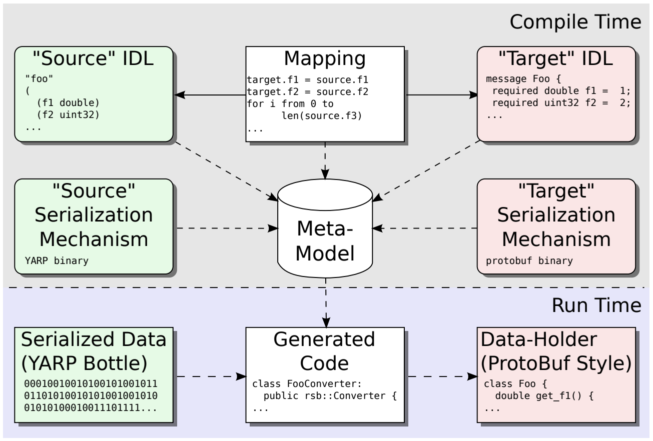


Fig. 1. User-supplied contents of the meta-model used to create serialization code and the application of this serialization code

important exception is the mapping of image types. The definitions in both frameworks are rather complex involving different byte representations, image depths, channels etc. While the meta-data representing these different image types can be mapped with the aforementioned operations, the actual image data needs to be manipulated if no matching representation modes exist. This requires a more fine grained operation on the byte array contained in both message types.

We also analyzed a mapping from RST to data types used in YARP-based systems [2] and vice versa. Here the situation is different, as no formal message definitions exist. To find out commonly used message formats we analyzed the data continuously sent by the simulator for the iCub robot. For publisher-based data two semantically different types could be identified:

1. messages containing joint angle information as a list of doubles
2. messages for images with the following format (Lisp-like notation):

```
((VOCAB mat) (VOCAB rgb) ((INT 3) (INT 230400)
 (INT 8) (INT 320) (INT 240)) (BLOB 230400))
```

A mapping of the first message type requires arithmetic operations and potentially generators (e.g. to fill out the joint names which are not given in the bottle) and subtype loops to convert the angle list to more specific types or vice versa. For the image type the same remarks are valid as in the ROS case. Additionally, for controlling the robot, messages in the following format are sent over RPC-based channels:

```
((VOCAB set) (VOCAB poss)
 ((DOUBLE 0.0) (DOUBLE 0.0) (DOUBLE 0.0)))
```

These messages are comparable to the joint angle lists except that also YARP vocabulary items need to be generated.

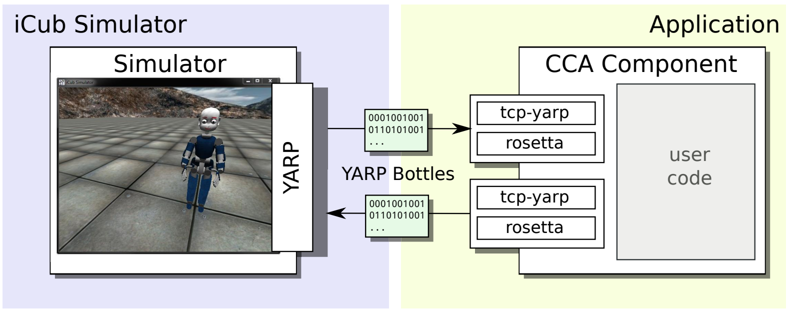


Fig. 2. Use case, connecting CCA components to the iCub simulator by using the Rosetta framework to interchange proprioceptive sensor feedback and an image stream from the iCub's internal cameras.

Based on the aforementioned observations our meta-model contains mapping abilities for the most common operations. This is realized by defining each mapping through a Lisp-like code block with a restricted feature set that allows facilitates the static code generation.

3.3 Application of the Meta-model for Code Generation

For being able to generate serialization code from the meta-model two additional aspects must be contained in the model. These are the data holder APIs and the serialization schemes. The API needs to be known to provide the native interface to client applications while the serialization scheme is required to provide transport-level compatibility of data types through the correct serialization scheme. Figure 1 summarizes the required data in the meta-model (upper part, supplied by the user) and how the data can be used to generate deserialization code for binary data received from a foreign framework. Summing up, by using a generic meta-model for the unified representation of data types and their relation we are able to create code that efficiently translates serialized data from one robotic framework to the native data holder API in another framework. As a result, frameworks are connected without requiring manually written bridge components and the native interfaces in both middlewares are preserved. This prevents changes in components and increases their reusability.

In the following sections we are now going to introduce a use case where the connection of different frameworks is beneficial. Afterwards we will describe the actual implementation of the Rosetta Stone system and its application for the use case.

4 Use Case

As a an exemplary use case for the aforementioned approach, we decided to integrate the iCub simulator from the RobotCub project into our component architecture CCA (*Compliant Control Architecture* [8]), which is based on the RSB middleware [7]. The use case is driven by the AMARSi[4] project, where CCA was developed and the iCub

[4] http://amarsi-project.org

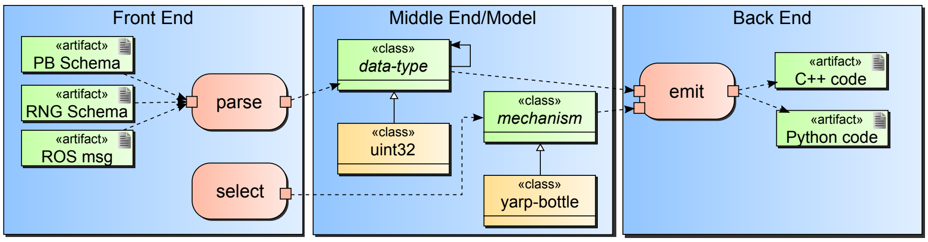


Fig. 3. Architecture overview for the Rosetta implementation

is used as one of the main robot platforms. However, development of the robot was done in a predecessor project and diverging requirements resulted in the existence of two frameworks. To be able to conduct experiments with CCA components on the iCub simulator, the Rosetta approach allows CCA components to communicate over YARP, the middleware used by the iCub simulator. For this purpose joint angles and camera image need to be transferred, which are two diverse but very common data types in robotics, hence reflecting a wide variety of possible applications.

In more details, the iCub simulator transmits images and joint angles as YARP bottles and CCA uses protocol buffer representation from RST for exchanging data between different processes. Therefore we need to translate between these two data representations. The anticipated result is that this is possible without changes in the implementation of the simulator or in CCA components, i.e. through configuration. Moreover, performance should not be degraded, which is specifically challenging for the image types with a significant payload size.

5 Rosetta Implementation

The Rosetta approach is realized as a compiler toolchain implemented in Common Lisp. As shown in the static view in Figure 3, the Rosetta implementation is structured as a traditional compiler with frontend, middleend and backend.

The frontend parses data type specifications expressed in existing IDL languages into meta-model elements resolving dependencies between type specifications. The current

Listing 1.1. Excerpt from the mapping specification for the iCub simulator joint angles

```
import "rst/kinematics/JointAngles.proto"
import "bottle-structure-icub-torso-command.bottle-schema"

data-holder: rst.kinematics.JointAngles
wire-schema: yarp.icub.torso.command

unpack-rules:
  len(.angles) = 3
  .angles[0] = .angles.a0
  .angles[1] = .angles.a1
```

implementation supports parsing of protocol buffer, LCM and ROS IDL definitions. Similarly, mapping specifications are processed by this component. Additionally, syntactic as well as basic semantic checks (e.g., duplicate field names or unresolved forward references) are performed by the parsers.The middleend manipulates meta-model elements encompassing data types, mappings and serialization mechanisms. According to the desired code generation target (either data holders or serialization code) suitable intermediate representations are generated. The backend generates output based on the intermediate representation and language-specific templates for different programming languages. Currently supported are C++, Python and Common Lisp[5]. Generated code artifacts include data holder classes mimicking a native API (e.g., protocol buffer) and serialization code for different robotic frameworks (e.g., ROS or YARP).

To provide a dynamic view of the Rosetta implementation a typical sequence of processing steps in relation to the use-case introduced in Section 4 is as follows:

1. One or more IDL files are read and parsed by the frontend (in the use-case: files containing RST types `JointAngles` and `Image`, and the descriptions of the corresponding YARP bottle structures[6]) .
2. Optionally, a mapping description is read (in the use-case: a mapping for joint-angle types and a mapping for image types, see Listing 1.1).
   At this point, data types and mappings are represented in the meta-model.
3. For the given serialization mechanism and data types the transformation rules specified in the mapping are applied (in the use-case: respective mapping rules for joint-angle types and image types).
   At this point, abstract intermediate code implementing the mapping and (de-) serialization has been generated.
4. A template for the given target language is instantiated and populated (in the use-case: C++ protocol buffer API templates).
5. The template is expanded and the result written into output file(s) (in the use-case: C++ code in separate files for data-holder and serialization code).

The resulting serialization code needs to be used by the framework which attaches to a foreign one, ideally through configuration changes.

6 Gained Experiences

Our use case served as a first qualitative evaluation of how the Rosetta framework improves the compatibility between different robotic frameworks, without the need for changes or adaptations on the implementation level in one of the involved frameworks or applications. In this case we evaluated what and how much work had to be done in i) the iCub simulator, ii) Rosetta and iii) the application (CCA components).

As anticipated, the iCub simulator was left untouched for integration with the CCA application. We completely relied on the YARP bottle format and serialization for joint angles and images, as required by the simulator. For Rosetta, we had to specify the

[5] For Common Lisp output of the middleend is directly passed to the integrated compiler.
[6] Cf. Section 6.

mapping between YARP bottles and the domain-types used in CCA. As YARP does not provide static types and a declarative syntax for them, we had to add such a schema description for YARP in Rosetta, in order to address fields from the YARP bottles in the mapping. Afterwards, mappings were defined between the YARP bottle format for images and joint angles, and the RST types used in CCA. Within the CCA application, the transport and the data converter for network communication had to be changed through RSB's configuration mechanism. The relevant ports of the involved CCA components had to be configured to publish over and listen to the `tcp-yarp` transport that is available as an RSB extension. Furthermore, the generated Rosetta serialization code (generated from IDL and mapping specifications) had to be registered for incoming and outgoing YARP image bottles and YARP joint angle bottles.

In summary, after specifying the Rosetta mappings between data representations and generating the serialization code, no changes were required for the simulator and necessary changes on CCA-side were limited to configuration aspects.

In addition to this qualitative evaluation we also analyzed the performance of the generated serialization code within the use case. For this purpose we compared the deserialization of a binary encoded YARP bottle containing double-precision joint angles using Rosetta and with the native YARP implementation. Rosetta deserialized to C++ RST types while YARP used the native bottle classes in the C++ API. The encoded message had a size of 200 bytes and was deserialized 1.000.000 times. On a Linux desktop computer with an Intel Xeon 8 core processor at 2.4 GHz the YARP implementation required 3.96 s *real time* whereas our own deserialization code needed 0.13 s. Test programs were compiled using GCC and the O2 optimization level. The huge performance boost can can be explained by the fact that Rosetta has an additional schema of the transmitted data (see above), while the native YARP implementation is completely dynamic. As a consequence, the Rosetta-generated deserialization code can skip many checks and decoding of several bytes that need to be deserialized by the native implementation to dynamically find out the structure of the bottle. This performance allows to receive simulation results in the use case without any performance degradation.

7 Conclusion

Nowadays, the middleware layers of most modern robotic frameworks allow a direct embedding of multiple transports acting as connectors to other frameworks. Furthermore, many of the current frameworks already employ a code generation approach where the necessary client API classes and serialization code are generated. However, still most of the frameworks lack a clear separation of concerns regarding the type representation at user-level and the resulting serialization format which is beneficial to achieve interoperability without component modification. Moreover, frameworks do not consider type mapping and transformation as an important concern.

The Rosetta approach proposed in this contribution addresses these aspects in order to achieve native interoperability between components of different robotics architectures. Based on an analysis of features commonly found in different IDLs and necessary mapping operations between a set of typical robotic data types expressed in different representations, required capabilities for a type mapping language were identified in this contribution. On this basis a meta-model for representing types and mapping functions was developed, which allows to generate several code-level artifacts for native robotic system interoperabiliy. The introduced approach is so far unique, because none of the frameworks addressed native type mapping and transformation routines in the process of generating serialization code for seamless interoperability.

Scientifically, a common representation for data types and their mappings will facilitate, e.g., analysis and comparison of different data types used in current robotic systems. Practically, the developed approach not only allows better interoperability and thus reusability of software components across robotic frameworks but also contributes to achieve integration within a single large-scale ecosystem such as ROS given the number of semantically equivalent but syntactically different robotics data types. Future work will concentrate on the development of a domain specific language which eases the specification of mappings and transformations between data types in robotic systems. Moreover, additional ways of applying the compiler architecture will be evaluated. E.g., in cases where frameworks lack interchangeable transports and serialization mechanisms Rosetta can generate serialization-to-serialization code for efficient and configuration-defined bridge components.

Acknowledgments. This work was partially funded by the European FP7 projects HUMAVIPS (grant no. 247525) and AMARSi (grant no. 248311). Many thanks to Jan Moringen for his work on the Rosetta toolchain.

References

1. Quigley, M., et al.: ROS: an open-source Robot Operating System. In: ICRA Workshop on Open Source Software (2009)
2. Metta, G., Fitzpatrick, P.: YARP: yet another robot platform. Journal on Advanced Robotics 3(1), 43–48 (2006)
3. Soetens, P.: A Software Framework for Real-Time and Distributed Robot and Machine Control. PhD thesis, Department of Mechanical Engineering, Katholieke Universiteit Leuven, Belgium (2006), http://www.mech.kuleuven.be/dept/resources/docs/soetens.pdf
4. Ando, N., Suehiro, T., Kotoku, T.: A Software Platform for Component Based RT-System Development: OpenRTM-Aist. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 87–98. Springer, Heidelberg (2008)
5. Biggs, G., Ando, N., Kotoku, T.: Native Robot Software Framework Inter-operation. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 180–191. Springer, Heidelberg (2010)
6. Buschmann, F., Henney, K., Schmidt, D.C.: Pattern-Oriented Software Architecture: A Pattern Language for Distributed Computing, vol. 4. Wiley, Chichester (2007)
7. Wienke, J., Wrede, S.: A middleware for collaborative research in experimental robotics. In: IEEE/SICE International Symposium on System Integration (SII 2011), Kyoto, Japan. IEEE (2011)
8. Nordmann, A., Rolf, M., Wrede, S.: Software Abstractions for Simulation and Control of a Continuum Robot. In: Ando, N., Brugali, D., Kuner, J., Noda, I. (eds.) SIMPAR 2012. LNCS, vol. 7628, pp. 113–124. Springer, Heidelberg (2012)

Integrated Software Development for Embedded Robotic Systems

Sebastian Wätzoldt, Stefan Neumann, Falk Benke, and Holger Giese

Hasso Plattner Institute for Software Systems Engineering
University of Potsdam, Germany
{sebastian.waetzoldt,stefan.neumann,holger.giese}@hpi.uni-potsdam.de,
falk.benke@student.hpi.uni-potsdam.de

Abstract. In the recent years, improvements in robotic hardware have not been matched by advancements in robotic software and the gap between those two areas has been widening. To cope with the increasing complexity of novel robotic embedded systems an integrated and continuous software development process is required supporting different development activities and stages being integrated into an overall development methodology, supported by libraries, elaborated tools and toolchains. For an efficient development of robotic systems a seamless integration between different activities and stages is required. In the domain of automotive systems, such an overall development methodology, consisting of different development activities/stages and supported by elaborated libraries, tools and toolchains, already exists. In this paper, we show how to adapt an existing methodology for the development of automotive embedded systems for being applicable on robotic systems.

1 Introduction

In novel robotics applications steady improvements in robotic hardware is not matched by advancement in robotic software leading to an increasing gap between those two areas. The increasing complexity of modern robotic systems requires to further support several different software development activities such as modeling, simulation and testing that allow the incremental development of robot systems, starting with a single sensor and resulting in a complex application. Elaborated tools and toolchains are required to support the different activities and integrate them into an overall and well structured development methodology. To realize an efficient software development process, on the one hand, one has to provide libraries supporting individual development activities at different levels, e.g., at the level of individual sensors and control functions or at the level of systems or sub-systems, being incrementally composed. On the other hand, a seamless migration between individual development activities and stages has to be achieved. Furthermore, one crucial aspect that needs to be considered for a large portion of robotic systems is real-time behavior.

Accordingly, the following aspects need to be considered for bridging the gap between hardware and software development in novel robotic systems: (I) An overall methodology is required that supports (II) different development activities like modeling, simulation and testing at (III) different stages, e.g., simulation, prototyping and (pre-)production. Such a methodology has to be supported by (IV) elaborated tools and (V) libraries integrated into (VI) an overall toolchain allowing a seamless migration between the different development stages and artifacts. (VII) Simulation and testing support is required for the stages, allowing to validate created functionality, developed sub-systems or systems, e.g., by providing executable functional models, simulation environments and plant models. (VIII) Last but not least, real-time constraints need to be reflected.

As an example, in the automotive domain large complex real-time embedded systems are developed using different development stages, e.g., simulation, prototyping, and pre-production. Advanced tools and libraries have emerged during the recent years, integrated into sophisticated toolchains supporting different development stages as well as a seamless migration between them. To deal with the increasing complexity and to further reduce software development costs as well as time, advanced frameworks for the distributed and component-based development have been developed. In this paper, we propose adapting the existing software development methodology used in the domain of automotive embedded systems to support the software development of novel, complex embedded robotic systems. The proposed methodology includes an overall development process consisting of tools included into an overall toolchain as well as libraries. We apply this existing approach to the domain of robotic systems and evaluate as a proof of concept, which modifications have to be made. The approach is evaluated using a mobile robot developed according to the adapted methodology. Special attention is given to real-time constraints that need to be considered in a slightly different way than in the case of automotive real-time embedded systems. Therefore, we show a new approach for combining hard and soft real-time behavior in the existing automotive framework.

The remainder of this paper is organized as follows. Section 2 briefly discusses the foundations of robotic as well as automotive systems and introduces a running example for this paper. Section 3 describes our development approach including different stages and highlights our used tools as well as simulation and verification possibilities. The paper discusses related work in Section 4 and concludes in Section 5.

2 Foundations - Robotic and Automotive Systems

2.1 Robot Laboratory

For the evaluation of our research activities, we use our MDELab robot laboratory consisting of three Robotino robots.[1] The robots can be equipped with several sensors (e.g., laser scanner, infrared (IR) distance sensors, GPS like indoor navigation systems) as well as different actuators (e.g., servo motors,

[1] www.festo-didactic.com

omnidirectional drive, gripper). The general idea of our evaluation scenario is the realization of a variable production setting, where robots are capable of transporting small pucks (representing goods in a production system) to different locations. Robots have to fulfill different requirement, e.g., they have to provide basic functionality like moving and avoiding obstacles in hard real-time (reacting on obstacles within a few milliseconds). Further, the robots have to reflect high level goals, e.g., energy saving of the battery, short routing to the destination points and optimizing the throughput while transporting the pucks. While basic functionalities, such as obstacle avoidance, have to be realized in hard real-time, we use existing libraries to realize higher functionalities such as path planning or creating a map by evaluating measured distance values. The latter can rarely be realized under hard real-time constraints because of insufficient libraries.[2] Furthermore, we run a RTAI Linux operating system[3] on the robot to enable hard real-time execution.

As a running example, we use a single robot with the following hardware/ software configuration: The robot has three wheels realizing an omnidirectional drive. The drive unit provides an incremental encoder to realize odometry functionality, which calculates the relative position over time according to the drive speed and the orientation of the omnidirectional drive of the robot. Due to the fact that this odometry calculation becomes more and more imprecise over time, we use an additional GPS like (*NorthStar*[4]) indoor navigation system to correct the position in the long run. IR distance sensors are used to avoid obstacles during movement. A more complex navigation logic uses these sensors for maintaining a map[2] as well as computing an appropriate route for the robot while avoiding obstacles.

2.2 Automotive Development Process

A commonly applied development process for the development of automotive embedded real-time systems according to [4] is depicted on the left in Fig. 2. The development process includes three different stages, namely the simulation, prototyping and pre-production stage. During the simulation stages models are extensively used for realizing control functionality as well as for representing the environment. At the prototyping stage, a transition from a model-based to a software centric development approach is realized. Often, this is achieved by using code generators that automatically derive source code from the models used in the previous stage. In the pre-production stage, more and more aspects of the real system are involved, e.g., by using prototyping HW including the processor type (with additional debugging support) that is later used. Furthermore, parts of the real plant enable a realistic validation of the real-time behavior.

[2] For path planning and creating a map the MRPT library is used (`www.mrpt.org`).
[3] `www.rtai.org`
[4] `www.evolution.com/products/northstar/`

2.3 AUTOSAR

The **AUT**omotive **O**pen **S**ystem **AR**chitecture was invented to further support the development of complex and distributed systems. AUTOSAR[5] is the new de facto standard in the automotive domain. It defines a layered architecture, standardized communication mechanism and a whole development methodology. Furthermore, it supports the interaction between different car manufactures and suppliers. Figure 1 gives an overview of the layered AUTOSAR architecture. The layer at the bottom represents the real hardware including microcontroller and communication busses. An abstraction layer on top of the real hardware, included in the basic software layer, offers standardized interfaces for accessing the HW. Further functionality realizing the OS behavior as well as functionality for realizing communication is included in the basic software layer. The AUTOSAR runtime environment (RTE) is responsible for realizing the communication from and to the top software application layer. Software components (SWCs) realize application functionality at the layer on top. There, the architecture style changes from a layered to a component based approach [12]. SWCs communicate over well-defined ports using AUTOSAR interfaces, which are realized by the RTE layer. Each SWC consists of an arbitrary number of so-called *Runnables* that specify the behavior entities of each component.[6] Such Runnable entities are mapped on OS tasks, which are scheduled and handled by the operation system included in the basic software layer.

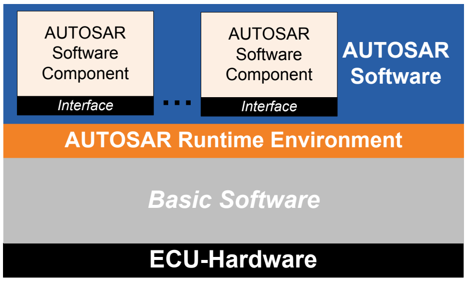

AUTOSAR Software Component
Interface
...
AUTOSAR Software Component
Interface
AUTOSAR Software
AUTOSAR Runtime Environment
Basic Software
ECU-Hardware

Fig. 1. The layered AUTOSAR architecture according to the specification in [12]

2.4 Automotive vs. Robotic Systems

In an automotive embedded system, usually applications are developed in such a fashion that hard real-time capable functionalities are separated from soft real-time applications. For example, it is quite common to deploy soft and hard real-time functionality on disjoint execution nodes and direct communication between them is avoided.

For robotic systems it is quite common to combine soft and hard real-time behavior into one application. For example, a mobile robot needs to avoid obstacles under hard real-time during navigation while calculating a route and updating a map. Both functionalities need to be combined while predicting the

[5] `www.autosar.org`

[6] The functionality of a Runnable can be realized by a C/C++ function.

execution time, e.g., of a route planing algorithm, is often not possible.[7] Thus, one difference between automotive and robotic systems concerning the real-time behavior is, that soft and hard real-time capable functionalities need to be more closely linked in robotic systems.

3 Development Environment

In this section, we describe our development environment, the tools and libraries used in the different development stages as well as our test and verification possibilities during system development. According to [4], we distinguish three development stages at different levels of abstraction targeting specific key aspects, namely simulation, prototyping and pre-production. Validation and verification activities are applied in each stage according to the given abstraction level. On the left in Figure 2, the overall process including the different stages is shown. In the following, we describe the applied validation and verification activities of each stage in the form of the libraries, methods and tools used. Furthermore, we show how to achieve an AUTOSAR conform system realizing the complex behavior of the robot incrementally developed, validated and verified during the different development stages.

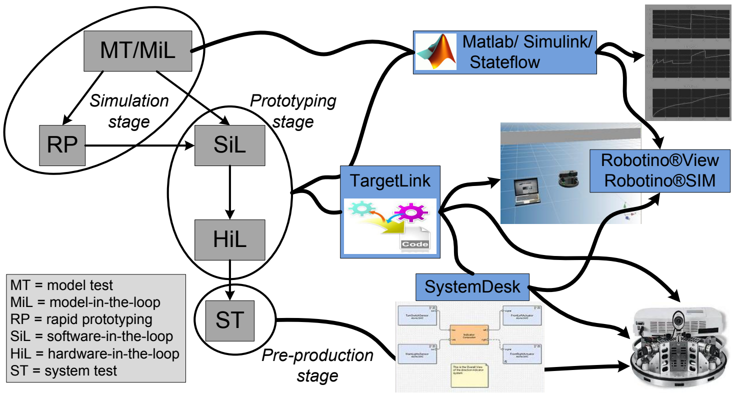


Fig. 2. On the left are the three development stages according to [4] in combination with our toolchain during software and system development on the right

[7] Execution time depends on the size of the map, which is usually not known before runtime.

3.1 Simulation Stage

Individual functions as well as composed behavior, resulting from multiple individual functionalities, are the subject of the simulation stage. Data flow models in the form of block diagrams (e.g., MATLAB/Simulink) usually in combination with control flow models like Statecharts (e.g. Stateflow) are used [6]. Normally, function development is done independent from platform specific limitations (memory capacity, floating point calculation or effects resulting from discretization). Additionally, environment specific signals and other real sensor values (e.g. produced by A/D, D/A converter or specific communication messages) are ignored for the sake of simplicity. The goal of the simulation stage is to prove that the functional behavior can work and as a result provides a first proof of concept for control algorithms.

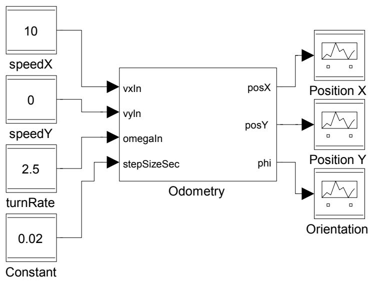


Fig. 3. Odometry in MATLAB, which calculates the position from fix drive speed and turn rate

As depicted in Fig. 2 and according to the aspect (IV), we mainly use the MATLAB tool suite including the Simulink and Stateflow extension in this development stage. Let us consider the MATLAB model shown in Fig. 3 , as an example modeling the functionality of an odometry. It reads data from moving sensors to calculate changes in the position over time according the actual orientation and movement speed of the robot. In the simulation stage, such a model is used to apply a so-called *model test (MT)*, where individual functionalities can be simulated sending static input values to the model (e.g., drive speed and turn rate of the robot as in Fig. 3) and plotting the computed output values as shown in Fig. 4. These one-shot/ one-way simulations are typical for the MT step and do not consider the interaction with the environment or a plant model. More complex behavior is constructed and validated in the form of individual functionalities and running *model-in-the-loop (MiL)* simulations [4] including preliminary environment models of the plant. At this point in time, feedback simulations validate the developed functionality considering the dynamic behavior of the environment. Outputs are sent to the plant model, which itself gives feedback used as input for the function blocks in the next iteration of the MiL simulation. In such a manner, the overall control law can be validated concerning basic constraints like stability, safety or reliability of the system (VII).

In the case of robotic systems, such a plant model can be represented at different levels, e.g., by using models representing a single sensor, the behavior of a single robot using multiple sensors or in the case of a complex simulation realizing the behavior of multiple robots as well as relevant parts of the logical and/or physical environment. Using such a plant model in the context of a MiL simulation, we must bridge the gap between our MATLAB models and the provided model of the plant (VI). For this purpose, on the one hand, we use the *RobotinoSim* simulator in combination with the graphical *RobotinoView* environment[8] to create plant models (cf. the upper path from the simulation stage in Fig. 2). Therefore, we implemented a block library for MATLAB in our development environment, which allows access to sensors (e.g., distance sensors, bumper, incremental encoder, electrical motors) and actuators according to requirement (V). The sensors and actuators can be accessed individually inside a MiL simulation supporting the validation of the models (VII). The RobotinoSim simulator provides optimal sensor values excluding effects such as sensor noise. Therefore, on the other hand, we can access the HW of the robot directly via a wireless LAN connection. Due to the fact that we use the concrete HW in this simulation setting, we could verify our functionalities and control algorithm with real sensor values including measure errors and sensor noise.

To sum it up, the first column in Table 1 gives an overview of the three tools used and our testing capabilities in the simulation stage as described previously. Additionally on the right in of Fig. 2, one can follow the toolchain used via the flow arrows.[9] However, we are not limited to the RobotinoSim tool in our development approach. We use this tool to show the proof of concept, but in general it is possible to create block libraries in MATLAB or use existing ones[10] for other robots, simulation frameworks or individual sensors/ actuators.

Table 1. Tools for testing and verification for each development stage

Tool \ Stage	Simulation	Prototyping	Pre-production
MATLAB/Simulink/Stateflow	✓	✓	
TargetLink		✓	
SystemDesk		✓	✓
RobotinoSim/View	✓	✓	
SystemDesk Simulator		✓	✓
Robotino Robot	✓	✓	✓

3.2 Prototyping Stage

The focus of this stage changes from design to implementation. While in the simulation stage models are the main artifacts, in this stage the source code plays a major role. In the following, we show how to support the prototyping stage at the level of more isolated functional parts as well as at the level of the system behavior by using the professional, commonly used tools of the automotive domain.

[8] In the following, we only mention the simulator, but we always use both tools together in combination. Tools see: `www.festo-didactic.com`

[9] The described RP flow to the real robot is not shown in the figure.

[10] For example this toolbox: `http://petercorke.com/Robotics_Toolbox.html`

Function Level – TargetLink: In the automotive domain, code generators are commonly used to derive an implementation for the specific target platform. Usually, the models from the simulation stage are directly used or refined until a code generation step is possible. In our development environment, the tool *TargetLink* from dSPACE is fully integrated into MATLAB and can automatically derive the implementation from behavior models in form of C-Code. In this step, we use the same MATLAB blocks as discusses in Section 3.1. So, we are able to seamlessly migrate (VI) our functions and control algorithm from the model level, realizing continuous behavior, to the implementation level, realizing a discrete approximation of the original continuous behavior.[11] We can configure several characteristics of the desired target platform/ HW.

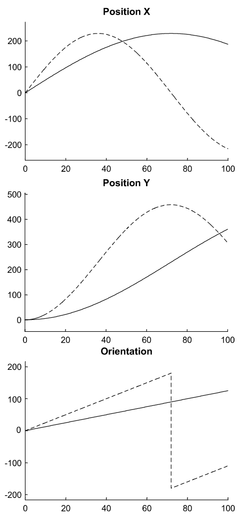


Fig. 4. MiL (dashed line) and SiL simulation values of the odometry block

Software-in-the-loop (SiL) simulation is a first step from the pure model execution to a code-based testing. Certain assumptions can be validated by replacing more and more models with code. While still executing the software on a host pc and not on the real HW, different effects can be analyzed, which result from chosen configuration parameters during code generation. Just as in the MiL simulation case, a SiL simulation can be applied in MATLAB using the generated source code instead of the original model. The developer can switch between the MiL and SiL simulation mode in MATLAB. Therefore, he can easily compare the simulation results. Fig. 4, for example shows the monitored results of the position as well as the orientation from the MiL and SiL simulation runs of the odometry. The simulations run against the RobotinoSim simulator. In the MiL run (dashed line), appropriate values for the actual position and orientation are calculated. Because of rounding (discretization) effects in the SiL run, the calculated values are much too low. So, the difference between pure model simulation and code generation becomes visible.

The problem in this special example could be fixed by choosing different values for the discretization over time. Calculating the position each 0.02 time units (corresponds to a scheduling with a period of 20 ms, cf. the constant value in Fig. 3) leads to very small offsets in the position, which is often rounded to zero due to discretization. After we identified the problem, we could easily fix it in the model. Instead of a 20 ms period, we double it to 0.04 time units for calculating the position. After generating code again, we could validate our assumption, which leads to a new requirement to trigger the functionality of the odometry with a period of 40 ms. Using code generators for automatically deriving the

[11] Discretization is applied at different levels. E.g., fixed point variables are used for the implementation at the data level or time continuous differential equations are mapped to discrete execution intervals at the timing level. For further details compare [4].

implementation realizing the behavior of initially created models support the seamless migration from the model level to the implementation level as well as allow to analyze effects arising from the implementation. Therefore, we cover the aspects IV, VI, and VII developing robotic systems at this point.

System Level – SystemDesk: For more complex system behavior resulting from the composition of multiple individual functionalities, we use the component-based architecture provided by the AUTOSAR framework. Individual functionalities provided by the MATLAB models are mapped on components such as those depicted in Fig. 5. The generated source code from TargetLink is mapped into the AUTOSAR SWC in the form of so-called *Runnables*.

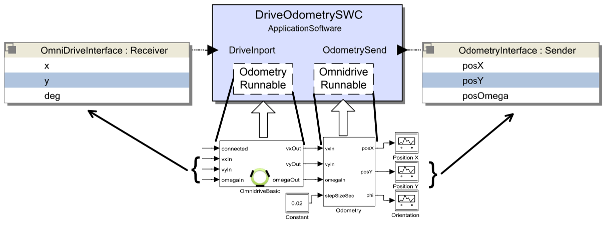


Fig. 5. Mapping from MATLAB models to SWCs

So, the same C-Code as in the SiL simulation is used and thus, a seamless integration (VI) of individual functions into the overall system behavior is achieved. In our example, we split the MATLAB model into two Runnables, namely *OdometryRunnable* and *OmnidriveRunnable*.[12] The SWC communicates to other ones over well defined ports. Furthermore, the input and output values are mapped to AUTOSAR interfaces with data entries and types respectively.

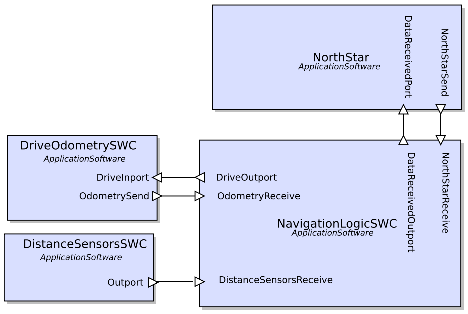


Fig. 6. SWC architecture in AUTOSAR

The AUTOSAR architecture consists of four SWCs[13] (see Fig. 6). It realizes the autonomous movement of the Robotino robot and includes the SWCs *DriveOdometrySWC*, *DistanceSensorsSWC*, *NorthStar* and *NavigationLogicSWC*. Each SWC provides the functionality such as that described previously in Section 2.1.

System Configuration: In addition to the architecture modeling and the separation of functions in different SWCs, SystemDesk supports a task specification for the underlying operating system. Runnables can be mapped to different tasks. Furthermore, several task activation events including periodic and sporadic ones

[12] This separation allows us to trigger the two Runnables with different periods.
[13] Due to a better understanding, we choose this simple excerpt of a larger architecture.

are supported and additional scheduling information like periods and priorities can be modeled.

For a system simulation, one has to specify a concrete AUTOSAR conform system configuration, which includes 1) a set of tasks, each consisting of one or more Runnables, 2) one or more electronic control units, which are specialized processors, and 3) communication capabilities (buses) with a concrete mapping of messages, which have to be exchanged. In the following, we describe the first point in more detail using our running example.

The Runnables *DistanceSensor*, *OmniDrive* and *CalculateDriveSpeed* are mapped to an OS task, which is executed with a period of 20 ms. A second task with the derived period of 40 ms contains the Runnable *Odometry* (cf. Section 3.2). The resulting execution of the Runnables and the schedule of the tasks is depicted in the upper time line of Fig. 7. These four basic functions run under hard real-time constraints, so we can be sure that all deadlines are met.

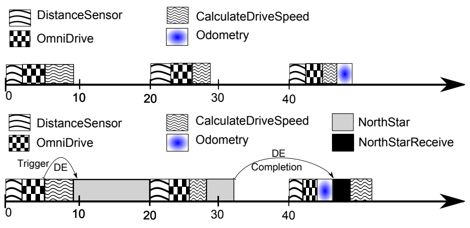


Fig. 7. Upper time line: scheduling of hard real-time functions. Lower time line: combined hard and soft real-time scheduling.

After adding more information to satisfy points 2) and 3), SystemDesk can realize a system simulation. It automatically generates the required simulation framework code according to the AUTOSAR standard, e.g., the RTE, messages, task bodies and trigger events. Furthermore, existing source files, generated by TargetLink (from the MATLAB models), are compiled and linked into the tasks. The complete system runs in a special simulation environment inside the SystemDesk tool and considers the HW configuration as well as OS task specifics. Again, this simulation is executed on a host PC and thus belongs to the prototyping stage. As depicted in Fig. 2 and the appropriate second column in Table 1, we can validate the overall system behavior in the three following scenarios considering the aspects (VI, VII, and VIII): First, we can monitor different output values, messages and variables inside the simulation environment itself. Second, we can connect the Robotino simulation environment as a plant model, which interacts with the SystemDesk tool. Finally, we are able to replace the plant simulator with the real robot. Therefore, we have to establish a W-LAN connection for the communication and to access the real sensors as well as actuators. Unfortunately, this unpredictable connection can destroy the timing behavior of the simulation, although the simulator tries to keep all deadlines. If we find errors during our validation processes, we can change the configuration, architecture or communication possibilities in SystemDesk and run our simulations again. Furthermore, we are able to re-import SWCs into MATLAB and therefore, switch between the different development stages.

According to the stage description in [4], *Hardware-in-the-loop (HiL)* simulations can be applied in the prototyping stage too. In these kind of simulations, the "unlimited" execution and testing hardware is often replaced by special evaluation boards with additional debugging and calibration interfaces, which are similar to the final hardware configuration. Due to limitations of our robot laboratory and missing evaluation boards, we do not use such HiL simulations. However, the integration of such boards can be carried out easily in the SystemDesk tool by changing the HW specification during the system configuration step.

Adaptation to Robotic Systems: In contrast to classic (hard) real-time applications in the domain of automotive embedded systems, robotic systems must realize functionalities, for which worst-case execution times (WCETs) are hard or impossible to predict. As a result, the integration of such behavior can only guarantee soft real-time constraints. In our application example, we use the NorthStar sensor, which is accessed via a serial USB port. Due to the fact that we use the default Linux OS driver, the timing behavior is unpredictable for that port. Additionally, we implement the navigation logic, which uses this NorthStar sensor, with library function from the MRPT library (cf. Section 2.1) for maintaining the map information of the explored topology. This includes the dynamic instantiation of an unknown number of C++ objects (classes) at runtime, what hinders the WCET estimation, too. Therefore, the WCET can rarely be estimated at the range of a few milliseconds.

Due to te fact that AUTOSAR does not directly support such a combination of soft and hard real-time behavior, we need to adapt the framework to realize it in such a way that: (1) the schedule guarantees the preservation of hard real-time constraints for the basic functionality and (2) the communication between soft and hard real-time functionality is achieved as such that only consistent data is read.

In the first step, we separate the hard and soft real-time functions/ Runnables and map them onto different OS tasks. A soft real-time task can be configured with a lower priority in such a way that it will be interrupted by all hard real-time tasks with a higher priority. Following this development guideline achieves the first requirement (1). For the second one, we use special data events (DE) in combination with Sender/Receiver-interfaces of the AUTOSAR standard. Such events can be used to trigger the execution of Runnables inside an OS task. A DE is sent from the hard real-time task (resp. Runnable) to trigger the execution of the soft real-time Runnable. The interruptible soft real-time function produces another DE, iff, the requested output data is in a consistent state (2). The hard real-time task can read the data in its next period and triggers the soft real-time function again if required.

The lower time line in Fig. 7 illustrates the combined scheduling of soft and hard real-time tasks. The soft real-time task is triggered via a DE generated by the *OmniDrive* Runnable. During execution, it is preempted in order to ensure the timing deadlines of the other hard real-time Runnables. After the *NorthStar* Runnable has finished its execution, it sends another DE to indicate completion, which includes that the consistent data results can be read in the next period of the *OmniDrive* Runnable.

Our described development approach supports the prototyping stage of robotic systems very well. We are able to incrementally refine more and more information to specify the system while seamlessly integrating artifacts of the previous stages (VI). Activities like function development and system configuration can be applied in a round-trip engineering approach (I, II). First, we develop the control functions in MATLAB (II). Afterwards, we generate code using the TargetLink code generation capabilities (IV). At this point, we can manually integrate additional, arbitrary functionality in C/ C++or use existing libraries (V). As soon as sufficient code artifacts and libraries are provided, we are able to use the code generation and simulation capabilities of the SystemDesk tool (IV, VII). Existing SWCs, e.g., developed in a previous project can be seamlessly integrated into the system architecture and new components can be exported as library elements for other projects. Additionally, we have shown the idea of creating a combination of hard and soft real-time tasks using the AUTOSAR framework during this stage (VIII).

3.3 Pre-production Stage

Within the pre-production stage, usually, a prototype of the real system is built. This prototype is tested against external environmental influences (such as temperature, vibration or other disturbances). The goal of this stage is to prove whether all requirements and constraints are still met on the real HW. During this last integration of all components and system parts, upcoming problems should be fixed as early as possible and before the final production of the product starts [4]. In our setting, we did not built any HW prototypes. Instead, we integrate the overall functions, components as well as the generated RTE and tasks to a complete system, compile and run it on the target processor of the robot[14]. So in this last step, we have no simulation semantic and W-LAN connection to other tools. We can fully operate the behavior of the robot in hard real-time. For verification, we use some hard real-time logging mechanism of the robot OS. Furthermore, we can change the hardware composition of the robot by adding or removing special sensors and actuators (see Section 2.1).

4 Related Work

Tackling the complexity of robotic and other embedded systems, we found a great deal of previous work covering partial aspects of developing such systems. According to our found aspects in Section 1 and our focus on the automotive domain, we combine the existing development methodology from [4] and the AUTOSAR standard. We evaluate our approach in a robotic production scenario using the component-based AUTOSAR architecture [12]. Other frameworks often cover only parts of the found aspects. RT-Middleware [1,2] and ORCA [5] focus on the specification of components including interfaces and ports (aspects

[14] We can automatically transform AUTOSAR compliant applications to the RTAI Linux.

IV, V, and VII). They lack the integration of an overall methodology as well as architecture specification. Very similar to the AUTOSAR approach but with the focus on the robotic domain is the MOAST framework [3], which covers the points II, III, IV, and partially VII. However, a seamless integration of an overall methodology and the support for different tools is missing. A good comparison with other frameworks can be found in [8].

In the embedded world, testing and simulation are the major activities to verify the behavior of the system [4]. We have made intensive use of the MATLAB, Robotino and SystemDesk simulators. However, other simulators like Gazebo [7] or Webots [9] are applicable as well.

Furthermore, there are other tools for modeling and simulation of AUTOSAR conform parts of the system architecture as the Real-Time Workshop(RTW) (MATLAB extension) from the MathWorks company.[15] The RTW extension is limited to component functionality and interfaces. The overall system architecture description is needed beforehand [11]. Parts of this description can be built by the Volcano Vehicle Systems Architect[16] (VSA), which can import and export AUTOSAR conform architecture description [10]. However, all these different tools can be used instead of the tools presented in this paper, if the integration in the overall methodology as well as the support for the different development stages is guaranteed.

Considering real-time constraints and combining hard and soft real-time tasks are important because of the support for library functionalities in different use-cases. For example, our navigation logic in this paper cannot be done in a predictive amount of time. Combining soft and hard real-time guarantees (1) a basic hard real-time behavior of the robotic system and (2) supports the development of complex algorithm and higher system components. Existing robotic frameworks as the Robot Operating System[17] or Microsoft Robotics Studio[18] are well established for developing complex robotic systems. They have drawbacks concerning the integration of hart real-time constraints.

5 Conclusion

We have shown in this paper an overall methodology (I) along with different exemplary development activities as well as artifacts on different levels of abstraction (II, III). We know that not all tasks can be executed in HRT. Therefore, we have shown the idea of combining different hard and soft real-time tasks into the overall system using the AUTOSAR approach (VIII). Furthermore, we are able to integrate several tools and external libraries into our overall toolchain (IV, V, VI). However, we are not limited to the tools we show in this paper. This provided flexibility is stabilized by a clear structure of different development stages (III) allowing a round-trip engineering for different functions, the integration of

[15] `www.mathworks.com/embedded-systems/`

[16] `www.mentor.com/`

[17] `www.ros.org`

[18] `www.microsoft.com/robotics/`

components as well as the simulation and testing of the development artifacts to the point of the complete system on the target platform. Therefore, we adapt ideas of the automotive domain to the development of robotic systems.

As future work, we want to build a complex robot production scenario applying the proposed methodology of this paper and evaluate the interaction of soft and hard real-time system parts.

References

1. Ando, N., Suehiro, T., Kitagaki, K., Kotoku, T., Woo-Keun, Y.: RT-middleware: distributed component middleware for RT (robot technology). In: 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 3933–3938 (2005)
2. Ando, N., Suehiro, T., Kotoku, T.: A Software Platform for Component Based RT-System Development: OpenRTM-Aist. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 87–98. Springer, Heidelberg (2008)
3. Balakirsky, S., Proctor, F.M., Scrapper, C.J., Kramer, T.R.: A Mobile Robot Control Framework: From Simulation to Reality. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 111–122. Springer, Heidelberg (2008)
4. Broekman, B., Notenboom, E.: Testing Embedded Software. Wesley (2003)
5. Brooks, A., Kaupp, T., Makarenko, A., Williams, S., Oreback, A.: Towards component-based robotics. In: International Conference on Intelligent Robots and Systems, pp. 163–168 (2005)
6. Giese, H., Neumann, S., Niggemann, O., Schätz, B.: Model-Based Integration. In: Giese, H., Karsai, G., Lee, E., Rumpe, B., Schätz, B. (eds.) MBEERTS. LNCS, vol. 6100, pp. 17–54. Springer, Heidelberg (2010)
7. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 2149–2154 (2004)
8. Manso, L., Bachiller, P., Bustos, P., Núñez, P., Cintas, R., Calderita, L.: RoboComp: A Tool-Based Robotics Framework. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 251–262. Springer, Heidelberg (2010)
9. Michel, O.: Webots: Professional Mobile Robot Simulation. International Journal of Advanced Robotic Systems 1, 39–42 (2004)
10. Sandmann, G., Seibt, M.: AUTOSAR-Compliant Development Workflows: From Architecture to Implementation - Tool Interoperability for Round-Trip Engineering and Verification and Validation. Tech. Rep. 2012-01-0962, SAE International (2012)
11. Sandmann, G., Thompson, R.: Development of AUTOSAR Software Components within Model-Based Design. Tech. Rep. 2008-01-0383, SAE International (2008)
12. AUTOSAR_EXP_LayeredSoftwareArchitecture.pdf, page id: 94ju5 (2011), http://www.autosar.org/

Combining IEC 61499 Model-Based Design with Component-Based Architecture for Robotics

Li Hsien Yoong, Zeeshan E. Bhatti, and Partha S. Roop

Department of Electrical and Computer Engineering
University of Auckland
Auckland, New Zealand

Abstract. The model-driven approach is an increasingly popular trend in software design. It provides many benefits in terms of system design, reusability, and automatic code generation. In the industrial automation domain, the IEC 61499 standard is a recent initiative that adopts this approach. It offers an open, platform-independent framework for designing distributed control systems, whereby the interface and behaviour of a component is described using a function block. On the other hand, in the robotics domain, the Robotic Technology Component specification proposes a framework that allows software components to be easily integrated in a robotic system. The focus of that specification is not so much on the definition of each component's internal behaviour, but rather on the management and interaction of those components. The combination of both these standards offers a comprehensive solution for designing robotic software components in a model-driven approach. This paper describes a tool-chain for doing so, and illustrates its viability through an example.

Keywords: IEC 61499, model-based design, Robotic Technology Component, synchronous programs.

1 Introduction

The use of model-based design [11] for software development is a growing trend in the industry. As in other engineering disciplines, the central idea behind model-based techniques in software engineering is to facilitate the specification and analysis of complex systems using abstractions. For the case of software, this practice results in a software system being decomposed into *components*, with each component *modelling* an aspect of the problem domain.

Software components used in model-based design have well-defined interfaces, which specify the functionality of a given component in terms of input/output behaviour. Multiple components can be connected together through their respective interface to form a complete software system. The adoption of component-based practices brings many benefits to software design, implementation, maintenance, and reuse. For instance, the interface of a component serves as a "contract" to other components, which simplifies the decomposition of the problem domain. This in turn facilitates rapid development by providing a clear delineation of functionality between components, as well as the ability to easily perform isolation testing.

One of the challenges faced by robot developers is the co-ordination of various software components within an integrated system [5]. The Robotic Technology Component (RTC) specification [10] offers an open architecture for accomplishing this. It proposes a middleware layer (called RT-middleware) that provides a simple way for integrating new functionality within a robotic system using modularized software components (called RT-components). Robotic systems built on this architecture consist of networks of RT-components connected to each other through their interfaces.

The need for reliable and deterministic software components has also emerged as an orthogonal challenge, especially for robotic systems used in safety-critical applications. Commercial pressures often require that this challenge be met without increasing development time and cost unreasonably. Existing literature has made a strong case that this goal can only be achieved by using processes that favour error prevention over error detection and forward prediction of correctness, rather than retrospective demonstration of correctness [1].

A key enabler to attain such *correct-by-construction* software components is the use of mathematically-sound formal methods. However, industry practitioners are often hesitant to go down this path due to their unfamiliarity with formal methods. A common trend in overcoming this barrier is to embed formal programming semantics within intuitive graphical notations that are already accepted by the industry. One such example of this is the formalization of the visual artefacts of the IEC 61499 standard [6] using the synchronous approach [13]. That standard provides a framework for designing distributed industrial automation systems using a component-oriented approach based on *function blocks*.

This paper proposes to utilize IEC 61499 function blocks to develop reliable and deterministic RT-components that can be used in safety-critical robotic applications. Unlike RT-components which only provide for the component interface to be specified, function blocks allow *both* the interface and the internal behaviour of the component to be specified. This paper leverages the synchronous approach for automatically generating code from function blocks [13], and extends that technique to transform the resultant code into RT-components. This provides a pathway to obtain robotic software components using a formal model-based approach that has already been proven in the automation domain.

The remainder of this paper will discuss how this has been achieved. In the next section, an overview of both the component-based technologies will be illustrated using an example. Then, Sect. 3 will describe the implementation of a tool-chain which enables function blocks to be specified, and code for RT-components to be automatically generated from that specification. Experimental evaluations of this tool-chain are provided in Sect. 4, before concluding in Sect. 5.

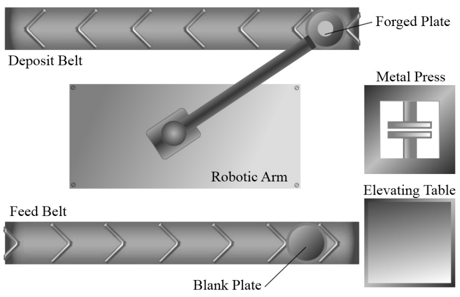


Fig. 1. Layout of a production cell

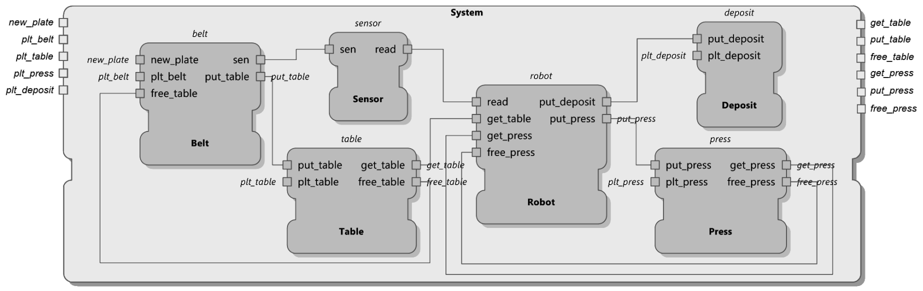


Fig. 2. IEC 61499 implementation of the production cell

2 Overview

Fig. 1 shows the simplified layout of an automated production cell, which was first described in a case study in [8]. That production cell has an elevating table which accepts a blank plate from a feeding conveyor belt. When a plate arrives, the table rises to the level of a robotic arm. The robotic arm then picks up the plate and places it into a forging press. Once the plate is forged, the robotic arm picks up that plate and puts it onto a deposit belt.

Using this as an example, the main design artefacts of IEC 61499 will be illustrated in the following subsection.

2.1 IEC 61499

Fig. 2 shows the function block model corresponding to the production cell of Fig. 1. The top-level function block representing the production cell has been named as *System*, while the other blocks that make up the production cell has been intuitively named as *Belt*, *Table*, *Sensor*, *Robot*, *Press*, and *Deposit*—each corresponding to a particular physical component being modelled.

The IEC 61499 standard prescribes three different kinds of function blocks, namely, the *basic*, *composite*, and *service interface* function blocks. Irrespective of the kind, each function block has a well-defined interface, as exemplified in

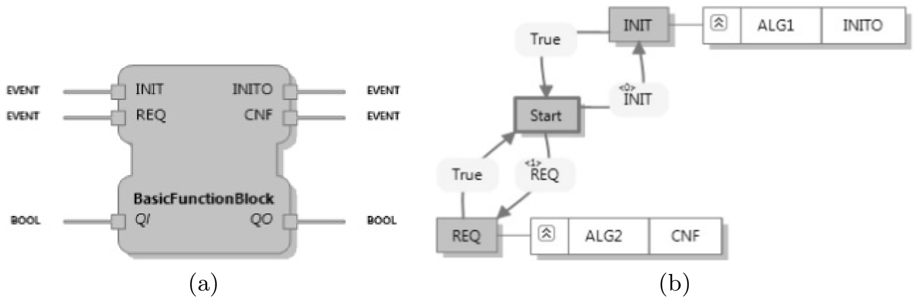


Fig. 3. Example of a basic function block, with (a) depicting the interface, and (b) the execution control chart

Fig. 3(a). Input ports are specified on the left, while output ports are specified of the right side of the block. Function blocks also distinguish between *event* and *data* ports: Event ports are drawn on the upper half of the block, while data ports are drawn on the lower half.

The behaviour of basic function blocks is defined by a Moore-type state machine, known as an *execution control chart* (ECC). This is illustrated in Fig. 3(b). States are represented by rectangles, with a bold (or double) outline used to indicate the initial state. A state can be associated with zero or more *actions*, which may consist of an output event to be emitted and/or an algorithm to be executed. An action that is associated to a state is performed once on every entry to that state. As an example, consider the *INIT* state in Fig. 3(b). The action consisting of the *ALG1* algorithm and the *INITO* output event is associated to that state. The *INIT* state will be entered whenever the *INIT* input event is present in the *START* state. Whenever that happens, the *ALG1* algorithm will be executed and the *INITO* event emitted exactly once.

Unlike basic function blocks, the behaviour of composite function blocks is defined by a network of other blocks instead of an ECC. This provides a means to hierarchically encapsulate networks of function blocks within a single block. Service interface function blocks, on the other hand, provide the means to encapsulate low-level details, such as device drivers and communication protocols, within a function block.

In Fig. 2, *System* is a composite function block, while all the other blocks are basic function blocks. As can be seen, this component-oriented approach for describing software is intuitive, and provides sufficient detail for fully executable code to be generated from it.

2.2 Robotic Technology Component

The Robotic Technology Component specification [10] defines a component model for robotic software (called RT-component), as well as a systematic infrastructure for composing those components (called RT-middleware) in a system. RT-components may have two kinds of ports, namely:

1. *data ports*, which are used for data exchange between RT-components, and
2. *service ports*, which are used for service-oriented communication (e.g, commands or function calls) between RT-components.

Every RT-component is created by a *manager* (known as the RTC manager), which is part of the middleware. Each RT-component goes through a common life-cycle that is made up of the *Created*, *Alive*, and *End* states. The *Alive* state consists of several sub-states, each having an associated callback function interface. The RTC manager is responsible for a component's life-cycle, as well as for invoking each callback associated to a given state. Thus, the behaviour of an RT-component is specified by coding the logic for every callback function required. As an example, the manager will call the `onInitialize` function after creating an instance of an RT-component, and the `onFinalize` function when the RT-component has finished. The main logic of an RT-component is implemented in the `onExecute` function, which will get called repeatedly when the component is in its active state.

Collectively, the states of the component and the callback functions form the *core logic* of the component. When an RT-component is created, it is assigned to an *execution context*, which then executes the RT-component's core logic.

The specification provides for a number of different execution semantics that may be applied to an RT-component, e.g., periodic sampled data processing (also known as *data flow*), and stimulus response processing (also known as *asynchronous* or *discrete event processing*). Depending on the desired execution semantics, an RT-component needs to be assigned to a different kind of execution context: The data flow semantics correspond to the *periodic* execution kind, while the asynchronous semantics correspond to the *event-driven* execution kind.

So, while the RTC specification does not provide a structured model to specify the internal behaviour of a component, like ECCs in IEC 61499, its strength lies in the management and integration of the components. In the next section, we will see how IEC 61499 may be combined with this specification to obtain a more comprehensive model-driven approach for developing robotic software components.

3 Implementation

In the model-based approach to software development, the typical workflow begins with a decomposition of the requirements specification into components, followed by a description of those models in a suitable modelling language, code generation from those models, and compilation of the code for deployment onto a particular target. For the approach proposed in this paper, this workflow, together with the tools for each step, is illustrated in Fig. 4.

In that figure, FBC refers to the function block compiler described in [12,14], while RTCTEMPLATE is a tool that generates templates (skeleton code and function stubs) for RT-components. RTCTEMPLATE is part of OpenRTM-aist [2], which is an open source implementation of the RTC specification. FB2RTC is

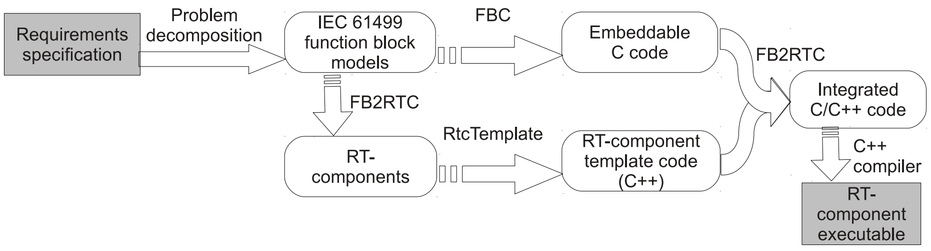


Fig. 4. Workflow of a model-based approach for creating robotic software components

a tool that has been developed in the context of this work for the purpose of deriving RT-components from IEC 61499 function blocks.

All these tools, together with the C++ compiler, form a tool-chain for deriving RT-component executables using a model-based design with function blocks. To enable this whole process to be done seamlessly, an integrated development environment (IDE), called BlokIDE (formerly known as TimeMe Studio) [3], has been extended. BlokIDE allows a designer to create function blocks, and to seamlessly generate RT-component code from them. The code generation itself relies on Fbc, RtcTemplate, and Fb2rtc.

The following subsections will describe the code generation process and the IDE in greater detail. We first provide a brief overview of the nature of the code generated by Fbc and RtcTemplate. Then, we describe how Fb2rtc maps function blocks into RT-components, before going on to show how the resultant code from Fbc and RtcTemplate are weaved together. Finally, we describe the IDE and explain how all these are brought together in a seamless tool-chain.

3.1 FBC and RtcTemplate

Using Fbc, code for a given function block description is generated following the synchronous model of execution [12]. With this approach, a portion of code for each function block within a network gets executed in a periodic cycle, called a *tick*. For basic function blocks, the execution in each *tick* corresponds to the evaluation of transitions in the current state of an ECC, and the subsequent computation of action(s) in the destination state, should a transition be taken. This use of a well-defined periodic cycle results in software with predictable temporal properties, making this a well-suited technique for designing safety-critical systems.

Fbc compiles every function block type it encounters into a separate C structure, whose members describe the event-data interface, as well as the local data (if any) of that function block. The compilation process also results in at least two functions, named `<FBType>init` and `<FBType>run`, for each function block type. `<FBType>init` serves as the constructor method for the corresponding function block type, since C does not have built-in support for object constructors common in object-oriented programming languages.

For basic function blocks, the `<FBType>run` function implements the execution code for the corresponding ECC. For composite function blocks, the `<FBType>run` function implements a netlist that describes the interconnection of components within the composite block. The execution of component blocks within the composite function block is in turn invoked through their respective `<FBType>run` functions. Both the `<FBType>init` and `<FBType>run` functions for a particular function block will eventually get embedded into the corresponding RT-component, as we shall see later.

RtcTemplate. Various types of template code can be generated for an RT-component using RtcTemplate. RtcTemplate may be invoked with various parameters to generate template code for RT-components having different properties, such as the desired component execution semantics and the corresponding execution context, as well as the number and types of various input/output data ports. The template code may be generated in C++, C#, Java, Python, or Visual Basic .NET.

For the purpose of this work, only C++ code was generated to enable simple integration with the output of Fbc. The generated header file contains the class definition of the RT-component, which includes members such as the input/output data ports of the component. The implementation file, meanwhile, contains stubs for the various callback functions. The main point to note here is that the generated code is merely a "skeleton," and does not define the component's behaviour in any way.

3.2 FB2RTC

Fb2rtc performs its operation in two phases:

- first, it extracts the port information from a given function block in order to produce an RT-component with equivalent data ports;
- then, it refines the behaviour of an RT-component by embedding the code generated by Fbc into the function stubs created by RtcTemplate.

These two phases are described next.

Generating RT-Components from Function Blocks. Fb2rtc begins this process by extracting the port descriptions of a function block and mapping them into equivalent data ports of an RT-component. The mapping of data ports from function blocks to RT-components is straightforward: A simple look-up table is all that is needed to map function block data types to RT-component data types. However, the mapping of event ports is not so straightforward, since RT-components do not have event ports.

With the synchronous approach, events in function blocks are either present or absent in a *tick*. Unlike data, events are not persistent, meaning that events that are present in a given *tick* will no longer be present in the next if it is not again emitted in that *tick*. For the purpose of generating RT-components,

```
RTC::ReturnCode_t FBBasicFunctionBlock::onExecute(RTC::UniqueId ec_id)
{
  if ( m_INITIn.isNew() ) {
    m_INITIn.read();
    _functionBlock._input.event.INIT = m_INIT.data;
  }
  if ( m_REQIn.isNew() ) {
    m_REQIn.read();
    _functionBlock._input.event.REQ = m_REQ.data;
  }
  if ( m_QIIn.isNew() ) {
    m_QIIn.read();
    _functionBlock._QI = m_QI.data;
  }
  BasicFunctionBlockrun(&_functionBlock);
  m_INITO.data = _functionBlock._output.event.INITO;
  m_INITOOut.write();
  m_CNF.data = _functionBlock._output.event.CNF;
  m_CNFOut.write();
  m_QO.data = _functionBlock._QO;
  m_QOOut.write();
  return RTC::RTC_OK;
}
```

Fig. 5. Code generated for the `onExecute` function for the function block in Fig. 3

FB2RTC simply maps event ports to Boolean data ports in the resulting RT-component. Events that are present are represented by a "true" value, while those that are absent are represented by a "false" value. This mapping enables ECC transitions that are guarded by events to be preserved. The transient nature of events, however, requires additional code to be explicitly generated. This takes place while weaving the outputs of FBC and RTCTEMPLATE together during the second phase of FB2RTC, which will be described next.

Once the port mappings are completed, FB2RTC invokes RTCTEMPLATE by passing it the appropriate port parameters. In addition, RTCTEMPLATE is given parameters to generate an RT-component that adopts the data flow execution semantics and runs using the periodic execution context. This choice of execution semantics and execution context is amenable to the synchronous model of function blocks used in [12]. In particular, the periodic execution rate maps directly to the notion of a *tick* in the synchronous approach. This results in software components that have predictable temporal properties, which is desirable for safety-critical applications.

Weaving the Generated Code together. The main task that FB2RTC performs in this phase is to embed the code generated by FBC into the skeleton code produced by RTCTEMPLATE. A sample of the `onExecute` function produced by FB2RTC for the function block of Fig. 3 is shown in Fig. 5.

The three main places where the code generated by RtcTemplate is modified are:

1. in the RT-component class definition, where an instance of the function block structure generated by Fbc is created;
2. in the `onInitialize` function stub, where a call to the `<FBType>init` function is inserted; and
3. in the `onExecute` function stub, where a call to the `<FBType>run` function is inserted (line 15 of Fig. 5). Also, additional code is generated by Fb2rtc at this point to pass information from the input ports of the RT-component to the encapsulated function block (lines 3–14), and from the output ports of the function block to the RT-component (lines 16–21).

To ensure that the transient properties of events are preserved, output events are cleared at the start of each *tick*, while input events are cleared at the end of each *tick* within the `<FBType>run` function. Moreover, the `isNew` method is called for each Boolean input port corresponding to an event input port before its value is read to ensure that only fresh values can result in new input events to the function block (lines 3–14). This ensures that the Boolean values representing events at the RT-component interface are represented faithfully to the encapsulated function block.

3.3 BlokIDE

BlokIDE is a Microsoft Visual Studio based integrated development environment that aims to facilitate the design of safety-critical embedded systems. BlokIDE comprises of model editors, a file-based project system, a debug engine, a model verifier, and a static timing analyser. It is built on the extensibility model [9] of Visual Studio, which provides a component-oriented architecture. The components interact with each other through the global service provider by providing and consuming services. The UML component diagram in Fig. 6 presents the component dependencies and interfaces.

The model editors allow a developer to create software in a model-driven fashion using the standard visual artefacts of IEC 61499. The timing analyser is an additional plug-in to BlokIDE that can be used to determine the worst-case reaction time of a function block network [7]. This facilitates the development of real-time systems. The custom debug engine allows a developer to perform cycle-based simulation (corresponding to each *tick*) of a model on a desktop machine to check its behaviour prior to the actual execution on the target platform. The model verifier provides formal verification to prove that the model holds certain safety and liveness properties [4]. In the case of a possible violation, the model verifier provides a counter-example. A custom project system aggregates all the models created by a developer, and provides the backbone for the build process, code generation, timing analysis, and simulation features.

The main components of BlokIDE that have been extended to support the seamless development of RT-components from function blocks are the Project

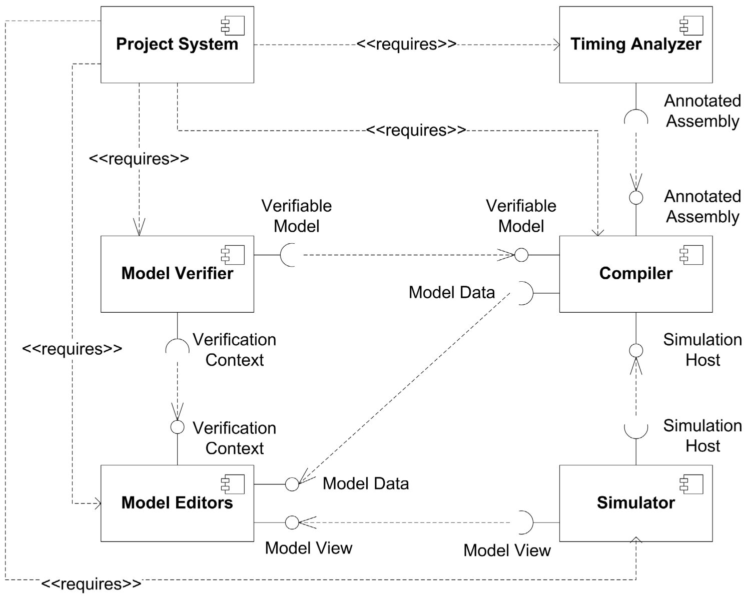


Fig. 6. The component diagram for BLOKIDE

System and Compiler components. A project in BLOKIDE can be created with several configurations. These configurations enable BLOKIDE to interact with the project elements using a variety of back-end tools for different purposes, such as formal verification, timing analysis, and code generation. For this work, a new configuration for mapping function blocks to RT-components was created. The primary back-end tool associated with this configuration is FB2RTC.

The Compiler component of BLOKIDE was also extended to provide the necessary interface to invoke FB2RTC, as well as to report any message or error coming from FB2RTC. With these enhancements, BLOKIDE is able to provide a seamless environment to automatically generate code for RT-components with their internal behaviour specified using function blocks.

4 Evaluation of the Tools

The tool-chain described in the previous section has been used to develop RT-components for the example in Fig. 2. Through this experience, we found that the proposed approach simplified the creation of RT-components in several ways.

In the initial development stage, BLOKIDE enabled an easy decomposition of the production cell logic into various software components. The behaviour of each software component could be designed individually as function blocks and tested using simulation. Once each function block's behaviour had been verified, the process of composing them together could be intuitively done by graphically connecting their ports together to form a network. Integration testing was also straightforward, as the entire network could be simulated, and specific issues pertaining to a particular block could be localised by "stepping" into that block while simulating the entire network.

The key advantage of the approach proposed in this paper was in the final code generation phase. An RT-component encapsulating the full functionality of the production cell was successfully generated and compiled using g++ for a Linux-based platform.

We also experimented with different deployment scenarios, where the various function blocks in Fig. 2 were generated as separate RT-components. It is here that the strengths of the RTC specification shine: While IEC 61499 allows for distributed deployment of function blocks through the use of special communication function blocks, the distribution of RT-components is completely transparent to the designer, as the middleware abstracts the communication between components. No changes were required in the logic of the original function blocks for this purpose.

5 Conclusions and Future Work

The combination of the model-based approach of IEC 61499 with the component-oriented versatility of the RTC specification offers the best of both worlds. It provides an intuitive manner to create reliable robotic software components following the synchronous model of execution, while enabling transparent communication between those components in centralized as well as distributed deployment scenarios. The interface, as well as the behaviour of an RT-component, can be automatically generated using the proposed approach.

At present, Fb2rtc generates RT-components with the data flow execution semantics only, as Fbc assumes a synchronous model of execution for a function block network. However, Fbc enables separate networks of function blocks (known as *resources* in IEC 61499), to communicate asynchronously with each other via special communication function blocks. We intend to enhance Fb2rtc to generate RT-components with asynchronous execution semantics directly from resources in the near future. The communication function blocks in each resource will be compiled as corresponding data ports in the resulting RT-components.

The ideas espoused in this paper has been demonstrated through an actual tool-chain capable of supporting realistic software engineering development. We have drawn from, and have enhanced some existing tools, as well as developed a new tool to support this endeavour. As the need for safety in robotic software components continues to grow, we believe that our approach, which combines model-based design, open standards, and synthesis tools founded on rigorous methods, offers a practical solution for the future.

Acknowledgments. The authors would like to thank I-Han Kuo for the assistance he provided while working with OpenRTM-aist.

References

1. Amey, P., Dion, B.: Combining model-driven design with diverse formal verification. In: 3rd European Congress on Embedded Real Time Software, Toulouse (January 2006)
2. Ando, N., Suehiro, T., Kotoku, T.: A Software Platform for Component Based RT-System Development: OpenRTM-Aist. In: Carpin, S., Noda, I., Pagello, E., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2008. LNCS (LNAI), vol. 5325, pp. 87–98. Springer, Heidelberg (2008)
3. Bhatti, Z.E.: A Model-Driven Approach for Safety Critical Systems. M.E. thesis, Department of Electrical and Computer Engineering, University of Auckland (2011), https://researchspace.auckland.ac.nz/handle/2292/6421
4. Bhatti, Z.E., Sinha, R., Roop, P.S.: Observer based verification of IEC 61499 function blocks. In: IEEE International Conference on Industrial Informatics (INDIN), pp. 609–614 (July 2011)
5. Biggs, G., Ando, N., Kotoku, T.: Coordinating Software Components in a Component-Based Architecture for Robotics. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 168–179. Springer, Heidelberg (2010)
6. International Electrotechnical Commission, Geneva: International Standard IEC 61499-1: Function blocks – Part 1: Architecture, 1st edn. (January 2005)
7. Kuo, M.M.Y., Sinha, R., Roop, P.S.: Efficient WCRT analysis of synchronous programs using reachability. In: 48th ACM/IEEE Design Automation Conference (DAC), San Diego, pp. 480–485 (June 2011)
8. Lewerentz, C., Lindner, T. (eds.): Formal Development of Reactive Systems. LNCS, vol. 891. Springer, Heidelberg (1995)
9. Microsoft: MSDN: Microsoft Visual Studio Extensibility Developer Center (2010), http://msdn.microsoft.com/en-us/vstudio/vextend.aspx
10. Object Management Group: Robotic Technology Component Specification, Version 1.0 (April 2008), http://www.omg.org/spec/RTC
11. Selic, B.: The pragmatics of model-driven development. IEEE Software 20(5), 19–25 (2003)
12. Yoong, L.H., Roop, P.S., Salcic, Z.: Efficient implementation of IEC 61499 function blocks. In: IEEE International Conference on Industrial Technology (ICIT), Gippsland (February 2009)
13. Yoong, L.H., Roop, P.S., Vyatkin, V., Salcic, Z.: A synchronous approach for IEC 61499 function block implementation. IEEE Transactions on Computers 58(12), 1599–1614 (2009)
14. Yoong, L.H., Shaw, G.D., Roop, P.S., Salcic, Z.: Synthesizing globally asynchronous locally synchronous systems with IEC 61499. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews Preprint(99), 1–13 (2012)

A Reuse-Oriented Development Process for Component-Based Robotic Systems

Davide Brugali[1], Luca Gherardi[1], A. Biziak[1], Andrea Luzzana[1], and Alexey Zakharov[2]

[1] University of Bergamo, DIIMM, Italy
{brugali,luca.gherardi,andrea.luzzana}@unibg.it
[2] GPS GmbH, Stuttgart, Germany
zakharov@gps-stuttgart.de

Abstract. State of the art in robot software development mostly relies on class library reuse and only to a limited extent to component-based design. In the BRICS project we have defined a software development process that is based on the two most recent and promising approaches to software reuse, i.e. Software Product Line (SPL) and Model-Driven Engineering (MDE). The aim of this paper is to illustrate the whole software development process that we have defined for developing flexible and reusable component-based robotics libraries, to exemplify it with the case study of robust navigation functionality, and to present the software tools that we have developed for supporting the proposed process.

1 Introduction

The routine use of existing solutions in the development of new systems is a key attribute of every mature engineering discipline. Software reuse is a state of the practice development approach in various application domains, such as telecommunications, factory automation, automotive, and avionics. Software Engineering has produced several techniques and approaches for promoting the reuse of software in the development of complex software systems. A survey can be found in [3] (Sidebar *A Historical Overview of Software Reuse*).

In Robotics, software reuse is typically conceived as cut and paste of code lines from program to program: this practice is called opportunistic software reuse and might work only for the development of simple systems (e.g. for educational purposes) or for unique systems (e.g. a research prototype).

In contrast, the development of industrial-strength robotic systems that aim to become commodity, require a systematic approach to software reuse. Systematic software reuse is the routine use of existing software or software knowledge to construct new software, so that similarities in requirements, architectures and design between applications can be exploited to achieve substantial benefits in productivity, quality and business performance.

If a company that commercializes integrated robotic systems wants to achieve customer value through large commercial diversity with a minimum of technical diversity at minimal cost, the best approach to software development is the Software Product Line (SPL)[5].

An SPL is a set of applications (products) that share many (structural, behavioral, etc.) commonalities and together address a particular domain. The term domain is used to denote or group a set of systems (e.g. mobile robots, humanoid robots) or functional areas (motion planning, deliberative control), within systems, that exhibit similar functionality. Each new application is built from the SPL repository of common software assets (e.g. architectural and design models, software components). In the BRICS project we have defined a software development process that exploits the SPL approach and accounts for two peculiarities of the robotics field:

– Today, a huge corpus of software applications, which implement the entire spectrum of robot functionality, algorithms, and control paradigms, is available in robotic research laboratories and potentially could be reused in many different applications. Typically, their interoperability or their extensions towards novel applications require high efforts. Any company that aims at developing professional software for complex robotic systems has to make an initial investment in refactoring and harmonizing existing open source robotics libraries that implement the robot functionalities offered by the SPL. This phase is typically called *software development for reuse.*
– Typically, robotic systems integrators are not software engineers and do not master advanced software development techniques adequately. For this reason, the proposed development process exploits the Model-Driven Engineering (MDE) [15] approach. According to the MDE approach, robotic system integrators use domain-specific languages to build models that capture the structure, behavior, and relevant properties of their software systems. A new application is developed by reusing these models, customizing them according to specific application requirements, and semiautomatically transform models and even generate source code using transformation engines and generators. This phase is typically called *software development with reuse.*

Figure 1 illustrates the phases of the reuse-oriented development process described in this paper. The first two phases, namely *software refactoring* and *product line design*, are intended to produce software *for reuse.* The remaining two phases, namely *Variability modeling* and *Variability resolution*, support the development of software *with reuse.* In the upper part of Figure 1, the conceptual and software tools that support the process are linked to the various phases, namely *Refactoring Patterns* [6], the *BRICS Component Model* (*BCM*)[9], the BRICS Integrated Development Environment (*BRIDE*)[9], and the BRICS tool for variability modeling and resolution (*FODA*). In the lower part of the figure the input open source libraries and the intermediate products of the development process are represented, namely the *BRICS class libraries*, the Product Line models, the *Variability models (features models)*, and the *Application models.*

This paper aims to illustrate the whole software development process that we have defined for developing flexible and reusable component-based robotics libraries, to exemplify it with the case study of the robust navigation, and to present the software tools that we have developed for supporting the robotic engineers in modeling and resolving variability in component-based robotics systems.

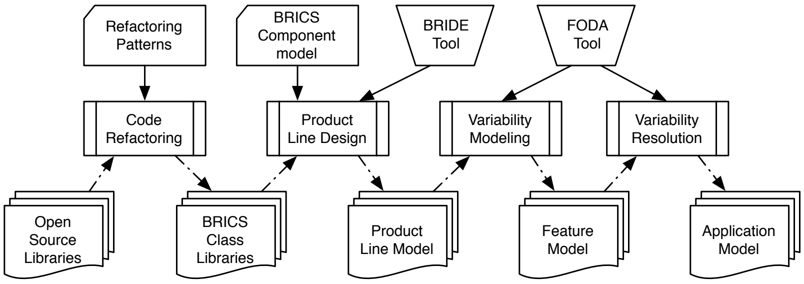


Fig. 1. The development process

The paper is structured as follows. Section 2 illustrates state of the art open source libraries that provide robust navigation functionality. Sections 3 to 6 present the four phases of the development process and exemplify them with the robust navigation case study. Finally Section 7 draws the relevant conclusions.

2 The Robust Navigation Case Study

Robust navigation is the ability of a mobile robot to perform autonomous navigating, while avoiding dangerous situations such as collisions with obstacles. It is a cross-sectional domain, which includes path planning, motion control and sensor data processing.

From an algorithmic point of view, the challenging task is to organize an efficient interaction between these functionalities in order to maximize performance, safety, and robustness. Mobile robot navigation algorithms have been a research topic for several decades (see [16] for a taxonomy). Existing algorithms could be roughly classified as one- and multi-step methods. One-step methods directly convert the sensor data to a motion commands. Majority of one-step algorithms are either based on classical planning or on the potential fields approaches [16]. Today, they are rarely used due to their inability to cope with dynamic environment and vehicle constraints. Multi-step methods (e.g. Dynamic Window Approach [8], Vector Field Histogram [17], Nearness Diagram [12]) overcome these limitations by creating a local map of the environment around the robot and performing local planning by computing possible motion directions (Nearness Diagram) and velocities (VHF) taking into account distance to the goal or to a precomputed path.

From a software development point of view, the challenge is to implement robust navigation functionality as a set of reusable components that can be assembled into *flexible* systems[1]. For this purpose, the BRICS project applies a novel approach to software development, which avoids to develop from scratch robot functionalities based on yet another software architecture. Instead, by collecting and analyzing well known open source libraries providing robotics

[1] The IEEE Standard Glossary of Software Engineering Terminology defines *flexibility* as the ease with which a system or component can be modified for use in applications or environments other than those for which it was specifically designed.

functionalities, we aim at identifying those architectural aspects (i.e. entities, data structures, interfaces, relationships) that are common to all or most of the implementations of the same family of functionalities, and those aspects that distinguish one implementation from another. These common architectural aspects represent the stable characteristics of a family of functionalities and are likely to remain stable through future implementations of the functionalities.

By analyzing existing robust navigation libraries (see table 1) we have realized that typically they refer to the same functionality using different names. In some cases, the functionality is not even mentioned but is implicit in the implementation.

– Motion planning (aka BaseGlobalPlanner, PathPlanner): is the process of computing a collision-free global path in a static environment between a given start position and a given goal position. The path is typically represented as a sequence of intermediate waypoints.
– Trajectory generation (aka ParameterizedTrajectoryGenerator, DWAPlanner): is the process of refining a path for introducing velocity information. A trajectory defines the planned positions of the robot over the time and is typically represented as a sequence of positions with an associated velocity.
– Obstacle detection and representation (aka CostMap2D, OgMap): is the process of using sensors information (e.g. laser scans) in order to detect the positions of the obstacles around the robot. This information is then used for creating and updating a map of the environment.
– Obstacle avoidance (aka Local LocalBaseNavigation, LocalNav, CAbstractHolonomicReactiveMethod): is the process of adapting the precomputed trajectory while the robot is moving in order to avoid unexpected obstacles that occlude the path.
– Position and velocity control (aka LocalBaseNavigation, LocalNav, MotionController): is the process of generating velocity commands to the robot in order to move it along the computed trajectory. This functionality has a strong dependency with the kinematics model, which is often implicit in the library implementations.
– Localization (aka FaithLocaliser, amcl): is the process of estimating the robot position with respect to a global reference frame. In the simplest case this functionality is implemented by using only the robot odometry but other sensors can be used to improve the odometric estimation.

3 Software Refactoring

Software refactoring is the process of changing a software system in such a way that it does not alter the external behaviour of the code yet improves its internal structure [7]. It occurs at two complementary levels: (a) *Syntactical refactoring* is a behavior preserving transformation that, through the adoption of good design principles (abstraction, information hiding, polymorphism, etc.) aims at making software artifacts modular, reusable, open. (b) *Semantic refactoring* is a domain-driven transformation that, through a careful analysis of the application

Table 1. Open source libraries for robust navigation

Type	Method	Library Name
Global Planners	Carrot planner, Dijkstra's alg.	ROS
	ARA*, Anytime D*, ANA*	SBPL/ROS
	Sampling-based Planning	OOMPL, BRICS_MM
Trajectory Generation	reactivenav::CPTG1 - CPTG7	MRPT
Local planners	VFF	MRPT
	VFH+	Player/Stage, ORCA
	Dynamic Window	ROS, Sunflower
	Trajectory Rollout	ROS
	Nearness Diagram	MRPT
	Elastic Band	ROS
Mapping	gmapping, costmap_2d	ROS
	ogNode, ogMap	ORCA
Localization	amcl	ROS
	ICP SLAM, RBPF SLAM, ...	MRPT

domain (commonality/variability and stability analysis), enhances software artifacts flexibility, adaptability, and portability. Software refactoring brings many advantages not immediately but in a long time. The initial cost in terms of time and effort spent for rewriting the code is balanced by the time gained in future. This gain is due to a code more readable, more reusable and more maintainable. The result of a refactoring process is a library of classes that are middleware-independent, are organized in a hierarchy of abstraction levels, provide harmonized interfaces (API), and implement a variety of algorithms.

In a previous paper [2] we have described how architecture refactoring patterns have been applied to refactor motion planning software libraries. Refactoring patterns provide guidelines to redistribute the responsibilities among the classes of a software library, to harmonize the common data structures and to reduce the coupling degree.

3.1 Case Study: The ROS Navigation Stack

This section presents the architecture of an open-source library, which provides a mobile-based navigation functionality. We selected ROS framework because of its popularity in robotic community. Figure 2 shows a portion of the class diagram that represents the architecture of the ROS navigation stack.

Class *BaseGlobalPlanner* is an interface of the global planners used in navigation stack. There are two implementations: *CarrotPlanner* and *NavfnROS*. First one is a simplistic planner, which connects a target pose and robot actual pose with a straight line and performs collision checks along this line. The second is a grid-based A* path-planner for circular robots.

Multiple functionalities are tightly coupled in the implementation of class *BaseLocalPlanner*, i.e. trajectory generation, adaptation and execution. It generates a number of trajectories for admissible linear and angular velocities of the robot. Each trajectory is scored according to an objective function, which includes goal heading, path heading and obstacle clearance. The trajectory with

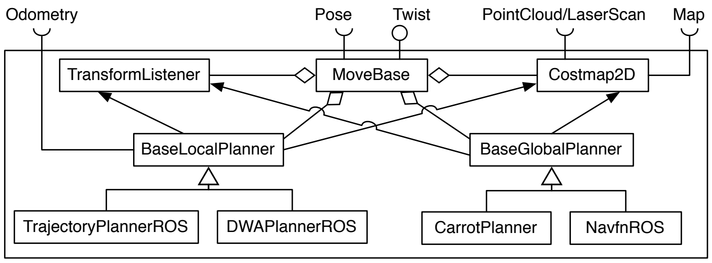


Fig. 2. Mobile base navigation component in ROS

maximum objective function is selected and its associated linear and angular velocity (twist) are sent to the robot driver. Two concrete implementations of this class are available, namely *TrajectoryPlannerROS* and *DWAPlannerROS*. Both assume implicitly that the robot has a differential drive kinematics model.

Class *CostMap2D* is an implementation of 2D occupancy grid-map. It embeds the data structures for representing a 2D tessellated representation of the environment. It is used for both path planning and obstacle avoidance.

The top-level class is the *MoveBase* class, which instantiates all the classes that implement specific functionality and starts several threads for their concurrent execution. Concurrent access to shared resources (e.g. the map of the environment) is synchronized by means of infrastructure mechanisms, such as a state machine, mutexes and numerous flags and conditions across the code. There is thus no guarantee that these functionalities are executed in real-time.

The *MoveBase* class is instantiated by a main function that starts a ROS node. Thus, all the functionalities for robust navigation are provided by a single component (ROS node). This component interacts with other components in the system (e.g. the robot base driver and the laser driver) by exchanging ROS messages. The set of exchanged messages represent the component interface. Unfortunately, the component interface is not clearly separated by the component implementation since ROS messages are produced and consumed by several classes that implement the component. Thus, the only way to understand how components interact with each others is to carefully look at the source code.

The ROS Navigation stack classes implementations are tightly coupled with the ROS infrastructure, thus they cannot be reused in different environments.

4 Product Line Design

In [4] we have defined a set of architectural principles for the development of flexible component-based systems that foster the separation of four design concerns originally identified in [13], namely *Computation*, *Coordination*, *Communication*, and *Configuration*.

According to these architectural principles, the robot functionalities are provided by *Computation* components that have harmonized interfaces and implement specific robotic algorithms. The clear separation between component interface and implementation guarantees interoperability of components that provide similar functionalities but implement different algorithms.

The mutual interactions of *Computation* components might change dynamically according to their current internal state. In order to improve their reusability, interaction policy should be implemented as finite state machines in specialized *Coordination* components that observe state transitions in the systems by listening to events notified by *Computation* components.

Typically, components rely on a middleware infrastructure to exchange data and events through the network. In order to make components implementations independent from any specific middleware, components should be designed and implemented according to an abstract component model (meta-model), which defines middleware-independent communication ports. For this purpose, similarly to the OMG initiative[2], the BRICS project has defined a new component model (BCM [9]), which provides the following architectural elements:

– Component: is a software package that encapsulates a set of related functionalities and has its own thread of control.
– Port: represents the component interface. It explicitly defines (in terms of data types and contract) how a component provides (output port) or requires (input port) a service or a data-flow.
– Property: allows the component configuration. A property provides an interface for setting the value of a parameter defined in the component implementation (e.g. period, algorithm parameters, . . .).
– Connector: defines the connection between an input port and an output port and its communication mechanism.

Computation components and *Coordination* components can be assembled to build applications. A family of similar applications that are built reusing a set of software components and share the same architecture is called a Software Product Line. The product line architecture specifies the structural (data structures and application programming interfaces) and behavioral (data and control flow) commonalities among the products and the variations reflected in each product (variation points). It prescribes how software components (variants) can be assembled to derive individual products. For example, a variation point in the robust navigation product line is the algorithm for obstacle avoidance. Different algorithms (i.e. dynamic window approach [8] or vector field histogram [17]) are implemented as distinct software components. The product line architecture guarantees that these components are interchangeable.

The possible configurations of a software product line are represented in a *product line model*, which specifies (a) a set of components that can be used for building all the possible applications of the family (some of them are mandatory, some others are instead optional) and (b) a set of connections among components (some of them are stable, some others are variable). By selecting the optional components, their specific implementation, the values of their configuration parameters, and the variable connections, the variability of the product line is resolved and a specific *application model* is defined.

[2] `http://www.omg.org/spec/RTC/1.0/`

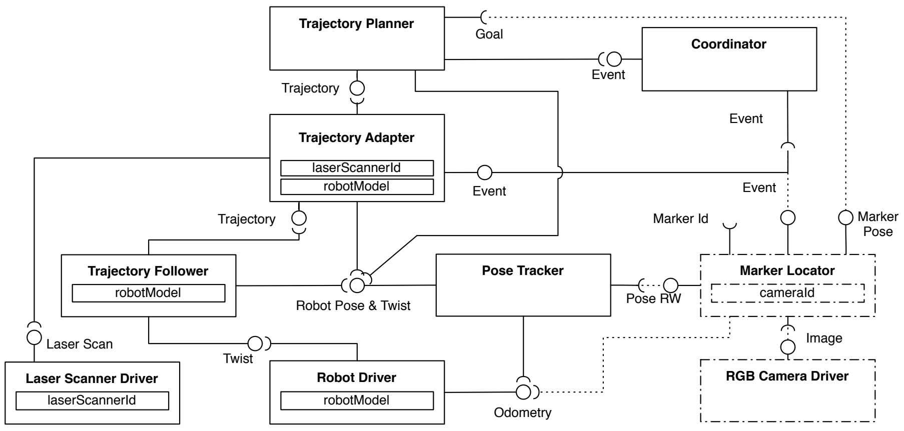


Fig. 3. The model representing the BRICS Robust Navigation Product Line

4.1 The BRICS Robust Navigation SPL

Figure 3 represents a first draft of the BRICS Robust Navigation Product Line. Continuous lines depict the default connections between input and output ports while dashed lines represent the optional connections that may be created to configure a specific application. Continuous boxes represent mandatory components, while dashed boxes represent optional components. Boxes inside components indicate their properties. More specifically:

- **Trajectory Planner** implements the motion planning and trajectory generation functionalities. It gets a goal position and the current robot position as input and produces a trajectory that is a vector of poses with twist.
- **Trajectory Adapter** interpolates the precomputed trajectory and produces an obstacle-free trajectory toward the next waypoint taking into account the sensor information produced by the laser scanner.
- **Trajectory Follower** receives the adapted trajectory and the robot estimated pose and produces as output a twist for following the input trajectory.
- **Robot Driver** drives the physical robot. It receives twist commands and produces the robot odometry.
- **Laser Scanner Driver** reads the raw data from the device and produces as output the laser scans expressed as a vector of distances and angles.
- **Pose Tracker** keeps track of the current pose and twist of the robot. It fuses odometry estimates with position estimates computed by other components.
- **Marker Locator** is an optional component, which is in charge of localizing visual markers placed in the environment and computing their positions with respect to a global reference frame. It receives as input an image and the odometry of the robot and produces as output the absolute marker position.
- **RGB Camera Driver** is an optional component, which reads data from the RGB camera and produces as output an RGB image.

– **Coordinator** implements the coordination logic among components. It monitors events generated by components that could represent abnormal situations (e.g. the Trajectory Adapter cannot generate a trajectory to avoid an obstacle) and generates events that triggers state changes in other components (e.g. the Trajectory Planner should plan a new trajectory).

There are some important differences between the proposed solution and the ROS navigation stack discussed in Section 3. First of all the navigation functionalities are mapped to finer grained components. Each component has only one thread of control. This allows to replace individual functionalities easier and to select the most appropriate frequency for each functionality. Accordingly, the trajectory follower and the trajectory adapter functionalities are implemented in two different components. This separation reflects the different operating frequencies of the two components: the Trajectory Follower runs at a higher frequency, as required by the closed loop position and velocity control algorithm. On the contrary the Trajectory Adapter component computes a new output only when receives a new laser scan or a new trajectory.

Unlike in ROS, the coordination mechanisms have been made independent from the components implementations by introducing a Coordinator component.

5 Variability Modeling

Building software systems according to the product line approach is economic and efficient [5]. Most work is about integration, customization, and configuration instead of creation. A system configuration is an arrangement of components and associated options and settings that completely implements a software product. Variants may exclude each others (e.g., the selection of a component implementing an indoor navigation algorithm excludes the choice of components providing GPS-based localization services) or one option may make the integration of a second one a necessity (e.g., a component implementing a visual odometry algorithm depends on a component that supplies images of the environment). Hence, only a subset of all combinations is the admissible configuration.

In order to model and symbolically represent the product line variation points, their variants and the constraints between them, a formalism called Feature Models was introduced in 1990 in the context of the Feature Oriented Domain Analysis (FODA) approach [11]. These models make explicit the variability that is implicitly defined during the product line design.

A feature model is a hierarchical composition of features. A **feature** defines a software property and represents an increment in program functionality. Composing features, i.e. selecting a subset of all the features contained in a feature model, corresponds to select a specific product (application) that belongs to the product line described by the model. This selection is usually called **instance**.

Feature models are organized as a tree and the root feature, also called **concept**, defines the application family. Parent features are connected to children features by means of edges, which represent containment relationships. Features can be discerned in two main categories: **mandatory** and **optional**. *Mandatory* features have to be present in all the application of the product line (commonalities). They are graphically depicted by means of a black circle on the top. *Optional* features instead can be present but they are not mandatory (variation points). They are depicted by means of a white circle on the top.

Three types of containment relationships define containment constraints between the parent and the children features: **one-to-one**, **or** and **alternative**. The *one-to-one* containment means that the parent feature can (or has to) contain the child feature. The other two kinds of containments are containments between the parent feature and its children features. Here the parent feature is the variation point, while the subfeatures the variants. The *or* containment means that from the children features at least one has to be present in the application. The *alternative* containment (X-Or) instead means that from the children features only one can be present in the application.

Feature models also define two kinds of constraints between the features: **requires** and **excludes**. These constraints allow the definition of a subset of valid configurations. The *requires* constraint means that if a feature A is selected, then also a feature B has to be selected. The *excludes* constraint instead means that if a feature A is selected, then a feature B cannot be selected.

[10] presents an Eclipse plugin that allows the design of feature models. This plugin provides a formal meta-model (Ecore based), which defines the rules for creating feature models conform to the standard specification introduced above.

5.1 Variability in the BRICS Robust Navigation SPL

Figure 4 depicts the feature model describing the variability of the BRICS Robust Navigation Product Line. Gray boxes represent the **or** and **alternative** containments relationship and show the cardinality (1...n represents **or** and 1...1 represents **alternative**). It contains four main variation points:

- **Localization**: this variation point regards the information used for localizing the robot. Using the odometry is mandatory but a marker locator can be used in order to improve the pose estimation.
- **Navigation**: this variation point regards the navigation strategy. Two variants are available: map-based and marker-based. In the first case a Trajectory Planner computes a collision-free path according to the goal received as input from its client. In the second case instead the goal is provided by the Marker Locator and represents the position of a specific marker.
- **Obstacle Avoidance**: this variation point regards the algorithm used for avoiding obstacles. Two variants are available: Dynamic Windows Approach and Vector Field Histogram.
- **Sensors**: this variation point regards the sensors used in the application. Using the laser scanner is mandatory for obstacle avoidance. However, in order to recognize visual markers, the robot equipment can be extended by using a front camera and a camera held on a specific support.

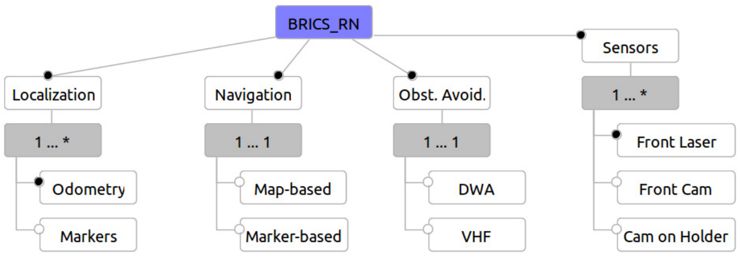


Fig. 4. The model representing the variability in the BRICS RN Product Line. It was designed with our Feature Model plugin.

The feature model also defines the following three constraints that limit the allowable combinations of features. (a) The use of the marker-based navigation strategies requires at least one of the two cameras. (b) The same requirements are valid when we want to use the markers for localization. (c) The selection of a marked based navigation excludes the use of the markers for the localization.

6 Variability Resolution

The last phase of development process regards the resolution of the variability in the Product Line model and has the goal of producing the model of a specific application. Thanks to our tool this can be done by selecting the set of variants (features), which reflect the application requirements, directly on the feature model. This selection should satisfy the explicit constraints, the containments cardinalities and the selection of the mandatory features defined in the feature model. The tool automatically checks the constraints satisfaction and only successively generates the application model, which can be transformed in the deployment file of a specific middleware by means of the model-to-text transformation provided by BRIDE.

In order to define how the Product Line model has to be modified for producing the model of a specific application, we introduced the concept of *transformation*. A transformation is an action that modifies the architectural elements of a Product Line model in order to replace a variation point with a specific variant. Different kinds of transformations are available, according to the elements that have to be modified (components implementations, connections, properties). The developer can associate to each feature one or more transformations. A selection of a set of features will hence result in the execution of a set of transformations.

Figure 5 illustrates how the meta-model of the feature models described in [10] has been extended for representing the transformation associated to the features. Currently it has been done only for the generation of Orocos RTT application. For this reason the meta-model has three links to three elements defined in the RTT-BCM meta-model: Task Context, Connection Policy, and Property [1]. However the approach can be easily extended to other middlewares such ROS and SCA [14]. The following list describes the different kinds of transformations.

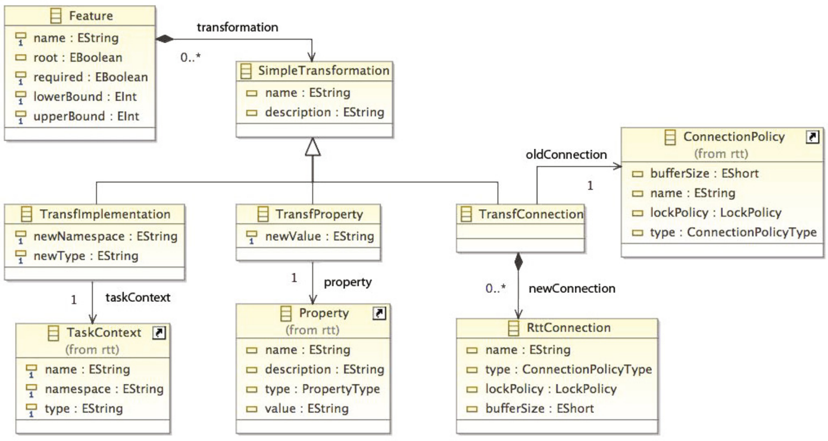


Fig. 5. The extension of the feature model meta-model that specifies how transformations are mapped to RTT elements (classes with the arrow on the top right corner)

- ***TransfImplementation***: defines which implementation will be used for a certain component, i.e. which algorithm (Computation and Coordination). The developer specifies a link to the desired component (defined in the Product Line model) and the class that implements its interface (by means of the namespace and the class name).
- ***TransfConnection***: defines how the components will be connected (Composition). The developer specifies a link to a connection that has to be removed from the Product Line model (if needed) and a set of connections that have to be created (with specific lock policies and the buffer sizes). As a result of these transformations unconnected components will be removed from the model.
- ***TransfProperty***: defines the value of a certain property (Configuration). The developer specifies a link to the desired property (defined in the Product Line model) and the value he wants to assign to it.

6.1 Variability Resolution in the BRICS Robust Navigation SPL

The product line presented in the previous sections allows the deployment of different applications that can be classified according to two navigation strategies: map-based and marker-based. The different strategies can be derived by resolving the *Navigation* variation point.

If a map-based navigation is chosen, then the Marker Locator, the RGB Camera Driver components and all the connections between these components and the others are not necessary. So these components can be removed from the model. In case of marker-based navigation instead, the Trajectory Planner component receives the goal from the Marker Locator. Hence the optional connections of Marker Locator and RGB Camera Driver have to be created.

In addition to these connections, the Coordinator implementation and its connections have to be defined according to the navigation strategy. The Coordinator always recovers from situation in which the Trajectory Adapter is unable to adapt the trajectory. This can happen due to the presence of obstacles all around the front part of the robot. In these cases, the Coordinator receives an event from the Trajectory Adapter and returns an event to the Trajectory Planner asking it to recompute a path. When a marker-based strategy is chosen instead, the Coordinator resolves also situations in which no markers can be detected. In these cases the Coordinator receives an event from the Marker Locator an returns an event to the Trajectory Planner asking it to create a trajectory for moving the robot in such a way to seek new markers. This configuration needs a connection between the Event interfaces of Marker Locator and Coordinator.

From this point several applications belonging to the two groups can be derived by resolving the other variation points. The Localization based on marker can be deployed by connecting the Pose RW interface of the Marker Locator to the Pose Tracker. The obstacle avoidance variation point can be resolved by setting the implementation of the Trajectory Adapter in order to use one of the two algorithms (DWA or VHF). Finally the Sensors variation point can be resolved by setting the implementation of the sensors drivers and configuring the property of the MarkerLocator in order to specify the camera Id. This value is needed in order to retrieve the camera position from the robot kinematic model.

The feature model depicted in figure 4 defines for each feature the transformations that allow the resolution of the variation points. Some examples of the transformations associated to the features are reported below.

- Feature *Marker-based*. (a) Connection transformation. New connections between: Goal interface of Trajectory Planner and Marker Pose interface of Marker Locator, Event interfaces of Coordinator and Marker Locator, Event interfaces of Trajectory Adapter and Coordinator. (b) Implementation transformation: set the implementation of the coordinator for resolving situations in which no markers are visible.
- Feature *DWA*. In this case a single Implementation transformation is defined, which sets the implementation of the Trajectory Generator for using the Dynamic Window Approach.
- Feature *Front Camera*. (a) Property transformation: set the value of the cameraId property of the marker locator to the Id of the front camera. (b) Implementation transformation: set the implementation of the RGB Camera Driver for using the driver of the appropriated camera.

7 Conclusions

In this paper we have presented the BRICS software development process for robotic component-based systems that builds on the concept of software flexibility. The key to achieving software flexibility is the possibility to identify the class of changes that are likely to occur in the environment over the lifespan of robotic software components and that affect components and systems portability, interoperability, and reusability.

The BRICS approach to robotic software development consists in refactoring existing open source robotic libraries according to flexible architectures, which clearly separate stable and variables characteristics.

By explicitly modeling software variability, it is possible to build new applications by selecting and integrating components that provide concrete implementations (variants) of robotic functionalities (variation points).

Acknowledgments. The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. FP7-ICT-231940-BRICS (Best Practice in Robotics). The authors would like to thank all the partners of the project for their valuable comments.

References

1. The RTT meta model, http://www.best-of-robotics.org/bride/rtt.html
2. Brugali, D., Nowak, W., Gherardi, L., Zakharov, A., Prassler, E.: Component-based refactoring of motion planning libraries. In: IEEE/RSJ Int. Conference on Intelligent Robots and Systems (IROS), pp. 4042–4049 (2010)
3. Brugali, D., Scandurra, P.: Component-based robotic engineering (part i)[tutorial]. IEEE Robotics & Automation Magazine 16(4), 84–96 (2009)
4. Brugali, D., Shakhimardanov, A.: Component-based robotic engineering (part ii)[tutorial]. IEEE Robotics & Automation Magazine 17(1), 100–112 (2010)
5. Clements, P., Northrop, L.: Software Product Lines: Practices and Patterns. Addison-Wesley (2002)
6. Demeyer, S., Ducasse, S., Nierstrasz, O.: Object-oriented reengineering patterns. Morgan Kaufmann (2008)
7. Fowler, M., Beck, K.: Refactoring: improving the design of existing code. Addison-Wesley Professional (1999)
8. Fox, D., Burgard, W., Thrun, S.: The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine 4(1), 23–33 (1997)
9. Garcia, H., Bruyninckx, H.: Tool chain (bride) delivered as brics software distribution. BRICS Deliverable 4.4 (2011)
10. Gherardi, L., Brugali, D.: An eclipse-based feature models toolchain. In: 6th Italian Workshop on Eclipse Technologies, EclipseIT 2011 (2011)
11. Kang, K.: Feature-oriented domain analysis (FODA) feasibility study. Technical report, DTIC Document (1990)
12. Minguez, J., Montano, L.: Nearness diagram (nd) navigation: collision avoidance in troublesome scenarios. IEEE Transactions on Robotics and Automation (2004)
13. Radestock, M., Eisenbach, S.: Coordination in Evolving Systems. In: Spaniol, O., Meyer, B., Linnhoff-Popien, C. (eds.) TreDS 1996. LNCS, vol. 1161, pp. 162–176. Springer, Heidelberg (1996)
14. Service Component Architecture (SCA), http://www.osoa.org
15. Schmidt, D.: Guest editor's introduction: Model-driven engineering. Computer 39(2), 25–31 (2006)
16. Siciliano, B., Khatib, O.: Springer handbook of robotics. Springer-Verlag New York Inc. (2008)
17. Ulrich, I., Borenstein, J.: Vfh+: Reliable obstacle avoidance for fast mobile robots. In: IEEE Int. Conference on Robotics and Automation (1998)

SwarmSimX: Real-Time Simulation Environment for Multi-robot Systems

Johannes Lächele[1], Antonio Franchi[1], Heinrich H. Bülthoff[1,2], and Paolo Robuffo Giordano[1]

[1] Max Planck Institute for Biological Cybernetics, Spemannstraße 38, 72076 Tübingen, Germany
{johannes.laechele,antonio.franchi,prg}@tuebingen.mpg.de
[2] Department of Brain and Cognitive Engineering, Korea University, Seoul, 136-713 Korea
hhb@tuebingen.mpg.de

Abstract. In this paper we present a novel simulation environment called SwarmSimX with the ability to simulate dozens of robots in a realistic 3D environment. The software architecture of SwarmSimX allows new robots, sensors, and other libraries to be loaded at runtime, extending the functionality of the simulation environment significantly. In addition, SwarmSimX allows an easy exchange of the underlying libraries used for the visual and physical simulation to incorporate different libraries (e.g., improved or future versions). A major feature is also the possibility to perform the whole simulation in real-time allowing for human-in-the-loop or hardware-in-the-loop scenarios. SwarmSimX has been already employed in several works presenting haptic shared control of multiple mobile robots (e.g., quadrotor UAVs). Additionally, we present here two validation tests showing the physical fidelity and the real-time performance of SwarmSimX. For the tests we used NVIDIA® PhysX® and Ogre3D as physics and rendering libraries, respectively.

Keywords: Real-Time, Multi-Robot, Simulation Environments, Software Framework.

1 Introduction

Software frameworks simulating the behavior of virtual environments are an indispensable tool in most engineering sciences. Within the robotics scope, simulation environments are of paramount importance for fast development and testing of new control algorithms for single robots or of complex behaviors for multiple interacting robots.

In this latter case, several software suites able to simulate multiple robots at the same time have been developed and are widely used in research. Simulators like ARGoS [1] are capable of handling multiple robots with a pure modular software design that allows for assigning different physics engines to different areas of the simulation. A simulation example involving thousands of robots is discussed, albeit only in a 2D environment. Also, the design of ARGoS is not specialized for *real-time* (RT) simulation, an essential feature for hardware-in-the-loop scenarios and for all those situations involving strict constraints on the inner simulation timing (e.g., whenever requiring online processing/filtering of signals acquired from the external world).

The crucial requirements that we identified for a robotics simulator have been the following: real-time execution, physical realism, exchangeable visual and physical representation, extendable software architecture, and full control over inherent information of all simulated robots (see, e.g., [2–4]). To the best of our knowledge, we were unable to find a solution meeting all of the requirements. Existing simulation environments, like OpenRAVE [5], MORSE [6] can be used for multi-robot simulation scenarios. MORSE is based on the open source 3D content creation suite Blender [7] and its game engine architecture. Program logic, algorithms or general extensions to the simulation software can be implemented in Python.Python is a highly portable, open source programming language and usually all code is being interpreted or compiled just-in-time (JIT). Third party system libraries or high performance low-level code can to be loaded and executed at runtime using wrapper mechanisms. Also OpenRAVE provides an extensive Python API enabling the user to easily add functionality to the simulation or any parts of it. However, using the Python approach may be a source for errors that are hard to trace and resolve. Additional effort is required to include libraries that in a later step may even prove difficult to interface with the real robots. Other simulation environments such as V-REP [8] or Webots [9] provide a more general simulation environment and may also be extended with RT capabilities. Nonetheless, these two software systems are tightly coupled to the underlying libraries used for the physical simulation, thus removing the possibility to exchange the technology being used in an easy fashion.

The only environment meeting almost all requirements is Gazebo [10], a versatile simulation environment following a modular software architecture design. Gazebo already supports a vast set of mobile robots and manipulators, as well as their sensors and control algorithms and is developed by an active community. Still, Gazebo is not orignally designed to run in real-time, hence colliding with the requirements stated earlier. Although a custom built plugin may alter the stepping of the simulation, the already provided libraries, plugins, and sensors may not respect real-time execution. In addition, Gazebo currently only supports the Open Dynamics Engine (ODE) as the core physics engine. The authors believe that other libraries, e.g. NVIDIA PhysX engine [11], may yield better, i.e. more precise, simulation results.

In this paper, we propose a novel Simulation environment, called SwarmSimX (SSX), able to address all the aforementioned challenges. SwarmSimX, which is currently available in a C^{++} implementation, is capable of simulating both the visual and physical properties of robots acting in a user-defined environment in real-time. Shared modules may be loaded at runtime, extending the simulation with new functionalities. SwarmSimX is also designed to be independent of the particular physics and render engines used in the simulation. The un-

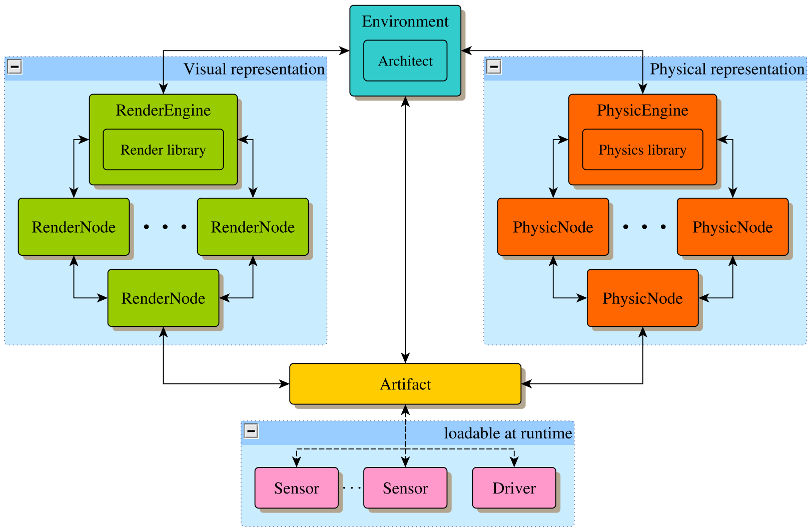


Fig. 1. Overview of the software architecture of SwarmSimX

derlying engines can be flawlessly exchanged in the variegate assortment of the current software depending upon the user needs (e.g., realism, efficiency, open-source, high-accuracy for a particular robotic platform, etc.). This feature also comes particularly in handy whenever major upgrades of the engine's library are released and the simulator has to follow these updates as well.

The rest of the paper is organized as follows: Sect. 2 illustrates the design principles and software architecture behind the inner workings of SwarmSimX, Sect. 3 reports the results of experiments aimed at validating the realism and real-time capabilities of the simulation software, and Sect. 4 draws concluding remarks and discusses future directions. A copy of SwarmSimX can be downloaded from the subversion repository at https://svn.kyb.mpg.de/kyb-robotics. Additional media can be found in the video section at http://laechele.eu/SwarmSimX/.

2 Software Architecture

The SSX simulation environment can be divided into three main parts: the visual representation; the physical representation; and *Artifacts*. Figure 1 gives an overview of the elements used in the simulation and how they are related among themselves. In the nomenclature of SSX, the visual representation is managed by the RenderEngine with individual elements being represented by RenderNodes. Symmetrically, the physical representation is managed by the PhysicEngine and the individual parts are called PhysicNodes. Both, Physic- and RenderNodes can be connected to form tree structures. Child nodes are defined w.r.t. the parent node to which they are associated, and may contain information about the position, orientation, mass, and similar quantities. Because of the abstract nature of the RenderEngine and PhysicEngine the whole software framework is not affected by a particular implementation using some specific *Render Library* or *Physics Library*.

The visual and physical representations are related to each other, as the physical representation is used to perform the actual simulation and the visual representation displays the behaviour of the objects being simulated. This relationship is expressed in the parallel structure of the design layout. At every timestep of the simulation, the execution of logical modules (namely *Drivers* and *Sensors*) is triggered. These modules can perform any kind of computation and are used to extend the simulation with custom logic or functionalities. Drivers implement the behaviour of objects within the simulation environment. A Driver can represent the control program of an *Unmanned Aerial Vehicle* (UAV) that applies the appropriate input forces and torques to the physical object in order to attain the desired angles received by an external navigation algorithm. Another example is the logical module of an automated door that opens whenever an object moves within a certain range and closes after a given period of time with no new sensor input. Sensors are used whenever information concerning the simulated environment or the simulator variables needs to be retrieved, e.g., in order to emulate the measurements of a real transducer or to obtain the current simulation time. For example, this allows an easy porting of a control algorithm implemented for real hardware to the simulation by properly emulating all the needed sensory inputs. One can also conceive virtual sensors able to measure the global state of the simulated environment. These are not meant to represent real sensor units, but rather to provide a helpful tool while developing and debugging new algorithms.

All these parts are stored together into a single object called *Artifact*. Artifacts state the main concept of SwarmSimX. Everything that can be placed in the simulation environment is represented as an Artifact. An Artifact may contain multiple or no references to RenderNodes and PhysicNodes. Also, Sensors and Drivers are not mandatory.

2.1 Main Execution Loop

The execution of the simulation timesteps is solely the responsibility of the Environment. At the beginning of a timestep, the update of the PhysicEngine gets triggered. After this event, the Drivers and Sensors of the simulation get the chance to perform their computation. This particular step exploits the parallel computing power of modern CPUs by triggering the execution of a separate thread associated with each Artifact which contains at least one Sensor or Driver. The workload of computations posed by user code is spread among the CPUs of the system, hence increasing the performance.

The execution of the threads needs to be synchronized with the triggering of the simulation timestep, meaning that the Environment waits until the calculation of all threads is complete before proceeding. Finally, the remaining time until the next timestep is due is calculated as $t_\mathrm{w} = t_\mathrm{d} - t_\mathrm{n}$, with t_d being the desired simulation time, i.e., the number of simulation timesteps N times the timestep τ, and t_n the current *wall-clock-time* (WCT). The execution of the

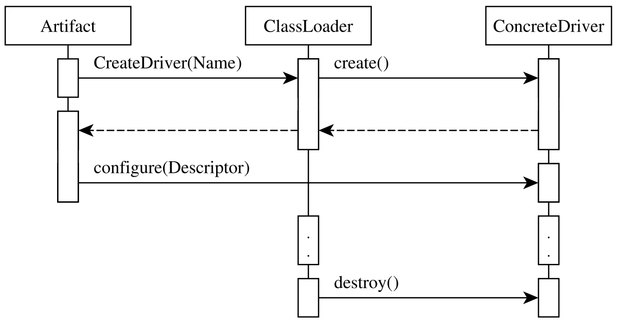


Fig. 2. Interaction involved when creating a ConcreteDriver that implements the Driver interface and the `create`, `destroy` functions both defined in an `extern "C"` {}-block

simulation is stopped for t_w using precise wait methods, if and only if t_w is positive, otherwise the next step is triggered immediately. In this case a warning message is issued stating the break of the RT-constraint.

It is important to note that the aforementioned wait step is most significant for the RT-capabilities of SSX. Several different libraries support precise methods for waiting and a previously conducted experiment showed the best results when utilizing the boost [12] library. Note also that, in the case of an offset between simulation time and WCT, this triggering paradigm will recover this error assuming the execution time of subsequent timesteps is smaller than τ.

2.2 Extending the Simulation

The goal of being able to create custom robots with sensors attached and program logic demands that the custom parts are independent from the simulation environment. Recompiling the whole simulation environment after adding new robots is time consuming and redundant. Even worse, exchanging projects between different users may prove difficult, as the code is tightly coupled to a certain simulation environment.

SwarmSimX provides the possibility to load program code of Drivers and Sensors at runtime. In both cases an abstract interface defines methods for creating, configuring, running and destroying the class representing the program logic. Drivers are always associated with an Artifact and only one Driver per Artifact may be defined. Sensors are also associated with an Artifact but, in contrast to the Driver, multiple Sensors per Artifact may be created.

Drivers, Sensors, and SimulationExtensions are loaded dynamically, i.e., at runtime, exploiting the features of the system libraries. The C language provides an easy to use library for loading and calling external libraries dynamically, called `dlopen`. For this, two functions are defined in a `extern "C" {}`-block that are responsible for creating and destroying the concrete class. The class itself derives from the `Driver`, `Sensor` or `SimulationExtension` abstract class that define a set of virtual methods.

Driver. The interface definition of the Driver is very slim compared to other classes. Methods for configuring the Driver using a Descriptor, setting the associated Artifact and Environment, and calculating a simulation step are provided. In addition to the interface definition, two functions need to be implemented for every Driver. As mentioned before, these two functions are necessary to dynamically load libraries and create instances of any Driver implemented by the user.

Artifacts are responsible for creating their Driver defined by the Artifact XML file. Figure 2 shows the interaction involved in creating a Driver. First an Artifact requests a Driver instance given the name. Then the *ClassLoader* loads the library and calls the create-method. If all these steps were successful, the ClassLoader returns an instance of the specified Driver. In a final step the Artifact registers itself to the Driver and triggers the configuration.

The ClassLoader is responsible for freeing all Driver instances created during the lifecycle of the simulation and is done when the simulation environment gets shut down by the user. This ensures that no method call will reach an already freed Driver, resulting in a segmentation error.

Sensor. Sensors provide information about the simulation environment. Utilizing features of the *Environment*, Sensors can access both engines and gain access to all created Artifacts of the simulation. All possible information is therefore available to the Sensor. Information provided by the Sensor is distributed to all registered *SensorCallBacks*. This implementation follows the Observer Pattern [13], which allows for distributing information without the need of polling for data. In this case the Sensor itself is responsible for triggering the update process. Loading Sensors at runtime follows the same paradigm as loading Drivers.

All Sensors are associated with the corresponding Artifact, but the Driver is responsible for registering SensorCallbacks. This responsibility can either be implemented in a separate class, which handles the Sensor output or within the Driver itself. As described earlier, Drivers have access to the Environment and therefore access to all Artifacts created in the virtual environment. The Sensor class does not check affiliation of the registered SensorCallBack. Therefore it is possible to register a callback at a Sensor that is associated with a different Artifact. This allows for a very fast way of sharing information between different robots represented by Artifacts, because no data needs to be copied, serialized or transmitted to other robots.

SimulationExtension. Drivers and Sensors allow Artifacts to be extended in their functionality and they may also be reused within the context of the SSX framework. But true reusable software must not be limited to a certain framework or even programming language [14].

SimulationExtensions provide a very simple interface with methods for initialization and shutdown of the extension. Libraries may be loaded by these modules to form bridges to other software frameworks. This approach ensures the reusability of SSX as a simulation environment module that is part of a more complex *Robotics Software System*. The init methods of the extensions are called by the simulation before the actual configure and start steps of the Engines. The shutdown-method is called after all Engines have stopped their execution. This paradigm of “first in, last out” ensures that the extensions are always valid while the simulation is running. A common use-case of this feature is the connection of SSX with some kind of middleware that is interacting with the actual control program of the robot. A SimulationExtension providing ROS [15] support has been implemented and used for the experiments performed during this work. Drivers and Sensors may request a reference to the ROS extension at runtime from the Environment. The ROS extension stores a main ROS node and is responsible for managing an asynchronous spinner thread. Using the main ROS node, new topics may be published or callbacks may be registered to already existing topics.

3 Validation

Our simulation environment SSX has been successfully employed to validate several multi-robot control algorithms proposed in recent papers by some of the authors, see, e.g., [16–18] and http://www.youtube.com/user/MPIRobotics for the related video selection. In the majority of these works we also performed real-robot experiments using the same algorithms and we observed a remarkable similarity between simulative and experimental results. In addition, in these works we effectively used SSX in two challenging scenarios requiring real-time simulation: (1) simulation of robots interacting with one or more human operators by means of real haptic interfaces, and (2) simulation of several obstacle sensors providing virtual measurements to the controllers of real robots in order to simulate virtual obstacles in the real environment. We refer the reader to those works in order to appreciate the use of SSX in an applied robotics context where fidelity and real-time are of extreme importance. In addition to that, in this section we report two quantitative studies investigating the fulfillment of two cardinal requirements of any robotic simulator: the physical fidelity, and the real-time capabilities with multiple robots. For these tests we used a single Linux machine running Ubuntu[19] 12.04 with the generic Ubuntu Linux kernel 3.2.0. The machine has a Intel® Xeon® CPU W3520 (endowed with 8 cores – HyperThreading enabled), 12 GB of main memory and a NVIDIA® GeForce®

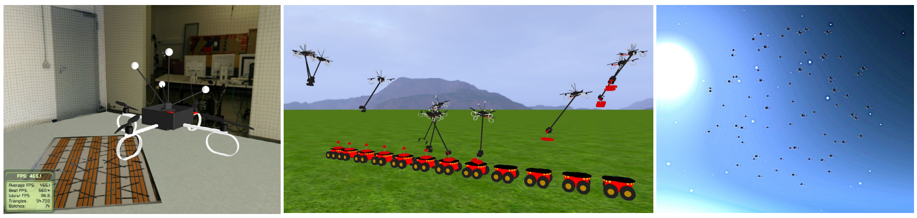


Fig. 3. Three screenshots taken from different simulations. *Left:* simulated quadrotor during the *physical fidelity* test. *Center:* stroboscopic sequence of a cooperative-aerial grasping performed by a quadrotor and a ground robot. *Right:* 70 quadrotors performing autonomous formation control on a spherical surface during the *real-time capabilities vs. number of robots* test.

9800 GT GPU. The version of SwarmSimX used for all experiments utilizes NVIDIA® PhysX® [11] 3.2.0 and Ogre3D [20] 1.7.4 as the Physics Library and Render Library, respectively.

The focus of this section is the validation of the quality of the physical simulation and the RT capabilities. Profound validations of the rendering performance of SSX in general and Ogre3D in particular have been omitted. For an impression of the quality of the rendering performed by Ogre3D we refer to Fig. 3 showing screenshots of different experiment scenarios.

3.1 Physical Fidelity

The main purpose of a simulation environment is to test and analyze new algorithms in a safe and controlled environment before porting them to the real world case. The more the behavior of a simulated robot is comparable to that of its real counterpart, the better the chances that an algorithm working in simulation will work in reality with comparable results. This is depending almost solely on the degree of physical fidelity of the simulation environment being used.

In order to test the physical fidelity of SSX we compared the tracking performances of a real and a virtual quadrotor flying two different eight-shaped trajectories, a vertical and a horizontal one, while maintaining a constant yaw velocity. The global position, orientation, linear and angular velocities of the real quadrotor have been recorded by means of an external motion capture system and used for the comparison.

In order to minimize the effect of external factors in the comparison, we used exactly the same flight-controller code and control parameters (e.g., gains) both for the virtual and the real cases. In particular we implemented a standard cascaded controller composed by two nested control loops. The outer loop controls the position of the center of mass of the robot: it reads the current robot state (e.g., the robot position and velocity) and provides the appropriate orientation and thrust to the inner control loop in order to track the desired centroid trajectory. The inner loop lets the quadrotor achieve the appropriate orientation by acting on the propeller speeds, i.e., on the total torque of the aerial vehicle. Nevertheless, to simulate in real-time the extremely complex aerodynamics of all the propellers is practically unfeasible. Therefore we experimentally identified the non-linear map that relates the rotational speeds to the attained forces and torques. This allowed us to efficiently simulate the UAV dynamics by directly applying the forces and torques resulting from that map to the UAV body.

For the sake of fidelity we also used in simulation the same code executing the inner loop on the real quadrotor. This is implemented using only integers, runs on a fixed-point microcontroller, and uses the measurements coming from an onboard Inertial Measurement Unit (IMU). For this purpose a virtual IMU has also been implemented to provide linear acceleration and angular velocity with noise characteristics similar to that of the IMU mounted on the real quadrotor.

The cumulative distribution of the tracking error (in short: *cumulative error*) is a valid tool to give an overall view of the tracking behavior of a robot following a desired trajectory. In order to define this function, consider a desired trajectory

$\boldsymbol{p}^d$ defined over a time interval $[t_0, t_f]$ and the actual trajectory executed by the robot $\boldsymbol{p}$ in the same interval, the cumulative distribution function is defined as:

$$F_{\boldsymbol{p},\boldsymbol{p}^d,[t_0,t_f]}(d) = \frac{1}{t_f - t_0} \int_{t_0}^{t_f} H(\|\boldsymbol{p}^d(t) - \boldsymbol{p}(t)\| - d)\, dt,$$

where $d \geq 0$ represents a distance, and $H : \mathbb{R} \rightarrow \{0, 1\}$ is the Heaviside (or unit-step) function, which returns 0 when its argument is negative and 1 otherwise. In other words the cumulative error returns, for every distance d, the percentage of time in which the actual trajectory $\boldsymbol{p}$ has been closer than d to $\boldsymbol{p}^d$. The faster F converges to 1 for increasing d the better is the tracking behavior of the controller.

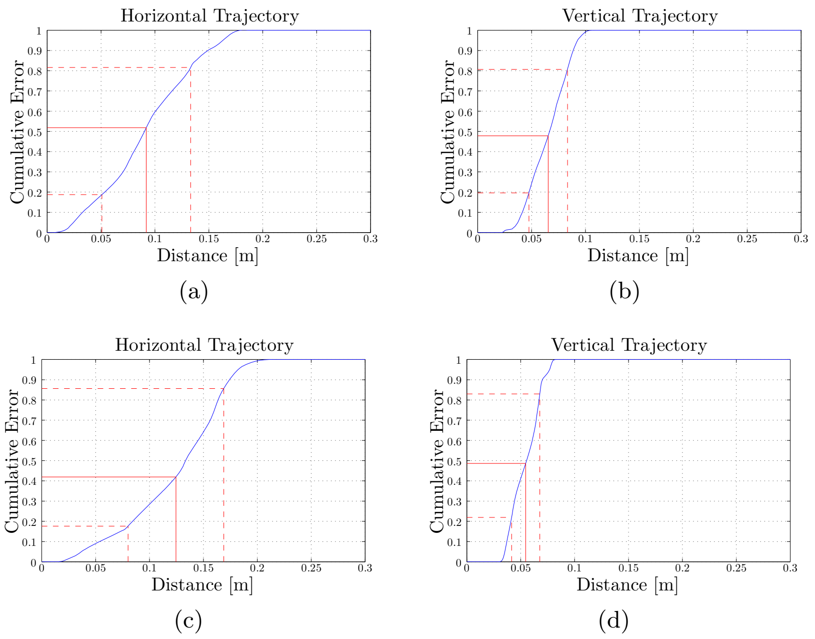


Fig. 4. Comparison between the cumulative distributions of the tracking errors. (a): Real quadrotor hori. traj. (mean: 0.0919 m, std: 0.0412 m); (b): Real quadrotor vert. traj.(mean: 0.0654 m, std: 0.0179 m); (c): Simulated quadrotor hori. traj. (mean: 0.1245 m, std: 0.0444 m); (d): Simulated quadrotor vert. traj. (mean: 0.0546 m, std: 0.0131 m).

Nevertheless, here we are not interested in the absolute tracking performances of the two controllers but in the degree of similarity between the simulated and real behaviors. In fact, assuming the same software is controlling each quadrotor and using similar sensor values, a good simulator should show the same tracking behavior compared to the real case for the same desired trajectories. Only in this way testing a new controller in simulation can give a useful insight about the applicability of the proposed controller in reality. The results of our experimental test, showing the fidelity of our simulation, are reported in Fig. 4. Both in the case of horizontal (Figs. 4(a),(c)) and vertical (Figs. 4(b),(d)) trajectories it is possible to appreciate the similar shape of the cumulative-error plots between the simulated and the real cases. Note also that, as usual for the quadrotor, the vertical tracking performs much better than the horizontal one (in both the virtual and real case).

3.2 Real-Time Capabilities vs. Number of Robots

One feature of SSX is the capability to simulate dozens of 3D robots simultaneously in real-time. The possibility of obtain a real-time simulation depends mainly upon two factors: (1) the desired simulation time-step, i.e., the resolution of the physical integration, and (2) the size of "extra" computation that all the artifacts in the environment require at every time-step, e.g., measurement acquisition, estimation/control computation, inter-robot communication. Clearly, the influence of the second factor increases as the number of robots increases.[1] For users interested in real-time multi-robot applications, a basic information is given by how many robots can be simulated with a given physical accuracy (simulation time-step) and a given multi-robot coordination algorithm (comprising sensors, estimators, control, and communication). Therefore we conducted a Monte Carlo study to evaluate how the execution time of a whole simulation step is influenced by the number of simulated robots in SSX.

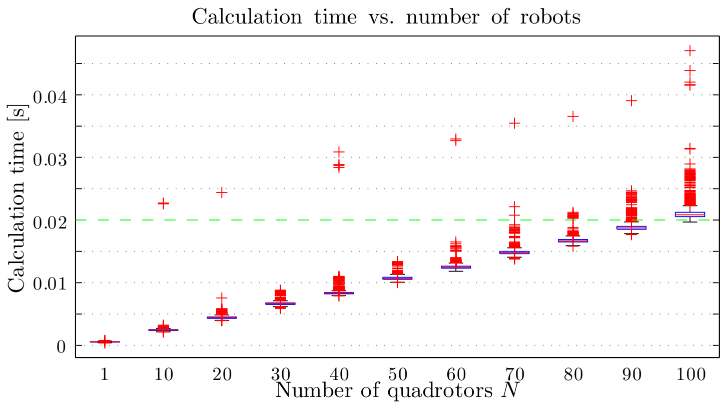


Fig. 5. Box-plots of the time needed for calculating a single timestep of SSX as a function of the number of quadrotors being simulated. Red crosses, red horizontal lines, blue boxes, and black whiskers represent outliers, median values, percentile margins, and max-min values (without outliers), respectively. The simulation timestep ($\tau = 0.02\,\mathrm{s}$) is denoted with a green dashed line.

As reference scenario we considered a group of quadrotors implementing a standard formation-control plus obstacle-avoidance algorithm based on artificial potentials. Like in the previous experimental test, every quadrotor runs a flight

[1] The influence of the number of robots on the first factor is almost negligible for NVIDIA® PhysX® used by SwarmSimX, up to a few thousands of rigid bodies.

controller able to command the torques generated by every propeller in order to stabilize the flight. In addition to that, in this test every robot runs the coordination/avoidance algorithm and a range sensor simulator to retrieve the relative position of other robots and objects within its neighboring environment.

Fig. 5 depicts our results of the comparison of the real-time capabilities of SSX and the number of robots included in the simulation. In total 11 different cases have been tested, each with a different number of robots being simulated simultaneously for a total of about 5 thousand samples of the simulation time per case. In all cases the simulation timestep was set to 0.02 s, i.e. 50 Hz, denoted with a green dashed line in Fig. 5. For each case a boxplot displays the median, 25th and 75th percentiles, and max/min values (using whiskers) of the computation time after removing extreme data points considered outliers (also plotted in the figure).

The median lines, percentile boxes, and whiskers stay compact in almost all the cases, showing a very small variance in the computational time needed to calculate a timestep. This behavior is essential for the RT reliability of the simulation. Note also, that the median calculation time increases almost linearly with the number of robots, resulting from the distributed formation control of the group of quadrotors. SwarmSimX was able to hold the RT-constraint of the timestep in almost all cases except the case with 100 quadrotors. Notice that in this case the median and percentile box is above the 0.02 s mark, resulting in a growing error offset between simulation and wall-clock time. Although the RT-constraints will always be broken in this case the simulation still produces valid results useful for offline simulation of algorithms, e.g., formation control, sensor fusion, etc.

4 Conclusion and Future Work

In this paper we have presented a novel simulation environment, called SwarmSimX (SSX), tailored for the real-time simulation of multiple robots acting within a 3D physical environment. The software architecture is designed to encapsulate the main parts, namely RenderEngine, PhysicEngine, and Architect, so as to ensure independence from the underlying libraries used for simulation.

Exploiting the features of the `dlopen` library, new robots and sensors can be added to the simulation environment without the need of recompiling SSX itself. In addition Simulation extensions may be used to directly extend the features of the simulation, e.g., to provide an easy interface to other middleware (ROS) or other external software packages.

Aside from the description of the internal design, we also performed several tests aimed at validating the physical fidelity and real-time capabilities of SSX. In particular, we showed that the tracking performance of a virtual quadrotor following a predefined trajectory can be compared to the flight behavior of a real quadrotor following the same trajectory. These results show a very good performance in terms of representing the actual flight behavior of a single UAV, and in running the simulating environment in real-time. With an acceptable time-step of 0.02 s SSX can simulate dozens (at least 90) quadrotors with their associated sensors/controllers simultaneously.

Currently SwarmSimX is designed to simulate rigid-body dynamics only, but future works may focus on the inclusion of additional concepts provided by modern physics engines, e.g., joints, vehicle models, cloths, fluids. A redesign of the software architecture to allow an online exchange of the implementation of the main modules, e.g., RenderEngine, could increase even more the independence from specific implementations.

Acknowledgements. The authors like to thank Martin Riedel for his valuable work on the middleware software design used for the control programs.

This research was partly supported by WCU (World Class University) program funded by the Ministry of Education, Science and Technology through the National Research Foundation of Korea (R31-10008).

References

1. Pinciroli, C., Trianni, V., O'Grady, R., Pini, G., Brutschy, A., Brambilla, M., Mathews, N., Ferrante, E., Di Caro, G., Ducatelle, F., Stirling, T., Gutierrez, A., Gambardella, L.M., Dorigo, M.: ARGoS: A modular, multi-engine simulator for heterogeneous swarm robotics. In: 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5027–5034 (September 2011)
2. Craighead, J., Murphy, R., Burke, J., Goldiez, B.: A survey of commercial & open source unmanned vehicle simulators. In: Proc. IEEE Int. Robotics and Automation Conf., pp. 852–857 (April 2007)
3. Alex, A.L., Brunyé, T., Sidman, J., Weil, S.A.: From gaming to training: A review of studies on fidelity, immersion, presence, and buy-in and their effects on transfer in PC-based simulations and games (November 2005)
4. Boeing, A., Bräunl, T.: Evaluation of real-time physics simulation systems. In: Proceedings of the 5th International Conference on Computer Graphics and Interactive Techniques in Australia and Southeast Asia, GRAPHITE 2007, pp. 281–288. ACM, New York (2007)
5. Diankov, R.: Automated Construction of Robotic Manipulation Programs. PhD thesis, Carnegie Mellon University, Robotics Institute (August 2010)
6. Echeverria, G., Lassabe, N., Degroote, A., Lemaignan, S.: Modular openrobots simulation engine: Morse. In: Proceedings of the IEEE ICRA (2011)
7. Blender Foundation: Blender, `http://www.blender.org/` (accessed August 2012)
8. Freese, M., Singh, S., Ozaki, F., Matsuhira, N.: Virtual Robot Experimentation Platform V-REP: A Versatile 3D Robot Simulator. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 51–62. Springer, Heidelberg (2010)
9. Michel, O.: Cyberbotics Ltd. Webots TM: Professional Mobile Robot Simulation. International Journal of Advanced Robotic Systems 1, 39–42 (2004)
10. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: 2004 IEEE RSJ International Conference on Intelligent Robots and Systems, IROS IEEE Cat No04CH37566, vol. 3, pp. 2149–2154 (2004)
11. NVIDIA®: PhysX®, `http://www.geforce.com/hardware/technology/physx` (accessed May 2012)
12. Boost: boost C++ libraries, `http://www.boost.org/` (accessed May 2012)

13. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software, 1st edn. Addison-Wesley Professional (November 1994)
14. Biggs, G., Makarenko, A., Brooks, A., Kaupp, T., Moser, M.: Gearbox: Truly reusable robot software (poster). In: Proc. IEEE/RSJ Int. Conference on Intelligent Robots and Systems, Nice, France (September 2008)
15. ROS.org community: ROS Wiki, http://www.ros.org (accessed May 2012)
16. Franchi, A., Secchi, C., Son, H.I., Bülthoff, H.H., Robuffo Giordano, P.: Bilateral teleoperation of groups of mobile robots with time-varying topology. Accepted to IEEE Trans. on Robotics (2012)
17. Franchi, A., Secchi, C., Ryll, M., Bülthoff, H.H., Robuffo Giordano, P.: Bilateral shared control of multiple quadrotors: Balancing autonomy and human assistance with a group of UAVs. Conditionally Accepted to IEEE Robotics & Automation Magazine (2012)
18. Robuffo Giordano, P., Franchi, A., Secchi, C., Bülthoff, H.H.: Passivity-based decentralized connectivity maintenance in the bilateral teleoperation of multiple UAVs. In: 2011 Robotics: Science and Systems, Los Angeles, CA (June 2011)
19. Canonical Ltd.: Ubuntu, http://www.ubuntu.com (accessed October 2012)
20. Torus Knot Software Ltd.: Ogre3D, http://www.ogre3d.org/ (accessed May 2012)

Evaluating the Effectiveness of Mixed Reality Simulations for Developing UAV Systems

Ian Yen-Hung Chen[1], Bruce MacDonald[1], and Burkhard Wünsche[2]

[1] Dept. of Electrical and Computer Engineering,
University of Auckland, New Zealand
{i.chen,b.macdonald}@auckland.ac.nz
[2] Dept. of Computer Science,
University of Auckland, New Zealand
burkhard@cs.auckland.ac.nz

Abstract. The development cycle of an Unmanned Aerial Vehicle (UAV) system can be long and challenging. Mixed Reality (MR) simulations can reduce cost, duration and risk of the development process by enabling the replacement of expensive, dangerous, or not yet fully developed components with virtual counterparts. However, there has been little validation of such hybrid simulation methods in practical robot applications. This paper evaluates the use of MR simulations for prototyping a UAV system to be deployed for a dairy farming monitoring task. We show that by augmenting the robot's sensing with a virtual moving cow using an extensible Augmented Reality (AR) tracking technique, MR simulations could help to provide efficient testing and identify improvements to the UAV controller. User study findings reveal the importance of both virtual and MR simulations to robot development, with MR simulations helping developers transition to development in a more physical environment.

1 Introduction

Unmanned Aerial Vehicles (UAVs) are ideal for livestock and vegetation monitoring in agriculture. The high mobility of UAVs enables fast exploration of agriculture fields to collect information in remote areas that are otherwise difficult to access by farmers. The development of UAV systems is however difficult, especially during testing due to potentially dangerous operations, site availability, weather conditions, and considerable resource requirements.

Mixed Reality (MR) simulation [4] can provide a cost-effective solution to robot experimentation. The simulator is founded on the concept of MR [11] and enables developers to design various test scenarios for evaluating robot systems involving real and virtual components. There have been increasing interests in applying MR to robotics. The literature reveals that similar work exists in using hybrid or MR simulation techniques for developing robot systems in areas such as humanoid robotics [13,12], underwater robotics [7], and aerial robotics [8]. Existing work proposes that the use of these simulation techniques benefits development in terms of cost and safety, however, efforts are primarily limited to conceptual designs and software implementations with minimal evaluations.

This paper presents a case study evaluation of MR simulations for prototyping a vision-based UAV system to be deployed for a cow monitoring task in agriculture. There are novel challenges in creating MR simulations for this task. We assess our MR robot simulator's [4] capability to a) provide robot developers a safe and efficient way of creating test scenarios for this high-risk operation, and b) accurately augment the robot's visual sensing with virtual inputs under the erratic motion of the UAV platform. A user study is also conducted to examine the robot developers' use of MR simulations for prototyping UAVs, and an analysis of its results forms one of the main contributions of this paper. Similar user studies have been presented such as in [6], but the focus is on the visualisation aspects of MR for robot development. Our previous work [5] investigated the user's perception of the MR simulation technology and its visual interfaces in a robot operation task, which did not involve any implementations. In comparison, this paper describes a user study that identifies how MR simulations can help robot programmers in implementing a robot software component, the development stages and tasks which MR simulations are suitable for, and how they compare with virtual simulations.

The remainder of the paper is organised as follows. Section 2 gives background on the cow monitoring project. Section 3 describes the MR simulation created for testing the UAV prototype. Section 4 presents evaluation results. Section 5 describes the user study and its findings.

2 Cow Monitoring Project Background

One of the main hurdles of efficient dairy farming is the lack of support for monitoring and tracking cow status on the farm. The farmers are interested in information such as a cow's health status and temperature (for indicating signs of pregnancy or oestrus), which affect its productivity. However, a single farmer is typically responsible for looking after a herd size of several hundreds of cows, which makes the process of information management difficult. In response, a vision-based cow monitoring UAV system is proposed for this task. As a proof of concept, the development began with a micro aerial vehicle. The Hummingbird quadrotor from Ascending Technologies [1] is used. Hummingbird comes with an onboard controller that implements attitude stabilisation based on the onboard IMU. A Point Grey Firefly MV camera with 4mm lens is mounted directly beneath the centre of the quadrotor and oriented to look downwards for the cow monitoring task. 640x480 colour images are transferred over the firewire cable to the ground station where the computations are performed. The goal is to enable the UAV to autonomously detect a target cow on the ground using the onboard camera and hover over the target so that any data of interest can be collected by specialised instruments carried onboard the UAV. The quadrotor should try to stay hovering above the target and follow its movement for the data collection process. The development is at the prototyping stage, and testing is primarily carried out in an indoor environment. Figure 1 shows the robot in the experimentation space.

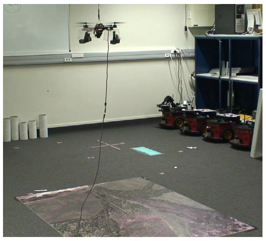


Fig. 1. Indoor UAV system test setup on a mock-up agricultural environment. The experimental setup is not to scale.

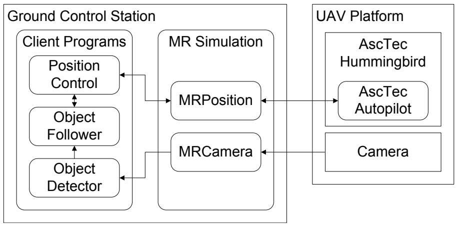


Fig. 2. MR simulation system diagram for testing the UAV prototype.

Three prototype components have been implemented: 1) The **Object Detector** processes image data and identifies target location in image coordinates (x_i, y_i) using a colour blob detection algorithm. 2) The **Object Follower** generates position commands in world coordinates (x_w, y_w, z_w, ψ_w) based on the target's image location to keep it in the view of the robot. The **Position Control** implements Proportional–Integral–Derivative (PID) controllers for robot's x, y, z, and *yaw* motion. Controller feedback is in the form of vision pose estimates, notably using the Augmented Reality (AR) tracking algorithm described in Section 3.1 to provide the pose estimates in a similar way to [2].

3 Mixed Reality Simulation

The project was in need of an efficient and cost-effective method of testing the prototype UAV system. Conducting field tests in this early stage is considered infeasible as it could harm the real animal and the surrounding agricultural objects in case of failures. MR simulation is able to help in testing the UAV system by simulating a virtual animated model of the target cow to be followed. It provides developers complete control over the virtual target and enables them to efficiently create repeatable tests of various scenarios, e.g. moving pattern and speed of the cow. To provide as much insight to real world tests as possible, the real UAV is used. The MR simulation enables the robot developers to observe the actual UAV flight behaviour as it responds to the simulated input in real time. This is believed to have a benefit over testing using pre-recorded videos.

3.1 Augmenting the Robot's Sensing

To facilitate interaction between the real robot and the virtual cow, the primary task is to augment the robot's vision input to reflect the presence of this virtual cow. This task is, however, difficult due to the erratic motion of the UAV platform. Registration of the virtual cow must be accurate since the behaviour of the robot is directly dependent on the resulting augmented image data produced by the MR simulation. Large errors in augmentations could lead to unexpected UAV movements and cause a serious crash.

To achieve this real-virtual interaction, we integrate an extensible AR tracking solution based on the Parallel Tracking and Mapping (PTAM) algorithm [9]. PTAM takes a structure-from-motion approach to track the camera while building a map of the environment in real time. Tracking is performed efficiently in a separate thread from mapping, relying on a large number of FAST corners and a motion model. The mapping thread incrementally builds the map from carefully selected image frames, known as keyframes, to ensure quality, and applies bundle adjustments to refine the pose estimates of the map elements and keyframes.

However, PTAM's manual map initialisation process creates the base map in an inconsistent scale and an unpredictable origin. We modify the original map initialisation process to associate the two initial keyframes (which triangulate the base map) with pose estimates from the markerless AR algorithm described in [3], allowing the map to be constructed in the real world metric space and coordinate system defined by the user. Pose estimates from the extensible AR system are used for registering the virtual cow onto the live video imagery. This generates augmented image data that will be delivered to the Object Detector component of the UAV system.

A system diagram of the components in this MR simulation is shown in Figure 2. Instead of reading images from the real camera, the Object Detector processes augmented images generated by MRCamera for cow detection.

3.2 MR Interfaces

MR interfaces are provided for monitoring the MR simulation, as shown in Figure 3. The AR interface provides users a view of the environment from the robot's onboard camera, reflecting the presence of the virtual cow. On the other hand, an Augmented Virtuality (AV) interface is designed to offer multiple views of the simulation from virtual perspectives.

4 Evaluation

The MR simulation was built on the MR robot simulator detailed in [4]. To assess its performance in simulating the prototype UAV system, an experiment was conducted that involved the robot following a moving virtual cow. The purpose is to investigate whether the augmentation in the robot's vision sensor data would be sufficiently robust and accurate to facilitate reliable robot responses. The virtual cow was animated to move in three different maneuver patterns within the space covered by the aerial map on the floor. Pose estimates of the robot given by the extensible AR tracker were recorded as it followed the cow.

The simulation results are shown in Figure 4. The robot followed the cow in each case, demonstrating successful interactions between the real robot and the virtual cow. The extensible AR tracker was able to cope with several sudden and erratic motions of the UAV and accurately augmented the virtual cow in the

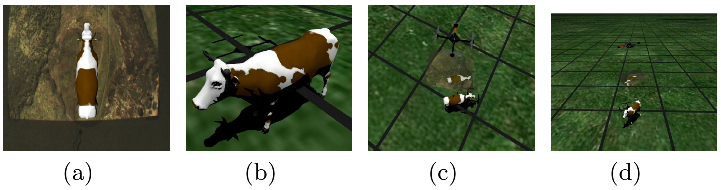


Fig. 3. a) The AR interface showing a view of the scene from the robot's perspective. b) The AV interface with free-look camera mode: in this example, the virtual camera is moved to focus on the target cow to be monitored. c) The AV interface with tethered camera mode: the virtual camera follows and tracks the movement of the UAV. d) The AV interface with fixed camera mode: similar to the tethered camera mode but with a fixed virtual camera.

view of the camera. The AR tracking errors were measured to be approximately 0.02m in x, 0.03m in y, and 0.06m in z. Figure 5 shows screenshots from the Object Detector window, which displays the processing being carried out on the augmented vision data generated by the MR robot simulator. The virtual cow remained aligned with the real world scene as it was animated to move across the aerial map. While no significant errors were observed in the pose estimates, it was noticed that the extensible AR tracker picked up fewer points in scenes with large portions of grasslands due to the lack of textured content. This is an expected behaviour of the underlying PTAM algorithm used. Nevertheless, there were sufficient textured patches surrounding this grassland region in this experiment to maintain the tracking quality. It must be noted that future outdoor experiments involving scenes with very few distinct features may cause the algorithm to produce poor results.

Overall, the use of the MR simulation helped to provide initial insights into the behaviour of the prototype UAV system for this cow monitoring task. The UAV trajectories in Figure 4 (especially the sine wave pattern) revealed to the robot developers the flight instability of this prototype UAV system due to the poor control logic of the object following algorithm, identifying the need for improvement.

5 User Study

Our user study examines the user's development approach and preferences in using MR simulations for implementing robot software components. It targeted a user group of experienced robot developers and computer programmers who were required to write software code for controlling the UAV and test their program in simulations. The participants were provided with three development methods, including MR simulations, and it was up to the participants how they used them. We acknowledge that the task given may take the participants through different development stages, and the experimental setup may need to vary accordingly to meet their needs. Instead of restricting the participants to a single method for the whole development process, it is more practical to let them experiment

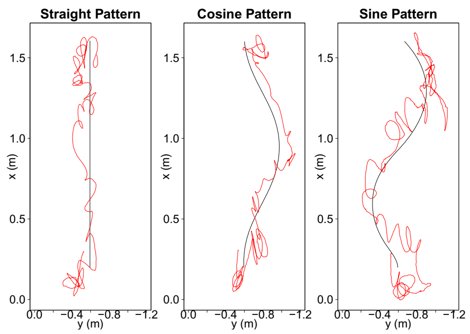


Fig. 4. UAV flight paths (red) from following three moving patterns of the virtual cow (black)

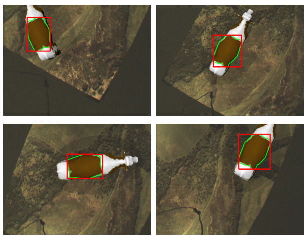


Fig. 5. Screenshots illustrating the image processing carried out by the Object Detector. Simple colour detection was implemented as a proof of concept for cow detection.

freely using the provided setups. The participants were video-taped for post-study analysis. Desktop videos were also taken to examine their actions in using the MR simulation tool and its interfaces.

5.1 Task

The participants occupied the role of a robot programmer. The primary task was to implement an algorithm for controlling the robot's movement and test their implementation in simulation. They were given the fully implemented Object Detector and Position Control components, and a skeleton code of the Object Follower component which they had to complete. The algorithm had to generate position commands for moving the robot in order to keep the detected target (given in image coordinates) in its view. Three development methods were provided, two of which are MR simulations.

1. **Method One (Virtual simulation):** Simulation is provided by Gazebo [10]. No complex aerodynamics is integrated in the simulation. A simple dynamics model is used to mimic the behaviour of the UAV under the control of the PID controller.
2. **Method Two (Free-moving Camera):** This setup involves manual movement of the onboard camera over the aerial map. A virtual cow is placed in the environment and augmented images are transferred to the UAV system. The AV interface reflects the UAV movement in real time.
3. **Method Three (UAV + Camera):** This setup involves the real UAV taking flight over the aerial map under the control of the Position Control component. A virtual cow is placed in the environment and augmented images are transferred to the UAV system. The AV interface reflects the UAV movement in real time.

The participants were free to switch between the development methods for testing their implementation as the development progressed. Sample test programs for moving the virtual cow were provided. They were asked to keep in mind the stage of development which they considered themselves at when using the simulations, assuming the development cycle could be divided into:

1. **Stage One:** Initial implementation and logic validation
2. **Stage Two:** Debugging and fixing errors in code
3. **Stage Three:** Evaluating performance of algorithm
4. **Stage Four:** Tuning and refining parameters

5.2 Procedure

The cow monitoring project was first explained to the participants, and the design of the UAV system was shown. Before starting the task, a quick demonstration of the three simulation methods was given. When a participant indicated task completion, the working of the system was verified in a test run that tested whether the UAV could correctly follow the virtual cow moving in a sine wave maneuver pattern. If the UAV failed to follow, the participant was given a chance to refine their code and conduct further testing. A second and final test run was held upon indication by the participant. A maximum of two and a half hours were given for the task.

5.3 Questionnaire

The questionnaire comprised four sections. Section one collected the participant's demographic information. Section two collected the participant's time distribution in using the provided development methods over the four stages of the development cycle. Section three measured the participant's experience in using the different development methods on a 7-point Likert scale. The last section collected the participant's preferences in using the development methods.

5.4 Hypotheses

1. The participants would transition to a more physical experimental setup as the development progresses. Method One and Method Three are believed to be essential, while Method Two benefits users who take a more cautious and systematic approach to testing.
2. Method One is useful for initial implementation (Stage One) and debugging errors (Stage Two). It may also be useful for experimenting with different object following strategies (Stage Three).
3. Method One is safer and easier to use compared to other methods. However, it provides the participants lower confidence in the actual behaviour of the UAV system in comparison to Method Three.

4. Method Two is useful for helping users understand the robot's field of view, and for validating the position commands (Stage One and Stage Two) by comparing them against the real world experimentation area before commencing test flights.
5. Method Two may also be used as a safer alternative for tuning and refining the control parameters of the algorithm (Stage Four).
6. Method Three is heavily used in the later development stage for evaluating performance (Stage Three) and tuning and refining parameters (Stage Four).
7. Method Three is effective and results in fewer mistakes. The risk of test flights is also higher and the method may require a longer learning curve.

5.5 User Study Results

10 participants were recruited (4 academic researchers, 5 postgraduate students, 1 software engineer), all of whom were experienced computer programmers (mean 8.95 years of experience, SD 6.82). 7 participants had experience in robot development (mean 3.93 years of experience, SD 2.05). None had experience in developing aerial robot systems. The 10 participants are coded P1, P2, P3, etc.

5.6 Participant's Development Approach

9 participants successfully completed the task within the given time. The remaining participant, P3, reported having struggled to understand the relationship between the different coordinate systems, a respect in which the development methods and the MR interfaces were unable to help the user.

More than half of the participants (6 participants) did not choose to use Method Two. Responses were because the participants believed a) the method was not useful for developing control algorithms (4 comments), and b) the same benefit of this method could be obtained using Method One and/or Method Three (2 comments). Interestingly, different opinions were collected from 2 participants who used Method Two. P1 and P8 commented that it was necessary to see the real environment to understand the robot's field of view when determining the starting control parameters. The non-responsiveness of the robot in this method was considered beneficial and they were able to debug their algorithms while the robot stayed in a particular position relative to the target. The findings supported hypothesis 4 to only some extent since only a minor proportion of the participants used this method. A useful comment was that the benefit of using Method Two could be leveraged if the simulation provided readily available graphical aids to help visualise the position commands in the MR interfaces.

Observations found that 9 participants began with Method One. P8 was the only exception and decided to use Method Two to collect all the parameters and implement the algorithm before testing it using Method One. All participants used Method Three as the last development method. The results suggest that the participants slowly transitioned from virtual simulation to MR simulation (with the real UAV) as the development progressed, supporting hypothesis 1.

Table 1. Time distribution in using the three development methods (Method One: M1, Method Two: M2, Method Three: M3)

Stage \ Method	M1	M2	M3	Subtotal
Stage One	**16.5%**	1.0%	3.5%	21.0%
Stage Two	**7.8%**	0.6%	6.1%	14.5%
Stage Three	4.5%	3.0%	**14.5%**	22.0%
Stage Four	6.5%	1.5%	**34.5%**	**42.5%**
Total				100%

Table 2. Users' recommended development methods for approaching each stage of the development cycle

Stage \ Method	M1	M2	M3
Stage One	**10**	1	0
Stage Two	**9**	2	2
Stage Three	6	2	**9**
Stage Four	2	2	**9**

The average percentage of time spent in each development method at each stage is shown in Table 1. Participants relied heavily on using Method Three for tuning and refining control parameters of the algorithm (34.5%). Observations also found that using virtual simulation is still important in the evaluation stage, with a majority of the participants (8 participants) constantly switching back to virtual simulation for evaluating their object following strategies. P8 pointed out that the time for using Method Three was precious since it required more time to set up and also consumed battery resources, and therefore, Method One was a cheaper and faster alternative for testing. Table 2 shows the user preferences if they were to develop a similar robot program in the future. Their recommendations conform to the time distribution findings. Method One was the most preferred method for initial implementation and debugging, while Method Three was more suitable for evaluating performance and tuning parameters. The results supported hypothesis 6 and the first part of hypothesis 2.

5.7 Monitoring UAV Operations

Reviewing the recorded desktop videos identified that all participants relied on the first-person AR view for evaluating the robot's performance, either by focusing on the AR interface or the Object Detector's window which displayed the same view. This enabled them to evaluate whether the robot was able to achieve the goal of the task, i.e., keeping the target object in the view, while ensuring the aerial map remained in the view for the Position Control component to function correctly. It was observed that participants who used the AV interface (4 participants) mainly chose the free-look camera and/or the tethered camera mode. However, 3 participants commented that although the interfaces helped to spot unusual UAV behaviour, reacting in time to prevent a crash was difficult and required experience. Again, this suggested the need for additional visualisations to help users foresee and take actions to prevent failures.

5.8 Statistical Analysis

The questionnaire results on the development method experience are shown in Figure 6. Method One and Method Three were rated by all the participants, and

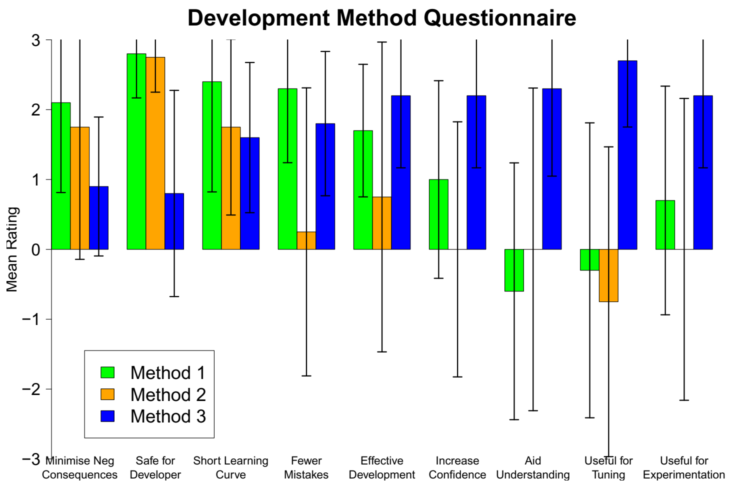


Fig. 6. Users' experiences in using the three development methods

Method Two was rated by the 4 participants who used this method. As Method Two had very different characteristics from the other two conditions, statistical tests were only performed to identify significant differences between Method One and Method Three. Method One and Method Three data were compared using Two-tailed Wilcoxon Signed Rank tests, while user ratings on Method Two provided an indication of how the method fits the related hypotheses.

As expected (hypothesis 3), using a real UAV had a greater impact on safety. The participants rated Method One significantly higher than Method Three for keeping negative consequences of failures minimal (Z = -2.20, p < 0.05), and being safe for the developer (Z = -2.87, p < 0.05). Nevertheless, the average ratings for Method Three in terms of safety are overall positive.

The results did not indicate that Method One had a significantly shorter learning curve than Method Three (Z = -1.54, p = 0.12), although it should be noted that Method Three required more preparation steps, such as initialising AR and starting an extra Position Control process, which took more effort for the users to set up the simulation.

The results did not indicate that Method Three was significantly more effective than Method One (Z = -1.00, p = 0.32). Similarly, there was no significant difference between Method Three and Method One for giving users greater confidence (Z = -1.56, p = 0.12), or making fewer mistakes in the development process (Z = -0.91, p = 0.36). Consequently, the results did not support hypothesis 7. However, observations and post-study video analysis identified that an unexpected hardware influence on the UAV behaviour in Method Three may have led some participants to create mistakes during development. Upon transitioning to Method Three, the majority of the participants found their algorithm did not produce reliable robot performance and began to make incorrect changes to their code before performing additional testing. The true cause for this UAV behaviour was due to the effect the camera cable had on the x-movement of the UAV, which was not simulated in Method One. Unfortunately, MR simulations and the interfaces were not designed to help them isolate this problem.

The participants rated Method Three significantly higher than Method One for understanding the UAV flight behaviour as expected (Z = -2.57, $p < 0.05$), since it involved the real UAV platform in simulation. The finding could change if a high-fidelity flight dynamics model was integrated into the virtual simulation. This needs to be investigated in the future.

Method Three was rated significantly higher than Method One for tuning control parameters (Z = -2.81, $p < 0.05$). The findings agreed with the development method time distribution data analysis and the user recommendations, suggesting Method Three was more suitable for the later stages of the development; this further supported hypothesis 6.

Statistical analysis did not suggest that Method Three was more useful than Method One for experimenting with different flight maneuvers (Z = -1.80, p = 0.07), though the result was approaching significance. While both average ratings were positive, there was high variance in Method One ratings, thus the results could not fully support the second part of hypothesis 2. Method Two was found to be unsuitable for tuning, which rejected hypothesis 5. P1 and P8 commented later in the interview that Method Two was useful for *identifying* (not tuning) initial control parameters, which were later tuned using Method Three.

5.9 Discussions

The results on the overall participant's development approach may not seem surprising but there were specific findings which were not expected. In this study, statistical analysis did not indicate that the use of MR simulation (with real UAV) was significantly more effective than virtual simulation or was it more useful for experimentation. The findings were however not unfavourable because MR simulation does not intend to replace existing simulation methods but to complement them during the robot development process; a hypothesis that was supported by the results. Further research is necessary to validate the strengths of MR simulations in a more focused, long term study that compares the use of MR simulation alone against using only virtual simulation.

6 Conclusions

This paper contributes a case study evaluation on using MR simulations for the development of a prototype UAV system. The simulation augmented the robot's onboard vision data with virtual inputs using a modified PTAM algorithm for testing the cow monitoring operation. The use of the MR simulation has been shown to help provide valuable insights into the UAV system performance that are otherwise difficult to obtain in real world tests. User study results indicated that MR simulations complemented virtual simulations, providing an intermediate experimental environment before moving onto testing in the real world. It helped robot programmers understand UAV flight behaviours and tune control parameters for better robot performance. Future work includes providing more visualisation aids to improve monitoring of UAV tasks, and conducting a user study to identify cases where the benefits of using MR simulations are more evident compared to traditional simulation approaches.

References

1. Ascending Technologies, `http://www.asctec.de/`
2. Blösch, M., Weiss, S., Scaramuzza, D., Siegwart, R.: Vision based MAV navigation in unknown and unstructured environments. In: Proceedings of the IEEE International Conference on Robotics and Automation, pp. 21–28 (2010)
3. Chen, I.Y.H., MacDonald, B., Wünsche, B.: Markerless augmented reality for robots in unprepared environments. In: Proceedings of the Australasian Conference on Robotics and Automation, Canberra, Australia, December 3-5 (2008)
4. Chen, I.Y.H., MacDonald, B., Wünsche, B.: Mixed reality simulation for mobile robots. In: Proceedings of the IEEE International Conference on Robotics and Automation, Kobe, Japan, May 12-17, pp. 232–237 (2009)
5. Chen, I.Y.H., MacDonald, B., Wünsche, B., Biggs, G., Kotoku, T.: Analysing Mixed Reality Simulation for Industrial Applications: A Case Study in the Development of a Robotic Screw Remover System. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 350–361. Springer, Heidelberg
6. Collett, T., MacDonald, B.: An augmented reality debugging system for mobile robot software engineers. Journal of Software Engineering for Robotics 1(1), 18–32 (2009)
7. Davis, B., Patron, P., Lane, D.: An augmented reality architecture for the creation of hardware-in-the-loop & hybrid simulation test scenarios for unmanned underwater vehicles. In: OCEANS, pp. 1–6 (2007)
8. Göktoğan, A., Sukkarieh, S.: An Augmented Reality System for Multi-UAV Missions. In: Proceedings of the Simulation Conference and Exhibition, SimTect, May 9-12. Citeseer, Sydney (2005)
9. Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: Proceedings of the Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality, Nara, Japan, pp. 225–234 (November 2007)
10. Koenig, N., Howard, A.: Design and use paradigms for Gazebo, an open-source multi-robot simulator. In: Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems, September 28-October 2, vol. 3, pp. 2149–2154 (2004)
11. Milgram, P., Colquhoun, H.: A taxonomy of real and virtual world display integration. In: Mixed Reality-Merging Real and Virtual Worlds, pp. 5–28 (1999)
12. Nishiwaki, K., Kobayashi, K., Uchiyama, S., Yamamoto, H., Kagami, S.: Mixed reality environment for autonomous robot development. In: Proceedings of the IEEE International Conference on Robotics and Automation, Pasadena, CA, USA, pp. 2211–2212 (May 2008)
13. Stilman, M., Michel, P., Chestnutt, J., Nishiwaki, K., Kagami, S., Kuffner, J.: Augmented reality for robot development and experimentation. Tech. Rep. CMU-RI-TR-05-55, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA (November 2005)

Comprehensive Simulation of Quadrotor UAVs Using ROS and Gazebo

Johannes Meyer[1], Alexander Sendobry[1], Stefan Kohlbrecher[2], Uwe Klingauf[1], and Oskar von Stryk[2]

[1] Department of Mechanical Engineering, TU Darmstadt, Germany
[2] Department of Computer Science, TU Darmstadt, Germany

Abstract. Quadrotor UAVs have successfully been used both in research and for commercial applications in recent years and there has been significant progress in the design of robust control software and hardware. Nevertheless, testing of prototype UAV systems still means risk of damage due to failures. Motivated by this, a system for the comprehensive simulation of quadrotor UAVs is presented in this paper. Unlike existing solutions, the presented system is integrated with ROS and the Gazebo simulator. This comprehensive approach allows simultaneous simulation of diverse aspects such as flight dynamics, onboard sensors like IMUs, external imaging sensors and complex environments. The dynamics model of the quadrotor has been parameterized using wind tunnel tests and validated by a comparison of simulated and real flight data. The applicability for simulation of complex UAV systems is demonstrated using LIDAR-based and visual SLAM approaches available as open source software.

1 Introduction

Quadrotor UAVs have successfully been used both in research and for commercial applications in recent years. Impressive results have been shown using quadrotor aircraft of various sizes and in different scenarios. The inherently instable nature of quadrotor flight can lead to loss or damage of UAVs easily, especially when evaluating prototype soft- or hardware. The lack of a simulation environment for quadrotor UAVs that covers realistic flight dynamics, camera and range sensors and an easy integration with existing robotic middleware solutions motivated this work. We present a comprehensive framework to simulate our quadrotor, that has been developed during the last few years. It is based on the Gazebo open source simulator and the Robot Operating System (ROS), that has become a de facto standard in robotics research and facilitates integration of contributions by other researchers. Common sensors for autonomous robots like LIDAR devices, RGB-D and stereo cameras are already available for Gazebo and can be attached to the robot, while plugins for other, more UAV-specific sensors like barometers, GPS receivers and sonar rangers have been added as part of this work.

The remainder of this paper is organized as follows: After the discussion of related work in section 2, section 3 presents the simulation model considering geometry, flight dynamics and control and how the model is implemented in Gazebo. Comparative results from flight tests and simulation runs as well as a demonstration of applicability for evaluating high-level algorithms are presented in section 4.

2 Related Work

As we aim at a comprehensive approach for simulation of quadrotor UAV systems, we provide an overview both of simulation/ground truth tracking approaches as well as quadrotor control approaches. Most approaches for quadrotor simulation focus on vehicle dynamics for controller design, often using specialized tools like Matlab/Simulink [20]. Sometimes other tools like the Flightgear open source simulation framework are used for visualization [18,9]. Using such approaches, testing of sensor-based high level control and behaviors is not possible or requires significant additional implementation effort. Quadrotor UAVs can be simulated using USARsim [7], but a recently published ROS integration [2] is of limited scope. In [1], the use of a simulator also providing sensor data is mentioned, but not made available for testing.

Several authors have proposed dynamics models for the simulation of quadrotor aircraft which are based on the same flight mechanical principles [10,3,9,11,20]. While dealing with different aspects in detail, none of them considers motor and propeller dynamics, aerodynamics, external disturbances (e.g. wind), and noisy sensor signals and state estimation in an integrated fashion.

Recently, external optical tracking for the acquisition of ground truth data has been used with great success [8,17]. The installation of such systems however is costly and often not feasible due to space constraints. Even if such a system is available, testing of multi-UAV control approaches in simulation is advantageous, as potential collisions or other faults incur no cost in simulation.

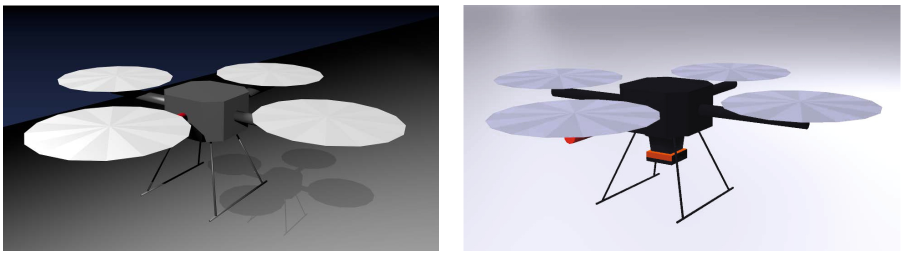

(a) (b)

Fig. 1. Mesh-based quadrotor model: (a): Model shown rendered in Blender. (b) Model used in Gazebo. A Hokuyo UTM-30LX laser scanner is mounted below the main body.

3 Model Description and Simulation

As we aim at comprehensive simulation of all relevant components including low level sensing, system dynamics and control, we provide an overview of these parts independently. Gazebo provides a multi-robot simulation environment including dynamics simulation, which is provided by the ODE or bullet physics engines. While the simulator considers gravity, contact forces and friction by its own, it does not cover aerodynamics and propulsion systems that are especially required for aerial vehicles. A plugin systems enables the user to add custom controllers for simulated robots and sensors or to influence the environment.

3.1 Geometry

The robot geometry has been modeled using the open source software Blender. To be able to provide different colors (both texture or material based) for the model, the visual geometry is provided using the COLLADA format, while the collision geometry is modeled as a .stl mesh. The model is designed to have a low polygon count and still retain the relevant aspects of quadrotor geometry. As a trade-off between visual fidelity, collision detection and dynamics modeling needs, the propellers are modeled as discs.

3.2 Dynamics Model

One of the main advantages of the quadrotor concept is the simplicity of its propulsion and steering system, consisting only of four independent motors and propellers with fixed pitch, where each pair of opposite propellers rotates in one direction to avoid yaw torque during roll and pitch movements. As a result, the overall system dynamics are mainly determined by the thrust and torque induced by the individual motor/propeller units.

Flight Dynamics. The movement of a rigid body can be described by the sum of all forces $\boldsymbol{F}$ and torques $\boldsymbol{M}$ acting on the vehicle:

$$\dot{\boldsymbol{p}}^{\mathrm{n}} = \boldsymbol{v}^{\mathrm{n}} \tag{1a}$$

$$\dot{\boldsymbol{v}}^{\mathrm{n}} = m^{-1}\mathbf{C}_{\mathrm{b}}^{\mathrm{n}}\boldsymbol{F} \tag{1b}$$

$$\dot{\boldsymbol{\omega}}^{\mathrm{b}} = \boldsymbol{J}^{-1}\boldsymbol{M} \tag{1c}$$

Here, $\boldsymbol{p}^{\mathrm{n}}$ and $\boldsymbol{v}^{\mathrm{n}}$ are the position and velocity of the body's center of gravity in the (inertial) navigation coordinate system, $\boldsymbol{\omega}^{\mathrm{b}}$ is its angular rate given in body coordinates and $\mathbf{C}_{\mathrm{b}}^{\mathrm{n}}$ is the rotation matrix that transforms a vector from body (index b) to navigation coordinates (index n).

The mass m and inertia $\boldsymbol{J}$ of the quadrotor need to be known and have been estimated by weighing the individual components and using the geometric model. The force vector $\boldsymbol{F}$ comprises motor thrust $\boldsymbol{F}_{\mathrm{M}}$, drag forces $\boldsymbol{F}_{\mathrm{d}}$ and the gravity vector $\boldsymbol{F}_{\mathrm{g}}$. The torque vector $\boldsymbol{M}$ is divided into propulsion torque $\boldsymbol{M}_{\mathrm{M}}$ and drag moments $\boldsymbol{M}_{\mathrm{d}}$. Drag forces and moments are given by:

$$\boldsymbol{F}_{\mathrm{d}} = -\boldsymbol{C}_{\mathrm{d,F}} \cdot \mathbf{C}_{\mathrm{n}}^{\mathrm{b}} \cdot \left|\boldsymbol{v}^{\mathrm{n}} - \boldsymbol{v}_{\mathrm{w}}^{\mathrm{n}}\right| (\boldsymbol{v} - \boldsymbol{v}_{\mathrm{w}}) \tag{2a}$$

$$\boldsymbol{M}_{\mathrm{d}} = -\boldsymbol{C}_{\mathrm{d,M}} \cdot \left|\boldsymbol{\omega}^{\mathrm{b}}\right| \boldsymbol{\omega}^{\mathrm{b}} \tag{2b}$$

with the diagonal drag coefficient matrices $\boldsymbol{C}_{\mathrm{d,F}}$ and $\boldsymbol{C}_{\mathrm{d,M}}$ and the wind vector $\boldsymbol{v}_{\mathrm{w}}^{\mathrm{n}}$. Finally, the gravity force is given by

$$\boldsymbol{F}_{\mathrm{g}} = m \cdot \mathbf{C}_{\mathrm{n}}^{\mathrm{b}} \cdot \left[0\ 0\ g_e\right]^{\mathrm{T}} \ . \tag{3}$$

With these forces and torques resulting from self motion of any system in space and the propulsion forces and torques described in the following section, the vehicle movement can be obtained by solving equations (1).

Motor Dynamics. The propulsion system of our quadrotor UAV consists of four brushless DC motors. The dynamic behavior of a brushless DC motor has been derived from [12] with some simplifications. Assuming a very low inductance of the motor coils, the current rise time can be neglected. The motor dynamic behavior therefore simplifies to a PT_1 element and is described by Eqs. (4) - (6). In steady state the induced anchor voltage U_{A} depends on the rotation speed $\boldsymbol{\omega}_{\mathrm{M}}$ and the anchor current I_{A}:

$$U_{\mathrm{A}} = R_{\mathrm{A}} I_{\mathrm{A}} + \Psi \boldsymbol{\omega}_{\mathrm{M}} \tag{4}$$

The electromagnetic torque M_{e} for each motor is given by

$$M_{\mathrm{e}} = \Psi I_{\mathrm{A}} \tag{5}$$

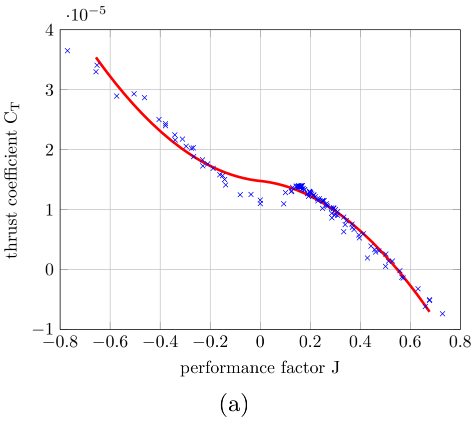
With the mechanical torque M_{m} and the motor inertia J_{M} the change in rotation speed can be calculated through:

$$\dot{\omega}_{\mathrm{M}} = \frac{1}{J_{\mathrm{M}}} \cdot (M_{\mathrm{e}} - M_{\mathrm{m}}) = \frac{1}{J_{\mathrm{M}}} \cdot \left(\frac{\Psi}{R_{\mathrm{A}}} \cdot (U_{\mathrm{A}} - \Psi \boldsymbol{\omega}_{\mathrm{M}}) - M_{\mathrm{m}} \right) \tag{6}$$

The nonlinear term M_{m} describes the torque resulting from bearing friction as well as load friction (i.e. drag) of the airscrew. It can be written as $M_{\mathrm{m}} = k_{\mathrm{T}} \cdot T$ where T is the thrust of a single airscrew [15] which is a a broad simplification of a former approach [21] without loss of accuracy.

Thrust Calculation. In contrast to the former approach we now use a nonlinear quadratic approximation for thrust calculation, similar to [9]. This approach has been selected based on wind tunnel tests (cf. Fig. 2(a)) and is sufficiently accurate as to not require the use of more complex thrust models [10]. With the dynamic expression of the motor's rotational speed $\boldsymbol{\omega}_{\mathrm{M}}$ from equation (6) it is straightforward to calculate the thrust force T for a single motor-airscrew combination:

$$T = C_{\mathrm{T},0} \boldsymbol{\omega}_{\mathrm{M}}^2 + C_{\mathrm{T},1} v_1 \boldsymbol{\omega}_{\mathrm{M}} \pm C_{\mathrm{T},2} v_1^2 \tag{7}$$



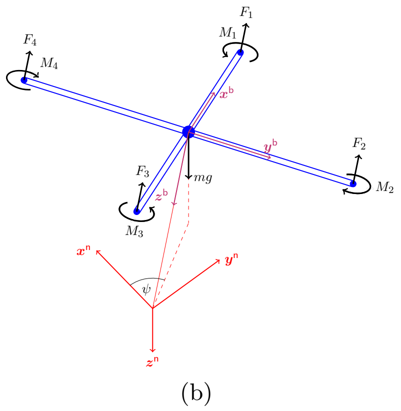


Fig. 2. (a) Thrust coefficient C_{T} of an airscrew as a function of its performance factor J. Crosses mark wind tunnel measurements, while the solid line represents the approximation. (b) Sketch of the quadrotor to show the different coordinate systems and naming conventions.

Dividing the above equation by $\boldsymbol{\omega}_{\mathrm{M}}^2$ and using the performance factor $J = v_1/\boldsymbol{\omega}_{\mathrm{M}}$ the thrust coefficient $C_{\mathrm{T}}(J)$ is given by:

$$C_{\mathrm{T}}(J) = C_{\mathrm{T},0} + C_{\mathrm{T},1}J \pm C_{\mathrm{T},2}J^2 \tag{8}$$

where the parameters $C_{\mathrm{T},i}$ have been identified in wind tunnel test. A negative v_1 (meaning a falling quadrotor) results in a positive prefix of $C_{\mathrm{T},2}$. In Fig. 2(a) the polynomial approximation of $C_{\mathrm{T}}(J)$ is shown. For a quadrotor helicopter the free stream velocity v_1 in general is different for each of the rotors. It can be calculated through geometric inspection of the vehicle shown in Fig. 2(b):

$$(v_1)_i = -\left[0 \;\; 0 \;\; 1\right] \cdot \left(\boldsymbol{v}^{\mathrm{b}} + \left(\boldsymbol{\omega}^{\mathrm{b}} \times \boldsymbol{e}_i\right) \cdot l_{\mathrm{M}}\right) \tag{9}$$

with the unit vectors

$$\boldsymbol{e}_1 = \left[1\;0\;0\right]^{\mathrm{T}}, \qquad \boldsymbol{e}_2 = \left[0\;1\;0\right]^{\mathrm{T}}, \qquad \boldsymbol{e}_3 = -\left[1\;0\;0\right]^{\mathrm{T}}, \quad \boldsymbol{e}_4 = -\left[0\;1\;0\right]^{\mathrm{T}}$$

for the four different motors. l_{M} is the distance between the geometric centers of motors and quadrotor. With the coordinate system conventions shown in Fig. 2(b) the following expression for the overall wrench of the quadrotor can be determined:

$$\boldsymbol{F}_{\mathrm{M}}^{\mathrm{b}} = \begin{bmatrix} 0 \\ 0 \\ -\Sigma_{i=1}^{4} F_i \end{bmatrix} \qquad \boldsymbol{M}_{\mathrm{M}}^{\mathrm{b}} = \begin{bmatrix} (F_4 - F_2) \cdot l_{\mathrm{M}} \\ (F_1 - F_3) \cdot l_{\mathrm{M}} \\ -M_1 + M_2 - M_3 + M_4 \end{bmatrix} \tag{10}$$

The four single forces F_i are calculated by solving equation (7) while the moments M_i are obtained through combining equation (4) and (5). The incorporation of blade flapping effects which can be used to aid state estimation [4] is subject of future work.

We implemented two plugins that calculate propulsion and drag forces acting on the aircraft given the internal state of the vehicle, the four motor voltages and the wind vector. The current wind can be specified as constant vector or provided by an external model or from real log data. Gazebo then applies the calculated forces and torques to the quadrotor body for each simulation step.

3.3 Sensor Simulation

As attitude, position and velocity cannot be measured directly, accurate models are needed to simulate the signals from various sensors needed for estimating the state of the UAV. These sensors have been implemented as independent Gazebo plugins and can be attached to the model by including them in the robot URDF description. The plugins accept parameters covering the error characteristics and the WGS84 position, altitude and orientation of the Gazebo reference frame in the world coordinate system wherever necessary.

Error Model. All sensors share a common first order Gauss Markov error model [5], permitting simulation of sensors with different error characteristics. Each simulated measurement $y(t)$ at time t is given by

$$y = \hat{y} + b + w_y \tag{11a}$$

$$\dot{b} = -\frac{1}{\tau} b + w_b \tag{11b}$$

where $\hat{y}$ is the true value or vector, b is the current bias and w_y and w_b are independent, zero-mean white Gaussian noise variables. w_y is additive noise acting directly on the measurement and w_b describes the characteristics of the random drift with time constant τ.

Inertial Measurement Unit. The inertial measurement unit (IMU) is the most important sensor for the stabilization of quadrotor flight as it measures the angular velocities and accelerations of the vehicle body in the inertial frame. Integration of these values provides a good reference of attitude and speed over short time intervals with fast response times, but is not suitable for long-term reference due to the significant drift of available low-cost sensors. Also note that an observer onboard the vehicle cannot distinguish gravity from other external forces and therefore the acceleration of the body in the world frame cannot be measured directly without knowing the orientation of the body.

Barometric Sensor. For simulating the static pressure at the present altitude, we use the International Standard Atmosphere (ISA) model as defined by the International Civil Aviation Organization (ICAO), which describes the pressure, temperature and density of the earth's atmosphere under average conditions at mid latitudes. The elevation of the simulation reference frame above mean sea level and the simulated pressure (only required for the output of pressure values in hPa) at sea level can be specified as parameters.

Ultrasonic Sensor. For controlling the height during the takeoff and landing phases and for switching on and off the motors, the range estimate from an downward pointing ultrasonic sensor is used. This device transmits short ultrasound impulses and returns the distance corresponding to the first echo returned from the ground or an object within it's field of view. Available ultrasound sensors have a maximum range of about 3 to 6 meters. The simulated ultrasonic sensor uses the Gazebo ray sensor interface to determine ray-casting based distances to world geometry. The distance value returned is the minimum of all rays (9 by default).

Magnetic Field Sensor. The earth magnetic field serves as a reference for the heading or yaw angle of the quadrotor. As using a single axis compass would lead to significant errors with increasing roll and pitch angles, three-axis magnetometers are commonly used for UAVs. With the assumption that the earth-fixed magnetic field vector is constant within the area of operation, it is straightforward to calculate the body-fixed vector given the declination, inclination and field magnitude. Deviation errors through interference from parts of the robot itself are covered by the generic error model.

GPS Receiver. Pseudo range measurements and the resulting position and velocity solution are influenced by different factors like the satellite ephemeris errors, atmospheric errors or receiver errors [19]. These error sources are approximated using the Gauss-Markov error model, with the parameters of our uBlox receiver module having been determined experimentally. To reproduce the interdependency of position and velocity errors we use the noise-affected velocity measurement error ($\boldsymbol{v}_{\text{GPS}} - \hat{\boldsymbol{v}}_{\text{GPS}}$) instead of $\boldsymbol{w_b}$ in Eq. (11b) for the integration of the position error. A more detailed consideration of GPS errors and especially multipath effects in the vicinity of buildings is left for future work. To calculate WGS84 coordinates from the simulated position and velocity in Cartesian coordinates we use a simple equirectangular projection that is based on a flat world assumption. This projection is accurate enough in the vicinity of the chosen reference point and outside the polar regions.

3.4 State Estimation and Control

Although state estimation and control are not specific to simulation, both components are required to close the loop between simulated sensor signals and the resulting motor voltages required to stabilize and control the quadrotor.

For estimating the state of the system we use an Extended Kalman Filter (EKF) to fuse all available measurements to a single navigation solution containing the orientation, position and velocity of the vehicle as well as observable error states like the IMU bias errors. This approach is usually referred to as integrated navigation.

Our controller is implemented as a set of cascaded PID controllers, with the inner loop controlling the attitude, yaw rate and vertical velocity and an outer loop controlling the horizontal velocity, heading and altitude (Fig. 3). This

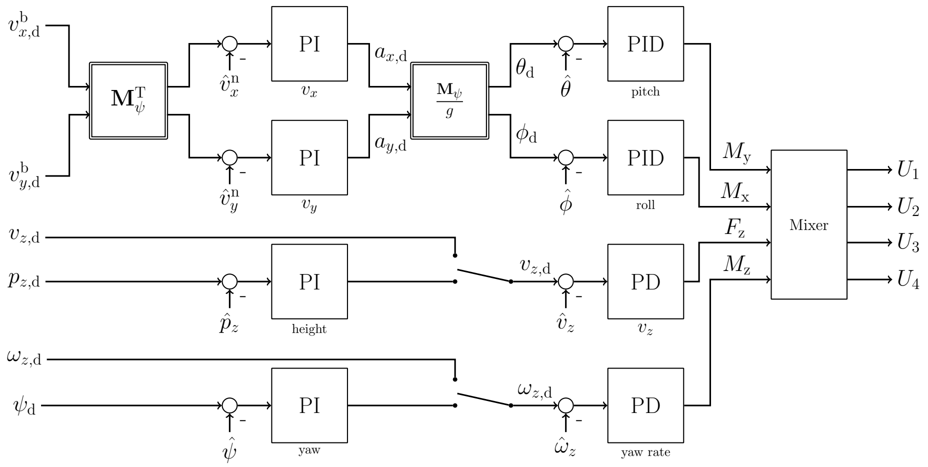


Fig. 3. The controller is realized through separate cascaded PID controllers controlling roll and pitch movement, yaw rate and vertical velocity

approach assumes that each axis and the altitude can be controlled independently, which is valid for moderate deviations from the hovering state. The output of the inner loop are commanded torques and vertical thrust, which are translated to motor voltages either by using a static mixture matrix or by feeding them into an inverted model of the propulsion system presented in section subsection 3.2.

For simulation we use exactly the same implementation as on the real quadrotor. It is based on the Open Robot Control Software (Orocos) toolchain [6], which provides interfaces to ROS and executes tasks satisfying hard realtime constraints on the onboard PC system. This software-in-the-loop approach offers great flexibility for testing advanced control algorithms before the deployment on the real vehicle and therefore minimizes the risk of damage or loss dramatically. Implementation details can be found in previous publications [16].

4 Experiments

Different aspects of simulation are validated using experiments in this section. We also show examples of comprehensive simulation scenarios using the flight dynamics model as well as leveraging existing ROS open source software.

4.1 Validation of Dynamics Model

To validate the dynamics model, we let both the real and simulated UAV perform a test trajectory consisting of transitions between different velocities. All measurable variables of the real quadrotor show the same characteristics as the corresponding simulated counterparts. The power spectrum densities (PSD) of

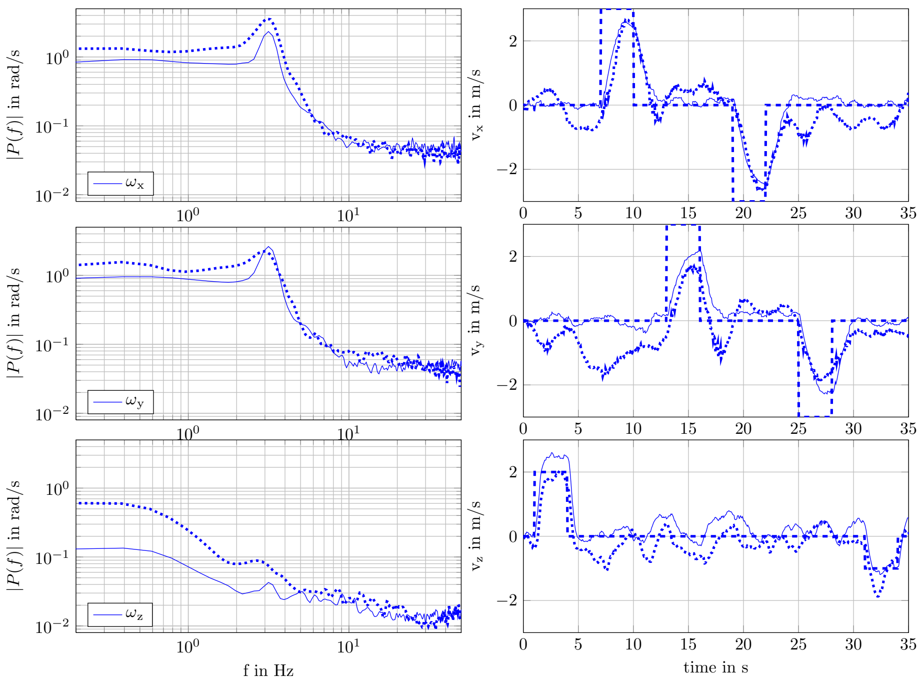


Fig. 4. Diagrams of simulated and measured angular and translational velocity. Dotted lines represent measurements while solid lines are simulated data. The left side shows the PSD of the angular rates. On the right side the estimated velocity both in simulation and reality with the commanded speeds (dashed line) is shown.

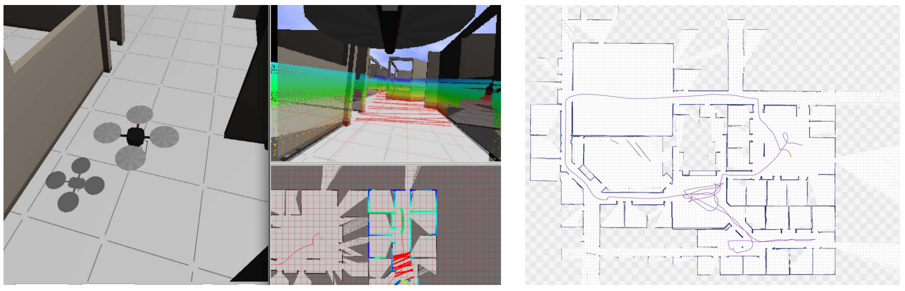
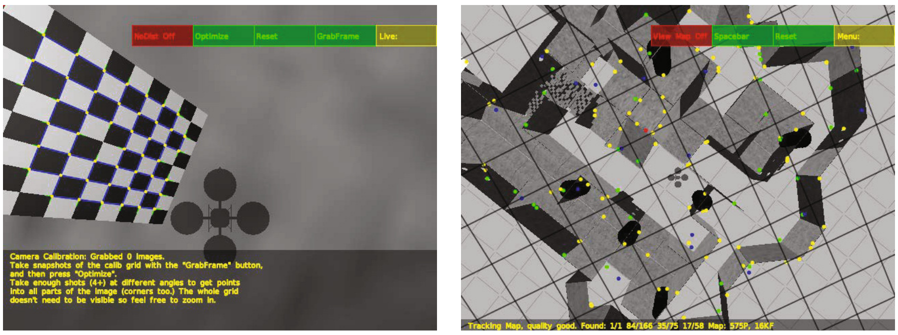


Fig. 5. Indoor SLAM simulation: (a): Screenshot of the GUI. On the left the Gazebo simulation environment is visible. On the top right the view of the forward facing camera is shown, with LIDAR point cloud and map data projected into the image. A top down ortho view is visible on the bottom right (b) Final map generated after teleoperation of the UAV through the scenario.



Camera Calibration: Grabbed 0 Images.
Take snapshots of the calib grid with the "GrabFrame" button,
and then press "Optimize".
Take enough shots (4+) at different angles to get points
into all parts of the image (corners too.) The whole grid
doesn't need to be visible so feel free to zoom in.

Tracking Map, quality good. Found: 1/1 84/166 35/75 17/58 Map: 575P, 16KF

(a) (b)

Fig. 6. Visual SLAM simulation: (a): Calibration of camera system in simulation. (b) Screenshot of PTAM being used for visual SLAM on a quadrotor hovering above a simulated NIST standard arena for response robots.

the angular rates and and the velocities are shown in Fig. 4. The controller and a dead time of about 15 ms cause the quadrotor to oscillate slightly with a frequency of about 3 Hz which is easily visible in the frequency domain. Differences in velocity between simulation and reality are mainly due to a gusty wind of about 5 m/s which was apparent during the outdoor tests. In simulation, we therefore defined a constant wind of 5 m/s.

4.2 Example Scenarios

In this section, different example scenarios are shown, demonstrating the comprehensive nature of quadrotor simulation and the interfacing with other open source ROS software. Instructions for reproducing all presented scenarios are provided on the *hector_quadrotor*[1] website on ros.org.

In- and Outdoor Flight Scenarios. We flew the simulated quadrotor through two example indoor and outdoor worlds to evaluate the quality of high-level sensor data. Using the estimated state or ground truth data, the quadrotor pose can be visualized along with sensor data.

To demonstrate the applicability for indoor SLAM simulation, we deploy a previously developed SLAM approach [14] on the quadrotor UAV. The Willow Garage office environment is part of the Gazebo ROS package, demonstrating the applicability and interoperability of the quadrotor simulation with existing Gazebo environments. The quadrotor UAV is teleoperated using a gamepad for this demonstration. As shown in Fig. 5(a), sophisticated visualization including projection of visualization data into camera images is possible by leveraging available ROS tools like *rviz*. The final map learned is shown in Fig. 5(b) and of comparable quality to those learned in real world scenarios.

A video of outdoor flying is available online[2].

[1] `http://www.ros.org/wiki/hector_quadrotor`

[2] `http://www.youtube.com/watch?v=9CGIcc0jeuI`

Visual SLAM. To demonstrate simulated image based state estimation, we deploy a modified version [22] of the original PTAM system [13] for visual SLAM. As demonstrated in Fig. 6(a), checkerboard-based calibration of camera parameters can also be performed in simulation. Fig. 6(b) shows a screenshot of the PTAM GUI while the simulated quadrotor UAV hovers above an example scenario, successfully tracking features in the image and estimating the aircraft pose. It should be noted that the default camera simulation in Gazebo is of limited fidelity as it does not exhibit effects like motion blur.

5 Conclusion

We presented a framework for the simulation of quadrotor UAV systems employing ROS and the Gazebo simulator. The tight integration with existing (and future) ROS tools permits the comprehensive simulation of quadrotor UAVs including low level sensing, flight dynamics and external sensing using any sensor available for Gazebo simulation. The level of detail can be adapted depending on the application, e.g. by using ground truth data for control or bypassing the propulsion model.

Acknowledgments. This work has been funded by the Research Training Group 1362 “Cooperative, Adaptive and Responsive Monitoring in Mixed-Mode Environments” of the German Research Foundation (DFG).

References

1. Achtelik, M., Bachrach, A., He, R., Prentice, S., Roy, N.: Autonomous navigation and exploration of a quadrotor helicopter in GPS-denied indoor environments. In: Robotics: Science and Systems Conference (2008)
2. Balakirsky, S.B., Kootbally, Z.: USARSim/ROS: A Combined Framework for Robotic Control and Simulation. In: ASME 2012 International Symposium on Flexible Automation (ISFA 2012). ASME (2012)
3. Bouabdallah, S., Siegwart, R.: Full control of a quadrotor. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 153–158 (November 2007)
4. Bristeau, P., Callou, F., Vissière, D., Petit, N., et al.: The navigation and control technology inside the AR Drone micro UAV. In: 18th IFAC World Congress, Milano, Italy, pp. 1477–1484 (2011)
5. Brown, R., Hwang, P., et al.: Introduction to random signals and applied Kalman filtering. Wiley, New York (1992)
6. Bruyninckx, H.: Open robot control software: the OROCOS project. In: IEEE International Conference on Robotics and Automation (ICRA), vol. 3, pp. 2523–2528. IEEE (2001)
7. Carpin, S., Lewis, M., Wang, J., Balakirsky, S., Scrapper, C.: USARSim: a robot simulator for research and education. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 1400–1405 (2007)

8. Ducard, G., D'Andrea, R.: Autonomous quadrotor flight using a vision system and accommodating frames misalignment. In: IEEE International Symposium on Industrial Embedded Systems (SIES), pp. 261–264. IEEE (2009)
9. Goel, R., Shah, S., Gupta, N., Ananthkrishnan, N.: Modeling, Simulation and Flight Testing of an Autonomous Quadrotor. In: IISc Centenary International Conference and Exhibition on Aerospace Engineering, ICEAE, Bangalore, India, pp. 18–22 (2009)
10. Hoffmann, G.M., Huang, H., Wasl, S.L., Tomlin, E.C.J.: Quadrotor helicopter flight dynamics and control: Theory and experiment. In: AIAA Guidance, Navigation, and Control Conference (2007)
11. Huang, H., Hoffmann, G., Waslander, S., Tomlin, C.: Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3277–3282 (May 2009)
12. Isermann, R.: Mechatronische Systeme: Grundlagen (German Edition). 1. Auflage 1999, 1. korrigierter Nachdruck - Studienausgabe edn. Springer (December 1999)
13. Klein, G., Murray, D.: Parallel tracking and mapping for small AR workspaces. In: 6th IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR), pp. 225–234. IEEE (2007)
14. Kohlbrecher, S., Meyer, J., von Stryk, O., Klingauf, U.: A Flexible and Scalable SLAM System with Full 3D Motion Estimation. In: IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR). IEEE, Kyoto (2011)
15. Leishman, G.: Principles of Helicopter Aerodynamics, 2nd edn. Cambridge Aerospace Series. Cambridge University Press (April 2006)
16. Meyer, J., Strobel, A.: A flexible real-time control system for autonomous vehicles. In: 41st International Symposium on Robotics (ISR) and 6th German Conference on Robotics (ROBOTIK). VDE (2010)
17. Michael, N., Mellinger, D., Lindsey, Q., Kumar, V.: The GRASP Multiple Micro-UAV Testbed. IEEE Robotics Automation Magazine 17(3), 56–65 (2010)
18. Qiang, Y., Bin, X., Yao, Z., Yanping, Y., Haotao, L., Wei, Z.: Visual simulation system for quadrotor unmanned aerial vehicles. In: 30th Chinese Control Conference, pp. 454–459 (July 2011)
19. Rankin, J.: An error model for sensor simulation GPS and differential GPS. In: Position Location and Navigation Symposium, pp. 260–266. IEEE (1994)
20. Rodić, A., Mester, G.: The Modeling and Simulation of an Autonomous Quad-Rotor Microcopter in a Virtual Outdoor Scenario. Acta Polytechnica Hungarica 8(4) (2011)
21. Sendobry, A.: A Model Based Navigation Architecture for Small Unmanned Aerial Vehicles. In: European Navigation Conference. Royal Institute of Navigation (RIN) (November 2011)
22. Weiss, S., Scaramuzza, D., Siegwart, R.: Monocular-SLAM–based navigation for autonomous micro helicopters in GPS-denied environments. Journal of Field Robotics 28(6), 854–874 (2011)

Author Index