Studies in Computational Intelligence 341

Stéphane Doncieux
Nicolas Bredèche
Jean-Baptiste Mouret (Eds.)

New Horizons in Evolutionary Robotics

Extended Contributions from the 2009 EvoDeRob Workshop

Springer

Stéphane Doncieux, Nicolas Bredèche, and Jean-Baptiste Mouret (Eds.)

New Horizons in Evolutionary Robotics

Studies in Computational Intelligence, Volume 341

Editor-in-Chief
Prof. Janusz Kacprzyk
Systems Research Institute
Polish Academy of Sciences
ul. Newelska 6
01-447 Warsaw
Poland
E-mail: kacprzyk@ibspan.waw.pl

Stéphane Doncieux, Nicolas Bredèche,
and Jean-Baptiste Mouret (Eds.)

New Horizons in Evolutionary Robotics

Extended Contributions from the
2009 EvoDeRob Workshop

Springer

Stéphane Doncieux
ISIR - Pierre and Marie Curie University,
CNRS UMR 7222
Boite courrier 173
4, place Jussieu
75252 Paris cedex 05
France
E-mail: stephane.doncieux@isir.upmc.fr

Jean-Baptiste Mouret
ISIR - Pierre and Marie Curie University,
CNRS UMR 7222
Boite courrier 173
4, place Jussieu
75252 Paris cedex 05
France
E-mail: jean-baptiste.mouret@isir.upmc.fr

Nicolas Bredèche
Université Paris-Sud
TAO - Univ. Paris-Sud, INRIA Saclay, CNRS
LRI, Bat. 490
F-91405 Orsay
France
E-mail: nicolas.bredeche@lri.fr

ISBN 978-3-642-18271-6 e-ISBN 978-3-642-18272-3

DOI 10.1007/978-3-642-18272-3

Studies in Computational Intelligence ISSN 1860-949X

Typeset & Cover Design: Scientific Publishing Services Pvt. Ltd., Chennai, India.

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

Preface

Robot design usually follows a reductionist approach where mechanics, electronics, control loops and software are designed in sequential order, without much "feedback" from higher level components. This methodology is arguably the best suited to human engineers and have already provided impressive results. However, it also neglects numerous situations in which, for instance, a minor change in mechanics can substantially simplify software and/or electronics. Conversely, natural evolution allows living organisms to take full advantage of interactions across levels, as it provides an automatic and integrated design process considering effectiveness of organism as a whole.

Loosely inspired by nature, Evolutionary Algorithms (EAs) now provide mature optimization tools that have successfully been applied to the design of many artifacts, from antennas to complete robots. In robotics, it has culminated in the Evolutionary Robotics research field. Modular robotics, swarm robotics or any robot with non-conventional mechanics (e.g. high redundancy, dynamic motion, multi-modality) are challenging robotics applications for which such an integrated approach could prove useful.

Exploring this idea, the workshop entitled "Exploring New Horizons in Evolutionary Design of Robots" was held on October, 11th, 2009, in Saint Louis (USA) during the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-2009). The workshop was intended to discuss recent trends in Evolutionary Robotics (ER) and provided fruitful interactions between roboticists and computer scientists working in Evolutionary Robotics.

This book is a follow-up to the workshop. Firstly, Stephane Doncieux, Jean-Baptiste Mouret, Nicolas Bredeche, the workshop organizers, and Vincent Padois, a roboticist, present a position paper dedicated to bridging the gap between ER and Robotics. Then, Josh Bongard, Kenneth Stanley and Philippe Bidaud, the invited speakers to the workshop, provide each a short position papers about the current trends and challenges at the cross-road of robotics and evolutionary computation. Then, a collection of papers is presented, covering various topics and providing a glimpse to some of the current trends in evolutionary robotics. These papers are extended versions of contributions originally submitted to the workshop, which program commitee was composed of: Josh Bongard (University of Vermont, USA), A.E. Eiben (Vrije Universiteit Amsterdam, Netherlands), Evert Haasdijk (Vrije Universiteit Amsterdam, Netherlands), Jean-Arcady Meyer (Univ. Pierre et Marie Curie, ISIR/CNRS, France), Andrew Philippides (University of Sussex, UK) and Marc Schoenauer (TAO/INRIA, France).

Paris,
June 2010

Stephane Doncieux
Jean-Baptiste Mouret
Nicolas Bredèche

Contents

List of Contributors

Joshua E. Auerbach
Morphology, Evolution and Cognition Laboratory, Department of Computer Science, University of Vermont, Burlington, VT 05405, USA

Philippe Bidaud
Université Pierre and Marie Curie – Paris 6, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, 4 place Jussieu, F-75005, Paris, France

Josh C. Bongard
Morphology, Evolution and Cognition Laboratory, Department of Computer Science, University of Vermont, Burlington, VT 05405, USA

Nicolas Bredeche
TAO - Univ. Paris-Sud, INRIA, CNRS LRI, Bat. 490, F-91405 Orsay, France

Michel de Broissia
Bouygues Travaux Publics, 1, av. E. Freyssinet, 78062 Saint Quentin en Yvelines, France

Karl Crailsheim
Artificial Life Lab of the Department of Zoology, Karl-Franzens University Graz, Austria

Stephane Doncieux
Université Pierre and Marie Curie – Paris 6, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, 4 place Jussieu, F-75005, Paris, France

Jose de Gea
Robotics Innovation Center (DFKI), Robert-Hooke-Str. 5, D-28359 Bremen, Germany

Hongliang Guo
Department of Electrical and Computer Engineering, Stevens Institute of Technology, NJ 07030, USA

Satyandra K. Gupta
Maryland Robotics Center, Department of Mechanical Engineering and Institute for Systems Research, University of Maryland, College Park, MD 20742, USA

Heiko Hamann
Artificial Life Lab of the Department of Zoology, Karl-Franzens University Graz, Austria

Mohamed Hamdaoui
MAS Ecole centrale Paris, Grande Voie des Vignes Chatenay-Malabry, France

Jonathan D. Hiller
Cornell University, Ithaca, USA

Yaochu Jin
Department of Computing, University of Surrey, Guildford, Surrey GU2 7TW, UK

Yohannes Kassahun
Robotics Group, University of Bremen, Robert-Hooke-Str. 5, D-28359 Bremen, Germany

Frank Kirchner
Robotics Group, University of Bremen, Robert-Hooke-Str. 5, D-28359 Bremen, Germany

Hod Lipson
Cornell University, Ithaca, USA

Yan Meng
Department of Electrical and Computer Engineering, Stevens Institute of Technology, NJ 07030, USA

Jean-Marc Montanier
TAO - Univ. Paris-Sud, INRIA, CNRS LRI, Bat. 490, F-91405 Orsay, France

Jean-Baptiste Mouret
Université Pierre and Marie Curie – Paris 6, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, 4 place Jussieu, F-75005, Paris, France

Vincent Padois
Université Pierre and Marie Curie – Paris 6, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, 4 place Jussieu, F-75005, Paris, France

Sebastien Rubrecht
Bouygues Travaux Publics, 1, av. E. Freyssinet, 78062 Saint Quentin en Yvelines, France

Thomas Schmickl
Artificial Life Lab of the Department of Zoology, Karl-Franzens University Graz, Austria

Jakob Schwendner
Robotics Group, University of Bremen, Robert-Hooke-Str. 5, D-28359 Bremen, Germany

Ekta Singla
Université Pierre and Marie Curie – Paris 6, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, 4 place Jussieu, F-75005, Paris, France

Kenneth O. Stanley
School of Electrical Engineering and Computer Science University of Central Florida, Orlando, FL 32816 USA

Jürgen Stradner
Artificial Life Lab of the Department of Zoology, Karl-Franzens University Graz, Austria

Petr Svec
Maryland Robotics Center, Department of Mechanical Engineering and Institute for Systems Research, University of

Maryland, College Park,
MD 20742, USA

Michael T. Tolley
Cornell University, Ithaca, USA

Lutz Winkler
Institut für Prozessrechentechnik,
Automation und Robotik (IPR);
Karlsruher Institut für
Technologie - KIT

Part I
Introduction

Chapter 1
Evolutionary Robotics: Exploring New Horizons

Stéphane Doncieux, Jean-Baptiste Mouret, Nicolas Bredeche, and Vincent Padois

Abstract. This paper considers the field of Evolutionary Robotics (ER) from the perspective of its potential users: roboticists. The core hypothesis motivating this field of research is discussed, as well as the potential use of ER in a robot design process. Four main aspects of ER are presented: (a) ER as an automatic parameter tuning procedure, which is the most mature application and is used to solve real robotics problem, (b) evolutionary-aided design, which may benefit the designer as an efficient tool to build robotic systems (c) ER for online adaptation, i.e. continuous adaptation to changing environment or robot features and (d) automatic synthesis, which corresponds to the automatic design of a mechatronic device and its control system. Critical issues are also presented as well as current trends and pespectives in ER. A section is devoted to a roboticist's point of view and the last section discusses the current status of the field and makes some suggestions to increase its maturity.

1.1 Introduction

The advent of genetic algorithms in the sixties, as a computational abstraction of Darwin's theory of evolution, promised to transfer the richness and efficiency of living organisms to artificial agents, such as robotic systems. This envisioned future inspired a whole field of research, now called *Evolutionary Robotics* (ER) [28, 70, 80], in which researchers create evolutionary algorithms to design robots, or some part of robots such as their "artificial brain". The long-term goal of this field is to

Stéphane Doncieux · Jean-Baptiste Mouret · Vincent Padois
ISIR Pierre and Marie Curie University, CNRS Pyramide Tour 55 Boite courrier 173
4, place Jussieu 75252 Paris cedex 05 France
e-mail: `stephane.doncieux@isir.upmc.fr`, `jean-baptiste.mouret@isir.upmc.fr`, `vincent.padois@isir.upmc.fr`
Nicolas Bredeche
TAO - University Paris-Sud, INRIA, CNRS LRI, Bat. 490, 91405 Orsay, France
e-mail: `nicolas.bredeche@lri.fr`

obtain an automatic process able to design, and even build, an optimal robot given only the specification of a task; the main underlying hypothesis is that Darwin's theory of evolution is the best source of inspiration, in particular because Nature demonstrated its efficiency; the main hope is to obtain machines that fully and robustly exploit the non-linear dynamics offered by their structure and their environment without having to model them explicitly.

After almost twenty years of ER research, simple crawling robots have been automatically designed then manufactured [64]; neural networks have been evolved to allow wheeled robot to avoid obstacles then autonomously charge their battery [29]; neural networks have also been evolved to drive walking [50, 55] and flying [78, 90] robots, as well as self-organizing swarm of robots [5, 38].

These results demonstrate that it is *possible* to automatically design robots or parts of robots with evolutionary algorithms. However, most evolved robots or controllers are not yet competitive with human-designed solutions. What was seen as complex challenges for robotics twenty years ago (walking robots with many degrees of freedom, non-linear control, simple but emergent reactive behaviors, ...) has now been widely investigated in robotics and many efficient solutions have been proposed.

Concurrently with the advances in robotics, evolutionary robotics matured too, both with regards to the basis of evolutionary computation and to its application, and it may be time to reconsider its place with regards to the robotics field. Consequent to this analysis, this paper tackles the simple question: how current evolutionary algorithms can be used in current robotics? After a short reminder of Evolutionary Algorithms (EA) (section (1.2)), we describe the conditions of EA applicability (section 1.3), i.e. when ER should be taken into consideration. We then review the main techniques developed in the ER field by dividing them into mature techniques (section 1.4.1), current trends (section 1.4.2 and 1.4.3) and long-term research (section 1.4.4). We discuss the current challenges of ER and the corresponding perspectives (section 1.5). The point of view of a roboticist is presented in section 1.6 and a discussion on ER as a scientific field together with suggestions to make it more mature end the paper.

1.2 A brief Introduction to Evolutionary Computation

Evolutionary Computation (EC) has been investigated for more than 40 years, with pioneering works both in Computer Sciences (Genetic Algorithms [48]) and Applied Mathematics (Evolution Strategies [85, 89]). Today, the term includes several sub-branches that share the common background of taking a more or less loose inspiration from Darwin's principles of natural selection and blind variations [17]. Moreover, the field has made great progress both considering fundamental concepts (e.g. with the advent of developmental representations [95]) as well as theoretical grounding, from Evolution Strategies [8] to symbolic regression in Genetic Programming [2]. Tools from Evolutionary Computation are now widely accepted within the engineer's meta-heuristic toolbox, and have been successfully applied to several domains, such as automatic design of NASA satellite antennae [65][66][67], chairs [41, 42], electronic circuits [37], photonic crystals [83], polymer optical fibers [68], and real world crawling robots for locomotion [64, 82] to cite a few. What these works have in common is the fact that the objective function is defined such that it gives minimal information on the performance of the evaluated design (e.g. travelled distance, radio signal strength, structure stability). However, the relative freedom in the design definition made it possible to achieve impressive results: satellite antennae from [66] actually ended up being more efficient and more compact than human designed alternatives, and were integrated in the design process of a satellite and sent to space.

From a practical viewpoint, Evolutionary Algorithms are population-based metaheuristics that provide the human engineer with a set of tools to address particular optimization problems[1]. The core principles are built upon two complementary mechanisms, inspired from Darwin's original principles: blind variations (favoring the introduction of new candidates) and survival of the fittest (favoring pressure towards the best individuals). Figure 1.1 illustrates this process with the example of offline behavior optimization of an autonomous agent. The left part of the image illustrates the evolutionary loop: an initial population of random individuals is generated randomly, each individual corresponding to a genome (e.g. a set of parameters or specifications) that defines a particular configuration of robot. Individuals are ranked according to their performance in order to select a subset of these individuals. These "parents" will then be used to generate new "children" individuals, whose genomes are created using stochastic variations, either by recombining several parent genomes and/or mutating a specific parent. This optimization process is termed iterative as it goes on until a pre-defined criterion is reached (e.g. maximum number of evaluations, desired performance, etc.). The right part of the image gives an example of a navigation task (i.e., a two-wheel robot should explore a maze). In this example, deliberately simplified, the aim is to design an automatic control architecture allowing an autonomous mobile robot to explore a maze. On the one side, the genome encodes the parameters of an artificial neural network connecting sensory inputs to motor outputs. On the other side, the performance of this genome is assessed by the behavior of the autonomous robot in the environment. An important remark is that the nature of the evaluation process is completely independent from the viewpoint of evolution in the offline setting, and only results in fitness values to be used for further ranking and selection.

However, the actual expertise of the human engineer is crucial to the success of such algorithms, both with regards to representation issues and evolutionary mechanisms. In fact, several practical questions must be answered before actually launching the evolutionary design process: how to describe a candidate solution? how to explore new candidate solutions? what is the structure of the problem? Moreover, fundamental issues should also be addressed related to the nature of the design process and the viability of the solution, especially regarding *robustness* and *scalability*.

[1] The interested reader is referred to [25] for a complete introduction to Evolutionary Computation and to [80] for an in-depth introduction to Evolutionary Robotics.

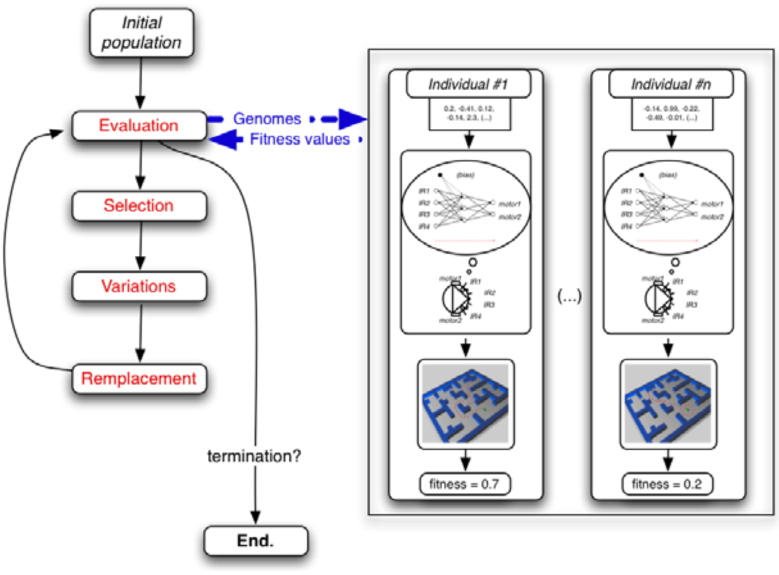


Fig. 1.1 A typical scheme of Evolutionary Algorithm for Autonomous Robot Control Architecture Optimization. **Left:** Evolutionary Process: starting from a population of randomly generated individuals, each individual is evaluated. Based on the outcome of this evaluation, individuals are selected depending on their performance. Then, a new population is generated using two kinds of variation operators: mutation (i.e. a new individual is created as a modified clone of a previous one) and recombination (i.e. a new individual is created by merging several individuals of the previous generation). The evolutionary process goes on until a stopping criterion is reached. **Right:** from the evolutionary algorithm viewpoint, the evaluation operator is simply seen as a blackbox function that maps a set of parameters or structures (i.e. the genome values) to a real value (the "fitness" value of this particular genome). In the particular case of evolutionary robotics, evaluation also encompasses a set of transformation, from the genome values to the actual phenotypic representation of a candidate solution (e.g. a robot with a specific morphology and controller), and then to the fitness value, which is the result of the behavior produced by a particular robot. It should be noted that this terminology is sometimes used in a different fashion in other application domains within EC (merging phenotype and behavior, as in most case there is no temporal aspect in the evaluation process).

1.3 When to Use ER Methods?

Despite the large amount of papers about ER, the question of the underlying hypothesis of this approach is seldom discussed.

While there exists some active research providing sounded theoretical basis of Evolutionary Algorithm [7], the practical use of such methods does not require strong mathematical know-how so as to be efficient in any context. This section attempts to provide an overview of some critical aspects of using Evolutionary Algorithm in the context of Robotics.

1.3.1 Absence of "Optimal" Method

The first and foremost remark concerns the relevance of applying Evolutionary Algorithms rather than another existing methods to solve a given problem. Evolutionary Algorithms do not guarantee convergence towards a global optima, but merely provide an efficient way to address problems that are usually left aside because of their intrinsic difficulties (ill-defined, poorly-defined, implying complex dynamics, etc.). In this scope, ER results from a compromise between applying an iterative algorithm, that may be very slow compared to analytical method, and obtaining approximated solutions rather than no solution at all. Moreover, a key advantage of Evolutionary Robotics is its anytime nature, i.e. the ability to provide one or several solutions, more or less valid, whenever the algorithm is stopped.

1.3.2 Knowledge of Fitness Function Primitives

EA principles consist in producing some diversity and then applying a selective pressure to, statistically, keep the best solutions and discard the others. The key question is that of defining what makes a solution better than the others? The behavior of solutions needs to be quantitatively described. To this end, descriptors of the behavior have to be defined and measured during an evaluation. Such descriptors are the fitness function primitives that should lead the search process towards interesting solutions.

There is no handbook to guide the design of such functions. It is often easy to define objectives able to discriminate between individuals that solve the problem – the preference going to those solving it faster or more efficiently – and likewise it is trivial to discriminate between individuals solving the task and those who don't solve it at all. The most difficult part of a fitness function design comes when individuals not solving the task at all have to be discriminated. For the algorithm to work, this discrimination should lead towards interesting solutions, but naive fitness functions often lead to local extrema, far from interesting solutions. Examples of such cases are numerous, the most famous probably being the obstacle avoidance problem. If simply defined as a count of collisions to be minimized, then the best way to minimize it is ... not to move at all ! Even if the robot is forced to move, it is simpler to find a way to turn round in a safe area, rather than taking the risk of coming close to obstacles and then of learning to use sensors.

1.3.3 Knowledge of Phenotype Primitives

The phenotype is the system to be designed by evolution. In Evolutionary Robotics, it may be the morphology of a robot, its control system or both. The goal is to find a design that best answers to the requirements on the exhibited behavior, requirements quantitatively described in the fitness function.

Evolutionary algorithms can do more than mere numerical optimization, it can also design complex structures like graphs (neural networks, for instance), set of rules, etc. Actually, EA may both assemble and parameterize sets of primitive elements and explore open search spaces, like the space of graphs, for instance. Solving a problem other than parameter optimization with an ER approach implies to find appropriate phenotype primitives and their corresponding genotype primitives that will be assembled or modified by the genetic operators.

For the search to be efficient (and at least more efficient than a pure random search), the encoding, i.e. the way a phenotype or solution is represented in the genotype space, must be carefully chosen. [86] presents a survey of encoding related issues. If strings of symbols are used, the building block hypothesis, direct consequence of the schemata theorem of genetic algorithms, states that alphabet should be minimal and building blocks as small and independent as possible [35]. Schemas have been extended to the concept of forma [84], defined on the basis of an equivalence relation between genomes; this concept can be used to define desired properties of the crossover operator. Likewise, the genotype-phenotype mapping can be studied in order to determine how it changes the difficulty of the problem [87]. When used to generate structures, other rules have been formulated [39], see [53] for a review.

1.4 Where and How to Use EA in the Robot Design Process?

We will distinguish four different uses of EA in a robot design process:

- parameter tuning
- evolutionary aided design
- online evolutionary adaptation
- automatic synthesis

All of them do not have the same maturity. Parameter tuning consists (figure 1.2 (a) and (b)) in using EA as an optimization tool, this is their most frequent use, for which very efficient algorithms now exist, like CMA-ES [44], for instance, or NSGA-II for multi-objective problems [19]. Evolutionary aided design is a more recent trend that differs from parameter tuning in the use of the results. Whereas in parameter tuning, finding optimized parameters is the goal and generally comes at the end of the design process, in evolutionary-aided design, these optimized parameters are to be analyzed by experts to get a better understanding of the problem. Experts will then be able to propose new solutions[2] in a further step. Embodied evolution consists in using EA not only during the design step, but also during robot lifetime, in order to allow it to adapt on-line to drastically changing situations. Lastly, one promising use of EA is evolutionary synthesis. Evolutionary Synthesis is indeed the original motivation behind ER, i.e. building from scratch an autonomous agent by taking some inspiration from the actual evolution mechanisms with the goal to better exploit robot features and environment than what an engineer would do. However, due to its challenging goal, it is also the less mature use of ER as many issues remain to be studied.

[2] Whose parameters might be further tuned with an EA.

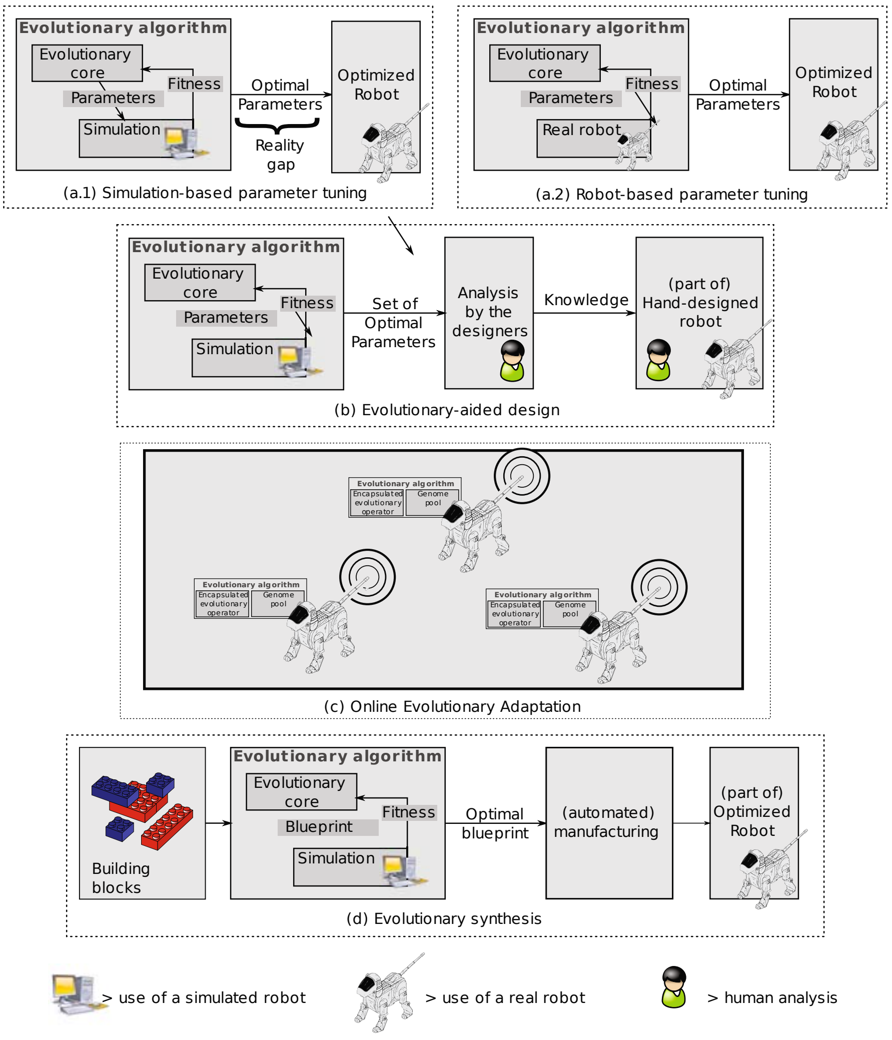


Fig. 1.2 Overview of the different uses of evolutionary algorithms in robotics. On this figure, "evolutionary core" denotes the basic evolutionary loop (see section 1.2) excluding fitness evaluation. (a.1) Parameter tuning based on a simulation then a transfer to the real robots; (a.2) Parameter tuning that uses the real robot to evaluate the fitness; (b) Evolutionary-aided design (e.g innovization); (c) Online Evolutionary Adaptation (e.g. with Embodied Evolution); (d) Evolutionary synthesis (building blocks can be neurons, physical blocks, ...).

1.4.1 Mature Techniques: Parameter Tuning

Evolutionary algorithms, and especially modern evolution strategies [43], are now mature tools for black-box optimization. As they don't impose any constraint on the objective function(s), they can be employed to tune some parameters (constants used in control laws, width of some parts, lengths, ...) of a robot with regards to a set of defined objectives. Typical applications work from a dozen to one hundred real parameters; they involve one to four objectives [18].

One of the easiest setup is to use a robot simulator combined with an EA to find the optimal parameters of a control law [27]. For instance, Kwok and Sheng [59] optimized the parameters of PID controllers for a 6-DOF robot arm with a genetic algorithm. The fitness function was the integral of sum of squared errors of joints, evaluated with a dynamic simulation of the robot. In addition to the many papers that propose to optimize classical control laws, a substantial litterature employed EAs to find optimal parameters of neural networks or fuzzy controllers (see [27] and [28] for some overviews), especially because such controllers are difficult to tune by hand.

Since simulators are never 100% realistic, results obtained in simulation often face what is called “the reality gap”: the optimal parameters obtained in simulation may not be optimal on the real robot; in many cases, the optimized controller may even rely on so badly simulated behaviors that it does not work at all on the real robot. The potential solutions to bridge this reality gap will be described in section 1.5.1.

1.4.2 Current Trend: Evolutionary Aided Design

A growing trend in evolutionary robotics is to use evolutionary algorithms for analysis and exploration tool instead of optimization. Hence, the main goal is not to find an optimal set of parameters but to answer questions such as:

- is it possible to solve a given problem using the system parameterized for another problem?
- what efficiency is to be expected if a given choice is made?
- given a set of different objectives, how antagonistic are they? Can we find a solution that is optimal with regards to all these objectives?
- does some regularities exist between optimal solutions?
- what are the critical parameters?

The typical process is divided into three steps: (1) run an evolutionary algorithm (typically with a simulated system to evaluate the fitness); (2) analyze the results to have a better understanding of the studied system; (3) implement a solution on the real robot with classic (non-evolutionary) techniques but by exploiting the new knowledge to improve the design.

Such an approach was followed by Hauert et al. [47] to evolve decentralized controllers for swarms of unmanned aerial vehicles (UAV). They first evolved neural networks to automatically discover original and efficient strategies. In a second step, they reverse-engineered the obtained controllers to hand-design controllers which capture the simplicity and efficiency of evolved controllers. The hand-design step allows to check the generality of the controllers and to use well etablished methods – to guarantee the stability of controllers, for instance – while taking advantage of the potential innovations brought by the evolutionary process.

Deb and Srinivasan recently demonstrated how multiobjective evolutionary algorithms (see [18]) can bring knowledge of a given system through the analysis of Pareto-optimal solutions, a process called *innovization* [20, 21]. The first step consists in selecting two antagonistic objectives (or more); an evolutionary algorithm is then employed to find the best possible approximation of the Pareto Front; last, Pareto-optimal solutions are analyzed, for instance to find relations between parameters. Typical conclusions are:

- A given parameter is constant for all the Pareto-optimal solutions;
- A given parameter can be computed as a function of another, well identified, parameter;
- A given parameter is stated to be critical;
- Performance seems limited by the range of authorized values for a specific parameter;

This analysis can then be employed to reduce the number of parameters and/or to hand-design some efficient solutions. This approach has been successfully employed to design motors [21] and controllers of a flapping-wing robot [22].

1.4.3 Current Trend: Online Evolutionary Adaptation

Evolutionary Design tools for Robotics are considered as a specific flavor in the Optimization toolbox. Broadly, Evolutionary Design is applied in an off-line manner, prior to the actual use in production of the best solution(s). Whether it is a relevant morphology and/or control architecture, a given solution may or may not feature some kind of generalization capabilities. Indeed, the outcome of the optimization process is still limited to address a specific problem or class of problems, within a limited range of variability, constrained by the experimental setting it was designed in. On the other hand, Online Learning in Machine Learning addresses problem settings where the very definition of the problem is subject to change over time, either slowly or abrutly [10]. In this scope, the goal is to provide a continuously running algorithm providing adaptation in the long run, that is the conception and production phases happen simultaneously. In the scope of ER, Online Evolutionary Adaptation is currently being explored from different perspectives, ranging from endowing robots with some kind of resilient capacity [13] with regards to environmental changes, to adapting known evolutionary algorithms to perform online evolution for single robot or multiple robots [100] or addressing environment-driven evolutionary adaptation [15] (refer to [24] for an overview).

Within Embodied Evolutionary Robotics [100], an online onboard evolutionary algorithm is implemented into one robot or distributed over a population of robots, so as to provide real time adaptation in the environment - An example is shown in figure 1.2-(c). This example illustrates Embodied Evolution in a population of robots: each robot is running an evolutionary algorithm. At time t, only one genome is ”active” and used for robot control. Genomes migrate between robots. Execution of evolutionary operators (variation, selection and replacement) takes places inside the individual robots, but communication and interaction between robots is possibly required for genome migration. In this setup, the evolutionary algorithm is distributed and is running online, i.e. there is no distinction between the design process and the actual use of solution in a real world situation.

Advantages of this approach include the ability to address a new class of problems (problems that require on-line learning), the parallelization of the adaptation (direct consequence of population-based search) and a natural way to address the reality gap (as design constraints enforce onboard algorithms). However this also comes with a price to pay: the lack of control over the experimental setup, such as the difficulty to reset the starting position of the robots inbetween evaluations, may dramatically slow down the optimization process. However, this field of research looks promising as it naturally addresses the unavailability of human intervention and control over the environment as the algorithm is supposed to be completely autonomous from the start. Indeed, a direct consequence is that most of the works in this context have been conducted on real robots [14, 71, 97, 100, 101], which is sufficiently unusual in ER to be mentionned.

The long term goal of online evolutionary adaptation in ER is to provide continuous online adaptation by combining the ability to address the task specified by the human supervisor (the goal) with a priori unknown environmental constraints – that is constraints that cannot be expressed within the fitness function because of the a priori unpredictable nature of the environment. Hence, this field is at the crossroad of traditional optimization techniques (there is an explicitly defined goal to address), open-ended evolution (the environment particularities are to be taken into account during the course of the adaptation process), and online machine learning (the motivation is to provide an efficient algorithmic solution to solve the problem at hand). Compared to other online learning techniques, evolutionary algorithms rely on the same advantages as for black-box optimization: the ability to provide robust optimization through stochastic operators in the scope of problems with limited expert's domain knowledge.

1.4.4 Long Term Research: Automatic Synthesis

As Nature demonstrates it daily, Darwinian evolution is not solely an optimization tool, it is also a powerful automatic *design* process. The marvels accomplished by evolution inspired many researches with the long term goal of automatically designing and even manufacturing complete robotics "lifeforms" with as little human intervention as possible. From the robotics point of view, such an automatic design process could lead to "morpho-functional machines" [45], i.e. robots that can fully adapt the dynamics that emerge from the interactions between their morphology and their controller in order to optimally solve a task. The challenges raised by the automatic synthesis problems range from the understanding of biological evolution (what is the role of development to evolve complex shapes? how did living organisms evolved to modular systems?) to complex engineering problems (how could a robot be automatically manufactured, including its battery and its actuators?).

In a seminal paper, Sims [93] demonstrated how the morphology and neural systems of artificial creatures can be generated automatically with an evolutionary algorithm. Individuals were described as labeled directed graphs, which were then translated to morphology and artificial "brains". Sims was able to obtain creatures that managed to walk, swim and follow a target in a 3-dimensional dynamics simulator. The Golem project [64] put Sims' work in the robotics field by employing a 3D rapid prototyping machine to build walking robots whose morphology and controller were automatically designed by an evolutionary algorithm.

Despite these stimulating results, obtained creatures are by far many order of magnitudes simpler than any real organism. Many researchers hypothesized that designs have to be encoded using a representation that incorporates the principles of modularity (localization of functions), repetition (multiple use of the same substructure) and hierarchy (recursive composition of sub-structures) [62], three features of most biologically-designed systems but also of most engineered artifacts. Such principles led to several generative evolutionary processes that evolve programs that, once executed, generate a blueprint for a robot [49] or a neural network [40, 74]. Abstractions of the development process based on chemical gradients are also investigated [16, 32] and mostly employed to evolve neural networks. However, it has been found that these principles could need to be linked to appropriate selective pressures to be fully exploited [75], hence emphasizing that the synthesis problem may not be solely an encoding problem.

1.5 Frontiers of ER and Perspectives

ER still has many open issues. Here are several of the most critical:

- how to avoid the reality gap? Or, how to limit the risks of using an imperfect simulation to evaluate the performance of a system within an opportunistic learning scheme;
- how does it scale relative to behavior complexity? This question reveals to be actually tightly linked to fitness landscapes and exploration abilities of the EA. We will consider this question under this point of view;
- genericity of evolved solutions? For CPU time considerations, evaluations are as short as possible, and correspond thus to the behavior of the robot within only a limited set of conditions;

We will briefly discuss them in this section and sketch out current work and perspectives.

1.5.1 Reality Gap

The reality gap problem is clearly the most critical one with regards to practical applications. In theory, the reality gap should not even exist as the optimization process could be achieved directly on the target robotics setup. Several works have actually achieved evolution on real robots, such as for evolving homing behavior for a mobile robot [29], optimizing the walking gait of an AIBO robot [50], of a pneumatic hexapod robot with complex dynamics [63] or even a humanoid robot [102]. While the optimization on the real robot guarantees the relevance of the obtained solutions, this has several major drawbacks as it can be quite consuming in term of time. As a consequence, only small populations (most of the time less than 30) and few generations (often less than 500) are performed in such a context, therefore limiting the problem that can be addressed to rather simple tasks.

Given that simulation is difficult to avoid in most practical situations, a new question arises regarding how to avoid, or at least limit, the reality gap effect, or, stated differently, how to ensure that the fitness function gives similar results within the simulation and on the robot. As a perfectly accurate simulation is highly unlikely to be available, many works focus on coping with the simulation intrinsic approximations and mistakes. A representive contribution is that of Jakobi [51] with minimal simulations: only the accurately simulated parts of the environment are taken into account and random noise is added to keep the evolutionary process from being mistakenly optimistic. Another approach consists in estimating how well a particular controller transfers to the reality on the basis of a few experiments on the real robot and then use this objective to push towards solutions that transfer well [57].

Instead of learning behaviors, ER techniques may be used to directly learn a model of a real mechanical device [11, 12, 56, 88]. Learning techniques can even be used to correct model errors online [33] or even to learn a complete model of the robot in action [13], thus opening the way towards robots able to adapt to motor failures in an online evolution scheme.

1.5.2 Fitness Landscape and Exploration

While Evolutionary Robotics has long been intended to address challenging problems, most of the achievements so far concern quite simply defined robotics problems: wall avoidance, food gathering, walking distance maximization, and other simple navigation tasks [80]. One of the major pitfalls is that the difficulty of a problem often arises with the complexity of the fitness landscape: while a smooth, convex fitness landscape with no noise will be quite easy to deal with, most of the problems from the real world often comes with multimodal, noisy fitness landscapes that feature neutrality regions. The direct consequence is that search may often get stalled, would it be at the very beginning of the algorithm execution (i.e. a bootstrap problem) or during the course of evolution (i.e. premature convergence), with no hint on how to escape a local optimum or on how to direct the search within a region where all neighboring candidate solutions are equally rewarded.

Exploiting expert knowledge is a good way to escape from local optima, but as it is not always available, several solutions have been considered, the most prominent ones are listed here:

- decomposing the problem into sub-problems, each of them being solved separately, either implemented manually or learned. The resulting behaviors can then be combined through an action-selection mechanism, that may itself eventually be tuned through evolution [34, 54, 99];

- reformulating the target objective into an incremental problem, where the problem is decomposed into possibly simpler fitness functions of gradually increasing difficulties, ultimately leading to what is reffered to as incremental evolution [36];
- reformulating the target objective into a set of fitnesses optimized independantly in a multi-objective context [73]. As opposed to the previous point, a multi-objective formulation of the problem makes it possible to avoid ranking sub-fitnesses difficulties, which is often a tricky issue;
- using co-evolution to build a dynamically changing evaluation difficulty in competitive tasks [79, 96];
- changing the evaluation during evolution to focus first on simpler problems and make the robot face progressively more difficult versions of the same task[4];
- likewise exploring solutions of increasing complexity with mechanisms protecting innovation to give new solutions a chance to prove their value [94];
- searching for novelty of behavior instead of efficiency [60, 61]. This avoids getting trapped in local optima while enhancing the search ability over robot behaviors;
- in a multi-objective scheme, adding an objective that explicitely rewards the novelty or diversity of behaviors [23, 72, 76, 77];
- putting the human into the loop. For instance, this is the kind of approach that has previously been called "innovization" [20], where the search algorithm is used to provide a basis for the expert to refine the optimization process and to provide original solutions.

1.5.3 Genericity of Evolved Solutions

One major requirement of optimization in the context of ill or poorly defined problems is to provide solutions capable of generalization, or robustness. It may indeed be very difficult to grasp all the aspects of a problem during the conception phase as the combinatorial explosion makes it impossible to generate all possible test cases. A typical example is that of a walking robot where all inclinations or textures of the ground cannot be generated during optimization, but where generalization is possible over examples. In this setup, both the experimental setup and the representation formalism are of the utmost importance. For example, relying on a test case generator or adding noise during the course of evaluation is an efficient way to enforce generalization [46]. Also, some specific representations are more fitted for generalization: artificial neural network, for example, are naturally biased towards generalization. Anyway, the actual robustness of evolved solutions remains an open question that has been seldom studied.

1.6 A Roboticist Point of View

From the Robotics point of view, control parameters tuning is a critical task since the resulting controllers dictate the behaviours of the robots. Considering the parametric identification of the dynamics model of a robot, it is important to recall that the considered model is an approximation of the real system dynamics. Thus, the identified parameters are a compromise that captures some physical properties of the system as well as some of the unknown or unmodelled dynamics. Equivalently, when tuning a PID controller for the control of a joint at the position level, the goal is to find a compromise that will best reject perturbations, most of them being hard to model (friction, backlash). In these two examples, parameters tuning is, by essence, meant to be achieved on the real system in order to best capture properties which cannot be accounted for a priori. In fact, this approach is often retained in Robotics and parametric identification and PID control tuning are then widely covered subjects in Robotics textbooks [52], [91].

One may thus argue that the use of EA in such contexts is probably not appropriate nor needed. This is only partially true. In fact, these ”roboticists” methods are well suited for problems where robots do not physically interact much with their environments. When this is not the case, either the interactions are restricted to a specific context and can be modelled using simple representation of the environment or they are not restricted to specific objects or modes of contact and in that case it is hard to tune a parametric controller that will fit a wide variety of situations. The latter case can actually not be tackled with ”low-level” controllers only and higher level decision making is often required. That is where EA may contribute: either by helping to understand what are the important physical parameters to consider within the context of a complex interaction between a robot and its environment or by tuning higher level decison making controllers that cannot be tuned using physics-based approaches only. From a more general point of view, challenges in Robotics are at the interface between high level planning methods and lower level control. This is probably the space where learning based approaches [92] and EA can best contribute in current Robotics problems.

However, EA may also still bring strong contributions in the domain of robots design. As a matter of fact, the emergence of service Robotics raises the problems of energetic efficiency and physical compliance (which is one of the prerequisite to safety) at a level such that the design problem can no longer be considered from the sole structural perspective [98], [103]. In fact, when trying to design energy efficient and compliant robots, one should consider the design and control problems as a whole. New control modes have to be explored together with new types of actuators and transmissions and EA could be one of the tools used to tackle this vast exploration problem.

1.7 Discussion

Evolutionary Robotics is a young field of research that needs time to mature and to identify its place in the engineering and science ecosystems. From a general viewpoint, there is a need of positionning ER with related research fields. On the one hand, *Machine learning* has successfully proved its strong potential impact on robotics current challenges as demonstrated in [1] and the emergence of a new research field at the frontier between both domains is illustrated by recent publications (e.g. [92]). Although ER may be included in a weak definition of Machine Learning, there is a definitly stronger emphasis on structural design and weakly formalized open-ended problems. On the other hand, *Developmental robotics*[3] [3, 69, 81] also shares many common concerns with ER. Developmental robotics takes its inspirations from Developmental Psychology rather than Evolutionary Biology, and is concerned with learning of efficient behavior. The constraint of realism with respect to the developmental psychology ideas and models is at the core of this approach: the motivation is to provide efficient solutions as well as to validate models actually observed in Nature, while ER takes loose inspiration from it.

Up to now, the vast majority of ER published papers are proofs of concept that demonstrate, usually with a small set of experiments, that a given technique has enough potential to be further investigated. For instance, Floreano et al. [30] showed in several papers that the weights of a fully-connected continuous time recurrent neural network can be evolved to make a real robot avoid obstacles. This experiment showed that evolution *can* automatically design a controller for such a simple but useful behavior. It didn't show (and didn't aim at showing) that this method was the best to implement obstacle avoidance on a real robot.

Proofs of concept are undoubtedly important to explore new ideas. When Bongard et al. introduced resilient robotics with evolutionary algorithms [13], they did the spadework for new robotic abilities. Their work opened a previously almost not explored area of research, hence emphasizing the key role of proof of concepts. Nevertheless, such proofs of concept are only the first — often the easiest — step of an original scientific work. In an applicative context, using a given method requires to be convinced that it is one of the best method available or, at least, that this method will work with a large probability of success. This requires more than a proof of concept: strength and limitations need to be well understood, the alternatives methods should be extensively compared, the success on a large set of problems should be demonstrated. The difficulties are similar to include a given method in a larger scientific work: each brick must be strong enough to support the higher-level bricks. The abundance of proof of concepts in ER (instead of more solid knowledge building) may be the main reason for which the field is not improving faster: almost no paper re-use the results from previous papers by other authors; most of the time, researchers create a new system from scratch.

This line of thought lead us to the following conclusion: to mature, *ER need less proofs of concepts and more solid results*. To our opinion, ER has one foot in robotic engineering and one foot in experimental sciences. It therefore has much to gain from importing the best practices from these two fields.

[3] Sometimes also referred to as *Epigenetic robotics*.

1.7.1 Good Robotic Engineering Practices

To be useful in engineering, ER needs to show that it can solve important problems in robotics better than other methods. The only approach available to reach this goal is to draw extensive comparisons in which ER methods are compared to state of the art methods. The comparison should be fair and thus include at least a discussion on the most important aspects of the considered methodologies:

- the main properties of the methodology and the initial goals of the designers;
- the relative efficiency of each methodology as it is classically measured by roboticists;
- the knowledge required to apply each methodology: what should be known from the problem? What has to be done to apply each methodology?
- the constraints for the methodologies to be applied;
- the running time or the required CPU power (this is critical in the case of online ER with mobile robots).

A common pitfall is to ignore the constraints that may have driven the development of a particular methodology. In this case, the comparison isn't fair.

Another good engineering practice is to avoid to re-invent new solutions to already solved problems. The universality of the Darwinian principles suggests that everything could be designed by evolutionary algorithms. As a consequence, many ER papers deal with simple problems, for instance reactive obstacle avoidance, with the ambition that the algorithm should scale up to interesting problems, once refined. While this kind of simple tests is a way to validate the basic feature of an algorithm, this is at best a waste of computational power from a practical point of view. Smart humans spent years to develop efficient, if not optimal, approaches to solve many problems. It is currently pretentious to hope that even the best evolutionary algorithm could surpass them in a few hours (or days) of computation. On the other side, some problems (see section 1.6) are open from a roboticist point of view and ER could significantly contribute to their solution. Put differently, ER researchers should start with the state of the art for the studied problem and improve it, instead of trying to reach it from scratch, at least when then intend to show the potential of ER approaches.

Our last point is a famous engineering slogan that may seem obvious: keep it simple and stupid[4]. Many current projects are so intricate that it is impossible to replicate them in a slightly different setup. Moreover, they often are a combination of weak bricks that are not well understood. In other words, making ER a mature science requires to simplify the methods to identify the essential parts and discard everything else. If two non-critical algorithms are available to perform a sub-task, the simpler one should be chosen, even if it is less efficient; it will be easier to re-implement and to understand and will not restrict the conclusions.

[4] KISS, see `http://en.wikipedia.org/wiki/KISS_principle`

1.7.2 Good Experimental Sciences Practices

Altough the theory of evolutionary computation is improving, the evolutionary algorithms employed in ER rely on complex fitness functions and, most of the time, complex genotypes and phenotypes (e.g. neural networks). ER consequently mostly depends on empirical proofs and not on theorems. This is a significant departure from computer science, in which complexity and proofs of convergence reign, and makes ER closer to experimental sciences such as biology.

The first and foremost lesson from experimental sciences is the use of statistical tests. A single run is not sufficient to conclude anything except that there exists a solution. Comparing two sets of experiments without checking the statistical significance of the comparison is also meaningless. Additionally, comparing two methods on a single benchmark prevents the authors to conclude anything about the generality of the introduced approach. Most of the time, a Student T-test is employed to compare experiments. This statistical test assumes that the results follows a Gaussian distribution. This is often false in ER and especially if several problems are used [31]. As described in [31] for evolutionary computation, non-parametric tests, such as the Wilcoxon signed-rank test, appear more adapted to ER than parametric tests. Additionally, an experimental methodology is still lacking in ER. For instance, how to guarantee that the optimal parameters were used during a comparison? Some progress have been recently accomplished to transfer the "design of experiments" [26] approach to evolutionary computation [6, 58]. Some ideas can also be borrowed from the machine learning literature [9]. This work could be a starting point for more a rigorous design of ER experiments.

The second practice in experimental sciences that must be imported into ER is the habit of reproducing experiments. Most ER experiments are *never* reproduced by independant researchers. Contrarily to theoretical work, it is difficult to rely on experiments that have never been reproduced. However, it is often difficult to reproduce ER experiments because of the intricacy of many ER systems and the large number of parameters.

ER experiments are most of the time done in simulation and therefore internet provides a simple solution to this problem: distributing the source code, which contains every details of the algorithms. Although the solution is simple, it is not that often used. It must be emphasized that the primary goal of distributing the source code of an experiment is to let people have access to every detail of the experiment: without the code some important data may lack to reproduce the experiment. It is not to distribute a well polished and documented code (sadly, no researcher has time to do it). Furthermore, in front of the huge number of parameters, it would be worth sharing the experience on one's work reproduction to better understand what is important, what is not and how robust a particular algorithm is[5].

[5] The EvoRob_Db web site (`http://www.isir.fr/evorob_db`) aims at facilitating such exchanges.

References

1. Abbeel, P., Coates, A., Quigley, M., Ng, A.: An application of reinforcement learning to aerobatic helicopter flight. In: Advances in Neural Information Processing Systems (NIPS), vol. 19. MIT Press, Cambridge (2007)
2. Amil, M., Bredeche, N., Gagné, C., Gelly, S., Schoenauer, M., Teytaud, O.: A statistical learning perspective of genetic programming. In: Proceedings of the 12th European Conference on Genetic Programming at Evostar 2009 (2009)
3. Asada, M., Hosoda, K., Kuniyoshi, Y., Ishiguro, H., Inui, T., Yoshikawa, Y., Ogino, M., Yoshida, C.: Cognitive developmental robotics: a survey. IEEE Transactions on Autonomous Mental Development 1(1), 12–34 (2009)
4. Auerbach, J., Bongard, J.: How Robot Morphology and Training Order Affect the Learning of Multiple Behaviors. In: Proceedings of the IEEE Congress on Evolutionary Computation (2009)
5. Baele, G., Bredeche, N., Haasdijk, E., Maere, S., Michiels, N., van de Peer, Y., Schmickl, T., Schwarzer, C., Thenius, R.: Open-ended on-board evolutionary robotics for robot swarms. In: IEEE Congress on Evolutionary Computation, CEC 2009 (2009)
6. Bartz-Beielstein, T., Preuss, M.: Experimental research in evolutionary computation. In: Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation, pp. 3001–3020. ACM, New York (2007)
7. Beyer, H.G.: The Theory of Evolution Strategies. Springer, Heidelberg (2001)
8. Beyer, H.G., Schwefel, H.P.: Evolution strategies – A comprehensive introduction. Natural Computing 1, 3–52 (2002)
9. Birattari, M., Zlochin, M., Dorigo, M.: Towards a theory of practice in metaheuristics design: A machine learning perspective. RAIRO–Theoretical Informatics and Applications 40, 353–369 (2006)
10. Blum, A.: On-line algorithms in machine learning. In: Proceedings of the Workshop on On-Line Algorithms, Dagstuhl, pp. 306–325. Springer, Heidelberg (1996)
11. Bongard, J., Lipson, H.: Nonlinear system identification using coevolution of models and tests. IEEE Transactions on Evolutionary Computation 9(4), 361–384 (2005)
12. Bongard, J., Lipson, H.: Automated reverse engineering of nonlinear dynamical systems. Proceedings of the National Academy of Sciences 104(24), 9943–9948 (2007)
13. Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. Science 314(5802), 1118–1121 (2006)
14. Bredeche, N., Haasdijk, E., Eiben, A.: On-line, On-board Evolution of Robot Controllers. In: Evolution Artificielle / Artificial Evolution. Strasbourg France (2009)
15. Bredeche, N., Montanier, J.-M.: Environment-driven Embodied Evolution in a Population of Autonomous Agents. In: Schaefer, R., et al. (eds.) PPSN XI. LNCS, vol. 6239, pp. 290–299. Springer, Heidelberg (2010)
16. D'Ambrosio, D.B., Stanley, K.O.: A novel generative encoding for exploiting neural network sensor and output geometry. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007 (2007)
17. Darwin, C.: On the Origin of Species by Means of Natural Selection or the Preservation of Favoured Races in the Struggle for Life. John Murray, London (1859)
18. Deb, K.: Multi-objectives optimization using evolutionnary algorithms. Wiley, Chichester (2001)
19. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)

20. Deb, K., Srinivasan, A.: Innovization: Innovating design principles through optimization. In: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, pp. 1629–1636. ACM, New York (2006)
21. Deb, K., Srinivasan, A.: INNOVIZATION: Discovery of Innovative Design Principles Through Multiobjective Evolutionary Optimization. In: Multiobjective Problem Solving from Nature: From Concepts to Applications, pp. 243–262 (2007)
22. Doncieux, S., Hamdaoui, M.: Evolutionary Algorithms to Analyse and Design a Controller for a Flapping Wings Aircraft. In: New Horizons in Evolutionary Robotics: Post-Proceedings of the 2009 EvoDeRob Workshop. Springer, Heidelberg (2010)
23. Doncieux, S., Mouret, J.B.: Behavioral diversity measures for evolutionary robotics. In: IEEE Congress on Evolutionary Computation, CEC 2010 (to appear, 2010)
24. Eiben, A., Haasdijk, E., Bredeche, N.: Embodied, on-line, on-board evolution for autonomous robotics. In: Levi, P., Kernbach, S. (eds.) Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution, Cognitive Systems Monographs, vol. 7, pp. 361–382. Springer, Heidelberg (2010)
25. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Natural Computing Series. Springer, Heidelberg (2003)
26. Fisher, R.: Design of Experiments. British Medical Journal 1(3923), 554 (1936)
27. Fleming, P.J., Purshouse, R.C.: Evolutionary algorithms in control systems engineering: a survey. Control Engineering Practice 10(11), 1223–1241 (2002)
28. Floreano, D., Mattiussi, C.: Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies. In: Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge (2008)
29. Floreano, D., Mondada, F.: Evolution of homing navigation in a real mobile robot. IEEE Transactions on Systems, Man, and Cybernetics-Part B (1996)
30. Floreano, D., Mondada, F.: Evolutionary neurocontrollers for autonomous mobile robots. Neural Networks 11, 1461–1478 (1998)
31. García, S., Molina, D., Lozano, M., Herrera, F.: A study on the use of non-parametric tests for analyzing the evolutionary algorithms' behaviour: a case study on the CEC 2005 special session on real parameter optimization. Journal of Heuristics 15(6), 617–644 (2009)
32. Gauci, J.J., Stanley, K.O.: Generating large-scale neural networks through discovering geometric regularities. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2007 (2007)
33. Gloye, A., Wiesel, F., Tenchio, O., Simon, M.: Reinforcing the driving quality of soccer playing robots by anticipation (verbesserung der fahreigenschaften von fu ballspielenden robotern durch antizipation). IT - Information Technology 47, 250–257 (2005)
34. Godzik, N., Schoenauer, M., Sebag, M.: Evolving symbolic controllers. In: Evo Workshops, pp. 638–650 (2003)
35. Goldberg, D.: Genetic Algorithms in Search and Optimization. Addison-Wesley, Reading (1989)
36. Gomez, F., Miikkulainen, R.: Incremental evolution of complex general behavior. Adaptive Behavior 5(3-4), 317–342 (1997)
37. Greenwood, G.W., Tyrrell, A.M.: Introduction to Evolvable Hardware: A Practical Guide for Designing Self-Adaptive Systems. Wiley-IEEE Press (2006)
38. Gross, R., Bonani, M., Mondada, F., Dorigo, M.: Autonomous self-assembly in swarm-bots. IEEE Transactions on Robotics 22(6), 1115–1130 (2006)
39. Gruau, F.: Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. Ph.D. thesis, Claude Bernard-Lyon I University (1994)

40. Gruau, F.: Automatic definition of modular neural networks. Adaptive Behaviour 3(2), 151–183 (1995)
41. Hamda, H., Jouve, F., Lutton, E., Schoenauer, M., Sebag, M.: Compact unstructured representations in evolutionary topological optimum design. Applied Intelligence 16, 139–155 (2002)
42. Hamda, H., Schoenauer, M.: Adaptive techniques for evolutionary topological optimum design. In: Parmee, I. (ed.) Evolutionary Design and Manufacture, pp. 123–136. Springer, Heidelberg (2000)
43. Hansen, N., Muller, S.D., Koumoutsakos, P.: Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). Evolutionary Computation 11(1), 1–18 (2003)
44. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary computation 9(2), 159–195 (2001)
45. Hara, F., Pfeifer, R.: Morpho-Functional Machines: The New Species: Designing Embodied Intelligence. Springer, Heidelberg (2003)
46. Hartland, C., Bredeche, N., Sebag, M.: Memory-enhanced evolutionary robotics. In: IEEE Congress on Evolutionary Computation (2009)
47. Hauert, S., Zufferey, J.C., Floreano, D.: Reverse-engineering of Artificially Evolved Controllers for Swarms of Robots. In: IEEE Congress on Evolutionary Computation (2009)
48. Holland, J.H.: Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence. University of Michigan Press, Ann Arbor (1975)
49. Hornby, G.S.: Measuring, enabling and comparing modularity, regularity and hierarchy in evolutionary design. In: Proceedings of the 2005 conference on Genetic and evolutionary computation, pp. 1729–1736 (2005)
50. Hornby, G.S., Takamura, S., Yokono, J., Hanagata, O., Yamamoto, T., Fujita, M.: Evolving robust gaits with aibo. In: IEEE International Conference on Robotics and Automation, pp. 3040–3045 (2000)
51. Jakobi, N.: Evolutionary robotics and the radical envelope-of-noise hypothesis. Adaptive Behavior 6(2), 325–368 (1997)
52. Khalil, W., Dombre, E.: Modeling, Identification and Control of Robots, 3rd edn. Taylor & Francis, Inc., Abington (2002)
53. Kicinger, R., Arciszewski, T., Jong, K.: Evolutionary computation and structural design: A survey of the state-of-the-art. Computers & Structures 83(23-24), 1943–1978 (2005)
54. Kim, K.J., Cho, S.B.: Robot Action Selection for Higher Behaviors with CAM-Brain Modules. In: Proceedings of the 32nd ISR (International Symposium on Robotics), vol. 19, p. 21 (2001)
55. Kodjabachian, J., Meyer, J.A.: Evolution and development of neural networks controlling locomotion, gradient-following, and obstacle-avoidance in artificial insects. IEEE Transactions on Neural Networks 9, 796–812 (1997)
56. Koos, S., Mouret, J.B., Doncieux, S.: Automatic system identification based on coevolution of models and tests. In: IEEE Congress on Evolutionary Computation, CEC 2009 (2009)
57. Koos, S., Mouret, J.B., Doncieux, S.: Crossing the reality gap in evolutionary robotics by promoting transferable controllers. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO 2010, ACM, New York (2010)

58. Kramer, O., Gloger, B., Goebels, A.: An experimental analysis of evolution strategies and particle swarm optimisers using design of experiments. In: Proceedings of the 9th Annual Conference on Genetic and Evolutionary Computation, pp. 674–681. ACM, New York (2007)
59. Kwok, D.P., Sheng, F.: Genetic algorithm and simulated annealing for optimal robot arm PID control. In: Proceedings of the First IEEE Conference on IEEE World Congress on Computational Intelligence, pp. 707–713 (1994)
60. Lehman, J., Stanley, K.O.: Exploiting Open-Endedness to Solve Problems Through the Search for Novelty. Artificial Life 11, 329 (2008)
61. Lehman, J., Stanley, K.O.: Abandoning objectives: Evolution through the search of novelty alone. Evolutionary Computation (2010)
62. Lipson, H.: Principles of Modularity, Regularity, and Hierarchy for Scalable Systems. In: Genetic and Evolutionary Computation Conference (GECCO 2004) Workshop on Modularity, regularity and Hierarchy (2004)
63. Lipson, H., Bongard, J., Zykov, V., Malone, E.: Evolutionary robotics for legged machines: from simulation to physical reality. Intelligent Autonomous Systems 9, 9 (2006)
64. Lipson, H., Pollack, J.B.: Automatic design and manufacture of robotic life forms. Nature 406(406), 974–978 (2000)
65. Lohn, J., Crawford, J., Globus, A., Hornby, G.S., Kraus, W., Larchev, G., Pryor, A., Srivastava, D.: Evolvable systems for space applications. In: International Conference on Space Mission Challenges for Information Technology (2003)
66. Lohn, J., Hornby, G., Linden, D.: An evolved antenna for deployment on NASAs space technology 5 mission. In: Genetic Programming Theory and Practice II, pp. 301–315 (2004)
67. Lohn, J.D., Linden, D.S., Hornby, G.S., Kraus, W.F., Rodriguez-Arroyo, A.: Evolutionary design of an x-band antenna for nasa's space technology 5 mission. In: Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware, EH 2003, IEEE Computer Society Press, Washington (2003)
68. Manos, S., Large, M.C.J., Poladian, L.: Evolutionary design of single-mode microstructured polymer optical fibres using an artificial embryogeny representation. In: GECCO 2007: Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation, pp. 2549–2556. ACM, New York (2007)
69. Metta, G., Sandini, G., Vernon, D., Natale, L., Nori, F.: The iCub humanoid robot: an open platform for research in embodied cognition. In: Permis: Performance Metrics for Intelligent Systems Workshop. Washington DC, USA (2008)
70. Meyer, J.A., Guillot, A.: Biologically-inspired Robots. In: Handbook of Robotics. Springer, Heidelberg (2008)
71. Montanier, J.M., Bredeche, N.: Embedded evolutionary robotics: The (1+1)-restart-online adaptation algorithm. In: Proceedings of IROS Workshop Exploring New Horizons in the Evolutionary Design of Robots (2009)
72. Mouret, J.B.: Novelty-based multiobjectivization. In: Proceedings of IROS Workshop Exploring New Horizons in the Evolutionary Design of Robots (2009)
73. Mouret, J.B., Doncieux, S.: Incremental evolution of animats' behaviors as a multi-objective optimization. In: Asada, M., Hallam, J.C.T., Meyer, J.-A., Tani, J. (eds.) SAB 2008. LNCS (LNAI), vol. 5040, pp. 210–219. Springer, Heidelberg (2008)
74. Mouret, J.B., Doncieux, S.: MENNAG: a modular, regular and hierarchical encoding for neural-networks based on attribute grammars. Evolutionary Intelligence 1(3), 187–207 (2008)

75. Mouret, J.B., Doncieux, S.: Evolving modular neural-networks through exaptation. In: IEEE Congress on Evolutionary Computation, CEC 2009 (2009)
76. Mouret, J.B., Doncieux, S.: Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In: IEEE Congress on Evolutionary Computation, CEC 2009 (2009)
77. Mouret, J.B., Doncieux, S.: Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In: GECCO 2009: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation. ACM, New York (2009)
78. Mouret, J.B., Doncieux, S., Meyer, J.A.: Incremental evolution of target-following neuro-controllers for flapping-wing animats. In: Nolfi, S., Baldassarre, G., Calabretta, R., Hallam, J.C.T., Marocco, D., Meyer, J.-A., Miglino, O., Parisi, D. (eds.) SAB 2006. LNCS (LNAI), vol. 4095, pp. 606–618. Springer, Heidelberg (2006)
79. Nolfi, S., Floreano, D.: How co-evolution can enhance the adaptive power of artificial evolution: Implications for evolutionary robotics. In: Proceedings of the First European Workshop on Evolutionary Robotics (EvoRobot 1998), pp. 22–38 (1998)
80. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press, Cambridge (2001)
81. Oudeyer, P.Y., Kaplan, F., Hafner, V.: Intrinsic motivation systems for autonomous mental development. IEEE Transactions on Evolutionary Computation 1(11), 265–286 (2007)
82. Pollack, J.B., Lipson, H.: The golem project: Evolving hardware bodies and brains. In: EH 2000: Proceedings of the 2nd NASA/DoD workshop on Evolvable Hardware, p. 37. IEEE Computer Society, Los Alamitos (2000)
83. Preble, S., Lipson, H., Lipson, M.: Two-dimensional photonic crystals designed by evolutionary algorithms. Applied Physics Letters 86 (2005)
84. Radcliffe, N.: The algebra of genetic algorithms. Annals of Mathematics and Artificial Intelligence 10(4), 339–384 (1994)
85. Rechenberg, I.: Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Frommann-Holzboog, Stuttgart (1973)
86. Ronald, S.: Robust encodings in genetic algorithms: a survey of encoding issues. In: IEEE International Conference on Evolutionary Computation, pp. 43–48 (1997)
87. Rothlauf, F.: Representations for Genetic And Evolutionary Algorithms. Springer, GmbH & Co. K, Heidelberg (2006)
88. Schmidt, M., Lipson, H.: Distilling free-form natural laws from experimental data. Science 324(5923), 81–85 (2009)
89. Schwefel, H.P.: Numerical Optimization of Computer Models. John Wiley & Sons, Inc., New York (1981)
90. Shim, Y., Husbands, P.: Feathered Flyer: Integrating Morphological Computation and Sensory Reflexes into a Physically Simulated Flapping-Wing Robot for Robust Flight Manoeuvre. In: Almeida e Costa, F., Rocha, L.M., Costa, E., Harvey, I., Coutinho, A. (eds.) ECAL 2007. LNCS (LNAI), vol. 4648, pp. 756–765. Springer, Heidelberg (2007)
91. Siciliano, B., Sciavicco, L., Villani, L., Oriolo, G.: Robotics: Modelling, Planning and Control. Springer, Heidelberg (2008)
92. Sigaud, O., Peters, J. (eds.): From Motor Learning to Interaction Learning in Robots. Studies in Computational Intelligence, vol. 264, pp. 1–12. Springer, Heidelberg (2010)
93. 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)

94. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10(2), 99–127 (2002)
95. Stanley, K.O., Miikkulainen, R.: A taxonomy for artificial embryogeny. Artificial Life 9(2), 93–130 (2003)
96. Stanley, K.O., Miikkulainen, R.: Competitive Coevolution through Evolutionary Complexification. Journal of Artificial Intelligence Research 21, 63–100 (2004)
97. Usui, Y., Arita, T.: Situated and embodied evolution in collective evolutionary robotics. In: Proc. of the 8th International Symposium on Artificial Life and Robotics, pp. 212–215 (2003)
98. Vanderborght, B., Verrelest, B., Van Ham, R., Van Damme, M., Beyl, P., Lefeber, D.: Development of a compliance controller to reduce energy consumption for bipedal robots. Autonomous Robots 24(4), 419–434 (2008)
99. Wahde, M.: A method for behavioural organization for autonomous robots based on evolutionary optimization of utility functions. Proceedings of the I MECH E Part I Journal of Systems & Control Engineering 217(4), 249–258 (2003)
100. Watson, R.A., Ficici, S.G., Pollack, J.B.: Embodied evolution: Embodying an evolutionary algorithm in a population of robots. In: 1999 Congress on Evolutionary Computation, pp. 335–342 (1999)
101. Watson, R.A., Ficici, S.G., Pollack, J.B.: Embodied evolution: Distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems 39(1), 1–18 (2002)
102. Wolff, K., Sandberg, D., Wahde, M.: Evolutionary optimization of a bipedal gait in a physical robot. In: IEEE Congress on Evolutionary Computation, CEC 2008, pp. 440–445 (2008)
103. Zinn, M., Khatib, O., Roth, B., Salisbury, J.: Playing it safe [human-friendly robots]. IEEE Robotics Automation Magazine 11(2), 12–21 (2004), doi:10.1109/MRA.2004.1310938

Part II
Invited Position Papers

Chapter 2
The ‘What’, ‘How’ and the ‘Why’ of Evolutionary Robotics

Josh Bongard

Abstract. The field of embodied artificial intelligence is maturing, and as such has progressed from *what* questions (“what is embodiment?”) to *how* questions: how should the body plan of an autonomous robot be designed to maximize the chance that it will exhibit a desired set of behaviors. In order to stand on its own however, rather than a reaction to classical AI, the field of embodied AI must address *why* questions as well: why should body and brain both be considered when creating intelligent machines? This report provides three new lines of evidence for why the body plays an important role in cognition: (1) an autonomous robot must be able to adapt behavior in the face of drastic, unanticipated change to its body; (2) under-explored body plans raise new research questions related to cognition; and (3) optimizing body plans accelerates the automated design of intelligent machines, compared to leaving them fixed.

2.1 The *What* of Embodiment

Classical artificial intelligence proceeded under the assumption that cognition could be realized in computer programs that were not able to directly sense or affect the physical surroundings of the computer in which they were housed. This approach has led to many successful computer applications, but has shed relatively little light on the nature of cognition. Since the 1980s however, there has been a growing awareness that an agent must be able to act and be acted on by its surroundings, and sense the repercussions of those actions. This requires that an agent be both situated and embodied: it must have the ability to directly sense the world, and have a body with which to act on the world.

Josh Bongard
Morphology, Evolution and Cognition Laboratory, Department of Computer Science, University of Vermont, Burlington, VT 05405, USA
e-mail: josh.bongard@uvm.edu

If a body plan, actuation and sensation are necessary to realize intelligent behavior, the question then arises as to how to choose an appropriate body plan for the desired behavior. Although it is generally agreed that the design of a robot controller for complex behavior is best left to automated optimization, many hold that human intuition can be applied to the design of robot body plans:

> "Humans are much better at designing physical systems than they are at designing intelligent control systems: complex powered machines have been in existence for over 150 years, whereas it is safe to say that no truly intelligent autonomous machine has ever been built by a human." ([6], p. 22).

However, there are many explicit and implicit design decisions that must be made when choosing a robot body plan. As a simple example, a wheeled robot may be appropriate for rapid, efficient travel over flat terrain, but a legged body plan may be more suitable for rough terrain. If a legged body plan is chosen, how many legs should the agent have? Should the spine be flexible or rigid? If flexible, how flexible? This raises the issue then of how to systematically make these decisions.

2.2 The *How* of Embodiment

In order to overcome the infinitude of design decisions that must be made when designing a robot body plan, some researchers in the field of biorobotics design robot morphologies based on animal body plans. However, as has been discovered in the history of engineering many times over, direct copying of nature's designs does not always succeed. This is due to the face that the ecological niche inhabited by the animal and the robot are not usually equivalent: many aspects of the animal's body plan may have evolved for that niche and are therefore not applicable to the robot's niche. For instance, it took the realization that because aircraft are much larger than the birds which originally inspired their design, fixed wings are required rather than flapping wings. This then requires the roboticist to determine, for each aspect of the animal's body plan, which are design innovations that will still serve in the robot's ecological niche, and which are innovations evolved to meet biology-specific constraints. As an example of a biology-specific constraint, it has been hypothesized that the reason why wheels never evolved in animals larger than bacteria is because it would be difficult to provide nutrients across a freely-rotating axle (see for example [4], p. 542).

This difficulty argues against the claim made in [6] that human designers can formulate appropriate robot body plans. Instead, rather than mimic a product of natural selection, one can mimic natural selection itself, and harness it to design a robot's body plan along with its control policy. This is the field of evolutionary robotics, in which an evolutionary algorithm is used to automate the process of robot design.

2.3 The *Why* of Embodiment

However, such resulting algorithms are notoriously complex, and require more than the above verbal defense to justify their use. To this end, three arguments for why one should evolve body plans are given below:

- a physical robot that undergoes physical damage and thus sustains an unanticipated and unobservable change to its body plan can evolve a new description of its topology, and use the evolved model to generate a compensatory controller;
- virtual robots may initially be evolved with few body parts, and then gradually evolved with more body parts to accelerate evolution; and
- as the desired task become more difficult, the ability for evolution to alter robot body plans even slightly increases the probability of evolving a successful robot.

2.4 Why Consider Topological Change to a Robot’s Body Plan?

One of the repercussions of considering that the body plans an important role in intelligent behavior leads one to consider how agents should deal with changes to that body. One class of such change is physical damage in which the robot loses one or more body parts. This leads to a change in the topology of the robot’s body plan, and typically necessitates a different control policy to recover functionality.

This scenario was investigated in [2], in which we employed a physical quadrupedal robot (Fig. 2.1). This robot begins by evolving a body plan (i.e. its mental model) that accurately reflects the topology of its (initially unknown) physical body plan. This is accomplished by actuating the physical robot with a random control policy and recording the resulting sensor data; actuating a candidate mental model with the same control policy and recording the resulting virtual sensor data; and computing the error of the model as the difference between the physical and virtual sensor data. The evolutionary algorithm then searches the space of virtual body plans for one that minimizes this error within a given time period.

A second evolutionary algorithm then optimizes a controller on the mental model such that the mental model performs the desired behavior, which in that work was forward locomotion. Once such a controller is found, it is executed by the physical robot, which often results in the desired behavior.

Physical damage was then simulated by removing part of one of the robot’s leg. As it could not directly sense the damage, the first evolutionary algorithm was restarted, which continues the search for an appropriate body plan. A new candidate body plan, when actuated with a control policy the physical robot has already performed, must reproduce the actions of the damaged robot; this rapidly leads to the evolution of a virtual robot that reflects the damage of the physical robot.

The first evolutionary algorithm is again paused, and the second evolutionary algorithm is restarted. This latter algorithm now searches for a compensating controller that allows the virtual, damaged robot to recover forward locomotion. Once found, this new control policy is executed by the physical, damaged robot; this often results in the recovery of locomotion.

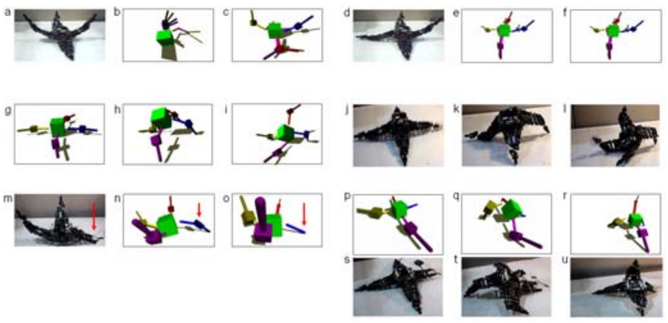


Fig. 2.1 The Resilient Machine Project. The robot performs a random action (**a**). A set of random models (one of which is shown in **b**) is synthesized into approximate models (one of which is shown in **c**). A new action is then synthesized to create maximal model disagreement and is performed by the physical robot (**d**), after which further modeling ensues. This cycle continues for a fixed period or until no further model improvement is possible (**e-f**). The best model is then used to synthesize a behavior (in this case, forward locomotion, the first few movements of which are shown in **g-i**). This behavior is then executed by the physical robot (**j-l**). The robot then suffers damage (the lower part of the right leg breaks off; **m**). Modeling then recommences with the best model so far (**n**), and using the same process of modeling and experimentation, eventually discovers the damage (**o**). The new model is then used to synthesize a new behavior (**p-r**), which is executed by the physical robot (**s-u**), allowing it to recover functionality despite this unanticipated change.

Thus, by considering the fact that a complex physical robot may sustain several physical damage, a solution presents itself in the form of an algorithm that evolves a virtual body plan to match the current state of the physical robot's body plan. This allows for not only the automatic generation of walking controllers, but also the automated recovery of walking after unanticipated damage.

2.5 Why Evolve Robot Body Plans Initially at a Low Resolution?

In attempting to automatically design robots capable of increasingly sophisticated behavior, it is often necessary to smooth the fitness landscape so that. This allows for incremental improvements from primitive behaviors to more complex ones. Several techniques exist for doing this, such as robot shaping (eg. [5]).

Another approach to shaping aside from simplifying the task environment is to simplify the robots' initial body plans. This approach has recently been explored in [1]. In that work purely passive three-dimensional structures were evolved to maximize their displacement when they fell. Initially, structures were grown at 'low resolution': relatively few, large spheres could be used to build the structure. Once structures with high fitness were found, the existing structures in the population were re-grown at 'high resolution': more, smaller spheres were allowed for construction. An evolved low- and high-dimensional structure are shown in Fig. 2.2.

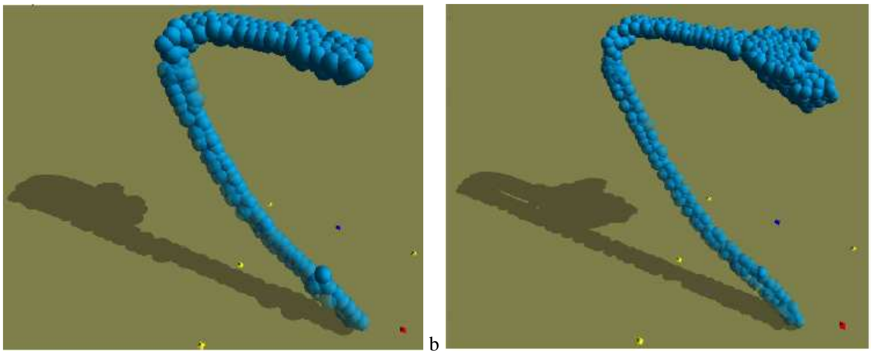
a  b

Fig. 2.2 Gradually Increasing Morphological Complexity. a: A structure evolved for maximum displacement when dropped. **a:** The same structure, but regrown at higher resolution.

Several runs in which low-resolution structures were initially evolved and then further evolved at higher resolution were performed, and compared against a set of control runs in which structures were evolved at high resolution throughout. It was found that both runs found structures of about the same fitness, but the variable-resolution runs found such structures in significantly less time.

This time savings was realized for two reasons. First, the low-resolution structures could be simulated and thus evolved more rapidly. Second, the evolutionary method employed increased the probability that a structure regrown at high resolution would have similar behavior—and thus similar fitness—as the structure originally grown from the same genotype at lower resolution.

This latter property was realized by adapting the neuroevolution technique HyperNEAT. HyperNEAT encodes a genome that takes as input a vector *p*, which indicates a position in a high-dimensional space, and outputs a value that can be used to construct a phenotype. The way in which genomes are encoded ensures that positions near one another output similar values.

In previous work in which the genotype was used to label the weights of a neural network, it was shown that HyperNEAT could evolve smaller neural networks that still functioned when regrown at higher resolution, because the synapses in the larger neural network lay near to their corresponding synapses in the smaller neural network.

This principle was exploited in [1]: during growth, the HyperNEAT genome takes as input a candidate position for placing a sphere, and the output indicates whether the sphere should be placed or not. When a structure is regrown at higher resolution, similar positions are queried as were queried during growth of the low-dimensional structure. This thereby increases the chance that the high-resolution structure will have the same shape—and thus behavior and fitness—as the low-dimensional structure.

2.6 Why Allow Body Plans to Change during Behavior Optimization?

A third reason for evolving robot body plans is that allowing evolution to change the body plan slightly increases the probability of finding a successful robot. This was demonstrated in [3]. For many robot body plans, slight changes will lead to slight changes in behavior, and thus slight changes in fitness. It is well-known that mutations that have slight phenotypic effect has a higher probability of being beneficial, compared to mutations that cause large phenotypic change. Thus, placing some aspects of a robot's body plan under evolutionary control can smooth the fitness landscape, and increase the evolvability of the overall system.

This principle was demonstrated using an anthropomorphic arm robot that was evolved to perform one or more object manipulation tasks: grasping an object, lifting an object, and or distinguishing between different objects. When robots were evolved for only one of these tasks, there was little evolutionary difference between runs in which only the control policy was evolved, and runs in which control and morphology were optimized. As the task became more challenging however by selecting for robots that could grasp, lift and distinguish between objects, runs in which control and morphology were evolved together performed much better than runs in which only control was evolved.

It is hypothesized that when the robots were evolved to perform all three tasks at once, the fitness landscape is more rugged because there are several trade-offs between these different competencies. For example a mutation that improves the robot's grasp may adversely affect its ability to distinguish between different objects: if it grasps all objects more tightly, it may fail to generate different sensor signatures when grasping the different objects and cannot thus distinguish between them.

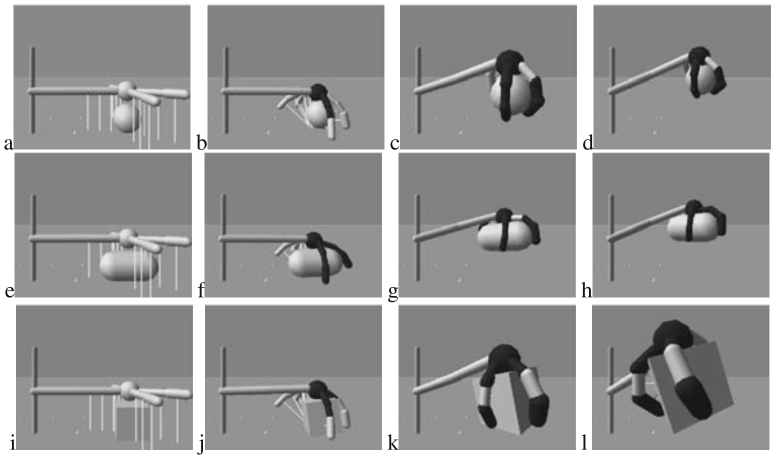


Fig. 2.3 The anthropomorphic arm evolved to lift, grasp and distinguish between different-shaped objects.

However, by allowing evolution to slightly change the radii of the fingers (as seen in Fig. 2.3), when it grasps objects of different shapes, different parts of the fingers may come into contact with the object. Each finger part contains a touch sensor, so grasping different objects may cause different subsets of the touch sensors to fire, and these different patterns can be used to distinguish between the objects. This is visualized in Fig. 2.3, in which finger parts are blackened if the touch sensor in it is firing, and different black-and-white patterns can be observed when the robot grasps different objects. Thus, slight changes to the robot’s morphology can smooth the fitness landscape by reducing the tradeoffs between the different behaviors being evolved.

References

[1] Auerbach, J., Bongard, J.: Evolving CPPNs to Grow Three-Dimensional Physical Structures. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO (to appear, 2010)
[2] Bongard, J., Zykov, V., Lipson, H.: Resilient machines through continuous self-modeling. Science 314, 1118–1121 (2006)
[3] Bongard, J.C.: The utility of evolving simulated robot morphology increases with task complexity for object manipulation. Artificial Life (2010), doi:10.1162/artl.2010.Bongard.024
[4] Chrisley, R., Begeer, S.: Artificial intelligence: critical concepts. Taylor & Francis, Abington (2000)
[5] Dorigo, M., Colombetti, M.: Robot shaping: Developing situated agents through learning. Artificial Intelligence 70(2), 321–370 (1994)
[6] Nelson, A., Barlow, G., Doitsidis, L.: Fitness functions in evolutionary robotics: A survey and analysis. Robotics and Autonomous Systems 57(4), 345–370 (2009)

Chapter 3
Why Evolutionary Robotics Will Matter

Kenneth O. Stanley

Abstract. While at present Evolutionary Robotics (ER) is generally not studied in mainstream robotics, the main idea in this article is that ER has the opportunity to gain relevance by taking seriously its natural inspiration. The chasm that separates the behavior of robots today from the robustness and fluidity of organisms in nature is most naturally addressed by an approach that indeed respects the process through which such organisms originated. Yet the challenge is to identify the elusive missing ingredient that would allow ER to realize its full potential.

3.1 Joining the Mainstream

Evolutionary Robotics (ER) is not a mainstream topic in robotics. It is easy to find syllabi for "Introduction to Robotics" courses on the Internet without even a mention of ER in the entire semester schedule. Yet ER *should* be important to robotics as an active subcommunity that aims to address many of the same challenges. The question is how this divide between mainstream robotics and ER will ultimately be bridged. This article attempts to address this question by looking mainly forward toward the promise of ER and how that promise will make it increasingly relevant to robotics as a whole.

An important goal for robotics in general is to avoid the stereotypical stilted, jittery motion of awkward machines and move instead towards fluid, natural behaviors. In this light, it is interesting to note the tools with which mainstream robotics proposes to address this challenge. The 2007 *Introduction to Robotics* syllabus at Stanford University [7] gives a sense of what these tools are in the mainstream view: spatial descriptions, forward kinematics, Jacobians for velocities and static forces, computer vision, inverse kinematics and trajectory generation, acceleration and inertia, dynamics, PID control, joint space control, operational space control,

Kenneth O. Stanley
Department of Electrical Engineering and Computer Science,
University of Central Florida, Orlando, FL 32816 USA
e-mail: kstanley@cs.ucf.edu

and force control. Note that nowhere in the syllabus is ER mentioned. The classic ER book *Evolutionary Robotics* [12] is rarely cited as a textbook or even an auxiliary reference in robotics courses either. The day when ER genuinely impacts robotics and not only evolutionary computation remains ahead.

Yet even a cursory look at nature hints at why ER does have the potential for dramatic impact. After all, every organism on Earth is the product of *evolution*, and no robot yet created through conventional means even comes close to the stealth of a lion, the grace of a bird, or the balance of a human. Even in simulation such capabilities are not convincingly reproduced, so it is not just a matter of mechanical limitation. Both the robustness and fluidity of natural movement remain symbols of how far we have to go.

For example, when a human sprains an ankle, he or she may walk with a limp, but the entire motor system recalibrates almost instantaneously to compensate for the disability without the risk of falling. Throughout childhood the body changes in height and proportion, yet walking remains seamless. At the same time, although it is a qualitative observation, the *fluidity* of natural movement is unmatched. One need only watch horses at play or monkeys swinging from branches to appreciate the gaping chasm between natural fluidity and robotics today.

If ER could produce robots with the same robustness and fluidity, it would earn its place as a canonical topic in mainstream robotics. Yet the problem is that ER does not presently produce such behavior any more than mainstream robotics does. For example, a survey of evolved bipeds, while impressive for the progress that it demonstrates so far, shows that the products of ER today are nevertheless still brittle and highly stereotyped compared to nature's flexible solutions [1, 6, 9, 14, 23].

3.2 Bridging the Gap

The future of ER lies in taking this achievement gap seriously. We need to acknowledge that when applied to robotics, evolutionary algorithms *should* produce artifacts that remind us of nature, which provides our primary inspiration for running evolution in the first place. That does not mean that human-level intelligence is necessary to achieve, but fluid motion and robustness belong more realistically within scope. If evolutionary computation can offer nothing else, at least it should offer that.

The almost-magical elegance and grace of the products of natural evolution is rarely acknowledged within the technical-minded confines of the research community yet nevertheless deserve our attention as a source of inspiration and indeed as a proof of what is possible through ER. It is exactly that magical ingredient that mainstream robotics seems to lack. That exquisite, seamless fluidity of motion that unfolds without apparent effort is a clue to what may be possible. By ignoring this elusive facet of life on Earth, mainstream robotics risks missing what ER is positioned to gain.

Then what does it mean that ER does not today exhibit that same magic? The answer is that ER is poised at the brink of opportunity; the essential prerequisite to our progress as a field is to acknowledge that something fundamental is missing. Yet whatever that missing ingredient is, it is closer to our purview than to the traditional tools of mainstream robotics. Rather than a negative sentiment, acknowledging this missing link suggests a profound opportunity for change just over the horizon.

3.3 Realizing the Promise

Of course, the natural next question is what shape that change may take. Someday, we should hope to evolve e.g. a single biped (or quadruped) neurocontroller that works in almost *any* biped morphology, just as our brains allow us to walk as our body grows and changes. Rather than starting life walking right away, as almost every evolved biped does today [1, 6, 9, 14, 23], it should learn on its own the dimensions and dynamics of its new body and rise from the ground to walk after some experimentation. In effect, the hypothesis is that the robustness and fluidity of nature is earned at the expense of a period of adaptation and habituation that occurs early in life, just as babies learn to walk.

If this hypothesis is right, it suggests that adaptation, which in artificial neural networks follows from synaptic plasticity, may be an important part of any model that approaches the elusive superiority of nature. Yet synaptic plasticity remains an open area of investigation in neuroevolution [2, 4, 11, 13, 17, 18, 19, 20]. Recent work in our research group aims to combine synaptic plasticity with the indirect encoding in HyperNEAT [5, 21], which would allow regular patterns of plasticity rules to be distributed across the network [15]. At present, although there is already precedent for incorporating synaptic plasticity into simple ER models (e.g. in controlling wheeled Khepera robots [4]), it has not yet been combined with controllers that must attempt feats like learning to walk with variable morphology during their lifetime.

In any case, simply combining neuroevolution [3, 22, 24] and synaptic plasticity alone is not a complete answer. The model of plasticity will likely need to be especially sophisticated and refined to be able to support unprecedented robustness. Furthermore, new computational abstractions of evolution may need to be developed that capture the open-ended process through which the products of nature were discovered [8, 10, 16]. Thus significant research lies ahead. Yet these research directions provide a hint of where opportunity may lie.

The important point for this article is that if we can evolve a controller that wakes up inside any body and learns to make it work, all without the need for any traditional analysis whatsoever, there is the potential to revolutionize mainstream robotics. It happened in nature and it should therefore be possible in ER. So while today some in the mainstream may see ER as unnecessary or suboptimal, its promise is in its inspiration, which encompasses the most robust robotic systems on Earth: *nature*.

References

[1] Allen, B., Faloutsos, P.: Complex networks of simple neurons for bipedal locomotion. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS (2009)

[2] Baxter, J.: The evolution of learning algorithms for artificial neural networks. In: Green, D., Bossomaier, T. (eds.) Complex Systems, pp. 313–326. IOS Press, Amsterdam (1992)

[3] Floreano, D., Dürr, P., Mattiussi, C.: Neuroevolution: from architectures to learning. Evolutionary Intelligence 1, 47–62 (2008)

[4] Floreano, D., Urzelai, J.: Evolutionary robots with online self-organization and behavioral fitness. Neural Networks 13, 431–443 (2000)

[5] Gauci, J., Stanley, K.O.: Autonomous evolution of topographic regularities in artificial neural networks. Neural Computation (to appear, 2010)

[6] Hein, D., Hild, M., Berger, R.: Evolution of biped walking using neural oscillators and physical simulation. In: Visser, U., Ribeiro, F., Ohashi, T., Dellaert, F. (eds.) RoboCup 2007: Robot Soccer World Cup XI. LNCS (LNAI), vol. 5001, pp. 433–440. Springer, Heidelberg (2008)

[7] Khatib, O.: Introduction to robotics (CS223A) syllabus. Stanford University (2007)

[8] Lehman, J., Stanley, K.O.: Exploiting open-endedness to solve problems through the search for novelty. In: Bullock, S., Noble, J., Watson, R., Bedau, M. (eds.) Proceedings of the Eleventh International Conference on Artificial Life (Alife XI). MIT Press, Cambridge (2008)

[9] McHale, G., Husbands, P.: Gasnets and other evolvable neural networks applied to bipedal locomotion. In: From Animals to Animats 8 (2004)

[10] Mouret, J.-B.: Novelty-based multiobjectivization. In: Proceedings of the Workshop on Exploring New Horizons in Evolutionary Design of Robots IEEE/RSJ International Conference on Intelligent Robots and Systems (2009)

[11] Niv, Y., Joel, D., Meilijson, I., Ruppin, E.: Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors. Adaptive Behavior 10(1), 5–24 (2002)

[12] Nolfi, S., Floreano, D.: Evolutionary Robotics. MIT Press, Cambridge (2000)

[13] Nolfi, S., Parisi, D.: Learning to adapt to changing environments in evolving neural networks. Adaptive Behavior 5, 75–98 (1996)

[14] Reil, T., Husbands, P.: Evolution of central pattern generators for bipedal walking in a real-time physics environment. IEEE Transactions on Evolutionary Computation 6(2), 159–168 (2002)

[15] Risi, S., Stanley, K.O.: Indirectly encoding neural plasticity as a pattern of local rules. In: Doncieux, S., Girard, B., Guillot, A., Hallam, J., Meyer, J.-A., Mouret, J.-B. (eds.) SAB 2010. LNCS, vol. 6226, pp. 533–543. Springer, Heidelberg (2010)

[16] Risi, S., Vanderbleek, S.D., Hughes, C.E., Stanley, K.O.: How novelty search escapes the deceptive trap of learning to learn. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2009. ACM Press, New York (2009)

[17] Soltoggio, A.: Neural Plasticity and Minimal Topologies for Reward-Based Learning. In: Proceedings of the 8th International Conference on Hybrid Intelligent Systems, pp. 637–642. IEEE Computer Society, Los Alamitos (2008)

[18] Soltoggio, A., Bullinaria, J.A., Mattiussi, C., Dürr, P., Floreano, D.: Evolutionary advantages of neuromodulated plasticity in dynamic, reward-based scenarios. In: Artificial Life XI, pp. 569–576. MIT Press, Cambridge (2008)

[19] Soltoggio, A., Dürr, P., Mattiussi, C., Floreano, D.: Evolving neuromodulatory topologies for reinforcement learning-like problems. In: Proceedings of the IEEE Congress on Evolutionary Computation (2007)
[20] Stanley, K.O., Bryant, B.D., Miikkulainen, R.: Evolving adaptive neural networks with and without adaptive synapses. In: Proceedings of the 2003 IEEE Congress on Evolutionary Computation, CEC 2003. IEEE Press, Canberra (2003)
[21] Stanley, K.O., D'Ambrosio, D.B., Gauci, J.: A hypercube-based indirect encoding for evolving large-scale neural networks. Artificial Life 15(2), 185–212 (2009)
[22] Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 10, 99–127 (2002)
[23] Van de Panne, M., Lamouret, A.: Guided optimization for balanced locomotion. In: Terzopoulos, D., Thalmann, D. (eds.) Sixth Eurographics Workshop on Animation and Simulation, pp. 165–177. Springer, Heidelberg (1995)
[24] Yao, X.: Evolving artificial neural networks. Proceedings of the IEEE 87(9), 1423–1447 (1999)

Chapter 4
Evolutionary Algorithms in the Design of Complex Robotic Systems

Philippe Bidaud

4.1 Introduction

To expand the potential of the robotic systems by introducing innovative mechanical structures and to endow the systems of sophisticated behaviors taking into account in particular the variability of the environments in which they are intended to evolve constitutes challenges for the research in design. The needs in terms of rationality and efficiency in these new challenges for robotic engineering lead the development of systems to assist engineers in the different phases of their design activities. Generally speaking, design is defined as a process in which for a given description of desired functions and constraints to satisfy called specifications we try to produce the description of an artifact or several of them fitting the specifications. The design process may entail the creation of new solutions or evolutions from an existing solution. To this aim, series of activities are performed by which designer perception of a problem is transformed into an output satisfying the problem. The entire design process is basically an iterative process which can be viewed at a conceptual level as a chain of activities which consists in 1) Clarifying the requirements and needs (the output is frequently a problem statement since it is rarely given at the beginning) 2) Defining the constraints by analyzing the operating environment (functional and performance specification) 3) Generating concepts and solutions 4) Modeling and analyzing the behavior 5) Testing and evaluating the proposed design(s) 6) Refining and optimizing the design of selected solutions. Design methodologies in the field of mechanical engineering has been first introduced on the base of prescriptive models then on descriptive models and more recently as computer-based models [1]. The aim of a prescriptive model is typically to provide guidelines or frameworks to organize and structure the process of creating instructional design activities.

Philippe Bidaud
Institut des Systemes Intelligents et de Robotique,
Université Pierre et Marie Curie – CNRS UMR 7222
e-mail: bidaud@isir.upmc.fr

Traditionally, these design recommendations are based on the experience coming from identical or similar developments. It is quite evident that this kind of concept generally inhibits innovation.

Descriptive design methods rely on an explicit description of designers practice extracted from previous experiences. Collected data and information about reasoning, decisions, options, trade-offs, etc . . . are organized into patterns to capture design rationale and construct a formal or semi-formal structure so that the design rationale can be used in the decision-making process.

The progresses in computer sciences have contributed in relaxing the design process to approach the problem in a more global way mixing prescriptive and descriptive approaches and by introducing more formal theory and methodology and more rigorous practices for the comparison of design alternatives, this by introducing mathematically-based optimization methods, simulation-based performance evaluation and assisted decision processes [4]. In addition, more iterative approaches have also been adopted and it has been observed that interactions between the different stages were an important factor for controlling the development of the design process as well as the designer preferences [2].

The need for global system design rather than for component has made particularly desirable to take into account simultaneously a significant number of criteria and this in a coherent way with regard to the function to be realized. Besides, the investigation of alternative solutions, the introduction of the knowledge acquired by the analysis of the results to refine the specifications constitutes now essential elements in the process of innovation and the mastery of optimal solutions.

If rationality and logic are essential components in technical progress, we also need powerful tools to stimulate creativity. Emergence of new solutions may result from a dialogue between science and serendipity, effect by which one discovers something fortunate by an intelligent exploration.

Robotics systems relate clearly this last frame. They constitute by nature complex systems which require a global and creative approach for their design.

4.2 Particularities of the Robotic System Design

Robotics offers a technology with which the integration of various activities as well as the flexibility to adapt to task/environment changes can be achieved. Basically a robotic system is made up with a mechanical structure, which can be an articulated multi-body system with a fixed or variable topology or a deformable structure (continuous or discrete). An external power is provided to the mechanical structure by discrete or continuous actuators to produce motions and/or physical actions over the working environment. The internal state of the system as well as its activity with respect to the task can be observed through different sensors integrated in the powered structure. The state and the interaction with the environment is controlled by a controller which act at different levels, from the low-level control which considers joint performances to the higher levels which may consider advanced autonomous or interactive behaviors. Examples of these systems can be for instance manipulation and/or locomotion systems which may have different topology : serial architectures, parallel architectures, legged system, wheel-legged systems, compliant structures, tensegrity and deployable structures, polymorphic structures.

Traditionally, a hierarchical design approach for developing and implementing robotic systems is used. This model seeks to provide a structure under which robotic systems (existing or new) can be rapidly developed and implemented. This solution is badly adapted to robotic system design where it is a question of mastering the global performances of systems which depend at the same time on its structure and its control. A way to proceed consists in considering simultaneously these various aspects through a phase of preliminary design based on a formal inspection of the high-level architectural design and of the control system. This phase is conducted to achieve confidence that the design satisfies the functional and nonfunctional requirements and that the system decomposition is in conformance with methods for multidisciplinary design and optimization. One of the most challenging problems in preliminary design is the ability to handle complex behaviors with numerous design variables and constraints. Classical optimization methodologies fall short in very large and complex domains.

Robotic systems belongs to large scale engineering systems for which the design problem can be defined as a nonlinear optimization problem under constraints with mixed continuous and discrete design variables. Among the methods adapted to this class of problem, branch and bound methods including heuristics rules, simulated annealing end evolutionary algorithms which are also implicit enumeration procedures have shown a real potential and are today considered as particularly relevant. The main advantage of these methods over the others traditionally used in mechanical design is that they do not require gradients and convexity of the problem and they will find a global optimum.

The potential of genetic algorithms is precisely situated among all the other optimization techniques in [3]. Since their introduction in the 60s, an extensive work have been performed about the adaptive mechanisms of evolution and their adaptation to solve real world problems with genetic algorithms and derived techniques in order to provide the most efficient and robust stochastic optimization processes.

4.3 Parameters and Evaluation of Robotic Systems

A robotic system as the one shown on the figure 4.1 is typically made of tens of different bodies articulated to each other to provide a sufficient mobility to overcome large obstacles. In the preliminary design stage of such a system, we look for an optimal structural design : a topology for the arrangement of the bodies and joints, joint and structural kinematic characteristics, mass and inertia distribution, power transmissions, etc . . . and a way to coordinate the movement of the system and to distribute the transmission forces for driving it over a given terrain.

For the design of such system, quantitative descriptions of the expected robot performances are a key issue. Performance measures which may be used

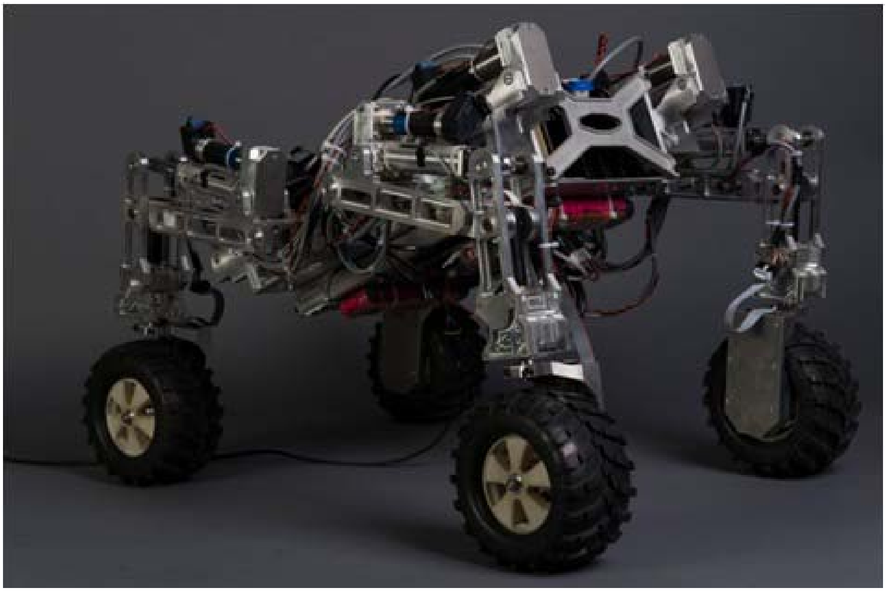


Fig. 4.1 Example of mobile robotic system with large range active suspensions for high crossover capabilities.

depend upon the nature of the system and on the nature of the task to be performed. They will make possible to assign a numerical value (the cost) to a system considering a particular manner (control and programming) of executing this task. Finding the best system and the best way to execute the task can be translated into an optimization problem in which the formulation of appropriate objective functions requires to take into account task variability.

In the preliminary design phase of a robotic system, the functional requirements which are frequently considered are related to motion capabilities of the system and its force transmission performances.

Kinematic and mechanism theories provide powerful tools for mobility analysis of the mechanisms supporting the robotic devices motions and constraints in mechanisms. They can be investigated from joint screw system characteristics with respect to their geometric particularities [5]. Stationary extreme and internal configurations which are directly related to the workspace of a system as well as uncertain configurations (non-controlable) are accessible from screw system associated with the set of joints. Different analysis based on the rank of the Jacobians in the input/output infinitesimal displacements (or velocities) transmission can be carried out for considering the motion transmission properties. Reciprocally, the force transmission properties may be explored based on the generalized concept of force ellipsoid [6, 7]. Both of them have expressions similar to the manipulability index which is a quantitative measure of the closeness of a robot configuration to singularity.

The dynamic response of the robot manipulator may also be investigated from the cartesian dynamic model of the system by incorporating link flexibilities as well as on the control law implemented on the system [8]. For a general impedance control law, several parameters can be adjusted to control the overall behavior of a mechanical system in SE(3). These parameters may also be adjusted to deal with transitions in between free and constraint motions as well as the physical contact stability in between the system and its physical environment [9]. All the basic performances mentioned above are configuration dependant. Moreover, if the system is redundant with respect to the task constraints, it will also depend on the use made of the self motion of the system.

Many of the criteria used in robotic system design are not aimed at capturing directional structure in the workspace, rather perhaps at smoothing the structure across the workspace. The initial manipulability measure introduced by Yoshikawa is directionless making it impossible to discern directional preferences. Task-oriented measures have been proposed to obtain quantitative indices of the system's capability, in each configuration, of its performances along given task space directions [12].

What is said here with regards to manipulation systems can also be said about locomotion systems. Terrain traversability, postural stability and others similar performances are dependant of the configuration of the system with respect to the tasks and for the dynamic performances, there are dependant also of the history of the system. Besides, these characteristics are generally influenced by the control of the system by which, in a lot of situations we try to take benefit of the redundancy either kinematic, or in actuation. Numerous illustrations can be given. For example, we can modify significantly the capacities of obstacles crossing for wheeled robot by integrating into the control the non-sliding constraints of for a better optimization of the distribution of traction forces [11].

4.4 Evolutionary Algorithms in the Robotic System Design

Design techniques from evolutionary algorithms have been successfully applied to a large number of mechanical design problems from the design of mechanical structures exhibiting complex behaviors to mechanisms realizing particular functions. In what follows, we return to some particular developments we have made on robotic system design.

4.4.1 Kinematic Design of Robot Manipulators

We have investigated task based design of modular robotic systems using Genetic Algorithms (GA) in the mid-90s, this based on a 3D kinematic description for modular serial manipulators and a two-level GA to optimize their topology from task specifications. The topology is evolved for adaptation to a global task constituted by several required end-effector configurations (subtasks). The implemented

GA optimizes several performance criteria under constraints. An adaptative multi-chromosome evolutionary algorithm (AMEA) is used to perform task based design of modular robotic systems. The kinematic design is optimized by the AMEA which uses both binary and real encoding (for kinematic and configuration parameters). In problems considered for illustration, the manipulator has to reach a predefined set of goal frames in a 3D cluttered environment. Its design is evaluated with geometric and kinematic performance measures. Optimization results for a 3D task are given and compared with a simple genetic algorithm. They clearly show the superiority of the multi-chromosome representation and adaptive operators in term of computing time and criteria optimization performance [13].

4.4.2 Modular Locomotion System Design

Simultaneous structure/control optimization of robotic system for locomotion over rough terrains has been adressed through Genetic Algorithms. This method is based on a modular approach to the system design. The objective in the design process is to find an optimal mechanical architecture and an associated control to achieve displacements over complex surfaces. For this, we use an evolutionary algorithm integrating a dynamic simulation of the robotic system in its environment as illustrated on figure 4.2.

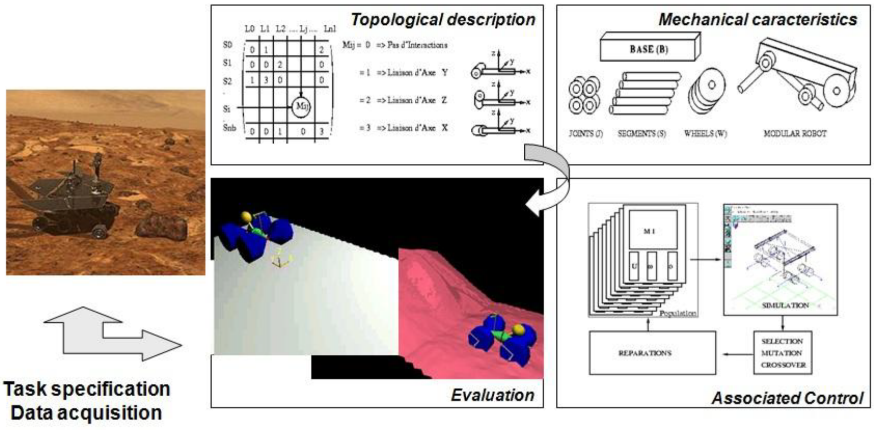


Fig. 4.2 Representation of the general framework developed for structure/command evolving of modular locomotion systems.

We used a hybrid encoding which allows for a simultaneous evolution of the mechanical structure and its control system. Specialized genetic operators have been designed to manipulate this encoding and to adapt their action to the evolving population. The robot performances are evaluated through a simulation in which all performances related to obstacle clearance capacities, stability of the system, speed and power consumption can be computed and exploited in real time. Hierarchical evaluation is then suggested for the improvement of computing time and the process is illustrated through a set of design examples

Based on a similar framework, the design of these micro-robots is based on a serial arrangement of articulated rings with associated antagonist SMA micro-actuators, whose configuration has to be adapted to the surgical operation constraints. The control strategies for an adaptation of the system geometry to the environment are based on a multi-agent approach to minimize the inter-module communication requirements. In this work, a technique based on Genetic Programming is proposed to generate approximated evaluation functions. Its aim is to significantly speed the design process up, while leading to robust evaluation. A specific adaptation of these principles has been investigated for the design of hyper-redundant robotic systems such as smart active endoscopes intended for minimally invasive surgery [14].

4.4.3 Inverse Model Synthesis

We have developed an original use of Evolutionary Algorithms in order to approximate by a closed form the inverse kinematic model (IKM) of analytical, non-analytical and general (i.e. with an arbitrary geometry) manipulators. The objective is to provide a fast and general solution to the inverse kinematic problem when it is extensively evaluated as in design processes of manipulators. A mathematical function is evolved through Genetic Programming according to the known direct kinematic model to determine an analytical expression which approximates the joint variable solution for a given end-effector configuration. Several implementations of this evolutionary symbolic regression process have been made to obtain approximations of the inverse kinematic models of the PUMA and the GMF Arc Mate robots as well as for general 6R manipulators with arbitrary geometry [15].

4.4.4 Multi-objective Task Based Design of Redundant Systems

The question of the optimization was approached by considering in particular problems posed by the search for kinematics and associated kinematic control adapted to the evolution of manipulation systems into constrained environments. In this aim we look, on the basis of a modular design, for mechanisms capable of realizing desired trajectories while satisfying the geometrical constraints related to the environment as well as the physical constraints relative to actuators. The control which has been considered here was a kinematic Cartesian control with active obstacles/constraints avoidance based on a gradient function projected into the null-space of the trajectory following task. The particular research carried out refers to the optimal design of high-mobility surgical instruments dedicated to minimally-invasive surgery. Its design is obtained by a generic evolutionary optimization process using pareto-based multi objective genetic algorithms (the evaluation was typically incorporating

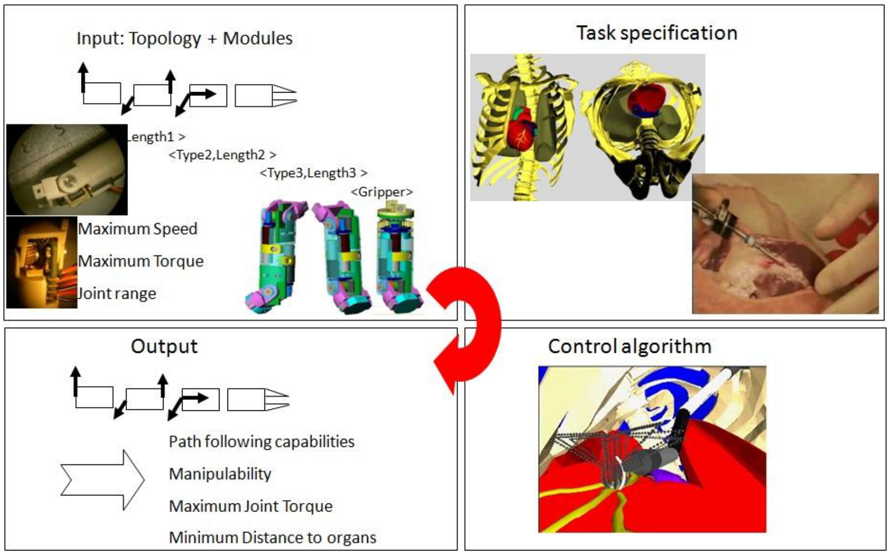


Fig. 4.3 Principles of the design process.

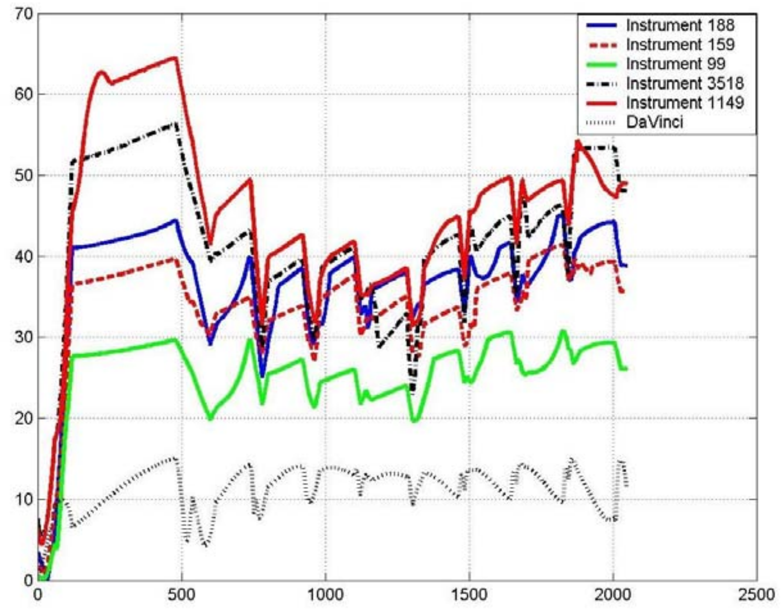


Fig. 4.4 Dexterity measure along the surgical trajectory.

diverse path tracking performances and force transmission, distance to obstacles, etc) and including highly realist simulations and experimental models of the surgical gesture. This design methodology has been applied to a coronary artery bypass grafting procedure. It results in a robotized surgical instruments made of 9 degrees of freedom whose performances in terms of dexterity along the prescribed trajectory was radically different from the one used on the well-known Da Vinci robot [17].

4.4.5 Flexible Building Block Design of Compliant Mechanisms

The design of compliant mechanisms has also been investigated. Compliant mechanisms are single-body, elastic continua flexible structures that deliver the desired motion by undergoing elastic deformation as opposed to jointed rigid body motions of conventional mechanisms. The methods for structural design of such systems can be divided into the homogenization methods and its variants the flexible building blocks method and the level set methods.

The flexible building blocks method which has been developed in [16] considers a compliant mechanism as an assembly of compliant building blocks. A multi-objective genetic algorithm is used to optimize the blocks distribution in the design space as well as the optimal set of structural parameters and material. Fixed node positions can also be considered as an optimization parameter. This topological optimization uses a genetic algorithm approach which allows true multicriteria optimization and the use of discrete variables. It has been applied notably to systems such as grippers for which the mechanical advantage may change in between the open and the close configurations. Figure 4.5 gives an example of results for a two finger gripper design.

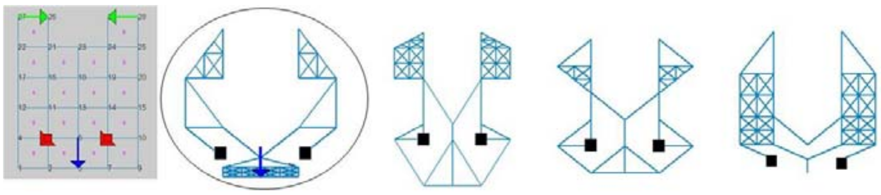


Fig. 4.5 Grippers designed with an evolutionary algorithm. (left) design domain and specifications (right) solutions obtained optimally satisfying the input/output transmission.

4.5 Conclusion

Evolutionary optimization methods are of general interest in the design of robotic systems for global and dynamic optimization in huge search spaces and problems with difficult constraints. Several implementations have been proposed in the last decade but numbers of extensions may be considered. One of the key question to be addressed in future research developments is related to the common assumption in design optimization which is that fitness functions are defined in advance. However, this is rarely true in practical cases, especially for conceptual design, where the fitness function may change or co-evolve with the generation.

References

1. Finger, S., Dixon, J.R.: A review of research in mechanical engineering design. Part I: Descriptive, prescriptive, and computer-based models of design processes. Research in Engineering Design 1(1), 51–67 (1989)
2. Thurston, D.L., Crawford, C.A.: A method for integrating end-user preferences for design. Journal of mechanical design, ASME Tansactions 116 (1994)
3. Orora, J.S., Huang, M.W., Hsieh, C.C.: Methods for Optimization of nonlinear problems with discrete variables: a review. In: Structural Optimization, vol. 8, pp. 69–85. Springer, Heidelberg (1994)
4. P.: Y Papalambros, Optimal design of Mechanical Engineering Systems. Journal of Mechanical Design, Trans. of ASME 117 (June 1995)
5. Mavroidis, C., Roth, B.: Method to determine uncertain configurations of 6R manipulators. In: Proceedings of the 9th World Congress on the Theory of Machines and Mechanisms (IFToMM), Milano, Italy, vol. 3, pp. 1987–1992 (1995)
6. Tsai, M.J., Lee, H.W.: Generalized evaluation for the transmission performance of mechanisms. Mechanism and Machine Theory 29(4), 607–618 (1994)
7. Amar, F.B., Bidaud, P.: Generic differential kinematic modeling of articulated mobile robots. Journal of Mechanism and Machine Theory 45(7), 997–1012 (2010)
8. Khatib, O.: Inertial properties of robotics manipulators, an object-level framework. The International Journal of Robotics Research 13(1) (1995)
9. Szewczyk, J., Plumet, F., Bidaud, P.: Planning and Controlling Cooperating Robots through Distributed Impedance. Journal of Robotic Systems 19(6), 283–297 (2002)
10. Grand, C., Benamar, F., Plumet, F., Bidaud, P.: Stability and traction optimization of a reconfigurable wheel-legged robot. Int. Journal of Robotic Research 23(10-11) (2004)
11. Amar, F.B., Jarrault, P., Bidaud, P., Grand, C.: Analysis and optimization of obstacle clearance of articulated rovers. In: IEEE/IROS Int. Conf. on Robots and Intelligents Systems, Saint-Louis, USA (2009)
12. Haschke, R., Steil, J.J., Steuwer, I., Ritter, H.: Task-oriented quality measures for dextrous grasping. In: In Proc. IEEE Conference on Computational Intelligence in Robotics and Automation, Espoo (2005)
13. Chocron, O., Bidaud, P.: Evolutionnary Algorithms in kinematic design of robotic systems. In: IEEE/IROS Int. Conf. on Robots and Intelligents Systems, Grenoble (1997)
14. Chapelle, F., Bidaud, P.: Evaluation functions synthesis for optimal design of hyper-redundant robotic systems. Mechanism and machine theory 41(10), 1196–1212 (2006)
15. Chapelle, F., Bidaud, P.: Closed form solutions for inverse kinematics approximation of general 6R manipulators. Mechanism and Machine Theory 39(3), 323–338 (2004)
16. Bernardoni, P., Bidaud, P., Gosselin, F.: A new compliant mechanism design methodology based on flexible building blocks. In: Proceedings of Smart Structure 2004, San Diego, USA (2005)
17. Sallé, D., Bidaud, P., Morel, G.: Optimal Design of High Dexterity Modular MIS Instrument for Coronary Artery Bypass Grafting. In: ICRA 2004, pp. 1276–1281 (2004)

Part III
Regular Contributions

Chapter 5
Evolving Monolithic Robot Controllers through Incremental Shaping

Joshua E. Auerbach and Josh C. Bongard

Abstract. Evolutionary robotics has been shown to be an effective technique for generating robot behaviors that are difficult to derive analytically from the robot's mechanics and task environment. Moreover, augmenting evolutionary algorithms with environmental scaffolding via an incremental shaping method makes it possible to evolve controllers for complex tasks that would otherwise be infeasible. In this paper we present a summary of two recent publications in the evolutionary robotics literature demonstrating how these methods can be used to evolve robot controllers for non-trivial tasks, what the obstacles are in evolving controllers in this way, and present a novel research question that can be investigated under this framework.

5.1 Introduction

What gives rise to intelligent behavior in natural and artificial agents? If you ask proponents of embodied artificial intelligence they will argue that such intelligent behavior arises out of the coupled dynamics between an agent's body, brain and environment [1, 6, 9, 17]. An extension of this idea is that the complexity of an agents's controller and morphology must match the complexity of the task or tasks that it is required to perform. However, when extending this idea to more complex agents in more complex environments it is not clear how to distribute responsibility for different behaviors across the agents's controller and morphology. Some have argued [8, 10] that controllers should be organized in a modular fashion such that different control components are responsible for different behaviors, but is this modularity

Joshua E. Auerbach
Morphology, Evolution and Cognition Laboratory, Department of Computer Science, University of Vermont, Burlington, VT 05405, USA
e-mail: joshua.auerbach@uvm.edu

Josh C. Bongard
Morphology, Evolution and Cognition Laboratory, Department of Computer Science, University of Vermont, Burlington, VT 05405, USA
e-mail: josh.bongard@uvm.edu

necessary? Recent work by our group and others has demonstrated that in fact, no, structural modularity is not always necessary [2, 3, 7, 14]. An example of how a monolithic (non-modular) controller can be evolved to enable a virtual autonomous robot to perform a non-trivial sequence of behaviors will be presented in the next section.

Besides modularity in the design of an agent's controller, roboticists often implicitly design their robots to have morphological modularity as well: different parts of the robot's body are responsible for different behaviors. For example, wheels or legs may allow for movement while a separate gripper module allows for object manipulation. But, what if this assumption is relaxed? In another recent publication [2], we demonstrated how a robot could be trained to locomote to and manipulate an object while the assumption of specialization of different body parts is relaxed: the robot had a segmented body plan in which the front segment was able to participate in locomotion and object manipulation, or it might have specialized such that it only participated in object manipulation. In this way, selection pressure dictated the presence and degree of specialization of the robot's morphology rather than enforcing such specialization *a priori*. Section 5.3 summarizes this work and discusses some of the insight gained from studying the variability observed in the degree of specialization of evolved controllers across different experimental regimes.

5.2 Learning Multiple Behaviors with a Monolithic Controller

Evolutionary robotics [12, 16] has been shown to be an effective technique for generating robot behaviors that are difficult to derive analytically from the robot's mechanics and task environment. In particular, such techniques are useful for realizing dynamic behaviors (eg. [13, 18]) in which individual motor commands combine in a nonlinear fashion to produce behavior, thereby making analytical derivations of optimal controllers infeasible. However, evolutionary algorithms alone are usually insufficient for evolving controllers capable of multiple dynamic behaviors. One method of augmenting evolutionary algorithms to achieve such controllers is incremental shaping ([11],[19] and [20]): the gradual complexification of an agent's task environment, also known in the developmental psychology literature as scaffolding [21], in order to first train controllers capable of performing a simplified version of a given task and then over time increase the task difficulty.

In a recent publication [3] we showed how using an incremental shaping technique makes it possible to train a virtual autonomous robot to overcome three learning milestones: object manipulation, dynamic forward legged locomotion toward an object, and directed legged locomotion toward an object, all using a single monolithic controller. Moreover, that work demonstrated the necessity of choosing an appropriate shaping trajectory or scaffolding schedule opening up several questions about how to choose such a schedule.

Specifically, two virtual quadruped robots (see Fig. 5.1) simulated in a physically realistic simulation engine[1] were experimented with. Both robots had a desired task

[1] Open Dynamics Engine: www.ode.org

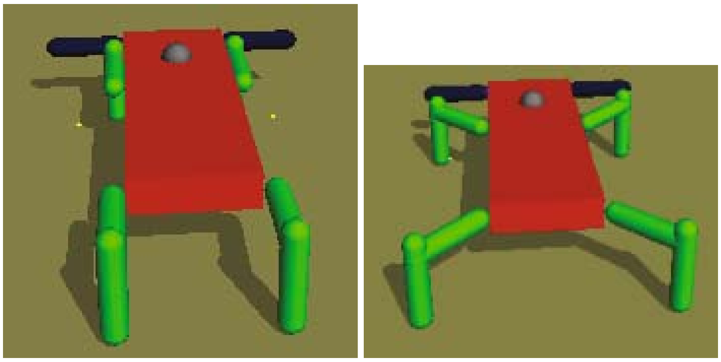


Fig. 5.1 The two virtual robots used in [3]: **Robot 1** (left), **Robot 2** (right).

of locomoting toward a distantly located object, grasping the object and then lifting the object onto their back. These robots each had 13 degrees of freedom and were actuated by a form of artificial neural network known as a continuous time recurrent neural network (CTRNN)[5]. For more details about the robots' morphologies and neural controllers please refer to [3].

In all cases training began in an environment where the target object was placed directly in front of the robot. Through a form of genetic algorithm (a hill climber) the CTRNN parameters were optimized until the robot was capable of grasping and lifting the target object. At this point the optimization process was paused and the environment was altered such that the target object was moved slightly further away from the robot. The optimization process then resumed until the robot was capable of reaching the target object at its new location, followed by grasping it and lifting it. After each such success, the process was paused, the environment was altered to make the task more challening, and then optimization was resumed. While this general process was the same for all experiments performed, what varied was the ways in which the target object was repositioned, known as the scaffolding schedule.

Specifically, four scaffolding schedules were investigated (see Fig. 5.2). The first scaffolding schedule, referred to as '**T**', placed the target object in front of the robot at increasing distances until the target object was a distance of three meters from the robot. It was observed that by this distance, the robot must have learned a stable gait to reach the target object. As distance was increased past three meters the target object was moved out in both directions along the line perpendicular to the robot's sagittal plane, requiring two sub-evaluations: one sub-evaluation with the target object placed in front and to the left, and another in which the target object is placed in front and to the right of the robot. This schedule forced the robot to learn forward locomotion with object manipulation followed by directed locomotion with object manipulation.

The second schedule used ('**C**') moved the target concurrently along the perimeter of circles with radius 5 meters and centers located at 5 and -5 meters with respect to the robot's initial position. In this case two sub-evaluations were always used. The final two schedules both moved the target object away from the robot linearly

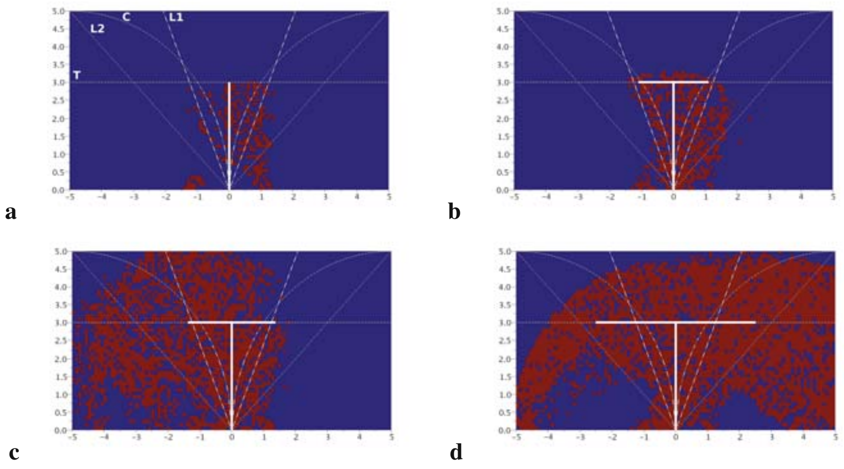


Fig. 5.2 Sample generalization plots from evolution of a generalized controller on robot 2 (red indicates the robot was successful at picking up the target object at that location) with the four scaffolding schedules superimposed. Specifically the plots shown are for controllers that were successful at distances of 3 meters (**a**), 3.2 meters (**b**), 3.3 meters (**c**) and 3.92 (**d**) the final training distance reached in this run.

on both sides. One did so with a slope $m = 1/\tan(22.5°)$ (‘**L1**’) and the other did so with a slope $m = 1/\tan(45°) = 1$ (‘**L2**’). All three of these schedules, to varying degrees, forced the robot to learn to turn towards the target before or while learning locomotion.

After completion of a given training experiment two metrics were used to evaluate success. The first was the adaptation rate: how far from the robot the shaping algorithm moved the target object during training. Since the target object was only moved further away when a controller was found to be successful at the previous distance this metric gave an indication of how rapidly the robot could adapt to a changing environment. However, it did not measure how successful a given CTRNN would be in unseen environments. For this purpose a second metric was devised. Know as a generalization metric, this metric involved creating a grid extending from 5 meters left to 5 meters right of the robot’s initial position and forward 5 meters, and systematically testing how well a given controller performed the task for a sampling of target object locations within this grid located at regular intervals. The fraction of these locations that the robot instantiated with this controller could succesfully complete the task would be the controller’s generalization score.

Figure 5.3 depicts the mean and standard error score achieved on both of these metrics across 100 independent runs per robot per scaffolding schedule. Notice how the **T** scaffolding schedule significantly outperformed the other three schedules both in training distance achieved and generalization for both robots. Comparing performances between robots, it is noted that the **T** schedule evolved significantly more generalized controllers with the second robot (left hand grouping in Fig. 5.3b) while

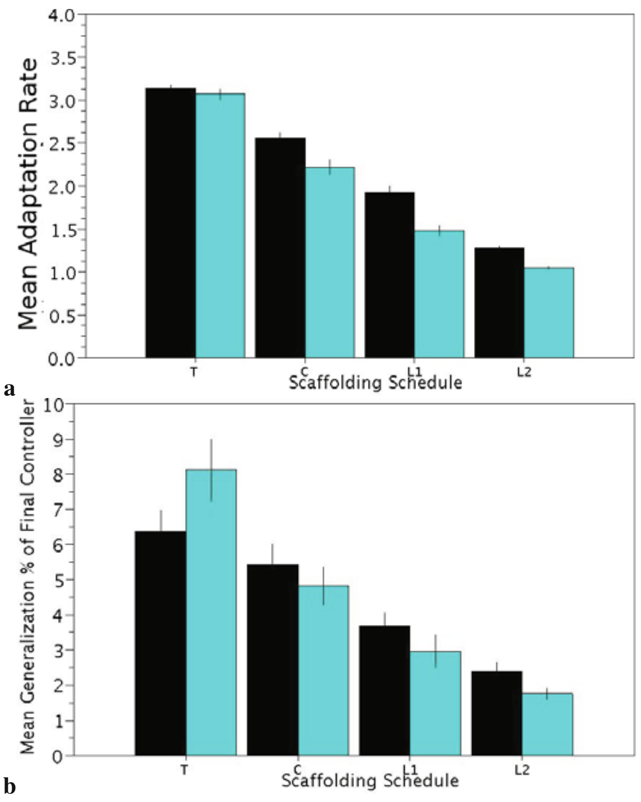


Fig. 5.3 Mean adaptation rate (**a**) and mean generalization % of final CTRNN (**b**) across the 100 runs for each of the two virtual robots (robot 1 in black, robot 2 in blue) and each of the four scaffolding schedules. All plots include standard error bars. Notice that while the mean generalization score for each set of runs was under 10% in all instances, there were runs in each set that found controllers with much higher generalization values. The generalization scores for the final controllers from the top five runs from each set are given in Table 5.1.

reaching similar final training distances as the first robot (left hand grouping in Fig. 5.3a). While the relative performance of the four schedules remained consistent across robots, the three other schedules led to slightly less generalized controllers with the second robot (three right hand groupings in Fig. 5.3b).

This means that both the morphology and training order are important for training a robot capable of completing the given task. Note that the schedules that pressured the robot to learn turning toward the target object either before or while learning to locomote were less successful than the one that pressured the robot to learn to locomote first. It therefore can be said that forward locomotion should be learned before turning, for both robot morphologies. As can be seen in Fig. 5.3 the probability of training a controller to enable taxis and object manipulation is inversely proportional to the pressure to learn turning before locomotion: the **T**, **C**, **L1**, and **L2** schedules

Table 5.1 Five best generalization values of final controllers from each set.

Schedule:	**T**	**C**	**L1**	**L2**
Robot 1:	53.6%	32.5%	23.3%	13.2%
	20.2%	28.3%	19.7%	12.7%
	16.6%	24.7%	14.9%	9.7%
	15.2%	24.3%	13.2%	9.2%
	15.1%	22.7%	11.5%	9.0%
Robot 2:	57.7%	26.3%	24.7%	12.6%
	40.4%	24.8%	24.1%	8.9%
	28.4%	21.4%	21.9%	7.6%
	27.4%	19.3%	19.6%	5.8%
	26.4%	19.1%	13.5%	4.8%

decline in performance, but increase in the pressure they exert to learn turning before locomotion.

This work demonstrated that with the proper scaffolding schedule (**T**) it is possible to evolve controllers capable of performing a non-trivial sequence of behaviors even in previously unseen environments. Moreover it demonstrated that altering morphology can impact the performance achievable through incremental shaping: robot 2 resulted in more generalized behaviors than robot 1. However, for the two morphologies considered the sequence in which behaviors should be learned remained the same. Robot 2's splayed legs made turning easier (see [3] for a discussion of this), however scaffolding schedules that selected for turning before locomotion was learned were not better able to integrate object manipulation, turning and locomotion into a controller using this body plan. Therefore it is concluded that the task environment, the learning algorithm, and/or the evolvability of CTRNNs dictate learning sequence more than morphology does.

More work remains to be done to strengthen this conclusion. Does this result hold across additional, uninvestigated, morphologies? How would evolving the robot's body plan along with its controller effect the sensitivity of the training procedure to the order in which behaviors are learned. The intuition is that evolving morphology would reduce this dependency and yield a more scalable method for realizing multiple dynamic behaviors in intelligent agents.

5.3 Specialization in a Morphologically Homogeneous Robot

Another recent publication [2] used a similar experimental framework as the work just discussed to investigate a different problem. In this case the research question was not about the order in which the behaviors should be learned, but about what variables influence the frequency of finding functionally specialized controllers – that is, controllers that devoted part of the robot's body (it's front legs) to a single behavior (object manipulation) rather than using that body part for multiple

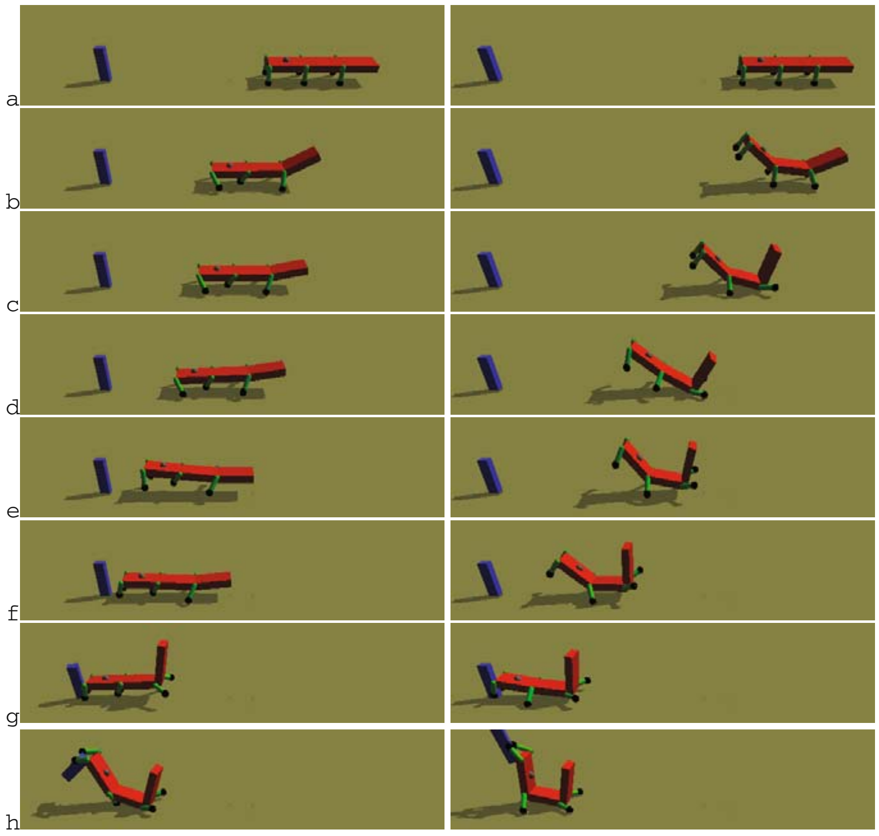


Fig. 5.4 **Left** Sample functionally generalized controller. This controller used the robot's front legs for propulsion during locomotion and for grasping and lifting of the target object. **Right** Sample functionally specialized controller. The robot's front body segment was raised and the front feet are kept off the ground during locomotion, i.e. they were only used for grasping the target object.

behaviors. Specifically the virtual robot investigated (Fig. 5.4) was a hexapod composed of three homogenous segments. It was designed such that the front segment could participate in locomotion and object manipulation (Fig. 5.4, left), or it may have become specialized such that it only participated in object manipulation (Fig. 5.4, right). In this way, selection pressure dictated the presence and degree of functional specialization rather than enforcing such specialization *a priori*.

This robot, like those described in the previous section, was trained with an incremental shaping algorithm coupled to a hill climber. Additionally, like those robots, this robot was controlled by a CTRNN. Several different experimental regimes were investigated with different initial environmental conditions and robot sensor configurations aimed at biasing the search process towards different solutions. For example, the first regime started with the robot's front segment rotated upwards 90° such that it was perpendicular to the ground with the front feet pointing forward and the target object initially placed directly in front of the robot. This configuration intuitively should have biased the evolutionary process towards finding controllers that specialized the front legs for grasping, since there was initially no evolutionary pressure for them to participate in locomotion, and indeed many of the runs from this regime found specialized controllers.

A second regime, conversely, started with the robot having all 6 feet on the ground. It was thought that this would bias the search toward controllers that did incorporate their front legs into their locomotion strategy, but this turned out to not be the case: a similar number of runs from this regime as compared to the first found controllers that specialized the front legs for object manipulation. An additional experiment began with the target object moved two meters in front of the robot, here it was thought that this would provide further bias towards incorporating the front legs into the locomotion strategy since, while learning to locomote initially there was no pressure for the front legs to be used for anything else, but once again a similar number of specialized controllers was found as compared with the previous two regimes.

Finally, a fourth regime with the same initial environmental conditions as the second regime, but with two additional sensors added to the robot and wired to its controller: joint angle sensors for the two joints connecting the body segments. The controllers that evolved in this regime not only performed better in the sense that they adapted more rapidly to changes in the target object's position during training as compared to the second regime, but also were more likely to be functionally specialized when compared to the other three regimes (blue bars in Fig. 5.5).

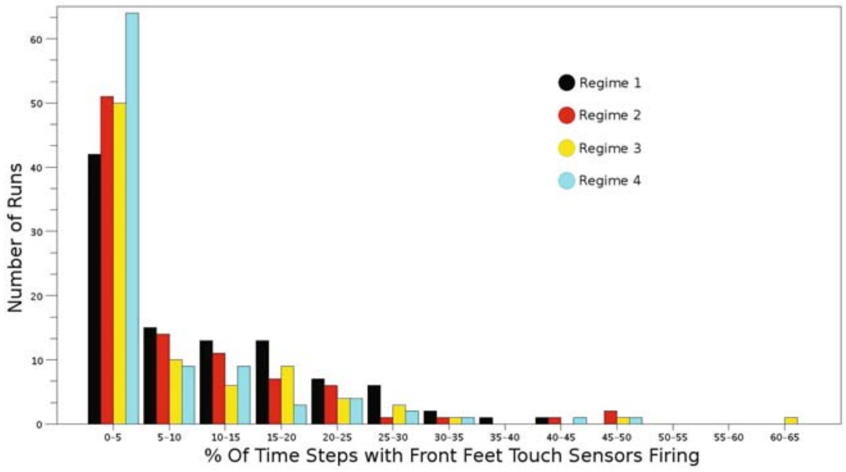


Fig. 5.5 Histogram of a specialization metric for each of the four regimes. All runs in which the target object reached at least three meters are included. See [2] for a description of this metric.

After noting that all four regimes were able to successfully learn both locomotion and object manipulation in the majority of trials the question arises as to why evolution tended to converge on functionally specialized behaviors, and why the inclusion of additional sensors caused an increase in the frequency of converging on such behaviors. Three possible hypotheses are: (1) functionally specialized controllers are more evolvable, and therefore supplanted less specialized controllers during an evolutionary run, (2) evolution initially discovered a specialized or generalized controller, and subsequently improved on that behavior but did not increase or decrease specialization, and (3) functionally specialized behaviors more easily allow for active perception [15].

While the first two hypothesis seem to be quite plausible, both were invalidated in [2]. The remainder of the space here will be spent discussing the more likely and potentially more interesting hypothesis number 3. According to that hypothesis, it may be that the robot was better able to actively perceive the proximity of the object—and therefore determine desirable conditions for lifting—if the front legs did not participate in locomotion, because then the touch sensors would only fire when in contact with the target object. Indeed, it has been demonstrated in the literature that active categorical perception may evolve in learning agents [4]. Moreover, providing the robot with additional proprioceptive feedback in regime 4 not only increased the prevalence of functional specialization (as shown in Fig. 5.5), but also the adaptation rate within those runs that produced specialized controllers. It is plausible that these added sensors allowed for better active perception as the touch sensors and sensed body posture may have together indicated appropriate conditions for object manipulation.

Several additional experiments were designed to test this hypothesis. These experiments followed the theme of the second and fourth regimes: fixing the initial environmental conditions but varying the sensors that the robot was provided with. It was demonstrated that adaptation rate declined as the included sensors provided less information in regards to desirable conditions for lifting. Specifically it was found that the main body joints were the most informative, while the front leg angles provided some information about the relative position of the front feet, but as the sensors are moved toward the rear of the body less of this relevant information would be available, and so the adaptation rate declined. This point was further demonstrated by an experiment that included joint angle sensors on every single joint. In this case the adaptation rate was not substantially improved compared with just including the most useful pair (those on the main body segments). Additionally it was shown in a further experiment that additional touch sensors improve performance even more so than any angle sensors do, because touch sensors provided the most direct evidence as to which feet are on the ground and/or touching the target object.

To verify that the additional sensors provided relevant information useful for the task and did not merely aid in locomotion, virtual robots were instantiated with the same sensor configurations and were evolved for locomotion alone. This consisted of expanding the range of the robot's distance sensors and placing the target object a large (100 m) distance away. Fitness was calculated as the fraction of distance

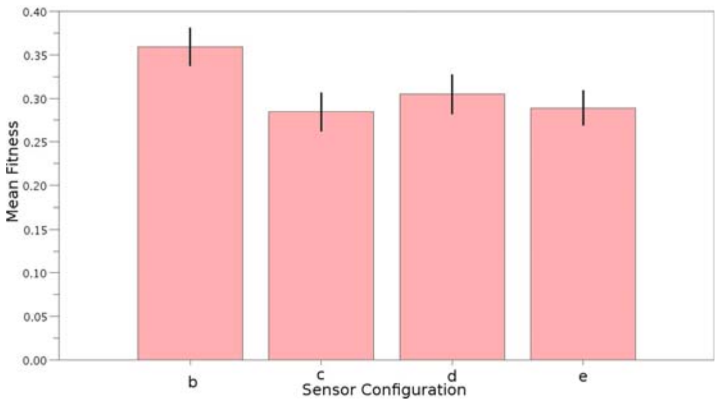


Fig. 5.6 Mean fitness with standard errors when selecting for just locomotion with four different pairs of joint angle sensors: **(b)** joint angle sensors on inter-segmental joints, **(c)** front leg joint angle sensors, **(d)** middle leg joint angle sensors, and **(e)** rear leg joint angle sensors.

between the start location and the target object location that the robot was able to cover in a set amount of time. Fig. 5.6 shows the mean fitnesses along with standard error bars from these experiments grouped by sensor configuration. Note that while including the joint angle sensors on the joints connecting the main body segments (**b**) led to improved locomotion performance, there was no significant difference between the performance of the other three sensor sets. This provided further evidence that the differences observed across these configurations above were due to active perception.

In conclusion, it was shown here that evolution can tune the amount of functional specialization of different parts of the body. It is predicted that if the morphology as well as the controller of the robot were under evolutionary control evolution would then specialize both the morphology and function for different body parts as the task environment dictates. Future work will test this prediction by evolving morphology as well as control. The hope is that this will prove to be a more fruitful method for realizing robots capable of an increasing number of behaviors, rather than fixing the body plan and manually assigning function to structure.

References

1. Anderson, M.: Embodied Cognition: A field guide. Artificial Intelligence 149(1), 91–130 (2003)
2. Auerbach, J., Bongard, J.C.: Evolution of functional specialization in a morphologically homogeneous robot. In: Proceedings of the Genetic and Evolutionary Computation Conference, GECCO (2009)
3. Auerbach, J., Bongard, J.C.: How robot morphology and training order affect the learning of multiple behaviors. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC (2009)

4. Beer, R.: The Dynamics of Active Categorical Perception in an Evolved Model Agent. Adaptive Behavior 11(4), 209 (2003)
5. Beer, R.D.: Parameter space structure of continuous-time recurrent neural networks. Neural Comp. 18(12), 3009–3051 (2006)
6. Beer, R.D.: The dynamics of brain-body-environment systems: A status report. In: Calvo, P., Gomila, A. (eds.) Handbook of Cognitive Science: An Embodied Approach, pp. 99–120. Elsevier, Amsterdam (2008)
7. Bongard, J.: Behavior chaining: incremental behavioral integration for evolutionary robotics. In: Bullock, S., Noble, J., Watson, R., Bedau, M.A. (eds.) Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, pp. 64–71. MIT Press, Cambridge (2008)
8. Brooks, R.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, [legacy, pre - 1988] 2(1), 14–23 (1986), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1087032
9. Brooks, R.: Cambrian intelligence. MIT Press, Cambridge (1999)
10. Calabretta, R., Nolfi, S., Parisi, D., Wagner, G.P.: Duplication of modules facilitates the evolution of functional specialization. Artificial Life 6(1), 69–84 (2000)
11. Dorigo, M., Colombetti, M.: Robot shaping: Developing situated agents through learning. Artificial Intelligence 70(2), 321–370 (1994)
12. Harvey, I., Husbands, P., Cliff, D., Thompson, A., Jakobi, N.: Evolutionary robotics: the sussex approach. Robotics and Autonomous Systems 20, 205–224 (1997)
13. Hornby, G., Takamura, S., Yamamoto, T., Fujita, M.: Autonomous evolution of dynamic gaits with two quadruped robots. IEEE Transactions on Robotics 21(3), 402–410 (2005), doi:10.1109/TRO.2004.839222
14. Izquierdo, E., Buhrmann, T.: Analysis of a dynamical recurrent neural network evolved for two qualitatively different tasks: walking and chemotaxis. In: Bullock, S., Noble, J., Watson, R., Bedau, M.A. (eds.) Artificial Life XI: Proceedings of the Eleventh International Conference on the Simulation and Synthesis of Living Systems, pp. 257–264. MIT Press, Cambridge (2008)
15. Noë, A.: Action in Perception. MIT Press, Cambridge (2004)
16. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology,Intelligence,and Technology. MIT Press, Cambridge (2000)
17. Pfeifer, R., Bongard, J.: How the Body Shapes the Way We Think: A New View of Intelligence. MIT Press, Cambridge (2006)
18. Reil, T., Husbands, P.: Evolution of central pattern generators for bipedal walking in a real-time physics environment. IEEE Transactions on Evolutionary Computation 6(2), 159–168 (2002), doi:10.1109/4235.996015
19. Saksida, L., Raymond, S., Touretzky, D.S.: Shaping robot behavior using principles from instrumental conditioning. Robotics and Autonomous Systems 22, 231–249 (1997)
20. Singh, S.P.: Transfer of learning across sequential tasks. Machine Learning 8, 323–339 (1992)
21. Wood, D., Bruner, J., Ross, G.: The role of tutoring in problem solving. J. Child Psychol. Psychiatry 17(2), 89–100 (1976)

Chapter 6
Evolutionary Algorithms to Analyse and Design a Controller for a Flapping Wings Aircraft

Stéphane Doncieux and Mohamed Hamdaoui

Abstract. Evolutionary Algorithms are now mature optimization tools, especially in a multi-objective context. This ability is used here to help explore, analyse and, on this basis, propose a controller for a complex robotics system: a flapping wings aircraft. A multi-objective optimization is performed to find the best parameters of sinusoidal wings kinematics. Multi-objective algorithms generate a set of trade-off solutions instead of a single solution. The feedback is then potentially more informative in a multi-objective context relative to the one of a single objective setup: the set of trade-off solutions can be analyzed to characterize the studied system. Such an approach is applied to study a simulated flapping wing aircraft. The speed-energy relation is empirically evaluated and the analysis of the relations between the parameters of the kinematics and speed has led, in a further step, to the synthesis of an open-loop controller allowing to change speed during flight.

6.1 Introduction

Evolutionary Algorithms (EA) nowadays belong to the classical toolbox of engineers as powerful optimization algorithms [5]. They differ from other techniques in the fact that they are black box, derivative free optimization algorithms. The function to be optimized, i.e. the cost function, also called the fitness function, may even not be known but just evaluated through a dedicated experimental device. This makes such tools valuable for non-linear and dynamical systems design and first

Stéphane Doncieux
ISIR Pierre and Marie Curie University, CNRS Pyramide Tour 55 Boite courrier 173 4, place Jussieu 75252 Paris cedex 05 France
e-mail: stephane.doncieux@isir.upmc.fr

Mohamed Hamdaoui
MAS Ecole centrale Paris, Grande Voie des Vignes Chatenay-Malabry, France
e-mail: mhcsdl@gmail.com

applications have actually been performed directly on a real device [20, 21] as cited by [1].

Furthermore, as they are population-based, they can easily be adapted to a multi-objective context [6] and efficient multi-objective evolutionary algorithms (MOEA) now exists, as for instance NSGA-II [8] or ε-MOEA [7]. Actually, most real world problems involve multiple objectives to be optimized: cost and efficiency, for instance. When these objectives are antagonistic, there is no single optimal solution, but rather a set of optimal trade-off solutions. MOEA generate at once an approximation of this set of the best trade-off solutions whereas most other multi-objective algorithms only discover one trade-off solution at a time or require to know in advance some features of this set [16]. Examples of applications of MOEA are then numerous [3, 4, 22].

The cloud of optimal trade-off solutions generated by a multi-objective algorithm can be used to extract useful data on the considered problem. Actually, all these points share the specificity of being optimal relative to some user defined objectives and are arbitrarily close. They materialize then the possible values of the objectives and the relations between them and the evolved parameters. An expert of the field can exploit these data to design new and innovative solutions that go beyond the solutions handled by the MOEA. Deb and Srinivasan [9] suggested a systematic approach to these questions and called it *innovization*, for *innov*ation through optim*ization*. Their approach consists in first finding the set of Pareto-optimal solutions according to problem specific objectives. On the basis of these data, an analysis is made to identify regularities and relations between parameters. In this article, such an approach is applied to a simulated flapping-wings aircraft. The energy-speed relation is empirically evaluated and a simple open-loop speed controller is deduced from the analysis of the relations between the parameters and the speed.

Many studies have been performed by biologists that have observed natural devices of this kind. Thanks to statistical studies on flying animals or insects, biologists have identified relations between significant parameters like wing area, cruising speed, wing span, flapping wings frequency, wing load or mass [23]. Wing kinematics have also been studied for several species. Tobalske and Dial have observed, for instance, that pigeons and magpies use a relatively constant flapping frequency across their complete range of speed, i.e. 4 to $14 m.s-1$ [25]. Meanwhile, Park et al. have studied the swallow in a wind tunnel and observed an U-shaped relation between frequency and speed [18], concluding that such relations may vary upon bird species. Physicists also try to unravel the physical mechanisms underlying the flapping flight. Some experiments have been conducted in which prototypes with a given kinematics were put in a wind tunnel to study and characterize their behavior [11]. In this case, the kinematics is given, but if we turn to a roboticist's point of view, the question becomes : for a prototype with known features, what are the interesting kinematics ? To our knowledge, no theory nor methodology do exist today to answer such practical questions. Likewise, no efficient flapping wing aircraft do exist together with its controller to help infer relations between the involved parameters. Birds or bats are sources of inspiration, but equivalent engineered flyers have fewer degrees-of-freedom and different aerodynamical features. Biological observations can't then be directly used. The proposed methodology aims at facilitating the design of such aircrafts by providing some simple yet efficient solutions that can be analyzed to better understand such systems before building more sophisticated solutions.

Stochastic optimization tools are very interesting in the context of flapping wings aircrafts due to the complexity of the relation between the parameters of a particular kinematics and the resulting speed (or aircraft crash...) [17, 19, 24] . The work presented here is different in that it does not aim at generating a single and particular optimal controller, but rather at generating a set of them, that are not the goal *per se* but rather a mean to study properties of the system. [15] had a similar goal, but a small number of speeds were initially chosen in the study and runs were launched for each value. In the present work, a continuum of speeds is explicitely sought. It should be noticed that the following results have been obtained in simulation and are thus highly dependent on simulation accuracy. Actually, such experiments may be done on a real prototype.

6.2 Method

The proposed approach consists in first generating a set of Pareto-optimal points. These points are particular in the sense that they share a common specificity: they are all optimal relative to some (antagonistic) user-defined objectives. Finding the common features of those points corresponds then to find what characterizes Pareto-optimal points: what are the critical parameter values ? How are they related to objectives? Furthermore, the MOEA we will use, i.e. NSGA-II, aims at finding a *dense* approximation of the Pareto front. This means that the generated solutions will be arbitrarily close one to the other. This proximity between points will make the analysis of the solutions easier and results, generally, in a continuum between the features of these solutions. A more detailled description of the approach follows.

The first step consists in choosing two antagonistic objectives (or more). The objectives explicitely need to be antagonistic in order to guarantee the existence of a set of trade-off solutions, rather than a single optimum. Although it may seem at first sight to restrict the field of application, in practice, it is easy to find such objectives: cost or energy related objectives are, for instance, generally antagonistic to performance related objectives (efficiency relative to the task to be solved, for instance).

Once these objectives are known, a search space has to be choosen. The search space defines the set of candidate solutions the EA will explore. All further results and analysis will be relative to this search space, its choice must then be done carefully.

The next step is straightforward: launch the optimization to find a good approximation of the Pareto front. Deb and Srinivasan [9] suggest to do it in several different steps: perform a standard multiobjective search, perform a single objective optimization to find the extremum points of the Pareto front and then use NCM [16] to find a set of points representing an uniformly distributed sampling of the Pareto front. All these different steps only aim at providing a better confidence in the quality of the Pareto front approximation. In most real-world applications, there is no way to know it with precision, it is then important to do whatever possible to increase the confidence in the quality of the results, as the quality of the further analysis will critically depend on this part of the process. In the work reported here, several independant runs have been launched and also single objective optimizations have been tried for both extremum points and Pareto front points. As the points generated by single objective algorithms were clearly dominated by the points generated by NSGA-II[1], the focus in the following is only on the points generated by this last MOEA.

The cloud of Pareto-optimal points can then be used to extract useful information just like in [9]. The Pareto front in the objectives space shows the performance of the best solutions to be found relative to the given search space. Each parameter can also be plotted against a target objective or parameter and analytical functions approximating this relation can be found by regression, as it is done in [9]. But this can become cumbersome as the dimensionality of the Pareto front increases in the objectives or parameters spaces. For example, with n objectives and p parameters all the couples with the $n+p$ quantities have to be studied, which result in $\frac{(n+p-1)(n+p)}{2}$ plots to scrutinize. Looking at these plots individually can dramatically increase the time needed to extract the information from the Pareto front and may sometimes lead to inconclusive results. That is why specific visualization methods that can easily handle high dimensional data and can give at a glance a global view of the Pareto front in the objectives and/or parameters spaces have to be used to detect the groups of quantities that seem to be related and only for which, plots should be drawn.

In our case, the Pareto front is of dimension 8 in the parameters space and 2 in the objectives space. Therefore, visualizing the relationships between parameters and/or objectives on the Pareto front can become a lengthy and laborious process ($\sim$ 45 plots to review). Kohonen networks [12] are used here to facilitate this step. These networks are a nonlinear topology preserving projection method which allows to handle comfortably high dimensional data. The objective of Kohonen networks is to form a discrete, topological mapping of a Q-dimensional input space. The algorithm outputs a Kohonen map composed of output units often arranged in a 2-D rectangular or hexagonal grid, each ouput unit being characterized by its 2-D vector of coordinates and a Q-dimensional vector called codebook. To construct the mapping, the network is trained on a finite dataset of Q-dimensional vectors (input space) and produces a Kohonen map whose units are 2-D vectors (output space) in such a way that the topology of the data is preserved and that each vector of the input Q-dimensional dataset is associated to a unit on the map via its codebook. This unit is called *BMU* or best matching unit. Thus, Kohonen networks build a tractable low dimensional (the Kohonen map) representation of high dimensional datasets allowing a comfortable visualization of the data. A MATLAB® implementation

[1] We tried a simple rank-based EA.

of the method is freely provided by the SOM Toolbox[2]. This technique is used to visualize the Pareto front taking into account, for each Pareto-optimal solution, both parameters and objectives.

In the case of parameters describing a robot controller, this knowledge extraction step results in the ability to build a new and more complex controller relative to what EA has explored and generated : if the parameters are modified online using the found relationships we have a controller able to move along the Pareto front and thus to adapt to the context. Such an approach requires that such online modification is possible and efficient, what is not guaranteed, as the MOEA only explored constant parameters controllers. Anyway, the proposed new controller may be the subject of further refinements through another innovization step or through a simple optimization of its parameters concentrated on the transition phase between changes of parameters.

6.3 Experimental Setup

To illustrate potential use of the method, a flapping wings controller able to adapt the speed of the aircraft to a given desired speed is designed while exploiting the data extracted from the Pareto-optimal set analysis. The principle of the controller consists in modifying online the parameters of the wings kinematics. The applied set of parameter values for the wing kinematics are the closest possible to those of Pareto-optimal points associated with a similar speed. This requires to extract the relationships between the speed and the different parameters of the kinematics and this is what the MOEA analysis is used for.

A cloud of Pareto-optimal points that correspond to different speeds is generated and analysed, using Kohonen maps, in both objectives and parameters spaces to estimate qualitatively the relationships between the different quantities at hands. This qualitative study will ease the identification of the parameters that noticeably vary with the velocity while allowing us to spot interesting regions of the Pareto front. Then, by doing some regression on the withheld kinematical parameters versus the velocity, a law of variation for each parameter relative to speed will be inferred and used to drive the online wing kinematic parameter settings.

Tests will then be performed in simulation to see if it actually allows to change the speed of the flapping wings aircraft. Starting from a simple family of functions, i.e. sinusoidal functions, allowing to fly at a constant speed, a controller able to adapt the speed of the aircraft will be designed[3].

The first goal is to generate the set of pareto-optimal points to be studied. Such points should represent a varying speed: speed will be the first objective. An antagonistic objective is required to generate a set of individuals and not a single point. Energy will be used to this end. Two optimizations will be performed: one will try to maximize speed, while the other one will try to minimize it. The largest range of

[2] `http://www.cis.hut.fi/somtoolbox/`

[3] This will still be an open-loop controller, as it won't take into account real speed, we will propose an extension to design a closed-loop controller in the discussion.

speeds is required to make the speed control interesting. In the case of an U-shaped relation between speed and energy, two kinds of experiments need to be launched:

- speed minimization will materialize the lowest speed points
- speed maximization will materialize the highest speed points

The points generated by the two experiments are then merged to generate the cloud of points to study. Wing kinematics are described by the following equations:

$$DI = a_{DI}\sin(2\pi t/p_{DI}) \quad (6.1)$$
$$TWi = r_{TWi} + a_{TWi}\sin(2\pi(t/p_{DI} + p_{TWi})) \quad (6.2)$$
$$TWe = r_{TWe} + a_{TWe}\sin(2\pi(t/p_{DI} + p_{TWe})) \quad (6.3)$$

where DI is the wing dihedral, TWi the internal twist and TWe the external twist. Wing kinematics is then described by eight parameters subject to the evolutionary optimization. The chosen ranges for each value is the following:

- amplitude of the dihedral: a_{DI} in $[0;45^o]$
- period of the dihedral (and of all other degrees of freedom): p_{DI} in $[0.2;1s]$
- reference of the internal twist: r_{TWi} in $[-22.5;22.5^o]$
- amplitude of the internal twist: a_{TWi} in $[0;45^o]$
- phase of the internal twist: p_{TWi} in $[0;1]$
- reference of the external twist: r_{TWe} in $[-22.5;22.5^o]$
- amplitude of the external twist: a_{TWe} in $[0;45^o]$
- phase of the external twist: p_{TWe} in $[0;1]$

NSGA-II is used to perform the search, with a population size of 500 and during 1000 generations. Evolved parameters are represented as vectors of real values with a polynomial mutation and a sbx crossover, as described in [6], p124.

Aerodynamical forces created by wing movements are computed with a semi-empiric, quasi-steady-aerodynamics model. Each wing is decomposed in three panels. For each panel, the local incident airspeed is evaluated and the leading edge lift, the parachute drag and the friction drag are computed and summed. See [10, 15] for a detailed description of the model. The parameters of the aircraft are given in Appendix. The panels are considered as not deformable solids, connected to their neighbors via joints, as shown on figure 6.1. The integration of the forces and the movements of the parts are computed using ODE[4].

The objectives to maximize are the following:

- experiment 1:
  - + average aircraft speed
  - - average mechanical power
- experiment 2:
  - - average aircraft speed
  - - average mechanical power

[4] `http://ode.org`

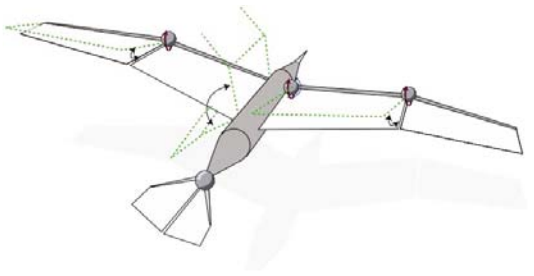


Fig. 6.1 Degrees-of-freedom of the simulated flapping wing aircraft.

Following [15], the instantaneous mechanical power is computed as the scalar product between the instantaneous torque (τ) and the instantaneous rotational speed (ω) for a joint: $P = \sum_i |\tau_i.\omega_i|$. The mechanical power objective only takes into account the shoulder joints which are the main contributors to energy consumption. It should be noted that the power is always considered as positive. The mechanical power is then over estimated, as the torques required to accelerate or to slow down are considered as equivalent.

6.4 Results

Three different runs have been launched for experiment 1 and experiment 2. Each run shows a similar Pareto front, except for points at highest energy. This result is not surprising as these individuals are at the limits of the simulation we used: small changes may have a huge consequence on the stability of the simulation, thus making such part of the search space difficult to explore.

For each experiment, the set of non dominated solutions out of the three runs is extracted and the two resulting sets are merged (figure 6.2). Dots follow an U-shape, with a minimal energy consumption of 24W at $10.7 m.s^{-1}$. It can be noticed that the sets of non dominated solutions for experiment 1 and experiment 2 coincide for this point of minimal energy consumption with no particular discontinuity.

A Kohonen map of the Pareto front in both objectives and parameters spaces is built. Multiple representations of this map, colored according to the objectives and kinematic parameters, are provided on Figures 6.3-6.5. For example, the Figure 6.3 top represents the map colored by the values of the U-matrix with the color code specified on a colorbar next to the map. The U-matrix is a matrix of distances between the units of the map computed in the Q-dimensional input space: as each unit of the map has a Q-dimensional codebook, the distance between units in the Q-dimensional space is the distance (euclidian for example) between the Q-dimensional codebook vectors associated to these units. To quantify the relative importance of the different components of the Q-dimensional codebook vector in the U-matrix computation, some components may be set to zero while the others are taken into account in computation of the distance to assess their importance for

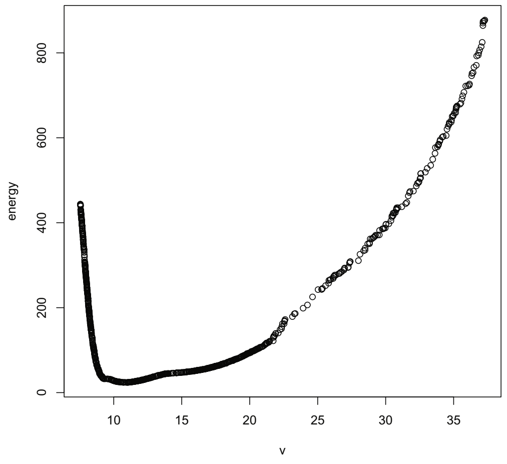


Fig. 6.2 Pareto-optimal points extracted from the results of the three speed optimization and the three speed minimization experiments. Each dot represents the optimal solution found by the EA for a given speed relative to the energy consumption. X-axis: speed in $m.s^{-1}$, Y-axis: instantaneous energy in W.

the U-matrix. The U-matrix allows to spot groups of similar solutions (areas on the map where the U-matrix takes low values). The mapping error is quantified by computing the average distance between each vector in the data and its BMU. The error associated with the built mapping is of 8 %.

Four main similar areas or clusters can be seen on the map colored by the values of the U-matrix (see Figure 6.3, top). Three of them (Cluster 1-3) are located in the region of low velocities and the other one (Cluster 4) is in the region of high velocities as can be seen by comparing Figure 6.3, top and Figure 6.3, middle. The separation line between Cluster 1-3, in one hand, and Cluster 4, in the other hand, seems to correspond to the separation between the two merged Pareto fronts (one obtained for high velocities and the other for low velocities). To make sure of that we computed an U-matrix based on the values of the velocity only, represented on Figure 6.3, bottom, left, which shows that this separation line is strongly related to the velocity.

The internal twist amplitude a_{TWi} (see Figure 6.3, bottom, right) does not change much on the map as it takes high values for very low velocities and low values the rest of the time. This means that this kinematic parameter is not sensitive to the velocity except for very low velocities.

On the opposite, the internal twist reference r_{TWi}, internal twist phase p_{TWi}, external twist amplitude a_{TWe}, external twist reference r_{TWe} (see Figure 6.4) and external twist phase p_{TWe} (see Figure 6.5, top, left) are related to the velocity on the map. Indeed, the separation line between Cluster 1-3, in one hand, and Cluster 4, on the other hand appears on the Kohonen maps colored by the values of r_{TWi}, p_{TWi}, a_{TWe}, r_{TWe} or p_{TWe}. As this separation line is closely related to the evolution of the

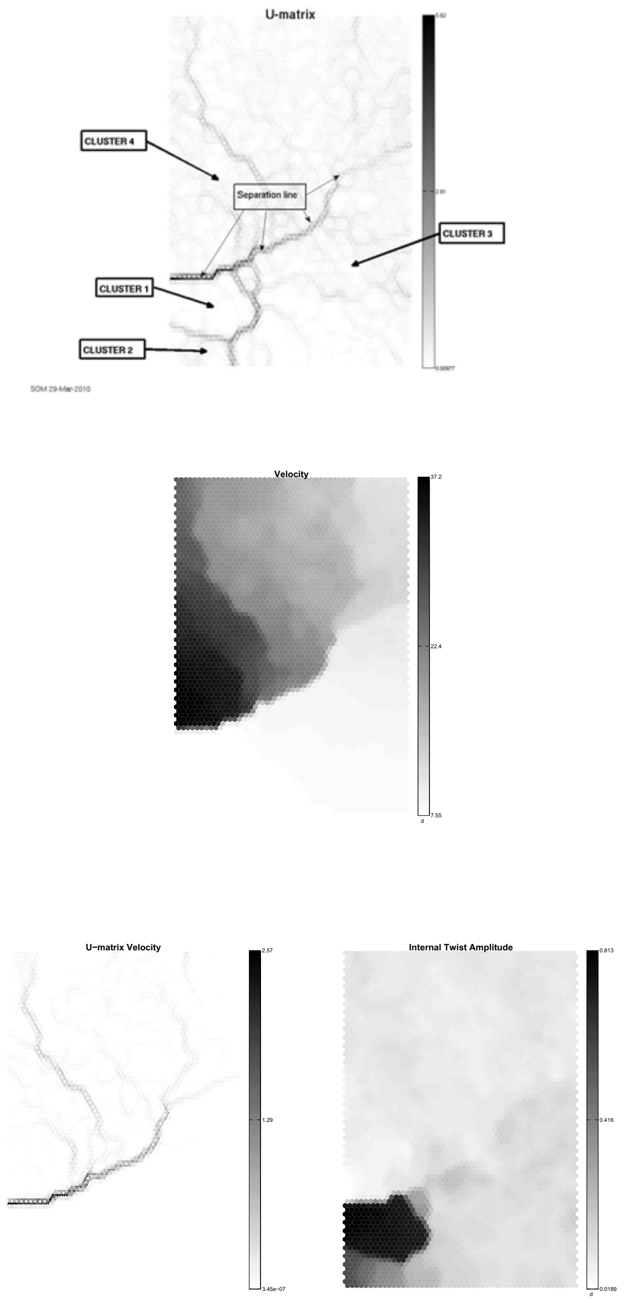


Fig. 6.3 Top, : the U-matrix and the four clusters. Middle: the Kohonen map colored by the velocity values. Bottom, left: the U-matrix based on the velocity only. Bottom, right: the Kohonen map colored by the Internal Twist Amplitude a_{TWi}.

velocity as it appears on the Kohonen map colored by velocity, this means that these kinematic parameters are closely related to the velocity.

This separation line appears too on the Kohonen map colored by the values of the energy (see Figure 6.5, top, right), which means that the energy is related to the velocity too. On the opposite we do not see this separation line for the dihedral period p_{DI} (see Figure 6.5, bottom, left) or the dihedral amplitude a_{DI} (see Figure 6.5, bottom, right), which means that these two kinematical parameters might not be related to the velocity, what can be confirmed by a plot of these two kinematic parameters against the velocity.

Finally, building a Kohonen map allowed us to know that the internal twist amplitude a_{TWi} does not strongly depend on the velocity (except for very low velocities) and that the other kinematic parameters except the dihedral period p_{DI} and the dihedral amplitude a_{DI} clearly depend on the velocity.

All the parameters of the sinusoidal wing kinematics can be plotted, for each Pareto optimal point, in order to check their relation to speed (figure 6.6).

The relation between each parameter and the speed has been approximated with polynomial regression methods. The corresponding relations are reported in Appendix. Resulting approximations are also plotted on figure 6.6.

We used then these approximated functions to pilot the parameters of the same simulated bird (figure 6.7). The control is oscillating, but as did the original controllers: the fitness did only measure the mean speed, not the ability of the controller to reduce its standard deviation. The control is not efficient at all for $8m.s^{-1}$, meaning that the approximation is not good for this speed. For other speeds, the simulated bird speed tends to oscillate but with a decreasing amplitude. For a speed of 10 or $11m.s^{-1}$ the bird altitude isn't perfectly maintained, resulting in an increasing speed error when the bird starts to dive. For each desired speed, the measured mean speeds are the following:

Desired speed	average speed	abs. error
8	10.15	27%
10	10.64	6.4%
11	11.23	2%
13	13.45	3.5%
15	15.39	2.6%
20	17.61	12%
25	33.94	36%

For speed ranging from 10 to $15m.s^{-1}$, the error is several percents, but it gets larger for small or high speeds. This is not surprising as the physical behavior is less stable in these cases.

We have also tried to change it online, i.e. during a flight. To avoid physically unrealistic behavior, we haven't abruptly changed the parameters, but we have waited for the wings' dihedral to pass near zero. Once a small angle is reached, we let the wings still and wait until the new kinematics also reaches a small angle and we

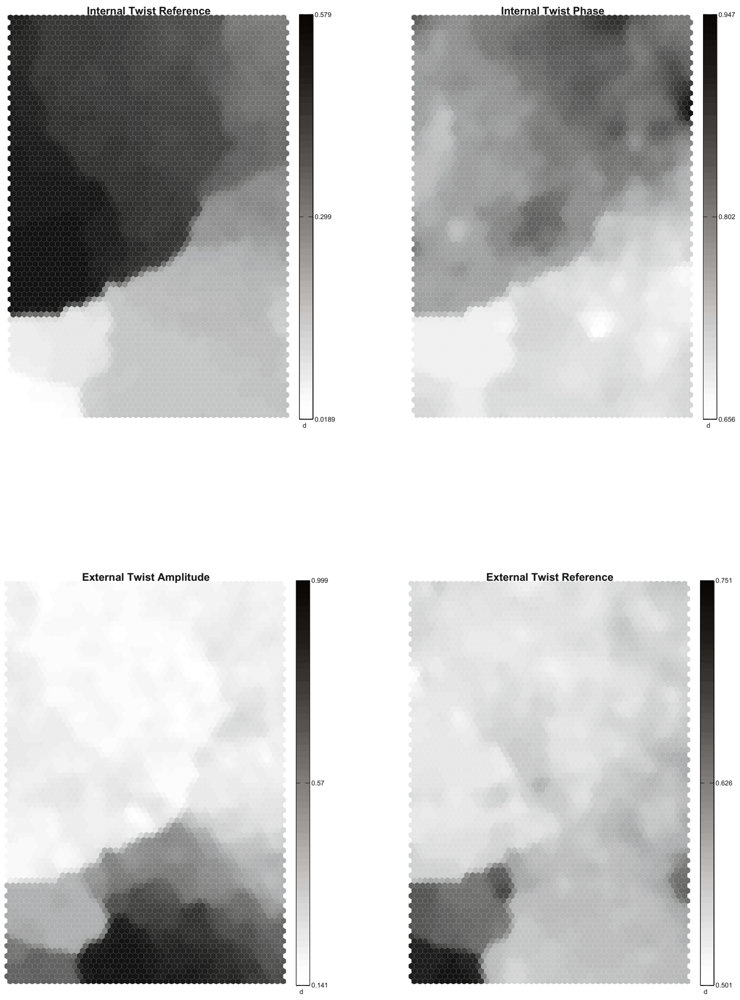


Fig. 6.4 Top, left: the Kohonen map colored by the Internal Twist Reference r_{TWi}. Top, right: the Kohonen map colored by the Internal Twist Phase p_{TWi}. Bottom, left: The Kohonen map colored by the External Twist Amplitude a_{TWe}. Bottom, right: the Kohonen map colored by the External Twist Reference r_{TWe}.

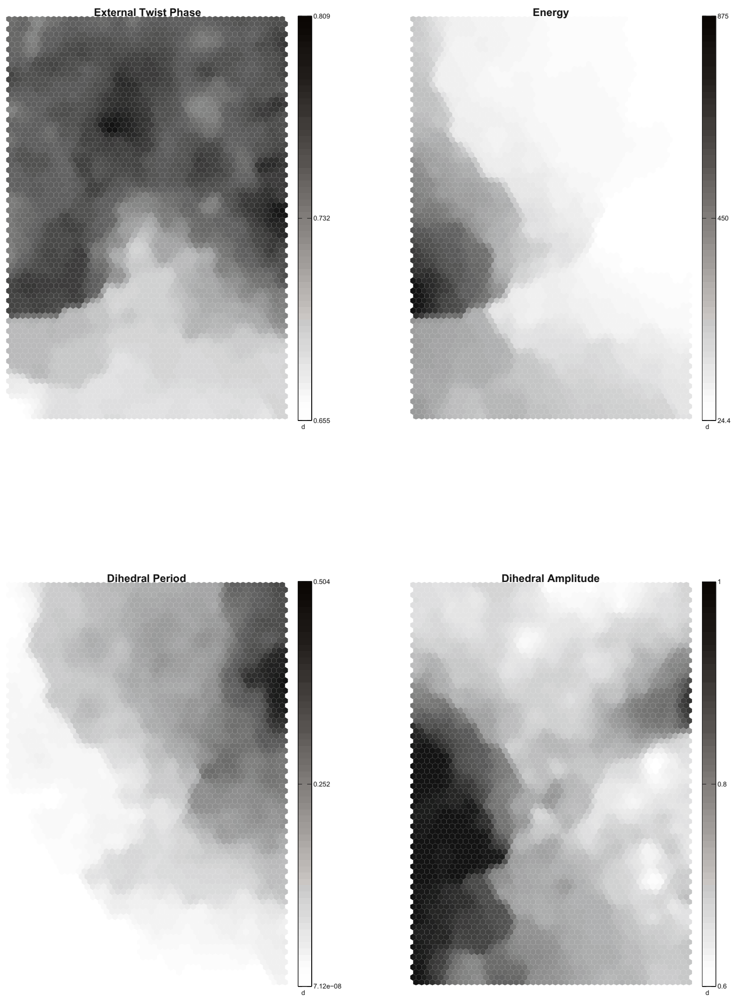
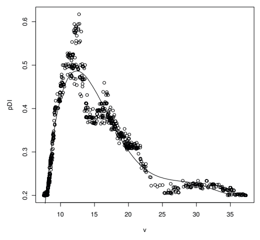
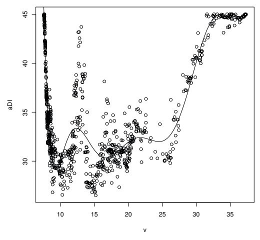


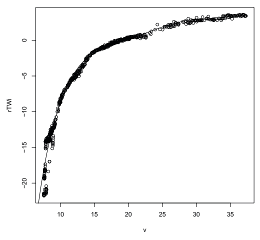
Fig. 6.5 Top, left: the Kohonen map colored by the External Twist Phase p_{TWe}. Top, right: the Kohonen map colored by the energy. Bottom, left: The Kohonen map colored by the Dihedral Period p_{DI}. Bottom, right: The Kohonen map colored by the Dihedral Amplitude a_{DI}.



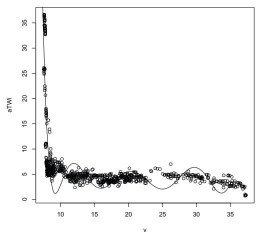
(a) dihedral period in *s*



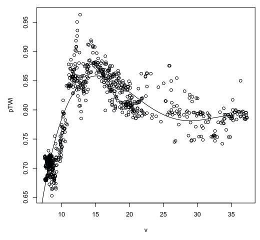
(b) dihedral amplitude in *deg*



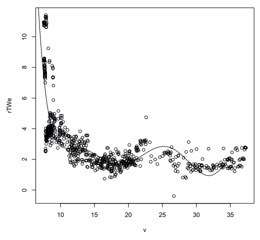
(c) internal twist reference in *deg*



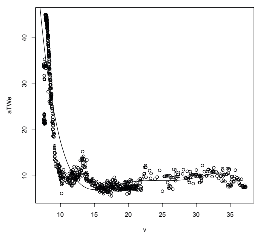
(d) internal twist amplitude in *deg*



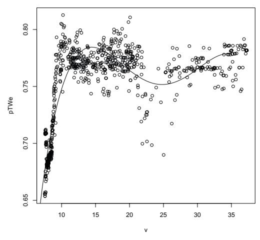
(e) internal twist phase



(f) external twist reference in *deg*



(g) external twist amplitude in *deg*



(h) external twist phase

Fig. 6.6 Empirical dependency of each parameter relative to speed. Plot of the repartition of Pareto-optimal points for each parameter. Each dot represents a pareto optimal solution. Solid lines represent the approximated fit with a polynomial relation whose parameters are given in Appendix, figure 6.9.

switch the kinematics only then. Results are reported on figure 6.8. What we observe is that the bird actually changes its speed dynamically, simply as a result of wing kinematics parameter change (open-loop modification). Actually, we observe a kind of undesired memory in the system: the control at a given speed may show different performances (different static errors) depending on the historical context, i.e. depending on the initial conditions when the switch is performed. During evolution, every individual started from a single condition: horizontal flight at $11 m.s^{-1}$; here, the 'starting' speed may be very different. Anyway the bird remains stable and at a relatively constant altitude (there is no active control at all: none on speed but none on altitude also).

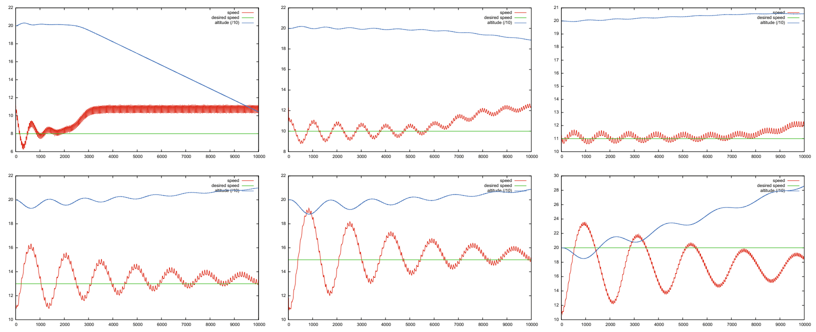


Fig. 6.7 Speed of the simulated bird controlled by a sinusoidal kinematics whose parameters are obtained from the regression at a given desired speed. The controller is an open-loop controller, real speed is not taken into account to finely tune it.

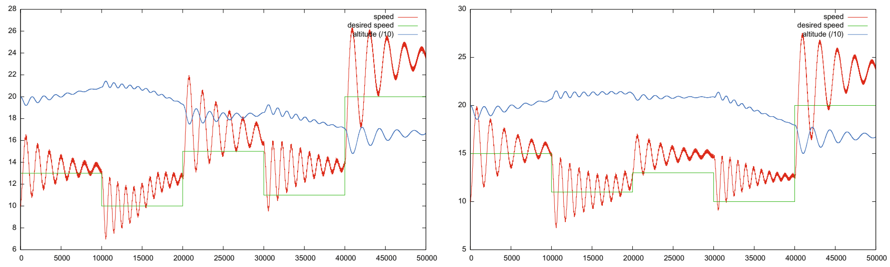


Fig. 6.8 Speed of the simulated bird controlled by a sinusoidal kinematics whose parameters are obtained from the regression at a given desired speed. The controller is an open-loop controller, real speed is not taken into account to finely tune it. The parameter of the wing kinematics are changed during flight.

6.5 Discussion and Future Work

All the search was performed on the basis of a simplified simulation. The conclusions are then only relative to the model that was used and are to be confirmed by experiments on a real device. Anyway, such an approach can be used directly on a real prototype. Its main drawback is then the high number of experiments required to find the Pareto front. Some solutions do exist to this problem. A simplified simulation may be used first to reject the most inefficient solutions while testing on the prototype only the most pertinent ones. The discrepancy between an experiment in simulation and reality may also be used to drive a model learning loop aimed at reducing the gap between the two [2, 13, 14].

The simple open loop controller that we have synthesized is clearly not optimal. An optimization trying to generate solutions more robust to the initial conditions might be required to avoid the observed memory effect. Another innovization step might also be launched to study the transition between different speeds, the compromise here being for instance between the speed of transition and its stability.

Likewise, starting from the generated open-loop controller, we could close the loop and optimize the parameter of a controller in which the speed error is added to the desired speed.

All this study is based on the features of Pareto optimal points found by the MOEA. The generated Pareto fronts correspond to performance to be expected with a simulated bird flying with sinusoidal wing kinematics. Changing the family of wing kinematics may change the shape of Pareto fronts. At least, using such an approach, we might empirically compare different kinds of kinematics and look at the advantage of using more generic periodic functions, for instance.

MOEA were used to generate the set of Pareto optimal solutions. Other methods could have been used. In many simple cases, an exhaustive search might do the job. Here, it would be difficult to use it as we have eight continuous parameters. If we discretize and consider only ten different values, then we have a 10^8 search space. A single MOEA run did around 200,000 different evaluations. As we did 6 different runs (3 for speed maximization and 3 for speed minimization), the total number of evaluations we made is then $1.2 * 10^6$, thus two orders of magnitude below an exhaustive search. Furthermore the search space explored by the MOEA was continuous and not a discretized one.

The main point here is that EA might be used for other purposes than mere optimization. Here, each generated point is not an interesting solution in itself, it was interesting as a mean to capture some regularities of the problem that have then been exploited manually. This suggests thus another use of EA as an exploration tool during the very first steps of a design process, whereas it is usually used at the very end, when there just remains several parameters to tune. This particular use requires the analysis of an expert and is not aimed at automatically designing a solution. It is rather a tool aimed at helping the engineer or the scientist to gain better insights about the problem to be solved.

6.6 Conclusions

In this work, we have exploited the ability of MOEA to generate a set of Pareto optimal points not just to discover such points and choose one among them, but rather to get some insights on the relationships between those points. We have used it to empirically evaluate the trade-off between speed and energy with a constant morphology.

The Pareto front has then ben visualized by Kohonen maps to spot the kinematic parameters that noticeably vary with the velocity. Each selected parameter have then been expressed as a function of speed thanks to a regression performed on Pareto-optimal solutions. An open-loop controller able to change speed along flight has been synthesized and tested in simulation.

Acknowledgments

This work has been supported by the DGA/D4S REI research grant #06.34.022.

Appendix

Parameters of the MOEA:

- MOEA: NSGA-II
- population size: 500
- number of generation: 1000
- number of independant run performed for each experimental context: 3
- mutation rate 0.1
- mutation type: polynomial, η_m: 15 and η_c: 10
- crossover: sbx

param	v^0	v^1	v^2	v^3	v^4	v^5	v^6	v^7	v^8	v^9
St_r	1.3e2	57.0	11	-1.1	6.5e-02	-2.3e-3	4.5e-5	3.7e-7	-	-
p_{DI}	-3.89e0	1.07e0	-9.68e-2	4.09e-3	-8.27e-5	6.45e-7	-	-	-	-
r_{TWi}	-1.16e2	2.44e1	-2.07	8.78e-2	-1.82e-3	1.47e-5	-	-	-	-
p_{TWi}	-2.89e-1	2.10e-1	-1.35e-2	3.58e-4	-3.39e-6	-	-	-	-	-
a_{TWi}	1.30e4	-6.60e3	1.44e3	-1.76e2	1.34e1	-6.57e-1	2.09e-2	-4.14e-4	4.66e-6	-2.28e-8
r_{TWe}	3.55e2	-1.38e2	2.24e1	-1.94e0	9.62e-2	-2.76e-3	4.22e-5	-2.68e-7	-	-
p_{TWe}	2.11e-2	1.49e-1	-1.04e-2	2.97e-4	-3.03e-6	-	-	-	-	-
a_{TWe}	3.00e2	-6.69e1	5.94e0	-2.55e-1	5.35e-3	-4.38e-5	-	-	-	-

Fig. 6.9 Parameters of the polynomials approximating the relations $parameter = f(v)$ for each parameter of wing kinematics submitted to optimization.

Parameters of the aircraft:

- wing span: $1.93m$
- aspect ratio: 8.5
- wing area: $0.407m^2$
- total mass: $1.3kg$
  - fuselage mass: $0.915kg$
  - wing mass: $0.4kg$
  - elevator/rudder: $0.038kg$

References

1. Back, T., Hoffmeister, F., Schwefel, H.: A survey of evolution strategies. In: Proceedings of the Fourth International Conference on Genetic Algorithms (1991)
2. Bongard, J., Lipson, H.: Nonlinear System Identification Using Coevolution of Models and Tests. IEEE Transactions on Evolutionary Computation 9(4), 361–384 (2005)
3. Coello-Coello, C., Lamont, G., Van Veldhuizen, D.: Evolutionary algorithms for solving multi-objective problems. Springer-Verlag New York Inc. (2007)
4. Coello-Coello, C., Lamont, G. (eds.): Applications of Multi-Objective Evolutionary Algorithms. World Scientific, Singapore (2004)
5. Dasgupta, D.Z.M. (ed.): Evolutionary Algorithms in Engineering Applications. Springer, Heidelberg (1997)
6. Deb, K.: Multi-objective optimization using evolutionay algorithms. Wiley, Chichester (2001)

7. Deb, K., Mohan, M., Mishra, S.: Evaluating the epsilon-Domination Based Multi-Objective Evolutionary Algorithm for a Quick Computation of Pareto-Optimal Solutions. Evolutionary Computatition 13(4), 501–525 (2005)
8. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
9. Deb, K., Srinivasan, A.: INNOVIZATION: Discovery of Innovative Design Principles Through Multiobjective Evolutionary Optimization. In: Multiobjective Problem Solving from Nature: From Concepts to Applications, p. 243 (2007)
10. Druot, T.: Technical report on the implementation and validation of a flight mechanics simulator for flapping articulated wings. Tech. rep. (2004)
11. Hubel, T., Tropea, C.: Experimental investigation of a mapping wing model. Experiments in Fluids 46, 945–961 (2009)
12. Kohonen, T.: Self-Organizing Maps. Springer series in information sciences, 3rd edn., vol. 30. Springer, Berlin (2001)
13. Koos, S., Mouret, J., Doncieux, S.: Automatic system identification based on coevolution of models and tests. In: Proceedings of IEEE Conference on Evolutionary Computation (2009)
14. Koos, S., Mouret, J.B., Doncieux, S.: Crossing the Reality Gap in Evolutionary Robotics by Promoting Transferable Controllers. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation, GECCO 2010. ACM, New York (2010)
15. de Margerie, E., Mouret, J.B., Doncieux, S., Meyer, J.A.: Artificial evolution of the morphology and kinematics in a flapping-wing mini UAV. Bioinspired and Biomimetics 2, 65–82 (2007)
16. Messac, A., Mattson, C.: Normal constraint method with guarantee of even representation of complete pareto frontier. AIAA journal 42(10), 2101–2111 (2004)
17. Mouret, J.B., Doncieux, S., Meyer, J.A.: Incremental evolution of target-following neuro-controllers for flapping-wing animats. In: Nolfi, S., Baldassare, G., Calabretta, R., Hallam, J., Marocco, D., Meyer, J.A., Miglino, O., Parisi, D. (eds.) From Animals to Animats: Proceedings of the 9th International Conference on the Simulation of Adaptive Behavior (SAB), Rome, Italy, pp. 606–618 (2006)
18. Park, K., Rosen, M., Hedenstrom, A.: Flight kinematics of the barn swallow (hirundo rustica) over a wide range of speeds in a wind tunnel. Journal of Experimenal Biology 204(15), 2741–2750 (2001)
19. Rakotomamonjy, T.: Modelisation et controle du vol d'un microdrone a ailes battantes. Ph.D. thesis, Université Paul Cezanne (2006)
20. Rechenberg, I., Eigen, M.: Evolutionsstrategie: Optimierung technischer systeme nach prinzipien der biologischen evolution (1973)
21. Schwefel, H.: Evolutionsstrategie und numerische Optimierung. TU Berlin, Germany (1975)
22. Tan, K., Khor, E., Lee, T.: Multiobjective evolutionary algorithms and applications. Springer, Heidelberg (2005)
23. Tennekes, H.: The simple science of flight (from insects to jumbo jets). MIT Press, Cambridge (1996)
24. Thomson, S., Mattson, C., Colton, M., Harston, S.P., Carlson, D., Culter, M.: Experiment-based optimization of flapping wings kinematics. In: Proceedings of the 47th Aerospace Sciences Meeting (2009)
25. Tobalske, B., Dial, K.: Flight kinematics of black-billed magpies and pigeons over a wide range of speeds. Journal of Experimenal Biology 199(2), 263–280 (1996)

Chapter 7
On Applying Neuroevolutionary Methods to Complex Robotic Tasks

Yohannes Kassahun, Jose de Gea, Jakob Schwendner, and Frank Kirchner

Abstract. In this paper, we describe possible methods of solving two problems encountered in evolutionary robotics, while applying neuroevolutionary methods to evolve controllers for complex robotic tasks. The first problem is the large number of evaluations required to obtain a solution. We propose that this problem can be addressed by accelerating neuroevolutionary methods using a Kalman filter. The second problem is the difficulty of obtaining a desirable solution that results from the difficulty of defining an appropriate fitness function for a complex robotic task. The solution towards this problem is to apply the principles of behavior based systems to decompose the solution space into smaller subsolutions with lower number of intrinsic dimensions, and incrementally modify the fitness function. We present two case studies towards the solutions to the stated problems.

7.1 Introduction

Designing parts of a control software that allows an already designed (existing) robot to accomplish a certain task through neuroevolution is difficult due to the complexity of the task environment. Two of the problems that one usually encounters are:

1. Very long evolution time (a large number of evaluations) until a working solution is found.
2. Difficulty of defining a fitness function that results in a desired solution.

The first problem is caused due to the fact that the task we want to solve is most of the time noisy (noisy sensors and actuators) and partially-observable in addition

Yohannes Kassahun · Frank Kirchner
Robotics Group, University of Bremen, Robert-Hooke-Str. 5, D-28359 Bremen, Germany

Jose de Gea · Jakob Schwendner · Frank Kirchner
German Research for Artificial Intelligence (DFKI), Robotics Innovation Center, Robert-Hooke-Str. 5, D-28359 Bremen, Germany

to being complex (high dimensional state and action spaces). Partially-observable problems have been a challenging domain for machine-learning algorithms. The fundamental reason for this is that such problems place limits on an agent's ability to fully perceive the states of the environment, and in doing so, limit the information upon which an agent can base its decisions. Neuroevolutionary methods have delivered promising results in recent years as methods of solving such learning tasks. Traditionally, neuroevolutionary methods solve partially-observable problems by using recurrent connections within the neural network, which provides a system with memory, enabling it to recover the missing state information. A major drawback of recurrent connections is the difficulty in training them, which means that they require a significant amount of training time to find a solution. The reason is that it takes a large portion of the evolution time for neuroevolutionary methods that evolve both topology and parameters of the neural networks to determine the topology of the recurrent neural network for a given problem that requires only a small amount of evolution time. Similarly, for a fixed topology neuroevolutionary methods, it is usually difficult for the domain expert to come up with a topology for the recurrent neural network that needs only a small amount of evolution time to find a solution. For neuroevolutionary methods to be effective for complex tasks, we proposed an alternative to using recurrent neural networks in neuroevolution for overcoming the effects of partially-observable domains. In the alternative solution we try to simplify the topology of the evolved neural network, and as a consequence, reduce the time required to find a solution. The approach exploits the use of a Kalman filter as an input layer for the neural network to be evolved. The Kalman filter inherently provides the system with memory (as recurrent connections would) to estimate and thus recover the unobserved missing state variables. From the viewpoint of the neural network to be evolved (whose inputs are the outputs of the Kalman filter layer), the state information has been augmented and is noise-free. Clearly, this additional information provided to the system permits a simpler neural network solution (see Section 7.2.3), thus significantly reducing the time required to find a solution. The work presented in the first case study (see Section 7.2) differs from other works [8, 18, 27, 28, 31] which use extended Kalman filter to train feed-forward or recurrent neural networks. In this work, the Kalman filter is an integral part of the solution, and its parameters are optimized simultaneously with the parameters of the feed-forward neural network. Thus, simultaneous optimization of parameters of the feed-forward neural network (topology, weights, etc) and parameters of the state estimator (Kalman filter) can be considered as a contribution of this work to the state-of-the-art in neuroevolution. Traditionally, the parameters of the state estimator are first determined, and then the parameters of the neural network representing the policy are learned.

The second problem is caused due to the fact that an evolutionary algorithm learning a complex robotic task on a robot having many degrees of freedom and many sensory inputs finds many solutions that are not desirable when using a fixed single fitness function. The problem with a single fitness function is that it may not reward intermediate solutions that would ultimately lead to the desired operating properties. A possible way to solve such a problem is to decompose the solution space into smaller subsolutions with lower number of intrinsic dimensions. One way of decomposing a complex robot control task into subsolutions is by using the design principles of behavior based systems [2, 5], and to incrementally modify the fitness function that results in desired operating properties as the subsolutions are learned. In addition to resulting in a desired operating properties, the use of behavior based systems solves the *scaling problem* of evolutionary algorithms in large state spaces. In the Hierarchical Reinforcement Learning (HRL) research community, methods of decomposing robot's task into a hierarchy of organized behaviors either manually or automatically have been investigated extensively [4]. The motivation behind the development of HRL algorithms is the poor scaling problem of standard reinforcement learning in large state spaces with sparse reinforcement. HRL approaches require the definition of a global reward function that should result in desired operating properties, and the definition of local reinforcement signals for learning elementary actions or behaviors composed of other behaviors and/or elementary actions. In contrast to the HRL methods, we present in this paper a principled way of incrementally modifying a single fitness function (return[1]) as new behaviors are learned and added to the system. An interesting example of the application of HRL on a real robot is the work of Kirchner [23], where the elementary swing and stance movements of individual legs and the overall coordination scheme to perform forward movements are learned. In the evolutionary robotics research community, efforts have been made to solve the scalability problems of evolutionary algorithms when applied to learning complex robot tasks. An earlier attempt is the work by Gruau and Quatramaran [15] for learning the locomotion pattern of an 8-legged robot using modular genetic neural networks. Later, Calabretta et al. [6] showed that modular architectures produce better results than non-modular architectures in terms of both speed of evolution and the final steady state behavioral performance. In both of the above works, a global fitness function is defined to learn a global behavior, which may not result in the desired performance properties of the final controller. Mouret and Doncieux [26] applied a pareto-based multi-objective evolutionary algorithm to evolve modular neural networks. They point out that modularity is a key for enabling a phenomenon called exaptation [11] in evolutionary algorithms. In a closely related work to ours, Lee et al. [24] use the concept of behavior based systems to design a controller capable of displaying multiple behaviors, and thereby ease the process of defining fitness functions that result in desired behaviors. In contrast to [24], we present a second case study (see Section 7.3) of incrementally modifying the fitness function as subsolutions are learned. Moreover, in this paper we show that while learning a given subsolution, the already learned subsolutions can remain active in the control so that we do not need extra training for learning the coordination process separately. In addition to this, unlike the work in [24], where digital circuits cabable of realizing the mapping between discrete state and actions are used as behavior modules, the work in this paper exploits a parameterized behavior module implemented in the form of an augmented neural network with Kalman filter (see Section 7.2) suitable for continuous state partially observable domains, which

[1] A return is defined as a function of the reward sequence and can be considered as the equivalent of a fitness function in evolutionary computation.

are common in robotics. A recently introduced method [3] of gradually incorporating new behaviors for a dynamically behaving robot uses dynamic scaffolding [32], where the robot's environment is restructured for incorporating a new behavior into a monolithic controller which can exhibit multiple attractor states. In this method a global fitness function is used, and a sequence of behaviors is mastered by a dynamical neural network. It is, however, difficult to train and use monolithic controllers, where multiple behaviors should be active at the same time to accomplish a given task (e.g Quadrocopter control).

7.2 Case Study 1: Augmented Neural Network with Kalman Filter (ANKF)

The augmented neural network with Kalman Filter (ANKF) to be evolved is made up of a neural network and a predictor that can estimate the next state based on the *current partially-observable state* (which is possibly corrupted by noise). The predictor we use is composed of n Kalman filters ($\alpha\beta$ filters, see Section 7.2.1) $\{KF_1, KF_2, \ldots, KF_n\}$ one for each of the n sensory readings, as shown in Figure 7.1. The outputs of these Kalman filters are connected to a feed-forward neural network NN, whose outputs control the plant. A Kalman filter KF_i is used to estimate the sensor value $\hat{x}_i$ and the missing value $\hat{\hat{x}}_i$ from the measured (observed) value z_i, where $i \in [1,n]$ and n is the number of observable state variables. The quantity u_j, where $j \in [1,m]$, represents a control signal that is used to control a plant. The use of Kalman filters provides memory to the system and as a result enables the system to recover missing variables. Because of this, it is not necessary for the neural network to have a recurrent connection, and the use of a feed-forward neural network for the

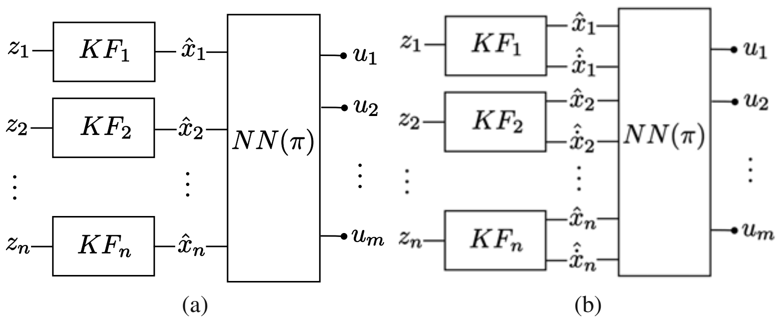


Fig. 7.1 Two ways of using the augmented neural network: (a) The Kalman filter KF_i is used to estimate the sensor value $\hat{x}_i$ from the measured (observed) value z_i in the case of complete state variables. (b) The Kalman filter is used to estimate the sensor value $\hat{x}_i$ and the missing value $\hat{\hat{x}}_i$ from the measured (observed) value z_i in the case of incomplete state variables. The quantity u_j represents the control signal that is sent to the plant to be controlled. NN is a feed-forward neural network representing a policy π.

policy π to be learned is sufficient. The whole controller is a non-linear function given by

$$u_j = f(w_1,\ldots,w_k,\gamma_1,\ldots,\gamma_n;z_1,\ldots,z_n,\hat{x}_1(0),\ldots,\hat{x}_n(0),\hat{\dot{x}}_1(0),\ldots,\hat{\dot{x}}_n(0)), \quad (7.1)$$

where u_j ($j \in [1,m]$) is one of the outputs of the feed-forward neural network, $w_1,\ldots,w_k$ are the weights of the neural network, $\gamma_1,\ldots,\gamma_n$ are the tracking indices (see Section 7.2.1) of the $\alpha\beta$ filters realizing $KF_1,\ldots KF_n$, $z_1,\ldots,z_n$ are the measured values of input sensors, and $\hat{x}_1(0),\ldots,\hat{x}_n(0)$ and $\hat{\dot{x}}_1(0),\ldots,\hat{\dot{x}}_n(0)$ are initial estimates of the state variables. The index k is greater than or equal to n ($k \geq n$) in the case of complete state variables, and it is greater than or equal to $2n$ ($k \geq 2n$) in the case of incomplete state variables[2]. Table 7.1 summarizes the usage of the augmented neural network for different task environments.

Table 7.1 Usage of the augmented neural network (ANKF) for different task environments.

	Noise-free environment	**Noisy environment**
Complete state variables	*Completely observable domain.* Use the setup shown in Figure 7.1 (a).	*Partially observable domain due to noise.* Use the setup shown in Figure 7.1 (a).
Incomplete state variables	*Partially observable domain due to missing state variables.* Use the setup shown in Figure 7.1 (b).	*Partially observable domain due to noise and missing state variables.* Use the setup shown in Figure 7.1 (b).

7.2.1 The $\alpha\beta$ Filter

A Kalman filter KF_i in Figure 7.1 is realized using an $\alpha\beta$ filter, which is a particular case of the general Kalman filter where the velocity is assumed to be constant. The filter is usually used in tracking applications. The neural network equivalent of the filter is shown in Figure 7.2. As can be seen in the figure, all the weights of the filter have a magnitude of 1 except for α, β and T, where T is th sampling period. The optimal values for α and β are derived by Kalata [20] for assumed variance of both measurement and process noises (σ_v and σ_w) and are given by $\alpha = 1-r^2$ and $\beta = 2(1-r)^2$ respectively, where $r = \frac{4+\gamma-\sqrt{8\gamma+\gamma^2}}{4}$ and $\gamma = \frac{T^2\sigma_w}{\sigma_v}$. The term γ is referred to as a *tracking index*. Since the parameters α and β depend only on γ, an optimization algorithm *needs only to find a single parameter* γ per filter that results in the desired filter performance. An extension of the $\alpha\beta$ filter is the $\alpha\beta\gamma$ filter [12], which is based on a constant acceleration model and is better suited for the tracking of complex signals. Like the $\alpha\beta$ filter, an optimization algorithm needs only to find

[2] By incomplete state variables we mean a set of state variables whose first order derivative with respect to time are missing.

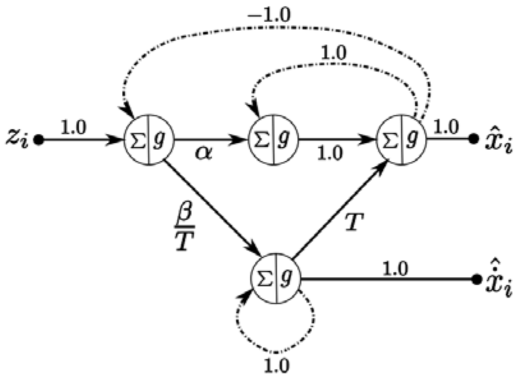


Fig. 7.2 An $\alpha\beta$ filter for a single input z_i. The activation function g is the identity function, and the recurrent connections have a unit delay.

a single tracking index γ per filter to get the desired filter performance. This enables one filter to be easily exchanged with the other.

7.2.2 *Evolving ANKF*

There are three cases to consider while evolving ANKF: completely observable domains, noise-free partially observable domains and noisy partially observable domains.

7.2.2.1 Completely Observable Domains

Normally for completely observable domains, the use of feed-forward neural networks is sufficient. But if we want to apply the augmented neural network to such domains, we have two possible ways of optimizing the parameters of the augmented neural network: (a) We can set the tracking index γ_i of each of the $\alpha\beta$ filters to a higher value and optimize only the parameters of the feed-forward neural network. We found out that a tracking index that results in $\alpha_i \geq 0.95$ performs well under noise-free scenario. Figure 7.3 shows the performance of an $\alpha\beta$ filter in noise-free scenario for two different values of γ. (b) We can optimize the tracking indices of the $\alpha\beta$ filters and the parameters of the feed-forward neural network simultaneously. Note that for learning tasks in completely observable domains, an optimization algorithm using the augmented neural network has to optimize n more variables (parameters of the tracking indices of the $\alpha\beta$ filters) than an algorithm optimizing only the parameters of a feed-forward neural network, whose structure is the same as the structure of the feed-forward neural network used in the augmented neural network. The quantity n is the number of inputs to the augmented neural network.

7.2.2.2 Noise-Free Partially Observable Domains

In noise-free partially observable domains, there are some state variables that are missing. The purpose of the Kalman filters in this case is simply to estimate the

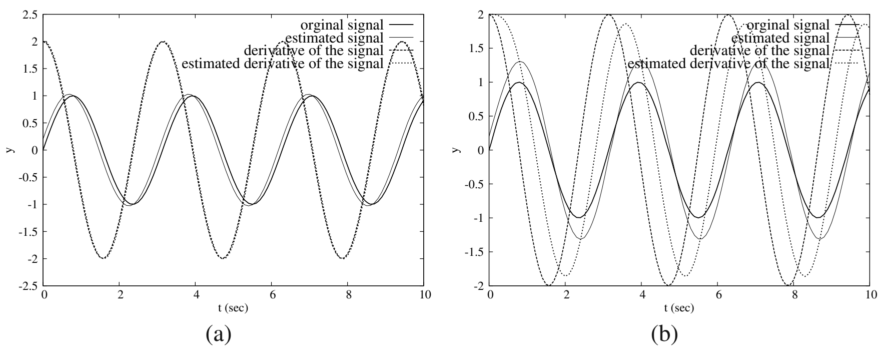


Fig. 7.3 An $\alpha\beta$ filter tested on a noiseless sinusoidal signal $y = \sin 2t$ sampled at frequency of 10Hz. (a) Performance of the filter for $\gamma = 8.0$. Note that the velocity $\dot{y} = 2\cos 2t$ is estimated very well, and the estimated signal lags the original signal by small amount of time. (b) Performance of the filter for $\gamma = 0.1$. The estimated signal and the estimated velocity are not as good as the estimates in (a).

missing variables. For noise-free partially observable domains, we have also two possible ways of optimizing the parameters of the augmented neural network: (a) We can set the tracking index γ_i of each of the $\alpha\beta$ filters to a higher value and optimize only the parameters of the feed-forward neural network. This is equivalent to using direct numerical differentiation for estimating missing state variables (velocities). (b) We can optimize the tracking indices of the $\alpha\beta$ filters and the parameters of the feed-forward neural network simultaneously.

7.2.2.3 Noisy Partially Observable Domains

Partially observable domains which contain noise are the most general case, since virtually all real-world problems are noisy and partially observable. In this case of partially observable domains due to noise, the $\alpha\beta$ filters must filter the noise, and in the case of partially observable domains due to noise and incomplete state variables, the filters not only have to predict the missing state variables, but they must filter the noise at the same time. We need, therefore, to optimize for both cases the tracking indices of the $\alpha\beta$ filters as well as the parameters of the feed-forward neural network. After optimization, a solution will be found that is robust against noise. Figure 7.4 shows performance of an $\alpha\beta$ filter in noisy scenario. An attractive feature of using an $\alpha\beta$ or $\alpha\beta\gamma$ filter is that we only need to optimize n extra parameters in the case of noisy partially observable domains, where n is the number of observable variables of the state of the system. An important contribution of using the augmented neural network (ANKF) unlike the traditional technique is that it allows the optimization of the parameters of the feed-forward neural network and the $\alpha\beta$ filters simultaneously.

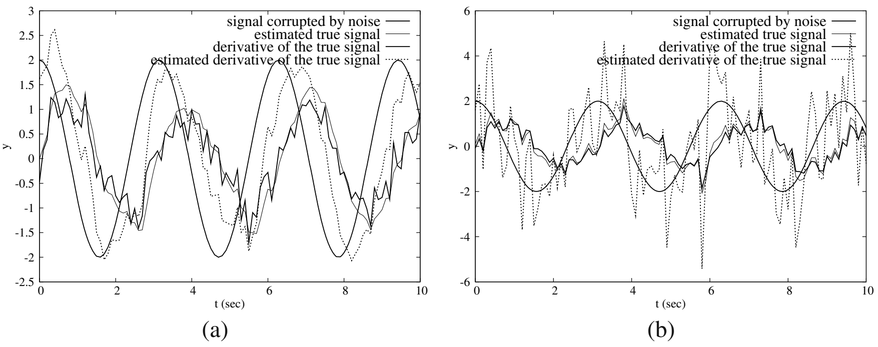


Fig. 7.4 An $\alpha\beta$ filter tested on a sinusoidal signal $y = \sin 2t$ corrupted by a Gaussian noise of mean zero and standard deviation of 0.3. The signal is sampled at frequency of 10Hz. (a) Performance of the filter for $\gamma = 0.1$. Note the delay between the true velocity $\dot{y} = 2\cos 2t$ and the estimated velocity. An optimization algorithm should find a tracking index γ that balances the tradeoff between estimation accuracy and delay for a given learning task. (b) Performance of the filter for $\gamma = 1.0$. The estimated velocity is very noisy and the estimated signal follows the corrupted signal.

7.2.3 Comparison of Number of Parameters to be Optimized for ANKF and Recurrent Neural Networks

For comparing the number of parameters to be optimized for ANKF and recurrent neural networks, we consider a worst case scenario for both of them. Full connectedness results in the maximum number of parameters to be optimized, and therefore can be considered as the worst case scenario for both ANKF and recurrent neural networks. Let m_p be the number of neurons in a fully connected feed-forward neural network of an ANKF and a fully connected recurrent neural network, and let m_n be the number of input variables excluding the bias input to the ANKF and the recurrent neural network. The quantity m_p is the sum of the output neurons m_o and hidden neurons m_h. An example of such networks is shown in Figure 7.5. An interesting question to ask is what is the number of parameters to be optimized for different task environments for an ANKF and a fully connected recurrent neural network having the same number of neurons as the feed-forward neural network of the ANKF. Assuming that we do not optimize the parameters of the activation functions of the neurons, and both the feed-forward neural network of the ANKF and the recurrent neural network have single hidden layer, Tables 7.2 and 7.3 give formulas for calculating the number of parameter to optimize for the ANKF and the recurrent neural network, respectively. A closer look at the formulas given in Tables 7.2 and 7.3 reveals that the number of parameters to be optimized for the ANKF is directly proportional to the number of hidden neurons, while the number of parameters to be optimized for the fully connected recurrent neural network increases quadratically as the number of hidden neuron nodes increases. Let us calculate the number of parameters to be optimized for ANKF and recurrent neural network similar to those

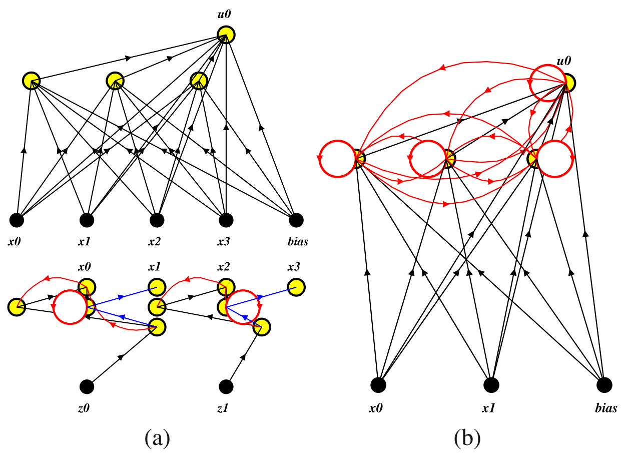


Fig. 7.5 (a) An augmented neural network (ANKF). Note that the network is made up of a fully connected feed-forward neural network and two $\alpha\beta$ filters. (b) A fully connected recurrent neural network having the same number of neurons as the feed-forward network of the ANKF.

shown in Figure 7.5. One can easily see that the networks observe incomplete state variables and each have a bias input included, and they both have a single output $m_o = 1$. Figure 7.6 shows the number of parameters to optimize versus number of hidden neuron nodes for both ANKF with a fully connected feed-forward neural network and a fully connected recurrent neural network. As stated above, one can see from the figure that the number of parameters to optimize increases linearly for ANKF as the number of hidden neuron nodes increases, while the number of parameters to optimize increases quadratically for the fully connected recurrent neural network, as the number of parameters increases. This demonstrates how the use of $\alpha\beta$ filters simplify the topology of the evolving network, in this case decrease the number of parameters to optimize, and as a result reduce the evolution time to find a solution.

Table 7.2 The number of parameters to be optimized for ANKF.

	Complete state variables
No bias	$m = m_o(m_h+1) + m_n(m_o+m_h) + m_n$
Bias	$m = m_o(m_h+2) + m_n(m_o+m_h) + m_h + m_n$
	Incomplete state variables
No bias	$m = m_o(m_h+1) + 2m_n(m_o+m_h) + m_n$
Bias	$m = m_o(m_h+2) + 2m_n(m_o+m_h) + m_h + m_n$

Table 7.3 The number of parameters to be optimized for the fully connected recurrent neural network. Note that $m_p = m_o + m_h$.

	Complete state variables
No bias	$m = m_o(m_h+1) + m_n(m_o+m_h) + m_p^2$
Bias	$m = m_o(m_h+2) + m_n(m_o+m_h) + m_h + m_p^2$
	Incomplete state variables
No bias	$m = m_o(m_h+1) + m_n(m_o+m_h) + m_p^2$
Bias	$m = m_o(m_h+2) + m_n(m_o+m_h) + m_h + m_p^2$

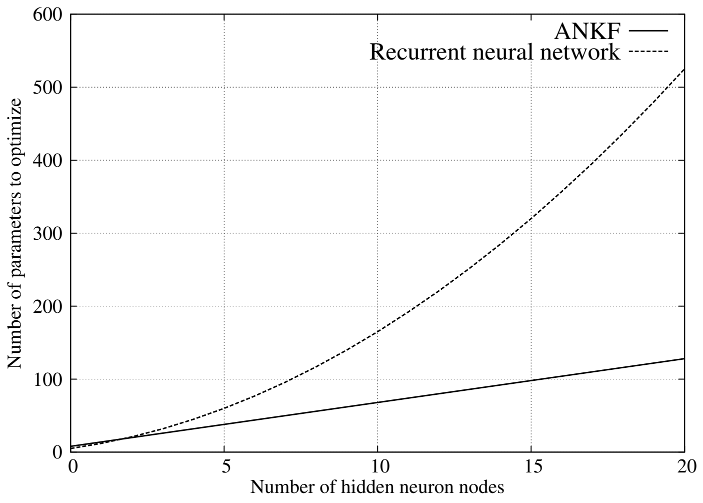


Fig. 7.6 Number of parameters to optimize versus number of hidden neuron nodes for both ANKF and recurrent neural network.

7.2.4 *Results Obtained for ANKF on the Double Pole Balancing without Velocities Benchmark*

ANKF has been tested on the *double pole balancing without velocities* benchmark, and has achieved significantly better results on this benchmark than the published results of other algorithms to date[3]. In the double pole balancing without velocities benchmark, the controller observes only the positions x, θ_1, and θ_2, but *not* the velocities $\dot{x}$, $\dot{\theta}_1$, and $\dot{\theta}_2$. With this benchmark, we used the Gruau's fitness function [14]. The Gruau's fitness function is a weighted sum of two separate fitness measurements $f = 0.1 f_1 + 0.9 f_2$ taken over 1000 timesteps.

[3] All results for ANKF in this paper can be reproduced using the software that can be downloaded at `http://sourceforge.net/projects/eant-project/`.

$$f_1 = t/1000$$

$$f_2 = \begin{cases} 0 & \text{if} \quad t < 100 \\ \frac{0.75}{\sum_{i=t-100}^{t}\left(|x_i|+|\dot{x}_i|+|\dot{\theta}_{1,i}|+|\dot{\theta}_{2,i}|\right)} & \text{otherwise,} \end{cases} \tag{7.2}$$

where t is the number of time steps the pole is balanced starting from a fixed initial position. In the initial position, all states are set to zero except $\theta_1 = 4.5°$. The angle of the poles from the vertical must be in the range $[-36°, 36°]$. The defined fitness function favors controllers that can keep the poles near the equilibrium point and minimize the amount of oscillation. The first fitness measure f_1 rewards successful balancing, while the second measure f_2 penalizes oscillations. The evolution of the neural controllers is stopped when a champion of a generation passes the following test. It has to balance the poles for 10^5 timesteps starting from the 4.5° initialization. After the champion has passed the test, the number of successful balances of the poles for 1000 steps starting from 625 different initial states is recorded for the champion. The number of successful balances is a measure of the generalization performance of the best solution. Each start state is chosen by giving each state variable $(x, \dot{x}, \theta_1, \dot{\theta}_1, \theta_2, \dot{\theta}_2)$ one of the values 0.05, 0.25, 0.5, 0.75, 0.95, 0, 0, scaled to the range of each input variable. The ranges of the input variables are ± 2.16 m for x, ± 1.35 m/s for $\dot{x}$, $\pm 3.6°$ for θ_1, and $\pm 8.6°$ for $\dot{\theta}_1$. Table 7.4 shows the best result obtained by the ANKF along with the best results as reported in literature. The experiments have been done for an ANKF having a feed-forward network with single hidden layer. Motivated by the experimental setup used in [19], each experiment is run using 16 different configurations of the augmented neural network, which are formed from the combinations of the following criteria.

Table 7.4 Results for the double pole-balancing benchmark using Gruau's fitness measure. Average over 50 independent evolutions.

Without velocities		
Method	**Evaluations**	**Generalization**
CE[13]	840,000	300
SANE [25]	451,612	-
CNE [30]	87,623	-
ESP [10]	26,342	-
AGE [7]	25,065	317
EANT [22]	15,762	262
NEAT [29]	6,929	-
CoSyNE [9]	3,416	-
CMA-NeuroES [19]	1,141	-
ANKF [21]	482	455

1. Usage of linear activation function given by $g = x$ versus non-linear activation function given by $g = \tanh(x)$ for the neuron nodes of the feed-forward neural network.
2. Usage of bias versus no bias as an input to the feed-forward neural network.
3. Usage of different number of hidden neurons m_h for the feed-forward neural network, where $m_h \in \{0,1,4,8\}$.

Table 7.5 shows the detailed results obtained for the behavior module in solving the double pole balancing task without velocities. From the results listed in Table 7.4, one can conclude that ANKF outperforms by a large margin all other neuroevolutionary methods tested on this benchmark. ANKF is more than two times faster than the latest evolutionary method CMA-NeuroES, though both methods use the same optimization algorithm CMA-ES [17]. Moreover, the generalization performance of ANKF is significantly better than the generalization performance of other neuroevolutionary methods reported in the literature.

Table 7.5 Results obtained for ANKF for the double pole balancing without velocities benchmark using Gruau's fitness function. In the table, **A**, **B**, **H**, **EV.**, **GEN.**, and **STD.** stand for activation function used, bias, number of hidden neurons evaluations, generalization and standard deviation, respectively.

				Evaluations/ Generalization										
No.	**A**	**B**	**H**	**Mean**		**STD.**		**Median**		**Min.**		**Max.**		**Failures**
				Ev.	**GEN.**	**Ev.**	**GEN.**	**Ev.**	**GEN.**	**Ev.**	**GEN.**	**Ev.**	**GEN.**	
1	linear	no	0	594	420	530	107	440	472	165	138	3575	539	0
2	linear	no	1	482	455	235	93	429	487	143	157	1352	543	0
3	linear	no	4	531	395	263	116	480	423	150	134	1125	542	0
4	linear	no	8	553	429	279	112	510	462	221	63	1326	541	0
5	linear	yes	0	702	400	459	129	600	420	204	90	2856	545	0
6	linear	yes	1	693	393	310	130	624	436	234	81	1599	543	1
7	linear	yes	4	593	384	330	124	520	387	208	88	2016	543	0
8	linear	yes	8	640	336	525	144	595	331	102	102	2006	540	0
9	tanh	no	0	624	442	534	107	462	490	143	89	4267	543	0
10	tanh	no	1	620	312	250	161	539	326	169	36	1495	542	0
11	tanh	no	4	594	264	213	131	562	251	285	32	1335	517	0
12	tanh	no	8	654	185	291	142	552	159	238	25	1632	512	0
13	tanh	yes	0	746	400	348	121	624	422	300	93	1347	538	1
14	tanh	yes	1	789	294	466	156	663	297	234	13	2171	519	1
15	tanh	yes	4	791	198	400	157	672	138	336	8	2208	537	0
16	tanh	yes	8	768	198	301	145	739	159	289	17	1598	526	0

7.3 Case Study 2: Incremental Modification of Fitness Function

In this case study, we first present the testbed (quadrocopter) followed by the control architecture used, and proceed to the description of incremental modification of fitness function and the results obtained with it.

7.3.1 Quadrocopter

Quadrocopters (see [16]) belong to the class of Vertical Take-Off and Landing (VTOL) aerial vehicles. The main advantage of this type of aerial vehicle is the

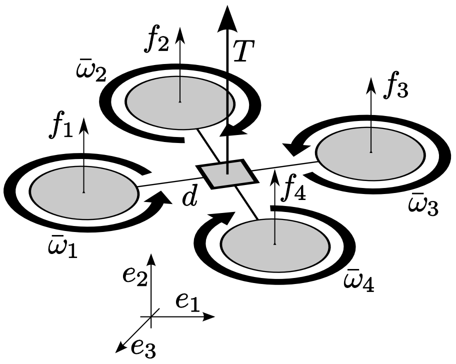


Fig. 7.7 Rotor configuration of the quadrocopter layout.

ability to launch from the stowing position without the need for complex launching facilities. Using Newton's equations of motion, we can describe the system dynamics for the rigid body of the flyer-frame (see [16] for more details).

$$\dot{\xi} = v \qquad m\dot{v} = F - mge_2 \tag{7.3}$$

$$\dot{R} = R[\omega]_\times \qquad I\dot{\omega} = -\omega \times I\omega + \tau, \tag{7.4}$$

where v, ξ are the position and linear velocity in the inertial frame $\mathscr{I}$, and ω is the rotational velocity in the body fixed frame $\mathscr{B}$. The orientation of the body is given by the rotation matrix $R : \mathscr{B} \to \mathscr{I}$, and $[\omega]_\times$ denotes the skew-symmetric matrix for which $[\omega]_\times v = \omega \times v$. Further, m is the mass of the quadrocopter, g is the gravitational constant and e_2 is the second unit vector. The inertia tensor given in body fixed coordinates is given by

$$I = \mathrm{diag}(2dm_m + d_b m_b, 4dm_m, 2dm_m + d_b m_b) \tag{7.5}$$

where d is the distance from the center of gravity (COG) to the rotors' centers, and d_b is the distance from the COG to the battery's center. The masses for the motors m_m and the battery m_b are assumed to be point masses, and m is the entire body mass. The dynamic force $F \in \mathscr{I}$ and torque $\tau \in \mathscr{B}$ are defined by the rotor forward thrust, reactive torque as well as the gyroscopic effects of the four individual motors [16].

$$I_r \dot{\bar{\omega}}_i = \tau_i - Q_i \qquad Q_i = k\bar{\omega}_i^2 \qquad f_i = b\bar{\omega}_i^2 e_2, \tag{7.6}$$

where I_r is the rotor inertia, $\bar{\omega}_i$ is the rotational velocity of the rotor i, Q_i is the rotor drag due to air resistance, and b, k are rotor thrust and drag constants. With

$T = \sum_{i=1}^{4} |f_i|$ being the total thrust and $\tau_a = \{\tau_a^1, \tau_a^2, \tau_a^3\}$ the aerodynamic input to the system torque in (7.4), the rotational velocity of the rotors can be directly related by

$$\begin{pmatrix} T \\ \tau_a^1 \\ \tau_a^2 \\ \tau_a^3 \end{pmatrix} = \begin{pmatrix} b & b & b & b \\ 0 & db & 0 & -db \\ -k & k & -k & k \\ db & 0 & -db & 0 \end{pmatrix} \begin{pmatrix} \bar{\omega}_1^2 \\ \bar{\omega}_2^2 \\ \bar{\omega}_3^2 \\ \bar{\omega}_4^2 \end{pmatrix} = A \begin{pmatrix} \bar{\omega}_1^2 \\ \bar{\omega}_2^2 \\ \bar{\omega}_3^2 \\ \bar{\omega}_4^2 \end{pmatrix}. \tag{7.7}$$

The thrust $T \in \mathscr{B}$ is always produced orthogonal to the rotor plane, from which follows that $F = R_q T$. For the total torques τ we are only missing the gyroscopic effect of the rotors, which can be given as

$$G_a = -\sum_{i=1}^{4} I_r(\omega \times e_2)\bar{\omega}_i, \tag{7.8}$$

leading to $\tau = \tau_a + G_a$. Ignoring the gyroscopic effect for now, it can be seen in the matrix A, which is of full rank, that we can directly relate the upward thrust and the individual torques for yaw, pitch and roll to the squares of the rotor velocities. Using the inverse matrix A^{-1}, the control input can be decoupled into the primary rotation axis and an upward thrust component. The thrust vector T is always pointing in the direction of the positive e_2 axis in the body frame $\mathscr{B}$. Any deviation from the central position on the pitch and roll axis will lead to a force component orthogonal to the gravity vector and move the quadrocopter sideways. Height can be controlled by increasing or decreasing the magnitude of the thrust vector. The position control is invariant to the orientation around the yaw axis, which is usually fixed to a constant value. Table 7.6 shows the parameters of the quadrocopter used in the experiments in this paper. The dynamical system of equations of the quadrocopter is solved using fourth-order Runge-Kutta method with step size $\tau = 0.02 sec$.

Table 7.6 Parameters of the quadrocopter.

m_m	m_b	d_b	d	g
$0.120kg$	$0.250kg$	$0.05m$	$0.30m$	$9.8\frac{m}{sec^2}$

I_r	b	k	m
$80 \cdot 10^{-6}$	$12.42 \cdot 10^{-6}$	$360.6 \cdot 10^{-9}$	$1.08kg$

7.3.2 Control Architecture Developed for the Quadrocopter Using the Principles of Behavior Based Systems

The control architecture for the quadrocopter is shown in Figure 7.8. The architecture represents the manual decomposition of the control task into behaviors which should cooperate to make the robot accomplish a given mission. Each behavior module to be evolved is made up of a fixed structure neural network and a single $\alpha\beta$ filter (see Section 7.2) since each module has only one input. The architecture has six modules: height control, heading control, roll control, attitude control, roll angle

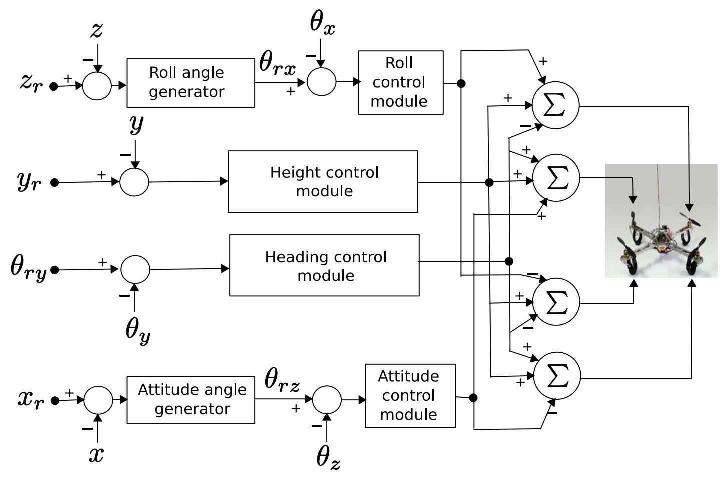


Fig. 7.8 Behavior based control architecture. The quantities x_r, y_r, z_r, θ_{rx}, θ_{ry} and θ_{rz} represent the reference values for the state variables, and the quantities x, y, z, θ_x, θ_y and θ_z represent the feedback state variables to the controller.

generator, and attitude angle generator. The outputs of the height control, heading control, roll control and attitude control modules are superimposed before they are sent to the motors. The height control module sends equal magnitudes of control signal to all of the rotors. The attitude control module sends opposite but of equal magnitude signals to the counterclockwise rotating rotors, and similarly the roll control module sends opposite but of equal magnitude signals to the clockwise rotating rotors. The heading control module sends a control signal to a pair of counterclockwise rotating rotors and the same but opposite signal to a pair of clockwise rotating rotors. The outputs of the attitude angle and roll angle generator modules give the reference angles to the attitude and roll control modules, respectively. These modules control the position of the quadrocopter in the x and z directions by generating appropriate reference angles for the attitude and roll controllers. In this case study, we assume that we have an algorithm which determines the pose $(x, y, z, \theta_x, \theta_y, \theta_z)$ of the flying robot with respect to a frame $\mathscr{I}$ located on the ground. An example of such an algorithm is presented in [1]. The flying robot has to successfully complete the mission, track the ground reference point and successfully land on it. The state of the robot that can be observed by the whole controller is thus given by $s(t) = [x(t), y(t), z(t), \theta_x(t), \theta_y(t), \theta_z(t)]$, where $x(t), y(t), z(t)$ is the 3D postion of the quadrocopter and $\theta_x(t)$, $\theta_y(t)$, and $\theta_z(t)$ are the rotations of the quadropcopter about the x, y and z axes, respectively. Note that the state is partially observable to the controller (without velocity information $\dot{x}(t), \dot{y}(t), \dot{z}(t), \dot{\theta}_x(t), \dot{\theta}_y(t), \dot{\theta}_z(t)$), and a behavior module should be able to estimate the missing state variable by adjusting the tracking index γ of its filter.

7.3.3 Incremental Modification of Fitness Function

In this section we detail the steps needed to incrementally modify the fitness function. Following these steps results in a scalable solution for learning complex robotic tasks.

1. First develop a control architecture using the principles of behavior based systems. In the case of the quadrocopter control, this was shown in Section 7.3.2.
2. Choose a fixed order for training the modules (subsolutions) of the control architecture that was developed.
3. Start with the first module and define a fitness function for it. Let its fitness function be f_1, and the training fitness function be F. Train the module with the fitness function $F = f_1$.
4. Activate the next module to be trained, and define a fitness function for it. Let the fitness function of the next module be f_j. If the next module is parallel to the previously trained module (i. e. its output goes to some common actuators and/or modules where the output of the previously trained module module goes), modify the training fitness function F by *adding to it* the fitness function of the next module (i. e. $F \leftarrow F + f_j$). If the next module to be trained is in series with the already trained module, then modify the training fitness function F, where the fitness function of the next module to be trained *replaces* the fitness function of the module to which its output goes. While training the next module, keep all previously trained modules active in the control loop, and fix their parameters.
5. Repeat step 4 until all the modules in the control architecture are trained.

The byproduct of incrementally modifying the fitness function in this way is that the cooperative coordination of subsolutions is automatically learned without a separate learning process.

7.3.4 Experiments and Results

In this section, we will use the control architecture shown in Figure 7.8 to validate the contributions of this case study, which are: (1) A principled incremental method for modifying the fitness function as the subsolutions (behaviors) are learned. (2) The automatic learning of the cooperative (symbiotic) coordination of subsolutions without the need for a separate learning process. The modules (subsolutions) shown in Figure 7.8 are trained sequentially. The sequence of training is height control, roll control, attitude control, heading control, roll angle generator and attitude angle generator. Each of the modules is represented using ANKF as discussed in Section 7.2.1 by a fixed structure neural network and a single $\alpha\beta$ filter (since each module has only one input). The fixed structure neural network has two inputs, a single output, and no hidden neurons. A total of 5 parameters are optimized for a module using CMA-ES[17]. For the Kalman filter, the tracking index γ_i are optimized, and for the neural network, the two input weights w_1 and w_2, and the output weight w_0 are optimized, as well as the constant a of the output neuron's activation function. This activation function takes on the form $\tanh(ax)$, where $x = w_1\hat{x}_i + w_2\hat{\hat{x}}_i$, and $\hat{x}_i$

and $\hat{x}_i$ are the outputs of the $\alpha\beta$ Kalman filter (and inputs to the neural network). All the sensor values are perturbed by Gaussian noise to make the system robust against disturbances. The modules are expected to attain a given refence state variable value starting from the initial state variable value, which is zero for all state variables. Training controllers by a constant reference value for such a system results in controllers that generalize very well over different reference values. The fitness function f_j of module j takes the form $f_j = \sum_i r_j - s_j(\Delta t)$, where r_j is the reference (desired) value for the state variable for which the behavior module is responsible and $s_j(\Delta t)$ is a feedback signal corresponding to the state variable. Please note that in all of these experiments, the fitness functions have been defined such that smaller fitness values correspond to better solutions. The training procedure, the incremental modification of the fitness functions and the results obtained with it are outlined as follows:

We started with the height controller and defined its fitness function as $f_1 = \sum_{i=0}^{N} |y_r - y(i\Delta t)|$. The training fitness function $F = f_1 = \sum_{i=0}^{N} |y_r - y(i\Delta t)|$ was used, where y is the current height of the quadrocopter, and N is the total number of time steps. In the experiments, $y_r = 10.0$, $N = 3000$, and $\Delta t = 0.02$sec. Figure 7.9a shows the development of the training fitness function $F = f_1$ for the height controller over generations. Figure 7.9b shows the response of the best individual over all evolutionary runs for a novel or unseen reference value $y_r = 4.0m$. As can be seen in the figure, the best individual is able to follow this reference value.

We then proceeded to the training of the attitude control module. We defined the fitness function of the attitude controller as $f_2 = \sum_{i=1000}^{N} |\theta_{rz} - \theta_z(i\Delta t)|$, where $\theta_{rz} = \pi/8$, and θ_z is the rotation of the quadrocopter about z-axis. Since the attitude controller is parallel to the height controller, we modified the training fitness function as $F = f_1 + f_2 = \sum_{i=1000}^{N} |y_r - y(i\Delta t)| + \sum_{i=1000}^{N} |\theta_{rz} - \theta_z(i\Delta t)|$ The index i starts from $i = 1000$ so that the height controller can first reach its desired height $y = 10m$ before the evaluation begins. This is used to avoid an early collision of the quadrocopter with the ground during the evaluation process. Note that while the attitude control module is being trained, the previously learned parameters of the height controller are held constant. Figure 7.10a shows the development of the

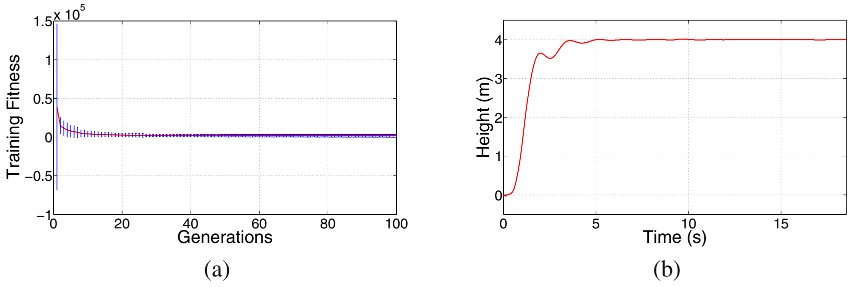


Fig. 7.9 (a) Training fitness $F = f_1$ versus number of generations for the height controller and (b) the response of the learned height controller for a refrence value $y_r = 4m$.

training fitness function $F = f_1 + f_2$ of the attitude controller versus generations. Figure 7.10b shows the height and attitude angle versus time, when both the attitude and the height controllers are simultaneously active. As can be seen in the figure, the two behaviors (subsolutions) cooperate with each other to attain both the target height and attitude angle, proving our claim that if we incrementally modify the fitness function systematically, the byproduct is the automatic learning of the cooperative (symbiotic) coordination of subsolutions without the need for a separate learning process.

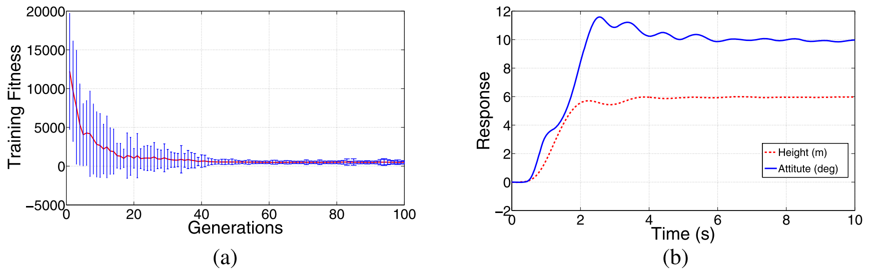


Fig. 7.10 (a) Training fitness function $F = f_1 + f_2$ versus number of generations for the attitude controller and (b) height and attitude controllers attaining their respective desired values.

Next we trained the the roll control module. We defined the fitness function of the roll controller as $f_3 = \sum_{i=1000}^{N} |\theta_{rx} - \theta_x(i\Delta t)|$, where $\theta_{rx} = \pi/8$, and θ_x is the rotation of the quadrocopter about x-axis. Since again the roll controller is parallel to both the height controller and the attitude controller, we modified the training fitness function to be $F = f_1 + f_2 + f_3$. While the roll control module is being trained, the parameters of the height and the attitude control modules are held fixed. While the roll controller is being trained, the reference values of the height controller and the attitude controller are set to $x_r = 10$ and $\theta_z = 0.0$. Figure 7.11a shows the result of the training. Figure 7.11b shows the height y, attitude angle θ_z, and roll angle

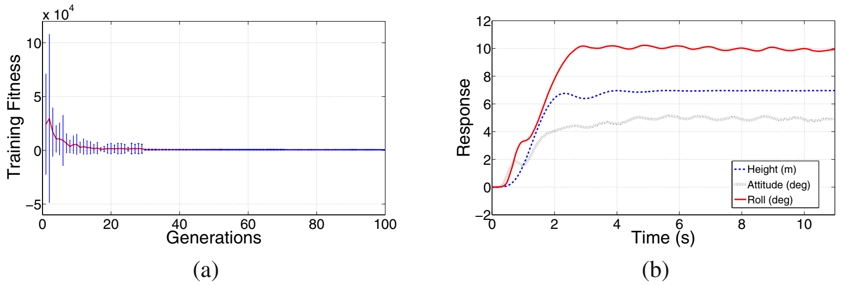


Fig. 7.11 (a) Training fitness function $F = f_1 + f_2 + f_3$ versus number of generations for the roll controller and (b) height, attitude and roll controllers attaining their respective desired values.

θ_z versus time for a novel reference height $y_r = 7m$, novel reference attitude angle $\theta_{rz} = 5°$, and novel reference roll angle $\theta_{rx} = 10°$. As can be seen in the figure, the behaviors cooperate with each other to attain their respective reference values.

After this we trained the heading controller. We defined the fitness function of the heading controller in the same way as we defined the fitness functions of the attitude and roll controller and it is given by $f_4 = \sum_{i=1000}^{N} |\theta_{ry} - \theta_y(i\Delta t)|$, where $\theta_{ry} = \pi/8$, and θ_y is the rotation of the quadrocopter about y-axis. While training the heading controller, we set the reference angles of both attitude and roll controllers to zero. Again since the heading controller is parallel to all of the above controllers, the training fitness function is modified as $F = f_1 + f_2 + f_3 + f_4$. Note that we kept the parameters of the height, attitude and roll control modules constant, while optimizing the parameters of the heading control module. Figure 7.12a shows the development of the training fitness function $F = f_1 + f_2 + f_3 + f_4$ versus number of generations. Figure 7.12 b shows the height y, attitude angle θ_z, roll angle θ_z, and heading angle θ_y versus time for a novel reference height $y_r = 7m$, novel reference attitude angle $\theta_{rz} = 5°$, novel reference roll angle $\theta_{rx} = 10°$, and novel reference heading angle $\theta_{ry} = 5°$. As can be seen in the figure, the heading controller cooperates with the height, roll, and attitude controllers, which results in all behaviors attaining their reference values.

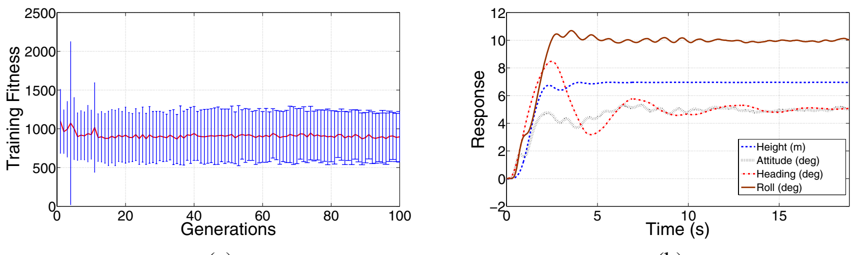
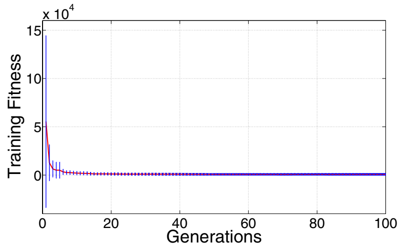


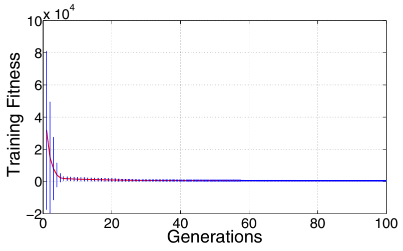
Fig. 7.12 (a) Training fitness function $F = f_1 + f_2 + f_3 + f_4$ versus number of generations for the heading control module and (b) height, attitude, roll and heading controllers attaining their respective desired values.

Next we trained the attitude angle generator and defined its fitness function as $f_5 = \sum_{i=1000}^{N} |x_r - x(i\Delta t)|$, where $x_r = 10.0$, and x is the current position of the quadrocopter along the x-axis. Since the attitude angle generator is in series with the attitude control module (see Figure 7.8), we modified the training fitness function such that we replaced the fitness function of the attitude controller f_2 by the fitness function of the attitude control generator f_5. The resulting training fitness function is given by $F = f_1 + f_5 + f_3 + f_4$. While training the attitude angle generator, we set the reference state values of the roll controller and the heading controller to zero, and held the parameters of height, attitude, roll, and heading modules constant. Figure 7.13a shows the training fitness function versus number of generations for the attitude angle generator.

Finally, we trained the roll angle generator and defined its fitness function as $f_6 = \sum_{i=1000}^{N} |z_r - z(i\Delta t)|$, where $z_r = 10.0$, and z is the current position of the quadrocopter along the z-axis. As can be seen in the control architecture, the roll angle generator is in series with the roll controller. Thus we modified the training fitness function such that the roll controller fitness function f_3 was replaced by the roll angle generator fitness function f_6. The resulting training fitness function is given by $F = f_1 + f_5 + f_6 + f_4$. While training the roll angle generator, we set the reference state values of the attitude angle generator and the heading controller to zero, and kept the parameters of height, attitude, roll, heading, and attitude angle generator modules constant. Figure 7.13b shows the training fitness function versus number of generations for the roll angle generator.



(a)



(b)

Fig. 7.13 (a) Training fitness function $F = f_1 + f_5 + f_3 + f_4$ versus number of generations for the attitude angle generator module and (b) training fitness function $F = f_1 + f_5 + f_6 + f_4$ versus number of generations for the roll angle generator module.

7.3.4.1 Noise Free Trajectory Following Task

After we trained all of the modules in the control architecture, we tested the the controller on noise free trajectory following task. First the quadrocopter was commanded to ascend to a height of 15m above the ground (from A to B as shown in Figure 7.14). Then, it was commanded to follow a circle (BCDEB in Figure 7.14) in a vertical plane, where the reference value z_r was set to zero, and the reference value θ_{ry} was also set to zero. The equations describing the reference trajectory are given by

$$y_r(i\Delta t) = \begin{cases} 15 & \text{if} \quad i < 500 \\ 25 - 10\cos(2\pi \frac{i-500}{N-500}) & \text{if} \quad i >= 500 \end{cases} \tag{7.9}$$

and

$$x_r(i\Delta t) = \begin{cases} 0 & \text{if} \quad i < 500 \\ 10\sin(2\pi \frac{i-500}{N-500}) & \text{if} \quad i >= 500 \end{cases}, \tag{7.10}$$

where $i \in [0,N]$ and $N = 3000$. As can be seen in the figure, the quadrocopter was able to follow the given trajectory successfully, showing again the successful automatic learning of the cooperation of behaviors through incremental modification of the training fitness function. Note that the overshoot at the end of the vertical motion is a result of having chosen a constant reference y_r during the vertical ascent.

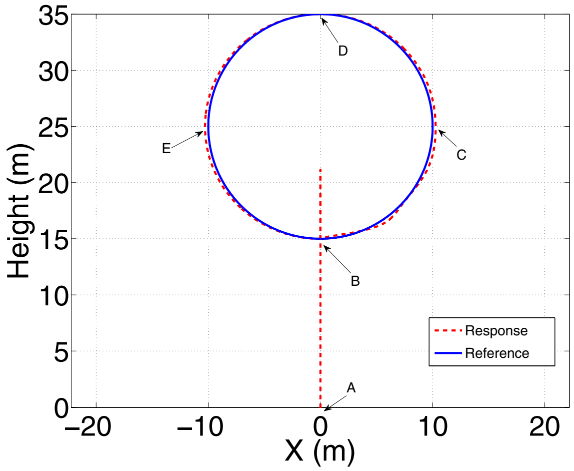


Fig. 7.14 Performance of the controller in trajectory following task.

7.3.5 *Task Decomposition with a Definition of a Single Global Fitness Function Is Not Necessarily Sufficient for Solving Complex Robot Tasks*

The aim of this section is to demonstrate that if we use a single global fitness function like the final fitness function given by $F = f_1 + f_5 + f_6 + f_4$ rather than incrementally modifying the fitness function, we could have difficulty in obtaining a solution with desired operating properties. For the experiment in this section, we used the control architecture and the behavior modules described in Section 7.3.2. Unlike the experiments in the previous sections, we simplified the learning task such that the system is trained in a noise free scenario, meaning that it has to optimize only four parameters per module: the output weight w_0, the two input weights w_1 and w_2 of the neural network and the constant a of the output neuron's activation function $\tanh(ax)$. The reference values for the experiment were $y_r = 5m$, $x_r = 2m$, $z_r = 0.0$, and $\theta_y = 0.0$.

Figure 7.15 shows the fitness development over 200 generations for the control architecture. The minimum value of the average training fitness after 1000 generations when training the whole controller was 17444, while the average minimum value of the final fitness value obtained after 6 *times* 100 generations by the controller trained by incremental modification of the fitness value is 2516. This shows

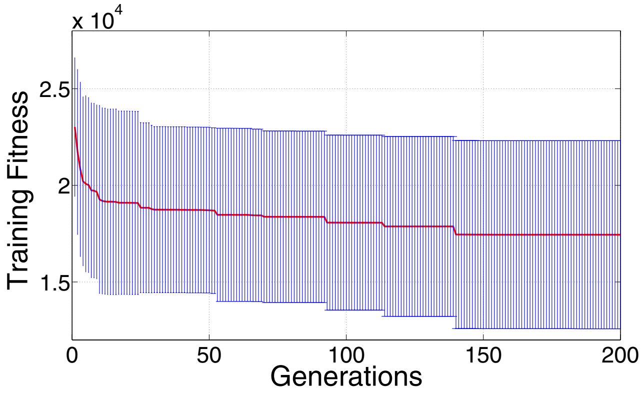


Fig. 7.15 Training fitness function $F = f_1 + f_5 + f_6 + f_4$ versus number of generations for the control architecture.

that decomposing the solution alone and using a global fitness may not result in a solution with desired operating properties. Therefore, one has to incrementally modify the fitness function, as the solution is being complexified (as more modules are learned and added to the system).

7.4 Conclusion

In order to apply neuroevolutionary methods effectively on robots, one has to overcome several problems. The major ones are long evolution time due to noisy sensors and actuators, and the difficulty of defining a global fitness function that results in the desired solution. In this paper, we proposed solutions to both problems. A solution towards the first problem is to accelerate neuroevolutionary methods through Kalman filtering, and the solution towards the second problem is to exploit the design principles of Behavior Based Control (BBC), and to incrementally modify the fitness function that results in a controller with desired operating properties.

References

1. Altug, E., Ostrowski, J.P., Taylor, C.J.: Quadrotor control using dual camera visual feedback. In: Proceedings of the IEEE International Conference on Robotics and Automation, ICRA 2003, pp. 4294–4299 (2003)
2. Arkin, R.C.: Behaviour-Based Robotics. MIT Press, Massachusetts (1998)
3. Bongard, J.: Behavior chaining: Incremental behavior integration for evolutionary robotics. Artificial Life (2008)
4. Botvinick, M., Niv, Y., Barto, A.C.: Hierarchically organized behavior and its neural foundations: A reinforcement learning perspective. Cognition 113, 262–280 (2008)
5. Brooks, R.A.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation 2(1), 14–23 (1986)

6. Calabretta, R., Nolfi, S., Parisi, D., Wagner, G.P.: Duplication of modules facilitates the evolution of functional specialization. Artificial Life 6, 69–84 (2000)
7. Dürr, P., Mattiussi, C., Floreano, D.: Neuroevolution with analog genetic encoding. In: Proceedings of the 9th Conference on Parallel Problem Solving from Nature (PPSN IX), pp. 671–680 (2006)
8. Feldkamp, L.A., Puskorius, G.V.: Training controllers for robustness: multi-stream DEKF. In: IEEE International Conference on Neural Networks, pp. 2377–2382 (1994)
9. Gomez, F., Schmidhuber, J., Miikkulainen, R.: Accelerated neural evolution through cooperatively coevolved synapses. Journal of Machine Learning Research 9, 937–965 (2008)
10. Gomez, F.J., Miikkulainen, R.: Incremental evolution of complex general behavior. Adaptive Behavior 5, 317–342 (1997)
11. Gould, S., Vrba, E.: Exaptation; a missing term in the science of form. Paleobiology 8(1), 4–15 (1982)
12. Gray, J.E., Murray, W.: A derivation of an analytic expression for the tracking index for alpha-beta-gamma filter. IEEE Transactions on Aerospace and Electronic Systems 29(3), 1064–1065 (1993)
13. Gruau, F.: Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. PhD thesis, Ecole Normale Superieure de Lyon, Laboratoire de l'Informatique du Parallelisme, France (January 1994)
14. Gruau, F., Whitley, D., Pyeatt, L.: A comparison between cellular encoding and direct encoding for genetic neural networks. In: Koza, J.R., Goldberg, D.E., Fogel, D.B., Riolo, R.L. (eds.) Genetic Programming: Proceedings of the First Annual Conference, Standford University, CA, USA, pp. 81–89. MIT Press, Cambridge (1996)
15. Gruau, F., Quatramaran, K.: Cellular encoding for interactive evolutionary robotics. Technical report, Amsterdam, The Netherlands (1996)
16. Hamel, T., Mahony, R., Lozano, R., Ostrowski, J.: Dynamic modelling and configuration stabilization for an X4-flyer. In: International Federation of Automatic Control Symposium, IFAC (2002)
17. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation 9(2), 159–195 (2001)
18. Haykin, S.: Kalman Filering and Neural Networks. Wiley, Chichester (2001)
19. Heidrich-Meisner, V., Igel, C.: Neuroevolution strategies for episodic reinforcement learning. Journal of Algorithms (2009), doi:10.1016/j.jalgor.2009.04.002
20. Kalata, P.R.: Alpha-beta target tracking systems: A survey. In: American Control Conference, pp. 832–836 (1992)
21. Kassahun, Y., de Gea, J., Edgington, M., Metzen, J.H., Kirchner, F.: Accelerating neuroevolutionary methods using a Kalman filter. In: Proceedings of the 10th Genetic and Evolutionary Computation Conference, GECCO 2008, pp. 1397–1404 (2008)
22. Kassahun, Y., Edgington, M., Metzen, J.H., Sommer, G., Kirchner, F.: A common genetic encoding for both direct and indirect encodings of networks. In: Proceedings of the 9th Genetic and Evolutionary Computation Conference, GECCO 2007, pp. 1029–1036 (2007)
23. Kirchner, F.: Q-learning of complex behaviors on a six-legged walking machine. Journal of Robotics and Autonomous Systems 25, 256–263 (1998)
24. Lee, W., Hallam, J., Lund, H.: Learning complex robot behaviours by evolutionary computing with task decomposition. In: Learning Robots: 6th European Workshop, pp. 155–172 (1998)

25. Moriarty, D., Miikkulainen, R.: Efficient reinforcement learning through symbiotic evolution. Machine Learning 22, 11–33 (1996)
26. Mouret, J.-B., Doncieux, S.: Evolving modular neural-networks through exaptation. In: CEC 2009: Proceedings of the Eleventh conference on Congress on Evolutionary Computation, Piscataway, NJ, USA, pp. 1570–1577. IEEE Press, Los Alamitos (2009)
27. Ruck, D.W., Rogers, S.K., Kabrisky, M., Maybeck, P.S., Oxley, M.E.: Comparative analysis of backpropagation and the extended Kalman filter for training multilayer perceptrons. IEEE Transactions on Pattern Analysis and Machine Intelligence 14(6), 686–691 (1992)
28. Singhal, S., Wu, L.: Training multilayer perceptrons with the extended Kalman algorithm, pp. 133–140 (1989)
29. Stanley, K.O.: Efficient Evolution of Neural Networks through Complexification. PhD thesis, Artificial Intelligence Laboratory. The University of Texas at Austin., Austin, USA (August 2004)
30. Wieland, A.: Evolving controls for unstable systems. In: Proceedings of the International Joint Conference on Neural Networks, pp. 667–673 (1991)
31. Williams, R.J.: Training recurrent networks using the extended Kalman filter. In: Proceedings International Joint Conference on Neural Networks, pp. 241–246 (1992)
32. Ziemke, T., Bergfeldt, N., Buason, G., Susi, T., Svensson, H.: Evolving cognitive scaffolding and environment adaptation: a new research direction for evolutionary robotics. Connection Science 16(4), 339–350 (2004)

Chapter 8
Evolutionary Design of a Robotic Manipulator for a Highly Constrained Environment

S. Rubrecht, E. Singla, V. Padois, P. Bidaud, and M. de Broissia

Abstract. This paper presents the design of a manipulator working in a highly constrained workspace. The difficulties implied by the geometry of the environment lead to resort to evolutionary-aided design techniques. As the solution space is likely to be shaped strangely due to the particular environment, a special attention is paid to support the algorithm exploration and avoid negative impacts from the problem formulation, the fitness function or the evaluation. In that respect, a specific genome able to encompass all cases is set up and a constraint compliant control law is used to avoid the arbitrary penalization of robots. The presented results illustrate the methodology adopted to work with the developed evolutionary-aided design tool.

8.1 Introduction

In the field of robotic manipulator design, the classical methods [17] turn out to be inefficient when the problem is highly constrained, as the expressions of the constraints (obstacles) cannot be formalized into a classical design formulation. Thus, it is hard to check if a solution complies with the constraints. Moreover, the solution space may be very large, and as the validations are time consuming, it is relevant to use performance indicators and to consider the problem as a multiobjective optimization.

The presence of multiple objectives in a problem gives rise to a set of optimal solutions, instead of a single optimal solution. This set of solutions is known as the

S. Rubrecht · M. de Broissia
Bouygues Travaux Publics. 1, av. E. Freyssinet, 78062 St Quentin en Yvelines
e-mail: {s.rubrecht,m.debroissia}@bouygues-construction.com

E. Singla · V. Padois · P. Bidaud
Université Pierre and Marie Curie – Paris 6, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, F-75005, Paris, France
e-mail: {singla,padois,bidaud}@isir.upmc.fr

set of Pareto-optimal solutions and rely on the notion of *Pareto-dominance* [7] to treat simultaneously and independently each performance indicator.

In a typical minimization problem where the fitness f is composed of n functions f_i $(1 \leq i \leq n)$, a solution $\mathbf{x}$ is dominating an other solution $\mathbf{x}'$ if

$$\exists\ i \text{ such as } f_i(\mathbf{x}) < f_i(\mathbf{x}') \tag{8.1}$$

and

$$\forall j \neq i, f_j(\mathbf{x}) \leq f_j(\mathbf{x}') \tag{8.2}$$

Based on this principle, the solution of a multiobjective optimization is a set of non-dominated solutions (Pareto-optimal solutions) to the problem . In the absence of any further information, one of these Pareto-optimal solutions cannot be said to be better than the other.

Evolutionary Algorithms (EAs) have been widely used in robotics design optimization ([6],[12]) as they are very well adapted for optimization over vast, non continuous search space. This field of application of EA is a growing trend and is mentioned as *evolutionary-aided design* in the introductory chapter of this book. One of the first robot design problems using EA was carried out by Sims [21], generating creatures competing in walking, jumping, swimming, etc.

Since 1990, a large number of MultiObjective Evolutionary Algorithms (MOEAs) have been proposed ([7, 8, 9, 11, 24, 27]). The primary reason for this is their ability to find multiple Pareto-optimal solutions in one single simulation run. Since EAs work with a population of solutions, a simple EA can be extended to maintain a diverse set of solutions. EAs are now widely used, from the whole system structure design to robots reconfiguration [10], controller design, and in various domains such as cooperative robotics [23] and mini-invasive surgery [19]. Amidst several works presented for optimal designs of fundamental robots, Snyman et al. utilized in [22] EAs for the design of a 3R industrial robot while aiming at minimizing joint torque over an entire given trajectory. Another eminent contribution by Ceccaralli and Lanni ([3]) and Carbone et al. in [2] involved the formulation of the robot design problem as a multiobjective optimization problem. However, the complexity associated with cluttered environments and larger number of Degrees Of Freedom (DOFs) is left unaddressed.

This paper details some of the key issues in the design of a robotic serial arm in a highly constrained environment with a special attention to keep the best conditions for the EA to explore the solution space. In that scope, the way to set up the problem (genome choice), the algorithm itself (type and genetic operators), the evaluation step (trajectories, control law) and the indicators retained are fundamental elements. The work in [4, 5, 25] come close to the presented approach. In general, these approaches employ modular robots to cater the task specifications and recommend specific encoding systems to employ EAs to handle varying number of DOFs. However, the work presented in these papers is limited to some specific workspaces and trajectories.

In section 8.2, the environment and context of our work is presented. Section 8.3 details the resolution method, from the problem analysis to the implementation. Section 8.4 exposes the first results. Finally, the last section presents concluding remarks and the future work to be done on this subject.

8.2 Case Study

This research work is led within the framework of a project dedicated to Tunnel Boring Machine (TBM, see Fig. 8.1). The usual tasks are maintenance operations in hostile conditions: hyperbaric atmosphere, high temperature, and even operation immersed in mud.

The geometry of the problem is a typical excavation room geometry (diameter 10m, depth 1 m). The missions defined for fitness are trajectories tracking all around the upper part of the cutter head, focusing on key points to clean or inspect. Transmission arms are obstacles to take into account. The basis of the robot is fixed near the top of the excavation room, at the exit of the airlock.

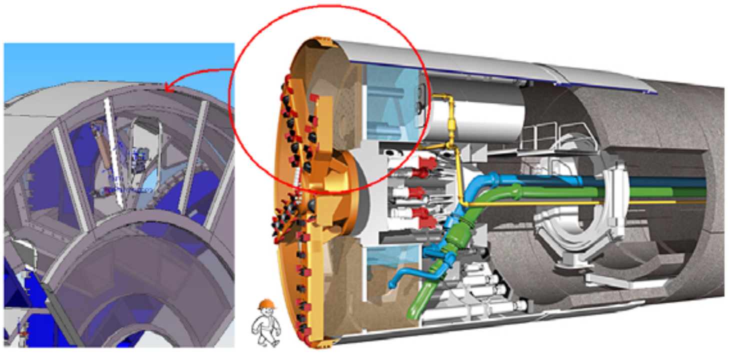


Fig. 8.1 Example of a manipulator in a TBM.

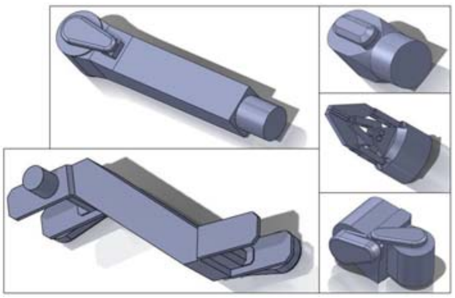


Fig. 8.2 Example of robot segments of the Maestro manipulator from which the elements of the individuals are inspired.

The robots (EA individuals) are composed of elements taken from a pool of robot segments inspired (shapes and joint limits) from the real robot segments of the Maestro manipulator (Cybernetix[1], see Fig. 8.2). In particular, only 1-degree-of-freedom rotational joints are allowed.

8.3 Genetic Algorithm and Implementation

In this section, our implementation of the genetic algorithm is introduced. This implementation relies on SFERES [15] which provides a general framework for evolution based optimization.

8.3.1 Genetic Algorithm

The efforts made to approach the Pareto-optimal front involves two (possibly conflicting) objectives. First is the *convergence* — minimizing the distance between the final Pareto front and the optimal front and second is the *diversity* — maximizing the difference in the generated solutions in terms of objectives or parameter values. To consider both items, the popular technique of Nondominated Sorting Genetic Algorithm II (NSGA-II) [8] is considered suitable for our design problem. This technique possesses the features of elitism and parameter-less sharing. Elitism is the process of selecting better solutions out of the *combined population* of parent and child generations and, therefore, avoid the elimination of any good solution.

For a problem with the population size as N, NSGA-II works on $2N$ solutions at each iteration. These solutions are sorted with respect to their non-domination and are arranged into different Pareto optimal fronts. This is termed as *non-dominated sorting*. To send N solutions to the next iteration, a new list is formed. Since each Pareto front contains equally good candidates, therefore, unless there is less space than the number of elements in a front, all the elements of each front are kept adding to the new list. For further sorting at a particular level, say $r-th$ front, *crowded distance sorting* is utilized. Based on this, the upper ranked elements of the $r-th$ Pareto front are included in the new list. This sorting is based on the maximum distance available around an element, in the objective function space, within which there exists no other element. This helps maintaining some significant *diversity* in the resulting solution, by selecting widely spread population.

The genetic operators are generally used at various rates. In our case, values are taken as:

- Mutation rate: 10%
- Cross over rate: 13%
- Generations: 500
- Individuals: 150

[1] http://www.cybernetix.fr/

8.3.2 Genome

The design process focuses exclusively on the robot morphology using the elementary segments shown in Fig. 8.2. In that framework, each robot is described as a concatenation of segments. A segment is composed of a link having a joint (rotational or prismatic[2]) or not. Two frames are associated to each elementary segment. The first one represents the three possible joint axes: every link is oriented along its z axis. The second one represents the three possible orientations of the next segment.

According to this description, there are 11 elementary segments:

- 3 with a rotational joint about the x axis, the following segment being oriented along x, y or z (called **rxx**,**rxy**, and **rxz** respectively)
- 3 with a rotational joint about the y axis (**ryx**, **ryy** and **ryz**)
- 2 with a rotational joint about the z axis (**rzx** and **rzz**)
- 3 segments without joint (**ex**, **ey** and **ez**)

rzy is not mentioned as it is the same as **rzx** rotated by $\frac{\pi}{2}$ *rads* around z axis.

In addition we define 10 possible lengths for the segments between 0.05 m and 1.05 m. The association table is presented in Table. 8.1.

As an example, a portion of a robot is represented on Fig. 8.3 (left). Each robot is defined by a chromosome of 16 genes, each one representing a segment or not: the genes from 100 to 109 does not match anything (segment "None") which is consistent with the fact that we do not want every robot to have 16 DOFs.

When a fixed segment appears in the genotype of an individual (gene from 190 to 219), a segment combination is done, thus offering the possibility to have segments which orientation differs from the x, y and z axes (Fig. 8.3 right).

Table 8.1 Genome. Each gene is a number composed of 3 digits: the 2 first are the joint type, the last being the link length. A gene value is between 100 and 219.

Gene **ABC**	**AB**: Joint type
	C: length (m)

Joint number - **AB**	10	11	12	13	14	15
Joint type	None	rxx	rxy	rxz	ryx	ryy

Joint number - **AB**	16	17	18	19	20	21
Joint type	ryz	rzx	rzz	ex	ey	ez

Length number - **C**	0	1	2	3	4
segment length (m)	0.05	0.15	0.25	0.35	0.45

Length number - **C**	5	6	7	8	9
segment length (m)	0.55	0.65	0.75	0.85	0.95

[2] Prismatic joints are not used within the framework of the considered application.

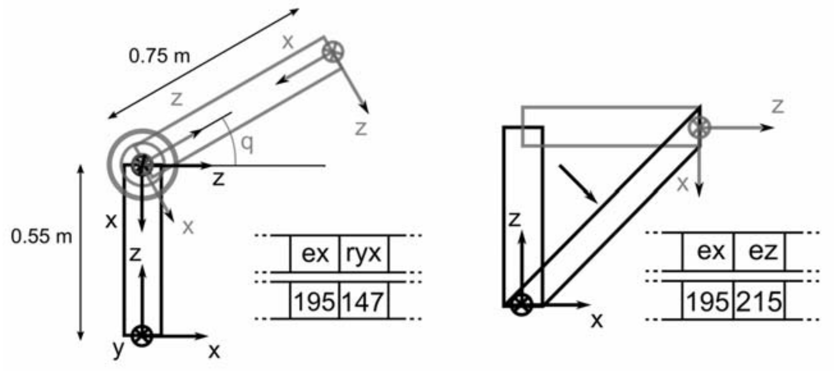


Fig. 8.3 Genotypes examples: portion of robot and combination of 2 segments. q is the rotational joint angle.

8.3.3 Trajectory Tracking

The aim of the fitness function is to qualify the ability of an individual (robot) to carry out a maintenance mission in the TBM. An efficient way to check the motion skills of a robot is to simulate a trajectory tracking in the 3D environment. So, a relevant 3D trajectory has been defined (sequence of 361 3D points for a total length of approximately 8.5 *m*, which comes out to a mean distance of 24 *mm* between 2 points) and the fitness function is a trajectory tracking. Dynamics is not computed as it does not impact on the indicators retained (see 8.3.5). Each individual has to track the same trajectory. The simulator uses the Kinematic and Dynamic Library (KDL) which is part of the OROCOS project [1].

8.3.4 Control Law

The control law of the robots is in charge of computing at each simulation iteration the joint velocities to reach the current point in the sequence of 3D points composing the trajectory. In our case, two specifications led us to design our own control law to handle the problem of tracking a given trajectory by any manipulator in a cluttered environment. First, a guideline of this work is to compensate the impact of cluttered environment by supporting the EA exploration. Consequently, the control law should neither be penalizing in terms of configurations (e.g. to deal properly with singular configurations or proximity to constraints) nor in terms of robots (redundant or not). Second, the framework of evolutionary-aided design requires meaningful fitnesses to be efficient: so, the constraints violations (such as collisions between the robot and the environment), which would never occur in real conditions, are not accepted.

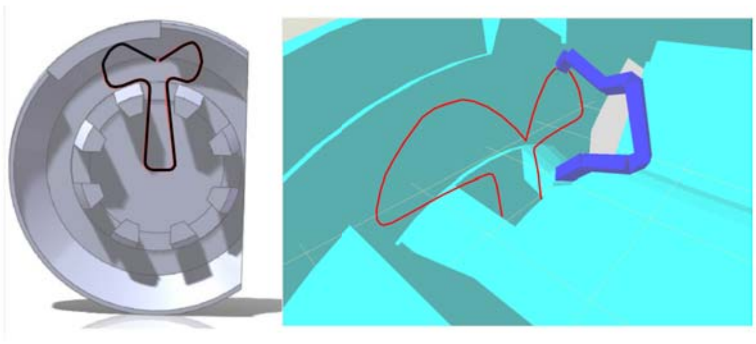


Fig. 8.4 Section view of the TBM trajectory and manipulator example.

8.3.4.1 Control Framework

The huge number of individuals evaluations (trajectory trackings of manipulators) prevents from using prediction or planification techniques in the control strategy (for obvious computation time reasons), so the control law is reactive. Regarding the framework, a velocity kinematics framework has been retained rather than a purely kinematic one. Actually, as the robot is not known a priori, a general inversion method is needed to be applied at each simulation iteration. In the kinematic framework, the model linking the joints and the operational positions is not linear and this operation is complex and time consuming.

8.3.4.2 CCC

The control law briefly described here - Constraint Compliant Control (CCC) - is detailed in [18]. The CCC is an iterative velocity kinematics control law. It is a strict prioritized multiobjective law ([13], [20]) with 3 hierarchical levels:

1. The first level gathers the terms relative to *passive avoidance*: to satisfy the considered constraints (collisions) formulated as inequalities, the robot motion is stopped along the directions of the critical constraints. The critical constraints are determined through an iterative process;
2. The second level gathers the operational tasks, in our case the trajectory tracking;
3. The third level gathers the active avoidance terms: it tends to get the robot away from the constraints. Most of the time, as the environment is cluttered, these terms cannot be all satisfied, which justifies the existence of the first term.

The main property of the CCC is that it never violates its constraints, even if they are not compatible with the trajectory tracking. As a result, indicators related to these constraints are useless in the design process. Moreover, the CCC resorts to the Damped Least Square (DLS) inverse [16], to avoid inconvenient behaviors around kinematic singularities. As a consequence, the compliance with the constraints and the approximation around singularities directly impacts the trajectory tracking error rather than imposing dedicated indicators. It appears more relevant as it limits the number of indicators without creating meaningless weighted sums of scores based on non realistic behaviors.

8.3.4.3 Practical Implementation

In practice, a computationally efficient implementation of the CCC does not perfectly ensure collisions avoidance (one constraint per segment may not always be sufficient, see [18]). So, the number of collisions per segment per iteration is included in the set of indicators. Anyway, the use of the CCC is justified as only a small portion of the evaluated robots collides, which minimizes the impact of this indicator on the complexity of the problem (see 8.3.5).

8.3.5 Indicators

The indicators are the scores obtained by the robot through the fitness function. They are voluntarily simple and composed of a single magnitude (no weighted sums representing a priori tradeoffs between different magnitudes). The trajectory tracking quality but also intrinsic parameters are rated, such as the number of DOFs. All the indicators, listed below, are to be minimized:

- **Maximum linear error along the trajectory tracking.** There are no strategic points on which to compute the error with a higher weight w.r.t to others: the current design being a preliminary design, the trajectory should be equally tracked;
- **Number of DOFs.** The number of DOFs is a technological difficulty (manufacturing, energy, control), even if a more redundant robot has, in general, better reachability skills in a cluttered environment.
- **Robot total length.** The shorter robot able to perform the trajectory tracking has usually better adaptability to other tasks.
- **Number of collisions per segment per iterations.** As mentioned previously, despite the use of the CCC, the number of collisions is added as a fourth indicator to minimize. However, the CCC considerably reduces the number of collisions w.r.t. usual control laws. As a result, almost every robot obtain 0 (no collision), and the size of the problem is not much increased by the presence of this indicator. Actually, a robot failure on this indicator is most often due to a control failure rather than because of antagonism between indicators. In order not to penalize the individuals for which this failure occurs, it has been decided not to take this indicator as a *constraint* (in the EA sense : criteria which, if not respected, disqualifies the individual).

8.4 Results

Even if the design process is still under progress, the results presented here are conclusive regarding our particular problem.

8.4.1 Design with Simple Trajectory

Preliminary designs have been realized with a simple trajectory to set up the process properly. The robot in Fig. 8.5 is a solution obtained with only 3 indicators:

- Maximum linear error of the trajectory tracking;
- Number of DOFs;
- Number of collision per segment per iteration.

The retained manipulator possesses 5 DOFs and tracks the path with a maximum error of 80 *mm* (shown on Fig. 8.5). However, the robot cannot be considered acceptable as the link lengths are too large, with a total robot length of 6.40 *m*. Such results encouraged to include the links size in the set of indicators of the optimization process.

In order to obtain more reasonable robots, the total length of the robot had been added to the set of objective functions. The resulting robot for the presented case is much shorter (1.60 m), as shown in Fig. 8.6. The number of DOFs is 5 and the maximum trajectory tracking error is 90 *mm*, which remains acceptable and tends to prove that the robot total length is not antagonistic with other indicators in this case.

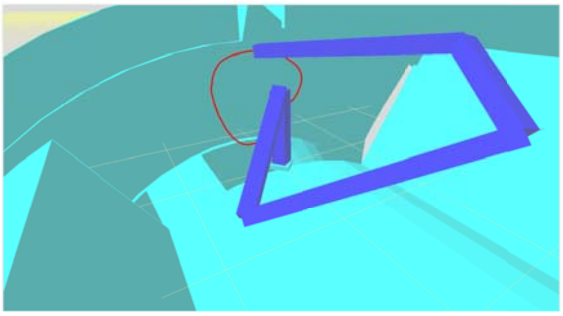


Fig. 8.5 Robot 1, indicators: linear tracking error, number of DOFs and collisions per segment per iteration.

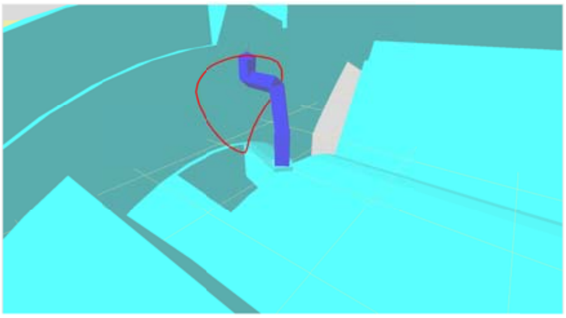


Fig. 8.6 Robot 2, indicators: linear tracking error, number of DOFs, collisions per segment per iteration and robot total length.

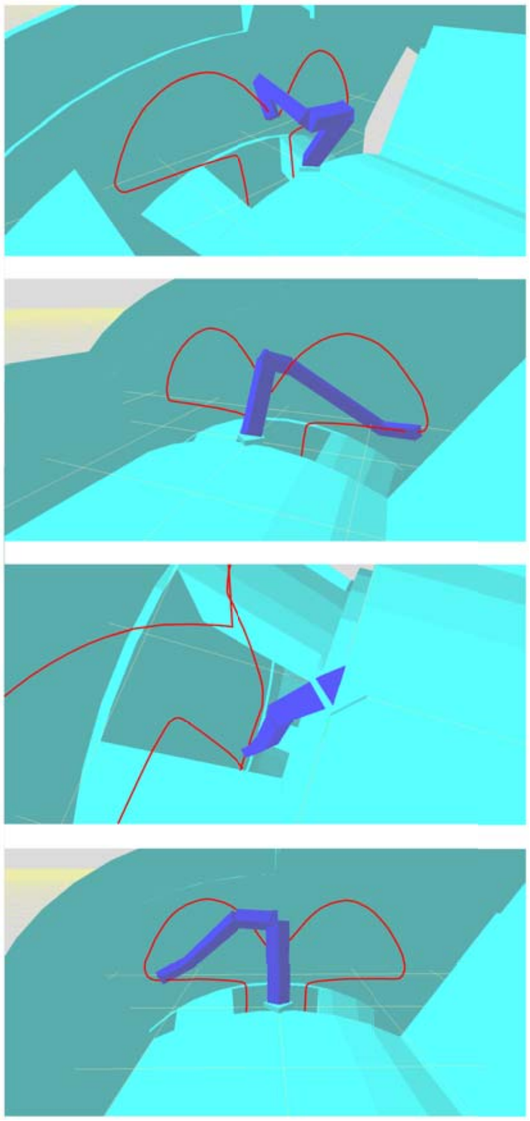


Fig. 8.7 Sequence of the complex trajectory tracking. Robot 3, indicators: linear tracking error, number of DOFs, collisions per segment per iteration and robot total length.

8.4.2 *Design with Complex Trajectory*

As the solutions fit the specifications for the simple trajectory, a similar work had been carried out with the trajectory representing a maintenance mission (inspection of the cutter head). This path includes the complications of navigating the robot deep into the narrow space, available between the cutter head and the robot base (see 8.3.3 and Fig. 8.4). Using the 4 indicators (linear tracking error, number of DOFs, total length and number of collisions per segment per iteration) turns out to be sufficient to obtain appropriate robots.

The tracking of one of the final robots is represented in a sequence of simulation pictures, shown in Fig. 8.7. This robot has 5 DOFs with a maximal linear error of 120 *mm* and a length of 2.80 *m*.

The selection of a suitable robot out of all the possible solutions of the final front is another task in the complete design process, which is not a part of this paper. However, an example of pareto front obtained is represented on Fig. 8.8. It is worth mentioning here that since collision avoidance is an inherent part of the chosen motion controller, the working of the resulting robots would certainly be free from any collision when using such a controller.

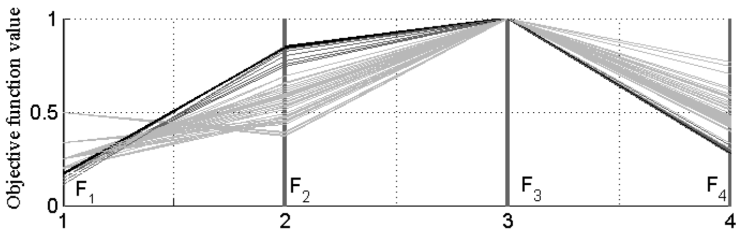


Fig. 8.8 Example of Pareto Front. The scores have been inversed and scaled between 0 and 1 (1 is the best value). F1: linear tracking error; F2: number of DOFs, F3: collisions per segment per iteration, F4: robot total length. The black line is the robot presented in 8.4.2.

8.5 Conclusions and Future Works

8.5.1 *Conclusions*

The work presented here finds its justification in the maintenance of TBM in hostile conditions. The environment being very constrained, evolutionary design offers many advantages, but attention must be paid to preserve a good exploration as the solution space is not well shaped. A simple and exhaustive genotype has been set up to cover easily all robot possibilities, including segment directions not only along the absolute frame axes. The constraints compliant control law resorts to passive constraint avoidance to make the behavior more realistic. It avoids individuals penalization, makes indicators more meaningful and reduces the impact of the collision based indicator on the problem size. The results obtained are suitable solutions to our problem.

8.5.2 Future Works

A sensitivity analysis is needed to estimate the impact of our work toward diversity maintenance in the population. In addition, more sophisticated indicators are currently being tested, such as manipulability [26]. Finally, implementation of recent works in evolutionary design will be carried out, such as:

- Variation of the genetic operators individual by individual according to their position and repartition along the Pareto front [5];
- Evolution of the indicator along the process to prevent from bootstrap [14].

Acknowledgements

This work is a part of the TELEMACH project and is supported by the French National Research Agency (ANR), Interactive Systems and Robotics Program 2007 (PSIROB 07).

References

1. Bruyninckx, H.: Open robot control software: the orocos project. In: IEEE International Conference on Robotics and Automation, vol. 3, pp. 2523–2528 (2001)
2. Carbone, G., Ottaviano, E., Ceccarelli, M.: An optimum design procedure for both serial and parallel manipulators. Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science 221(7), 829–843 (2007)
3. Ceccarelli, M., Lanni, C.: A multi-objective optimum design of general 3R manipulators for prescribed workspace limits. Mechanism and Machine Theory 39(2), 119–132 (2004)
4. Chedmail, P., Ramstein, E., CMAO-Productique, N.: Robot mechanism synthesis and genetic algorithms. In: IEEE International Conference on Robotics and Automation, vol. 4, pp. 3466–3471 (1996)
5. Chocron, O.: Evolutionary design of modular robotic arms. Robotica 26(03), 323–330 (2008)
6. Chocron, O., Bidaud, P.: Genetic design of 3D modular manipulators. In: IEEE International Conference on Robotics and Automation, vol. 1, pp. 223–228 (1997)
7. Deb, K.: Multi-objective optimization using evolutionary algorithms. John Wiley & Sons, Chichester (2001); The book in use for refering any term and process relatted multi-objective optimization. It emphasizes on Evolutionary techniques but gives significant difference between classical and evolutionary techniques
8. Deb, K., Pratap, A., Agrawal, S., Meyrivan, T.: A fast and elitist multi-objective genetic algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)
9. Fonseca, C., Fleming, P.: Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In: Fifth International Conference on Genetic Algorithms, pp. 416–423 (1993)
10. Han, J., Chung, W.K., Youm, Y., Kim, S.: Task based design on modular robot manipulator using efficient genetic algorithm. In: IEEE International Conference on Robotics and Automation, vol. 1, pp. 507–512 (1997)

11. Horn, J., Nafploitis, N., Goldberg, D.: A niched Pareto genetic algorithm for multiobjective optimization. In: First IEEE Conference on Evolutionary Computation, vol. 1, pp. 82–87 (1994)
12. Leger, C.: Automated synthesis and optimization of robot configuration: an evolutionary approach, ph.d. dissertation. Ph.D. thesis, Canegie Mellon University (1999)
13. Liégeois, A.: Automatic supervisory control of the configuration and behavior of multibody mechanisms. IEEE Transactions on Systems, Man and Cybernetics 7(12), 868–871 (1977)
14. Mouret, J.B., Doncieux, S.: Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In: IEEE Congress on Evolutionary Computation, pp. 1161–1168 (2009)
15. Mouret, J.B., Doncieux, S.: Sferes_v2: Evolvin' in the Multi-Core World. In: WCCI 2010 IEEE World Congress on Computational Intelligence, Congress on Evolutionary Computation (CEC), pp. 4079–4086 (2010)
16. Nakamura, Y., Hanafusa, H.: Inverse kinematics solutions with singularity robustness for robot manipulator control. Journal of Dynamic Systems, Measurement and Control 108, 163–171 (1986)
17. Pahl, G., Beitz, W.: Engineering Design. Springer, Berlin (1984)
18. Rubrecht, S., Padois, V., Bidaud, P., de Broissia, M.: Constraint compliant control for a redundant manipulator in a cluttered environment. In: Proceedings of the 12th International Symposium on Advances in Robot Kinematics, pp. 367–376 (2010)
19. Salle, D., Bidaud, P., Morel, G.: Optimal design of high dexterity modular mis instrument for coronary artery bypass grafting. In: IEEE International Conference on Robotics and Automation, vol. 2, pp. 1276–1281 (2004)
20. Siciliano, B., Slotine, J.J.: A general framework for managing multiple tasks in highly redundant robotic systems. In: IEEE International Conference on Advanced Robotics, vol. 2, pp. 1211–1216 (1991)
21. Sims, K.: Evolving virtual creatures. In: Computer Graphics, SIGGRAPH 1994, pp. 15–22 (1994)
22. Snyman, J., van Tonder, F.: Optimum design of a three-dimensional serial robot manipulator. Structural and Multidisciplinary Optimization 17(2), 172–185 (1999)
23. Sotzing, C., Htay, W., Congdon, C.: Gencem: A genetic algorithms approach to coordinated exploration and mapping with multiple autonomous robots. IEEE Congress on Evolutionary Computation 3, 2317–2324 (2005)
24. Srinivas, N., Deb, K.: Muiltiobjective optimization using nondominated sorting in genetic algorithms. Evolutionary Computation 2(3), 221–248 (1994)
25. Yang, G., Chen, I.: Task-based optimization of modular robot configurations: minimized degree-of-freedom approach. Mechanism and machine theory 35(4), 517–540 (2000)
26. Yoshikawa, T.: Manipulability of robotic mechanisms. The International Journal of Robotics Research 4(2), 3–9 (1985)
27. Zitzler, E., Thiele, L.: Multiobjective optimization using evolutionary algorithms - A comparative case study. In: Eiben, A.E., Bäck, T., Schoenauer, M., Schwefel, H.-P. (eds.) PPSN 1998. LNCS, vol. 1498, p. 292. Springer, Heidelberg (1998)

Chapter 9
A Multi-cellular Based Self-organizing Approach for Distributed Multi-Robot Systems

Yan Meng, Hongliang Guo, and Yaochu Jin

Abstract. Inspired by the major principles of gene regulation and cellular interactions in multi-cellular development, this paper proposes a distributed self-organizing multi-robot system for pattern formation. In our approach, multiple robots are able to self-organize themselves into various patterns driven by the dynamics of a gene regulatory network model. The pattern information is embedded into the gene regulation model, analog to the morphogen gradient in multi-cellular development. Various empirical analysis of the system robustness to the changes in tasks, noise in the robot system and changes in environment has been conducted. Simulation results demonstrate that the proposed method is both effective for pattern formation and robust to environmental changes.

9.1 Introduction

Distributed multi-robot systems (MRS) can be used to fulfill tasks that are quite difficult or even unfeasible for a single robot, especially in the presence of uncertainties, incomplete information, distributed control, and asynchronous computation. Compared with centralized systems, distributed MRS can provide more flexibility, robustness, and adaptiveness in the tasks inherently distributed in space and/or time. Some real-world applications of distributed MRS include urban search and rescue, battlefield surveillance and scouting, space and interplanetary exploration, hazardous material collection.

Yan Meng · Hongliang Guo
Department of Electrical and Computer Engineering, Stevens Institute of Technology, NJ 07030, USA
e-mail: yan.meng@stevens.edu, hguo@stevens.edu

Yaochu Jin
Department of Computing, University of Surrey, Guildford, Surrey GU2 7TW, UK
e-mail: yaochu.jin@surrey.ac.uk

Since we are using distributed approaches, it is difficult to predict the emerging behaviors only from local interactions of individual agents; neither is it easy to design rules for local interactions to generate a desired global behavior. We find that, biological systems, from macroscopic swarm systems of social insects to microscopic cellular systems, can generate robust and complex emerging behaviors through relatively simple local interactions subject to various kinds of uncertainties [14]. Therefore, many researchers got inspiration from biological systems and proposed many bio-inspired methods for MRS [17, 18, 20, 22].

Embryonic development of multi-cellular organisms is governed by gene regulatory networks (GRNs). As we know, GRNs are models of genes and gene interactions at the expression level. It is a collection of DNA segments in a cell which interact with each other indirectly through their RNA, protein product, other chemicals in the cell, and governs the rates at which genes in the network are transcribed into mRNA. GRNs play a central role in understanding natural evolution and development [2]. To this end, various models of GRNs have been suggested [6, 8, 11, 16].

Some work has been conducted on multi-robot pattern formation. Albayrak [1] proposed a decentralized control algorithm for line and circle formations. More complex shapes were considered in [21]; however, this approach requires each individual robot to have an estimate of all the other robots positions, and thus imposes serious limitations. Leader-follower approaches [7] require the assignment of different controllers and different set-points to different robots. Shen et al. [20] proposed a digital hormone model (DHM) as a distributed control method for robot swarms. They applied Turings reaction-diffusion model [23] to describe the interactions between the hormones. The DHM integrated a dynamic network, stochastic action selection, and hormone reaction-diffusion. No global information is dynamically embedded into the DHM model. All these bio-inspired methods mainly focus on developing some heuristic rules to fill the robots/agents in predefined simple close-form shapes. Hsieh and Kumar [12] proposed a potential-field based approach for pattern generation with multiple robots. Very few researchers have applied the GRN-based model for multi-robot systems. Taylor [22] has proposed a gene regulatory network inspired real-time controller for a group of underwater robots for a simple clustering task.

In this paper, we will propose a distributed, multi-cellular based self-organizing approach for a multi-robot system for pattern formation tasks. The major contributions of our proposed method include: (1) the systems global information, such as the shape function, can be embedded into the GRN dynamics directly; (2) the dynamics of the GRN-inspired model can automatically drive the robots to their target positions while avoiding collision between the robots and obstacles inside the environment; (3) the system is self-organizing and robust to various environmental changes. Part of this work has been published in our previous paper [10].

The paper is organized as follows. Section 9.2 introduces the biological background that inspired this work. The GRN-based distributed control approach for a self-organized multi-robot system is described in Section 9.3. To evaluate the proposed method, several case studies of pattern formation using a multi-robot system have been conducted, and the robustness analysis to environmental changes and system noise are also provided in Section 9.4. Conclusion and future work are discussed in Section 9.5.

9.2 Biological Background

When a gene is expressed, information stored in an organisms genome is transcribed and translated into proteins. Some of these proteins are transcription factors that can regulate the expression of their own or other genes. Thus, these proteins are under regulatory control, resulting in complex networks of interacting genes. These gene regulatory networks control a number of important cellular processes, such as responding to the environment, regulating the cell cycle and guiding the development of an organism.

It is very challenging to gain a thorough understanding of the emergent behaviors of complex patterns from the interactions of genes in a regulatory network. Therefore, mathematical modeling and simulation of gene regulation processes are indispensible. A large number of GRN models have been suggested, of which ordinary differential equations (ODEs) have been used to model the reaction kinetics of regulatory systems with a long history. In this section, we will introduce the ODE form of GRN models for multi-cell organisms.

In a multi-cell organism, it is necessary to model the intercellular communication. In addition to the internal dynamics of the cell, we should also consider external factors such as protein gradients and physical interactions between cells. Turing [23] proposed one of the earliest models for pattern formation, where a pair of coupled reaction-diffusion equations was proposed to describe a system consisting of two morphogens. As two morphogens diffuse across a spatial field and react with one another, a variety of patterns emerge depending on parameter values. The gradients of protein concentrations across cells are a critical feature in embryonic development. The reaction-diffusion equations have been widely used in mathematical biology to study pattern formation in development [9, 13, 15].

Salazar-Ciudad et al. [19] proposed a GRN model with reaction-diffusion mechanism as follows:

$$\frac{dx_{ij}}{dt} = f_j(\mathbf{x}_i, \mathbf{u}) - \gamma_i x_{ij} + D_j \nabla^2 x_{ij}, 1 \le i \le n, 1 \le j \le m \tag{9.1}$$

where x_{ij} is the concentration of gene product j i cell i. The first term specifies the production of x_{ij}, the second term is its degradation, and the last term specifies the diffusion component at diffusion rate D_j. f_j is a nonlinear update function of gene product j, which can be defined as a sigmoid function as $f(x) = \frac{1}{1+e^x}$. $\mathbf{u}$ is the vector of external input signals. γ_i is the degradation rate of product i. n is the number of gene products in one cell, and m is the number of cells.

9.3 The Approach

In this paper, our objective is to deploy multiple robots to various patterns in a distributed manner. During the deployment, it is assumed that each robot can only perceive its local environment and makes its own movement decisions without communicating with other agents. The global pattern to be formulated in the environment is predefined. Initially, all robots are randomly distributed within the environment. It is assumed that robots know their own initial position in the environment and can self-localize themselves in the environment using their onboard sensors.

9.3.1 The GRN-Based Dynamics

Inspired by the multi-cellular GRN model, it is assumed in this paper that each robot corresponds to a single cell. Within each cells genome, there are two genes, one for x-position and one for y-position in a 2D environment. Each gene can produce a certain protein. Each protein can provide the following three functions: (1) To regulate the expression of the gene that produced it (i.e. auto regulation); (2) To adjust the robots behaviors; (3) To be able to diffuse proteins to its neighbors to prevent collision from each other.

The system dynamics of the GRN-based approach for a multi-robot system for pattern formation tasks are defined as:

$$\begin{aligned}\frac{dg_{i,x}}{dt} &= -a\mathbf{z}_{i,x} + mp_{i,x} \\ \frac{dg_{i,y}}{dt} &= -a\mathbf{z}_{i,y} + mp_{i,y}\end{aligned} \tag{9.2}$$

$$\begin{aligned}\frac{dp_{i,x}}{dt} &= -cp_{i,x} + kf(\mathbf{z}_{i,x}) + bD_{i,x} \\ \frac{dp_{i,y}}{dt} &= -cp_{i,y} + kf(\mathbf{z}_{i,y}) + bD_{i,y}\end{aligned} \tag{9.3}$$

where $g_{i,x}$ and $g_{i,y}$ are the expression levels of the i^{th} robot's gene for x-position and y-position, respectively. $p_{i,x}$ and $p_{i,y}$ are the concentration of the i^{th} robots proteins for x-position gene and y-position gene, respectively. a, m, c, k, and b are coefficient factors.

In order to embed the predefined 2D shape, which is the global information, into the dynamic equations, we define $f(\mathbf{z}_i)$ as the following sigmoid functions:

$$\begin{aligned}f(\mathbf{z}_{i,x}) &= \frac{1-e^{-\mathbf{z}_{i,x}}}{1+e^{-\mathbf{z}_{i,x}}} \\ f(\mathbf{z}_{i,y}) &= \frac{1-e^{-\mathbf{z}_{i,y}}}{1+e^{-\mathbf{z}_{i,y}}}\end{aligned} \tag{9.4}$$

where $\mathbf{z}_{i,x}$ and $\mathbf{z}_{i,y}$ are the gradients along x-axis and y-axis, respectively, of a predesigned function h at the robots current gene expression level, which are defined as:

$$\mathbf{z}_{i,x} = \frac{\partial h}{\partial g_{i,x}}, \mathbf{z}_{i,y} = \frac{\partial h}{\partial g_{i,y}}, \tag{9.5}$$

where the predesigned function h is the function of the desired shape where robots are supposed to be deployed uniformly. We can also treat function h as the predefined gradient for cell migration. To facilitate the generation of the desired dynamics, we defined h as the square of the desired shape function. For example, if we want to deploy the robots onto a unit circle. The shape function can be defined as:

$$s(g_{i,x}, g_{i,y}) = g_{i,x}^2 + g_{i,y}^2 - 1 = 0 \tag{9.6}$$

Then function h can be defined as:

$$h = (g_{i,x}^2 + g_{i,y}^2 - 1)^2 \tag{9.7}$$

We use D_i to define the protein diffusion which aims at keeping the robot away from its neighbors. The size of neighborhood varies according to different shapes and different number of robots. In the example of a circular shape, the neighborhood size can be defined as $\frac{2\pi r}{N}$, where r is the radius of the circle, and N is the total number of robots which are expected to deploy on the circle.

When a robot detects its neighbor, it will receive the protein emitted from that neighbor so that it would keep itself away from that neighbor to avoid collision. After summing all the neighbors diffused protein together, we have

$$D_{i,x} = \sum_{j=1}^{N_i} D_{i,x}^j, D_{i,y} = \sum_{j=1}^{N_i} D_{i,y}^j, \tag{9.8}$$

where N_i denotes the number of its neighbors, and $D_{i,x}^j$ and $D_{i,y}^j$ are the diffusions along x-axis and y-axis, respectively, on robot i emitted from the neighbor robot j, which is defined as:

$$D_{i,x}^j = \frac{|g_{i,x} - g_{j,x}|}{\sqrt{(g_{i,x} - g_{j,x})^2 + (g_{i,y} - g_{j,y})^2}} \tag{9.9}$$

$$D_{i,y}^j = \frac{|g_{i,y} - g_{j,y}|}{\sqrt{(g_{i,x} - g_{j,x})^2 + (g_{i,y} - g_{j,y})^2}} \tag{9.10}$$

where the directions of $D_{i,x}^j$ and $D_{i,y}^j$ are defined as from robot j to robot i along x-axis and y-axis, respectively.

Initially, the robots are located randomly in a 2D space. By following the dynamic equations defined in Equations 9.2 and 9.3, eventually multiple robots can be deployed uniformly on the predefined pattern automatically. In other words, the system can be stabilized to an equilibrium state defined by the pattern. Essentially, the pattern information is the global information, which can be elegantly embedded into the dynamics of each individual robot through function $f(\mathbf{z}_i)$.

In general, each robot is regulated by two forces: one is the morphogen gradient embedded in the regulation dynamics that drive the robot to the predefined pattern, and the other is diffusion dynamics that is used to avoid collision between the robots. A good balance of the two dynamics can be achieved by optimizing the parameters in the model using a genetic algorithm, which will be discussed in the next sub-section.

9.3.2 Convergence Analysis of System Dynamics

In this section, a theoretical proof of the systems convergence to the predefined shape using the proposed GRN based algorithm is provided. Considering Equations 9.2 and 9.3, since the system convergence property is the same for x-axis and y-axis, only the convergence of one axis is needed to prove. Furthermore, since summing up the diffusion forces of all the robots, the protein diffusion impact will counteract with each other, which allows us to omit the diffusion force in the proof of the systems convergence. For the sake of clarity, we rewrite Equation 9.2 and 9.3 and remove the protein diffusion part as follows:

$$\frac{dg}{dt} = -a \cdot z + m \cdot p \tag{9.11}$$

$$\frac{dp}{dt} = -c \cdot p + k \cdot f(z) \tag{9.12}$$

Where g, z, p, D can be either g_{ix}, z_{ix}, p_{ix}, D_{ix} or g_{iy}, z_{iy}, p_{iy}, D_{iy}.

Before proving the system convergence, we would like to propose and prove the following theorem first.

Theorem 9.1. $|f(x)| \leq |x|$ *for all* x.

$f(x)$ is a two sided sigmoid function with the analytical form $f(x) = \frac{1-e^{-x}}{1+e^{-x}}$, using basic mathematic deduction, we can easily get the following conclusions. (Please refer to the appendix for the proofs of the following two conclusions).

Conclusion 1: $|f(x)| \leq 1$
Conclusion 2: $0 \leq f(x) \leq 1$ Therefore, when $x = 0$, $f(x) = x = 0$, and when $x \neq 0$, from Integration Median Theory, we can get $f(x) = f'(\varepsilon) \cdot x$. Therefore, from conclusion 2, we can get $|f'(\varepsilon)| < 1$, so $|f(x)| = |f'(\varepsilon) \cdot x| \leq |x|$. We can get $|f(x)| \leq |x|$ and $|f(x)| = |x|$ only if $x = 0$.

According to the Lyapunov theory, the state vector in Equation 9.14 and 9.15, i.e. g and p, will converge to a stable vector asymptotically if we can find a Lyapunov function $V(g,p,t)$ which satisfies the following constraints:

1. $V(g,p,t)$ is positive definite;
2. $\dot{V}(g,p,t)$ is negative definite.

where g and p is the state vector in Equations 9.11 and 9.12.

There are four parameters, a, m, c, and k, that need to be adjusted. First, all of the parameters must be positive. Then we construct an energy function of the system dynamics in the following form:

$$V(g,p,t,) = h(g) + \frac{1}{2} s \cdot p^2 \tag{9.13}$$

Here, s is a positive variable whose definition will be provided later on. We follow the steps of Lyapunov theory to prove the system's convergence.

(1) $V(s) \geq 0$

(2)
$$\begin{aligned}
\tfrac{dV}{dt} &= \frac{dh}{dt} + s \cdot p \cdot \frac{dp}{dt} \\
&= \frac{\partial h}{\partial g} \cdot \frac{dg}{dt} + s \cdot p \cdot \frac{dp}{dt} \\
&= \frac{\partial h}{\partial g} \cdot (-a \cdot \frac{\partial h}{\partial g} + m \cdot p) + s \cdot p \cdot (-c \cdot p \cdot k \cdot f(\frac{\partial h}{\partial g})) \text{ since } z = \frac{\partial h}{\partial g} \\
&= -az^2 - c \cdot s \cdot p^2 + m \cdot p \cdot z + k \cdot s \cdot p \cdot f(z) \\
&\leq -a \cdot z^2 - c \cdot s \cdot p^2 + m \cdot |p| \cdot |z| + k \cdot s \cdot |p| \cdot |f(z)| \\
&\quad \text{(from Theorem 1, we can get} |f(z)| \leq |z|) \\
&\leq -\left(\sqrt{az} - \sqrt{c \cdot s \cdot p}\right)^2 - 2\sqrt{a \cdot c \cdot s} \cdot |z| \cdot |p| + (m + k \cdot s) \cdot |p| \cdot |z| \\
&= -\left(\sqrt{az} - \sqrt{c \cdot s} \cdot p\right)^2 + (k \cdot s - 2\sqrt{a \cdot c \cdot s} + m) \cdot |p| \cdot |z| \\
&= -\left(\sqrt{az} - \sqrt{c \cdot s \cdot p}\right)^2 + k \cdot \left[\left(\sqrt{s} - \frac{\sqrt{a \cdot c}}{k}\right)^2 - \frac{a \cdot c}{k^2} + \frac{m}{k}\right] \cdot |p| \cdot |z|
\end{aligned}$$

Let $s = \frac{a \cdot c}{k^2}$, we can get

$$\frac{dV}{dt} \leq -\left(\sqrt{a} \cdot z - \sqrt{c \cdot s} \cdot p\right)^2 + k \cdot \left(-\frac{a \cdot c}{k^2} + \frac{m}{k}\right) \cdot |p| \cdot |z|$$

So if $-\frac{a \cdot c}{k^2} + \frac{m}{k} \leq 0$,
which can be rewritten as $m \cdot k \leq a \cdot c$ to ensure $\frac{dV}{dt} \leq 0$.

After checking the invariant set which satisfies $\frac{dV}{dt} = 0$, we can see that the invariant set contain the points on the predefined shape. Thus we can conclude that the system's dynamics is convergent under the condition of $m \cdot k \leq a \cdot c$ and $k, c, a, m > 0$.

9.3.3 The Evolutionary Algorithm for Parameter Tuning

Besides achieving convergence, the system performance can be further evaluated with the following two objectives: the total traveling distance of all the agents and the system convergence time. This is a multi-objective optimization (MOO) problem, where the objective function is no longer a scalar value, but a vector. As a consequence, a number of Pareto-optimal solutions should be achieved instead of one single solution. In this paper, NSGA-II [5] has been adopted for evolution, which is a popular and efficient evolutionary algorithm for solving multi-objective optimization problems. In our work, simulated binary crossover (SBX) [3] and polynomial mutation [4] have been employed to generate offspring. After the offspring population is generated, the elitist crowded non-dominated sorting is used for selecting parents for the next generation. Different from single objective optimization algorithms, in which only one optimal solution is achieved, NSGA-II produces a set of Pareto-optimal solutions, i.e. in our case, the solutions that balance the convergence time and travel distance of the robots to the final formation. We will analyze the solutions in discussing the simulation results using NSGA-II.

9.4 Simulation and Results

To evaluate the efficiency and robustness of the proposed method, we test a sequence of case studies using MATLAB. Five parameters, a, m, c, k, and b in Equations (2) and (3), need to be optimized using the NSGA-II. The goal of the optimization is to minimize the robots travel distance and the convergence time, while assuring the system stability.

For the simulation, we set up the number of robots as 20, and set the population size for NSGA-II to be 100. The crossover probability is set to 0.9 and the distribution index for the SBX crossover is 20. Mutation probability is set to be inversely proportional to the number of the decision variables, which is 5 in our case, therefore, the probability is set to be 0.2 and the distribution index for mutation is set to be 20. The simulation runs for 50 generations. Initially, parameters k, c, a, and m are assigned with random numbers ranging from 1 to 100 and b is assigned with random numbers ranging from 200 to 1000. Here, we predefine a large value for protein diffusion coefficient b to allow the robots to move far away from each other so that they can select other target positions when robots are close to each other.

To optimize both the traveling distance and the convergent time, the following parameters are selected for the simulation after running the NSGA-II method: $a = 62.6459$, $m = 63.9489$, $c = 70.2715$, $k = 45.5208$, and $b = 380.4215$.

9.4.1 Case Study 1: Multi-robots Forming a Unit Circle

First, we conduct a set of experiments on a 2D environment to formulate a unit circle by multiple robots, as shown in Fig. 9.1. Robots are initially distributed randomly within the environment. The target pattern is defined as a unit circle centered at (0, 0) and of a radius of 1. A set of experiments using four different groups of robots has been conducted to evaluate the proposed method, which contains 5, 10, 15, and 20 robots, respectively. For each group, 35 independent runs have been performed. Fig. 9.1 shows a case of 20 robots being deployed on a unit circle. The experimental results of the average convergence time and average position error (average over the robots) are shown in Table 9.1 including both the mean and standard deviation. From Table 9.1, we can see that the average position error does not increase as the number of robots increases. However, the average convergence time becomes much slower as the number of robots increases. This is mainly because that robots need to be evenly distributed on the unit circle and more time has been spent on the fine tuning of the final positions.

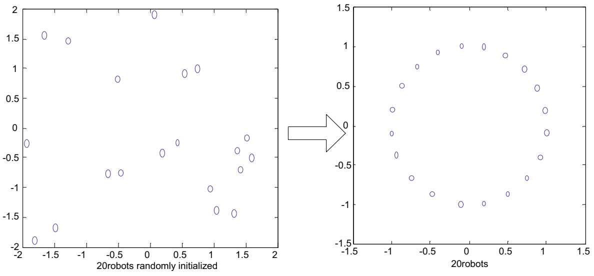


Fig. 9.1 20 robots are deployed uniformly on a unit circle.

Table 9.1 Mean and Standard Deviation of the Convergence Time and position Errors for 5, 10, 15, and 10 Robots in a Circle deployment.

Robots	Convergence Time	Position Errors
5	Mean: 229.3, STD: 30.7	Mean: 0.0913, STD: 0.0136
10	Mean: 530.3, STD: 75.6	Mean: 0.0784, STD: 0.0155
15	Mean: 969.3, STD: 94.8	Mean: 0.0372, STD: 0.0139
20	Mean: 1871.9, STD: 205.1	Mean: 0.0415, STD: 0.0147

9.4.2 *Case Study 2: Multi-robots Forming a Unit Square*

In this case, we plan to deploy a number of initially randomly distributed robots to a unit square. The unit square is defined as follows: lower-left point at (-0.5,-0.5) and the upper-right point at (0.5, 0.5). It is a little bit tricky to define the shape function of the unit square compared to the unit circle. We first set up a circle with the center at (0, 0) and radius of $\sqrt{\frac{1}{2}}$. Therefore, the shape function of a circle can be defined as: $s_1(g_{i,x}, g_{i,y}) = g_{i,x}^2 + g_{i,y}^2 - \frac{1}{2} = 0$ and $f_1 = (s_1(g_{i,x}, g_{i,y}))^2$. Then, through the first function, we can deploy the robots to the specific circle first, and then define $s_2(g_{i,x}, g_{i,y})$ according to the following rules:

$$s_2(g_{i,x}, g_{i,y}) = \begin{cases} (g_{i,x} + 1/2) & g_{i,x} \leq 0 & & -1/2 \leq g_{i,y} \leq 1/2 \\ (g_{i,x} - 1/2) & \text{if } g_{i,x} \geq 0 & \text{and} & -1/2 \leq g_{i,y} \leq 1/2 \\ (g_{i,y} + 1/2) & g_{i,y} \leq 0 & & -1/2 \leq g_{i,x} \leq 1/2 \\ (g_{i,y} - 1/2) & g_{i,y} \geq 0 & & -1/2 \leq g_{i,x} \leq 1/2 \end{cases} \tag{9.14}$$

$$f_2 = (s_2(g_{i,x}, g_{i,y}))^2 \tag{9.15}$$

Note that it is impossible that a robots position satisfies both of the two aforementioned conditions, since after the first phase deployment, the robots are on the circle and will satisfy only one condition as depicted in Fig. 9.2.

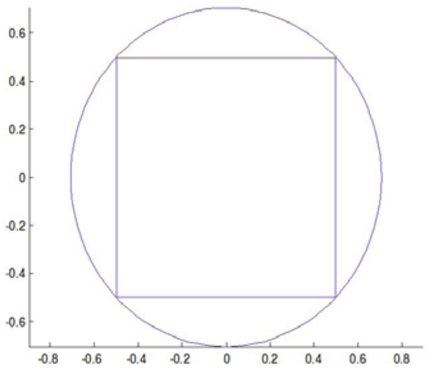


Fig. 9.2 The relationship between the circle and the expected square.

We have tested two groups of robots to deploy them to the unit square with 8 robots and 12 robots, respectively. Fig. 9.3 shows the simulation results of deploying 12 robots to a unit square. The simulation results for 8 robots are similar. Table 9.2 lists the simulation results on the average convergence time and average position errors. From Table 9.2, we can see that the average position error does not increase as the number of robots increases. However, the average convergence time becomes much slower as the number of robots increases. The reason for this has been discussed in case study 1.

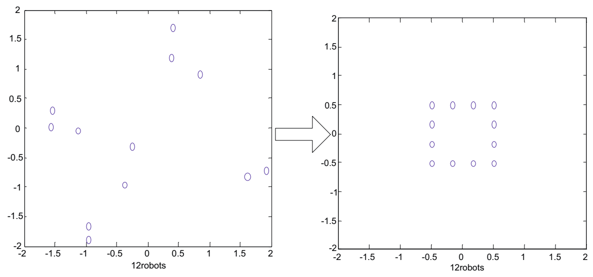


Fig. 9.3 12 robots are deployed uniformly on a unit square.

Table 9.2 Mean and Standard Deviation of the Convergence Time and position Errors for 8 and 12 Robots a Square deployment.

Robots	Convergence Time	Position Errors
8	Mean: 353.2, STD: 24.8	Mean: 0.0640, STD: 0.0116
12	Mean: 783.3, STD: 64.2	Mean: 0.0517, STD: 0.0097

9.4.3 Case Study 3: Self-reorganization

From previous two case studies, we can see that the proposed GRN-based control algorithm can automatically drive multiple robots to predefined shapes. In this case study, we would like to evaluate the systems capability of self-reorganization. More specifically, when new robots join the team, can the original team reorganize themselves to incorporate the newcomers? 9.4 provides the trajectories of robots initial shape formation as well as self-reorganization to incorporate the new comers. Fig. 9.4 demonstrates the self-reorganization capability of our proposed approach for various sizes of multi-robot systems.

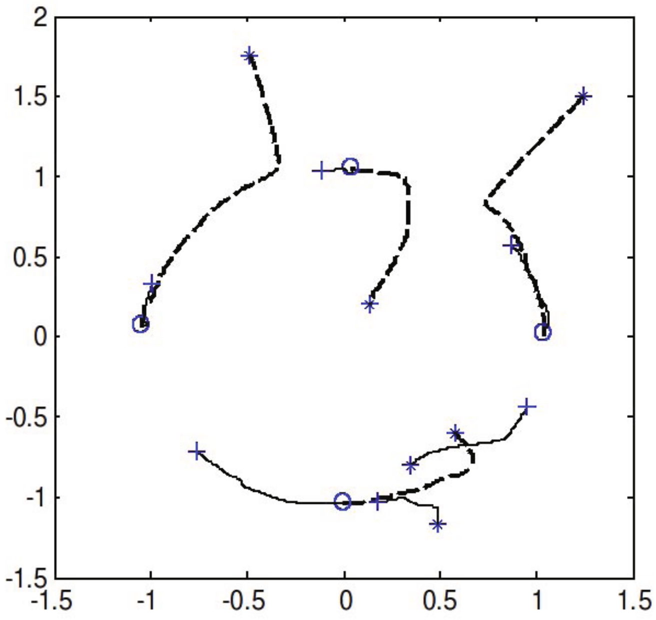


Fig. 9.4 Trajectories of 4 agents during self-organization with 2 newcomers. The initial positions of the robots are plotted as *, the intermediate states where the first batch of robots are located are plotted as o, and the final states of all the robots are plotted as +, the dash lines are the initial deployment trajectories of the first batch of robots and the solid lines are the trajectories of all the robots after incorporating newcomers.

9.4.4 Case Study 4: Robustness Tests to Sensory Noise

Since sensory noise is one of the constraints in the real-world robotic systems, In this case study, we will consider the sensory noise and evaluate the robustness of the proposed method to the sensory noise. We first consider the influence of sensory noise by deliberately adding noise into the sensory measurements and localization. We perform 35 independent runs using 10 robots with random position initialization, and calculate the mean and standard deviation of the final position errors to the unit circle with 5% and 10% noise in distance measurement. The experimental results are shown in Table 9.3. Then, we perform 35 independent runs using 10 robots with random position initialization, and calculate the mean and standard deviation of the final position errors to the unit circle with 5% and 10% noise in robot localization, and list the experimental results in Table 9.4. From the tables, we can conclude that the position errors of the system are insensitive to the noise in both distance measurements and self-localization.

Table 9.3 Mean and Standard Deviation of the Convergence Time and position Errors When the distance measurement are subject to sensory noise.

Without noise	5% noise	10% noise
Mean: 0.0421 STD: 0.024	Mean:0.0437 STD: 0.096	Mean: 0.0470 STD: 0.0100

Table 9.4 Mean and Standard Deviation of the Convergence Time and position Errors When the Self-Localization is subject to sensory noise.

Without noise	5% noise	10% noise
Mean: 0.0421 STD: 0.024	Mean:0.0437 STD: 0.096	Mean: 0.0470 STD: 0.0100

9.4.5 Case Study 5: Self-adaptation to Environmental Changes

In this case study, we want to evaluate the self-adaptation of the proposed method to the environmental changes.

It is assumed that the robots are evenly distributed on a circle initially. One moving obstacle in the environment moves toward the robots. Fig. 9.5 shows two snapshots of this procedure. From Fig. 9.5, we can see that the robots deployed on the circle are able to move temporarily away to avoid a collision with the moving obstacle. Although only one moving obstacle is implemented in our simulation, similar behaviors can also be observed for multiple moving obstacles.

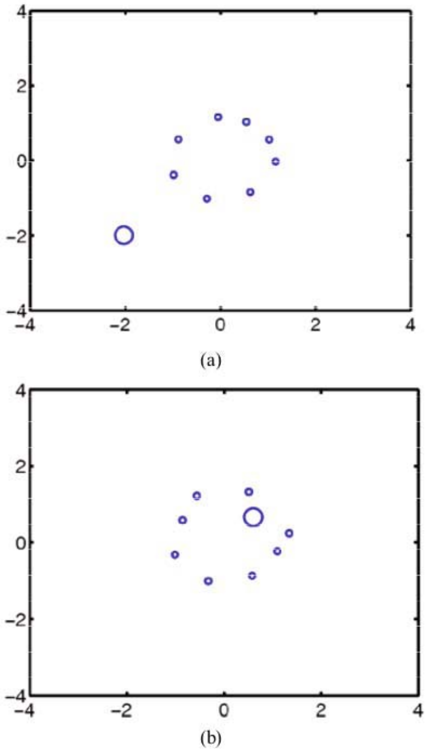


Fig. 9.5 A set of snapshots of robots behavior of reacting to a moving obstacle. (a) The obstacle is moving towards the robots; (b) the robots are adapting to avoid the obstacle.

9.5 Conclusion and Future Works

In this paper, we have presented a novel GRN inspired distributed control approach for a multi-robot system to self-organize themselves to some predefined patterns. A theoretical proof of the system convergence is also provided. Simulation results show the effectiveness, robustness and adaptation of the proposed algorithm. One major weakness of that model is that the global shape information is predefined and fixed in a centralized manner, which imposes serious limitations for those tasks with dynamic environment. In the future, we will investigate these issues and aim at developing new mechanisms on top of the current model, where the system has the ability to generate new global mission based on the dynamic environment autonomously and finish the mission accordingly.

References

[1] Albayrak, O.: Line and circle formation of distributed autonomous mobile robots with limited sensor range. Ph.D. dissertation, Naval Postgraduate School, Monterey, CA (June 1996)

[2] Alon, U.: An Introduction to Systems Biology: Design Principles of Biological Circuits. Chapman & Hall/CRC (July 2006)

[3] Deb, K., Agrawal, R.B.: Simulated Binary Crossover for Continuous Search Space. Complex Systems 2(9), 115–148 (1995)

[4] Deb, K., Goyal, M.: A Combined Genetic Adaptive Search (GeneAS) for Engineering Design. Computer Science and Informatics 4(26), 30–45 (1996)

[5] Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A Fast and Elitist Multiobjective Genetic Algorithm: NSGA-II. IEEE Transactions on Evolutionary Computation 6(2), 182–197 (2002)

[6] de Jong, H.: Modeling and simulation of genetic regulatory systems: a literature review. Journal of Computational Biology 9(1), 67–103 (2002)

[7] Desai, J., Ostrowski, J., Kumar, V.: Modeling and control of formations of nonholonomic mobile robots. IEEE Transactions on Robotics and Automation 17(6), 905–908 (2001)

[8] Endy, D., Brent, R.: Modeling cellular behavior. Nature 409, 391–395 (2001)

[9] Gierer, A.: Generation of biological patterns and form: some physical, mathematical, and logical aspects. Prog. Biophys. Mol. Biol. 37, 1–47 (1993)

[10] Guo, H., Meng, Y., Jin, Y.: Self-Adaptive Multi-Robot Construction using Gene Regulatory Networks. In: IEEE Symposium on Artificial Life (ALIFE 2009), Nashville, TN, USA, March 30- April 2 (2009)

[11] Hasty, J., McMillen, D., Isaacs, F., Collins, J.J.: Computational studies of gene regulatory networks: In numero molecular biology. Nat. Rev. Genet. 2, 268–279 (2001)

[12] Hsieh, M.A., Kumar, V.: Pattern Generation with Multiple Robots. In: Proceedings of the 2006 IEEE International Conference on Robotics and Automation, Orlando, Florida (May 2006)

[13] Kauffman, S.A.: The origins of order: self-organization and selection in Evolution. Oxford University Press, New York (1993)

[14] Kelly, K.: Out of Control – The New Biology of machines, Social Systems and Economic World. Basic Books, New York (1994)

[15] Maini, P.K., Painter, K.J., Nguyen, P.C.: Spatial pattern formation in chemical and biological systems. J. Chem. Soc., Garaday Trans. 93(20), 3601–3610

[16] McAdams, H.H., Arkin, A.: Simulation of prokaryotic genetic circuits. Ann. Rev. Biophys. Biomol. Struct. 27, 199–224 (1998)

[17] Meng, Y., Gan, J.: LIVS: Local interaction via virtual stigmergy coordination in distributed search and collective cleanup. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2007), San Diego, CA, USA (2007)

[18] Meng, Y., Gan, J.: A Distributed Swarm Intelligence based Algorithm for a Cooperative Multi-Robot Construction Task. In: IEEE Swarm Intelligence Symposium, St. Louis, Missouri (September 21-23, 2008)

[19] Salazar-Ciudad, I., Garcia-Fernandez, H., Sole, R.V.: Gene networks capable of pattern formation: from induction to reaction-diffusion. Journal of Theoretical Biology 205, 587–603 (2000)

[20] Shen, W., Will, P., Galstyan, A.: Hormone-Inspired Self-Organization and Distributed Control of Robotic Swarms. Autonomous Robots 17, 93–105 (2004)

[21] Suzuki, I., Yamashita, M.: Distributed anonymous mobile robots: Formation of geometric patterns. SIAM Journal on Computing (1999)
[22] Taylor, T.: A Genetic Regulatory Network-Inspired Real-Time controller for a Group of Underwater Robots. In: Proceedings of Eighth Conference on Intelligent Autonomous Systems, IAS-8 (2004)
[23] Turing, A.M.: The chemical basis of morphogenesis. Philos Trans. R. Soc. London B 237 (1952)

Chapter 10
Novelty-Based Multiobjectivization

Jean-Baptiste Mouret

Abstract. Novelty search is a recent and promising approach to evolve neuro-controllers, especially to drive robots. The main idea is to maximize the novelty of behaviors instead of the efficiency. However, abandoning the efficiency objective(s) may be too radical in many contexts. In this paper, a Pareto-based multi-objective evolutionary algorithm is employed to reconcile novelty search with objective-based optimization by following a multiobjectivization process. Several multiobjectivizations based on behavioral novelty and on behavioral diversity are compared on a maze navigation task. Results show that the bi-objective variant "Novelty + Fitness" is better at fine-tuning behaviors than basic novelty search, while keeping a comparable number of iterations to converge.

10.1 Introduction

Two recent papers [14, 21] introduced a new but radical approach to evolve robot controllers: they propose to maximize the novelty of behaviors *instead of* the efficiency (the fitness) of these behaviors. This method, called "novelty search", relies on a user-defined distance between behaviors and the NEAT [22] evolutionary framework to synthesize neural networks. It could have a least two benefits over traditional fitness-based evolutionary methods: (1) it could override the deceptiveness of some fitness functions and (2) the evolutionary process may be more open-ended.

Abandoning the efficiency objective(s) may be too radical in many contexts and especially in engineering processes – in which *solutions* to a *problem* are sought. If the search space is very large, novelty search can run for a very long time before finding any interesting solutions. In such situations, the objective is often the best guide to explore an interesting subset of the search space. Moreover, novelty

Jean-Baptiste Mouret
Université Pierre and Marie Curie – Paris 6, Institut des Systèmes Intelligents et de Robotique (ISIR), CNRS UMR 7222, F-75005, Paris, France
e-mail: `mouret@isir.upmc.fr`

search can not fine-tune results because the process is not aware that it should stop the exploration and begin a local search. Even if near-optimal solutions are found, the evolutionary pressure is kept towards finding individuals with a significantly different behavior; however, when the evolutionary process has almost found an optimal solution, we can reasonably expect it to cover the last small steps without any difficulty.

In this paper, a Pareto-based Multi-objective Evolutionary Algorithm (MOEA, see [4]) is employed to reconcile novelty search with objective-based optimization. Once defined a novelty objective *and* a fitness objective, the Pareto dominance relation can ensure that novel but inefficient candidate solutions will be selected, as in novelty search, but also that efficient but less novel ones will be considered equally interesting. As it will be demonstrated in this paper, this *multiobjectivization* – the transformation of a single-objective optimization problem to a multiobjective problem – has the potential to overcome the previously described limitations of novelty search.

To cope with the same limitations, an alternative to multiobjectivization is to switch from novelty search to fitness-based search once a criteria is met. We rejected this approach for three main reasons: (1) such a method would require to define an additional and hard to define switch criteria, (2) Pareto-based MOEA are well-studied algorithms that are preferable to custom ones and, (3) a MOEA "automatically" switches back and forth between fitness-based search and novelty search, according the context (in effect, it follows both approaches at the same time, but also all the trade-offs between them). From another point of view, novelty search is an exploration procedure and fitness-based algorithms are an exploitation procedure. Using a MOEA, we thus explicitly model the classic trade-off between exploration and exploitation (see e.g. [23]).

Hence this paper explores one main question: can novelty search be combined with objective-based search using a MOEA? Additionally, we also want to improve the knowledge of novelty search and especially how it compares to diversity maintenance techniques and whether the NEAT encoding is required.

To that aim, we prolong the maze experiments described in the original novelty search paper [14] by comparing them to our multi-objective method. We also compare these results to behavioral diversity [17, 18], another multiobjectivization [11, 13] that uses Pareto dominance to improve the exploration abilities of evolutionary algorithms.

10.2 Related Work

10.2.1 Novelty Search

Novelty search relies on three main concepts:

- a distance between *behaviors* of robots instead of a distance between genotypes as used in classical diversity preservation mechanism [8];

- a novelty measure, based on this distance and an archive of previously encountered behaviors;
- the search for novel behavior instead of the optimization of a fitness function.

Lehman and Stanley [14] measured the novelty $\rho(i)$ of an individual i by computing the mean behavioral distance between i and its k nearest neighbors:

$$\rho(x) = \frac{1}{k}\sum_{j=0}^{k} d_b(x, \mu_j)$$

where k is a user-defined parameter and μ_j is the j-th nearest neighbor of x with respect to the distance d_b. The neighbors are computed using the current population and an archive of all the previous novel individuals. An individual is added to the archive if its novelty is above a minimal threshold ρ_{min}. This last parameter is adjusted dynamically during an experiment:

- if more than 2500 evaluations occurred and no new individual were added in the archive, ρ_{min} is multiplied by 0.95;
- if more than 4 individuals were added during the same generation, ρ_{min} is multiplied by 1.05.

It is important to note that by using both the archive and the population, this nearest neighbors computation combine a diversity preservation mechanism (maximizing the novelty with respect to the current population) and a history-based search. It should also be noticed that, from a computational point of view, the time to find the k nearest neighbors grows continuously (at best logarithmically if a suitable data structure is employed).

10.2.2 Multi-Objective Evolutionary Algorithms

Recent research in evolutionary computation proposed numerous algorithms to simultaneously optimize several conflictual objectives without aggregating them (see [4]). Most of them rely on the concept of Pareto-dominance and generate the so-called Pareto Front:

Definition 10.1. A solution $\mathbf{x}^{(1)}$ is said to dominate another solution $\mathbf{x}^{(2)}$, if both conditions 1 and 2 are true:

1. the solution $\mathbf{x}^{(1)}$ is not worse than $\mathbf{x}^{(2)}$ with respect to all objectives;
2. the solution $\mathbf{x}^{(1)}$ is strictly better than $\mathbf{x}^{(2)}$ with respect to at least one objective.

The non-dominated set of the entire feasible search space is the globally Pareto-optimal set (Pareto front).

MOEA are designed to find an approximation of the Pareto-front, i.e. the set of all the Pareto-optimal trade-offs. Most of the time, they implement two mechanisms:

- a selection procedure that ensures that non-dominated individuals have a selective advantage;

- a diversity-preservation method designed to spread non-dominated individuals along the whole Pareto front, that is to ensure that as many different trade-offs as possible are explored.

A typical MOEA sorts individuals with respect to dominance. Non-dominated individuals may, for instance, be ranked 1, making them the most suitable for reproduction. Individuals which are only dominated by non-dominated ones may be ranked 2, and so on. The diversity on the Pareto front can be maintained using several ways such as fitness sharing [7], crowding [5] or clustering methods [25].

10.2.3 Multiobjectivization

While MOEAs have been primarily designed to optimize several conflicting objectives, a growing trends in the MOEA community is to apply them to improve the optimization of single-objective problems. The term "multiobjectivization" [10, 11, 13] has recently been coined to describe this cast of a single objective problem to a multiobjective one. Two paths can be followed to achieve this reformulation: adding new objectives or decompose the original objective function.

The decomposition of the main fitness function was studied in evolutionary robotics in a few papers, with very encouraging results. In [15] and [16], objectives were designed for each sub-task of a complex problem. This resulted in a fully automatic evolutionary process that can exploit sub-tasks without having to specify their order and when to switch between them. Moreover, several hypothesis about the usefulness of each sub-task were explored in parallel.

The addition of objectives were mostly investigated by adding a genotype-based diversity objective to the fitness [1, 3, 12, 24]. All the published papers about this topic concluded that adding this objective substantially improved the efficiency of evolutionary algorithms. Following this idea, [18], [6] and [17] proposed to employ a behavior-based distance to compute a diversity objective. It had, in particular, been employed to evolve neural networks to compute a Boolean function [18] and to evolve neuro-controllers for a mobile robot [6, 17]. According to this approach, called *behavioral diversity*, instead of maximizing the fitness $F(i)$, two objectives have to be maximized using a MOEA:

$$\begin{cases} F(i) \\ B(x) = \frac{1}{|P_n|}\sum_{j\in P_n} d_b(x,j) \end{cases} \tag{10.1}$$

where P_n denotes the population at generation n, $d_b(ij)$ the distance between the behaviors of individuals i and j, similar to the one used in novelty search.

10.3 Method

10.3.1 Experiment

To draw a fair comparaison with the original novelty search, we used the same maze navigation task as in [14]. In this task, a simulated mobile robot has to find a goal

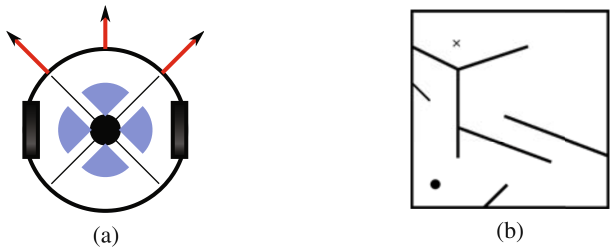


Fig. 10.1 (a) Overview of the simulated mobile robot. Each arrow outside of the robot's body is a range sensor (e.g. a laser range finder) that returns the normalized distance to the closest obstacle in that direction. The robot is also equipped with four pie-slice sensors that act as a compass towards the goal. This compass can "see" through walls. The robot is moved by two motors, the neural network setting the speed of each of them. (b) Map employed in this set of experiments (size: 512×512 units). The circle denotes the starting position and the cross denotes the goal.

zone in a maze by relying only on its three laser range sensors and a goal sensor (see Fig. 10.1(a)).

The robot is controlled by an evolved neural network (the topology and the synaptic weights are evolved) with 7 inputs (3 range sensors and 4 inputs for the goal sensor) and two outputs (the speed of the wheels). To keep our experiments simple and repeatable, we employed a typical graph-based direct encoding for neural-networks in which two kinds of mutations are possible[1]:

- structural mutation: addition/removal of a random neuron or a random connection;
- parametric mutation: change of a randomly chosen synaptic weight or a neuronal bias; we used here a change in a set of 9 possible values (see appendix).

Cross-over was not employed. This encoding has been used in several previous works [6, 17, 18] in which a basic encoding for neural-networks was needed.

This maze navigation task is interesting because maps with dead ends near the goal can be easily designed. This makes the typical fitness function (the opposite of the distance to the goal) very deceptive. Moreover, we can visualize the part of the maze explored by the robots and thus understand how the search space has been explored.

The map used in this paper (Fig. 10.1(b)) is similar to the "hard map" used by [14], with a very deceitful cul-de-sac in the direction of the goal.

[1] The NEAT [22] encoding could have been used, as in the original novelty papers. One argument to use NEAT is its incremental nature: the search begins with simple neural networks that are complexified during the evolution. This way of exploring topologies well suits the concept on novelty search in which simple behaviors are first explored. However, in this paper we would like to deal with the selective pressure with the simplest encoding as possible. Positive results with a basic graph-based direct encoding would ensure that novelty search does not critically depends on NEAT (which adds noticeably speciation and cross-over).

10.3.2 Fitness Function and Distance between Behaviors

The fitness $F(x)$ of an individual i (to be maximized) is the opposite of the minimum Euclidean distance to the goal obtained during the whole trajectory:

$$F(x) = - \min_{t=0}^{t=T} ||\mathbf{p_r}(t) - \mathbf{p_g}||$$

where $\mathbf{p_r}(t)$ is the position of the robot at time t, $\mathbf{p_g}$ the position of the goal and T the total time of one evaluation.

To describe the behavior of an individual x, we use the position of the robot *at the end of the experiment*:

$$\mathbf{b}(x) = \mathbf{p_r}(T)$$

The distance between individual i and j is then the Euclidean distance between their position at time T:

$$d_b(i,i) = ||\mathbf{b}(i) - \mathbf{b}(j)||$$

Many other distances between behaviors can be designed, such as measuring the difference between trajectories or exploiting the Kolmogorov complexity [9]. We will not explore these ideas in this paper in order to match the original novelty search algorithm.

10.3.3 Variants

To evaluate the contribution of a novelty-based multiobjectivization and improve the knowledge about novelty search, we investigate 6 variants of the experiment. Each of them aims at answering a particular question about novelty search.

1. *Control experiment.*
   Question: is this fitness really too deceptive for a classic evolutionary approach?

$$\text{Maximize } F(x)$$

2. *Novelty search (basic).*
   Question: can we confirm/infirm the published results about novelty search [14] without using NEAT?

$$\text{Maximize } \rho(x) = \frac{1}{k}\sum_{j=0}^{k} d_b(x, \mu_j)$$

   where μ_j is computed using the archive *and* the population.
3. *Novelty search (no diversity).*
   Question: is it necessary to compute novelty using both the archive and the current population?

$$\text{Maximize } \rho(x) = \frac{1}{k}\sum_{j=0}^{k} d_b(x, \mu_j)$$

   where μ_j is computed using the archive.

4. *Fitness + Novelty (MOEA).*
   Question: can novelty and fitness be reconciled with a MOEA?

$$\text{Maximize} \begin{cases} F(x) \\ \rho(x) = \frac{1}{k}\sum_{j=0}^{k} d_b(x, \mu_j) \end{cases}$$

   where μ_j is computed using the archive *and* the population.
5. *Fitness + Diversity (MOEA).*
   Question: how does novelty search compare to multiobjective behavioral diversity [17]?

$$\text{Maximize} \begin{cases} F(x) \\ B(x) = \frac{1}{|P_n|}\sum_{j \in P_n} d_b(x, j) \end{cases}$$

6. *Fitness + Diversity + Novelty (no diversity) (MOEA).*
   Question: can diversity and novelty be combined with a multi-objective approach instead of aggregating them?

$$\text{Maximize} \begin{cases} F(x) \\ B(x) = \frac{1}{|P_n|}\sum_{j \in P_n} d_b(x, j) \\ \rho(x) = \frac{1}{k}\sum_{j=0}^{k} d_b(x, \mu_j) \end{cases}$$

   where μ_j is computed using only the archive.

10.3.4 Expected Results

We expect the control experiments not to converge at all because of the deceitful local extrema. Following Lehman and Stanley [14], we can expect novelty search to work well to go close to the goal. However, it should encounter some difficulties to exactly reach the goal. “Novelty search + fitness” should overcome this problem. This last setup should not be substantially faster than novelty search because the fitness function does not provide a good guide to reach the goal. Nevertheless, adding the fitness objective to novelty search should not decrease the convergence speed. The three-objectives approach should be mostly equivalent to the two objectives “fitness + novelty” variant. However, MOEA give better results with two objectives than three [19, 20], consequently this variant may encounter some difficulties.

Given the good results observed with the addition of a diversity objective in previous experiments [6, 17, 18], we can reasonably expect a good convergence rate. The convergence speed may be slower because less data are employed to guide the search.

10.3.5 Experimental Parameters

To get statistically significant data, 30 runs of each variant were launched. NSGA-II [5], a popular and efficient MOEA, was used to optimize the objectives.

In the single objective case, it is equivalent to a tournament-based evolutionary algorithm. The detailed parameters of the direct encoding are available in appendix.

To compute the novelty, we used $k = 15$ (the same value as [14]) and $\rho_{\min} = 10$. Population size was 200 and each run lasted 1000 generation; put differently, our budget was of 200000 evaluations.

10.4 Results

10.4.1 Average Fitness

Figures 10.2 and 10.3 show the average (over the 30 runs) best fitness for each variant.

The single objective control experiments converged to a fitness of 60 that corresponds to the distance between the main dead-end and the goal. This means that most runs were trapped in the local maximum (20% of them found a controller that managed to drive the robot near the goal, see section 10.4.2).

Novelty search and multiobjective novelty search ("fitness + novelty") required about 200 generations to get all the runs close to the goal. Novelty search was slightly faster but only the multiobjective variant is able to reach the optimal value. This difference can be easily spotted on Fig. 10.2(b), which displays a zoom on the high fitness values.

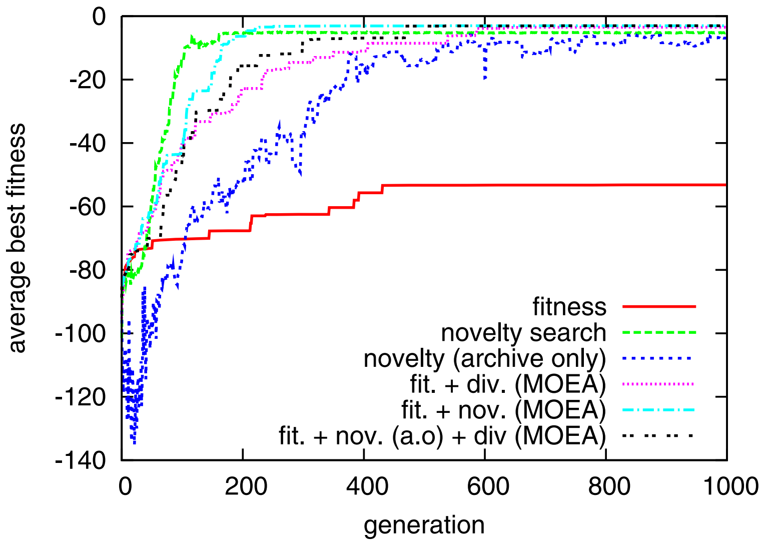


Fig. 10.2 Average maximum fitness in the population for each generation and for each variant (30 runs).

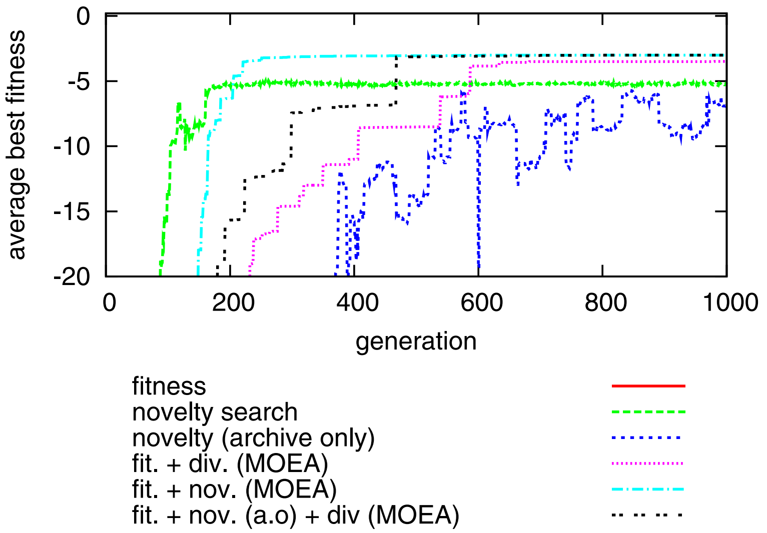


Fig. 10.3 Zoom of figure 10.2 on the best fitness values: the novelty-based multiobjectivization (fit + nov.) get the optimum fitness while the basic novelty search get close to it but never reach it.

If novelty is computed using only the archive, novelty search still converges a lot faster than the control experiment but it requires significantly more generations to converge. Surprisingly, it is also more noisy, suggesting that individuals close to the goal are often lost from the current population in favor of more novel ones. Last, the three objectives approach lead to results very similar to those obtained with the multiobjectivization “Novelty + Diversity”.

10.4.2 Convergence Rate

Considering that a run “converged” when one robot get at less than 10 units to the goal (the optimum is 3 units in these experiments; the choice of 10 is here related to ρ_{min} in novelty search), the convergence rate of each variant can be analyzed (Fig. 10.4 and Fig. 10.5). 29% of the control experiments (fitness only) converged in spite of the attractive local maximum. Nevertheless, 100% of the other experiments converged in less than 1000 generations (200000 evaluations). Novelty search and multiobjective novelty search are the fastest to converge while substantially more generations are needed for the novelty search variant that uses only the archive.

If we count as converged only the experiments that reach the optimum fitness (-3, Fig. 10.4(b)), only the novelty-based multiobjectivization runs get a 100% convergence rate whereas 90% of the novelty search runs converged.

To understand if the observed difference are statistically significant, the average generation to converge ($F(i) > -10$) has been computed (table 10.1). Unpaired Student T-tests (table 10.1) show that these results are statistically significant except

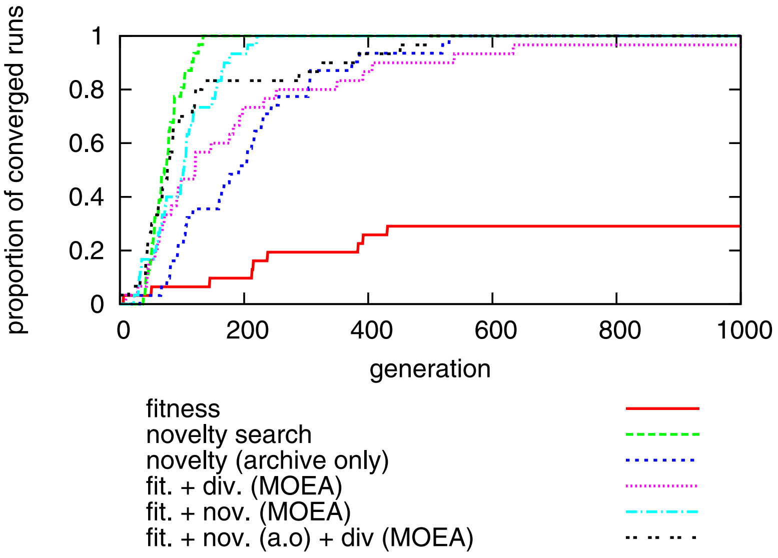


Fig. 10.4 Proportion of converged runs with regards to the generation number when considering that a run has converged when the best individual get at less than 10 units to the goal.

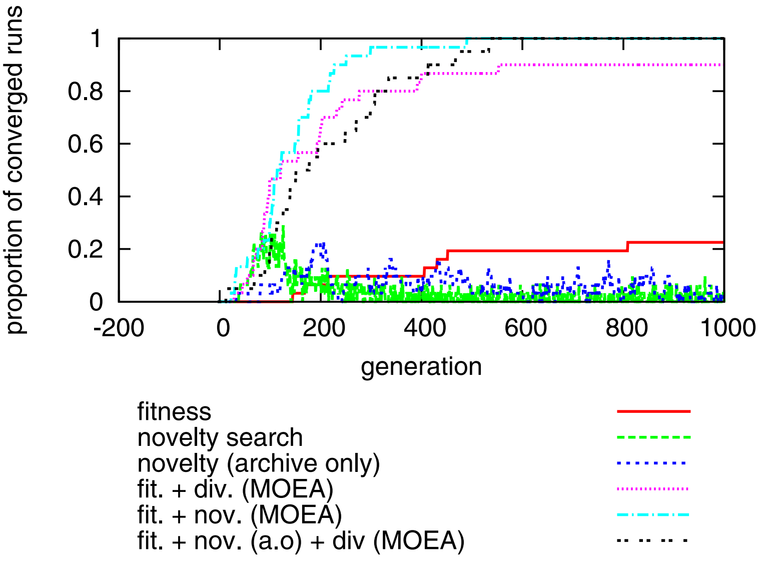


Fig. 10.5 Proportion of converged runs with regards to the generation number when considering that run has converged when the best individual get at less than 3 units to the goal (the optimum).

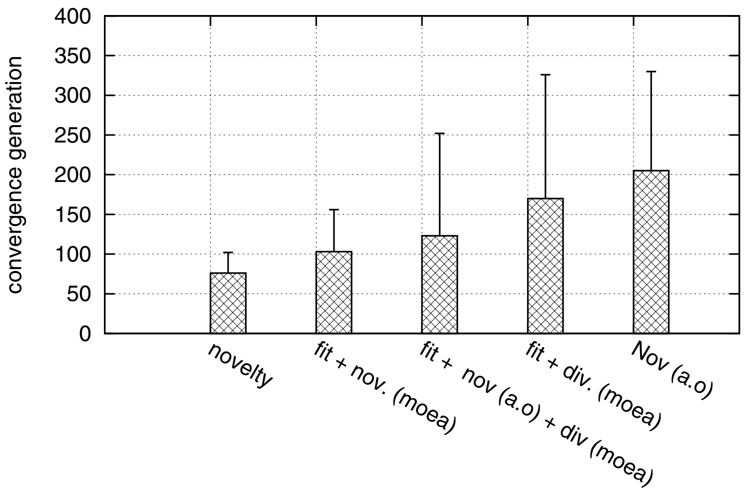


Fig. 10.6 Average generation to converge (i.e. $F(i) > -10$). Each generation is 200 evaluations.

Table 10.1 Average generation to converge (i.e. $F(i) > -10$) and results of Student's T-test. Each generation is 200 evaluations.

	Nov.	fit.+nov.	fit.+div.	Nov.(a.o)	f.+n.+d.
Aver. gen.	76	103	170	205	123
Std. dev.	26	53	156	125	129
T-test					
nov.	$p = 1.0$	$p = 0.01$	$p < 0.01$	$p < 0.01$	$p = 0.05$
fit.+nov.	$p = 0.01$	$p = 1.0$	$p = 0.03$	$p < 0.01$	$p = 0.34$
fit.+div.	$p < 0.01$	$p = 0.03$	$p = 1.0$	$p = 0.34$	$p = 0.20$
nov.(a.o.)	$p < 0.01$	$p < 0.01$	$p = 0.34$	$p = 1.0$	$p < 0.02$
f.+n.+d.	$p = 0.05$	$p = 0.34$	$p = 0.20$	$p < 0.015$	$p = 1.0$

between the "diversity + fitness" and the "novelty (archive only)" variants and between "fitness + diversity + novelty" and most of the other variants. Despite a small difference in average generation of convergence (25 generations/500 evaluations), novelty search appears statistically faster than the multiobjective variant. This small difference is easily understood by taking into account that, in the multiobjective variant, a part of the population is kept only because of its good fitness; the deceptiveness of the fitness makes these individuals almost useless to the algorithm. This drawback may be compensated by the better fitness obtained with the multiobjective variant (Fig. 10.2(b)). Results obtained with novelty search and multiobjective novelty search are of the same order of magnitude when they are compared to the "fitness only" runs: in both cases, only a few generations are required to converge.

A more complex problem, with a larger search space, could allow to draw a real efficiency distinction between these two approaches.

The diversity multiobjectivization was significantly slower to converge than the novelty variants. However, its computational cost is substantially lower than novelty search. To compute the diversity, only N^2 distances (N being the size of the population) need to be computed. With novelty search, the archive is an order of magnitude larger than the population (in our experiments, the typical archive size at the end of the experiments was about 1500 whereas the population size was 200).

10.4.3 Exploration

Figure 10.7 depicts the position at the end of the experiment of each evaluated until the first one reaches the goal. The first surprising observation is that, in all runs (successful or not), only the bottom left part of the maze has been explored. This can be explained by understanding where lies the difficulty of this task; two behaviors are the key to reach the goal: turning towards the right at the beginning of the experiment and efficiently avoiding obstacles. The turn of the bottom right is difficult to negotiate so if an individual manages to overcome this part of the maze, it is likely able to also come close to the goal. Since the fitness rewards the minimum distance to the goal during the whole experiment, robots don't have to adjust precisely their trajectory to get above the convergence threshold (10 units), they only have to wander near the goal.

The second surprising observation is that the explored points are not as different as suggested by the convergence rates. The points may be more evenly distributed when a novelty objective is used but the difference is not obvious. The multiobjectivization "fitness + novelty" (Fig. 10.7(b)) put a clear emphasis on the top left part of the maze, near the goal but in the dead-end, especially if it is compared to the original novelty search (Fig. 10.7(e)). However, it seems hard to predict which run will succeed using only the explored points.

This similarity may be interpreted in term of selective pressure: the key to converge is not to explore the good points but to recognize which point could lead to the solution. In the control experiment (Fig. 10.7(a), the robots that end at the bottom right part of the maze will be ranked last, because they are far away from the goal. However, with a novelty objective, the same behavior will be ranked first because there is almost no previous behaviors in this area. One may wonder why there is not many points in this part of the maze in the novelty experiments compared to the left part. A possible explanation is that such behaviors are hard to obtain, because a small variation of an individual with this behavior may more likely drive the robot in a wall near the starting point than in the same area. Consequently, it is only by persistently trying to modify the bottom right individuals that a successful individual can emerge. To test this hypothesis, one should evaluate how many children of such individuals reach the same area as their parents.

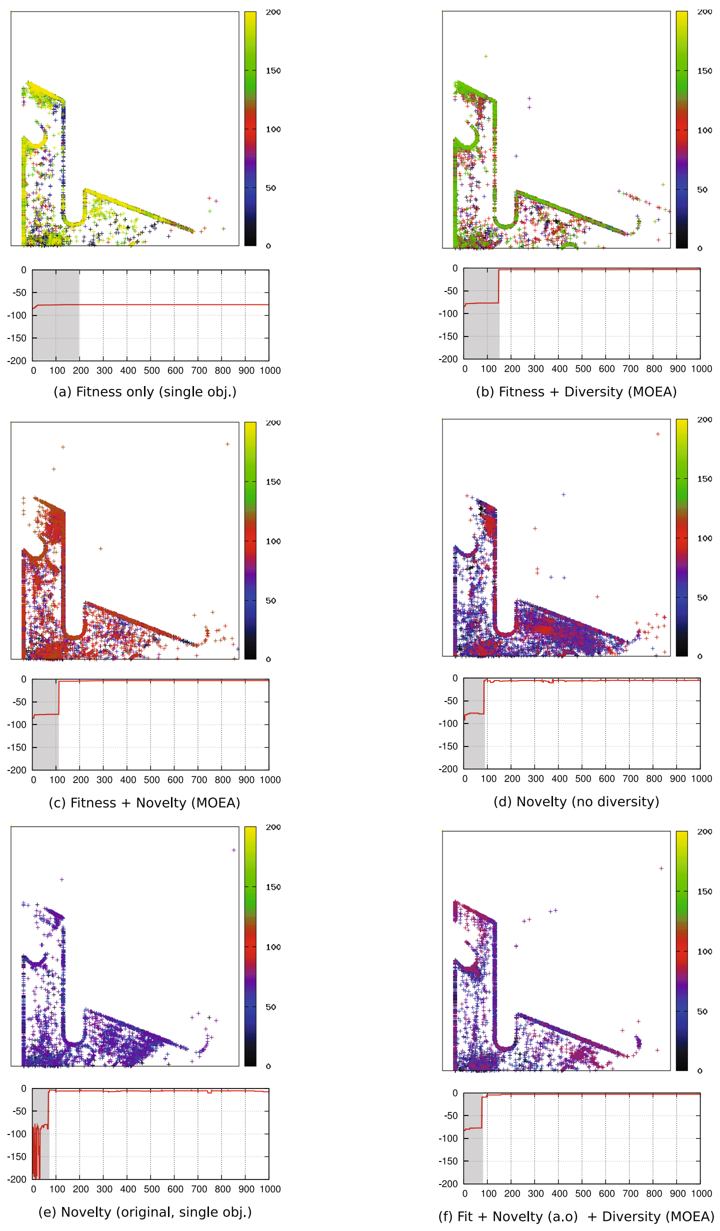


Fig. 10.7 (top) Position of each individual present at each generation until the first one reaches the goal (i.e. the robot is at less than 10 units from the goal), for a typical run of each variant. The control experiment (fitness only) did not converge so the first 200 generations are displayed. The color of a point denotes the last generation the corresponding individual has been part of the population. (bottom) Maximum fitness with regards to the generation number. The gray zone corresponds to the number of generations displayed in the top diagram.

10.5 Conclusion and Discussion

This paper investigated several ways to combine novelty search and objective-based search using a multiobjectivization. These methods were compared to behavioral diversity, a multiobjectivization that is also based on a distance between behaviors of candidate solutions.

The experimental results show at least that:

- novelty search is well adapted to the investigated deceptive task (this confirms the previously published results [14]);
- adding a fitness objective to novelty search does not harm the efficiency of the search;
- adding a fitness objective to the novelty objective allows to fine-tune the result more efficiently than with the original novelty search algorithm;
- diversity-based multiobjectivization requires more generations to converge but also lead to successful results in a few hundred of generations, whereas its computational cost is lower than novelty search;
- a straight-forward graph-based encoding leads to similar results than NEAT in this setup (at least qualitatively).

Overall, the successful experiments suggest that the difficulties encountered to evolve neural networks to solve complex tasks, especially observed in robotics, may currently be more a problem of good selective pressures than of bad encodings.

A more complex task, with a larger search space and/or with a less obvious dead-end, is needed to further compare novelty search to the multiobjectivizations introduced in this paper. The guide provided by the fitness in the maze experiment described here is deceptive by design but the success of evolutionary algorithms proves that fitness can be a good guide towards efficient solutions. We believe that the proposed multiobjectivization can combine the best of both worlds by using fitness when it is a good guide and novelty search when it is not.

The computational cost of novelty search compared to the one of behavioral diversity is a significant drawback of novelty search. One of the first idea to reduce the algorithmic complexity of novelty search is to efficiently structure the archive to compute the nearest neighbors as fast as possible. The typical structure used in computer graphics to this task is the kd-tree [2]; future work should investigate this idea. Another approach is to keep the size of the archive bounded by periodically removing the least significant elements.

References

1. Abbass, H.A., Deb, K.: Searching under multi-evolutionary pressures. In: Fonseca, C.M., Fleming, P.J., Zitzler, E., Deb, K., Thiele, L. (eds.) EMO 2003. LNCS, vol. 2632, pp. 391–404. Springer, Heidelberg (2003)
2. Bentley, J.L.: Multidimensional binary search trees used for associative searching. Communication of the ACM 18(9), 509–517 (1975)

3. Bui, L., Abbass, H.A., Branke, J.: Multiobjective optimization for dynamic environments. Proceedings of the IEEE-CEC 3, 2349–2356 (2005)
4. Deb, K.: Multi-objectives optimization using evolutionnary algorithms. Wiley, Chichester (2001)
5. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast elitist non-dominated sorting genetic algorithm for multi-objective optimization: NSGA-II. In: Proceedings of PPSN VI, pp. 849–858 (2000)
6. Doncieux, S., Mouret, J.B.: Single step evolution of robot controllers for sequential tasks. In: Proceedings of GECCO 2009, ACM, New York (2009)
7. Fonseca, C.M., Fleming, P.J.: Genetic algorithms for multiobjective optimization: formulation, discussion and generalization. In: Proceedings of the Fourth International Conference on Evolutionary Programming, pp. 416–423 (1993)
8. Goldberg, D.E., Richardson, J.: Genetic algorithms with sharing for multimodal function optimization. In: Proceedings of the Second International Conference on Genetic Algorithms, pp. 148–154. Morgan Kaufmann, San Francisco (1987)
9. Gomez, F.: Sustaining diversity using behavioral information distance. In: Proceedings of GECCO 2009 (2009)
10. Greiner, D., Emperador, J., Winter, G., Galván, B.: Improving Computational Mechanics Optimum Design Using Helper Objectives: An Application in Frame Bar Structures. In: Obayashi, S., Deb, K., Poloni, C., Hiroyasu, T., Murata, T. (eds.) EMO 2007. LNCS, vol. 4403, pp. 575–589. Springer, Heidelberg (2007)
11. Handl, J., Lovell, S.C., Knowles, J.: Multiobjectivization by decomposition of scalar cost functions. In: Rudolph, G., Jansen, T., Lucas, S., Poloni, C., Beume, N. (eds.) PPSN 2008. LNCS, vol. 5199, pp. 31–40. Springer, Heidelberg (2008)
12. de Jong, E.D., Watson, R.A., Pollack, J.B.: Reducing bloat and promoting diversity using multi-objective methods. In: Proceedings of GECCO 2001, pp. 11–18 (2001)
13. Knowles, J.D., Watson, R.A., Corne, D.W.: Reducing Local Optima in Single-Objective Problems by Multi-objectivization. In: Zitzler, E., Deb, K., Thiele, L., Coello Coello, C.A., Corne, D.W. (eds.) EMO 2001. LNCS, vol. 1993, pp. 268–282. Springer, Heidelberg (2001)
14. Lehman, J., Stanley, K.: Exploiting open-endedness to solve problems through the search for novelty. In: Proceedings of Artificial Life XI, pp. 329–336 (2008)
15. Mouret, J.B., Doncieux, S.: Incremental Evolution of Animats Behaviors as a Multi-objective Optimization. In: Asada, M., Hallam, J.C.T., Meyer, J.-A., Tani, J. (eds.) SAB 2008. LNCS (LNAI), vol. 5040, pp. 210–219. Springer, Heidelberg (2008)
16. Mouret, J.B., Doncieux, S.: Evolving modular neural-networks through exaptation. In: Proceedings of IEEE Congress on Evolutionary Computation, IEEE-CEC (2009)
17. Mouret, J.B., Doncieux, S.: Overcoming the bootstrap problem in evolutionary robotics using behavioral diversity. In: Proceedings of IEEE-CEC (2009)
18. Mouret, J.B., Doncieux, S.: Using behavioral exploration objectives to solve deceptive problems in neuro-evolution. In: Proceedings of GECCO (2009)
19. Praditwong, K., Yao, X.: How well do multi-objective evolutionary algorithms scale to large problems. In: Proceedings of IEEE-CEC, pp. 3959–3966 (2007)
20. Purshouse, R.C., Fleming, P.J.: On the Evolutionary Optimization of Many Conflicting Objectives. IEEE Transactions on Evolutionary Computation 11(6), 770–784 (2007)
21. Risi, S., Vanderbleek, S.D., Hughes, C.E., Stanley, K.O.: How novelty search escapes the deceptive trap of learning to learn. In: Proceedings of GECCO (2009)
22. Stanley, K.O., Miikkulainen, R.: Evolving neural networks through augmenting topologies. Evolutionary Computation 102(2), 99–127 (2002)

23. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
24. Toffolo, A., Benini, E.: Genetic diversity as an objective in multi-objective evolutionary algorithms. Evolutionary Computation 11(2), 151–167 (2003)
25. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. In: Evolutionary Methods for Design, Optimisation and Control with Application to Industrial Problems (EUROGEN 2001), pp. 95–100 (2001)

Parameters and Source Code

- MOEA: NSGA-II
- Population size: 400
- Neural network (direct encoding):
  - available weights / bias: $\{-2.0,-1.5,-1.0,-0.5,0.0,0.5,1.0,1.5,2.0\}$
  - proba. of weight/bias change (for each value): 0.1
  - number of inputs/outputs: 7/2
  - min./max. number of neurons (random generation): 10/20
  - min./max. number connections (random generation): 20/45
  - probability of adding/deleting a connection: 0.15/0.25
  - probability of changing a connnection: 0.1
  - probability of adding/deleting a neuron: 0.025/0.025
  - activation function for neurons: $y_i = \varphi\left(\sum_j w_{ij}x_j\right)$ where $\varphi(x) = \frac{1}{1+\exp(b-7x)}$
- source code: `http://www.isir.fr/evorob_db`

Chapter 11
Embedded Evolutionary Robotics: The (1+1)-Restart-Online Adaptation Algorithm

Jean-Marc Montanier and Nicolas Bredeche

Abstract. This paper deals with online onboard behavior optimization for a autonomous mobile robot in the scope of the European FP7 Symbrion Project. The work presented here extends the (1+1)-online algorithm introduced in [4]. The (1+1)-online algorithm has a limitation regarding the ability to perform global search whenever a local optimum is reached. Our new implementation of the algorithm, termed (1+1)-restart-online algorithm, addresses this issue and has been successfully experimented using a Cortex M3 microcontroller connected to a realistic robot simulator as well as within an autonomous robot based on an Atmel ATmega128 microcontroller. Results from the experiments show that the new algorithm is able to escape local optima and to perform behavior optimization in a complete autonomous fashion. As a consequence, it is able to converge faster and provides a richer set of relevant controllers compared to the previous implementation.

11.1 Introduction

Let's imagine an autonomous mobile robot tailored for exploration. This robot could be dropped in a wide variety of unknown environments, from a dense tropical forest to an exhausted gold mine abandoned 100 years ago. Even before starting to explore its environment, this kind of robot would need to be able to adapt to its immediate surrounding (i.e. figuring out what shape and/or what behavior is most fitted to sustain its energy level). In this set-up, the robot control architecture is first driven by the specific, unpredictable, properties of the environment.

This paper focuses on the design of a control architecture for an autonomous mobile robot in an unknown environment. For this type of set-up, design approaches can range from hand-crafted reactive behavior [3] to optimal control approaches [9].

Jean-Marc Montanier · Nicolas Bredeche
TAO - Univ. Paris-Sud, INRIA, CNRS
LRI, Bat. 490, F-91405 Orsay, France
e-mail: `forename.name@lri.fr`

The chosen approach depend on the problem to be solved. In the aforementionned situation, it is difficult, if not impossible, to a priori specify the environment and the task at hand. This implies that most of the existing approaches are not fitted. This is a typical problem in Robotics that may be addressed with learning and optimization[10, 21]. Moreover, we address the problem where little is known about the objective function. This means that the task is poorly described as a single efficiency measure (e.g. minimize energy consumption, maximize exploration, etc.), which is often delayed and noisy.

In this scope, Evolutionary Robotics provides optimization algorithms based on Evolutionary Algorithms. Evolutionary Robotics [9, 16] (”ER”) takes inspiration from nature by combining two partly antagonist mechanisms. On the one hand, *selection* of the most fitted individuals tends to ensure convergence of the algorithm toward the most fitted solution. On the other hand, *variation* over the properties of selected individuals through mutation and recombination tends to provide new original solutions. The general framework of these algorithms, termed Evolutionary Algorithms (”EA”), has been applied to a wide variety of problems [6]. In Evolutionary Robotics, EA is used as an optimizer for Artificial Neural Network architectures. Artificial Neural Network are used to control autonomous robots in a wide variety of task, from navigation (non-linear task) to swarm coordination (cooperation task)[1].

In ER, in order to compute the quality (or *fitness*) of a given genotype, the corresponding phenotype is created (e.g. an artificial neural network for robot control with optimized weights). This phenotype is *evaluated* in the environment to assess the performance of the resulting robot behavior. This evaluation methodology, is used on a *population* of *genotypes*. The (usually) better genotypes are selected and go through a variation process so as to renew the population. This process is then iterated until a termination criterion is matched (e.g. maximum number of evaluation, performance, etc.). This approach is usually referred as *off-line ER*.

While off-line ER can be used to address non-linear control problems (or poorly defined objective function), it fails to provide a continuous autonomous optimization process. This is because control over the initial conditions for genome evaluation is required. Therefore, either costly human interventation or the use of simulation [13] is needed. Moreover, evaluation of a genome requires reliability, ie. the fact that one evaluation session must be relevant with regards to the problem at hand.

Embodied ER, introduced in [8], is a sub-field of ER that precisely addresses the problem of changing environments without constant human manutention. In this setup, the Evolutionary Algorithm runs within the robot (or group of robots), acting as an embedded optimization algorithm. Embedded is defined as both online (the adaptation/learning process never stops) and onboard (the optimization algorithm and evaluation process are part of the control loop). To date, only few, but promising, works have adressed this topic: [7, 11, 12, 14, 15, 17, 20, 23, 24, 25, 26]. Running an embedded EA within a single robot provides strong advantages regarding

[1] See [10] for an introduction.

continuous adaptation and autonomy with regards to a human supervisor. However, this also emphasizes some specific issues:

- Unknown fitness landscape: the typical fitness landscape in ER is both multi-modal (many local minima) and partly neutral (many close genotype perform in a similar way). One reliable assumption (named Strong Causality) is that small variations in the genotypic space implies small variations in the fitness value [18]. A direct consequence is that any reliable algorithm should be able to perform both local search (to exploit this strong causality property) and global search (to avoid the pitfall of multi-modality);
- Evaluation reliability: the environmental condition vary over time depending on the robot location. Therefore, performance assessment (ie. fitness) of one genome might be completely different from one starting location to another (e.g. starting in a narrow bridge or starting in the middle of an empty arena). This is the problem of noisy fitness evaluation. A great number of independant evaluation are required to assess for the ”true” performance of one genome;

The (1+1)-online adaptation algorithm described in [4] has been shown to address these issues. Its ability to perform continuous adaptation efficiently has been demonstrated in the same paper[2]. The (1+1)-online algorithm is described as a genetic algorithm based on the (1+1)ES [19]. This algorithm uses only two genomes: a champion and a challenger. Some specific properties are employed so as to address online adaptation:

- **Local and global search :** A mutation operator is used to produce a child from a parent. This mutation operator is able to do both local and global search. A gaussian distribution $N(0,\sigma)$ is used. The switching between local and global search is done by modifying the value of σ. If this value is low, few modifications will be done to the genome, and the search will remain local. If the value of σ is high, the genome will be strongly modified, and the research will go global. On one hand, the value of σ is set to a low value when a new champion is found. This ensure a local search around a known good to solution in order to improve it. On the other hand, the value of σ is increased when the challenger is assessed worst than the current champion. This second mechanism ensure that the search will go more global if the current champion is in a local optima.
- **Re-evaluation :** Individuals may get lucky or unlucky during evaluation depending on the envrionment at hand. This is a typical problem related to fitness noise. An efficient solution is to reevaluate individuals, as proposed by Beyer [2].The reevaluated fitness overwrites the fitness of the champion. This is done to promote individuals with a low variance in their performances. One of the drawback of the overwriting method is that good individuals could be replaced by inferior but lucky individuals. If an individual is lucky during its first evaluation but has a low mean fitness it will not survive next-reevaluations. As a consequence, the evolutionary algorithm won’t be stuck with bad individuals.

[2] The demonstration was done on a single e-puck robot in Player/Stage, running a Cortex M3 micro-controller.

- **Recovery :** This work assumes the evolutionary algorithm should run without human intervention. It implies no repositioning of the robot after each evaluation of one individual. For example, a genome may be evaluated starting from completely different initial conditions, such as in front of a wall or in a tight corner. To avoid penalization of good genomes, a *recovery period* is introduced. During this time, the robot behavior is not considered for evaluation (ie. ”free of charge”), which favors genomes that display good performance whatever the starting position.

In this paper, we present an analysis of the global search feature of this algorithm. From this analysis, we identify a problem that negatively impacts the search. The basic idea is that the previous implementation of the (1+1)-online algorithm restrains, possibly drastically, the search space considered. This imply a limitation in the efficiency of the global search mechanism. This problem is described and a new algorithm, termed (1+1)-restart-online is devised. Preliminary experiments in simulation are described and show that the new algorithm actually performs a larger global search. Therefore the new algorithm, avoid the pitfall of getting stuck in a local optima for a long time. Moreover, this paper describes the implementation and successful evaluation of this new algorithm on a real robotic hardware set-up, a four wheels Bioloid mobile robot.

11.2 Extending the (1+1)-Online EA

This section shed some light on an intrinsic limitation of the (1+1)-online algorithm. Under very specific conditions (multi-modal objective function with few or no amount of noise), the adaptation process is slowed down. Then, an extension of the previous algorithm is described. This extension retains the properties of the original algorithm, and addresses the problem identified.

11.2.1 Limits of (1+1)-Online

In [4], the efficiency of the (1+1)-online algorithm has been shown. One of its main properties is to rely on a tunable gaussian mutation operator σ to switch between local and global search (see section 11.1). The update scheme of σ seems to be relevant in most cases, but it has a major drawback : only regions with better performing genomes can be considered. Figure 11.1 illustrates the shrinking effect of the search region considered. On this figure the fitness values of all genomes is shown (for the sake of simplicity, we assume this is a minimization task for a one dimension only problem). In this example, the current champion may be replaced *only* by a challenger which is *under* the dashed line. This holds for both local and global search. In this typical set-up, this isn’t a relevant strategy as the probability to randomly jump to the relevant region is very low. In comparison, it is more appealing to pick a genome from which local search may slowly but surely, lead to the best genome.

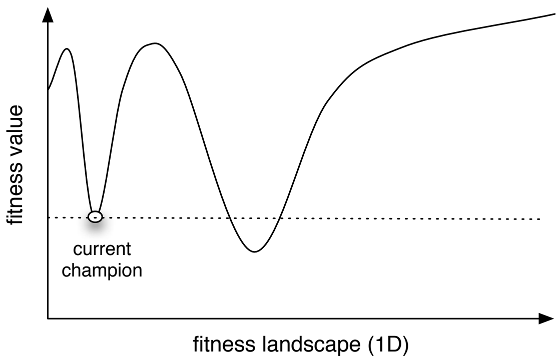


Fig. 11.1 Deceiving fitness landscape (minimization task).

The modification of σ is a good candidate to find new individuals. When it is increasing the search goes more global. But at some point the search area is too constrained. It becomes more interesting to simply restart the whole optimization process in order to obtain an unconstrained global search. To some extent, this problem may not occur in all situations. Firstly, this problem would never occur when optimizing a convex objective funtion, which is unfortunately quite scarce in this set-up. Secondly, a very noisy objective function may cope with this problem. That is because any good performing individual may eventually be re-evaluated with a low fitness value. Therefore the whole search space will be considered all over again – this was indeed the case in the experiments shown in [4].

11.2.2 The (1+1)-Restart-Online Algorithm

Escape from local minimum is a classical problem for the global search algorithms, and has been studied in different fields. A popular method is the restart strategy as used in some state of the art Evolution Strategies algorithm [1]. With this approach, the algorithm is restarted, either with similar or different parameters, whenever the search is identified as stalled. This approach provides interesting results on multi-modal problems as it ensures global convergence (ie. the algorithm tends to explore the whole search space through random search, as it is exactly what restarting is about).

In order to implement restart in the $(1+1) - restart - online$ algorithm, the restart criterion has to be considered. Options are mostly limited to the two following:

- **Monitoring the value of σ :** If σ reaches a maximal predefined value, a local minimum has been attained and the search is going global. To be sure that the algorithm will never be blocked in a local minimum, it can be restarted as soon as σ reaches its maximal value.

- **Limiting the number of champion reevaluations:** Whenever a champion isn't replaced, no better individuals have been found in its neighborhood. Thus, surviving many re-evaluations assesses the robustness of the champion with regards to both other challenger genomes and to the environment. Therefore, a high number of re-evaluations can be used to detect a good performing genome, but also that the search is stalled.

However, some issues remain to be addressed. On the one side, relying on the value of σ alone to restart the algorithm is too constraining: the maximum value for σ may be reached even if the champion was not reevaluated yet (ie. the champion may be unreliable). Moreover, even if the current champion has been successfully reevaluated while σ was increasing, it may still be improved by mutations. On the other side, if the champion survives many reevaluations, it is a good and reliable individual that will be hard to replace.

As a consequence, our implementation is to consider the number of reevaluations as a restart criterion in the (1+1)-restart-online algorithm described by algorithm 1. Hence, whenever restart is triggered, the algorithm is re-initialized, storing the current champion in a hall-of-fame archive, and setting a random genome as the new starting point. In the long term, this tends to converge towards a uniform sampling of the genotypic space.

11.3 Experiments and Results

This section presents the experimental set-up used to evaluate the performance of the (1+1)-restart-online algorithm. Results and preliminary experiments are also described and discussed.

11.3.1 Hardware Set-Up

The experimental evaluation has been conducted using a popular robotic microcontroller: a Cortex M3 board with 256 kb memory. The Cortex board runs a robot simulated by Symbricator3D, a physics-based 3D simulator developped within the Symbrion project and based on delta3d[3] (An Open Source game engine which can be used for physics-based simulations). After N time-steps, the evaluation of the current controller is complete. The controller parameters are replaced with values from a new genome. As described in the previous section, the new genome is evaluated from the location the previous controller left it in.

Figure 11.2 illustrates the experimental set-up with a Cortex board connected to the computer running the simulator based on delta3d. The simulated robot is equiped with two screws and 8 distance sensors (two per side). Details of the shape of the robot can be seen in figure 11.3. The maze environment and the cortex board are shown in figure 11.2.

[3] http://www.delta3d.org/

Algorithm 1. The (1+1)-RESTART-ONLINE evolutionary algorithm.

```
for $evaluation = 0$ to $N$ do
  if random() $< P_{reevaluate}$ then
    if $reevaluation_{count} < reevaluation_{max}$ then
      Recover(Champion)
      $Fitness_{Champion}$ = RunAndEvaluate(Champion)
      $reevaluation_{count} = reevaluation_{count} + 1$
    else
      $\sigma = \sigma_{min}$
      Champion = RandomGenome()
      $Fitness_{Champion} = 0$
      Challenger = RandomGenome()
      $Fitness_{Challenger} = 0$
      $reevaluation_{count} = 0$
    end if
  else
    Challenger = Champion $+ N(0, \sigma)$ $\{$Gaussian mutation$\}$
    Recover(Challenger)
    $Fitness_{Challenger}$ = RunAndEvaluate(Challenger)
    if $Fitness_{Challenger} > Fitness_{Champion}$ then
      Champion = Challenger
      $Fitness_{Champion} = Fitness_{Challenger}$
      $\sigma = \sigma_{min}$
    else
      $\sigma = \sigma \cdot 2$
    end if
  end if
end for
```

The robot is controlled by a simple perceptron with 9 input neurons (8 IR distance sensor values and one bias neuron) and 2 output neurons (translational and rotational velocities, which are combined to give actual actuator values).

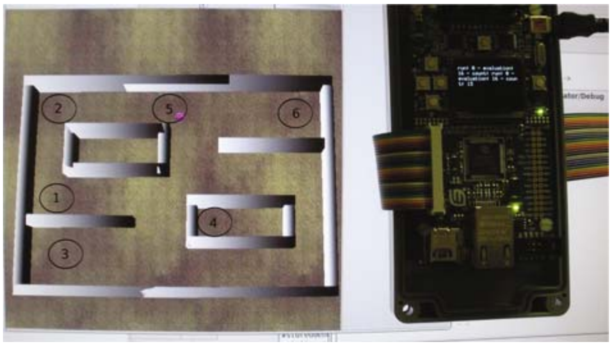


Fig. 11.2 The experimental set-up: the Cortex M3 board connected to Symbricator3d. The numbers show the reference starting positions for validating the Hall of Fame.

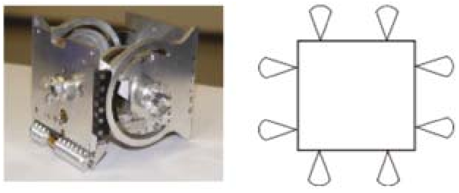


Fig. 11.3 Details of the Symbricator robot. (a) robot design (b) position of distance sensors (from above).

11.3.2 Experimental Set-Up

The objective function used is inspired from a well-known and classic fitness function first described in [16]:

$$fitness(x) = \sum_{t=0}^{n} V_t * (1 - V_r) * (1 - D_M)$$

where V_t is the translational speed, V_r is the rotational speed, and D_M the value of the most active distance sensor. All values are normalized between 0 and 1. More details about the parameters setting are given in appendix I. Distance sensors returns higher value when they are close to a wall. Therefore, individuals achieve high fitness value while moving fast and forward and avoiding walls.

The (1+1)-restart-online algorithm has been evaluated with a restart parameter fixed at 7 reevaluations. In order to compare the true performances of individuals obtained with (1+1)-online and (1+1)-restart-online, one Hall-of-Fame per experiment is computed from the results of the simulations. A Hall-of-Fame contains the best individuals (ie. the genome champions) from a given experiment. Champion genomes are ranked thanks to the sum of the re-evaluated fitness obtained during the experiments (ie. the larger the fitness value and the larger the number of re-evaluations, the better genome).

As the adaptation process could go on forever, the maximal number of evaluations is fixed to 600 for both experiments. Afterwards, the following experimental protocol is used to compare the best individuals from the Hall-of-Fames: every champions from the two Hall-of-Fames are evaluated from 6 predefined starting positions[4] shown in figure 11.2 and each evaluation lasts 120 time steps (ie. long enough to evaluate the quality of behaviors in a wide range of situations). The motivation of this validation protocol is to provide fair comparison between genomes.

[4] The starting position number 4 is an extreme case where the robot is tested in a hard environment never seen before.

11.3.3 Experimental Results

Figure 11.4 shows the course of evolution during a critical run of the (1+1)-online algorithm. Evaluations are denoted on the x-axis. The y-axis is divided in two parts. The top half shows the fitness of the current champion in green dashed line. The bottom half shows the number of re-evaluations of the current champion (downwards; the lower the line, the higher the number of re-evaluations). The red vertical markers near the x-axis indicate whenever a new champion is employed, i.e., when the challenger outperforms the current champion. During this run a good champion has been found at evaluation 180, and hasn't been replaced until evaluation 538 (after 64 reevaluation). Due to the use of another simulator this problem hasn't been detected in [4]. Differences between the two simulators are important with regards to noise between evaluations. This is a typical illustration of the problem identified in this paper with the original (1+1)-online algorithm: a less noisy set-up is prooved to be deceitful for the original algorithm.

Figure 11.5 shows evolution dynamics of a run of the (1+1)-restart-online algorithm. On this run the two main features of this algorithm – reevaluation and restart – are displayed. The reevaluation procedure is used at evaluation 132 on a lucky individual found at evaluation 126. After this reevaluation the champion obtain a low fitness and is soon replaced. In this run the restart procedure is used at evaluation 368, to replace a robust champion. According to preliminary experiments, the champion of evaluation 368 could still be improved. This imply that the restart strategy could be triggered later.

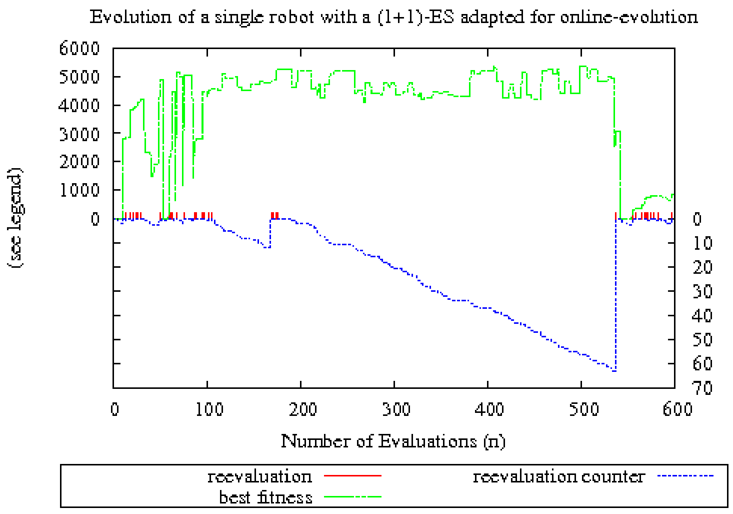


Fig. 11.4 Evolutionary dynamics of a critical run of the (1+1)-online algorithm.

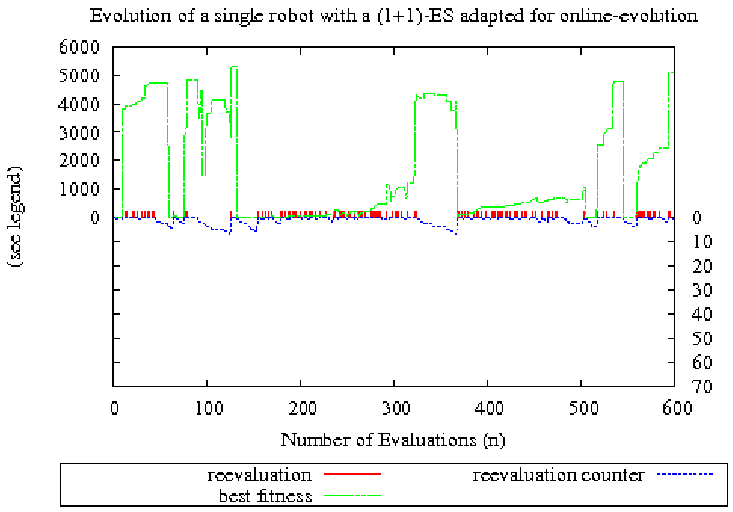


Fig. 11.5 Evolutionary dynamics of a run of the (1+1)-restart-online algorithm.

11.3.4 Hall-of-Fame Analysis

As described in section 11.3.2, two Hall-of-Fames were extracted from the results of the experiments. Each Hall-of-Fame is computed out of 14 independant runs of 600 evaluations. There are 1691 individuals in the Hall-of-Fame obtained by running the (1+1)-online algorithm. In comparison, 2158 individuals are in the Hall-of-Fame obtained by running the (1+1)-restart-online algorithm. This difference is a desired effect of the (1+1)-restart-online algorithm. The restart feature favors exploration by saving unnecessary reevaluations of champions whenever the algorithm is stalled. As a consequence, the $(1+1)-restart-online$ provides faster (in term of number of evaluation) the same number of Hall-of-Fame indivudals.

Performances of the best individuals generated by the (1+1)-restart-online algorithm and by the (1+1)-online algorithm are compared. As described in section 11.3.2, every individuals from the Hall-of-Fames are evaluated from six pre-defined positions (i.e. to provide comparable test cases). For each individual the mean performance obtained from those 6 positions has been computed. The figure 11.6 reports the distribution of individuals with respect to their fitness. The x-axis shows the different fitness obtained during the validation of the 628 best individuals of each Hall-of-Fame. The y-axis shows the number of individuals with the same fitness. It is clear that there is no loss of efficiency with the (1+1)-restart-online algorithm.

Therefore, the $(1+1)-restart-online$ algorithm is as reliable as the $(1+1)-online$ and faster - which is a key feature whenever ressources are limited.

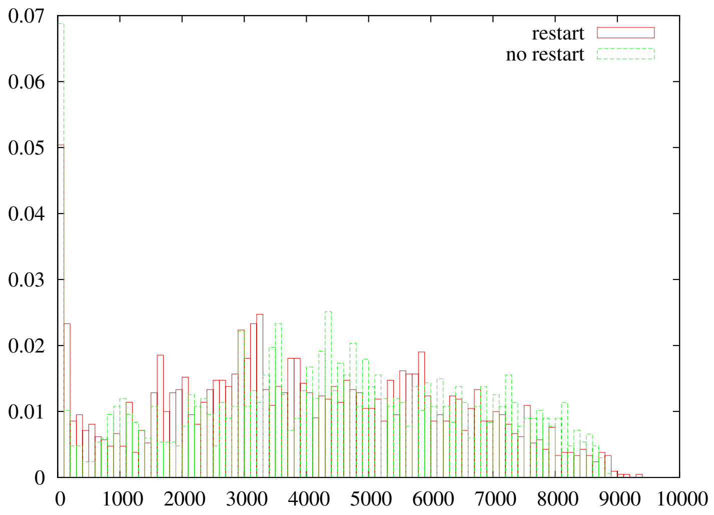


Fig. 11.6 Fitness density of the best individuals produced by the (1+1)-online algorithm, and the (1+1)-restart-online algorithm.

11.3.5 Real Robot Experiment

The (1+1)-restart-online has been tested on an autonomous four-wheels Bioloid robot. The Bioloid kit provides robotic parts and an ATmega128 microcontroller with 128Kb of memory[5]. Figure 11.7 shows the robot used in this work. It is equiped with 4 motors, and 7 distance sensors. The 7 red arrows in figure 11.7 shows the orientations of the distance sensors. The controller of the robot is a feedforward neural network with 8 inputs (7 distance sensors and 1 bias) and 2 outputs (left and right motor velocities). The two left wheel velocities are controled by the same output neuron, and the two right wheel velocities by the other output neuron. The fitness function used is the same as the one described in section 11.3.2. Each individual is recovering during 60 time step (7 seconds) and is evaluated during the 60 following time step (7 seconds). As in section 11.3.2 the restart threshold is fixed to 7 re-evaluations. The whole experiment lasted 1 hour and 10 minutes, which was more than sufficient to get examples of robust behaviors.

Figure 11.7 (b) shows the experimental set-up.

The algorithm provides similar results to what has been already shown in the previous experiment. The behavioral traces of the first two best evolved controllers from the Hall-of-Fame are illustrated in figures 11.8 and 11.9. These two control architectures efficiently avoid walls with simple yet efficient behaviors. The best controller (figure 11.8) is faster when moving in straight line, and displays sharper

[5] `http://www.atmel.com/dyn/products/product_card.asp?part_id=2018`

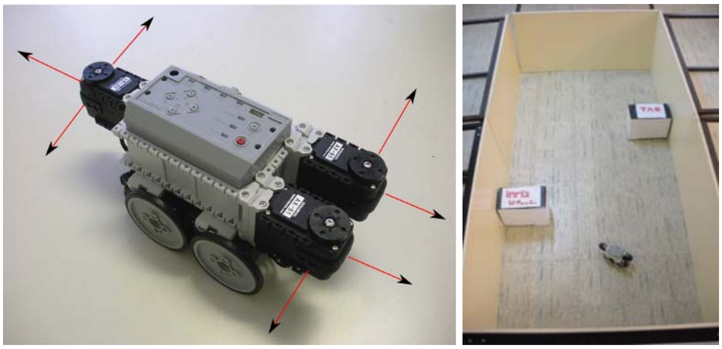


Fig. 11.7 (a) The robot and the directions of the 7 distance sensors, (b) the environment.

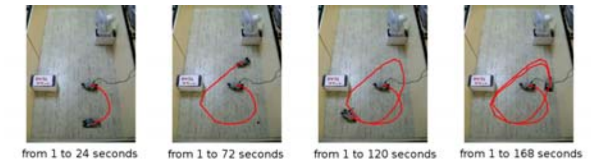


Fig. 11.8 Example of behavior for the best evolved controller.

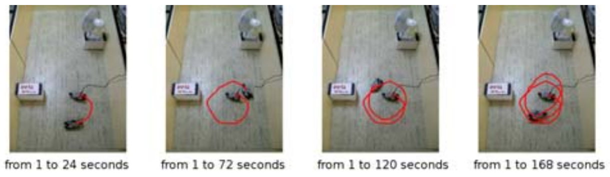


Fig. 11.9 Example of behavior for the 2nd best evolved controller.

turn trajectories. Other genomes have been empirically evaluated (not shown here) and display the same kind of behaviors as these two, with minor differences concerning the sharpness of the turn and/or the speed of the robot.

An important feature of our algorithm is also demonstrated here : the online nature of the algorithm makes it possible to easily avoid the reality gap [13]. Indeed, the algorithm needed exactly the same amount of work from the experimenter in simulation and reality. Moreover, *neither* human intervention *nor* external remote control was ever needed during the whole experiment with the real robot. Of course, this assumption must be taken with care as the fitness considered here is a rather simple one, chosen to focus on the validation of the algorithm features rather than on its ability to solve a complex problem.

11.4 Conclusion and Perspectives

In this paper, the problem of online onboard behavior adaptation for a single autonomous mobile robot has been adressed. Precisely, the (1+1)-online adaptation algorithm presented in [4] is studied. A limitation of this algorithm is identified and analysed regarding its ability to perform global search in the space of possible solutions. A new algorithm is described, termed (1+1)-restart-online. It is shown to efficiently address the trade-off between local and global search, by relying on a restart procedure whenever the algorithm is stuck in a local optimum. This restart procedure makes it possible to address a previous design flaw by relaxing some of the required constraints over the search space considered.

This algorithm has been evaluated both with real robotic hardware connected to a robot simulation as well as with a real robot. Results obtained have demonstrated that this new algorithm is actually able to provide a wider exploration of the search space, making it possible to visit many more local optima than previous implementation. Therefore, the probability to end up in a global optimum is increased, but also the diversity of obtained candidate solutions is increased. Moreover, this algorithm can be straight-forwardly used within a real robot in a complete autonomous fashion, providing a key-feature to relieve the expert from unnecessary and fastidious control over the experiment.

Perspectives from this work first concerns a careful study of the experimental parameters and have already been started in [5]. Also, an in-depth analysis of the distribution of the performance from all individuals in the Hall-of-Fame should be conducted. Moreover, the new restart feature in the algorithm must be carefully studied as there exists a possible trade-off in balancing the previous global search strategy and the new restart strategy. This trade-off can be reformulated as favoring global search over avoiding re-convergence towards already visited local optima. As a consequence, choosing between the two strategies clearly depends on both the shape of the fitness landscape and the actual local minimum.

Future works will also address the problem of noisy fitness evaluation by extending the $(1+1)$ stragegy into a variation of more complex strategies, from building challenger genomes out of family of successful genomes or distribution-based estimation of the relevant genotypic regions to explore (as in estimation of distribution algorithms). This roughly means that a reservoir, or a distribution, of champion genomes will be considered rather than only a single champion genome in order to build new candidate genome to be evaluated. Also, the extension towards multi-robots is an interesting perspective: one may consider the current adaptation algorithm to act within one island of a distributed evolutionary algorithm. In this set-up, each robot/island runs an embodied evolutionary adaptation algorithm. Additionally, the best genomes may be exchanged from one island to another, as in the well-known GA island model [22].

Acknowledgments

This work was made possible by the European Union FET Proactive Intiative: Pervasive Adaptation funding the Symbrion project under grant agreement 216342.

References

1. Auger, A., Hansen, N.: A restart cma evolution strategy with increasing population size. In: Proceedings of the IEEE Congress on Evolutionary Computation, CEC 2005 (2005)
2. Beyer, H.G.: Evolutionary algorithms in noisy environments: Theoretical issues and guidelines for practice. In: Computer Methods in Applied Mechanics and Engineering, pp. 239–267 (1998)
3. Braintenberg, V.: Vehicles: Experiments in Synthetic Psychology. MIT Press, Cambridge (1986)
4. Bredeche, N., Haasdijk, E., Eiben, A.: On-line, on-board evolution of robot controllers. In: Collet, P., Monmarché, N., Legrand, P., Schoenauer, M., Lutton, E. (eds.) EA 2009. LNCS, vol. 5975, pp. 110–121. Springer, Heidelberg (2010)
5. Eiben, A., Haasdijk, E., Bredeche, N.: Symbiotic Multi-Robot Organisms: Reliability, Adaptability, Evolution. In: Levi, P., Kernbach, S. (eds.) Symbiotic Multi-Robot Organisms. Cognitive Systems Monographs, vol. 7, pp. 361–382. Springer, Heidelberg (2010)
6. Eiben, A.E., Michalewicz, Z. (eds.): Evolutionary Computation. IOS Press, Amsterdam (1998)
7. Elfwing, S.: Embodied Evolution of Learning Ability. PhD thesis, KTH School of Computer Science and Communication, SE-100 44 Stockholm, Sweden (November 2007)
8. Ficici, S., Watson, R., Pollack, J.: Embodied evolution: A response to challenges in evolutionary robotics. In: Wyatt, J.L., Demiris, J. (eds.) EWLR 1999. LNCS (LNAI), vol. 1812, pp. 14–22. Springer, Heidelberg (2000)
9. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary robotics. In: Siciliano, B., Khatib, O. (eds.) Handbook of Robotics, pp. 1423–1451. Springer, Heidelberg (2008)
10. Floreano, D., Mattiussi, C.: Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies. In: Intelligent Robotics and Autonomous Agents. MIT Press, Cambridge (2008)
11. Floreano, D., Schoeni, N., Caprari, G., Blynel, J.: Evolutionary Bits'n'Spikes. In: Standish, R.K., Beadau, M.A., Abbass, H.A. (eds.) 8th International Conference on the Simulation and Synthesis of Living Systems (Alife 8). MIT Press, Cambridge (2002)
12. Haroun Mahdavi, S., Bentley, P.J.: Innately adaptive robotics through embodied evolution. Auton. Robots 20(2), 149–163 (2006)
13. Jakobi, N., Husband, 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, Springer, Heidelberg (1995)
14. Koenig, L., Jebens, K., Kernbach, S., Levi, P.: Stability of on-line and on-board evolving of adaptive collective behavior. In: European Robotics Symposium 2008. Springer Tracts in Advanced Robotics, vol. 44, pp. 293–302. Springer, Heidelberg (2008)
15. Nehmzow, U.: Physically embedded genetic algorithm learning in multi-robot scenarios: The pega algorithm. In: Prince, C., Demiris, Y., Marom, Y., Kozima, H., Balkenius, C. (eds.) Proceedings of The Second International Workshop on Epigenetic Robotics: Modeling Cognitive Development in Robotic Systems, LUCS, Edinburgh, UK, vol. 94 (August 2002)

16. Nolfi, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press/Bradford Books, Cambridge, MA (2000)
17. Perez, A.L.F., Bittencourt, G., Roisenberg, M.: Embodied evolution with a new genetic programming variation algorithm. In: ICAS, pp. 118–123 (2008)
18. Rechenberg, I.: Evolutionstrategie: Optimierung Technisher Systeme nach Prinzipien des Biologischen Evolution. Fromman-Hozlboog Verlag, Stuttgart (1973)
19. Schwefel, H.-P.: Numerical Optimisation of Computer Models. Wiley, New York (1981)
20. Simões, E.D.V., Dimond, K.R.: Embedding a distributed evolutionary system into population of autonomous mobile robots. In: Proceedings of the 2001 IEEE Systems, Man, and Cybernetics Conference (2001)
21. Thrun, S., Burgard, W., Fox, D.: Probabilistic Robotics. MIT Press, Cambridge (2005)
22. Tomassini, M.: Spatially structured evolutionary algorithms: Artificial evolution in space and time. In: Natural Computing Series (2005)
23. Usui, Y., Arita, T.: Situated and embodied evolution in collective evolutionary robotics. In: Proceedings of the 8th International Symposium on Artificial Life and Robotics, pp. 212–215 (2003)
24. Walker, J.H., Garrett, S.M., Wilson, M.S.: The balance between initial training and lifelong adaptation in evolving robot controllers. IEEE Transactions on Systems, Man, and Cybernetics, Part B 36(2), 423–432 (2006)
25. Watson, R.A., Ficici, S.G., Pollack, J.B.: Embodied evolution: Distributing an evolutionary algorithm in a population of robots. Robotics and Autonomous Systems 39(1), 1–18 (2002)
26. Wischmann, S., Stamm, K., Wörgötter, F.: Embodied evolution and learning: The neglected timing of maturation. In: Almeida e Costa, F., Rocha, L.M., Costa, E., Harvey, I., Coutinho, A. (eds.) ECAL 2007. LNCS (LNAI), vol. 4648, pp. 284–293. Springer, Heidelberg (2007)

Implementation Details of the (1+1)-Online Algorithm

- Each individual runned during 120 time step (60 time step of recovering and 60 time step of evaluation).
- 600 evaluations per experiments.
- 14 runs with (1+1)-online algorithm and 14 runs with restart (1+1)-online algorithm.
- random individual at a random position, at the beginning of a run.
- $P_{reevaluate}$ is set to 0.2.
- σ initial value is set to 1 and may range from 0.01 to 4.
- genes value in [-4,+4].

Chapter 12
Automated Planning Logic Synthesis for Autonomous Unmanned Vehicles in Competitive Environments with Deceptive Adversaries

Petr Svec and Satyandra K. Gupta

Abstract. We developed a new approach for automated synthesis of a planning logic for autonomous unmanned vehicles. This new approach can be viewed as an automated iterative process during which an initial version of a logic is synthesized and then gradually improved by detecting and fixing its shortcomings. This is achieved by combining data mining for extraction of vehicle's states of failure and Genetic Programming (GP) technique for synthesis of corresponding navigation code. We verified the feasibility of the approach using unmanned surface vehicles (USVs) simulation. Our focus was specifically on the generation of a planning logic used for blocking the advancement of an intruder boat towards a valuable target. Developing autonomy logic for this behavior is challenging as the intruder's attacking logic is human-competitive with deceptive behavior so the USV is required to learn specific maneuvers for specific situations to do successful blocking. We compared the performance of the generated blocking logic to the performance of logic that was manually implemented. Our results show that the new approach was able to synthesize a blocking logic with performance closely approaching the performance of the logic coded by hand.

12.1 Introduction

Development of increased autonomy for unmanned vehicles to successfully fulfill ordinary mission tasks, such as navigation between two locations while avoiding

Petr Svec
Institute for Systems Research and Department of Mechanical Engineering,
University of Maryland, College Park, MD 20742, USA
Phone: 301-405-5577
e-mail: `petrsvec@umd.edu`

Satyandra K. Gupta
Institute for Systems Research and Department of Mechanical Engineering,
University of Maryland, College Park, MD 20742 USA
Phone: 301-405-5306
e-mail: `skgupta@umd.edu`

dynamic obstacles, is still considered to be a challenge. Handling all the variations in the encountered environments requires writing and extensive tuning a large amount of lines of code by human programmers. Yet the real difficulty comes when the unmanned vehicle has to autonomously face human-competitive adversary utilizing deception. This truly challenging scenario is typical for a combat mission where even a single mistake in the decision of the vehicle can have serious consequences.

Computational synthesis (CS) [1] deals with the problem of how to automatically compose and parametrize a set of functional building blocks into increasing amount of modules that are further organized into a hierarchical solution structure with the desired functionality. This is in contrast to classical optimization in which the number and structure of modules and parameters being optimized is known in advance. The rapid growth of CS in the recent years can be accounted to increasingly affordable computational power and continuing advances in machine learning and simulation techniques.

For many years since its original inception, Genetic Programming (GP) [2, 3] has been emerged as an invention machine for automated synthesis of controllers that are simpler and more efficient than those engineered with other standard design methods. The random exploration feature of this technique allows generating creative human-readable solutions in contrast to human developers who are often limited by constraints of standard engineering methods. GP as one of the robust evolutionary techniques has been used for automatically generating computer programs in various domains. These programs usually have a tree structure and are generated using an algorithm similar to the traditional genetic algorithm (GA) [4]. In literature, there are many successful GP applications to numerous problems from different domains [5] including robotics, optimization, automatic programming, machine learning, etc. In robotics, GP is used as a methodology that uses evolutionary algorithms to automatically synthesize controllers and body configuration for autonomous robots. Most of the controllers were evolved for behaviors as obstacle avoidance [6, 7], wall-following [8], line following [9], light seeking, robot seeking [7], box pushing [10], vision-driven navigation [11], homing and circling [12], predator versus prey strategies [13], co-evolution of control and bodies morphologies [14], game playing [15-17] or group control for survey missions [18]. GP was also utilized for the automated synthesis of human-competitive strategies for robotic tanks run in a closed simulation area to fight other human-designed tanks in international leagues [19]. There was also some progress on development of limited machine intelligence for classical strategic games like backgammon or chess endgames [20].

Most of the evolved controllers, however, are purely reactive and contain a predefined number of modules. This substantially limits their usage for complex tasks that involve competitive adversaries, either machines or human operators themselves. We therefore propose a new approach for automated planning logic synthesis that can be viewed as an iterative learning process during which an initial version of the logic is automatically synthesized and then gradually improved by detecting and fixing its shortcomings. This is achieved by the interplay of data mining for state extraction and classification and GP for navigation code generation. During the automated development, no external information on how the logic should be generated is therefore needed. The planning logic is represented as a composition of one main navigation controller and a set of navigation plans. The navigation controller is used to control the vehicle's behavior in all situations besides the situations for which specific maneuvers are needed to boost the performance of the logic. The symbolic representation of the planning logic greatly simplifies its functional analysis in contrast to the artificial neural network representation that is difficult to analyze. Moreover, the symbolic representation allows integrating human knowledge naturally so the analysis of the logic can provide the basis for hand-coding logic for real-life applications.

We tested our new approach in the context of a larger project aimed at the development of a mission planning system [21] for the automated generation of planning logic for unmanned surface vehicles (USVs) [22, 23]. In this part of the work, our focus is specifically on automated generation of logic used for blocking the advancement of an intruder boat towards a valuable target. This task requires the USV to utilize reactive planning complemented by short-term forward planning to generate local navigation plans describing specific maneuvers for the USV. The intruder is human-competitive in the sense that its attacking efficiency approaches the attacking efficiency of deceptive strategies exhibited by human operators. Our aim is to reach the level 3 of autonomy as defined in [24]. In this level, the unmanned vehicle automatically executes mission-related commands when response times are too short for operator intervention. The operator, however, may cancel or redirect the vehicle's intended actions. As far as the performance evaluation of the logic is concerned, we 1) manually implemented USV's logic for blocking the hand coded intruder, and 2) compared its performance to the automatically generated USV's logic by pitting it against the same intruder in a large amount of test scenarios.

An overview of the overall approach is shown in Figure 12.1. First, we developed a physics-based meta-model of a full blown USV dynamics model to be able to test the planning logic in a simulation environment in real-time [25]. Second, we developed a mission planning system whose main part is GP based evolutionary module for generating individual components of the planning logic (see section 12.3). In order to combine the elements of the project into a cohesive system, we designed a USV simulation environment [26]. The USV simulation environment integrates various components of the project into a complete simulation system and acts as a simulation platform for the evolutionary module. One of the components of the USV simulation environment is the virtual environment (VE) based simulator (see Figure 12.2) which serves as an emulator of the real USV environment and contains gaming logic which allows human players to play against each other or against the computer. Finally, we evaluated the performance of the automatically generated USV's logic against a hand coded human-competitive intruder exhibiting deceptive behavior (see section 12.4).

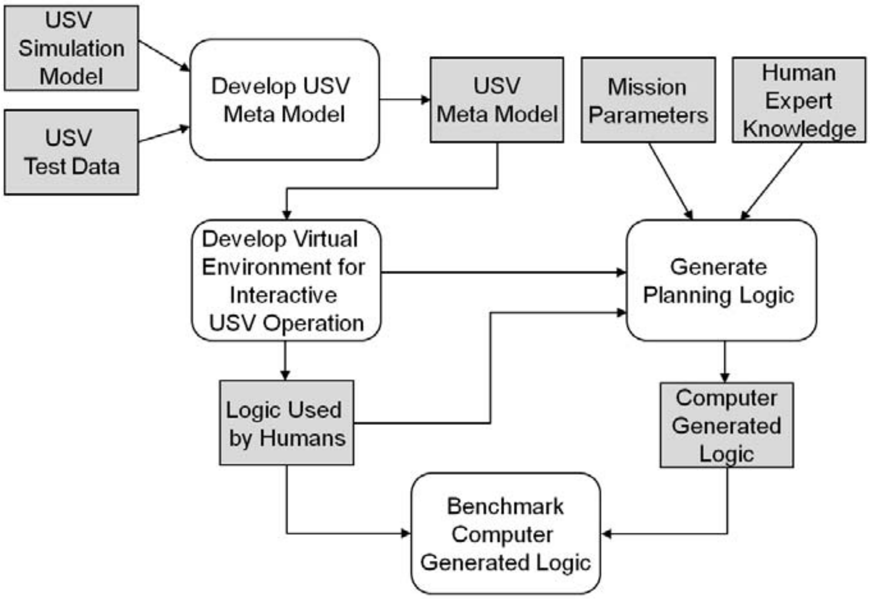


Fig. 12.1 Overview of the overall approach.

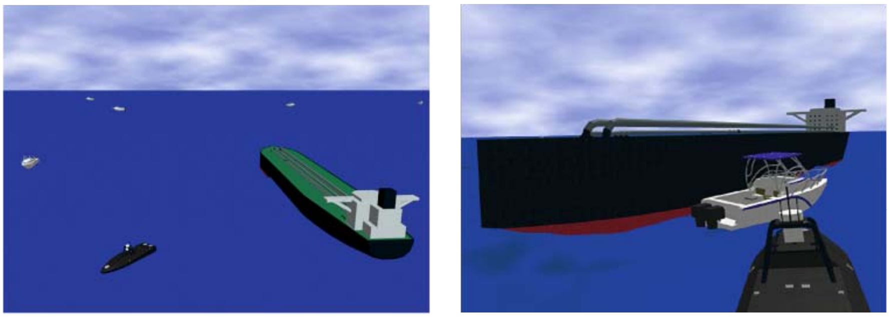


Fig. 12.2 Virtual environment.

12.2 USV System Architecture

The overall USV's system architecture consists of several modules that are responsible for different tasks, e.g. sensing, localization, navigation, planning, behavior control, communication, human interaction, or monitoring [22, 23]. The 6 degrees of freedom USV dynamics simulation model was implemented as described in [25]. This detailed model considers disturbances from the surroundings and is used for game playing inside the virtual environment. Due to its computational requirements, its simplified version that has 3 degrees of freedom is used for automated logic generation where evaluation speed is an issue.

12.2.1 USV Virtual Sensor Models

The role of sensing is to provide information about the current state of the vehicle in the environment. This information serves as a basis for decision making and control by taking other objects in the environment into account. A real USV is usually equipped by radar and multiple short-range visibility sensors to be able to capture the state of its surroundings. However, for the navigation system of the USV to be effective it only needs to process relevant key sensory information abstracted from the raw sensor data. The sensory information is thus represented as a vector containing only the features required for a successful fulfillment of the mission task. The values of the relevant features are computed using the data from the virtual sensors [27] that provide intermediate abstraction of the raw sensor data. Some of the features can also have one or more parameters using which their final values are computed.

The navigation system of the USV uses virtual visibility, relational, and velocity sensors. The virtual visibility sensor is a detection sensor with cone-shaped detection regions (see Figure 12.3). The dimension of the overall sensor area is defined by its reach and range parameters. Each region returns a boolean value expressing the presence of other objects and a normalized distance to the closest object. The relational virtual sensor provides relevant information on how other objects are situated to the USV or to each other. It provides boolean values for computation of the values of the relational features. The velocity virtual sensor returns velocities of other objects inside the environment.

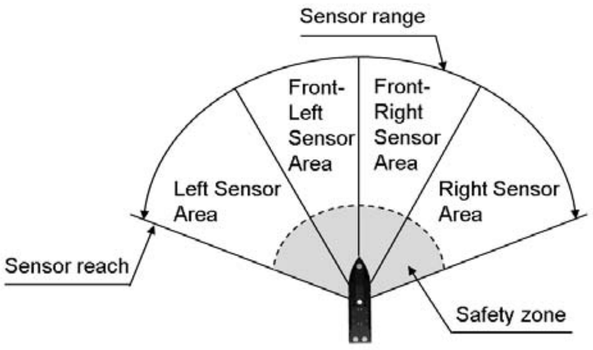


Fig. 12.3 Virtual visibility sensor model.

12.2.2 Planning Architecture

The complexity of interactions of a mobile robotic system suggests structured (non-monolithic) high-level planning architecture. The unmanned boats must behave based on the effect of several independent threads of reasoning. This is due to the highly parallel nature of events and processes in an uncertain and often dynamic environment. The control architecture can meet this requirement if it is modular, and when the modules can act simultaneously in a coordinated cooperation in real time.

12.2.2.1 Planning Logic Representation

The planning logic allows the USV to make a decision from a set of allowable actions in a particular situation during its run. It consists of a main navigation controller represented as a decision tree and zero or more navigation plans (see Figure 12.4). For the operators to understand and trust the vehicle's actions is of a great importance. Hence, the representation of the planning logic is symbolic which simplifies its functional verification by human auditing.

The main navigation controller operates the USV unless the vehicle approaches a state for which a specific short-term navigation plan exists. The navigation plan thus represents a certain maneuver the USV executes in a certain situation to increase its performance. The main components of the navigation controller are high-level parameterized navigation commands *NC* = {*go-intruder-front (front-left, left, front-right, right), turn-left (right), go-straight*}, conditional variables *CV* = {*intruder-on-the-left (right, front, at-the-back), intruder-has-target-on-the-left (right), usv-has-target-on-the-left (right), usv-intruder-distance-le-than, usv-closer-to-target-than-intruder, usv-facing-intruder, usv-left (right, front-left, front-right) visibility-sensor-area-activated, intruder-target-angle-between, intruder-velocity-le-than*}, standard boolean values and operators *BVO* = {*if, true, false, and, or, not*}, *program blocks PB* = {*seq2, seq3*}, and system commands *SC* = {*usv-sensor, usv-velocity, usv-match-intruder-velocity*}. The main components of the navigation plan are high-level commands *NC* and program blocks *PB*. The leaves of the decision tree can be conditional variables or navigation commands. The inner nodes can be conditionals, navigation commands, or system commands. Each navigation command corresponds to a particular high-level controller, which is a parameterized composition of simple behaviors according to the behavior-based control architecture [28]. The next section describes this in detail.

The conditional variables, navigation, and system commands are parameterized. The parameters of a navigation command define its underlying property. The positional commands (e.g. *go-intruder-front*) are defined using 5 parameters. The first two parameters represent the USV's relative goal position (in polar coordinates) around the intruder. This effectively allows the vehicle to cover all feasible positions, as defined by its planning logic, in a certain area around the intruder. The next two parameters represent a cone-shaped blocking area around the relative goal position. Once the vehicle gets inside the blocking area, it starts slowing down to limit the intruder's movement. The last parameter represents the length of the command execution. The turning navigation commands have two parameters that represent the turning rate and the length of the command execution. The translation velocity is explicitly controlled by the velocity commands. The *usv-sensor* system command effectively changes the parameters of the USV's sensors allowing it to explicitly control the obstacle avoidance behavior to be between very safe and very risky. Each parameter of the command is propagated to the underlying primitive behaviors of each corresponding high-level controller.

The input into the planning logic is data from the virtual sensors. The values of all conditional variables are computed using this data. So for example, the boolean value of the *intruder-on-the-left* variable is directly provided by the virtual relation sensor, while the data for computation of the *intruder-velocity-le-than* parameterized variable is provided by the virtual velocity sensor.

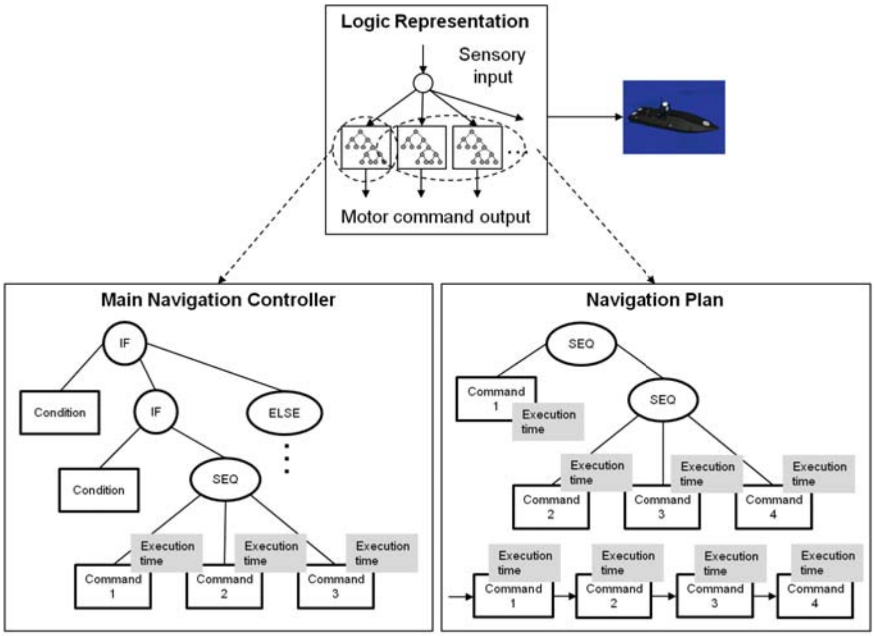


Fig. 12.4 Planning logic representation.

12.2.2.2 Hierarchical Control Architecture

During the mission, the USV periodically senses its surroundings and classifies its current state with respect to the intruder and the target. The classification mechanism of the planning logic executor decides whether the current USV's state is close enough to one of the states for which a corresponding navigation plan exists. If such a plan exists, the planning logic executor (see Figure 12.6) directly executes the plan, otherwise it executes the main navigation controller to generate a new plan. The decision whether to execute a specific navigation plan depends on the activation distance parameter δ. This parameter defines the minimal distance that has to be achieved between the current USV's state and any state in the predefined set to activate a specific navigation plan. The state space is thus divided into two regions where in the first region the USV generates and executes plans using the navigation controller, whereas in the other region the USV directly executes previously learned plans. The distance between normalized states is computed using the standard Euclidean distance metric.

The full state of the USV (see Figure 12.5) is a vector $s = \{\alpha_1, \alpha_2, \beta_1, \beta_2, \gamma_1, v_1, v_2, d\}$. The angle α_1 represents an angle between the USV's heading and the direction to the target, α_2 is an angle between the intruder's heading and the direction to the target, β_1 is the USV's steering angle, β_2 is the intruder's steering angle, $v1$ is the USV's translation velocity, $v2$ is the intruder's translation velocity, γ_1 is an angle between the USV's heading and the direction to the intruder, and d is the distance between the USV and the intruder.

By acquiring and processing sensory information in short-term cycles, and planning, the planning system determines a navigation command to be executed through the behavior-based control system to direct the vehicle inside the environment. The planning logic executor of the navigation system takes as inputs sensor data, mission parameters, USV meta model, and the planning logic. It decides which component of the logic to execute to generate a plan based on the current vehicle's state. The plan consists of a number of navigation commands, each being executed for a certain amount of time. The ultimate outputs of an activated command are way points that are directly translated by a low-level controller into motor commands for device drivers of a particular actuator.

The control architecture is hierarchical and follows the behavior-based paradigm [28]. It consists of planning, executive, and reactive layers (see Figure 12.6). The planning layer is responsible for interpreting stored planning logic and generating navigation plans that contain one or more navigation commands at a time. The commands are stored in a queue to be further dispatched for execution by the dispatcher in the executive layer. The commands are usually planned for short-term execution,

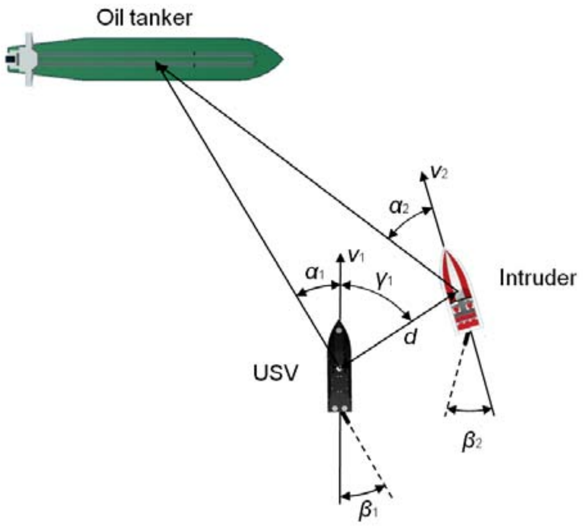


Fig. 12.5 USV's state in respect to the intruder and target.

such as planning of strategic maneuvers. The vehicle thus does not act purely reactively to its surroundings unless an exception occurs.

Each command corresponds to a high-level controller, which is a parameterized composition of simple behaviors organized into layers according to the behavior-based control architecture [28]. The executive layer is responsible for processing the commands in the queue and invoking the corresponding high-level controllers in a series for planned periods of time. The length of the execution is defined as a parameter of the command. The executive layer is also responsible for monitoring execution of the controllers and handling exceptions. An exception occurs if the current state of the vehicle substantially deviates from the predicted trajectory defined by the plan. The planning logic executor remains inactive until all the commands from the queue are processed in which case the dispatcher requests new commands from the planning logic executor and the control process continues.

The reactive layer implements the behavior-based subsumption architecture [28]. This architecture decomposes a complicated high-level controller into a set of simple behaviors (steer left / right, go straight, arrive) organized into layers. These primitive behaviors are finite state machines acting in response to sensor inputs and producing actuator action outputs. Multiple behaviors can be activated simultaneously producing different conflicting motor commands. This means that a certain amount of coordination is needed. Due to its robustness, we have chosen a priority-based arbitration mechanism, picking the actuator action output of the behavior with the highest priority as the overall action output of the currently activated high-level controller. This closely follows the behavior-competitive paradigm that imposes that only one behavior can have control over the robot's actuators while each of them can access all sensors. In this paradigm, the behavior in the highest layer has the highest priority (for example obstacle avoidance) while the behavior in the lowest layer represents the most abstract functionality. In the architecture, each high-level controller specifies a fixed priority ordering of behaviors as defined by [29].

The planning logic is executed in a perception-action high rate cycle. The input into the logic is processed sensor data and the ultimate output is an actuator action. The ability of the architecture to deal with highly dynamic and unpredictable scenarios is due to its underlying reactive behavior-based competitive component. This component triggers local obstacle avoidance mechanism at a high rate to immediately detect possible collision threats.

The primitive behaviors are of a great importance as they are able to quickly produce an action in a highly dynamic environment where fast response and responsibility are crucial. A behavior is a simple unit that produces an output of a pre-defined type, in our case a two-dimensional vector containing desired translation velocity and steering angle. Conditions for activating behaviors are preprogrammed. The architecture defines the following primitive behaviors: *obstacle avoidance, go to location, go straight, turn left*, and *turn right*.

The *obstacle avoidance* behavior implements a simple but efficient obstacle avoidance mechanism for dynamic environments and is a necessary part of all high-level controllers. It uses the virtual visibility sensor (see Figure 12.3) in order to identify the location of detectable obstacles. It directly produces desired translation velocity and steering angle to safely steer the boat away from the closest identified obstacles. The desired steering angle increases with the proximity to the obstacles while the translation velocity decreases.

The manual design of a flexible reactive obstacle avoidance mechanism is not an easy task. The USV should be able to prevent a collision with its surroundings while effectively executing its mission task. This brings plenty of challenges since the behavior of many standard obstacle avoidance methods is driven by its carefully tuned parameters. These parameters control the behavior of the vehicle, particularly how much steering should be applied when a nearby obstacle is positioned at a certain distance and angle, and how fast the vehicle should be moving in that situation. Hence for our mission, the resulting behavior can be quite different with different parameters essentially controlling the vehicle's safe distance from the adversary and blocking efficiency at the same time. Insufficient avoidance steering can lead to collisions. On the other hand, too much steering will veer the vehicle away from the adversary leading to ineffective blocking. Moreover, as far as computational requirements are taken into account, the obstacle avoidance mechanism should be very fast so that it does not consume much of the precious simulation time.

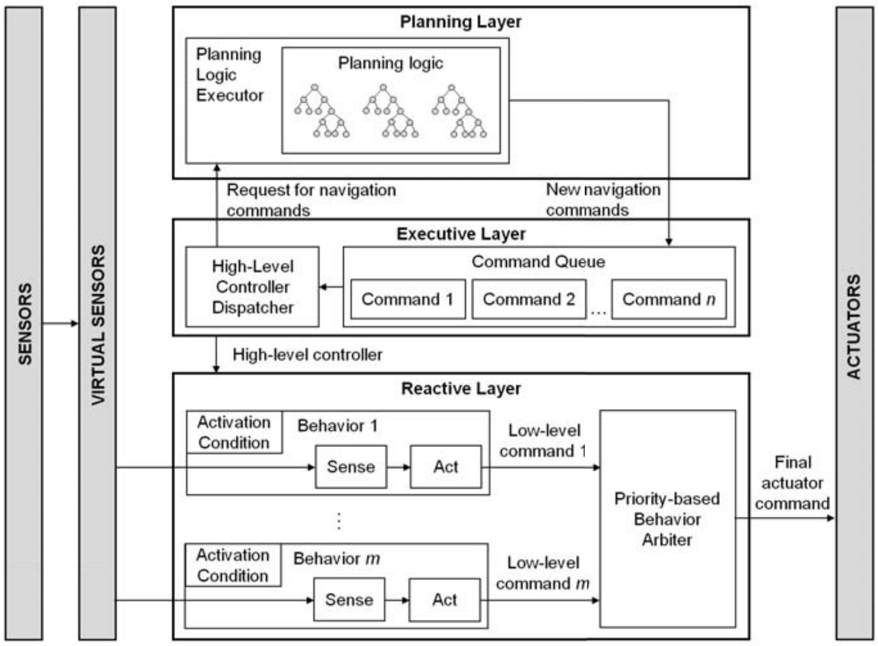


Fig. 12.6 Hierarchical behavior-based control architecture.

We have implemented a collision avoidance method that uses high-level sensory information, e.g. positions, orientations, and dimensions of obstacles, to directly decide which steering angle and translation velocity to request. The avoidance mechanism uses a fixed set of control parameters. However, the behavior can be conveniently controlled by modification of the underlying parameters of the visibility sensor. This way, the USV can get closer to obstacles than it would be otherwise possible and thus effectively define a balance between safe and aggressive maneuvering. The command *usv-sensor* of the planning logic modifies the reach and range parameters of the virtual visibility sensor cones.

By default, the behaviors always choose such translation and steering velocities that maximize the USV's performance. So for example, *go-straight* behavior uses maximum translation velocity to get to the requested position in the shortest amount of time. The planning logic can override this by calling *usv-velocity* system command. This command switches the vehicle to its controlled velocity mode in which the translation velocity of the USV is controlled by the higher-level planning logic.

12.3 Planning Logic Synthesis

12.3.1 Test Mission

Our task was to automatically generate a planning logic for the USV to slow down an intruder boat moving toward the protected object. The USV's blocking logic is defined in the context of a test mission. During this mission, the USV must protect an oil tanker by patrolling around it while avoiding collisions with friendly boats and scanning the environment for a possible intruder. The environment around the oil tanker is divided into danger and buffer zones (see Figure 12.7). Once the intruder enters the buffer zone, the USV approaches the intruder boat and circles it for surveillance purposes. If the intruder enters the danger zone, the USV does its best to block the intruder, slowing the intruder's progress toward the tanker.

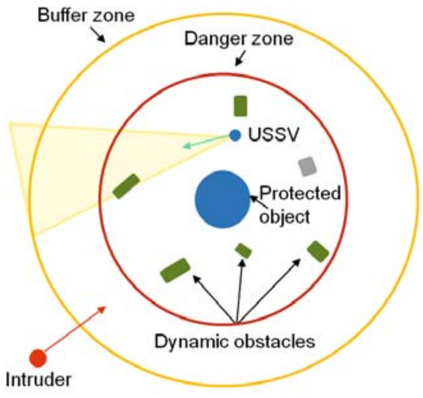


Fig. 12.7 Test mission.

12.3.2 Synthesis Scheme

The natural way of automated logic generation is to let the unmanned vehicle to autonomously learn its own logic for a given task. Learning the logic usually requires many simulations of different actions in different situations so that the vehicle can determine which actions are beneficial and which are not. Our approach for the automated logic synthesis can be viewed as a completely automated iterative process during which an initial version of the logic is automatically synthesized and then gradually improved by detecting and fixing its shortcomings.

The planning logic generation process consists of six main parts as shown in Figure 12.8. First, an initial version of the logic containing only a navigation controller is generated and consequently evaluated inside the simulator in m distinct evaluation runs. This evaluation returns a set of states of failure $SF = \{SOF_1, \ldots, SOF_n\}$, $n \leq m$ in which the vehicle fails its mission task. Given this set, a representative state of failure $SOF_{REP} \in SF$ is found for which the average of distances to its k nearest neighboring states is minimal. SOF_{REP} is thus a state located in the center of a cluster with the highest density of states of failure. The distance between the normalized states is computed using the Euclidean distance metric. Next, we compute a corresponding state of exception SOE_{REP} for the representative state of failure SOF_{REP}. SOE_{REP} defines a state in which proximity (given by the activation distance parameter δ) the vehicle can execute a specific navigation plan to decrease the probability of occurrence of the corresponding SOF_{REP} and all the failure states in its close neighborhood. Once the state of exception SOE_{REP} is computed, a new specific navigation plan is synthesized for this state. To prevent

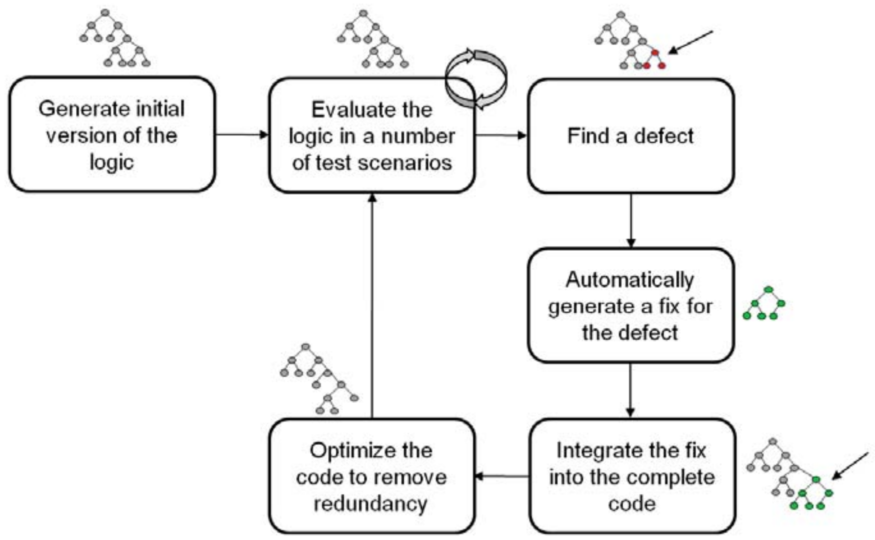

Generate initial version of the logic
Evaluate the logic in a number of test scenarios
Find a defect
Automatically generate a fix for the defect
Integrate the fix into the complete code
Optimize the code to remove redundancy

Fig. 12.8 Planning logic synthesis overview.

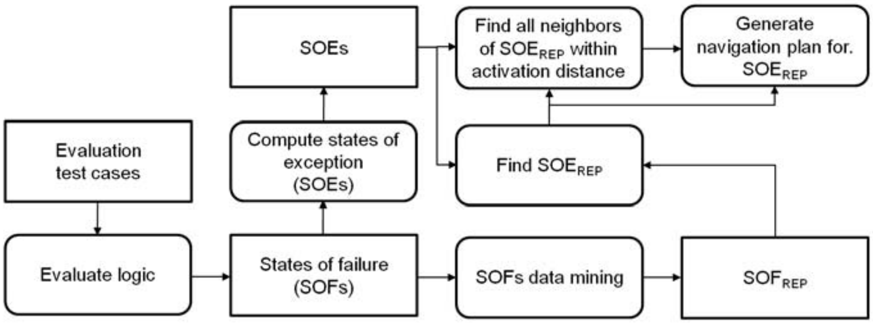

SOEs
Find all neighbors of SOE$_{REP}$ within activation distance
Generate navigation plan for. SOE$_{REP}$
Evaluation test cases
Compute states of exception (SOEs)
Find SOE$_{REP}$
Evaluate logic
States of failure (SOFs)
SOFs data mining
SOF$_{REP}$

Fig. 12.9 Extraction of states of exception.

overspecialization of the new plan, we evaluate its performance using all states of exception found within SOE_{REP} activation distance δ during the overall logic evaluation. The new plan together with its corresponding SOE_{REP} is then integrated into the whole logic, the logic is optimized, and the process starts again.

In the context of our mission, SOF defines a situation in which the USV has high probability of losing its future maneuver to block the intruder. Its corresponding SOE is found by reverting back in time for a certain number of time steps τ to record a state from which a new specific navigation plan can be executed to prevent a possible future failure. The activation distance parameter δ defines a minimum distance between the current USV's state and any previously recorded SOE_{REP} from which a corresponding navigation plan can be executed.

12.3.3 Planning Logic Components Evolution

Both the navigation controller and navigation plans as components of the planning logic are automatically generated using separate simulated evolutionary processes. The specific evolutionary method we used is the strongly-typed Genetic Programming imposing type constraints on the generated Lisp trees [15, 16]. This is a robust stochastic optimization method that searches a large space of candidate program trees while looking for the one with the best performance (fitness value).

The evolutionary process starts by randomly generating an initial population of individuals represented as GP trees using the Ramped half-and-half method [42]. We seed the initial population by human-coded Lisp fragments to speed up the process. The initial values of parameters of all navigation commands and conditionals are either seeded or randomly generated. The navigation controller of the planning logic is generated using a human written template for which GP supplies basic blocking logic. The first portion of the template encodes a maneuver using which the vehicle effectively approaches the intruder at the beginning of the run as there is no need for it to be explicitly evolved.

The terminal T and function sets F consists of navigation commands, system commands, conditional variables, boolean values and operators, and program blocks as defined in section 12.2.2. The sets are defined as

$$T_{controller} = T_{plan} = NC \cup SC$$

$$F_{controller} = CV \cup BVO \cup PB; F_{plan} = PB$$

Within the population, each individual has a different structure responsible for different way of how it responds to its environment. The individuals are evaluated in the context of the whole logic inside the simulator. The sensory-motor coupling of the individual influences the vehicle's behavior resulting in a specific fitness value that represents how well the USV blocks the intruder.

We favor individuals which can rapidly establish basic blocking capabilities and optimize them in such a way to push the intruder away from the target over the entire trial duration. To do so, the fitness F is defined as the sum of squared distances of the USV from the target over all time steps. This squared distance is normalized due to the different initial distances of the intruder from the target in the test scenarios. If a collision occurs, either caused by the USV or the intruder, the zero fitness value is assigned to the individual, and the selection pressure eliminates the logic component with low-safety guarantee. The fitness function is as follows:

$$F = \frac{1}{T}\sum_{i=1}^{T}(\frac{d_i}{d_0})^2 \tag{12.1}$$

where T is the total number of time steps, d is the distance of the intruder from the target at time step i, and d_0 is the initial distance of the intruder from the target in a particular test case. The total fitness value of the individual is being maximized and is computed as an average of fitness values resulting from all scenarios.

The navigation controller is evaluated using a human-competitive intruder in 8 different scenarios. In each scenario, the intruder has a different initial orientation, and the USV always starts from an initial position closer to the target. The evaluation lasts for a maximum amount of time steps which equals to 300 seconds in real time. The maximum speed of the USV is set to be 10% higher than the speed of the intruder, other properties of the vehicles are the same. The navigation plan is evaluated using all states of exception found within the activation distance of its corresponding SOE_{REP}. The evaluation lasts for a maximum amount of time steps which equals to 10 seconds in real time.

The individuals in the initial population mostly exhibit random behavior. By selecting and further refining the individuals with high fitness, their quality gradually improves in subsequent generations. During this process, the individuals are randomly recombined, mutated, or directly propagated to the next generation. These operations are applied with the predefined probabilities (see Table 12.1). The following evolutionary operators are used:

- Reproduction – copies one individual directly to the next generation without any modification. We use a strong elitism to propagate the best individual directly into the next generation. This makes sure that the best individual is not lost during the evolutionary process as a consequence of recombination.
- Mutation – we use three types of mutation operators: structural mutation of a randomly selected sub-tree, preventing bloat [30] by shrinking a randomly chosen sub-tree to a single node, and Gaussian mutation of chosen parameters.
- Crossover – randomly selects sub-trees from two input trees and swaps them.

During the logic synthesis, the USV learns the balance between a safe and dangerous maneuvering by mutating the reach and range parameters of its virtual visibility sensor. The logic is thus co-evolved with the sensor parameters of the vehicle to control the obstacle avoidance mechanism.

The optimization of the generated navigation controller removes all branches of the code that have not been executed during evaluation scenarios. Moreover, each generated navigation plan is truncated to contain only the navigation commands that do not exceed the execution time of the plan. This effectively prevents bloat and generates cleaner code.

A detailed description of the functionality of all the operators used can be found in [2]. The control parameters of the evolutionary process used for evolution of the navigation controller and plans are summarized in Table 1.

Table 12.1 GP Parameters

Population size / number of generations	500 / 100 (controller) 50 / 20 (plan)
Crossover probability	0.84
Tournament size	2
Structure mutation probability	0.05
Elite set size	1
Shrink structure mutation probability	0.01
Min. and max. initial GP tree depth	3 and 6 (controller) 2 and 4 (plan)
Mutation probability of parameters of navigation commands	0.5
Maximum GP tree depth	50 (controller) 10 (plan)
Crossover probability	0.84

12.4 Computational Experiments

12.4.1 General Setup

In the test mission domain, the intruder boat has to reach the target as quickly as possible, while the USV has to block and delay the intruder for as long time as possible. We set up an experiment to compare the performance of the automatically generated USV's logic for blocking to the USV's logic coded by hand. We compare the performance in terms of pure time delay imposed by the USV on the intruder. To get a fair assessment of the USV performance, the time values being compared must be normalized by 40 seconds baseline. This baseline represents the amount of time needed to reach the target if the intruder is completely unobstructed. Any additional time above this baseline thus represents the effective delay time of the intruder when being blocked by the USV.

The USV's logic is evaluated in 800 evaluation runs to account for the intruder's deterministic behavior interspersed with random actions. Each evaluation run lasts for a maximum amount of time steps which equals to 300 seconds in real time. The dimension of the scene is 800 x 800 m with the target positioned in the center. At the beginning of each run, the USV and the intruder are oriented toward each other with random deviation of 10 degrees and the USV is positioned on a straight line between the intruder and the target. The initial distance of the USV from the target is approximately 240 m, while the intruder's initial distance is 360 m. The maximum time for the evaluation run is set to 5 minutes. The USV's maximum velocity is 10 m/s, while the intruder's maximum velocity is 9 m/s.

12.4.1.1 Experimental Protocol

First, we implemented an initial version of the intruder's attacking logic and tested it against human players to evaluate its performance. The logic was further improved in multiple iterations in the span of 6 weeks. Its overall size reached 485 lines of Lisp code. The outline of the logic functionality is described in the next section. We evaluated the performance of the logic by pitting human players against it playing as USVs. The human players achieved 90 seconds of pure time delay imposed on the intruder in average. This shows that the intruder's attacking logic is quite sophisticated due to its deceptive behavior.

Second, we manually implemented the USV's blocking logic against the hand coded intruder. This involved implementation of its main navigation controller, manually finding 49 states of failure and their corresponding states of exception, hand coding the logic for each state of exception, and overall evaluation. The logic was improved iteratively in the span of 3 weeks. Its overall size reached 500 lines of Lisp code.

Third, we used the mission planning system to automatically generate the USV's blocking logic using the hand coded intruder as the competitor. The activation distance parameter δ was set to 0.2 for all navigation plans. The representative state of failure $SOF_{REP} \in SF$ was found as a state with the minimal average distance to its 7 nearest neighboring states. The number of time steps τ for computation SOE from SOF was set to 150. In the current version of the approach, SOF is determined to be a state in which the intruder is closer to the target than the USV.

Finally, we compared the performance of the automatically synthesized USV's logic to the hand coded USV's logic.

12.4.1.2 Intruder's Planning Logic

The hand-coded intruder's logic is represented as a single decision tree that contains standard navigation commands as well as their randomized versions. The partial non-determinism of the logic allows the intruder to use different actions in the same situations. The intruder repeatedly deceives the USV's logic by randomly taking actions in some specific situations so that the USV is not able to find a motion pattern in intruder's behavior that can be easily exploited for blocking.

The intruder's logic can be divided into five main sections. Each of these sections handles a different group of situations that can arise during the combat. The first section handles situations in which the distance of the intruder from the target is larger than 130 m and the angle between its translation direction and the target is more than 80 degrees. In these situations, the intruder attempts to rapidly change its direction of movement toward the target by aggressively turning left or right depending on the position of the USV.

The second section handles situations in which the USV is very close to the intruder, positioned relatively to its front left, and the target is on the intruder's left side (see Figure 12.10a). In this case, the intruder has two options. Either it executes a random turn with probability 0.9 or it proceeds with a complete turn. In both cases, the intruder can slow down rapidly with probability 0.3 to further confuse the adversary. This section handles also similar type of situations when the USV is on the front right of the intruder and the target is on the right.

The third section is very similar to the second section with the exception that the USV is directly on the left or right side of the intruder (see Figure 12.10b). In these cases, the intruder does its best to deceive the USV. The intruder would often slow down rapidly trying to get an advantageous position, randomly proceed with a complete turn, or execute a partial turn. The probability of the complete turn is 0.1 and the probability of slowing down is 0.2.

The fourth section deals with the situations during which the intruder is positioned behind the USV inside the rear grey area as shown in Figure 12.10c. The larger distance of the intruder from the USV gives it opportunity to exploit the USV's tendency to over shoot a little in the process of blocking. In this case, if the USV has high velocity, the intruder's logic would suddenly slow it down and turn it toward the stern of the blocking USV, passing the USV from behind. Otherwise, the intruder randomly turns with probability 0.7 or it proceeds with a complete turn (see Figure 12.10d). Again, the intruder can rapidly slow down with probability 0.3.

Finally, if the intruder is not in a close proximity to the USV, it computes the best sequence of moves in order to get to the target as fast as possible.

The intruder's logic can modify the reach and range parameters of its virtual visibility sensor to control the balance between a safe and aggressive obstacle avoidance mechanism. For example, if the intruder wants to make an aggressive turn in a close proximity to the USV it has to take risk by decreasing the reach of the sensor to be able to quickly proceed with the turn. In this case, the obstacle avoidance behavior sensitivity is reduced for a short period of time so that the intruder can easily pass

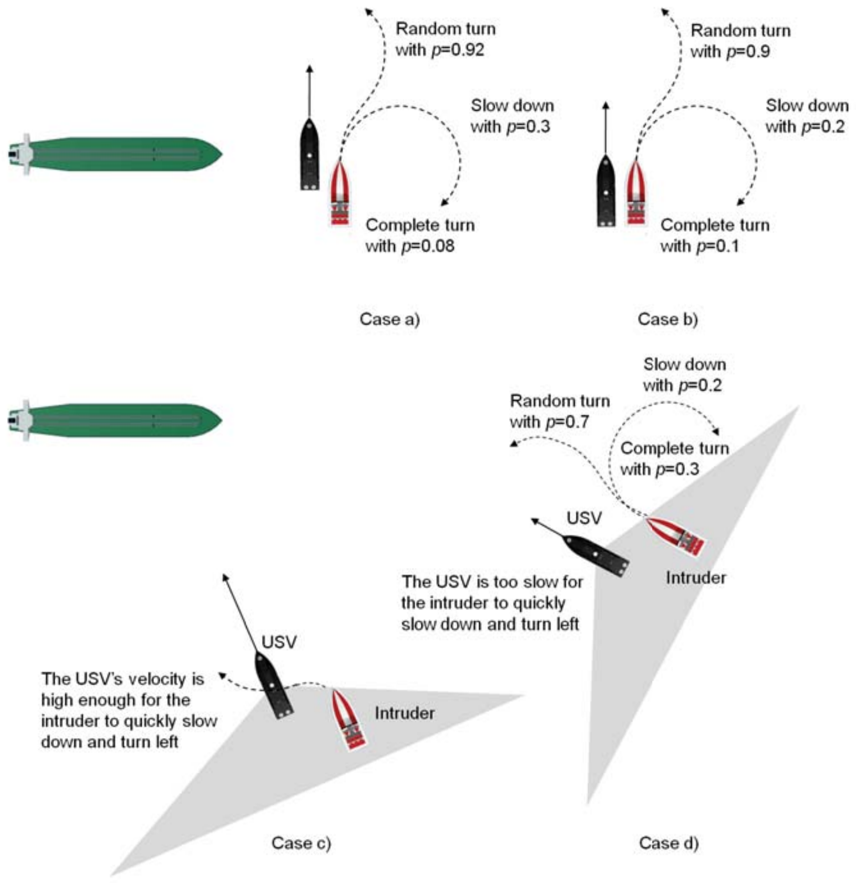


Fig. 12.10 Representative portions of intruder's planning logic.

the USV from behind. If the intruder always aimed to safely avoid the adversary, it would not get any chance to get to the target, especially if pit against a human player.

12.4.2 Results

The experimental run that generated a blocking logic with the highest performance is shown in Figure 12.11. The horizontal axe of the graph shows different versions of the logic consisting of gradually increasing amount of navigation plans. The vertical axe shows the blocking performance in terms of the intruder's pure time delay for each version of the USV's logic. The best performance is reached by the version 30 of the logic and amounts to 42 seconds of pure delay in median. This can be

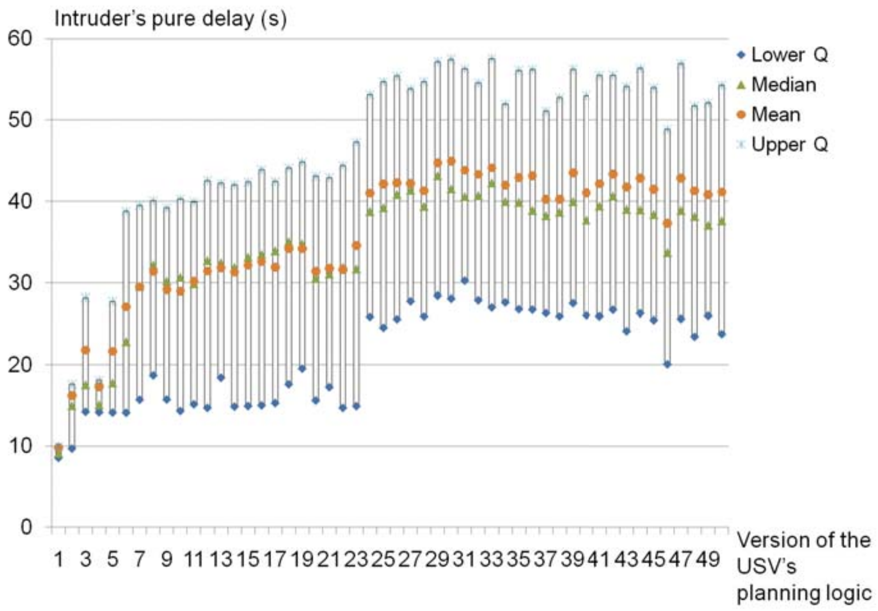


Fig. 12.11 Evaluation of the USV's blocking performance. The performance is expressed as a pure time delay applied on the intruder. Each version of the USV's logic was evaluated in 800 runs.

compared to the pure time delay of 46 seconds in median imposed by the hand coded USV on the same intruder. This result thus shows that the best performance of the automatically generated USV's logic closely approaches the blocking performance of the hand coded logic.

The automated generation of the logic took approximately 1 day to generate the main navigation controller and approximately 3 days on the average to generate navigation plans for 49 automatically defined states of exception. Its overall size reached 900 lines of code. From the set of 10 experimental runs, only 2 were able to find logic with the similar performance to the best one. The remaining 8 runs prematurely stagnated due to over-specialization of some of the evolved navigation plans. Even a single defective navigation plan synthesized for one of the key situations can significantly influence the performance of the whole logic. This shows that the learning of the USV's logic against the intruder utilizing attacking logic interspersed with randomized actions is a challenging task.

The first few versions of the logic have low performance as they contain only a few navigation plans describing specialized maneuvers for a small number of key situations. However, as the learning process progresses, more and more navigation plans handling new situations are added and the overall performance gradually improves. This continues until the version 30 of the logic after which the performance stagnates. This can be attributed to difficulty in solving new complex situations in which problems with the generalization of navigation plans arise.

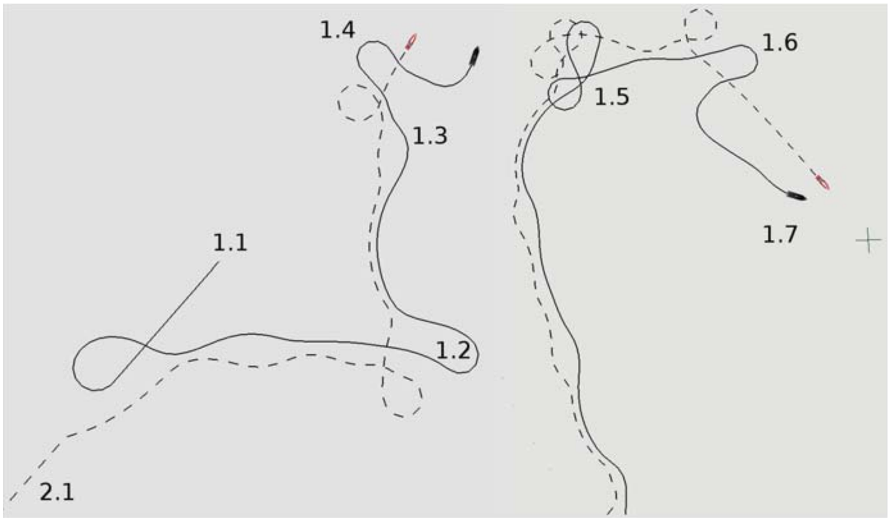


Fig. 12.12 Example of a run in which the USV managed to block the intruder for 45 seconds. The start position of the USV is marked as 1.1, while the start position of the intruder is marked as 2.1.

An example of a run in which the USV reached 45 seconds of pure time delay imposed on the intruder is shown in Figure 12.12. The USV starts at the location 1.1, while the intruder starts at the location 2.1. The first situation in which the USV executes a specific maneuver is marked as 1.2. In this situation, the USV steers sharply to the left in an attempt to intercept the intruder. The run continues until 1.3 where the USV attempts to deflect the intruder's heading by first carefully turning to the left and then aggressively blocking from the side. The intruder, however, instantly responds by executing a sharp left turn, which makes the USV to take another trial in intercepting him in the situation 1.4. Yet the USV overshoots in the process of blocking. The run continues for the next 23 seconds all the way up to the target. In the situation 1.5, the intruder executes a random sequence of two sharp turns to deceive the USV and thus to increase its chances for the attack. The USV, however, successfully follows and takes another attempt in intercepting the intruder but overshoots in 1.6 and the intruder finally reaches its goal 1.7.

12.5 Conclusions

We have presented a new approach for automated synthesis of a symbolic planning logic for an autonomous unmanned vehicle operating in an environment with a deceptive adversary. The idea behind this approach is to evolve an initial version of the logic first and then further improve its performance by evolving additional components that can reliably handle specific situations that can arise during the mission.

We used GP technique for automated generation of navigation plans for corresponding automatically extracted states of failure.

In the context of our test mission, we developed a mission planning system to automatically generate a planning logic for USV to block the advancement of an intruder boat toward a valuable target. The USV's logic consists of a navigation controller and multiple navigation plans describing specific maneuvers for specific situations. The intruder is human competitive and exhibits deceptive behavior so that the USV cannot exploit any regularity in its attacking tactic for blocking.

In our experiments, we compared the performance of the hand coded USV's blocking logic to the performance of the logic that was automatically generated. The results showed that the performance of the automatically generated USV's logic (42 seconds of pure delay in median) closely approaches the performance of the hand coded USV's logic (46 seconds). Both types of the USV's logic were evaluated against a human competitive intruder. Hence, the approach described in this chapter clearly demonstrates the viability of automatically synthesizing planning logic for autonomous unmanned vehicles in competitive environments with deceptive adversaries.

Acknowledgments

This research has been supported by Office of Naval Research and NSF Grants OCI-0636164 and CMMI-0727380. Opinions expressed here are those of the authors and do not necessarily reflect opinions of the sponsors.

References

[1] Lipson, H., Antonsson, E., Koza, J., Bentley, P., Michod, R., Alber, R., Rudolph, S., Andronache, V., Scheutz, M., Arzi-Gonczarowski, Z.: Computational synthesis: from basic building blocks to high level functionality, pp. 24–31 (2003)

[2] Koza, J.R., Keane, M.A., Streeter, M.J., Mydlowec, W., Yu, J., Lanza, G.: Genetic Programming IV: Routine Human-Competitive Machine Intelligence. Kluwer Academic Publishers, Dordrecht (2003)

[3] Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. MIT Press, Cambridge (1992)

[4] Goldberg, D.: Genetic Algorithms in Search and Optimization. Addison-Wesley, Reading (1989)

[5] Floreano, D., Mattiussi, C.: Bio-inspired artificial intelligence: theories, methods, and technologies (2008)

[6] Barate, R., Manzanera, A.: Automatic Design of Vision-based Obstacle Avoidance Controllers using Genetic Programming. In: Monmarché, N., Talbi, E.-G., Collet, P., Schoenauer, M., Lutton, E. (eds.) EA 2007. LNCS, vol. 4926, pp. 25–36. Springer, Heidelberg (2008)

[7] Nehmzow, U.: Physically embedded genetic algorithm learning in multi-robot scenarios: The PEGA algorithm (2002)

[8] Dain, R.: Developing mobile robot wall-following algorithms using genetic programming. Applied Intelligence 8, 33–41 (1998)

[9] Dupuis, J., Parizeau, M.: Evolving a vision-based line-following robot controller, p. 75 (2006)

[10] Koza, J., Rice, J.: Automatic programming of robots using genetic programming, pp. 194–194 (1992)

[11] Gajda, P., Krawiec, K.: Evolving a vision-driven robot controller for real-world indoor navigation. In: Giacobini, M., Brabazon, A., Cagnoni, S., Di Caro, G.A., Drechsler, R., Ekárt, A., Esparcia-Alcázar, A.I., Farooq, M., Fink, A., McCormack, J., O'Neill, M., Romero, J., Rothlauf, F., Squillero, G., Uyar, A.Ş., Yang, S. (eds.) EvoWorkshops 2008. LNCS, vol. 4974, pp. 184–193. Springer, Heidelberg (2008)

[12] Barlow, G., Oh, C.: Evolved Navigation Control for Unmanned Aerial Vehicles. In: Frontiers in Evolutionary Robotics, p. 596. I-Tech Education and Publishing, Vienna (2008)

[13] Haynes, T., Sen, S.: Evolving behavioral strategies in predators and prey. In: Weiss, G., Sen, S. (eds.) IJCAI-WS 1995. LNCS, vol. 1042, pp. 113–126. Springer, Heidelberg (1996)

[14] Buason, G., Bergfeldt, N., Ziemke, T.: Brains, bodies, and beyond: Competitive co-evolution of robot controllers, morphologies and environments. Genetic Programming and Evolvable Machines 6, 25–51 (2005)

[15] Jaskowski, W., Krawiec, K., Wieloch, B.: Winning Ant Wars: Evolving a Human-Competitive Game Strategy Using Fitnessless Selection. In: O'Neill, M., Vanneschi, L., Gustafson, S., Esparcia Alcázar, A.I., De Falco, I., Della Cioppa, A., Tarantino, E. (eds.) EuroGP 2008. LNCS, vol. 4971, pp. 13–24. Springer, Heidelberg (2008)

[16] Togelius, J., Burrow, P., Lucas, S.: Multi-Population Competitive Co-evolution of Car Racing Controllers, pp. 4043–4050 (2007)

[17] Doherty, D., O'Riordan, C.: Evolving Agent-Based Team Tactics for Combative Computer Games (2006)

[18] Richards, M., Whitley, D., Beveridge, J., Mytkowicz, T., Nguyen, D., Rome, D.: Evolving cooperative strategies for UAV teams, p. 1728 (2005)

[19] Shichel, Y., Ziserman, E., Sipper, M.: GP-Robocode: Using Genetic Programming to Evolve Robocode Players, pp. 143–154 (2005)

[20] Sipper, M., Azaria, Y., Hauptman, A., Shichel, Y.: Designing an Evolutionary Strategizing Machine for Game Playing and Beyond. IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews 37, 583–593 (2007)

[21] Schwartz, M., Svec, P., Thakur, A., Gupta, S.K.: Evaluation of Automatically Generated Reactive Planning Logic for Unmanned Surface Vehicles. In: Performance Metrics for Intelligent Systems Workshop, Gaithersburg, MD (2009)

[22] Cornfield, S., Young, J.: Unmanned Surface Vehicles - Game Changing Technology for Naval Operations. In: Roberts, G.N., Sutton, R. (eds.) Advances in Unmanned Marine Vehicles, 69th edn., The Institution of Electrical Engineers, Stevenage, United Kingdom (2006)

[23] Finn, A., Scheding, S.: Developments and Challenges for Autonomous Unmanned Vehicles: A Compendium. Springer, Heidelberg (2010)

[24] Committee on Autonomous Vehicles in Support of Naval Operations, National Research Council. The National Academies Press, Autonomous Vehicles in Support of Naval Operations (2005)

[25] Thakur, A., Gupta, S.K.: A Computational Framework for Real-Time Unmanned Sea Surface Vehicle Motion Simulation. In: ASME 2010 International Design Engineering Technical Conferences (IDETC) & Computers and Information in Engineering Conference (CIE), Montreal, Canada (2010)
[26] Svec, P., Schwartz, M., Thakur, A., Anand, D.K., Gupta, S.K.: A simulation based framework for discovering planning logic for Unmanned Surface Vehicles. In: ASME Engineering Systems Design and Analysis Conference, Istanbul, Turkey (2010)
[27] LaValle, S.: Filtering and Planning in Information Spaces (IROS tutorial notes) (2009)
[28] Brooks, R.A.: Intelligence Without Representation. Artificial Intelligence 47, 139–159 (1991)
[29] Brooks, R.: A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation 2, 14–23 (1986)
[30] Poli, R., Langdon, W., McPhee, N.: A Field Guide to Genetic Programming. Lulu Press (2008)

Chapter 13
Major Feedback Loops Supporting Artificial Evolution in Multi-modular Robotics

Thomas Schmickl, Jürgen Stradner, Heiko Hamann, Lutz Winkler, and Karl Crailsheim

Abstract. In multi-modular reconfigurable robotics it is extremely challenging to develop control software that is able to generate robust but still flexible behavior of the 'robotic organism' that is formed by several independent robotic modules. We propose artificial evolution and self-organization as methodologies to develop such control software. In this article, we present our concept to evolve a self-organized multi-modular robot. We decompose the network of feedbacks, that affect the evolutionary pathway and show why and how specific sub-components, which are involved in these feedbacks, should be subject of evolutionary adaptation. Self-organization is a major component of our framework and is implemented by a hormone-inspired controller governing the behavior of singular autonomous modules. We show first results, which were obtained by artificial evolution with our framework, and give an outlook of how the framework will be applied in future research.

13.1 Introduction

Evolutionary multi-modular robotics (EMMR) is a rather novel approach in the fields of biology, computer science and engineering. It outnumbers 'classical' evolutionary robotics concerning technical challenges: Evolving a functional controller

Thomas Schmickl · Jürgen Stradner · Heiko Hamann · Karl Crailsheim
Artificial Life Lab of the Department of Zoology
Karl-Franzens University Graz
e-mail: thomas.schmickl@uni-graz.at

Lutz Winkler
Institut für Prozessrechentechnik, Automation und Robotik (IPR)
Karlsruher Institut fr Technologie - KIT
e-mail: winkler@ira.uka.de

for a predefined fixed robotic morphology is already a challenging goal to reach [1, 2, 3]. Additionally in multi-modular robotics, a huge variety of robot morphologies are built from a set of joined robot modules. See Fig. 13.1 for an example of such a robotic organism. Each of these robots is controlled by a control program, which – in the joined organism – fuses into one meta-controller that moves the whole body. It is not just the set of these controllers that determines the final behavior of the organism, but also the set of physical constraints that are posed by the way of how the modules are coupled (joints, forces, ...).

In our EMMR approach, the robotic controllers should evolve along with the body shape. In addition, controllers of single modules should evolve in a way that enables them to build the joined organism shape from a former unconnected (swarm) mode of operation. We suggest a bio-inspired self-organized process [4, 5] that governs the organism formation in a decentralized way. As it is possible for robotic modules to fail or to end up in an unfavorable place in the organism, the organism's control should be extremely robust but still flexible enough to allow dynamic replacement or displacement of single robotic modules during runtime. Thus, the desired controllers, that we plan to evolve, are described by the following characteristics: decentralized control, self-organization, robust behavior, flexibility and scalability. These characteristics are typical for 'swarm-intelligent' systems and therefore we attribute our organism formation process and organism movement process to be a variant of evolutionary swarm robotics [6, 7].

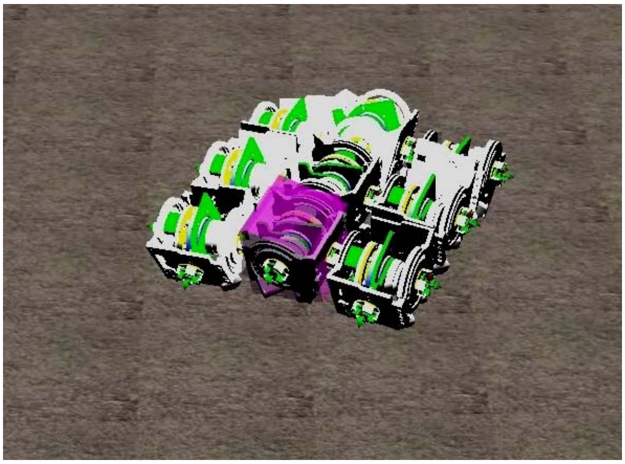


Fig. 13.1 Exemplary configuration of 9 robots arranged and coupled in a 3×3 layout in our proposed EMMR approach. The process that is able to form this body shape from a swarm of autonomously moving single robot modules has to be evolved. Artificial evolution should then also generate controllers that are able to move this robotic organism in a self-organized way. Multiple feedback loops, that allow self-organization to work at specific points of control, is proposed to enhance and support artificial evolution.

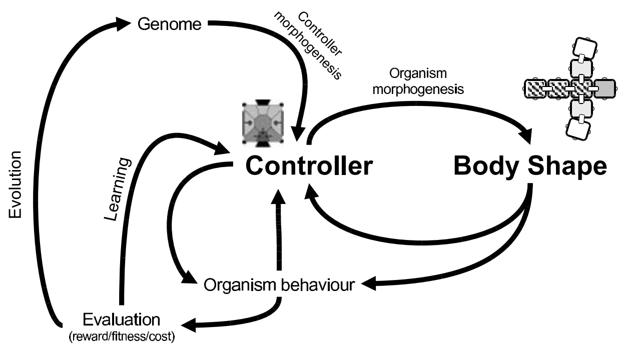


Fig. 13.2 The feedback loops that affect the evolution of organism shapes in our proposed EMMR system.

Several approaches have been proposed to achieve this goal: The studies of Shen et al. [8] suggest a framework in which artificial hormones that resemble hop-counts and messages exchanged among modules are used, instead of hard-coded IDs and 'gait table' numbers to coordinate a multi-modular robotic system. In [9], a robotic swarm mimics pheromone excretion of biological organisms and achieves swarm control in doing so. Also in [10] a hop-count-based system is used to control a robot swarm. Similar methods of hop-counts, which form linear gradients in the organism, were also used in [11] to construct dense objects from autonomously moving sub-units. A continuous gradient approach for navigation of modules based on non-linear gradients was investigated in [12] and in [13] within the I-SWARM project [14]. Based on these swarm techniques, we elaborated a hormone-inspired control paradigm for body formation and body control, aimed for multi-modular robots used in the EU-funded projects SYMBRION [15] and REPLICATOR [16].

In this article, we describe the artificial homeostatic hormone system (AHHS) which we've applied successfully to control a single robot in simulation [17] and in robotic hardware [18]. Using single robots, an AHHS was successfully evolved to move using a 'screw drive', which is non-trivial to control, to avoid obstacles, and to explore the arena [17]. Currently, we develop a system of artificial evolution (AE), that allows an elaboration of this AHHS controller: Our novel controller will be able to control the self-organized body formation process as well as the decentralized control of locomotion of joined robotic organisms. In the following, we describe the concept of our AHHS and discuss the major feedback loops (Fig. 13.2) that emerge within the system of AE and organism formation. Some of these feedback loops are not existent in 'classical' evolutionary robotics (ER) concerning single robots, others are missing in non-evolutionary multi-modular robotics.

The expected main advantages of this approach compared to others (e.g., classic approaches, artificial neural networks) are an intrinsic spatiality (hormone gradients in connected robots) and a supposed high evolvability (smooth fitness landscapes through mutations that gradually change the behavior). Explicitly defined hormone gradients, that span the whole robot organism, are exploited in the robot organism morphogenesis. The controller of the robot organism is embodied due to the hormone concentrations that are stored in the robot modules. Our hormone controller defines the resulting behavior through hormone production rates, decay rates, and hormone interaction rules that are gradually changed through mutations and therefore only gradually change the behavior. Thus, this approach promises to be successfully applied in EMMR scenarios.

In the following, we identify six feedback loops: classic control, learning, evolution, controller morphogenesis, robot organism morphogenesis, and body motion. In first case studies, we have tested the classic control loop in robotic hardware and in simulation [17, 18] and evolution in simulation [17]. In addition, the controller morphogenesis and the robot organism morphogenesis were tested in preliminary studies.

13.2 Artificial Homeostatic Hormone System

The basic characteristics and the implementation of our bio-inspired controller are described in [19]. The idea of an AHHS controller is inspired by second-messengers which communicate and 'compute' stimuli received through membrane-bound receptors in evolutionary 'simple' unicellular organisms (protozoa), bacteria and slime mold. In higher life-forms, such cell messengers act inside of cells and hormones allow to broadcast communication between tissues.

Stimuli received by robot sensors basically trigger the release of virtual hormones in an AHHS controller. The inner body of a single robot module is spatially represented by (virtual) compartments. Each sensor triggers the production of a specific hormone in the compartment with which it is associated. Virtual hormones decay over time, and diffuse to neighboring compartments. This allows information about current and past sensor activation to spread throughout the whole virtual 'internal body' of the robot. In an AHHS, hormones interact with each other: One hormone potentially increases or decreases the level of another hormone and is able to alter the sensitivity of sensors and/or actuators. Finally, at least one hormone has to activate one of these available actuators to manifest the robot's final behavior.

As a result of this actuation, future sensor stimulation is altered. Hence, a sensor–controller–actuator feedback loop emerges. From a cybernetic point of view [20], our AHHS controller actuates the robot such that specific hormone levels are kept at a homeostatic state.

13.2.1 Artificial Genome

Evolution provides an essential feedback loop in our proposed EMMR. As evolution always operates on a genome, which is the 'substrate' for adaptation, the specific configuration of an AHHS has to be kept persistent in a data structure that we call 'genome'. From this data structure, the AHHS controller has to be parametrized. The genome of our AHHS consists of two logical entities: *hormone chromosome* and *rule chromosome*. The hormone chromosome holds only one gene per hormone.

In contrast, the rule chromosome contains an arbitrary number of genes for each hormone. Each hormone excretion, each type of hormone-to-hormone interaction and each actuator activation by a hormone is described in a separate rule gene.

In the following, we give a detailed description of the data structure we developed for holding the needed genetic information of an AHHS (reprinted from [17]):

The hormone chromosome contains the following parameters:

- *hormone ID*
- *fixed decay rate*
- *diffusion coefficient*
- *maximum value of hormone* (value at which a saturation is reached)
- *base production rate* (amount that is produced per time step without sensory stimulation)

The rule chromosome contains the following parameters:

- *rule type*: condition to be met or triggering action
  1. **always:** Action triggered independent from threshold σ
  2. **greater than:** Action triggered if greater than threshold σ
  3. **smaller than:** Action triggered if smaller than threshold σ
- *trigger type*: type of triggered action (hormone concentration θ, actuator value α)
  1. **never triggered:** No action performed.
  2. **sensor influences hormone:** if $(\gamma(t) > \sigma)$ then $\theta(t+1) = \theta(t) + \gamma(t)\delta + \beta$ (sensor value γ)
  3. **hormone influences actuator:** if $(\theta(t) > \sigma)$ then $\alpha(t+1) = \alpha(t)\delta + \beta$
  4. **hormone influences other hormone:** if $(\theta_1(t+1) > \sigma)$ then $\theta_2(t+1) = \theta_2(t) + \theta_1(t)\delta + \beta$
  5. **hormone influences itself:** $\theta(t+1) = \theta(t) + \theta(t)\delta + \beta$

All of these values are integer values allowing fast execution on limited (embedded) hardware.

13.3 Feedback 1: Classic Control

The direct feedback loop between the controller and the behavior represents the classic approach of control theory. In control theory this loop is interpreted as a negative feedback because an error value is determined by subtracting the measured system state from the desired state. This error value is used to determine the new input. The controller checks the difference between the desired state and the measured state of the whole system (robot organism and environment) through its sensors. If there is a difference the controller changes the 'system input' (e.g., actuator input signals) that is fed into the system.

13.4 Feedback 2: Learning

The feedback loop controller–behavior–evaluation represents the field of unsupervised machine learning. The robot is interpreted as an agent that has to take actions in an environment in order to maximize a reward. The robot evaluates its behavior online, changes its controller and, hence, its behavior. There is a huge variety of possible approaches. An artificial neural network could be trained online, standard reinforcement learning techniques such as Q-learning could be applied, or even our novel controller approach could be used. The rules of such an AHHS controller can be optimized through learning. This could be done either as a complete learning task from scratch or by optimizing an evolved controller.

13.5 Feedback 3: Evolution

The loop controller–behavior–evaluation–evolution–genome is of high importance in our standard AE [21]. Hence, we produce a population of robot controllers that are evaluated and selected based on their fitness. A new generation is generated through mutation and recombination of the controllers.

Currently we have implemented a naive genetic algorithm to test first evolutionary approaches. In Fig. 13.3 the class diagram of our software design is shown. It consists basically of three classes: `EvolutionManager` (maintains the whole evolutionary process) which keeps a population of type `Evolvable` (contains evolution specific values such as fitness values) which holds a collection of type `AbstractController` (a container for the actual specific robot controller) for each module in the robot organism. Usually we have homogeneous organisms, that is, we have identical controllers for each robot module in the organism.

The currently evolved controller design is our AHHS controller. However, the software framework is independent from the specific controller design as far as possible – other approaches, such as artificial neural networks, could be used as

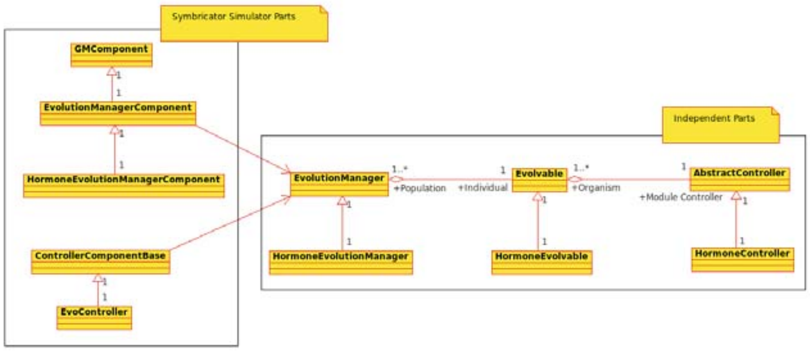


Fig. 13.3 Software design of our AE framework. It is embedded into the projects' 'Symbricator Simulator' which is based on the Delta3D open-source gaming/simulation engine [22].

well with few adjustments. Typically the first evolution run is initiated with a small population (20 to 30 individuals) of randomly generated AHHS controllers. These random controllers generate rather erratic behavior that is evaluated in simulation. The ‘Symbricator Simulator’, that was developed in both EU-projects REPLICATOR and SYMBRION, is based on the Delta3D open-source gaming/simulation engine [22]. The simulator provides a full simulation of physics, which is indispensable as the locomotion of our multi-robot organisms will usually depend on friction and statics. In addition, it is possible to import the CAD data of the current robot prototype design. For a limited time the behavior of the robot organism is evaluated. For example, in case we evolve simple collision avoidance behavior the evaluation can be based on the covered distance. Following [23] this type of fitness function is called ‘aggregate fitness function’ because it selects for high-level success (instead of rewarding any kind of motion).

The key challenges in the evolutionary approach to modular robotics are the high computational costs of the controller evaluations and the selection of an appropriate controller design. Due to computational costs only small numbers of generations are feasible within which a valid controller has to be found. Thus, we need a controller that is not only able to represent the desired behavior, but also a controller that shows high evolvability. With ‘high evolvability’ we refer to a fitness landscape that is as smooth as possible, because it is the preferred shape to avoid local optima. The shape of the fitness landscape is partially influenced by the controller design in connection with the mutation operator but also by the environment. Discrete (stepwise) changes in the controller by the mutation operator should be avoided, because the application of the mutation operator would most likely result in very different behavior and, thus, in very different fitness values. However, typically there is a trade-off between increasing the size of the search space and avoiding discrete changes through mutation.

13.6 Feedback 4: Controller Morphogenesis

In our AHHS controller, the compartmentalization of the inner body of a single robot module is an important feature. It allows ‘embodiment‘ of the controller, because sensors are allowed to trigger hormone excretion only in those compartments spatially associated with the sensor location on (or in) the robot’s body. Only hormones of the same compartment interact, this way the computation being performed in the AHHS, is localized. Therefore, the structure formed by the compartments is important for the behaviors generated by the AHHS. We made the compartmentalization a subject of AE as well and introduced another ‘rule chromosome’ (see section 13.2.1).

This chromosome contains genes that parametrize a process that forms the compartment structure. One way to achieve internal compartmentalization, is to use a different set of AHHS rules in a ‘constructor’ phase before the robot controller is started. During this phase, hormone values trigger rules in the AHHS from this third chromosome. The only difference compared to the second ‘rule chromosome’ (described in section 13.2.1) is that hormone values in this phase do not trigger an actuator of the robot. Instead, they trigger a division of one compartment into two compartments, similar to cell divisions in biological organisms: At the beginning, the AHHS starts with just one compartment. This compartment is then successively divided depending on local hormone values. Hence, a self-organized process creates the compartment structure, which is later affecting the robot's behavior.

AE alters the gene information on this chromosome by altering, deleting, and duplicating rules, by changing the initial starting conditions or by changing the length of the transient period. Fig. 13.4 shows exemplarily how the compartment structure is altered by a combination of two loci for point-mutations.

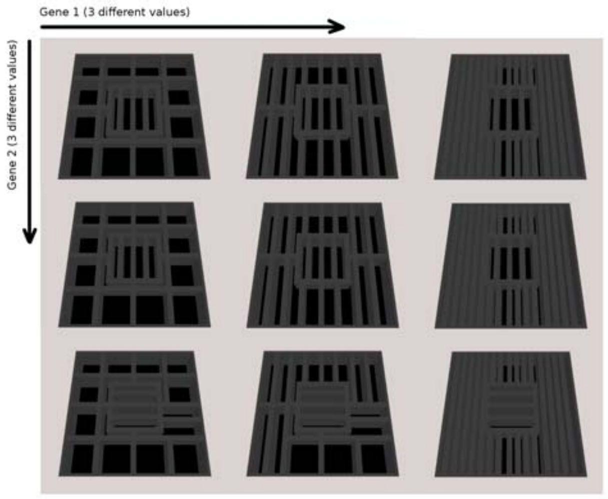


Fig. 13.4 Internal compartmentalization of the robot. This important structural feature in an AHHS controller is mutated by altering 'layout rules'. This figure shows 9 configurations that result from a combination of mutations of 2 genes (rules).

13.7 Feedback 5: Robot Organism Morphogenesis

When it comes to building and reconfiguring robot organisms that consist of autonomous robot modules, we suggest that our AHHS is able to perform this task in a self-organized manner. Thus, the feedback loop 'controller – body shape' (Fig. 13.2) emerges automatically. The main problem concerning the morphology of the robot organism is the trade-off between robustness and flexibility.

We think that there is no conceptional difference between to building a robot organism out of a swarm of single modules, on the one hand, and the reconfiguration process, on the other hand. In most cases for both processes, it is a precondition that an additional number of nearby single robot modules is available. If a join or a change of the morphology of the organism body shape is triggered by the environment, this trigger event has to be perceived by at least one of the modules and it needs to be communicated to other modules.

This kind of body formation process is depicted in Fig. 13.5. In step one, a module (marked by an exclamation point) detects a situation which is infeasible for a single module which serves as a trigger or a seed for the action of joining together. This perception is communicated to nearby single robots. These modules dock on the opposite of the detected seed. For example, starting from the module that started the joining progress, for example, a simple line is formed.

In such a joined organism (Fig. 13.5: step two), further environmental stimulation triggers the production of other hormones inside the organism, which consists of connected modules. This process results in the emergence of a gradient of hormone concentrations within the organism. The still existing sensor input, initially triggered the body formation, can now serve as a trigger for a differentiation into a head module and a tail module. Furthermore, a threshold of a 'head-' and a 'tail-hormone' determines, for example, the positions of legs in the middle of the organism (Fig. 13.5: step three). Despite the fact that this threshold is predefined, the body shape of the robot organism is not determined but influenced by environmental inputs. In this way, different body shapes are established by a self-organized reconfiguration process. The building of legs is based on the same principles as the process of building the main body.

We prefer this approach of exploiting self-organization processes as the main design paradigm in favor of non-adaptive approaches (e.g., predefined shapes) because the latter would lack any flexibility. The approach of self-organization described here in connection with evolutionary methods automatically influences the shape of the robot organism when a new or changed seed is detected by a (joined or free) module. The possibility of self-reconfiguration gives the organism the needed plasticity and adaptability.

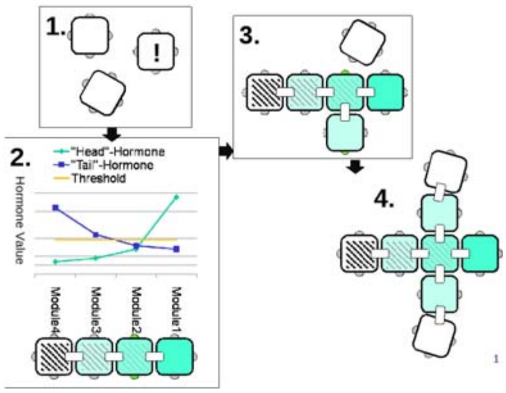


Fig. 13.5 The development of the body formation in an EMMR. The process of the progress from single module formation in a swarm to robot organism with legs is depicted in four steps. One possible way of achieving this with our AHHS is denoted as a schematic graph of hormone values of two hormones in step 2. For further explanation see text.

13.8 Feedback 6: Body Motion

In our AHHS control paradigm, there is, in principal, no difference between motion of individual robot modules and of joined robotic organisms. The parallel behavior of single modules sums up to the organisms behavior. Of course, there is a demand of coordination among the modules to achieve a regular motion of the organism. To allow this, hormones diffuse to neighboring robot modules, as soon as modules dock to each other. Hence, the internal body of the organism is structured (compartmentalized) as it is the case for a single robot. Therefore, a robot organism consists of two levels of compartmentalization. There is the logical level inside each single module and the physical level of connected modules.

To demonstrate the diffusion of hormones between robot modules, we performed AE with already joined robot organisms that were allowed to actuate only their 'hinges', which are the main actuators that bend the robot modules with an angle of $\pm 90^\circ$ from the default configuration. No wheels or screw-drives were allowed to be activated. In the following, we shortly describe an exemplary incremental course of AE in our framework:

13.8.1 Step 1: The First Oscillator

In a first period, we coupled two modules. For this organism, the only chance to move was to evolve a set of rules in the AHHS for both modules that actuate both hinges in an 'oscillatory way'. We used the distance the organism moved within 300 time steps as fitness function. The fittest controllers were selected and were subject to point mutation and cross-over producing 20 offspring. The three best individuals were moved to the next generation without any change (elitism) and two new AHHS controllers were generated randomly from scratch in each generation. A behavior that significantly moved the organism evolved within the first 10 generations in a population of 25 AHHS controllers. It increased its performance within the next 20 generations significantly. Fig. 13.6 shows snapshots of this organism's behavior.

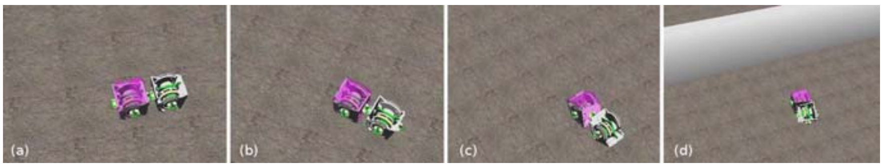


Fig. 13.6 Evolved motion of two joined robot modules in the projects' 'Symbricator Simulator'. The hinges of the two modules are contracted in an oscillations by the evolved AHHS. This pushes the organism forward.

13.8.2 Step 2: Motion of Bigger Organisms

We implanted this oscillating AHHS into robot organisms of increasing size by just adding robot modules at one end of the organism. All of these organisms were able to move slowly. The speed was significantly reduced compared to the former, smaller organisms. After 10-15 generations, the motion speed recovered almost to the prior level again, suggesting that AE successfully adapted the pre-evolved AHHS controller to the new body size. Finally, we ended up with a long line of seven connected robots, which nicely moved across the simulated arena in a caterpillar-like movement pattern. Fig. 13.7 shows snapshots of this organisms behavior.

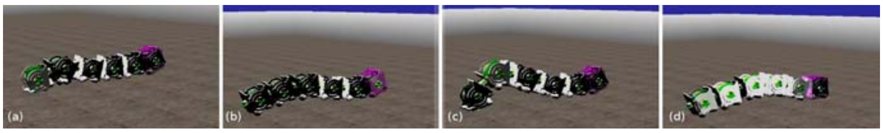


Fig. 13.7 Evolved motion of several joined robot modules in the projects' 'Symbricator Simulator'. The hinges of joined modules are contracted in delayed oscillations by the evolved AHHS. A caterpillar-like motion pattern was finally evolved.

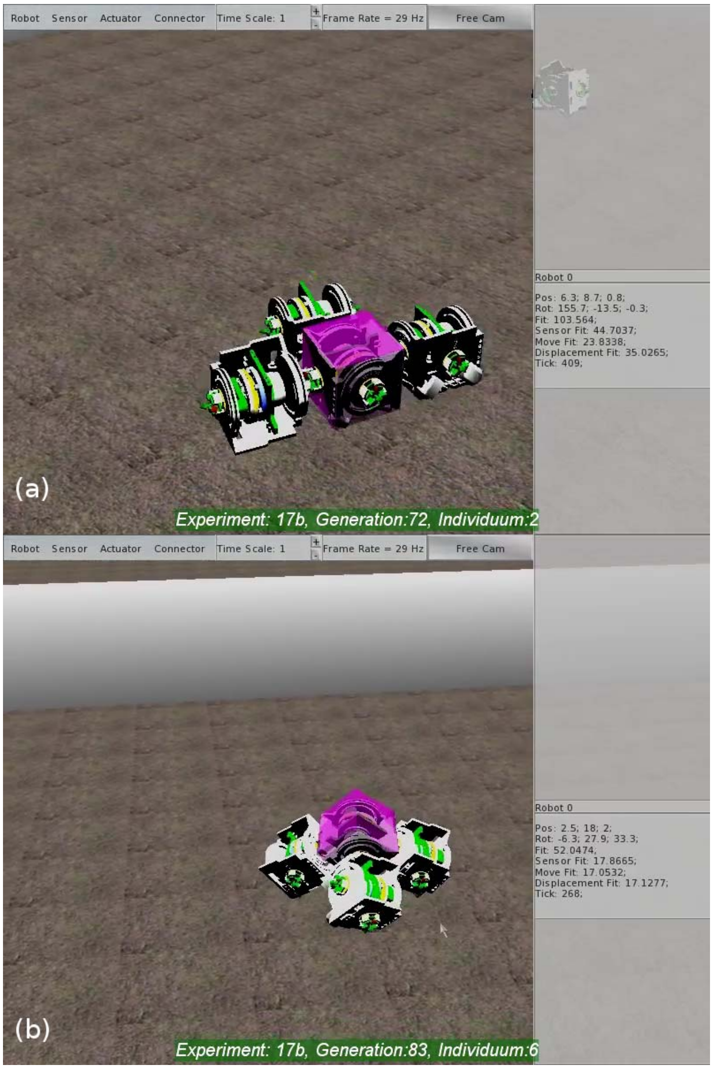
13.8.3 Step 3: Motion of More Complex Organisms

After these successful evolution experiments, we constructed more complex (nested) organism shapes, into which we implanted the pre-evolved AHHS controller described in subsection 13.8.1. All of these shapes evolved well-adapted AHHS controllers that were able to move the organism in the arena. Here we just want to discuss one example that underlines how the body shape influences the body movement: Fig. 13.8a and Fig. 13.8b show two different motion strategies that evolved for the same body shape successively: First, the outer two branches of the T-shaped organism move the organism by oscillatory contraction and release of their hinge while the 'tail' in the back pushes the organism further as well. The whole body is laying almost flat on the floor (Fig. 13.8a). Then, a different movement pattern emerges in evolution: The central module contracts its hinge which erects the whole organism. This way the three branches of the T-shaped body act like legs and the 'tripod' successfully moves through the arena (Fig. 13.8b).

13.9 Discussion

Here, we describe several feedback loops that affect body formation and body movement in an EMMR system. Based on the involved feedbacks, we characterize six levels of adaptation that are exploited by ourselves to generate a bio-inspired adaptive reconfigurable robotic system:

- **Classic control:** The controller–behavior feedback loop is always present in any reactive agent, thus also in any autonomous robot that is able to perform behavior of any kind in its environment. We did not elaborate on this 'classic' feedback loop in the concept presented in this article.



Robot 0
Pos: 6.3; 8.7; 0.8;
Rot: 155.7; -13.5; -0.3;
Fit: 103.564;
Sensor Fit: 44.7037;
Move Fit: 23.8338;
Displacement Fit: 35.0265;
Tick: 409;

Experiment: 17b, Generation:72, Individuum:2

Robot 0
Pos: 2.5; 18; 2;
Rot: -6.3; 27.9; 33.3;
Fit: 52.0474;
Sensor Fit: 17.8665;
Move Fit: 17.0532;
Displacement Fit: 17.1277;
Tick: 268;

Experiment: 17b, Generation:83, Individuum:6

Fig. 13.8 Two different motion patterns evolved successively with the same body shape. a: flat body, oscillators move peripheral hinges like fins. b: erected posture of the organisms, peripheral robots moved like legs.

- **Learning:** This feedback adapts the controller during runtime, based on the recent dynamics of the so called 'reward', 'fitness' or 'cost' function. We did not elaborate on this feedback loop in the concept presented in this article.
- **Evolution:** In this feedback loop, the main concern is feasibility due to high computational costs. Self-organizing processes generated by the general controller design, such as homeostatic tendencies in the hormone controller, need to be leveraged as well to obtain smooth fitness landscapes and to decrease the number of generations that are necessary before the desired behavior is evolved.
- **Controller morphogenesis:** In this article we showed in this article that the internal structure of the AHHS controller arises from a dynamic self-organized process, driven by the AHHS itself. Hence, it is subject to AE, together with the other rule set that acts in the AHHS. This compartmental layout is an essential feature to allow 'embodiment' in our approach.
- **Robot morphogenesis:** For the feedback loop of the controller and the body shape we propose a dynamical, self-organized body shaping process which influences the characteristics of the controller. When single modules are docking to or releasing from the robot organism the hormone values are altered and therefore the behavior of the controller itself changes. We think that this approach for a self-organized body formation process in combination with evolutionary learning of the controller could be able to perform the demanding task of flexible body shape.
- **Decentralized body motion:** Body-motion of joined organisms was successfully achieved by AHHS control and by our implementation of AE. Again, it is a self-organized process – consisting of positive and negative localized feedbacks and time delays – that achieves the desired motion patterns.

In our current research projects, we plan to implement all six feedback loops described above in real robotic hardware and in a sophisticated simulation software, that closely depicts the physical abilities and constraints, as well as the computational abilities of our final targeted robots [22]. Using this software, we already successfully implemented our AHHS controllers, evolved them to perform adaptive behavior on single robots and in joined robotic organisms. Morphogenesis of the controller and morphogenesis of the robot organism will be our next focus, as well as enhancing the efficiency, the computational power and the evolvability of our AHHS controllers. Also the necessity of all six feedback loops will be investigated. At the moment, each feedback loop is investigated separately. For example, the body motion was investigated with fixed predefined body shapes. In case of learning and evolution, we note that it is not necessary to have both of them in the system at the same time. In principle, they just differ in their time scales. Learning is achieved during a life time while evolution lasts over generations. For example, there might be scenarios in which learning does not improve the performance significantly because no optimization during runtime is needed.

However, all feedback loops interact in a complex way, which is the key point of this approach. For example, a change of the body shape through robot morphogenesis influences the controller and the body motion. These intertwined feedback loops encourage and challenge evolution to generate adaptive behavior.

We conclude that our AHHS approach allows for self-organization on multiple levels of the organism's formation and movement process. Our evolutionary framework alters the whole genome, which encodes for almost all parameters affecting the feedback networks mentioned above. This genome-based evolution allows us to evolve controller layouts, controller performance rules, virtual physics and virtual chemistry of hormones, organism formation and organism movement all-together in parallel. We think, this multi-level adaptation is essential to create a functioning EMMR approach, as it is desired in the projects SYMBRION and REPLICATOR.

Acknowledgments

We were supported by the following grants: EU-IST FET project 'SYMBRION', no. 216342, EU-ICT project 'REPLICATOR', no. 216240.

References

1. Mataric, M.J., Cliff, D.: Challenges in evolving controllers for physical robots. Robotics and Autonomous Systems 19(1), 67–83 (1996)
2. Harvey, I., Husbands, P., Cliff, D., Thompson, A., Jakobi, N.: Evolutionary robotics: the sussex approach. Robotics and Autonomous Systems 20(2-4), 205–224 (1997)
3. Floreano, D., Husbands, P., Nolfi, S.: Evolutionary Robotics. In: Siciliano, B., Oussama, K. (eds.) Handbook of Robotics, pp. 1423–1452. Springer, Berlin (2008)
4. Camazine, S., Deneubourg, J.L., Franks, N.R., Sneyd, J., Theraulaz, G., Bonabeau, E.: Self-Organizing Biological Systems. Princeton Univ. Press, Princeton (2001)
5. Seeley, D.T.: When is self-organization used in biological systems? Biological Bulletin 202(3), 314–318 (2002)
6. Marocco, D., Nolfi, S.: Origins of communication in evolving robots. In: Nolfi, S., Baldassarre, G., Calabretta, R., Hallam, J.C.T., Marocco, D., Meyer, J.-A., Miglino, O., Parisi, D. (eds.) SAB 2006. LNCS (LNAI), vol. 4095, pp. 789–803. Springer, Heidelberg (2006)
7. Bonabeau, E., Dorigo, M., Theraulaz, G.: Swarm intelligence. From natural to artificial systems. Santa Fe institute studies in the sciences of complexity. Oxford University Press, Oxford (1999)
8. Shen, W.M., Salemi, B., Will, P.: Hormone-inspired adaptive communication and distributed control for CONRO self-reconfigurable robots. IEEE Transactions on Robotics and Automation, 700–712 (2002)
9. Shen, W.M., Will, P., Galstyan, A., Chuong, C.M.: Hormone-inspired self-organization and distributed control of robotic swarms. Autonomous Robots 17, 93–105 (2004)
10. Payton, D., Daily, M., Estowski, R., Howard, M., Lee, C.: Pheromone robotics. Autonomous Robots 11(3), 319–324 (2001)
11. Stoy, K.: How to construct dense objects with self-reconfigurable robots. In: Christensen, H. (ed.) European Robotics Symposium 2006. springer tracts in advanced robotics, vol. 22, pp. 27–37. Springer, Heidelberg (2006)
12. Schmickl, T., Möslinger, C., Crailsheim, K.: Collective perception in a robot swarm. In: Şahin, E., Spears, W.M., Winfield, A.F.T. (eds.) SAB 2006 Ws 2007. LNCS, vol. 4433, pp. 144–157. Springer, Heidelberg (2007)

13. Schmickl, T., Crailsheim, K.: Trophallaxis within a robotic swarm: bio-inspired communication among robots in a swarm. Autonomous Robots 25(1-2), 171–188 (2008)
14. Seyfried, J., Szymanski, M., Bender, N., Estaña, R., Thiel, M., Wörn, H.: The I-SWARM project: Intelligent small world autonomous robots for micro-manipulation. In: Şahin, E., Spears, W.M. (eds.) Swarm Robotics Workshop: State-of-the-art Survey, pp. 70–83. Springer, Heidelberg (2005)
15. SYMBRION: Project website (2010), `http://www.symbrion.eu`
16. REPLICATOR: Project website (2010), `http://www.replicators.eu`
17. Stradner, J., Hamann, H., Schmickl, T., Thenius, R., Crailsheim, K.: Evolving a novel bio-inspired controller in reconfigurable robots. In: 10th European Conference on Artificial Life (ECAL 2009). LNCS, Springer, Heidelberg (in press, 2010)
18. Stradner, J., Hamann, H., Schmickl, T., Crailsheim, K.: Analysis and implementation of an artificial homeostatic hormone system: A first case study in robotic hardware. In: The 2009 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2009, pp. 595–600. IEEE Press, Los Alamitos (2009)
19. Schmickl, T., Crailsheim, K.: Modelling a hormone-based robot controller. In: 6th Vienna International Conference on Mathematical Modelling, MATHMOD (2009)
20. Wiener, N.: Cybernetics: or Control and Communication in the Animal and the Machine. MIT Press, Cambridge (1948)
21. Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. In: Natural Computing Series, Springer, Heidelberg (2003)
22. Delta 3D: Open-source gaming and simulation engine project website, `http://www.delta3d.org/`
23. Nelson, A.L., Barlow, G.J., Doitsidis, L.: Fitness functions in evolutionary robotics: A survey and analysis. Robotics and Autonomous Systems 57, 345–370 (2009)

Chapter 14
Evolutionary Design and Assembly Planning for Stochastic Modular Robots

Michael T. Tolley, Jonathan D. Hiller, and Hod Lipson

Abstract. A persistent challenge in evolutionary robotics is the transfer of evolved morphologies from simulation to reality, especially when these morphologies comprise complex geometry with embedded active elements. In this chapter we describe an approach that automatically evolves target structures based on functional requirements and plans the error-free assembly of these structures from a large number of active components. Evolution is conducted by minimizing the strain energy in a structure due to prescribed loading conditions. Thereafter, assembly is planned by sampling the space of all possible paths to the target structure and following those that leave the most options open. Each sample begins with the final completed structure and removes one accessible component at a time until the existing substructure is recovered. Thus, at least one path to a complete target structure is guaranteed at every stage of assembly. Automating the entire process represents a step towards an interactive evolutionary design and fabrication paradigm, similar to that seen in nature.

14.1 Introduction

While the use of evolutionary algorithms has gone a long way towards automating the design process, the automated fabrication of evolved designs remains a major challenge. This is especially true for designs with complex morphologies or active components. Some approaches use rapid prototyping [1],[2], with a growing effort to 3D-print active components such as actuators [3] and batteries [4], [5]. Alternatively, other approaches explore the construction of robots from many small modular active units [6]-[9]. As the number of modules increases and their size decreases, manufacturing methods need to scale appropriately. Here we explore a combined

Michael T. Tolley · Jonathan D. Hiller · Hod Lipson
Cornell University, Ithaca, USA
e-mail: `mtt33@cornell.edu`, `jdh74@cornell.edu`,
`hod.lipson@cornell.edu`

evolutionary and assembly process geared towards the design and construction of robots from such components.

A promising fabrication concept is to assemble active components that move about stochastically in a fluidic environment (table 14.1) [10]-[13]. This reduces the power, computation, and actuation demands on the individual modules, making it easier to scale down their size. The components are attracted to a growth substrate and to previously-assembled components to fabricate a target structure. This approach allows for the assembly of the complex shapes that arise out of evolution.

The process we describe in this chapter designs a target structure to fulfill prescribed functional requirements using an evolutionary algorithm, and then uses a second algorithm to plan the error-free assembly of this structure in a stochastic environment. Structures are evolved for prescribed loading conditions using a frequency-based representation [14]. The goal of the evolutionary process is to minimize the strain energy in the structures while keeping their overall mass near a prescribed value.

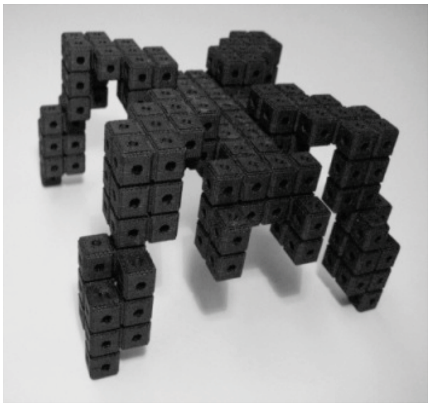
The assembly algorithm then plans the assembly of the best evolved structure without knowing the times or locations of component availability. At each stage of assembly this algorithm must determine the next set of locations to attract modules from among those available. This is accomplished by sampling the graph of all possible paths to the target structure and following those that leave the most options open. For each sample, the assembly problem is solved by beginning with the final structure and working backwards, removing one valid module at a time. Thus each sample is a valid path to a perfect assembly. The potential locations for the next assembly stage are then selected from among those most frequently encountered while sampling. Using this approach, at least one path to a perfectly complete final assembly is guaranteed at every stage of assembly.

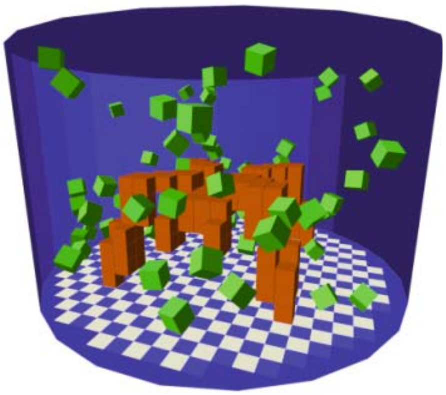
14.2 Target Structure Evolution

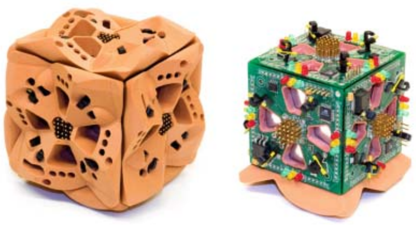
We evolved structures to be assembled from our stochastic modular robots to fulfill a particular task. In this case the task was to support a load on top with ground reaction forces applied only at four locations in the corner of a 20x14x10 cube workspace. The goal was to minimize the strain energy in the structure with 20% of the workspace volume filled. Thus, the resulting structures could be used as a type of customized bridge or a load-bearing legged robot (if the individual modules are capable of some form of actuation).

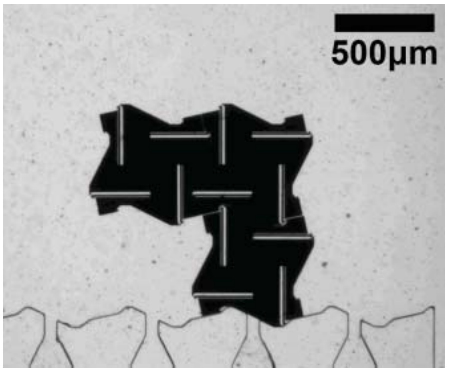
Smooth, freeform 3D structures were evolved using a frequency-based encoding. The objects were represented as a series of frequency amplitudes at harmonic multiples, and the phenotype was generated by applying an inverse discrete Fourier transform to these values. This approach is similar to that used in image compression, such as the jpeg format. In this case, the frequency representation was chosen for its evolvability, scalability in terms of the number of parameters, and ease of implementation. The fitness of the evolved structures was evaluated by calculating

Table 14.1 Stochastic Fluidic Assembly System. Robotic components are assembled by taking advantage of their stochastic motions within a fluidic system for transportation. A target shape (a) is assembled in simulation (b), where free and attached cubes are displayed in green and red, respectively. The assembly substrate is displayed as a checkered floor. (c) Centimeter [15], and (d) micrometer [16] scaled versions of stochastic fluidic assembly experimental modules.


(a)

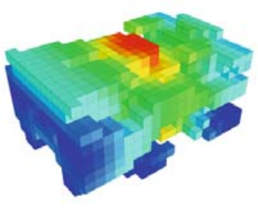

(b)

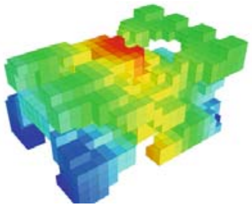

(c)

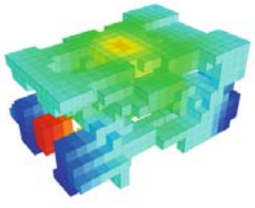


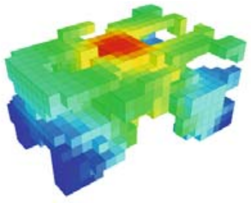
(d)

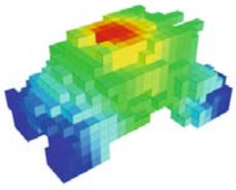
Table 14.2 Digital structure evolution. (a)-(f) Most fit individuals from indicated generations evolved to support a load on top while contacting the ground only at four perimeter locations. The colors indicate the global displacement of each component under the given loading conditions. (g) Plot of the best and average fitness versus the number of evaluations for the first 500 generations. (h) Exaggerated deformation of the final best individual in 5700 generations that was chosen as the target structure for the assembly algorithm.

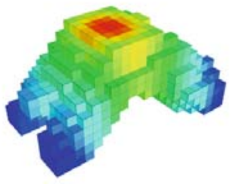

(a) Generation 2

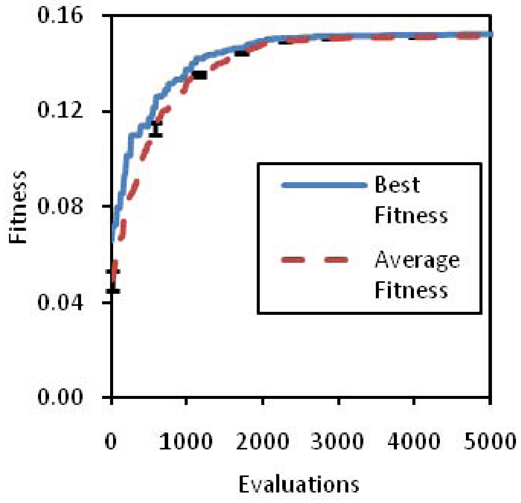

(b) Generation 8


(c) Generation 20

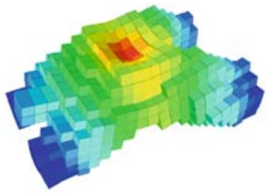

(d) Generation 56


(e) Generation 123


(f) Generation 4862



(g)

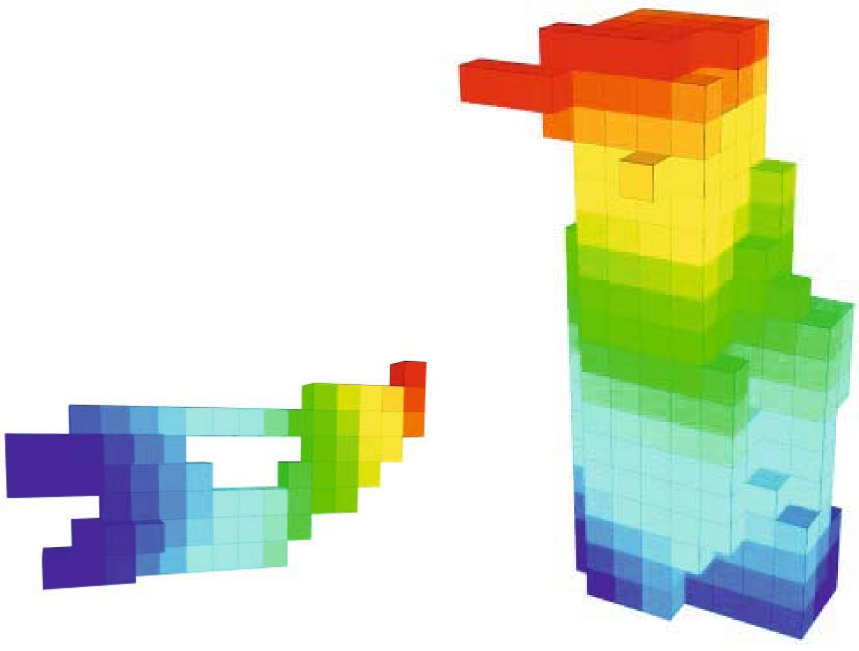

(h)

the total strain energy of the structure under the specified loading conditions. The structures strain energy was calculated using a custom finite element analysis (FEA) solver employing the direct stiffness method.

Using a frequency-based representation, we were able to evolve structures in our 2800-cube workspace using an encoding of only 120 parameters. We used a population of 10 individuals which were selected for crossover using deterministic crowding. Children were produced by selecting each frequency component randomly from one of the two parents. Children were also mutated at a rate of 0.2, with 20% of the frequency components mutated by up to 10% with each mutation. If a generated child was more fit than its most similar parent, the child replaced the parent in the population, otherwise it is discarded. More details on the encoding, evolution parameters, and fitness evaluation can be found in [14].

The results of the structure evolution can be seen in table 14.2. A representative sample of the best individuals from a number of different generations is shown with colors indicating the displacement of each module under the prescribed loading conditions (dark red and dark blue indicate maximum and minimum displacement, respectively). A plot of the best and average fitnesses vs. the total number of

Table 14.3 Evolved applications for stochastic modular robots. (a) Custom wrench evolved to withstand the applied forces necessary to manipulate an irregular mechanical component. (b) Custom prosthetic leg replacement evolved to fit prescribed kinematic and force constraints.


(a) (b)

evaluations shows the convergence of the GA on a successful structure (table 14.2 (g)). The deformation of the most fit individual is also shown in table 14.2 (h).

Even if we restrict ourselves to FEA-solvable loading conditions, many other goal tasks for a stochastic modular robotic system are possible. For example, a shape could be defined that corresponds to a non-standard bolt or other mechanical component that needs to be removed (table 14.3 (a)). A custom wrench shape could then be evolved to withstand loads required to remove the component. Alternatively, the kinematic and loading conditions could be defined for a damaged robot or human limb that needs to be replaced and an optimal replacement shape could be then be evolved to act as a prosthetic (table 14.3 (b)).

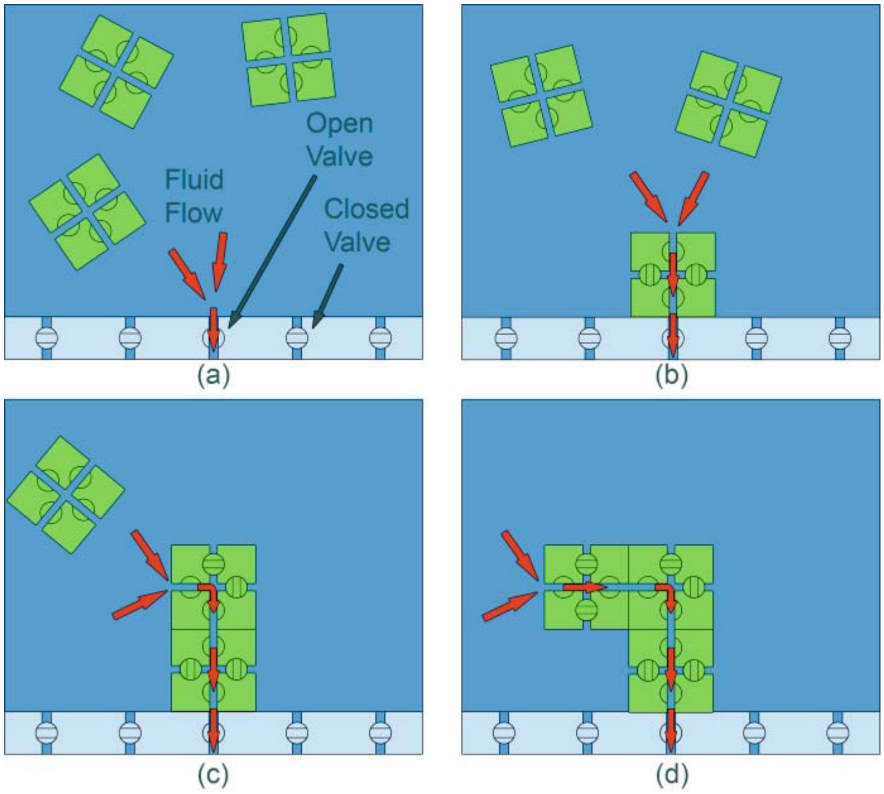
14.3 Stochastic Fluidic Assembly System Model

The structure evolved as described in Section 14.2 is to be assembled in a fluidic modular robotic system (table 14.1). In this system, the target structure is grown by adding components first to a planar substrate, then to the exposed surfaces of previously-assembled components (table 14.4). The components are homogenous, andtheir availability for assembly at any particular location at any point in time is determined by the stochastic motion of the fluid in the assembly chamber. An attraction can be activated by redirecting fluid flow through the internal structure to open a sink at any surface of the structure, which pulls in nearby components until one comes close enough to attach. However, there is a limit to the number of sinks that can be opened simultaneously (due to the increased flow rates required to maintain additional sinks). In addition, the attraction probability is affected by the local and/or global assembled structure (which affects the fluid flow in the chamber), and the number of sinks attracting to a given location. Kinematic constraints such as the inability to insert a module directly between two opposing modules also limit the viable assembly paths.

In our model of this system we assume limit of four attractions (sinks) that can be active at one time, with only one attraction possible per grid location. We also assume that the expected cube arrival time is a function of local geometry only. Taking the kinematic constraints into account, there are three possible configurations to which a cube can be attracted (table 14.5 (a)). By running simulations of the stochastic assembly system with a volume full of randomly initialized cubes being attracted to one of these three configurations, we are able to obtain a distribution of the assembly time in each case (table 14.5 (b)). Table 14.5 (c) shows the distributions of the assembly times in the three separate cases overlaid on the Poisson distributions for the same means, which are assumed to be the correct distributions in sampling assembly times during assembly simulations, as described below.

When multiple sinks are opened simultaneously with different expected cube arrival times (based on the current configuration), we require a method of determining which cube arrives first. This is necessary both as a way of simulating natures turn

Table 14.4 Stochastic Fluidic Assembly Concept [10]. (a) Fluid flow (indicated by arrows) into a substrate attracts a nearby module moving stochastically in fluid environment. (b) Once attached, the module draws power from the substrate to activate on-board valves and redirect fluid flow through internal channels, (c) attracting new modules at desired locations. (d) This process continues layer-by-layer until the structure is complete.

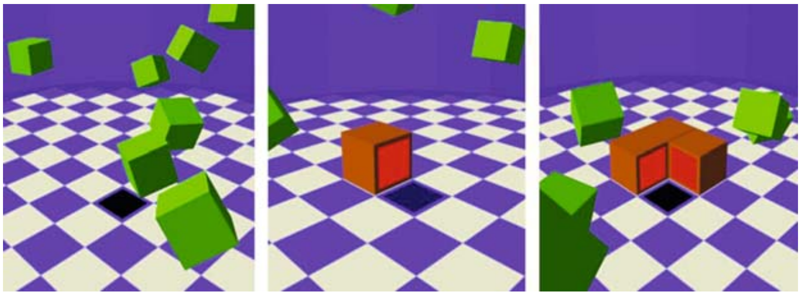


to decide which of the possible cubes arrives at a given step, and also as a method of selecting cubes in the assembly algorithm.

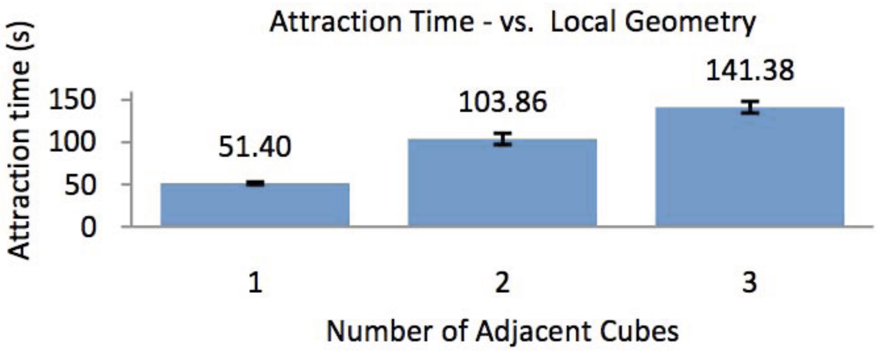
Drawing inspiration from chemistry, we can sample from the set of potential assembly events using a method developed to sample from the set of simultaneous reactions in a well-mixed stochastic fluidic environment with multiple chemical species [17],[18]. Gillespies method relies on the following probability density that the next reaction is μ and it occurs at time τ:

$$P(\mu,\tau)d\tau = a_{\mu}exp(-\tau\sum_{j}a_{j})d\tau \tag{14.1}$$

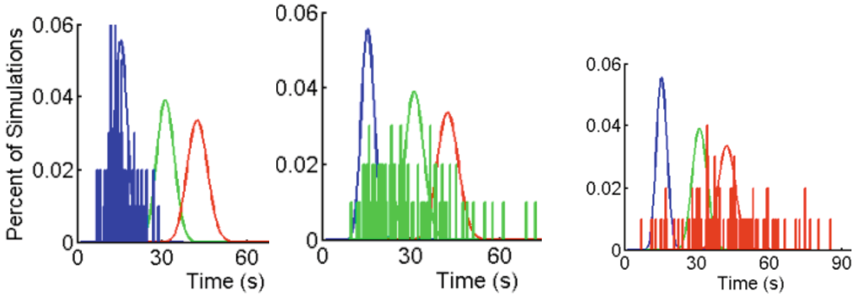
Table 14.5 Sink Configurations. (a) The three possible sink configurations to which a new cube can be attached. Green cubes represent free components floating stochastically in a fluidic environment while assembled cubes are colored red. Dark blue squares represent sinks that attract nearby cubes. (b) Plot of mean time to attraction over 100 simulation of the three configurations. (c) Plots of the distribution of assembly times grouped into one second segments overlaid on a plot of the Poisson distributions for the three mean assembly times.



(a)



(b)



(c)

where the a_μs are the respective reaction rate coefficients (that can depend on chemical concentrations, environmental conditions, etc.). Integrating $P(\mu,\tau)$ from $\tau = 0$ to inf gives the probability that the next reaction to occur is a given reaction μ:

$$P(\mu) = \frac{a_\mu}{\sum_j a_j} \tag{14.2}$$

Further, summing $P(\mu,\tau)$ over all reactions μ gives the distribution of times for the next reaction to occur:

$$P(\tau)d\tau = (\sum_j a_j)exp(-\tau \sum_j a_j)d\tau \tag{14.3}$$

Integrating this equation and solving for the time τ as a function of a selected random number P gives an equation for randomly sampling the time taken for the next equation to occur:

$$\tau = -\frac{ln(1-P)}{\sum_j a_j} \tag{14.4}$$

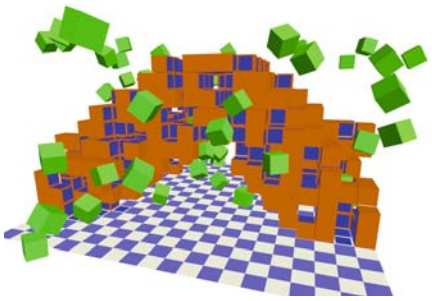
If we view the various parallel potential interactions that could occur at any point in time as chemical reactions in a well-mixed system, we can apply equations (1.2) and (1.4) to determine the time and location of the next cube arrival in our stochastic assembly system. In this case, the rate constants can be determined by inverting the mean times to assembly from table 14.5 (b).

14.4 Assembly Algorithm

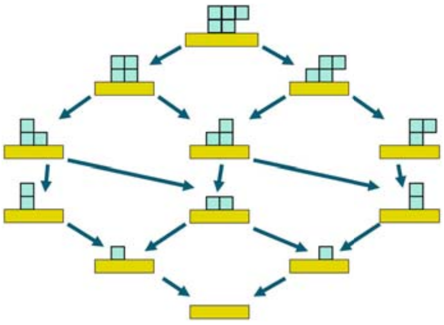
The main challenge of the assembly algorithm is to assemble error-free structures in a parallel way. Straightforward approaches, such as attracting components in a greedy manner wherever they are needed, result in porous structures with many unfillable holes ([10], table 14.6 (a)). Serial assembly approaches (e.g. filling one layer at a time) result in perfect structures but take a long time and are susceptible to single location failures. Otherwise, it is difficult to enumerate robust parallel assembly rules that result in a perfect target structure.

The approach described here is to look at the problem from the opposite direction: begin with the final structure and remove one accessible cube at a time, keeping track of the sequence. This approach efficiently generates a serial assembly sequence that results in a perfect assembly, starting with any partially-complete structure (assuming such a path exists). In order to generate a parallel assembly sequence, we would ideally want a graph of all of the possible routes to an error-free target structure (table 14.6 (b)). In practice, this graph is prohibitively expensive to compute for all but the simplest structures. However, using the serial disassembly approach, our strategy was to sample enough of the paths through this graph to determine which of the potential attraction locations were both valid (did not result in errors) and popular (many paths passed through this node). The idea was to avoid

Table 14.6 Assembly by disassembly. (a) Greedy assembly algorithms result in porous structures with unfillable holes (indicated by blue squares). (b) By starting the complete structure and removing one cube at a time, one could potentially create a graph of all possible assembly sequences that result in an error-free structure.



(a)



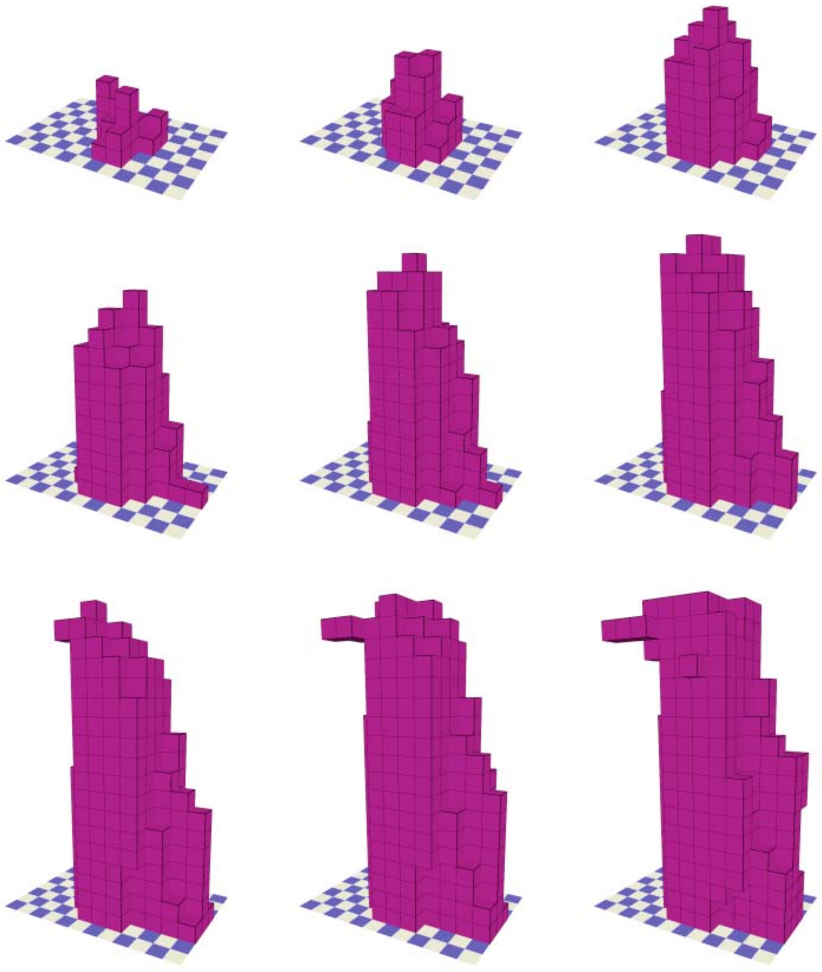
(b)

getting stuck in a part of the graph with a small number of paths to completion since this essentially degrades to serial assembly.

An example of applying this approach to the assembly of an evolved target structure can be seen in table 14.7. We evaluated multiple potential selection methods for choosing the next cube to be removed at each step of each disassembly sample (table 14.8 (a)). Most of these relied on obtaining a complete list of all of the accessible cubes at each step, and choosing among them to decide which cube to remove. This selection is made either randomly, favoring the attractions that take the longest (i.e. with three adjacent cubes), or shortest (one adjacent cube) first, greedily selecting the fastest or slowest attractions first, or using Gillespie sampling (Section 14.2) to choose from among the possible cubes. A final selection method was to randomly check cubes for availability and choose the first accessible cube found.

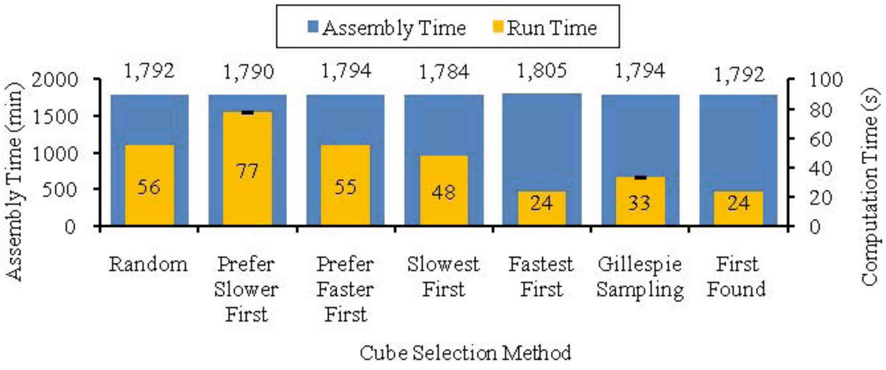
Surprisingly, all of these selection methods found serial assembly sequences with very similar expected assembly times. The best and worst selection methods were the greedy approaches, however neither of these are viable for sampling the overall assembly graph since they will always pick the same path through the graph.

Table 14.7 Assembly Sequence. Sequence of images taken from simulated assembly of robotic prosthetic from table 14.3 (b) using the parallel assembly algorithm.

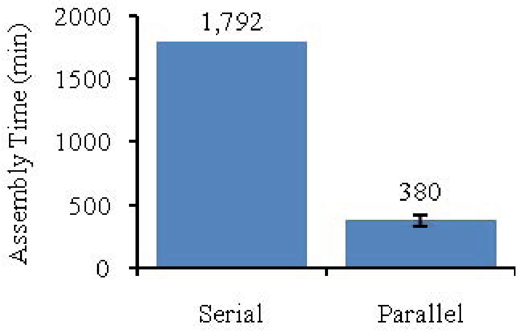


Discarding these options, simply choosing the first randomly found accessible cube turned out to be a much less computationally-intensive option since it avoided the necessity of constructing a list of all of the accessible cubes at every step of the disassembly. However, it is interesting to look at the different routes these approaches take to assembly (table 14.8 (c)-(e)). Because single neighbor attraction sites are the fastest, the Fastest First approach first assembles a skeletal structure while Slowest First assembles a solid core of cubes and builds outward. The more

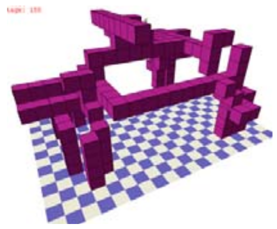
Table 14.8 Effect of cube selection method. (a) Total expected serial assembly time (blue), and computation time (red) averaged over 100 runs, for various selection methods. Error bars indicate standard error. (b) Comparison of average expected assembly time for serial and parallel assembly using the First Found selection method. (c)-(e) Partial structures assembled for Greedy - Slowest First, Greedy - Fastest First, and Gillespie Sampling selection methods (respectively). (f) Completely assembled target structure.



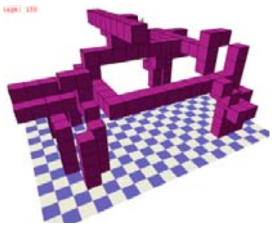
(a)



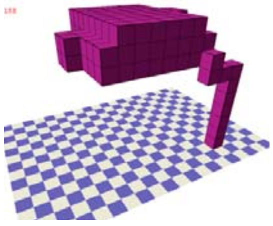
(b)



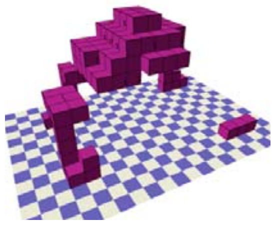
(c)



(d)



(e)



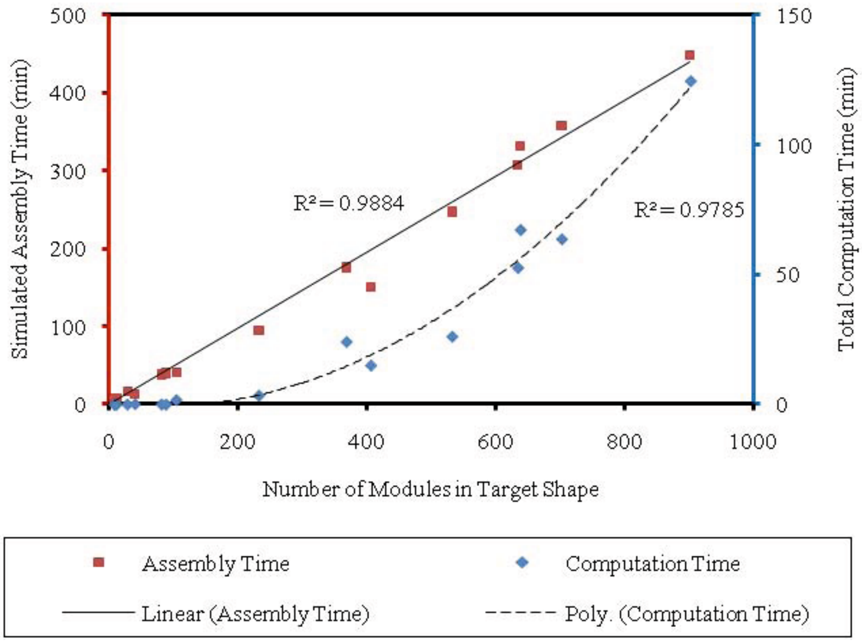
(f)

random approaches such as Gillespies method seem to be a hybrid of these two extremes.

We ran the parallel assembly algorithm by taking 20 samples of the overall assembly graph at each stage, and selecting the four most commonly encountered next locations to attract cubes in parallel. Based on the results shown in table 14.8 (a), we used the first randomly found accessible cube as the selection method for each disassembly step of each sample. Using the Gillespie method, we simulated which of the four locations attracted a cube at each assembly stage. The total assembly time for the structure was reduced from 1792 to 380 minutes (table 14.8 (b)), while still maintaining the guarantee of error-free assembly.

Finally, we studied the scaling of the simulated assembly times and the total assembly algorithm computation times as a function of the number of modules in the target structure (table 14.9). Interestingly, while the predicted structure assembly time scales linearly with the number of modules, the total time required to compute which locations to attract modules to at each stage of assembly increases with the square of the number of modules. Thus, after a certain structure size the assembly rate would be limited by algorithm computation as opposed to physical assembly. However, since the assembly algorithm is re-sampling the same

Table 14.9 Assembly algorithm scaling. The time taken to assemble a structure in simulation scales linearly with the number of modules while the total time taken to compute the locations to next attract modules at each stage scales with the square of the number of modules.



overall assembly graph over and over again, we believe it would be possible to employ a dynamic programming technique to reduce the algorithmic complexity by re-using previous samples.

14.5 Conclusion

We have described an automated approach to stochastic modular robotics that, given a target function, evolves a structure to achieve this function, and then directs the structures assembly. Finite Element Analysis was used to determine the fitness of the evolved structures and a frequency domain representation of the structures simplifies the number of parameters and enhances evolvability. Once a suitable structure was found, its assembly was planned using a sampling approach to map out a path through the graph of all possible assembly sequences that minimizes assembly time. Each sample of this graph was obtained by starting with the target structure and peeling away one accessible module at a time until the initial structure is revealed. This simple algorithm guarantees a valid path to assembly of the target structure that does not leave behind unfillable holes.

While our approach follows the traditional paradigm of first designing, then determining how to assemble a structure, the automated nature of the entire process allows for the possibility of feedback between the two parts. Not only could assembly simulations factor in to the evolutions fitness function, but the design itself could in fact be tweaked through evolution in response to current assembly conditions.

Acknowledgements

This work was funded by DARPA grant number W911NF-08-1-0140 (Programmable Matter Program), and by NSF Creative-IT grant 0757478. M. Tolley would also like to thank the Canadian NSERC for their support through the PGS Program.

References

1. Lipson, H., Pollack, J.B.: Automatic design and Manufacture of Robotic Lifeforms. Nature 406, 974–978 (2000)
2. Hornby, G.: Functional scalability through generative representations: the evolution of table designs. Environment and Planning B-Planning and Design 31, 569–587 (2004)
3. Malone, E., Lipson, H.: Freeform Fabrication of Ionomeric Polymer-Metal Composite Actuators. Rapid Prototyping Journal 12, 244–253 (2006)
4. Piqu, A., Arnold, C.B., Kim, H., Ollinger, M., Sutto, T.E.: Rapid prototyping of micropower sources by laser direct-write. Applied Physics A: Materials Science and Processing 79, 783–786 (2004)
5. Malone, E., Berry, M., Lipson, H.: Freeform Fabrication and Characterization of Zinc-Air Batteries. Rapid Prototyping Journal 14, 128–140 (2008)

6. Yim, M., Shen, W.-M., Salemi, B., Rus, D., Moll, M., Lipson, H., Klavins, E., Chirickjian, G.S.: Modular Self-reconfigurable robotic systems. IEEE Robotics and Automation Magazine 14, 43–52 (2007)
7. Klavins, E.: Programmable Self-Assembly. IEEE Control Systems Magazine 27, 43–56 (2007)
8. Goldstein, S.C., Campbell, J.D., Mowry, T.C.: Programmable matter. IEEE Computer 28, 99–101 (2005)
9. Gilpin, K., Kotay, K., Rus, D., Vasilescu, I.: Miche: Modular Shape Formation by Self-Disassembly. Int. J. Robotics Research 27, 345–372 (2008)
10. Tolley, M.T., Kalontarov, M., Neubert, J., Erickson, D., Lipson, H.: Stochastic Modular Robotic Systems: A Study of Fluidic Assembly Strategies. IEEE T. Robotics (in press)
11. White, P.J., Kopanski, K., Lipson, H.: Stochastic Self-Reconfigurable Cellular Robotics. In: Proc. Robotics and Automation (ICRA 2004), pp. 2888–2893 (2004)
12. White, P.J., Zykov, V., Bongard, J., Lipson, H.: Three Dimensional Stochastic Reconfiguration of Modular Robots. In: Proc. Robotics Science and Systems (2005)
13. Tolley, M.T., Krishnan, M., Lipson, H., Erickson, D.: Advances Towards Programmable Matter. In: Proc. Miniaturized Systems for Chemistry and Life Sciences, MicroTAS (2008)
14. Hiller, J., Lipson, H.: Multi Material Topological Optimization of Structures and Mechanism. In: Proc. Genetic and Evolutionary Computation Conference (2009)
15. Zykov, V., Lipson, H.: Experiment Design for Stochastic Three-Dimensional Reconfiguration of Modular Robots. In: Proc. Int. Conf. Intelligent Robots and Systems, Self-Reconfigurable Robotics Workshop (2007)
16. Tolley, M.T., Krishnan, M., Erickson, D., Lipson, H.: Dynamically Programmable Fluidic Assembly. Appl. Phys. Lett. 93, 254105 (2008)
17. Gillespie, D.T.: Exact stochastic simulation of coupled chemical reactions. Physical Chemistry 81, 2340–2361 (1977)
18. Gibson, M.A., Bruck, J.: Efficient Exact Stochastic Simulation of Chemical Systems with Many Species and Many Channels. Physical Chemistry A 104, 1876–1889 (2000)