A Simulation Based Architecture for the Development of an Autonomous All Terrain Vehicle

Gianluca Bardaro, Davide Antonio Cucci, Luca Bascetta, and Matteo Matteucci

Dipartimento di Elettronica, Informazione e Bioingegneria, Politecnico di Milano, Piazza Leonardo da Vinci 32, 20133 Milano, Italy

Abstract. In this work we describe a simulation environment for an autonomous all-terrain mobile robot. To allow for extensive test and verification of the high-level perception, planning, and trajectory control modules, the low-level control systems, the sensors, and the vehicle dynamics have been modeled and simulated by means of the V-Rep 3D simulator. We discuss the overall, i.e., high and low-level, software architecture and we present some validation experiments in which the behavior of the real system is compared with the corresponding simulations.

1 Introduction

In this work we present an Autonomous All-Terrain Robot developed starting from a commercial, fuel-powered, All-Terrain Vehicle (ATV), i.e., a Yamaha Grizzly 700. This robot is characterized by an Ackermann steering kinematic and the original vehicle commands have been replaced by servomechanisms controlling the handlebar position, the throttle, and the brake. Multiple sensors have been fitted on the robot to perform perception activities: two laser range-finders, a stereo camera rig, a GPS, an Inertial Measurement Unit (IMU), as well as, wheel and handlebar encoders.

Given the physical dimensions of the robot, the typical operating environment, and the complexity of the system architecture, it has been quite challenging to develop and test all the software components, especially in early stages of development. The main difficulties come from the intrinsic complexity in operating the robot, the little repeatability of experiments, the time consuming activity of fault detection and isolation. Moreover, meteorological and space issues further affected field evaluation, either because a suitable test area was not always available for experiments, and because of safety issues for the vehicle itself, which have a high roll-over risk, and for the people working with it.

To address these challenges we developed a simulation environment in which the vehicle and its sensors are substituted by a simulator in a way that is transparent with respect to the high-level perception and control software architecture. In contrast with respect to classical hardware-in-the-loop techniques, in which key elements of the real system, which might be difficult to model, replace their simulated counterpart, here the real system and the environment are replaced with models without changes in the robot control software.

D. Brugali et al. (Eds.): SIMPAR 2014, LNAI 8810, pp. 74–85, 2014.

In our work we employed the Virtual Robot Experimentation Platform (V-REP) [7], a physical simulator which relies on a distributed and modular approach and allows to model complex scenarios in which multiple sensors and actuators operate asynchronously at various rates. Other simulator were available, such as Gazebo [10], which is mainly focused on robotic applications, and Dymola [4], which instead focuses on highly accurate multi-domain, multi-body, physical, simulations. We decided to use V-REP instead of Dymola because of its capability in simulating the vehicle sensors and we preferred V-REP to Gazebo for its ease of use. The high-level perception and control architecture of the robot is implemented relying on the Robot Operating System [13], an open source framework which has recently become popular in the literature for its turn-on-and-go functionalities, easiness of deployment, large community, and support.

The use of simulators is common in robotics and several works related to the use of a simulator in the development of an autonomous all terrain robot, and autonomous robots in general, have been presented in the literature: in [9] a high fidelity model, including sensors, is developed to study the behavior of an autonomous ATV, focusing on the simulation itself, rather than the integration with the robot architecture. A simulator is also used in [8] in the actual robot architecture for real-time path planning, where the aim is to foresee potential collisions and change the plan accordingly. In [12], the SimRobot simulator is introduced and example applications in the RoboCup competition are discussed.

This work is organized as follows: in Section 2 the overall robot architecture is briefly discussed. Next we move to the high-level perception and control modules implemented employing the ROS framework. In Section 3 we present the simulation environment and we discuss sensor and vehicle models. Finally, in Section 4 we validate our approach comparing the behavior of the simulated system with the real one during autonomous trajectory following experiments.

2 The Quadrivio ATV

In this section we briefly review the vehicle specifications and the developed hardware and software architectures. The original vehicle used is a Yamaha Grizzly 700 (see Figure 1a), a commercial fuel-powered utility ATV, specifically designed for agriculture work. For the purposes of the project, the vehicle cover has been removed and substituted with an aluminum one; this new cover allows to easily accommodate for control hardware and sensors. Furthermore, the vehicle has been equipped with three low-level servomechanisms, each one with its own control loop, to automatically regulate the steer position, the throttle aperture and the braking force [2][3]. Figure 1b shows the vehicle after customization. The main characteristics of the vehicle are listed in Table 1.

2.1 The Hardware Architecture

In order to allow for teleoperation and autonomous navigation, an on-board hardware/software control architecture has been developed.

(a) Original Yamaha Grizzly 700 (b) Quadrivio ATV

Fig. 1. On the left the original all-terrain vehicle, on the right the vehicle after the changes to make it autonomous

The architecture is divided in two different layers: the higher level is developed using ROS and is responsible for acquiring data from external sensors, such as GPS, magnetometer, Inertial Measurement Unit (IMU), cameras and laser range-finders. Moreover, it hosts the modules for localization, path planning, high-level trajectory control and autonomous driving. The lower level acts as an interface between the vehicle servomechanisms and the ROS architecture: it receives desired setpoints from the higher level, reads the handlebar angle, throttle ratio, vehicle speed measurements and runs the low-level control loops.

To implement such an architecture that includes high-level and low-level tasks, a multi-layered and multiprocessor hardware/software architecture is required, which consists of: an industrial PLC, which allows a good compromise between the hard real-time requirements and high-level programming, and a standard i5 PC, on which runs the high-level ROS architecture (perception, localization,

Table 1. Vehicle characteristics

Main characteristics of the vehicle	
Engine type	686cc, 4-stroke, liquid-cooled, 4 valves
Drive train	2WD, 4WD, locked 4WD
Transmission	V-belt with all-wheel engine braking
Brakes	dual hydraulic disc (both f/r)
Suspensions	independent double wishbone (both f/r)
Steering System	Ackermann
Dimensions (LxWxH)	2.065 x 1.180 x 1.240 m
Weight	296 Kg (empty tank)

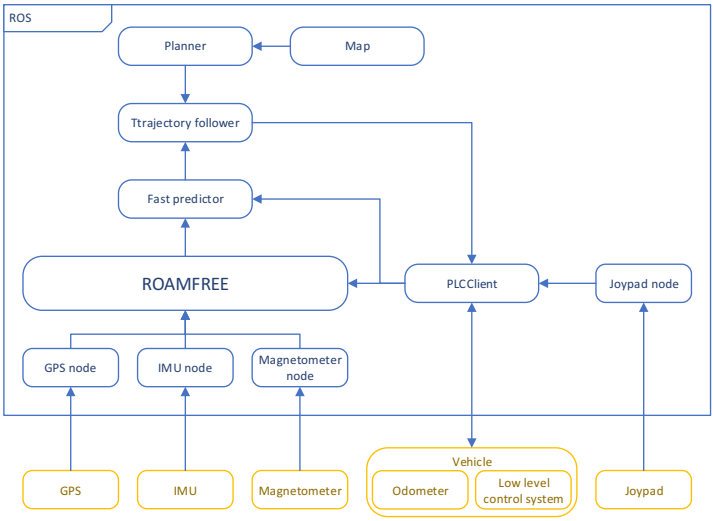


Fig. 2. The real system architecture

obstacle avoidance, medium-long range navigation, planning, etc.). Communication between the two layers is obtained through an Ethernet link.

2.2 The Software Architecture

Figure 2 shows the main modules of the high-level software architecture and their relations with the external sensors and the vehicle servomechanisms. These modules live as independent applications running on the standard PC and the communication between them is guaranteed by the ROS middleware.

The core part of the perception architecture consists in the localization node, which is based on ROAMFREE [5]. This open source framework provides out-of-the-box 6-DOF pose tracking fusing the information coming from an arbitrary number of information sources such as wheel encoders, inertial measurement units, and so on[1]. In ROAMFREE, high-level measurement models are used to handle raw sensor readings and provide calibration parameters to account for distortions, biases, misalignments between sensors and the main robot reference frame. The information fusion problem is formulated as a fixed-lag smoother and it runs in real time thanks to efficient implementations of the inference algorithms (for further details see [6], and [11]). At the present stage of development the localization module estimates the robot poses exploiting vehicle kinematic data (i.e., the handlebar position and the rear wheel speed), GPS, magnetometer, and the gyroscopes in the inertial measurement unit.

[1] `http://roamfree.dei.polimi.it`

The pose estimate is generated by the localization node at a frequency of 20 Hz. However, due to the latencies introduced by the ROS network, delays in the trajectory control loop which affect the system stability can occasionally arise. In order to prevent the detrimental effects of these delays, we introduced a predictor node. This node computes a prediction of the future robot pose at a frequency of 50 Hz; this prediction is based on the latest available global pose estimate and on the integration of the Ackermann kinematic model with the kinematic readings from the vehicle.

Given a map and a goal, a planner node, based on the SBPL library [1], produces a global path, which is then fed to a lower level trajectory following module. This module computes setpoints for the vehicle speed and handlebar angle, based on the current pose and velocity estimates, and on the planned trajectory. These setpoints are sent to the low-level regulators by a ROS node communicating with the PLC, which additionally acts as a multiplexer between the autonomous drive and the manual setpoints, depending on the current operating mode.

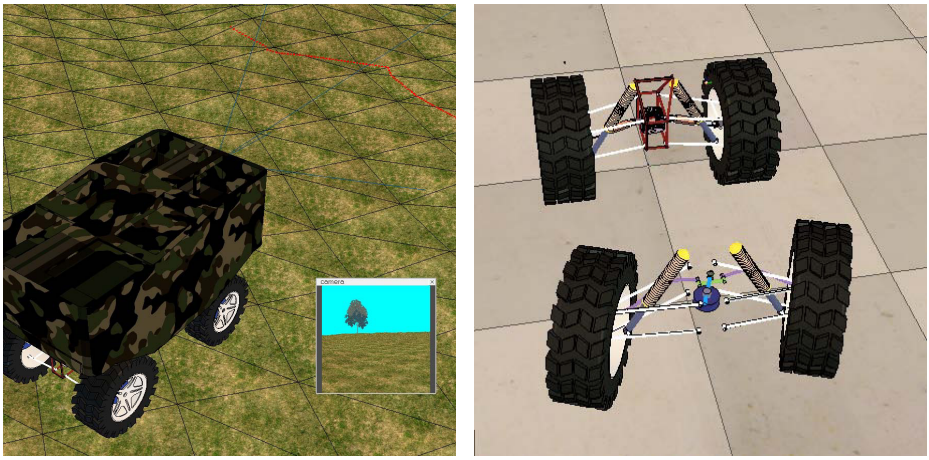
3 The Simulation Environment

The software architecture in Figure 2 has been designed introducing a decoupling layer between the real robot and the high-level perception and control software. This layer is composed by ROS nodes that respectively handle the GPS and the IMU sensors, and the communication module between the standard PC and the PLC. In this section we describe how we have replaced the real vehicle with a physical simulator and how we set up vehicle and sensor models so that they accurately mimic the real robot.

3.1 The Simulator

The vehicle simulator was developed using V-Rep [14], a software for robot modelling offering an accurate physics simulation. This software has been chosen for some of its features that fit particularly well with the requirements of our application. First of all, it is simple to set up and use with its integrated development environment, it has a library with various examples of robots already modeled, and one of them is particularly similar to our vehicle in terms of kinematics and suspension geometry. Another important feature is the possibility to control every object in the simulation with a remote API, that allows the integration with ROS, making it suitable for interacting with our software architecture.

One important issue which has to be addressed in coupling a simulator with a control architecture is to make sure that they share a global time reference. In our case, this was obtained enabling the `use_sim_time` parameter in ROS and having V-Rep publishing the current simulation time on the `clock` ROS topic. This is particularly useful when challenging simulations are run which involve complex terrain or environments and cannot be carried out in real-time.



(a) The customized model (b) Suspensions and Ackermann steering

Fig. 3. The vehicle model used in simulation

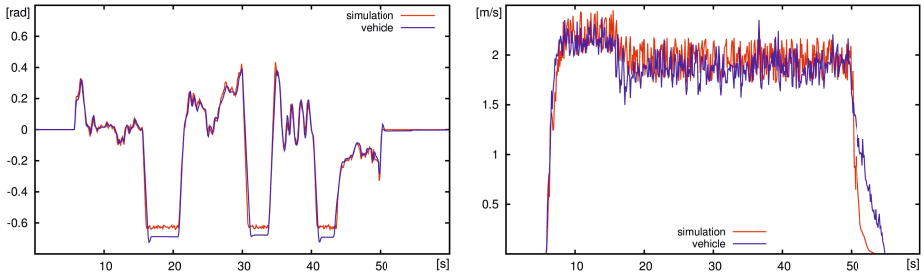
3.2 The Vehicle Model

V-Rep offers multiple built-in vehicle models. We chose one that shares the Ackermann steering and the suspension geometry with our vehicle and we customized it to match the Quadrivio ATV specifications. The vehicle characteristics required to set up the model are listed in Table 2, while Figure 3a shows the customized model in which it is possible to see the image from the camera and the trace of the laser range-finder on the terrain. Figure 3b shows details of the Ackermann steering and suspensions.

The next step in vehicle simulation was to ensure the real vehicle and the simulation model share the same dynamic and kinematic behavior. In particular, we required that the step response for the handlebar and the speed loops on the real vehicle and in simulation were similar. As we are not interested in reproducing the engine behavior or in studying the dynamics of the steering motion control system, but only in simulating the overall vehicle dynamics, the simulator does not include an accurate model of the steering column and of the engine. Instead, the steer and speed loops, both based on a PID controller, were

Table 2. Model parameters

Vehicle model specifications.	
Track	1920 mm
Wheelbase	1250 mm
Front wheel (WxH)	201 x 635 mm
Rear wheel (WxH)	247 x 635 mm
Weight (estimated)	390 kg



(a) Handlebar position response (b) Vehicle linear speed response

Fig. 4. Plot of the vehicle actuators step responses on the real vehicle and on the the simulated one

directly modelled in such a way that the simulated responses of these controlled systems were as close as possible to the experimental ones.

We recorded data for the handlebar step response of the real vehicle and then we tuned the PID regulators that control the handlebar column in the V-Rep simulator so that the simulated vehicle handlebar position step response matches the one of the real vehicle. In particular, setpoints for the handlebar position and vehicle speed were recorded while being sent to the real vehicle, then we feed the simulated vehicle with the same setpoints, allowing to tune the simulator response. Figure 4a shows a comparison between the handlebar response for the real vehicle (blue line) and the simulated one (red line). It is possible to see that the two behaviors substantially match, but the simulated steer cannot reach a value as high as the real one, because of geometric limitations in the original model. However, we obtained a reasonable behavior in common operation ranges.

For the speed step response we implemented a custom PID controller, which controls the torque applied to the motor joint minimizing the error over the target speed. It was not possible to use the one integrated in the simulator because the model uses a motorized joint, which models an electric motor, while the vehicle has a fuel-powered engine with a significantly different characteristic. After the tuning with field data of the motor PID, we obtained a good matching behavior in the acceleration phases, while there is still a slight difference in deceleration due to the difficulties in modeling the engine braking when the throttle setpoint is suddenly decreased (see Figure 4b).

3.3 The Sensor Models

After modeling the vehicle we added the sensors: GPS, IMU, magnetometer and odometer. Most of them were already available in V-Rep, but they lacked the ROS integration and they did not account for noise and fault situations.

The sensors are realized with a “two layers” approach; the first layer is implemented directly inside the simulator, it consists in the sensors itself and a script that prepares and publishes ROS messages. The second layer is outside

the simulator and it consists in a ROS node that reads the messages published by the simulator and converts them into a format which matches the one produced by sensors on the real vehicle. This double conversion has the aim of obtaining the decoupling between the simulator and the high-level perception and control architecture; indeed, from the point of view of the high-level architecture, there is no difference between the real sensors and the simulated ones.

The following are the sensor we have simulated with a brief description:

- GPS: V-Rep provides already a simulated GPS providing x/y/z-coordinates which are compatible with the East-North-Up (ENU) reference frame used in our architecture, so no further conversion was needed. The node coupled with this sensor builds the correct ROS message introducing some Gaussian noise (derived from real data) and allows to model random downtimes, to simulate for real world GPS unavailability;
- IMU: models for accelerometers and gyroscopes are provided by the simulator out of the box. The raw readings are published as ROS messages and converted in the desired format;
- Magnetometer: to implement this sensor we extract the current vehicle model orientation with respect to the global fixed reference frame; from it we can compute a simulated value for the Earth magnetic field reading in the sensor reference frame. However, on the real vehicle, hard and soft iron distortion affect the magnetometer readings. These are considered by employing the sensor model presented in [15], whose parameters have been calibrated by means of the sensor self-calibration capabilities of the ROAMFREE sensor fusion framework;
- Odometer: the vehicle rear wheel speed and the current handlebar position are extracted from the current status of the relevant joints in V-Rep, and a TCP socket is employed to communicate with the PLCClient ROS node, in a way that mimics the behavior of the X20 industrial PLC.

Moreover, there are two sensors that are simulated but not currently used for localization:

- Laser: the simulator provides a laser scanner out of the box. The associated script required some adjustment to publish the correct ROS message;
- Camera: V-Rep offers a highly customizable vision sensor that we used to realize a camera that matched our needs.

Figure 5 shows how the overall architecture changes when the simulator substitutes the vehicle and the real sensors. Every sensor now is substituted by its simulated counterpart, yet nothing changes from the point of view of the perception and control modules, since the communication is done on the same ROS topics. The PLCClient, in charge of communicating with the low-level control of the vehicle, interacts with a simulated low-level control system through a local socket connection. Inside the simulator a script converts setpoints from the PLCClient into setpoints of the model joints, another one collects odometry and sends it back to the ROS node. Migration from the simulated environment and the real vehicle is simple and requires only the change of few parameters.

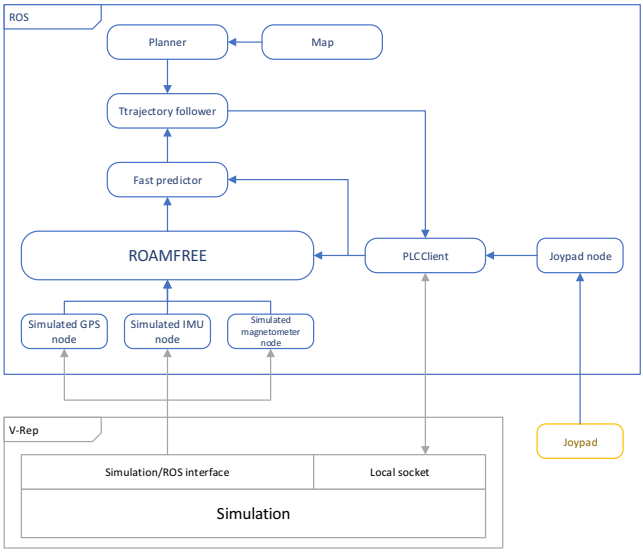


Fig. 5. Overall architecture in the simulation setup

4 Experimental Evaluation

In this section we discuss some autonomous trajectory following experiments done on the real vehicle and in the simulation environment. We employ an eight-shaped trajectory originating 1 meter ahead with respect to the current pose of the robot. The two circles have a diameter of respectively 18 and 12.5 meters.

Figure 6 shows the results of six experiments done with the real vehicle, in it we have plotted the reference path, the robot position, estimated by ROAMFREE, and the raw GPS readings. Especially in the first experiment (Figure 6a), but also in the other ones, it is possible to see how the trajectory is followed with reasonable accuracy, and with ROAMFREE being able to account for substantial multi-path effect compromising GPS readings.

Figure 7, instead, shows the results of the same experiments done with the simulator. In this case the GPS, like all the other sensors, is simulated and it is possible to appreciate its simulated faulty behavior. The robot position is estimated using measurements given by the simulated sensors as they were real, no special configuration is necessary to use them.

In Figure 7a, and 7b, it is possible to see that multipath effect compromises the real GPS sensor readings. This happens when the receiver tracks a replica of the GPS signal which is reflected by environmental features such as buildings and trees. This effect is hard to model and it has not been simulated in V-Rep, even though, if we restrict to its effect on localization, a noise model which accounts for a random transformation to be applied occasionally on the GPS readings could be considered.

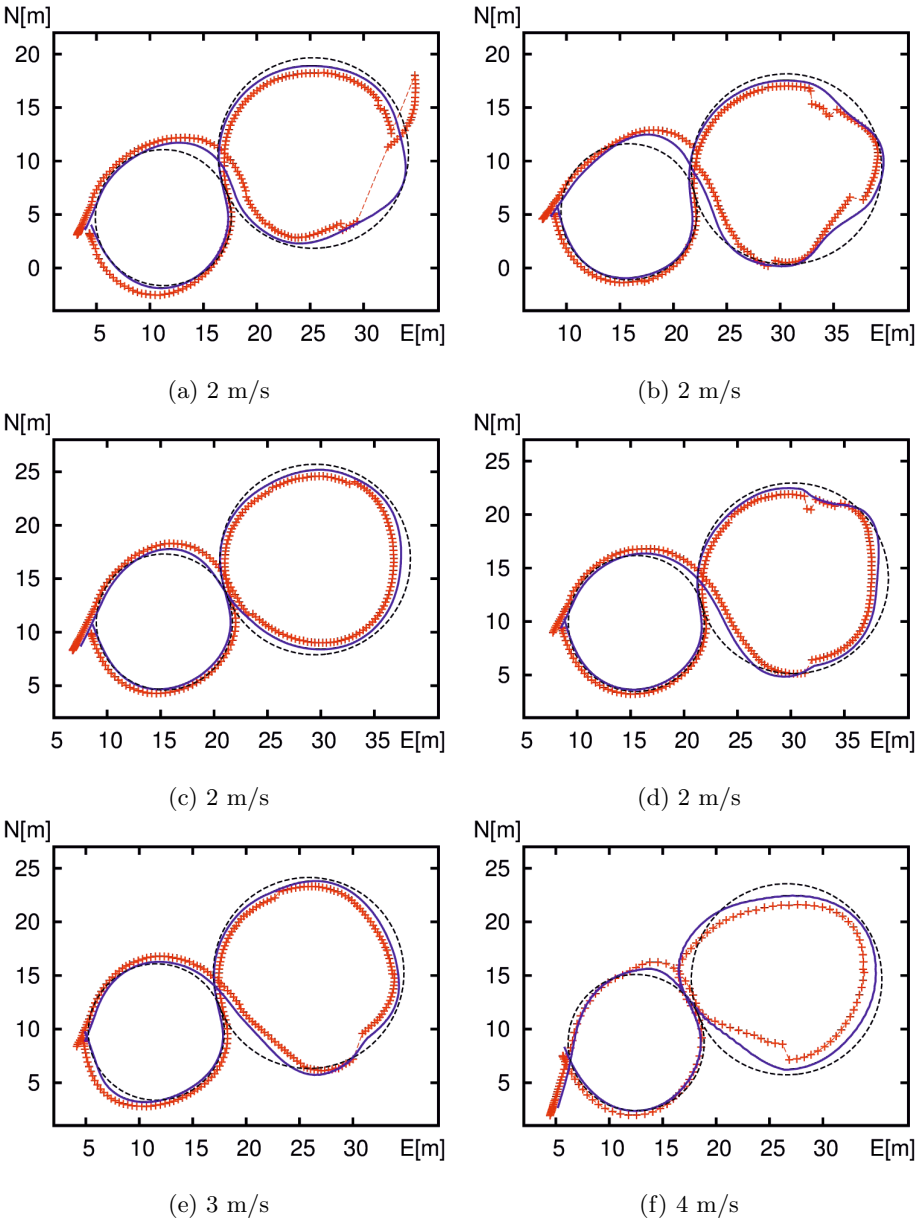
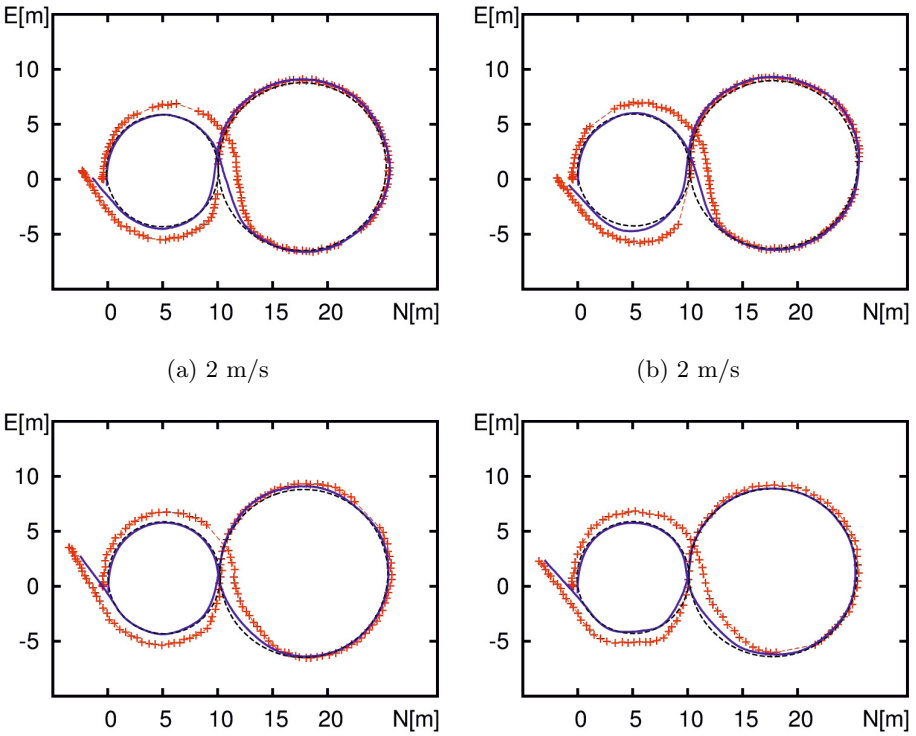


Fig. 6. Online trajectory following results. Reference path for the trajectory follower (black dashed line), the ROAMFREE position output (blue line), and the GPS readings (red crosses).



(a) 2 m/s (b) 2 m/s

(c) 3 m/s (d) 4 m/s

Fig. 7. Simulation trajectory following results. Reference path for the trajectory follower (black dashed line), the ROAMFREE position output (blue line), and the GPS readings (red crosses).

5 Conclusions

In this work we have presented and validated a simulated environment which provides an alternative when experiments on the real robots cannot be afforded, and ultimately simplifies the development and the testing of complex robotic architectures. As described in Section 3, the simulator transparently substitutes the real vehicle and its sensors. This is possible thanks to the highly modular ROS architecture and to the native integration of V-Rep with it. The simulator does not account for latency of sensors, either internal or caused by the communication, and this makes a simulation more ideal than we would like it to be, therefore a possible improvement to the current work could be addition of latency to sensors. The next step is to exploit the features of V-Rep to test the robot on rough terrains, since the simulator permits to add complex terrains that can be difficult to find in the real world, or that are too risky for the vehicle.

References

1. http://wiki.ros.org/sbpl
2. Bascetta, L., Magnani, G.A., Rocco, P., Zanchettin, A.M.: Design and implementation of the low-level control system of an all-terrain mobile robot. In: 2009 International Conference on Advanced Robotics (ICAR), pp. 1–6. IEEE (2009)
3. Bascetta, L., Cucci, D., Magnani, G., Matteucci, M., Osmankovic, D., Tahirovic, A.: Towards the implementation of a mpc-based planner on an autonomous all-terrain vehicle. In: Proceedings of Workshop on Robot Motion Planning: Online, Reactive, and in Real-time (IEEE/RJS IROS 2012), pp. 1–7 (2012), http://cs.stanford.edu/people/tkr/iros2012/schedule.php
4. Brück, D., Elmqvist, H., Mattsson, S.E., Olsson, H.: Dymola for multi-engineering modeling and simulation. In: Proceedings of Modelica, Citeseer (2002)
5. Cucci, D.A., Matteucci, M.: Position tracking and sensors self-calibration in autonomous mobile robots by gauss-newton optimization. In: 2014 IEEE International Conference on Robotics and Automation (ICRA). IEEE (to appear, 2014)
6. Cucci, D.A., Matteucci, M.: On the development of a generic multi-sensor fusion framework for robust odometry estimation. Journal of Software Engineering for Robotics 5(1), 48–62 (2014)
7. Freese, M., Singh, S., Ozaki, F., Matsuhira, N.: Virtual robot experimentation platform V-REP: A versatile 3D robot simulator. In: Ando, N., Balakirsky, S., Hemker, T., Reggiani, M., von Stryk, O. (eds.) SIMPAR 2010. LNCS, vol. 6472, pp. 51–62. Springer, Heidelberg (2010)
8. Hellstrom, T., Ringdahl, O.: Real-time path planning using a simulator-in-the-loop. International Journal of Vehicle Autonomous Systems 7(1), 56–72 (2009)
9. Jayakumar, P., Smith, W., Ross, B.A., Jategaonkar, R., Konarzewski, K.: Development of high fidelity mobility simulation of an autonomous vehicle in an off-road scenario using integrated sensor, controller, and multi-body dynamics. Tech. rep., DTIC Document (2011)
10. Koenig, N., Howard, A.: Design and use paradigms for gazebo, an open-source multi-robot simulator. In: Proceedings of 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004), vol. 3, pp. 2149–2154. IEEE (2004)
11. Kümmerle, R., Grisetti, G., Strasdat, H., Konolige, K., Burgard, W.: g^2o: A general framework for graph optimization. In: 2011 IEEE International Conference on Robotics and Automation (ICRA), pp. 3607–3613. IEEE (2011)
12. Laue, T., Spiess, K., Röfer, T.: SimRobot – A general physical robot simulator and its application in roboCup. In: Bredenfeld, A., Jacoff, A., Noda, I., Takahashi, Y. (eds.) RoboCup 2005. LNCS (LNAI), vol. 4020, pp. 173–183. Springer, Heidelberg (2006)
13. Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., Berger, E., Wheeler, R., Ng, A.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3 (2009)
14. Rohmer, E., Singh, S., Freese, M.: V-REP: A versatile and scalable robot simulation framework. In: IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1321–1326 (2013)
15. Vasconcelos, J., Elkaim, G., Silvestre, C., Oliveira, P., Cardeira, B.: Geometric approach to strapdown magnetometer calibration in sensor frame. IEEE Transactions on Aerospace and Electronic Systems 47(2), 1293–1306 (2011)