Simulation of Artificial Intelligence Agents using Modelica and the DLR Visualization Library

Alexander Schaub Matthias Hellerer Tim Bodenmüller
German Aerospace Center, Robotics and Mechatronics Center
Münchner Straße 20, 82234 Weßling

Abstract

This paper introduces a scheme for testing artificial intelligence algorithms of autonomous systems using Modelica and the DLR Visualization Library. The simulation concept follows the 'Software-in-the-loop' principle, whereas no adaptations are made to the tested algorithms. The environment is replaced by an artificial world and the rest of the autonomous system is modeled in Modelica. The scheme is introduced and explained by using the example of the ROboMObil, which is a robotic electric vehicle developed by the DLR's Robotics and Mechatronics Center.

Keywords: Simulation of Artificial Intelligence Agents; Autonomous Systems; Software-in-the-Loop; DLR Visualization Library; ROboMObil

1 Introduction

The variety of autonomous systems, or also known as artificial intelligence agents (AIA), can range from small toys like Lego mindstorms to full-sized robotic cars like the ROboMObil (ROMO)[1]. In all cases an agent consists of three essential parts: sensors, the core artificial intelligence for the agent's functionality, and actuators [2]. The agent perceives its current environment through its sensors, interprets it and plans the next actions to reach its goal before acting upon the environment through its actuators. For a sufficient simulation of an autonomous system the bidirectional connection of an agent to its environment must be considered.

In the past decade several open source simulation environments for autonomous systems, mostly for robots, have been launched due to increasing computational power and decreasing hardware costs, which have made the use of autonomous (mobile) robots feasible for education.

Published in 2001, the socket-based device server Player in combination with the multi-robot systems simulator Stage [3] was widely used in academia and industry. Player provides simple TCP sockets to external devices like sensors and actuators. Player is language neutral and uses the UNIX abstraction of devices being considered as files. Stage is a simulation environment for multiple robots with computationally cheap, but in terms of fidelity only sufficient models. The linear scaling with the population of the simulated world was very important. It is a 2D simulator for indoor scenarios. The simulated sensors are rather simple laser range finders or sonar than complex sensors like cameras.

In 2003 Gazebo [4] was released to satisfy the need for a 3D simulation environment for Player. It enables the simulation of cameras, uses rigid body models, and still works, despite the increased complexity, with simulating several autonomous systems concurrently.

Nowadays the Robot Operating System ROS [5] is the most popular environment for connecting algorithms, sensors, and actuators of robot systems. Many functions and drivers were adopted from Player. Moreover, it also uses interfaces to Stage for 2D and to Gazebo for 3D simulations.

Microsoft's Robotics Developer Studio [6] is a free, but not open source, development suite using user friendly techniques for visual programming, easy parallelization, and debugging via web-interfaces. It is equipped with a DirectX based Visualization, its own rigid-body physics engine, and provides interfaces to commercial products from FischerTechnik, iRobot, Lego etc.

Furthermore, there are also several commercial robot simulators like the Virtual Robot Experimentation Platform V-Rep [7] or Webots [8].

Proprietary simulation environments were developed for larger projects like Junior - Stanford's robotic car for the DARPA Urban Challenge [9]. That proprietary software can be adapted to special demands, which are not completely fulfilled by generalized tools like the ones named before.

All of the mentioned simulation environments use physics engines like Bullet Physics Engine [10] or Open Dynamics Engine [11], which have a strong gaming or computer animation background and provide rigid body modeling and collision detection. They try to reach a fast computation while providing a sufficient accuracy of the physics. Modelica provides several advantages being able to model complex physical systems containing e.g. flexible-bodies, electrical and hydraulic components. To be used for the simulation of artificial intelligence agents Modelica has to be extended by an advanced visualization like the DLR Visualization Library [12]. The combination of Modelica with the DLR Visualization Library creates a powerful tool for an efficient development of complex physical agent and environment models.
Our motivation for the presented scheme is a bidirectional autonomous systems simulation, which combines complex Modelica models of the ROMO with the artificial intelligence system used in the real vehicle.
The remaining chapters are organized as followed: The second chapter provides an overview of our simulation concept. Chapter three gives a detailed explanation of the used tools and interfaces. Afterwards, the results of a simple example will demonstrate the functionality of the AIA simulation scheme. Finally, we will conclude with a brief summary and outlook.

2 Concept of the AIA Simulation

The main target for the proposed scheme is a 'Software-in-the-Loop' simulation, which means that no changes are made to the algorithm that should be tested. In order to test the artificial intelligence of an autonomous system the perception, planning, and control algorithms are kept and its hardware and the environment are simulated. The system's hardware is substituted by a Modelica model, where the detail of the model varies depending on the purpose of the simulation. It can range from a rigid body model to an overall system model containing electrical, flexible, hydraulic, thermal, and tire (sub-)models.
The second step is the replacement of the environment by using the DLR Visualization Library, which extends Modelica by an advanced visualization and interactive simulation. Standard sensors for velocity, torque etc. are part of the basic Modelica library, but complex perception sensors like cameras require this advanced visualization for a sufficient simulation. The algorithms tested with this scheme and also their interfaces to the rest of the autonomous system do not have to be changed. Hence, the algorithms have to run outside the Modelica environment during the simulation, which is made possible by the interactive interfaces provided by the DLR Visualization Library.
The proposed simulation scheme using the example of the ROMO is depicted in Figure 1. The main distinction is made between the autonomy hardware and the simulation hardware. Both can run on the same PC, but the hardware of the autonomous system usually consists of several connected processing units. The intention is to follow the 'Hardware-in-the-Loop' principle and to connect the AIA system to a simulation PC.
The primary perception sensors of the ROMO are cameras, which are widely used in modern autonomous systems, as they provide a great variety of information [13]. A typical cycle of the scheme starts with the virtual cameras taking images of the simulated environment. The images and other sensor data is packed according to the SensorNet format and passed into the shared memory of the autonomy hardware. The interface from the Visualization library to SensorNet is described later in detail. Different algorithms that process and interpret those data can access the shared memory concurrently. The processed data is passed to the planning module both directly and via a module that updates the environment representation. The planned trajectory and other control data is passed via an interactive interface to the Modelica model. Sensors are triggered and the controller gets its reference input. In this example the vehicle dynamics controller is nested in the simulation module, since it does not run on the same hardware as the autonomous driving components in the real vehicle. With the controller commanding the actuators the ROMO model moves in the virtual world and the loop is closed.
Such a 'Software-in-the-loop' scheme for autonomous systems has several advantages. It is possible to test an algorithm under reproducible settings, which is usually not the case in reality. The camera-based perception is very sensitive to changing light conditions. Additionally, it is difficult to keep relative positions and velocities of objects the same in every test. The reproducibility is also desirable for comparing different algorithms and an essential requirement for reverse engineering.
Moreover, algorithms can be tested with optimal conditions. At the very beginning of an algorithm development it is helpful to see if the general concept is working while neglecting sensor and actuator noise

DOI 10.3384/ecp12076339

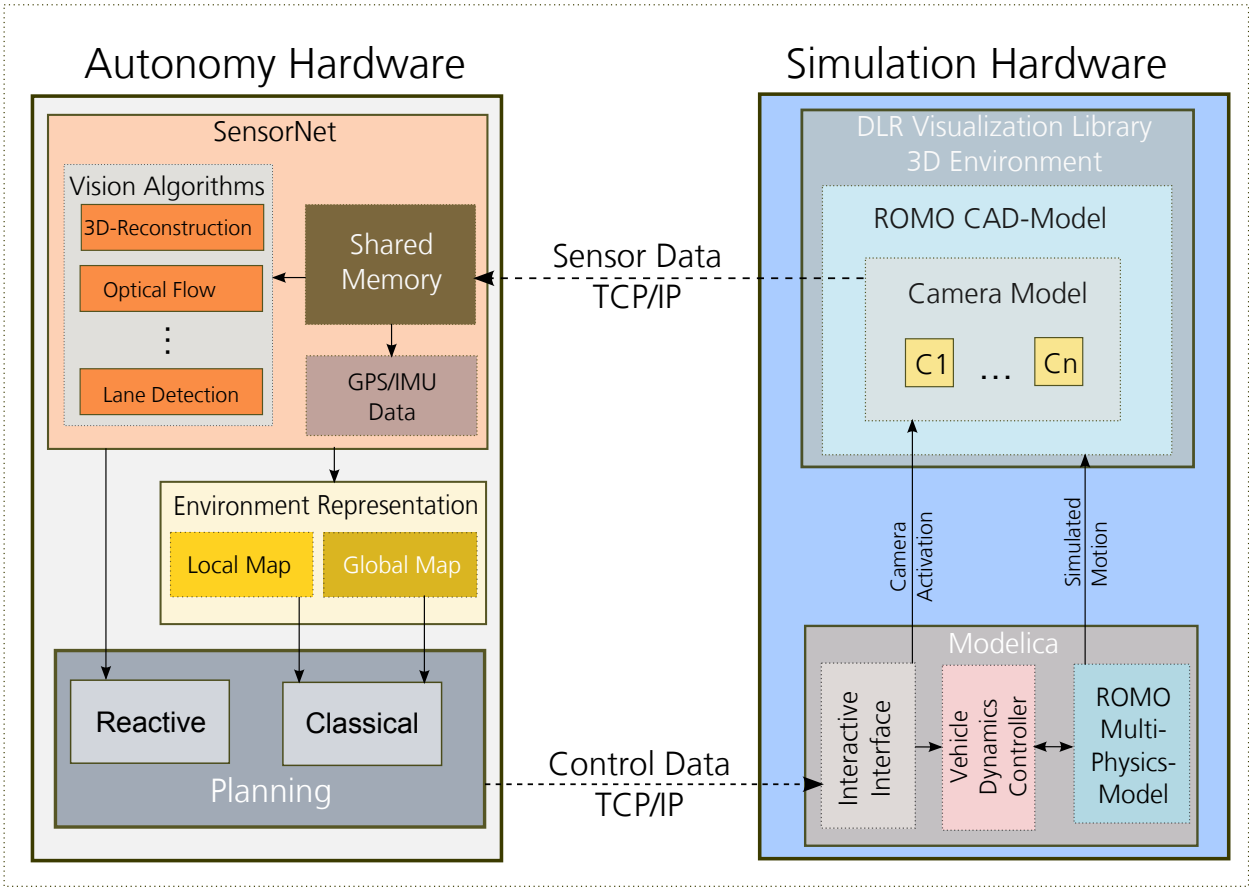

Figure 1: ROMO AI-Simulation Concept

and other disturbing influences. After the basic functionality has been proven the noise can be increased step by step to test different levels of robustness.
Another advantage is the adjustable level of abstraction. Autonomous driving software can be evaluated firstly with a simple model of the system's dynamics. Limitations of actuators can be neglected, sensors can be considered as all knowing and physical constraints can be softened. During the development process the model complexity can be raised to achieve a more realistic behavior of the simulated system. Furthermore, the developer can build a virtual world according to his needs. New types of sensors and systems can be modeled that do not exist yet. The preparation and execution of tests done by the simulation scheme is much faster than tests in reality. The simulation can be run faster than real time causing less costs and posing no harm for people and equipment. Nevertheless, there is still the need for tests with the real system, but their frequency can be considerably decreased. Hence, the 'Software-in-the-Loop' principle is very helpful for rapid prototyping.

3 Combining Virtual Reality with Perception

An essential part of the proposed simulation concept is the link between 3D simulation provided by the DLR Visualization Library and image processing algorithms, which utilize SensorNet for image data dispatching.

3.1 The DLR Visualization Library

The DLR Visualization Library is an extension to Modelica for 3D visualization of simulations. It is composed of two parts: a Modelica library and a standalone program.
The library part defines Modelica multi-body elements which do not influence the simulation's physics but are used for configuration of the simulation's visualization. The visualization is then displayed in a separate application called SimVis. An example of this can be seen in Figure 2. On the left it shows a Modelica model using the DLR Visualization Library library and on the right the corresponding visualization in SimVis. The DLR Visualization Library library provides a

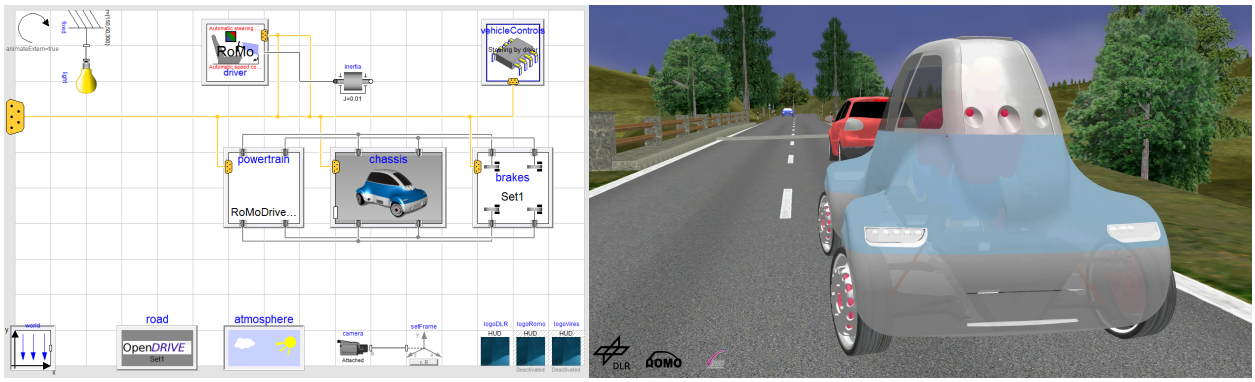

Figure 2: A Modelica model of the ROMO and the corresponding visualization

wide range of 3D objects from simple elements like boxes and gearwheels to complex 3D files to objects defining the representation like Head-Up-Displays showing variables or camera positions both in the 3D environment and their images on the screen.
From a technical perspective this is achieved by utilizing Modelicas C language interface to establish a TCP/IP connection between the simulation and the SimVis application, transmitting information about the configuration of the 3D elements to be displayed [12].

3.2 SensorNet

Modern robotics applications often use cameras and require the real-time analysis of images. The problem for this application is twofold:
First the amount of data is immense. For example a single VGA camera generates about $640 \cdot 480 \cdot 3Byte \cdot 30Hz = 28MByte/s$ of raw data. Moreover, recent video compression methods, e.g. mpeg4 or divx, are computational expensive and also degenerates the image quality and therefore should be avoided in image processing tasks. Additionally, robots interact with their environment. Therefore, real-time restrictions apply to the image processing. The time from image acquisition to a possible reaction has to be minimized. This requires extremely efficient dispatching of image data which is achieved by the communication framework SensorNet. It is designed to provide sensor data, e.g. from cameras, with low latency to multiple, concurrent applications. Therefore, previous concepts on local real-time communication via shared memory [14] and on unified description of camera and range sensor data [15] are combined and extended in the SensorNet data streaming concept. In detail, a ring buffer on a shared memory in conjunction with a signaling mechanism is used to distribute data from a server process to multiple client processes with low latency (<100 μs). The interface also comprises data type metadata that allows for type checking. Further, predefined, unified data types are used, e.g. color image or depth image, and act as a abstraction layer. As a result, sensors of same type can easily be exchanged by just replacing the server process. Data can be distributed across system borders by connecting shared memories on the different systems with UDP- or TCP-based data transfer. Additionally, a separate TCP-based configuration channel allows for setting and getting parameters, e.g. camera shutter time, without influencing real-time data streaming.

3.3 Interface

Acquiring images with real cameras under controlled conditions is not always feasible as described above. The intention is to reuse the existing solutions for 3D simulation and image data dispatching.
The DLR Visualization Library cameras are designed for displaying images on screen. Since Modelica is an object-oriented language the new camera model is derived from the existing solution and extended by additional parameters. The cameras are by default aligned within the 3D environment using rigid body transformations and displayed either in the SimVis window or full screen. In both cases the camera resolution is determined as a ratio of either the window size or the screen size for easier portability from one PC to another. In contrast real cameras have a fixed resolution in pixels. The simulation therefore requires this resolution as new parameter. Likewise the image data is always displayed on screen in RGB format, yet cameras often use different formats. This implementation currently supports two alternative formats: YUV and grayscale. Furthermore, SensorNet needs a name for the shared memory object to identify a specific camera and a role for the camera in a stereo setup. If the cam-

DOI 10.3384/ecp12076339

era is part of a stereo setup, both cameras use the same shared memory object. One camera has to be set as master and one camera has to be the slave. Both images will then be acquired simultaneously and put in the same shared memory object, whenever the master camera is triggered.
With these additional parameters set up in the DLR Visualization Library, SimVis can also reuse the majority of its existing camera implementation. The main difference lies in the render target. Normal cameras render to a frame buffer that is then displayed on screen. For the simulated cameras the render target is redirected to a frame buffer object (FBO), which is not displayed but read back to the main memory. Thereby, the image is rendered the same way but off-screen and accessible by the application as a data array in RGB format. This image data first has to be converted to the desired image format. Conversion into YUV is carried out using the following equation per pixel:

$$Y = 0.299R + 0.587G + 0.114B$$

$$U = 0.493(B - Y)$$

$$V = 0.877(R - Y)$$

This equation is also applicable for conversion in grayscale by only using the Y component describing the pixels luminance [16]. The preprocessed image is then packed into one of SensorNets default image formats, a time-stamp corresponding to the current simulation time is added and the image is released. Releasing an image in the SensorNet context makes it available for other applications. The primary focus lies on image interpretation algorithms that are part of an artificial intelligence agent.

4 Experimental Results

The proposed scheme is evaluated by a simple example, in which a testing environment for a vision based control (VBC) platooning algorithm is created.
The basic idea is that the ROMO follows a preceding car, while using only a front stereo camera pair for perception. Initially, the ROMO has only an image of the back of the target car, which was taken from an appropriate distance for following.
After the platooning mode is activated the ROMO tries to find the target car in the current camera image. Analog to vision based robot control [17] the goal is to see the target object in the same size and at the same angle as in the reference image. Therefore, the ROMO tries to reach and hold the very same relative position in which the initial image was taken. The target position can also be made velocity dependent for keeping the minimum safety clearance.
A simulation model in Dymola is created. First, the ROMO's top front camera pair, refer to Figure 3, is modeled by using the DLR Visualization Library's SensorNet camera class, which was developed for the proposed simulation scheme, with the appropriate parameterization. They are attached at their respective positions to a 3D geometry model of the ROMO, which is extracted from CAD. An overall model of the ROMO containing all actuators, sensors, the electrical systems etc. can be used, but the example focuses on the perception part, which is the main interest in this paper. Buildings, streets, and a surrounding land-

Figure 3: The DLR's ROboMObil

scape are placed in the virtual environment. For this the DLR Visualization Library provides an integration block for common 3D files like the '.3ds' format. The target car consists of an animated 3D model bound to a trajectory block that moves the object within the virtual world. Now the simulation is started and images are sent to the shared memory via SensorNet. The AIA algorithms receive a notification that new images are available and begin to run. First, a SensorNet implementation of the DLR's 3D reconstruction algorithm, called Semi-Global Matching (SGM)[18], calculates depth information out of the two images from the stereo camera. The result can be seen in Figure 4, whereas the left part shows the scene recorded by the stereo cam and the right part a visualization of the depth values. The color value of every pixel is set ac-

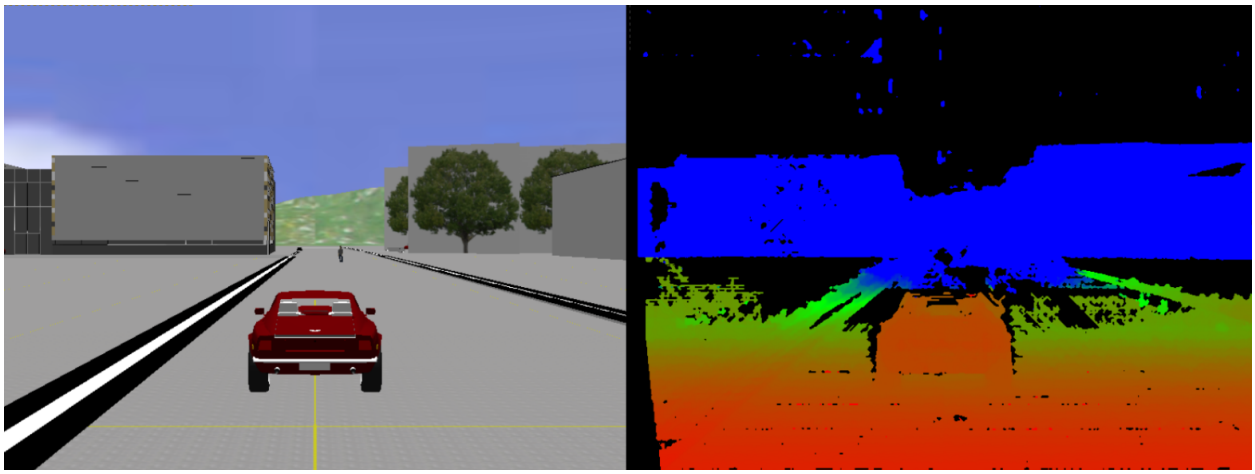

Figure 4: Semi Global Matching applied to the virtual images

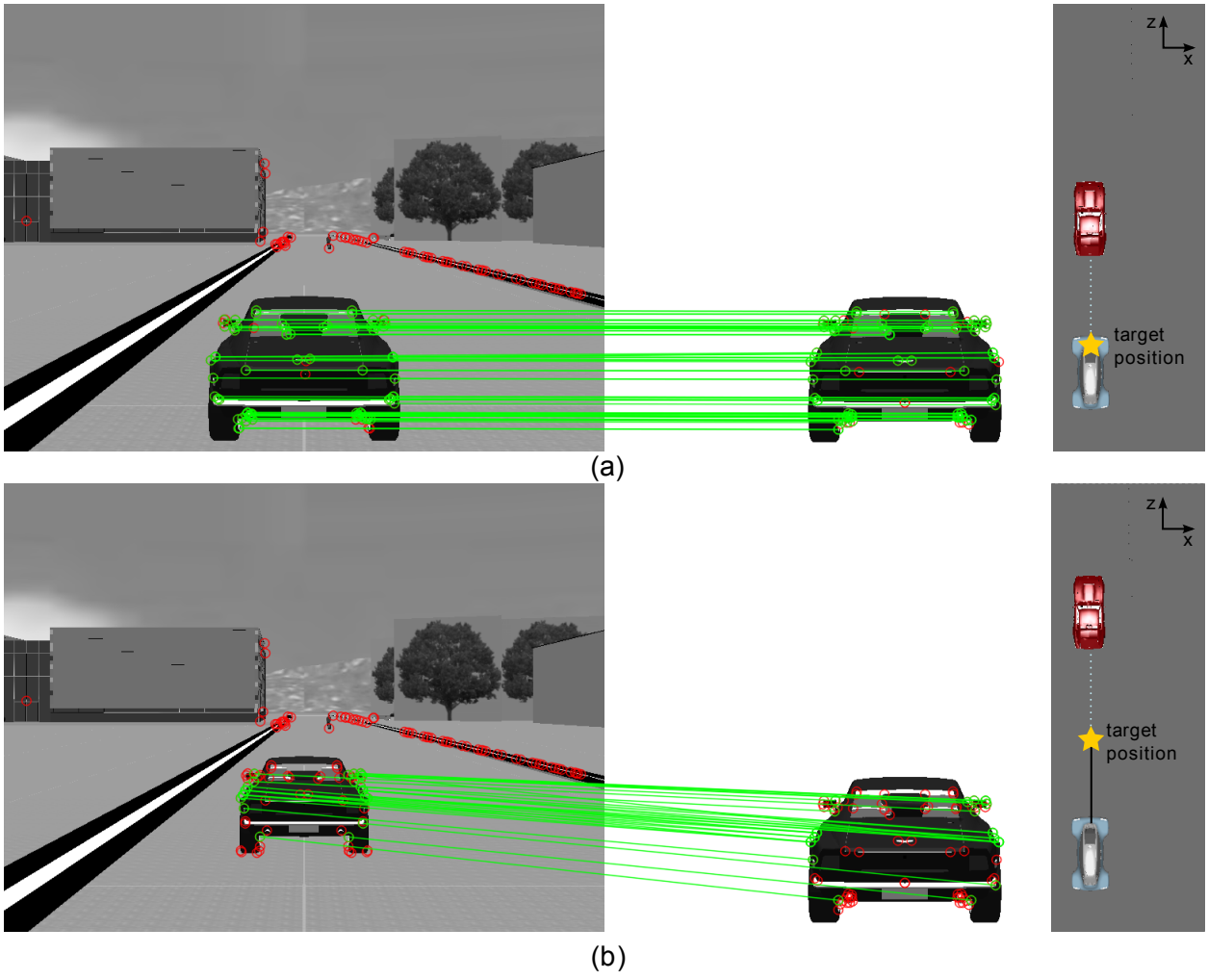

Figure 5: Matching features for estimating the relative position: (a) At target position (b) 4 meters deviation in camera direction

cording to the depth value at that point. Small values are colored red and with increasing depth they go from orange to green to blue. Black parts of the depth image are either too far away like the sky or cannot be reconstructed. This is often the case in regions with homogeneously textured surfaces, where the reconstruction algorithm cannot find matching points in both images. After calculating the depths the SGM writes a struc-

DOI 10.3384/ecp12076339

ture consisting of a rectified actual image, a quality map, and the depth image back into the shared memory. This structure is accessed by the VBC car following algorithm, which starts with running a feature detection algorithm, e.g. AGAST [19], on the actual image. A descriptor for matching is calculated for every feature point and the 3D coordinates of every feature point are determined using the depth image from SGM. The keypoints of the target image with their descriptors and 3D values are available a priori and so descriptors are compared to find matches in both images. The results can be seen in Figure 5, whereas the right is the target and the left one is the current camera image of the simulation. Matching keypoints are connected with a green line. Besides the markings there is no color in Figure 5, as the feature detection works with grayscale images. At least four matching keypoints are randomly selected. By using their respective 3D coordinates the rotation matrix R and translation vector T between the current keypoints and the target keypoints are calculated. The quality of the estimated R,T is measured by applying R,T to all keypoints of the current image that have matches. After that they are projected back into the 2D image space and the distance to their matching points in the target image is measured and summed up over all keypoints. The whole procedure is repeated with other sets of features until the quality of R,T is sufficient or a certain number of iterations is exceeded and the best iteration will be kept.

The preceeding car in the simulation starts at the target relative position and moves four meters in the camera's z direction, whereas the ROMO remains stationary. The matches at the beginning can be seen in Figure 5a and that the end of the movement is depicted in Figure 5b. The number of found correspondences decreases during the movement. This is normal on the one hand due to the changed perspective, but on the other hand it is additionally disadvantaged here by the simple textures, which lead to weak feature points. Incorrectly matched or too few feature points can disturb the results of the algorithm immensely.

Nevertheless, the simulation has shown the general functionality of the algorithm as can be seen in Figure 6. The z value of the deviation to the goal position changes from 0 to 4000mm, while the target car moves in z-direction. The deviaton to the real position can be due to badly chosen feature points, depth measurement errors, or imprecise calibration of the virtual cameras. Based on the computed rotation and translation a trajectory can be calculated and fed back into the Modelica simulation via a TCP/IP channel to control the simulated ROMO in its virtual environment.

In this early state of the algorithm's development the AIA simulation scheme is very helpful to identify weaknesses and increase robustness before tests with the real ROMO are possible.

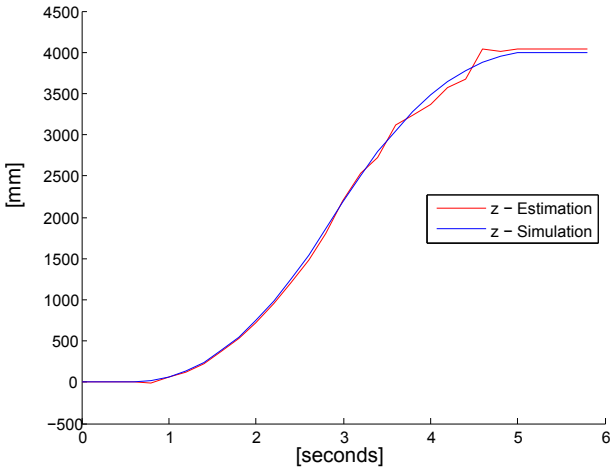

Figure 6: The deviation to the relative target position. z is in camera direction

5 Conclusions and Future Work

This paper presents a scheme for testing artificial intelligence algorithms for autonomous systems according to 'Software-in-the-loop' and 'Hardware-in-the-loop' principles. Existing multi-physics models are combined with the actual artificial intelligence algorithm that do not have to be adapted for the simulation. This is achieved by extending the DLR Visualization Library by an interface to the sensor data management tool SensorNet, which is utilized in real autonomous systems in DLR's Robotics and Mechatronics Center.

The capability of the concept is proven by a short example, in which the translation and rotation to a leading vehicle are determined by a vision based car following algorithm.

In further developments more sensor types, which are typically used in autonomous systems like a dGPS aided Inertial Measurement Unit (IMU), will be modeled. Camera models can be extended with more realistic effects such as lens distortions. Moreover, we plan to use virtual objects with more complex textures to generate more realistic virtual pictures. In order to validate the simulation results they have to be compared to those using data taken from vehicle tests.

Acknowledgements We thank the following colleagues for supporting us with their expert knowledge: Darius Burschka (image processing), Martin Otter (Modelica), Tobias Bellmann (DLR Visualization Library), Heiko Hirschmüller (SGM), Klaus Strobl (camera calibration), and the whole ROboMObil Team.

References

[1] Jonathan Brembeck, Lok Man Ho, Alexander Schaub, Clemens Satzger, and Gerhard Hirzinger. Romo - the robotic electric vehicle. In *22nd International Symposium on Dynamics of Vehicle on Roads and Tracks*. IAVSD, 2011.

[2] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, third edition, December 2009.

[3] B. Gerkey, R. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *11th International Conference on Advanced Robotics (ICAR 2003)*, Coimbra, Portugal, June 2003.

[4] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *Intelligent Robots and Systems, 2004. (IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, volume 3, pages 2149 – 2154 vol.3, sept.-2 oct. 2004.

[5] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[6] J. Jackson. Microsoft robotics studio: A technical introduction. *Robotics Automation Magazine, IEEE*, 14(4):82 –87, dec. 2007.

[7] Marc Freese, Surya P. N. Singh, Fumio Ozaki, and Nobuto Matsuhira. Virtual robot experimentation platform v-rep: A versatile 3d robot simulator. In Noriaki Ando, Stephen Balakirsky, Thomas Hemker, Monica Reggiani, and Oskar von Stryk, editors, *SIMPAR*, volume 6472 of *Lecture Notes in Computer Science*, pages 51–62. Springer, 2010.

[8] O. Michel. Webots: Professional mobile robot simulation. *Journal of Advanced Robotics Systems*, 1(1):39–42, 2004.

[9] M. Montemerlo, S. Thrun, and et al. Junior: The stanford entry in the urban challenge. *Journal of Field Robotics*, 2008.

[10] http://bulletphysics.org - 15.05.2012.

[11] Russell Smith. Open dynamics engine, 2008. http://www.ode.org/.

[12] Tobias Bellmann. Interactive simulations and advanced visualization with modelica. In *Proceedings 7th Modelica Conference, Como, Italy*, 2009.

[13] Alexander Schaub, Jonathan Brembeck, Darius Burschka, and Gerd Hirzinger. Robotic electric vehicle with camera-based autonomy approach. *ATZelektronik*, 2(2):10–16, April 2011.

[14] G. Hirzinger and B. Bauml. Agile robot development (ard): A pragmatic approach to robotic software. pages 3741 –3748, oct. 2006.

[15] T. Bodenmuller, W. Sepp, M. Suppa, and G. Hirzinger. Tackling multi-sensory 3d data acquisition and fusion. In *Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on*, pages 2180 –2185, 29 2007-nov. 2 2007.

[16] Charles Poynton. *Digital Video and HDTV Algorithms and Interfaces*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1 edition, 2003.

[17] F. Chaumette and S. Hutchinson. Visual servo control. i. basic approaches. *Robotics Automation Magazine, IEEE*, 13(4):82 –90, 2006.

[18] H. Hirschmuller. Stereo processing by semiglobal matching and mutual information. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 30(2):328 –341, 2008.

[19] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *Proceedings of the 11th European conference on Computer vision: Part II*, ECCV'10, pages 183–196, Berlin, Heidelberg, 2010. Springer-Verlag.