Preprint of a paper which appeared in the Proceedings of the 2008 International Conference on Simulation, Modelling and Programming for Autonomous Robots (SIMPAR)
Venice, Italy, Nov. 03 - 07 2008

Multilevel Testing of Control Software for Teams of Autonomous Mobile Robots

Sebastian Petters, Dirk Thomas, Martin Friedmann, and Oskar von Stryk

Technische Universität Darmstadt, Department of Computer Science
Hochschulstr. 10, D-64289 Darmstadt, Germany
{ petters, dthomas, friedmann, stryk }@sim.tu-darmstadt.de,
WWW home page: http://www.sim.tu-darmstadt.de

Abstract. Developing control software for teams of autonomous mobile robots is a challenging task, which can be facilitated using frameworks with ready to use components. But testing and debugging the resulting system as teached in modern software engineering to be free of errors and tolerant to sensor noise in a real world scenario is to a large extend beyond the scope of current approaches. In this paper multilevel testing strategies using the developed frameworks RoboFrame and MuRoSimF are presented. Testing incorporating automated tests, online and offline analysis and software-in-the-loop (SIL) tests in combination with real robot hardware or an adequate simulation are highly facilitated by the two frameworks. Thus the efficiency of validation of complex real world applications is improved. In this way potential errors can be identified early in the development process and error situations in real world operations can be reduced significantly.

1 Introduction

Development of control software for teams of autonomous robots imposes many challenges on the developer. The software is usually highly complex, containing modules for very different tasks (like motion generation, sensor data fusion or behavior control). To ensure operation of such systems, each module of the control software must (1) be free of errors and (2) tolerate noise and errors from other sources. A special class of robots targeted at in this paper are "lightweight" robot systems characterized by inertially stabilized high motion dynamics and limited onboard sensing and computing capabilities due to payload restrictions like small humanoid robots, small unmanned aerial or marine vehicles.

As autonomous mobile robots are operated in environments with large uncertaines, the software must be tolerant to noise and disturbances. To examine the abilities of an autonomous robot all individual modules of the control software as well as the complete system have to be tested extensively. Testing the software for autonomous mobile robots is a complicated challenge, which can only be met if the developer is equipped with appropriate tools. One major problem when testing such software is the fact, that the source of an error is often not obvious. An error can usually be caused by one of the modules involved, or it can be

caused by external influences like changing environmental conditions. Errors in the control software can also be shadowed by such external influences, so that such an error does not become obvious. E.g. if the tracking of an oject fails, the reason may be in the vision module, a calculation error in the world model, in the behavior control or an unexpected input like a falsely recognized new other object in the scene. It may also be possible that an error exists in the sensor fusion module which is misinterpreted as an effect of noisy sensor data.

In this paper several ways of ensuring the quality of robot control software through testing are discussed. These methods include component tests of the control software before using it, testing the software with software-in-the-loop (SIL) simulations, monitoring the performance of the control software during real world operations and using offline evaluation afterwards. Crucial for the efficiency of such tests is the availability of a robot middleware with potential capabilities for flexible monitoring and remote debugging as well as simulations of the robots capabilities for sensing of and interacting with the physical world on different levels of detail depending on the current SIL test.

For facilitating the testing process, the used software architecture should therefore in general provide the following features:

- Extendable testing framework to allow implementation of new component tests,
- modular design to test different parts of the control software independently,
- flexible and easy to use communication mechanisms to enable data exchange with a remote computer for debugging,
- an extendable graphical user interface for visualization,
- built-in features allowing offline debugging e.g. a recording/playback tool,
- and a simulation framework allowing different layers of realism.

2 Existing Technologies

2.1 Robot Control Software

In the last decades several architectures for robot control software have emerged. All of them try to facilitate the challenging and thus error-prone task of the developers by providing solutions for common problems as tested and ready to use components. Current approaches especially differ in the targeted robot platforms and the scope, for which components are provided.

Frameworks like Microsoft Robotics Studio [1] and the CORBA [2] based Miro [3] are focused on systems with a significant amount of computational power, e.g. multi processor systems, and provide effective communication mechanisms. For "lightweight" robot systems with only very limited onboard computational power these frameworks have the disadvantage of a relatively large overhead which further restricts computational resources available to robot control software.

Robot device interfaces try to standardize the access to sensors and actors by providing an easy to use driver layer. CLARAty [4] for example contains reusable

components which can easily be adapted to different robot platforms but does not support teams of robots. The Player Project [5] provides an interface to access different hardware over a network and supports multiple programming languages i.e. C++, Java and Python. There exists drivers for the simulation frameworks Stage (2D) and Gazebo (3D) [6], which allows development of robot control software without the real hardware. URBI [7] follows a similar approach, but only supports a C++ like scripting language. These device interfaces do not perform very well in the development of complex robot control applications for teams of autonomous robots due to the lack of flexible communication mechanisms which are essential for modular large scale applications.

Integrated robot control software architectures like Webots [8] or Saphira [9] allow the development of software for robots like the Pioneer 2DX, Bioloid (Robotis), AIBO (Sony) and Nao (Aldebaran Robotics), mainly for educational or research purposes. They contain graphical user interfaces and a simulator and provide components to construct own robots from commonly used sensors and actuators. Webots also allows the development of sofware for swarms of robots and evolutionary algorithms. Due to the focus on a specific plaform, it is not possible to develop software for teams of heterogeneous robots.

2.2 Robot Simulations

Most existing 3D simulations rely on external packages for physics simulation. Very often the Open Dynamics Engine (ODE) [10] is used, e.g. in Webots [8], SimRobot [11] or Gazebo [6]. Other packages used are PhysX [12] by NVIDIA (used in the Microsoft Robotics Studio [1]) or game engines like the Unreal Engine [13] (used in USARSim [14]).

Depending on the current testing task, requirements on the robot simulation vary widely. High physical accuracy may be necessary under some circumstances (e.g. for motion optimization), but not important for other scenarios (e.g. testing of team coordination). Also physics-based robot simulation may impact the real time performance severly. Often there is a tradeoff between accuracy of the simulation and size of the team.

If a simulation depends on external packages for physics simulation or other purposes, adjusting the accuracy (and thus the real-time performance) of the simulation is complicated. One solution to this problem is using different simulations, e.g. Gazebo [6] for 3D physics simulation and Stage for 2D simulation of large teams. As long as the simulations provide the same interface to the control software (as is given in the Player/Stage/Gazebo project), this approach is practicable. If this precondition is not fulfilled, it becomes necessary to model the robots and to provide suitable connections to the control software for each simulation.

2.3 Testing Strategies

Automated tests are a widely used tool in software engineering today. In contrast to formal verification, which is not feasible for complex systems, automated tests

check the correctness of a software component for a predefined set of samples. Testing can only be used to detect the effects of errors, but not the reasons for the errors. The absence of failed tests is not a proof for the correctness of the software as long as the perfomed tests do not cover all possible inputs and internal states.

To ensure a specified functionality automated unit-, regression-, integration- and stress-tests [15] are used during the development process of software components and applications. But even if there are many tools available to simplify the process of testing, studies show their acceptance heavily depends on the time needed to setup up and perform the tests [16]. As a consequence usefull tools actively have to support the developers by keeping the efforts of testing to a minimum.

2.4 Summary of Existing Technologies

Existing solutions for robot control software and 3D simulation packages aim at supporting the developers by providing easy to use components. The process of testing and validating the resulting system in real world applications is nevertheless mostly beyond the scope of the current approaches, especially for scenarios where multiple heterogeneous robots interact with each other. The capabilities of first approaches using logfiles and graphical user interfaces for later offline analysis [17] are quite limited. For a more detailed analysis during runtime, the debugging mechanisms should be tightly integrated into the whole system, easily accessible via small interfaces and with a low processing overhead.

The previously mentioned testing strategies from software engineering are only applicable to the low-level functionality of robot control software. It is impossible to specify test cases which cover all possible input data which could occur in an environment which is far from being fully predictable because of the infinite many situations of sensing of and interaction with the physical world. Due to potential hardware wearing, numerous automated tests with the real robot system may not be desired. For this reason the standard strategies of software engineering are only of limited use to test robot control software.

In situations where multiple components are developed independently, it is also necessary to enable tests of individual components. Depending on the test case it may also be desired to simplify the surrounding system by partly replacing other components using an adequate simulation matching the current situation. This approach reduces the overall complexity of the test scenario and thus facilitates the identification of the source of an error.

3 Developed Technologies

3.1 `RoboFrame`

`RoboFrame` [18, 19] has been developed in the authors group to meet the special requirements of heterogeneous teams of lightweight autonomous robots. The

source code is available for non-commercial research and educational usage. It is implemented in object oriented ANSI C++ and contains a platform abstraction layer to support Windows 2000/XP/Vista/CE as well as various Linux and Unix derivates and Mac OS X as underlying operating system. Due to short development cycles of new robot hardware components and fast changing requirements caused by complex scenarios, `RoboFrame` provides flexible communication mechanisms and easy exchangeable modules, which encapsulate algorithms for image processing, world modeling, behavior control and motion generation. Modules can be added to multiple threads, which can be executed at a given frequency or if new data to process arrives.

For data exchange between the modules in one application a shared memory can be used. The preferred way of data exchange of smaller data packages is a message based communication, which allows transparent communication even via network. Messages can be of arbitrary type or complex data structures to support any kind of application specific message. To handle application specific messages advanced serialization mechanisms are provided. Modules can request messages from other modules without having to worry about the current process layout. All data packages automatically get a source address to identify the sender of a message and are timestamped.

For communication via network, both unreliable, but fast UDP and reliable TCP is supported. Depending from the required reliability and performance, the appropriate protocol can be selected. Usually the faster UDP is used for team communication between the robots while TCP is used for debugging or remote control connections.

For debugging, monitoring and remote control purposes a graphical user interface (GUI) which allows connections to multiple robots is part of `RoboFrame`. All messages within an application or additional data for debugging purpose, which is only generated if requested, can be send to the GUI and can be visualized in their respective context by application specific dialogs. It is also possible to send data to an application, i.e. to reconfigure the application or a module or to test certain modules.

The GUI also contains a dialog to record messages sent from the connected applications. The messages can be replayed for later analysis or can be sent to the application. This allows repetitive tests with the same data and thus enables the investigation of changes made to the modules.

In contrast to other existing architectures `RoboFrame` itself does not make any assumptions about the applications on top of it. Neither any message types nor any modules are provided by the framework itself. Instead `RoboFrame` enforces the development of components which can be reused in different applications.

3.2 `MuRoSimF`

The **Mu**lti-**Ro**bot-**Sim**ulation-**F**ramework [20] enables to create simulations for heterogeneous teams of autonomous mobile robots. A key feature of `MuRoSimF` is that algorithms used for the simulation (e.g. simulation of robot motions or sensors) can be exchanged transparently. As algorithms for the same purpose

exist on different levels of physical detail resp. computational complexity (e.g. robot motion simulation based on kinematics or multibody system dynamics), simulations can be tailored to be adequate to a given testing task with respect to the level of detail and precision of simulation as well as number of robots simulated simultaneously at real-time.

`MuRoSimF` provides several algorithms for simulation of biped, quadruped and wheeled locomotion on different levels of detail. Algorithms for the simulation of external sensors like cameras and laser scanners as well as for internal sensors like gyroscopes, accelerometers and joint encoders are provided. All can be extended or replaced.

3.3 Integration of Simulation and Control Software

Simulations created with `MuRoSimF` can be connected easily to control programs based on `RoboFrame`. External software can be connected to the simulation using serial communication (virtual or real RS232 connections as well as TCP). `MuRoSimF` provides so called *controllers* which are software modules allowing to communicate with sensor and actuators of the simulated robots. Within the control application modules exist which can communicate with the respective controllers of the simulation. When connecting the control software to the simulation instead of the real hardware, only these modules have to be adapted while the core modules of the application remain the same. In case the real robot is connected by RS232 to the control computer, the connection to the simulation will be completely transparent, as RS232 is provided as a way of communication.

Many robot designs (e.g. [21]) incorporate special controller hardware for real time control of a reflex layer (e.g. gait generation and control for walking robots or motion control for wheeled vehicles). Such controllers have significant parts of software of their own. To enable the SIL-testing of this software, it is possible to recompile the central functions into a dynamic link library and execute these functions within the simulation.

The simulation framework also provides the capability to extract information from the scenario like ground truth data of the simulated objects. These information can be used to bypass some processing components in the real application to simplify the complexity of the application for testing purposes.

4 Multilevel Testing Strategies

In this section strategies for testing the control software for teams of autonomous robots will be discussed. Depending on the abstraction level of the software modules under consideration of a test, different approaches will be most useful. The following three testing strategies might all be carried out either *without any hardware*, with *real robots* or with *simulated robots*: (i) component tests, (ii) online testing, (iii) offline testing.

Even if these testing strategies are common knowledge in modern software engineering it may be more or less difficult to perform these tests depending on

the software architecture. The used software architecture and tools can vastly reduce the required affords to setup different types of test scenarios. Using a message based communication it becomes very handsome to alter the data flow of the application and to intercept or inject messages during runtime.

In the scenario of a team of autonomous soccer playing humanoid robots described later on some testing strategies are used as showcases. This scenario provides many challenges, as (1) noisy off-the-shelf sensors and limited onboard computation capacity, (2) the software involved has a high degree of complexity and different levels of abstraction and (3) communication between the robots is unstable. Similar challenges can be found in many other real world applications (e.g. cooperative search and rescue, exploration operations). The software architecture for the example scenario consists of several modules like image processing, world modeling, behavior control, motion generation and inter-robot communication (cf. Fig. 1).

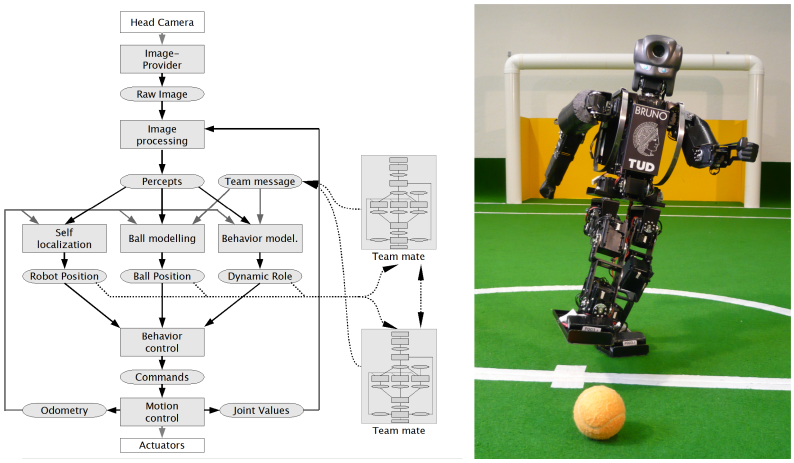

Fig. 1. Software architecture (left) for a team of autonomous soccer playing humanoid robots (right). Modules are depicted as boxes, messages as ellipses. Inter-robot communication is not stable and may be faulty.

4.1 Component Tests

In a deterministic and finite dimensional world unit tests would have a code coverage of nearly 100%. However, this is not ture for such a complex, real world application, since the efforts for creating unit tests for high level functionality are highly increasing. Therefore it is only applicable to parts of the software. In general the low level functionality which involves less source code is better testable using component tests than complex high level functionality.

To ensure the correctness of the message passing system, serialization mechanisms and the shared memory subsystem of `RoboFrame` these parts are covered by a set of component tests.

A prime example for unit tests in robotics are mathematical operations. Their tests do not involve any hardware and the functions are easily testable - mostly even in very small unit, which makes it even simpler to write the component tests. Since algorithms based on mathematical formulars are also better testable than other high-level functionality, some of the application specific models are also covered by component tests, e.g. the odometry model accumulates the odometry of the robot which is measured multiple times per second. Several internal robot modelings, e.g. the relative ball model and the self localization, are based on the integrated odometry model which performs the computations required by other models for different time intervals. A set of unit tests assures its correct functionality.

But other component tests might utilize simulated robots to assure that e.g. inter-robot communication of their own localization is working flawlessly. But the component test is neither implying that there is really any self localization done nor that the robot is really walking or driving around. Therefore an adequate simulation, which provides oracle data of exact robot poses and a simplified odometry reduces the amount of software to be covered by testing enormously by factoring out the influence of the not used code.

4.2 Online Testing

For several high level components unit testing is not a feasible approach. This also applies to cases where the data to check vary in a non-trivial matter e.g. because of noisy input data. A human can easily determine the correctness of the computed output data, where implementing a unit test would be quite expensive if not impossible. Therefore the application architecture must provide a rich set of features to monitor and debug a running application. Especially in mobile robotics the demand to work remotely is significant.

A good example to demonstrate the online testing capabilities of `RoboFrame` is the self localization in the scenario of Fig. 2. The humanoid robot uses an articulated, directed camera to determine it's position and orientation on the soccer field using a particle filter method. The self localization is based on a large set of input data: on one side the odometry model feed by the internal sensors, on the other side various objects recognized by an image processor software like goals, poles, field lines etc. Determining the quality of self localization is not only a test for correctness but even more a benchmark for accuracy of the localization method.

A component test can by its definition only detect the effects of an error but not the reason itself. For this test a different strategy must be used which involve the judgment of a human. Due to the large amount of data the GUI must be capable of visualizing these information in a way a human can easily comprehend any necessary details. These testing strategy allows a human to test a high level component based on the comprehensive visualization.

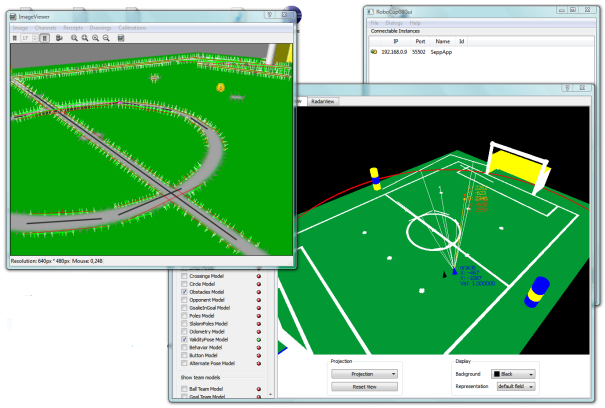

Fig. 2. Graphical user interface visualizing the detected field lines (white lines), position and orientation computed by the self localization (blue arrow) and the ground truth information provided from a ceiling camera or simulator (black arrow).

4.3 Offline Testing

In some scenarios it is not feasible to do testing online in real time. Even the best visualization might not be suitable when the state changes frequently. Furthermore it is not possible to use a debugger during online test to track down the reason of an error. Therefore the third testing strategy involves the logging and replay capabilities of `RoboFrame`.

Any messages saved to a logfile during a former online test can later be replayed and visualized with the same tools used for the live testing which have been described before. This allows feeding the application with e.g. saved sensor messages to repeatedly test the components with the same known input data.

4.4 Software in the Loop Testing

As described in Sect. 3.3 `RoboFrame` and `MuRoSimF` provide communication capabilites allowing SIL-testing of the control software. Depending on what kind of SIL-test is to be performed, different information may be transfered from the simulation to the control application. `MuRoSimF` is capable of providing *adequate* simulations for a wide variety of testing-scenarios. For testing the complete software the simulation can act as a replacement for the real robot's sensing and motion capabilities, processing motion requests from the control application and providing camera images in response.

Besides the normal sensor information of the robot, the simulation can provide any information on the state of the simulation, like position of simulated robots or ball. This ground truth data can be used in multiple ways. A simplified structure of robot control software and simulation is shown in Fig. 3.

One possibility is to verify the performance of a robot's self localization. To do this a complete robot is simulated and the simulation provides further

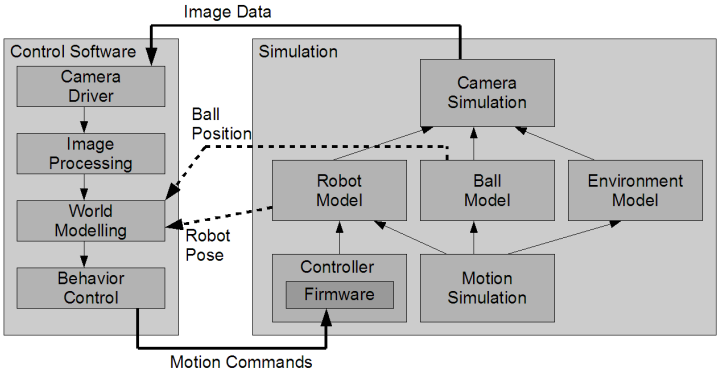

Fig. 3. Data exchange between control application and simulation. Solid lines indicate data similar to the data exchanged with the real robot. Dashed lines are additional informations provided by the simulation. The simulation can be extended to teams of robots by duplicating the robot data models and attaching the new models to the simulation algorithms used.

information on the robots current position and orientation. This information is compared to the output of the self localization modules of the control application.

Another possibility is testing behavior and communication for a team of robots. In this case, the function of the image processing parts of the robots are not investigated as their (potentially wrong) output may shadow errors in the modules under investigation. To perform an adequate test, the simulation will not provide simulated image data and just propagate position information to the control software, removing sources of errors not under investigation.

When testing the low level parts of the robot control software, even less information must be provided (and thus) simulated. If only the motions of the robot are of interest, only motion simulation must be simulated. For evaluation of a robot's actions it is possible to augment the simulation of the simulation with additional data, c.f. Fig. 4.

4.5 Selection of Adequate Testing Strategy

The choice to select one of the strategies for a specific test is always left to the developer of the application. Each and every of the depicted types of testing options have their advantages and disadvantages for a specific purpose as summarized in Table 1.

5 Summary and Outlook

Existing approaches used in modern software engineering are only of limited use for meeting the challenges involved in testing software for a team of autonomous

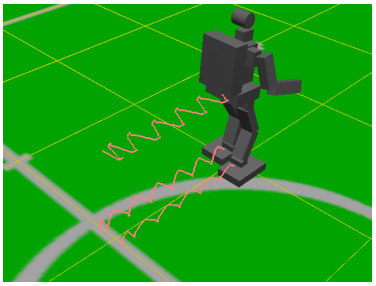

Fig. 4. Evaluation of a robot's walking motion. The simulated robot is augmented with the trajectories of feet and hip.

Table 1. The suitability of the test strategies for different test goals. (+) marks good, (-) marks bad suitability, (o) marks uncertain

Test strategies	Ensure correct computations	Evaluate algorithms with noisy input data	Track down source of an error
Component tests	+	o	-
Online test	-	+	o
Offline test	-	+	+

mobile robots operating in an uncertain environment. They must be extended by further testing techniques. Depending on the application the developers have to consider which testing strategy fits each part of the software best.

The software architecture `RoboFrame` was designed to meet the special requirements stated at the end of Sect. 1. It enables multilevel testing from unit testing over live testing of heterogeneous teams to offline testing with recorded real world input data. Due to the message based communication mechanisms and the dynamic runtime configuration of the framework the efforts to set up a test environment are highly reduced compared to other approaches. The framework `MuRoSimF` enables an adequate robot simulation for each different scenario. The algorithms vary from complex dynamics simulation for testing the motion generation in the loop to simple kinematics but providing ground truth data to concentrate on high level team behavior tests. Furthermore, any of the algorithms can be replaced by custom implementations to provide tailored solutions for any requirement. The source code of `RoboFrame` and `MuRoSimF` is available at no cost for research and educational purposes from the authors. Both developed software frameworks actively support the developers in testing and debugging their applications and thus improve the efficiency which speeds up the development process and results in a higher reliability of the final application.

References

1. Microsoft Robotics Studio, http://msdn.microsoft.com/robotics/, 2007.

2. OMG Object Management Group. CORBA - Common Object Request Broker Architecture, http://www.corba.org. 2007.
3. H. Utz, S. Sablatnög, S. Enderle, and G.K. Kraetzschmar. Miro – middleware for mobile robot applications. *IEEE Trans. on Robotics and Automation*, 18(4):493–497, 2002.
4. I. Nesnas, A. Wright, M. Bajracharya, R. Simmons, T. Estlin, and W.S. Kim. CLARAty: An architecture for reusable robotic software. In *SPIE Aerosense Conference*, Orlando, FL, April 2003.
5. B. P. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage project: Tools for multi-robot and distributed sensor systems. In *Intl. Conf. on Advanced Robotics (ICAR)*, pages 317 – 323, Coimbra, Portugal, 30 June - 3 July 2003.
6. N. Koenig and A. Howard. Gazebo - 3D multiple robot simulator with dynamics, website http://playerstage.sourceforge.net/gazebo/gazebo.html. 2003.
7. Gostai. Urbi - Universal Real-time Behavior Interface, website http://www.urbiforge.com. 2008.
8. O. Michel. Cyberbotics ltd. - webots(tm): Professional mobile robot simulation. *Intl. Journal of Advanced Robotic Systems*, 1(1):39–42, 2004.
9. K. Konolige. Saphira robot control architecture. Technical report, SRI International, 2002.
10. R. Smith. ODE - Open Dynamics Engine, http://www.ode.org. 2007.
11. T. Laue, K. Spiess, and T. Röfer. SimRobot - a general physical robot simulator and its application in RoboCup. In A. Bredenfeld et al., editor, *RoboCup 2005: Robot Soccer World Cup IX*, number 4020 in Lecture Notes in AI, pages 173–183. Springer, 2005.
12. AGEIA PhysX website, http://www.ageia.com/physx/. 2007.
13. Epic games, unreal engine, http://www.epicgames.com. 2007.
14. S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSim: a robot simulator for research and education. In *Proc. of the 2007 IEEE Intl. Conf. on Robotics and Automation (ICRA)*, 2007.
15. S.C. Ntafos. A comparison of some structural testing strategies. *IEEE Trans. Softw. Eng.*, 14(6):868–874, 1988.
16. S. P. Ng, T. Murnane, K. Reed, D. Grant, and T. Y. Chen. A preliminary survey on software testing practices in Australia. In *Proc. Australian Softw. Eng. Conf. (ASWEC'04)*, page 116, Washington, DC, USA, 2004. IEEE Computer Society.
17. J. Figueiredo, N. Lau, and A. Pereira. Multi-agent debugging and monitoring framework. In *First Proc. IFAC Workshop on Multivehicle Systems (MVS'06)*, Brazil, 2006.
18. S. Petters, D. Thomas, and O. v. Stryk. RoboFrame - a modular software framework for lightweight autonomous robots. In *Proc. Workshop on Measures and Procedures for the Evaluation of Robot Architectures and Middleware of the 2007 IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, San Diego, CA, USA, Oct. 29 2007.
19. S. Petters and D. Thomas. RoboFrame website, http://www.roboframe.info. 2008.
20. M. Friedmann, K. Petersen, and O. von Stryk. Scalable and adequate simulation for motion and sensors of heterogeneous teams of autonomous mobile robots. In S. Carpin et al., editor, *Proc. 1st Intl. Conf. on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR 2008)*, Lecture Notes in Artificial Intelligence, page to appear, Venice, Italy, November 2008. Springer.
21. M. Friedmann, J. Kiener, S. Petters, H. Sakamoto, D. Thomas, and O. v. Stryk. Versatile, high-quality motions and behavior control of humanoid robots. *International Journal of Humanoid Robotics*, pages accepted, to appear, 2008.