Evaluation and Enhancement of Common Simulation Methods for Robotic Range Sensors

Martin Friedmann, Karen Petersen, and Oskar von Stryk

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

Abstract. Distance sensors are an important class of external sensors used in many autonomous robots. Thus it is of importance to provide proper simulation for these sensors to enable software-in-the-loop testing of a robot's control software. Two different methods for distance calculation are commonly applied for the simulation of such sensors, namely reading back the depth buffer from 3D renderings and the calculation of ray-object-intersections. Many simulators impose restrictions on either method, none of the widely used robot simulators reviewed in this paper currently considers material dependent simulation of the distances measured.

In this paper extended versions of both methods are presented which provide additional information on the object perceived by distance sensors. Several methods for incorporating distance- and object-information into a complete distance-sensor simulation-module are discussed. Implementations of either method are compared for their performance depending on the sensor resolution on different computer systems. These measurements show, that the break even of the methods strongly depends on the hardware, thus stressing the importance of providing either method in a robot simulation in a transparent way in order to obtain optimal performance of the simulation.

1 Introduction

Laser range finders and other distance sensors like time-of-flight (ToF) cameras or ultrasound sensors are important for many fundamental capabilities of mobile, autonomous robots like mapping or obstacle avoidance. To support development of software for mobile autonomous robots, many existing simulators provide simulation of distance sensors.

Two major approaches are usually applied to simulate the distances measured by the sensor: calculating ray-object intersections using ray-casting or rendering the scene from the sensor's point of view and subsequently reading back the depth buffer. As shown in [1, 2] either method has specific advantages and drawbacks, depending on the geometry and resolution of the simulated device. Nevertheless many existing robot simulators are limited to only one of these methods, or impose limitations on the devices simulated with either method. Even though the material of the objects perceived by a distance sensor has significant impact on the error produced by the sensor (e. g. [3]) none of the investigated simulators provides material dependent error models for the sensors.

In this paper both commonly used methods are compared according their performance. Further on it is discussed how either method can be extended to provide information on the object perceived by the sensor.

The remainder of this paper is structured as follows: In Section 2 an overview of the simulation capabilities for distance sensors found in current simulators is given. In Section 3 the authors' implementations of both general methods including the aforementioned enhancements are presented. After this, simulators using these implementations and extensive measurements of the performance of the implementations are presented and discussed in section 4. The paper closes with concluding remarks and an outlook in Section 5.

2 Simulation of Distance Sensors in Robot Simulators

Many simulators used throughout the autonomous robots community provide simulation for distance sensors. In this section the respective simulation capabilities are discussed and summarized in Table 1.

Gazebo [4], the 3D dynamics simulation developed as part of the Player-Stage-project, is based on the Open-Dynamics-Engine (ODE) for physics simulation. The collision detection mechanisms provided by ODE are used to calculate ray-object intersections which are used to simulate distance sensors.

Microsoft Robotics Studio [5] provides a simulation for mobile robots, including laser range finders. It is not known, though, how this simulation of distance sensors is realized in detail.

Simbad [6, 7] is a 3D robots simulation implemented in Java. Simulation of distance sensors is based on the picking functions provided by the Java3D API.

SimRobot [8, 9] is a general 3D robot simulation based on ODE. Two methods for the simulation of distance sensors are provided: readings from the depth-buffer and calculation of ray-object intersections based on the ODE collision detection. The later, though, is limited to single rays.

Stage [10] is a 2D simulation for mobile robots which has been developed as part of the Player-Stage-Project. Simulation of distance sensors is only available for measurements in the 2D plane. It is based on calculating intersections between rays and the 2D scene representation. Version 3 of the simulation introduced a 3D representation for the scene [11]. In this version, distance sensors can be simulated by ray-intersection in 3D space or by evaluating the depth-buffer.

USARSim [12, 13] is a multi-robot-simulation based on the game engine of Unreal Tournament. The simulation is realized as a set of scripts executed by the game-engine as well as game levels representing the simulated scenarios. As the scripts only are allowed limited access to the internals of the game engine, simulation of distance-sensors can only be done by intersecting individual rays with the scene, but not by evaluating the depth-buffer.

Webots [14, 15] is a commercially available simulation for mobile robots. Simulation of range finders is provided by evaluating the depth-buffer of OpenGL based 3D renderings. Besides the standard pinhole projection a method for the simulation of spherical projections based on multiple rendering passes is provided. Other distance sensors can be simulated by calculation of ray-object intersections. This method is limited to single rays and cones of equally distributed rays (defined by aperture angle and number of rays). The readings of the simulated sensors can be post-processed to simulate different output characteristics including noise.

Table 1. Overview of simulation methods for distance sensors in existing simulators.

Simulator	Simulation method	
	Raycasting	Depthbuffer
Gazebo	yes	no
MSRS	yes (method unknown)	
Simbad	yes	no
SimRobot	yes (limited to single rays)	yes
Stage 2.1.1	yes (2D simulation, limited to plane)	no
Stage 3.2	yes (limited 3D support due to model of environment)	yes
USARSim	yes (only single ray, fixed cones and 2D-sweeps)	yes
Webots	yes (only single ray and predefined distributions)	yes

Discussion. Many of the discussed simulators impose restrictions on the directions of rays in the ray-intersection based simulated method. Not all simulators provide simulation of distance sensors based on the depth buffer, only few of those provide simulation beyond the pinhole model. These circumstances complicate a transparent exchange of the simulation method, if it is possible at all.

Some of the investigated simulators allow to modify the measurements of the simulated sensors to reproduce sensor specific output characteristics. Nevertheless none of the simulators considers the impact of different materials on the measurements. This limitation is most likely caused by the fact, that the methods used to determine the distances only provide information on the distance of objects, but not on their type.

3 Enhanced Simulation of Distance Sensors

Simulation of a distance sensor is a two step process. The first mandatory step consists of the calculation of the first intersection of each ray emitted by the sensor. Besides information on the length of the ray, this step may produce additional information on the kind of object hit by the ray. The data produced by this step can be considered as readings from a perfect sensor.

In a second, optional, step, the ray length is post-processed. Independent of the object hit by a ray, effects like noise or other sensor specific output characteristics can be simulated as a function of the length calculated before. If additional information on the object is present, further object or material specific processing of the sensor output can be calculated.

This leads to three distinct levels of accuracy for the sensor simulation:
1. Simulation of a *perfect* sensor,
2. Simulation of a sensor with specific output characteristics,
3. Simulation of a sensor considering material dependent behavior.

The highest level of accuracy can only be simulated, if information on the objects hit by the ray are provided by the first step.

In the following subsections two methods used in many simulators for the calculation of the ray length are discussed. Further on it is investigated, how these methods can be extended to provide further information on the objects hit by the rays. Both methods have been implemented as part of the Multi-Robot-Simulation-Framework (`MuRoSimF`) by the authors of this paper. Later on the methods will be compared for their respective performance concerning the different levels of accuracy.

`MuRoSimF` provides methods for the simulation of wheeled (e. g. [1]) and legged (e. g. [16]) locomotion as well as for different sensors of autonomous mobile robots. It allows an unrestricted combination of these simulation methods to create simulations with different levels of accuracy and abstraction.

3.1 Depth Buffer Based Method for Ray Simulation

This method is based on rendering the simulated scene from the sensor's point of view using OpenGL[1]. After rendering the scene, the depth-buffer used for hidden surface removal by OpenGL is read back to process memory for calculation of the length of each view-ray. Inverting the steps of the perspective projection manipulations of the depth coordinate described in [17] leads to

$$z = -\frac{f \cdot n}{\tilde{z} \cdot (f - n) - f} \tag{1}$$

with z being the orthogonal distance of a point with depth coordinate $\tilde{z}$ from the viewing plane. The parameters f and n denote the near and far clipping plane defining the viewing volume used for the rendering process (see Figure 1 for details). To calculate the length l of a ray, the direction α of this ray must be taken into account leading to

$$l = \frac{z}{\cos \alpha}. \tag{2}$$

If additional information on the object hit by a ray is desired, this information can be encoded into the image during rendering. To do this, an id describing the

[1] Note that the same method can be used using other 3D rendering systems, as long as they provide access to the depth buffer for the CPU.

surface of each object is transformed into an RGB-triplet which is used as color during rendering. Rendering itself is done without lighting, shading or blending effects, so that the color's RGB-values are preserved during the rasterization process. To obtain object information for the rays, the frame-buffer is read back to process memory and evaluated for each pixel leading to surface information for the object hit by each of the rays.

As discussed in [1, 2] the view-rays have an equal distribution in the image plane. This contradicts the equal angular distribution usually found in laser range finders. Different strategies to cope with this problem are discussed in Section 3.3.

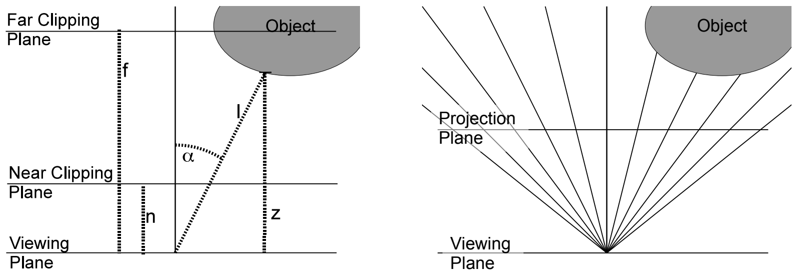

Fig. 1. Left: Length of a ray. Right: distribution of rays (adapted from [1]).

3.2 Ray Intersection Based Method for Ray Simulation

The second method for calculation of the rays is based on the explicit calculation of ray-object intersections. A sensor can be defined by an arbitrary number of rays, each with an individual direction. No limits are imposed on the aperture angle or the distribution of the rays.

To avoid intersecting each ray of the sensor with each object of the scene, the scene is decomposed into compounds containing the objects of the scene. For each compound and each individual object an axis-aligned bounding-box is calculated (and if the object is mobile, re-calculated for each time-step of the simulation). Ray-intersection is based on this hierarchy of bounding boxes, thus allowing to discard many objects before calculating time consuming tests on object-geometry-level. The whole hierarchy is provided by the collision-detection module which is provided as part of the dynamics-simulation of `MuRoSimF` [16]. Simulation of all distance sensors uses the same hierarchy, thus avoiding multiple re-calculations of the hierarchy.

As the bounding-box-hierarchy is aligned to the world-coordinate systems, the rays have to be transformed to the same coordinate system before intersection tests can be performed. To further speed up the simulation, rays are aggregated to sets of rays, which fit into a bounding box, so that further objects of the scene can be discarded for some of the sets.

3.3 Simulation of Different Distance Sensors

After calculation of the ray length (and optionally the material hit by the individual rays) additional post-processing of the data may be applied to simulate different kinds of sensors. In this section several of the possible steps are discussed.

Error modeling. Several possibilities for modeling the sensor's errors exist and can be applied after the ray-calculation on single-ray whole-scan level.

- On the level of single rays, an error can be calculated as a function of the length of the ray. This function may contain a randomized component to simulate noise. This function can be chosen individually for each simulated sensor.
- If additional information on the material hit by the individual ray is present, the function for error modeling can be chosen according to the material for each ray.
- On the level of the whole scan, effects like crosstalk between neighboring rays can be considered.

Handling errors of ray-direction. When using the depth-buffer based methods, the directions of the calculated rays are limited to those directions defined by the viewing geometry. These directions are not necessarily the same as the directions of the rays of simulated sensor. Especially laser scanners often have an even-angular distribution of the rays. Several options are available to cope with this problem:

- In case the error is small (enough for the desired application), the problem may be ignored.
- Interpolation can be applied to approximate the length of the rays of the simulated sensor.
- To reduce the angular error, a higher resolution for rendering can be chosen.
- If the aperture angle of the sensor is high, it may be necessary to use more than one rendering pass using different viewing directions to reduce the error.

A different kind of error is caused by the motion of the sensor during measurement. If the method used for ray simulation can be calculated efficiently for parts of a scan (optimally for single rays), it can be interleaved with the motion simulation. By this, distortions of the scan caused by the motion of the robot can be simulated.

Aggregation of rays. For distance sensors which produce one single value, e. g. ultrasound or infrared distance sensors, it nevertheless may be of interest to calculate intersections of several rays with the scene. This is especially interesting if the sensor has a significant cone characteristic, as it is the case with many ultrasound-sensors. The easiest way to simulate a sensor producing a single value is to calculate the minimum distance of all calculated rays. If further information on the sensor's characteristic is known, the directions of the rays may be used as additional argument for the calculation of the sensor output value.

4 Results

4.1 Applications

Both simulation methods have been implemented as part of the Multi-Robot-Simulation-Framework (`MuRoSimF`). With a robot simulation based on `MuRoSimF` it is possible to provide different simulation methods for the same purpose (like simulation of a robot's motion or a distance sensor). These methods can be exchanged transparently and independently of the other elements of the simulation and be adapted well to very different purposes of robot simulation [1, 2].

Several simulations for robots equipped with laser-scanners have been created using `MuRoSimF`. These include a simulation of simplified virtual robots for educational purposes and a simulation of a search-and-rescue vehicle (see Figure 2).

The simulation of the vehicle is based on a real robot used for urban search and rescue [18] which is equipped with two laser-scanners, among other sensors. To allow for a transparent integration of the sensor simulation with the robot-control-software, a simulation specific module was added which emulates the SCIP (see [19]) protocol used by the sensors of the real robot. For each laser-scanner of each simulated robot it is possible to choose the simulation method, resolution and rate independently in the configuration of the simulation. Besides selecting a resolution appropriate for a specific simulated experiment, it is possible to select the method providing the best performance on the respective computer. This selection can be made by running the simulation twice for a short time with either method and comparing the performance, before starting the main experiment. The simulation has been used for the development of sensing and mapping algorithms as well as for high-level behaviors of the vehicle which could be transferred successfully to the real vehicle.

4.2 Performance

The performance of the simulation methods has been measured with the search-and-rescue vehicle simulation situating the robot in a typical maze-like environment (see Figure 3). For the measurements only one laser-scanner of the robot was activated. In this scenario the time required to calculate one sweep of the laser scanner has been measured for several resolutions using both methods with and without determination of the material (see Table 3). For each of these setups a new run of the simulation was started and 300 the computation time for the sensor simulation was measured for 300 times. During all measurements the aperture angle of the sensor was 90 degrees (with the obvious exception of the 1x1-resolution). Measurements were taken on three different laptop-computers (see Table 2).

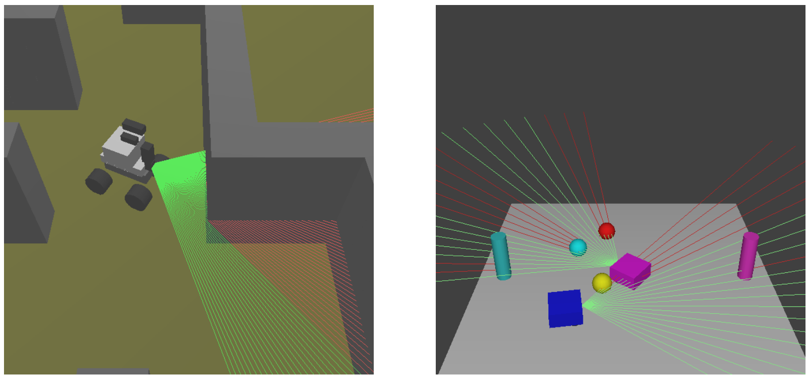

Fig. 2. Left: Simulation of a search-and-rescue robot. Right: Educational simulation of two virtual robots. Note that rays are rendered green as long as they did not touch an obstacle and red afterwards.

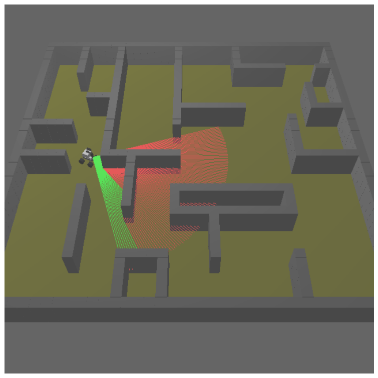

Fig. 3. The scenario used for performance evaluation.

Table 2. Computers used in experiments.

	Computer		
	A	B	C
Operating System	MacOS X 10.6.3 (32 Bit)	Ubuntu Lucid Lynx (64 Bit)	Windows Vista Business SP1 (32 Bit)
CPU	Intel Core 2 Duo T8300	Intel Core 2 Duo P9500	Intel Core 2 Duo P8600
Clock	2.4 GHz	2.53 GHz	2.4 GHz
RAM	2 GByte	4 GByte	4 GByte
Graphics Hardware	Intel GMA X3100 (chipset)	NVIDIA Quadro NVS 160M	NVIDIA Quadro NVS 160M

Table 3. Average computation time in ms for one scan at different resolutions for different simulation modes (R = ray-intersection, ZB = z-buffer, ...wM = ...with material determination).

	Computer A				Computer B				Computer C			
Resolution	R	RwM	ZB	ZBwM	R	RwM	ZB	ZBwM	R	RwM	ZB	ZBwM
1x1	0.021	0.023	6.3	6.33	0.048	0.05	3.87	23.2	0.03	0.033	1.57	1.64
1x3	0.017	0.018	6.2	6.28	0.041	0.041	4.17	23.3	0.027	0.031	1.52	1.61
1x10	0.066	0.068	6.14	6.17	0.19	0.193	3.89	4.19	0.116	0.127	1.57	1.58
1x30	0.165	0.165	6.22	6.27	0.468	0.569	3.96	3.98	0.256	0.272	1.52	1.6
1x100	0.467	0.474	6.18	6.24	1.35	1.31	4.11	3.88	0.758	0.782	1.55	1.58
1x300	1.33	1.35	6.12	6.27	4.03	4.03	3.88	3.94	2.17	2.19	1.5	1.7
1x1000	4.31	4.37	6.12	6.35	12.0	11.2	4.15	4.13	8.15	7.06	1.53	1.64
35x35	5.52	5.59	6.3	7.11	13.7	12.6	3.98	4.1	8.07	8.64	1.62	1.78
50x50	12.1	12.1	6.78	7.37	19.7	19.0	4.19	4.3	20.8	21.2	1.6	1.81
70x70	23.6	24.0	6.9	7.54	31.6	31.6	4.42	4.62	47.8	47.6	1.74	1.99
100x100	47.6	48.2	7.05	7.67	52.8	52.2	4.98	5.51	68.3	67.7	2.11	2.47
140x140	93.8	94.4	7.4	8.33	91.7	92.2	5.86	6.67	126	129	2.5	2.91
200x200	192	195	7.91	9.53	188	190	7.98	9.65	259	263	3.16	4.24

The main results from these measurements are:

- The ray-intersection-methods have smaller computation times at small resolutions, while the depth-buffer-methods have a better performance at high resolutions.
- Additional determination of the material hit by the respective rays has nearly no impact on the performance of the ray-intersection based methods. On the other hand, there is a visible impact at higher resolutions when using the depth-buffer-methods. Parts of this impact can be explained by the additional effort required to read back the frame-buffer (see discussion below). In some cases a method with additional determination of the material took less time than the method without the addition. This is most likely caused by disturbances due to background activities of the respective computer.
- The computation time of the ray-intersection methods scales quite well with the number of rays. Compared to this, the depth-buffer-methods start at a higher basic computation time, which does not change that much with rising resolutions. This can be explained by the fact, that independently of the actual resolution always the whole scene has to be rendered from the sensor's point of view.
- The break even of the computation time of the different methods is highly dependent on the current system.
- Two measurements on system B led to extreme average computation times (ZBwM at 1x1 and 1x3). Further investigations showed, that this effect appeared randomly for several resolutions on system B but not on the other systems. It is not clear what caused this effect.

Another result which came clear during these experiments is the strong impact of reading-back the frame- and depth-buffer on the overall performance of the depth-buffer based methods. Measurements of the time required to read back these buffers are given in Tab. 4. Interestingly reading back the frame buffer takes less time than reading the depth-buffer. To further investigate this effect, the reading order was inverted in an experiment on system A, leading to an exchange of the reading times.

Table 4. Time in ms required for read back of depth and framebuffer

	Computer A		Computer B		Computer C	
Resolution	depth-buffer	frame-buffer	depth-buffer	frame-buffer	depth-buffer	frame-buffer
1x1	1.24	0.075	0.076	0.036	0.312	0.086
1x3	1.21	0.075	0.078	0.036	0.313	0.093
1x10	1.18	0.077	0.169	0.068	0.246	0.088
1x30	1.20	0.079	0.116	0.056	0.289	0.087
1x100	1.21	0.090	0.130	0.060	0.271	0.089
1x300	1.17	0.111	0.135	0.056	0.311	0.096
1x1000	1.08	0.182	0.162	0.085	0.296	0.104
35x35	1.30	0.785	0.166	0.080	0.353	0.186
50x50	1.71	0.621	0.226	0.100	0.288	0.200
70x70	1.76	0.640	0.295	0.126	0.308	0.247
100x100	1.78	0.625	0.491	0.172	0.504	0.298
140x140	1.90	0.802	0.775	0.272	0.608	0.372
200x200	2.02	1.220	1.470	0.540	0.887	0.767

4.3 Discussion

The performance results from the previous section suggest, that it is highly desirable to provide both methods for ray simulation. Depending on the computer the simulation is executed on, this will allow to choose the better performing method.

If the methods are to be truly exchangeable within a simulation, it is necessary, that equal interfaces to the provided data are available. Any post-processing for sensor error simulation can be done independently of the calculation of the rays.

Independent of the performance other considerations can be made to choose either one of the methods: The depth-buffer based method has a fixed distribution of the view rays. Thus it can perform best, if either this distribution matches the simulated sensor (e. g. ToF-cameras), or is of no interest (e. g. if the rays will be aggregated later on for ultrasound cones). For sensors with different distributions of the rays (e. g. 2D or 3D laser scanners), additional processing steps are required as discussed in Section 3.3. Due to the high basic computation time of the method, it cannot be used to simulate single rays, as it is necessary when considering the motion of the sensor during measurement.

The ray-intersection based method imposes no limitations on the distribution of the rays. Further on the method's computation time scales well with the number of rays. It is feasible to even simulate single rays at a high rate, thus allowing the simulation of effects caused by the motion of the robot while sensing.

Both methods can be used for the additional calculation of the material hit by the individual rays. Only at higher resolutions, there is a visible impact on the computation time of the depth-buffer based method. Nevertheless the method performs much better than the ray-based method at these resolutions.

5 Conclusions and Outlook

Two different methods for the calculation of the distances measured by distance sensors were discussed in this paper, namely reading back the depth buffer from 3D renderings and the calculation of ray-object-intersections. Either method can be extended to determine the material hit by individual rays of the sensor. This allows for the simulation of sensor errors on different levels of accuracy which is not possible in current general purpose robot simulators commonly used in the autonomous robotics community.

Both methods have been implemented as part of the Multi-Robot-Simulation-Framework (`MuRoSimF`) and can be exchanged transparently within simulations. Extensive measurements of the performance of either method on different computer systems were made. The results of these measurements suggest that it is very useful to provide either method fully exchangeable within the simulation to allow the adaption of the simulation to different computers to provide optimal performance. Further on different advantages and drawbacks of each method were discussed to give an aid in choosing a method depending on the requirements of the individual simulation.

With the existing implementations of both methods the complete infrastructure for the simulation of distance sensors is provided. As a next step the authors are planning to determine concrete functions to model errors of real world sensors to be evaluated with the robots of the authors' group, thus providing a validated simulation of these devices. Identification of parameterized error models can be based on comparing the outputs of real and simulated sensors in experiments and applying optimization methods like least square fitting.

The good scalability of the ray intersection based simulation method opens up another opportunity: By simulating few or even single rays of a whole sensor sweep for every time step of the simulation of the robot's motion, effects caused by the motion of the sensor can be simulated.

Acknowledgment. Parts of this research have been supported by the German Research Foundation (DFG) within the Research Training Group 1362 "Cooperative, adaptive and responsive monitoring in mixed mode environments".

References

1. M. Friedmann, K. Petersen, and O. von Stryk. Simulation of Multi-Robot Teams with Flexible Level of Detail. In S. Carpin et al., editor, *Proc. of the 1st Intl. Conf.*

on Simulation, Modeling and Programming for Autonomous Robots (SIMPAR), LNAI, pages 29–40, Venice, Italy, Nov 4-6 2008. Springer.

2. M. Friedmann. *Simulation of Autonomous Robot Teams With Adaptable Levels of Abstraction*. PhD thesis, Technische Universität Darmstadt, Nov. 30 2009.
3. H. Kawata, A. Ohya, S. Yuta, W. Santosh, and T. Mori. Development of ultra-small lightweight optical range sensor system. In *Proc. of the 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2005.
4. N. Koenig and A. Howard. Design and Use Paradigms of Gazebo, an Open-Source Multi-Robot Simulator. In *Proc. of the 2004 IEEE/RSJ Intl. Conf. on Intelligent Robots and Systems (IROS)*, 2004.
5. J. Jackson. Microsoft Robotic Studio: A Technical Introduction. *Robotics and Automation Magazine*, 14(4):82–87, 2007.
6. L. Hugues and N. Bredeche. Simbad: an Autonomous Robot Simulation Package for Education and Research. In *Proc. of the 2006 Intl. Conf. on the Simulation of Adaptive Behavior (SAB)*, Rome, Italy, 2006.
7. Simbad website: http://simbad.sourceforge.net/, 2009.
8. T. Laue, K. Spiess, and T. Röfer. SimRobot - A General Physical Robot Simulator and Its Application in RoboCup. In A. Bredenfeld, A. Jacoff, I. Noda, and Y. Takahashi, editors, *RoboCup 2005: Robot Soccer World Cup IX*, number 4020 in Lecture Notes in AI, pages 173–183. Springer, 2006.
9. T. Röfer, T. Laue, A. Burchardt, E. Damrose, K. Gillmann, C. Graf, T. J. de Haas, A. Härtl, A. Rieskamp, A. Schreck, and J.-H. Worch. B-Human Team Report and Code Release 2008. Technical report, 2008.
10. B. P. Gerkey, R. T. Vaughan, and A. Howard. The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems. In *Proc. of the 2003 Intl. Conf. on Advanced Robotics (ICAR)*, pages 317–323, Coimbra, Portugal, 30 June - 3 July 2003.
11. R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2:189–208, 2008.
12. 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.
13. J. Wang and S. Balakirsky. UARSSim V3.1.3 - A Game-based Simulation of mobile robots. Technical report, NIST, 2008.
14. O. Michel. Cyberbotics Ltd. - Webots(TM): Professional Mobile Robot Simulation. *Intl. Journal of Advanced Robotic Systems*, 1(1):39–42, 2004.
15. Webots user guide 6.2.3. Technical report, Cyberbotics, Ltd., 2010.
16. M. Friedmann, K. Petersen, and O. von Stryk. Adequate Motion Simulation and Collision Detection for Soccer Playing Humanoid Robots. *Robotics and Autonomous Systems*, 57:786–795, 2009.
17. M. Segal and K. Akeley. The OpenGL Graphics System: A Specification (Version 2.0 - October 22, 2004). Technical report, Silicon Graphics, Inc., 2004.
18. M. Andriluka, S. Kohlbrecher, J. Meyer, K. Petersen, P. Schnitzspan, and O. von Stryk. RoboCupRescue 2010 - Robot League Team Hector Darmstadt (Germany). Technical report, Technische Universität Darmstadt, 2010.
19. URG Series Communication Protocol Specification SCIP-Version 2.0. Technical report, Hokuyo Automatic co., Ltd., 2006.