Device Level Simulation of Kurt3D Rescue Robots

Sven Albrecht, Joachim Hertzberg, Kai Lingemann, Andreas Nüchter, Jochen Sprickerhof, Stefan Stiene
University of Osnabrück
Institute for Computer Science
Knowledge-Based Systems Research Group
Albrechtstraße 28
D-49069 Osnabrück, Germany
nuechter@informatik.uni-osnabrueck.de
http://kos.informatik.uni-osnabrueck.de/download/UOSSim/index.html

***Abstract*— USARSIM is a worldwide used robot simulator deployed in Urban Search and Rescue (USAR) and in the context of the RoboCup Rescue Real Robot contest. This paper describes the USARSIM simulator for KURT2 and Kurt3D robot platforms, which we are using in both education and research. As it simulates on the device level, a seamless integration of real robot control software with the simulations becomes possible. We evaluate the performance for simulating laser range scans and the camera system. In addition, we show a simulation of the rescue robots.**

I. Introduction

Mobile robotics is a complex area of scientific research and education dealing with advanced technologies. Knowledge and experiences of developing intelligent systems include the domains electronics, mechatronics, computer hardware and software. Furthermore mobile robotics projects are tied with large investments. Realistic simulations and fast prototyping for developing mobile systems help to reduce the amount of time and to minimize the costs for hardware developments. In addition, simulations offer the possibility to concentrate faster on the interesting aspects of developing algorithms. Scientific education and research benefit from realistic, technically mature, and well-engineered simulators.

State of the art computer games are cost effective, due to the development for the mass consumer market. As an available free prerequisite, many students are already familiar with computer games and are extraordinarily motivated to go into more detail. Ego shooters simulate agents in a 3D environments and contain a physics simulation [8].

Application of the simulator USARSIM is the area of rescue robotics. Software of rescue robots covers artificial intelligence, knowledge representation and fast control algorithm design. RoboCup is a test and demonstration scenario for evaluating robots and their software.

The paper is organized as follows: First we introduce RoboCup Rescue and USARSIM, followed by a description of how to simulate environments and the Kurt2 robot platform, including a presentation of our USARSIM system architecture. Simulation performance and results conclude.

II. RoboCup, RoboCup Rescue and USARSIM

RoboCup is an international joint project to promote AI, robotics and related fields. It is an attempt to foster AI and intelligent robotics research by providing standard problems where a wide range of technologies can be integrated and examined. Though not as well-known as the RoboCup Soccer leagues, the Rescue league with its serious real-life background got more and more attention lately. The idea is to develop mobile robots that are able to operate in earthquake, fire, explosive and chemical disaster areas, helping human rescue workers to do their jobs. A fundamental task for rescue robots is to find and report injured persons. To this end, they need to explore and map the disaster site and inspect potential victims and suspicious objects. Current real deployed rescue robots have only limited usage and are mainly designed for searching for victims and paths through rubble that would be quicker to excavate, for structural inspection and for detection of hazardous material [3]. These robots are designed to go a bit deeper than traditional search equipment, i.e, cameras mounted on poles [3]. The RoboCup Rescue Contest aims at evaluating new rescue robot technology to speed up the development of working rescue and exploration systems.

In RoboCup Rescue, robots compete in finding as many "victims" (manikins) as possible, within a limited time, in a given, previously unknown arena, and reporting their life signs,

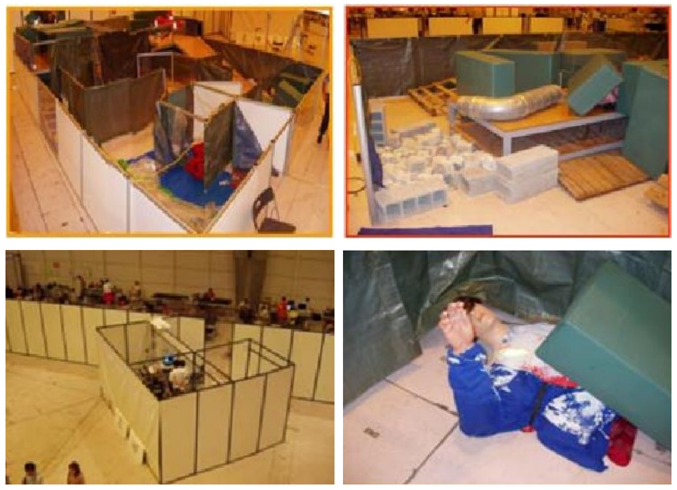


Fig. 1. Rescue arenas at RoboCup 2004, Lisbon. Top row: Orange and red area. Bottom left: Operator station. Bottom right: Example of a victim in a yellow area.

situations, and positions in a map of the arena, which has to be generated during exploration. The idea is that, in a real-life application, this map would help humans to decide where to send rescue parties. The arena consists of three subareas (yellow, orange, red) that differ in the degree of destruction, and therefore in the difficulty of traversal. In the "earthquake phase" between competition runs, the areas get completely rearranged, including the distribution of the victims. Fig. 1 shows some examples.

The robots in RoboCup Rescue are remotely controlled or surveyed by one or more operators. The operator has no direct view of the arena, only transmitted robot sensor data may be used for control. The degree of autonomy or telecontrol in the robots is at the team's discretion.

Scoring is based on an evaluation function that is modified between the competitions. This function incorporates the number of operators (the fewer the better), the map quality, the quality of the victim localization, the acquired information about the victim state, situation and tag, the degree of difficulty of the area, but also penalizes area bumping and victim bumping.

USARSIM is a simulation of robots and scenarios for disaster and rescue robotics. It was developed by M. Lewis and J. Wang [8] to match the physical test scenarios of the American National Institute of Standards (NIST) [1], [2]. The focus of the development was the evaluation of man–robot interaction as well as research of cooperative robots [6], [7].

Sophisticated robot simulation in USARSIM is based on a *game engine* that stems from the computer game Unreal Tournament 2003 or 2004. Due to using a game engine, the simulator shows the excellent graphics and physic simulation of a commercial software product. Since games are produced for a mass market, the costs are low: About $15 for a license.

Unreal Tournament is a multiplayer ego shooter for Windows, Linux and MacOS. The graphics is outstanding, as expected from a commercial product. The Unreal-environment includes a script language that offers developers the possibility to create objects and to control their behavior. The Unreal-editor that comes with the game and the open source program Blender were used to develop environments and models of robot platforms.

Multiplayer ego shooters realize a client server architecture where every player is a client, connecting to the game server. The fast rendering of the scene graphic is done by the client. The server coordinates the players and is responsible for their interaction. The communication protocol is proprietary. However, the software Gamebots modifies Unreal Tournament such that agents can be controlled using an open TCP/IP interface. This interface provides sensor information to the agent control program.

Physical simulation in Unreal Tournament is done by the Karma Physics Engine. Karma processes ridged body movements and allows to simulate motors, wheels, springs, hinges and joints. From these base modules, complicated objects are built through compounding. The compound object comply to the physics in simulation.

III. Simulation of Rescue Robots

A. *Environment Simulation*

USARSIM provides Unreal maps for all three RoboCup arenas. Fig. 2 shows a photo and an Unreal rendering of the orange and yellow arena. Using the Unreal editor, arbitrary scenes can be created. Fig. 3 shows a photo of our office corridor and the corresponding Unreal scene.

B. *KURT2*

KURT2 (Fig. 3, right) is a mobile robot platform with a size of 45 cm (length) $\times$ 33 cm (width) $\times$ 26 cm (height) and a weight of 15.6 kg. The core of the robot is a Centrino-1400 MHz with 256 MB RAM running Linux. An embedded 16-Bit CMOS microcontroller is used to control the motor.

The robot is equipped with a 2D laser range finder, a Logitech web cam, including a pan and tilt unit, as well as a

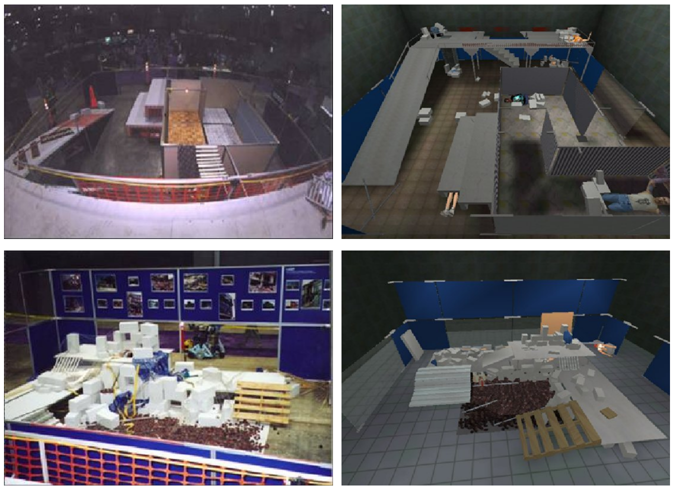


Fig. 2. Real and simulated rescue arenas. Top: Orange arena real and simulated. Bottom: Red arena real and simulated. Taken from [8].

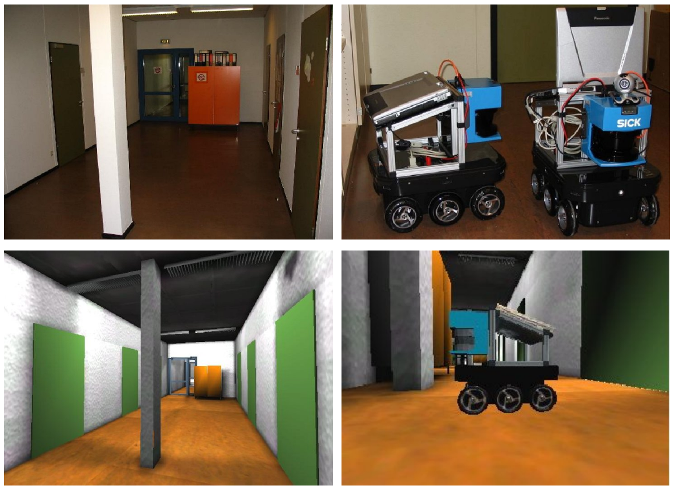


Fig. 3. Top: AVZ building Osnabrück and KURT2 real. Bottom: Building and KURT2 in simulation. More material available at `http://kos.informatik.uos.de/download/UOSSim/index.html`.

one-axis (horizontal) gyro and seven infrared sensors.

Simulation of KURT2 robots: For the simulation of the robot, a model of the hardware as well as a control software are needed. For the model, a mesh of the robot has to be generated, which is done using the Unreal editor and Blender. Fig. 3 (right) shows the real and the simulated robot. As for the control software, we extended the exiting software for the real robots with interfaces to access either the actual hardware or the corresponding components of the simulation software. This way, only small changes to the software were necessary, and any future improvements are beneficial for both applications, real world and simulated. The following code fragment shows an example of retrieving laser range data, either from a real SICK scanner or its simulated counterpart.

```
#ifdef USARSIM
   res = sim_client.SICK_read(fd_RS422,
                                  buf, 255);
#else
   res = read(fd_RS422, buf, 1);
#endif
```

Currently the following components are simulated:

- A motor that drives the robot. Pulse width modulated signals are simulated.
- Odometry determining wheel revolutions in ticks.
- A laser scanner yielding 181 distance values of one slice of the environment in front of the robot.
- A gyro that estimates the current heading of the robot.
- A camera that provides images of the environment.

The camera device drivers are fed from a camera server, that in turn is fed by a snapshot of an Unreal spectator client. Fig. 5 (left) sketches the software structure and the data flow. For fast simulation four computers form a cluster:

1) One computer is needed as Unreal server. The server simulates all robot sensors, except cameras.
2) The cameras are simulated on a second computer, running a small program that captures pictures from an Unreal spectator window.
3) The control loop of the KURT2 robot runs on a third computer, instead of the robot's notebook. Usually, the loop retrieves motor signals with 100 Hz and laser range scans with 75 Hz.
4) The user interface for driving the robot runs again on a separate computer. This computer is connected to the previous one, i.e., to the computer running the robot control loop. There are no direct connections to Unreal.

The right part of Fig. 5 shows the 4-PC simulation of KURT2. Fig. 6 shows the user interface of KURT2. The shown data is transmitted from the control loop of the robot.

C. Kurt3D

Kurt3D (Fig. 4, left) extends the KURT2 robot by a 3D laser range finder, i.e., the SICK 2D scanner is mounted on a tiltable unit, rotating the scanner by its horizontal axis. Furthermore, Kurt3D is equipped with two cameras, mounted on selfmade pan and tiltable units on both sides of the scanner. The robot's height increases to 47 cm, its weight to 22.6 kg.

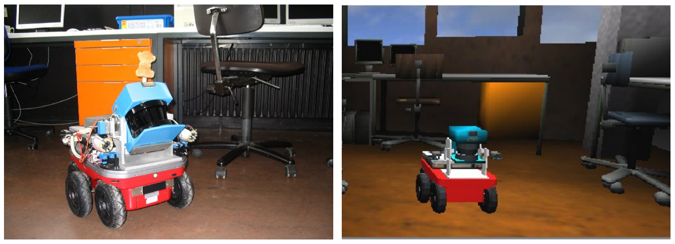


Fig. 4. Left: Real Kurt3D robot. Right: Simulated Kurt3D.

In addition, the control software is extended by a 3D environment mapping system, i.e., 6D SLAM (simultaneous localization and mapping) [4]. This system always yields a precise pose estimate in all six degrees of freedom (x, y, z position; pitch, roll and yaw angle), enabling the robot to create accurate three dimensional maps.

Simulation of Kurt3D robots: The simulation of the Kurt3D robot is done according to section III-B. In the simulation, the scanner is attached to a tilting unit, yielding 3D scans. Thus, the simulation is done in the same way as in reality.

The USARSIM 3D scanner model is not used, due to its unavailability in mobile robotics.

IV. Results

A. Simulation Performance

Our simulation has been tested on a PC cluster, consisting of four 3.0 GHz Pentium-IV computer, running Linux OS. Tab. I shows the performance of the system. To achieve a seamless integration of simulation and real robot control software we rewrote Kurt's control software to handle all devices in a non-blocking fashion. The control loop runs as fast as possible and whenever new device data is present, the data gets processed. Standard Linux device drivers are used to buffer and hold back data.

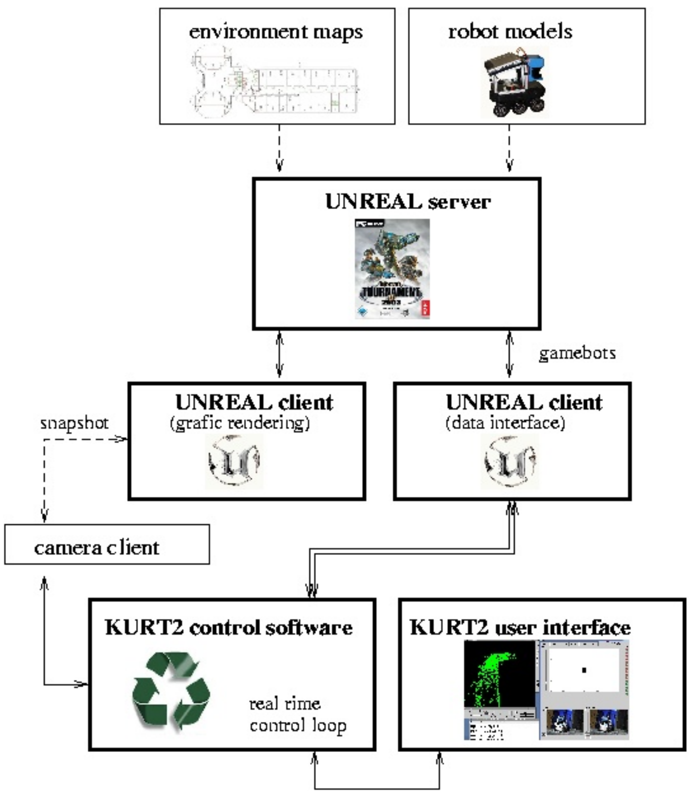
The simulated 3D scanner needs 61 sec for acquiring a 3D scan of 181 $\times$ 120 3D data points. This result is due to the fact that setting the servo motor values in Unreal is not instantaneous.

B. Simulated 3D mapping

We have tested our 3D environment mapping system [4] in the simulated AVZ building and in the USARSIM yellow arena as provided by NIST. Fig 7 shows images of the arena vs. simulated depth data. Fig. 9. presents the result of an octree representation (top right) and a marching cubes algorithm (bottom right) that extracts 3D meshes reliably from the data points and vectorizes the data.

V. Conclusions

The presented work introduces a simulation of KURT2 robots using USARSIM. The simulation is based on the computer game Unreal Tournament 2004. Excellent 3D graphic



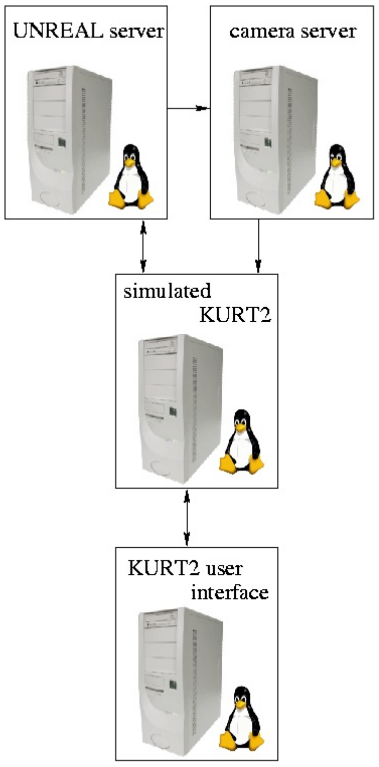


Fig. 5. Left: The software architecture for simulating KURT2 robots. The arrows show the data flow, lines represent TCP/IP connections, double lines are created, when programs are linked and dashed lines are generated, when data files are read. Right: Four computers are necessary for a fast KURT2 simulation. The lines represent TCP/IP connections and the arrows the data flow.

TABLE I

COMPUTING TIME OF THE DEVICES (PENTIUM-IV-3000) USING DIFFERENT SIMULATED ROBOCUP RESCUE ARENAS. FOR COMPARISON: THE REAL KURT2 YIELDS ENCODER TICKS WITH A FREQUENCY OF 100 HZ, LASER SCANS WITH 75 HZ AND CAMERA IMAGES WITH ABOUT 10 FPS.

Kurt3D device	**computing time (4 PC cluster)**	**computing time (single computer)**
scanner (181 values)	50ms	200ms
gyro (INU sensor)	50ms	200ms
encoder sensor	50ms	100ms
camera	400ms	400ms

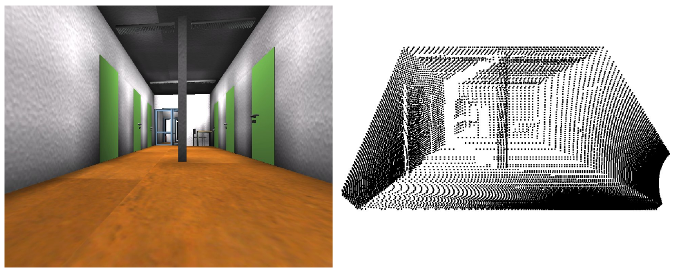


Fig. 7. Left: Rendered images from the AVZ building. Right: Simulated 3D scan.

and physical simulation make a seamless integration in already existing robot control architectures possible. We have strictly followed the principle for simulating device drivers, resulting in the need for four standard PCs for simulation, where two computers are used for the robot and interface software, respectively. We provide a truly realistic simulation environment for beta-testing our real robots, and experiment with new control software or simulations of sensors that you don't have yet physically. This is the goal towards which we have worked, given that we keep participating in the Real Robot league (Kurt3D in 2004 [4]; Deutschland1 in 2005 [5]).

Unfortunately, these results cannot be used in the RoboCup Rescue Virtual Robots league. We started to join USARSIM community with our original Kurt3D software, into which the simulator is integrated. Kurt3D's software development has lasted for 6 years, and is jointly developed with the Fraunhofer Institute AIS and with RTS, University of Hannover. The software underlies regulations and cannot be made available to the public, as it is demanded in the current rules. However, parts of it, namely the complete Unreal parts as well as the interface to our robot are available on our website.

Moreover, we believe that the Rescue Virtual Robots league should focus on the seamless integration of real robot control

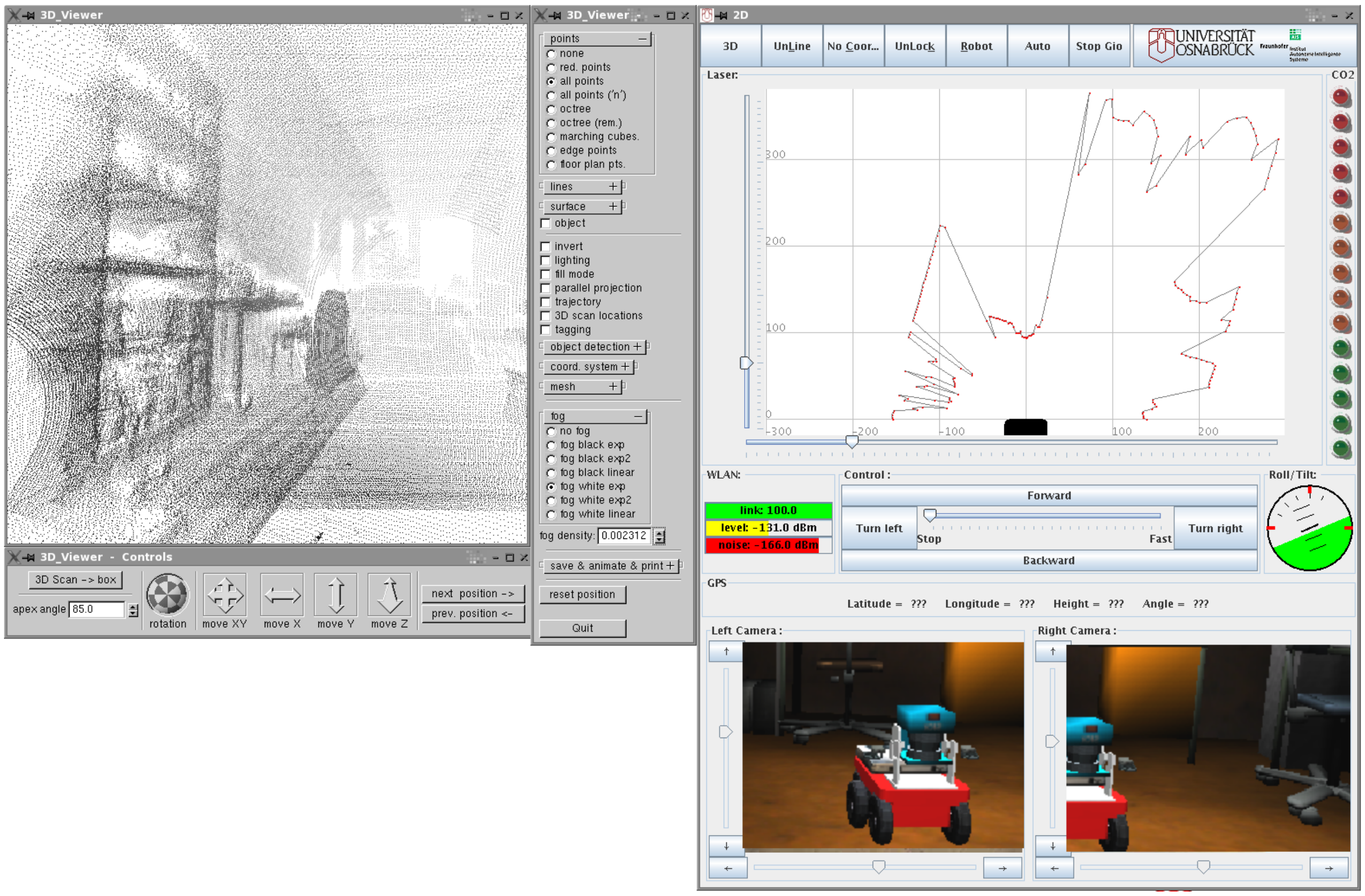


Fig. 6. The user interface for driving Kurt3D robots. The laser range data and the camera data originates from simulation.

software with the simulator. It does not make sense to develop programs just for driving a robot in an Unreal environment, without have the link to real robots.

Needless to say, a lot of work remains to be done. In future work, we plan

- to integrate simulations of the RTS Scandrive [5] and the FLIR infrared camera into our system,
- to enhance realism of the laser scanner. So called salt and pepper noise will be added to the simulation to generate jump edge outliers as in real scans, and
- to improve the simulation of the gyro in order to yield a drift similar to the real sensor.

In addition, we plan to use USARSIM in projects dealing with service robotics.

Acknowledgment

We would like to thank Jijun Wang and Mike Lewis for discussing simulator details. Many thanks to Christian Taubitz and Marvin Drogies for helping modeling the Osnabrück office environment. The marching cubes algorithm was implemented by Thomas Wiemann. Furthermore, we thank Kai Pervölz and Hartmut Surmann for preceeding joint research and Oliver Wulf and Bernardo Wagner for the work in the Deutschland1 team.

References

[1] A. Jacoff, E. Messina, and J. Evans. Experiences in deploying test arenas for autonomous mobile robots. In *Proceedings of the 2001 Performance Metrics for Intelligent Systems (PerMIS) Workshop, in association with IEEE CCA and ISIC*, Mexico City, Mexico, 2001.

[2] A. Jacoff, E. Messina, B. A. Weiss, S. Tadokoro, and Y. Nagakawa. Test arenas and performance metrics for urban search and rescue robots. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '03)*, Las Vegas, NV, U.S.A., October 2003.

[3] R. R. Murphy. Activities of the Rescue Robots at the World Trade Center from 11-21 September 2001. *IEEE Robotics & Automation Magazine*, 11(3):851 – 864, September 2004.

[4] A. Nüchter, K. Lingemann, J. Hertzberg, H. Surmann, K. Pervölz, M. Hennig, K. R. Tiruchinapalli, R. Worst, and T. Christaller. Mapping of Rescue Environments with Kurt3D. In *Proceedings of the IEEE International Workshop on Rescue Robotics (SSRR '05)*, pages 158 – 163, Kobe, Japan, June 2005.

[5] A. Nüchter, O. Wulf, K. Lingemann, J. Hertzberg, B. Wagner, , and H. Surmann. 3D Mapping with Semantic Knowledge. In *Proceedings of the RoboCup International Symposium*, Osaka, Japan, June 2005.

[6] J. Wang, M. Lewis, and J. Gennari. A game engine based simulation of the nist urban search and rescue arenas. In *Proceedings of the 2003 Winter Simulation Conference*, pages 1039 – 1045, 2003.

[7] J. Wang, M. Lewis, and J. Gennari. Usar: A game based simulation for teleoperation. In *Proceedings of the 47th Annual Meeting of the Human Factors and Ergonomics Society*, Denver, CO, October 2003.

[8] Jijun Wang. USARSim - A Game-based Simulation of the NIST Reference Arenas `http://usl.sis.pitt.edu/ulab/usarsim_download_page.htm` and `http://sourceforge.net/projects/usarsim`, 2005 – 2006.

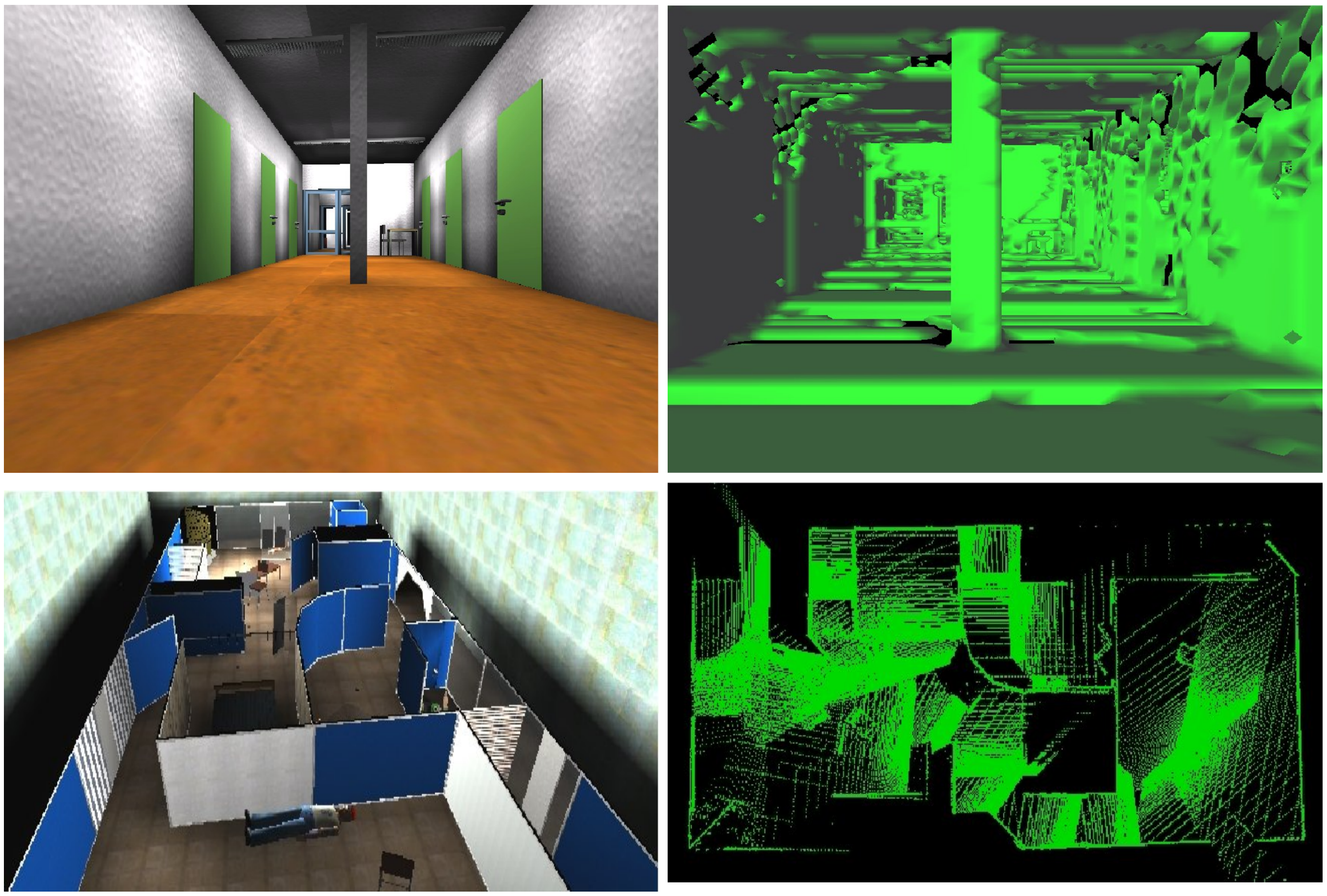


Fig. 8. Left: Unreal map of the AVZ building (top) and of the yellow arena (bottom). Right: Corresponding marching cube representation (top) and point cloud in a bird eyes view (bottom).

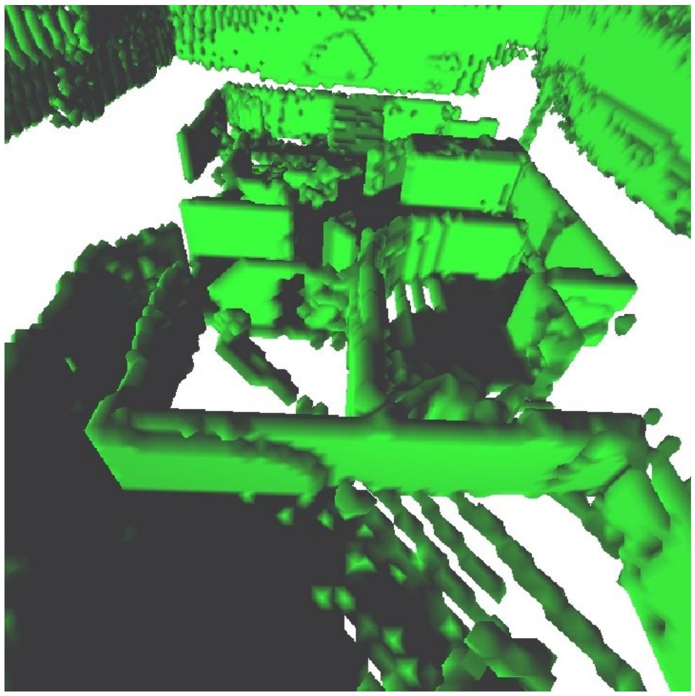


Fig. 9. Marching cube mesh of the yellow arena