Bridging the Gap Between Simulation and Reality in Urban Search and Rescue

Stefano Carpin[1], Mike Lewis[2], Jijun Wang[2],
Steve Balakirsky[3], and Chris Scrapper[3]

[1] School of Engineering and Science
International University Bremen – Germany
[2] Department of Information Sciences and Telecommunications
University of Pittsburgh – USA
[3] Intelligent Systems Division
National Institute of Standards and Technology – USA

Abstract. Research efforts in urban search and rescue grew tremendously in recent years. In this paper we illustrate a simulation software that aims to be the meeting point between the communities of researchers involved in robotics and multi-agent systems. The proposed system allows the realistic modeling of robots, sensors and actuators, as well as complex unstructured dynamic environments. Multiple heterogeneous agents can be concurrently spawned inside the environment. We explain how different sensors and actuators have been added to the system and show how a seamless migration of code between real and simulated robots is possible. Quantitative results supporting the validation of simulation accuracy are also presented.

1 Introduction

Urban search and rescue (USAR) can be depicted as the research field that experienced the most vigorous development in recent years within the robotics community. It offers a unique combination of engineering and scientific challenges in a socially relevant application domain [5]. The broad spectrum of relevant topics attracts the attention of a wide group of researchers, with expertise as diverse as advanced locomotion systems, sensor fusion, cooperative multiagent planning, human-robot interfaces and more. In this framework, the contest schema adopted by the RoboCup Rescue community, with the distinction between the real robots competition and the simulation competition, captures the two extremes of this growing community. Looking back at the past RoboCup events, tremendous progresses in short time characterized both communities. In 2002 the real rescue robots competition was described as a competition where teleoperated robots were mainly used because of the complexity of the problem [3]. In the simulation competition, emphasis was instead on the inter-agent communication models adopted [9]. The huge gap between these two extremes is evident. Only two years later [6], the real robot competition saw the advent of teams with three dimensional mapping software, intelligent perception, and the

G. Lakemeyer et al. (Eds.): RoboCup 2006, LNAI 4434, pp. 1–12, 2007.

first team with a fully autonomous multi-robot system. Within the simulation competition, teams exhibited cooperative behaviors, special agent programming languages and learning components. With these premises, it is evident that soon a mutual migration of relevant techniques will materialize. Nevertheless, certain logistic obstacles still prevent a seamless and profitable percolation of ideas and knowledge. Having set the scene, in this paper we present the latest developments of a simulation environment, called USARsim, that naturally plays the role of an in-between research tool where multi-agent and multi-robot systems can be studied in a artificial environment offering experimental conditions comparable to reality. After a demo stage during Robocup 2005 in Osaka, USARsim has been selected as the software infrastructure underlying the Virtual Robots competition, that was approved as the third competition within the RoboCup rescue simulation framework. In addition, we also offer an overview of the MOAST API, a component based software framework that can be used to quickly prototype control software, both in reality and on top of USARsim. Finally, we provide results supporting a quantitative evaluation of the simulator fidelity.

2 Software Structure

The current version of USARsim is based on the UnrealEngine2 game engine released by Epic Games with Unreal Tournament 2004. The simulation is written as a combination of levels, describing the 3-D layout of the arenas and modifications, and scripts redefining the simulations behavior. The engine to run the simulation can be inexpensively obtained by buying the game. The Unreal Engine provides a sophisticated graphical development environment and a variety of specialized tools. The engine includes modules handling input, output (3D rendering, 2D drawing, sound), networking and physics and dynamics. The games level defines a 3-D environment in much the same way as VRML (virtual reality markup language) and may use many of the same tools. The game code handles most of the basic mechanics of simulation including simple physics. Multiplayer games use a client-server architecture in which the server maintains the reference state of the simulation while clients perform the complex graphics computations needed to display their individual views. USARsim uses this feature to provide controllable camera views and the ability to control multiple robots. Unreal Tournament has two types of entities, human players who run individual copies of the game and connect to the server (typically running on the first players machine), and "bots" (short for robots), simulated players running simple reactive programs. Gamebots, a modification to the Unreal Tournament game that allows bots to be controlled through a normal TCP/IP socket [1], provides a protocol for interacting with Unreal Tournament. Because the full range of bot commands and Unreal scripts can be accessed over this connection GameBots provides a more powerful and flexible entry into the simulation than the player interface. The GameBot interface is ideal for simulating USAR robots because it can both access bot commands such as Trace to simulate sensors and exert complicated forms of control such as adjusting motor torques to control a simulated

robot. One of the client options, the spectate mode, allows the clients viewpoint (camera location and orientation from which the simulation is viewed) to be attached to any other player including "bots". By combining a bot controlled by GameBots with a spectator client we can simulate a robot with access to both simulated sensor data through the bot and a simulated video feed through the spectating client. By controlling the simulated robot indirectly through GameBots rather than as a normal client we gain the additional advantage of being able to simulate an autonomous robot (controlled by a program) a teleoperated robot (controlled by user input) or any level of automation in between.

3 Robot Interfaces

An intelligent system must translate a mission command into actuator voltages. While this may be done in a simple monolithic module, USARsim/MOAST implements a hierarchical control structure that compartmentalizes the control system responsibility and domain knowledge necessary to create each controller. The knowledge and control requirements of a typical robotic platform may be decomposed into the two broad areas of sensing and behavior generation. In turn, behavior generation may be decomposed into mobility behaviors and mission package behaviors. In this decomposition, mobility refers to the control aspects of the vehicle that relate only to the vehicle's motion (e.g. drive wheel velocities), sensing refers to systems that acquire information from the world (e.g. cameras), and mission packages are controllable items on the platform that are not related to mobility (e.g. camera pan/tilt or robotic arm). It is the authors' belief that decomposing a system in this way allows for the creation of a generic internal representation and control interface that is able to fully control most aspects of robotic platforms. USARsim is designed to implement this decomposition and provides developers with a modular interface into the low-level simulated hardware of the robotic platform. It provides for component discovery, and independent control of mobility, sensors, and mission packages. Coupling USARsim to the Mobility Open Architecture Simulation and Tools (MOAST) framework adds modularity in time by providing a set of hierarchical interfaces into these components. Two different physical control interfaces exist into the system. The first allows low-level control into USARsim and is based on sending ASCII text over a TCP/IP socket. Higher-level commands and status utilize the Neutral Message Language (NML) [8] that permits a physical interface of various types of sockets as well as serial lines.

3.1 USARsim Socket API

During the development of the interface to USARsim many factors were taken into account to ensure that the interface was both well-defined and standardized. Scientific standards and conventions for units, coordinate systems, and interfaces were used whenever possible. USARsim decouples the units of measurement used inside Unreal by ensuring that all units meet the International System of Units

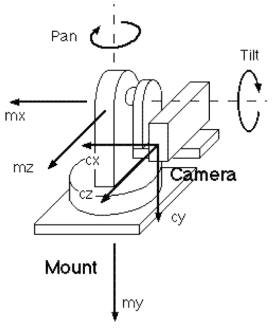

Fig. 1. Depiction of the Mission Package and the Sensor and their corresponding coordinate systems

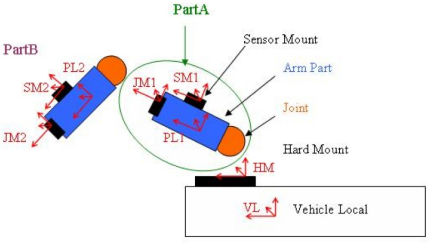

Fig. 2. Internal Representation of an Robotic Arm

(SI) standard conventions. SI Units are a National Institute of Standards and Technology (NIST) developed convention that is built on the modern metric system, and is recognized internationally. The coordinate systems for various components must be consistent, standardized, and anchored in the global coordinate system, as illustrated in Figure 1. USARsim leverages the previous efforts of the Society of Automotive Engineers, who published a set of standards for vehicle dynamics called SAE J670: Vehicle Dynamic Terminology. This set of standards is recognized as the American National Standard for vehicle dynamics and contains a comprehensive set of standards that describes vehicle dynamics through illustrated pictures of coordinate systems, definitions, and formal mathematical representations of the dynamics. Finally, the messaging protocol, including the primitives, syntax, and the semantics must be defined for the interface. Messaging protocols are used in USARsim to insure that infrequent and vital messages are received. The primitives, syntax, and semantics define the means in which a system may effectively communicate with USARsim, namely to speak USARsim's language. There are three basic components that exist currently in USARsim: robots, sensors, and mission package. For each class of objects there are defined class-conditional messages that enable a user to query the component's geography and configuration, send commands to, and receive status back. This enables the embodied agent controlling the virtual robot to be self-aware and maintain a closed-loop controller on actuators and sensors. The formulation of these messages are based on an underlying representation of the object, includes their coordinate system, composition of parts, and capabilities. This highlights a critical aspect underlying the entire interface; the representation of the components and how to control those components. For example, take a robotic arm, whose internal representation of an arm is visualized in Figure 2. In order for there to be a complete and closed representation of this robotic arm, the following aspects are defined as individual class conditional messages that are sent over the USARsim socket.

Configuration: How to represent the components and the assembly with respect to each other.

Geography: How to represent the pose of the sensor mounts and joints mount with respect to the part, and the pose of the part with respect its parent part.

Commands: How to represent the movements of each of the joints, either in terms of position and orientation or velocity vectors.

Status: How to represent the current state of the robotic arm.

3.2 Simulation Interface Middleware (*SIMware*)

Residing between USARSim and MOAST is the SIMware layer. This layer provides a modular environment and allows for a gradient of configurations from the purely virtual world to the real world. SIMware is designed to enable MOAST to conect to interfaces or APIs for real or virtual vehicles. It seemlessly connect to platforms with different messaging protocols, semantics, or different levels of abstraction. SIMware is made up of three basic components: a core, knowledge repository, and skins. The core of SIMware is essentially a set of state tables and interfaces that enables SIMware to administer the transference of data between two different interfaces. This transference is enabled through the use of knowledge repositories that provide insite into the target platform's capibilities and abstraction. The skins are an interface specific parsing utility that utilize the knowlege repository in order to enable the core to translate incoming and outgoing message traffic to meet the appropriate level of abstraction for the target interface.

3.3 MOAST API

The MOAST framework connects into USARsim via SIMware and provides additional capabilities for the system. These capabilities are encapsulated in components that are designed based on the hierarchical 4-D/RCS Reference Model Architecture [2]. The 4-D/RCS hierarchy is designed so that as one moves up the hierarchy, the scope of responsibility and knowledge increases, and the resolution of this knowledge and responsibility decreases. Each echelon (or level) of the 4-D/RCS architecture performs the same general type of functions: sensory processing (SP), world modeling (WM), value judgment (VJ), and behavior generation (BG). Sensory processing is responsible for populating the world model with relevant facts. These facts are based on both raw sensor data, and the results of previous SP (in the form of partial results or predictions of future results). The world model must store this information, information about the system self, and general world knowledge and rules. Furthermore, it must provide a means of interpreting and accessing this data. Behavior generation computes possible courses of action to take based on the knowledge in the WM, the systems goals, and the results of plan simulations. Value judgment aids in the BG process by providing a cost/benefit ratio for possible actions and world states.

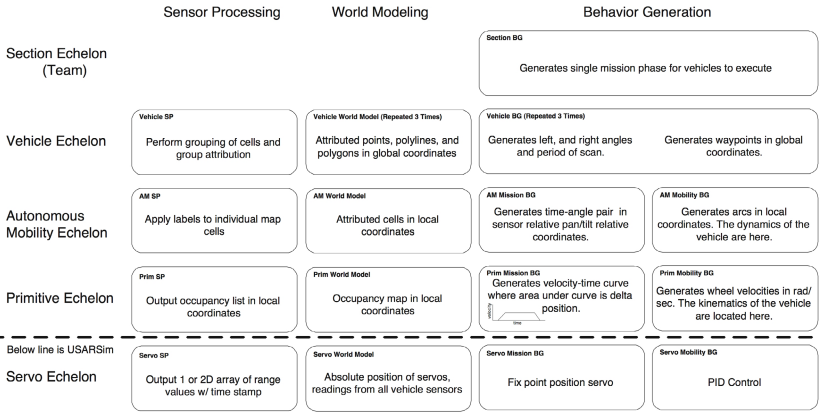

Fig. 3. Modular Decomposition of MOAST framework that provides modularity in broad task scope and time

The regularity of the architectural structure in 4-D/RCS enables scaling to any arbitrary size or level of complexity. Each echelon within 4-D/RCS has a characteristic range and resolution in space and time. Each echelon has characteristic tasks and plans, knowledge requirements, values, and rules for decision-making. Every component in each echelon has a limited span of control, a limited number of tasks to perform, a limited number of resources to manage, a limited number of skills to master, a limited planning horizon, and a limited amount of detail with which to cope.

This decomposition is depicted in Figure 3. Under this decomposition, the USARsim API may be seen as fulfilling the role of the servo echelon, where both the mobility and mission control components fall under BG. The sensors are able to output arrays of values, world model information about the vehicle self is delivered, and mission package and mobility control are possible. The remainder of this section will concentrate on the functioning and interfaces of the remaining echelons of the hierarchy.

Primitive Echelon. The primitive echelon behavior generation is in charge of translating constant curvature arcs or position constraints for vehicle systems into velocity profiles for individual component actuators based on vehicle kinematics. For example, the AM Mobility BG will send a dynamically correct constant curvature arc for the vehicle to traverse. This trajectory will contain both position and velocity information for the vehicle as a whole. For a skid steered vehicle, the Primitive Echelon BG plans individual wheel velocities based on the vehicle's kinematics that will cause to vehicle to follow the commanded trajectory. During the trajectory execution, BG will read vehicle state information from the Servo Echelon WM to assure that the trajectory is being maintained and will take corrective action if it is not. Failure to maintain the trajectory within the commanded tolerance will cause BG to send an error status to the AM Mobility BG.

The Primitive Echelon SP is in charge of converting sensor reports from sensor local coordinates to vehicle local coordinates. This information is read by the world model process which performs spatial-temporal averaging to create an occupancy map of the environment in vehicle local coordinates. This map is of fixed size and is centered on the current vehicle location. As the vehicle moves, distant objects fall off of the map. Future enhancements will allow for the population of newly added map area with any information that may be stored in the larger extents AM WM.

Autonomous Mobility Echelon. The Autonomous Mobility Echelon behavior generation is in charge of translating commanded way-points for vehicle systems into dynamically feasible trajectories. For example, the Vehicle Echelon mission controller may command a pan/tilt platform to scan between two absolute coordinate angles (e.g. due north and due east) with a given period. BG must take into account the vehicle motion and feasible pan/tilt acceleration/deceleration curves in order to generate velocity profiles for the unit to meet the commanded objectives. BG modules at this level may take advantage of all of the world model services provided o the Primitive Echelon in addition to the occupancy maps that have are maintained by the Primitive Echelon WM.

SP at this level extracts environmental attributes and in conjunction with WM labels the previously generated occupancy map with these attributes. Examples of attributes include terrain slope, and vegetation density.

Vehicle Echelon. The Vehicle Echelon behavior generation is in charge of accepting a mission for an individual vehicle to accomplish and decomposing this mission into commands for the vehicle subsystems. Coordinated way-points in global coordinates are then created for the vehicle systems to follow. This level must balance possibly conflicting objectives in order to determine these way-points. For example, the Section Echelon mobility BG may command the vehicle to arrive safely at a particular location by a certain time while searching for victims of an earthquake. The Vehicle Echelon mobility BG must plan a path that maximizes the chances of meeting the time schedule while minimizing the chance of an accident; and the Vehicle Echelon mission BG must plan a camera pan/tilt schedule that maximizes obstacle detection and victim detection. Both of these planning missions may present conflicting objectives.

SP at this level works on grouping cells from the AM WM into attributed points, lines, and polygons. These features are stored in a WM knowledge-base that supports SQL based spatial queries.

Section (Team) Echelon and Above. The highest level that has currently been implemented under the MOAST framework is the Section or Team Echelon. This level of BG has the responsibility of taking high-level tasks and decomposing them into tasks for multiple vehicles. For example, the Section Echelon mobility may plan cooperative routes for two vehicle to take in order to explore a building. This level must take into account individual vehicle competencies in order to create effective team arrangements. Higher echelon responsibilities

would include such items as planning for groups of vehicles. An example of this would be commanding Section 1 to explore the first floor of a building and Section 2 to explore the second floor. Based on the individual teams performance, responsibilities may have to be adjusted or reassigned.

4 Validation

The usefulness of a simulation such as USARsim as a research tool is strongly dependent on the degree to which it has been validated and the availability of validation data for use in choosing models and assessing the generalizability of results. The provision of common and standard tools allows researchers to compare results, share software and advances, and collaborate in ways that would be impossible otherwise. While many of these benefits accrue simply from standardization, others require a closer correspondence between simulation and reality. While a human-robot interaction (HRI) experiment may not demand full realism in the behavior of a PID controller, replicating constraints such as a narrow field of view and invisibility of obstacles obstructing wheels may be essential to achieving results relevant to the operation of actual robots. Researchers wishing to port code developed in simulation to a real robot by contrast may need the highest fidelity model of the control system attainable to get useful results. In validating USARsim we are attempting to measure correspondences as precisely as possible so they also may serve for lower fidelity uses and where this is not possible identify those areas in which only low fidelity results are available.

A comparison of feature extraction for the Orange Arena using a laser range finder (Hokuyo PB9-11) on an experimental robot and its simulation in USARsim was reported already in [4]. The mapped areas along with their Hough transforms were practically identical and adjustable parameters tuned using the simulation did not require change when moved to the real robot. We have since conducted validation studies investigating HRI for the Personal Explorer Rover (PER) [7] and the Pioneer, P2/P3-DX. Some of these results for the PER were reported in [10]. This HRI validation testing was conducted at Carnegie Mellons replica of the NIST Orange Arena using both point-to-point and teleoperation control modes for the PER and teleoperation only for the pioneer P2-AT (simulation)/P3-AT (robot). In this study driving performance was observed for different surfaces and simple and complex courses using point-to-point or teleoperation control modes. Participants controlled the real and simulated robots from a personal computer located in the Usability laboratory at the School of Information Sciences, University of Pittsburgh. For simulation trials the simulation of the Orange Arena ran on an adjacent PC. For the real robotic control trials the participants controlled robots over the Internet in a replica of the Orange Arena in the basement of Newell Simon Hall at Carnegie Mellon University (see figure 4). Measures such as the distance from the stopping point to the target cone were collected for both the physical arena and the simulation. A standard interface developed for RoboCup USAR competition [7] was used under all conditions. Participants in the direct control mode controlled the robots

Fig. 4. On the left side the orange arena at CMU. On the right side the simulated arena within USARsim. The yellow cone to be reached can be observed in both images.

using a joystick. Both robots were skid steered so forward backward movements of the joystick led to movement while right/left movements produced changes in yaw. In the waypoint control mode participants selected waypoints by clicking on locations on the video display. This input was interpreted by the control software as specifying a direction and duration of travel. Manual adjustments in the point-to-point condition were made using the cursor keys.

Procedure. In Stage 1 testing of the PER and Pioneer (direct control mode) we established times, distances, and errors associated with movements over a wood floor, paper, and lava rocks. These data were used to adjust the speed of the simulated PER and Pioneer and alter the performance of the simulated PER when moving over scattered papers. In Stage 2 testing, PER robots were repeatedly run along a narrow corridor with varying types of debris (wood floor, scattered papers, lava rocks) while the sequence, timing and magnitude of commands were recorded. Participants were assigned to maneuver the robot with either direct teleoperation or waypoint (specified distance) modes of control. There were five participants in each of the PER groups (real-direct, real-waypoint, simulation-direct, simulation-waypoint) and four in the Pioneer (real-direct, simulation-direct) groups. In the initial three exposures to each environment, participants had to drive approximately three-meters, along an unobstructed path to an orange traffic cone. In later trials, obstacles were added to the environments, forcing the driver to negotiate at least three turns to reach the destination. The distances from stopping position to the goal and task times were recorded for both simulated and real trials. A time-stamped log of control actions and durations were collected for both real and simulated robots.

Terrain effects. The paper surface had little effect on either robots operation. The rocky surface by contrast had a considerable impact, including a loss of trac-

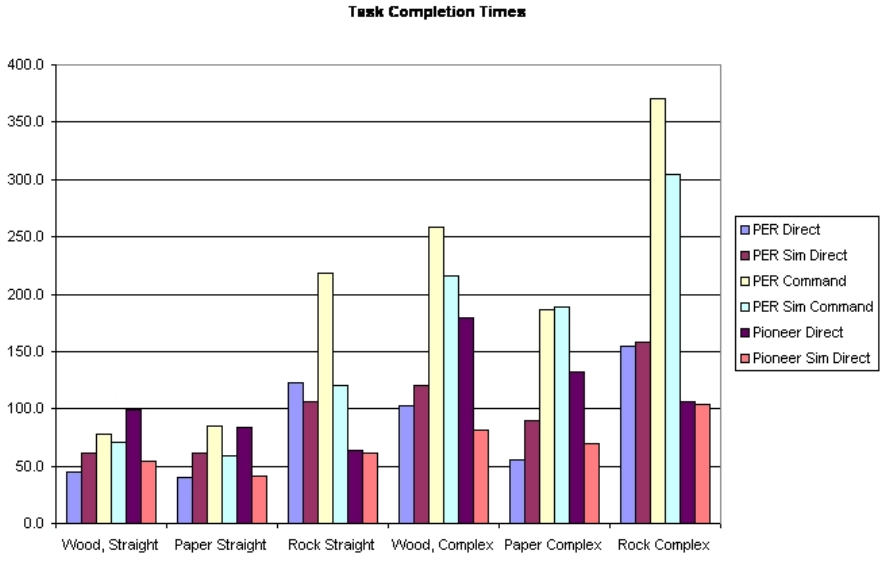

Fig. 5. Distribution of the times to complete the mission

tion and deflection of the robot. This was reflected by increases in the odometry and number of turn commands issued by the operators even for the straight course. A parallel spike in these metrics is recorded in the simulator data. As expected the complex course also led to more turning even on the wood floor.

Figure 5 shows these data for the simulated and actual PER and Pioneer.

Proximity. One metric on which the PER simulation and the physical robot consistently differed was the proximity to the cone acquired by the operator. Participants were given the instruction to get as close to the cone as possible without touching it. Operators using the physical robot reliably moved the robot to within 35cm from the cone, while the USARsim operators were usually closer to 80cm from the cone. It is unlikely that the simulation would have elicited more caution from the operators, so this result suggests that there could be a systematic distortion in depth perception, situation awareness, or strategy. In both cases the cone filled the cameras view at the end of the task. Alternatively, the actual PER was equipped with a safeguard to prevent running into objects while the simulated PER was not. Although this feature was not included in the instructions participants may have discovered it in controlling the robot and adopted a strategy of simply driving until the robot stopped. Figure 6 shows the distribution of these stopping distances. Another issue addressable from these data is the extent to which similarities in performance are a function of the platforms being simulated or differences between the simulation and control of real robots. Figure 5 suggests that both influences are present. As with our other data there are clear differences associated with platform and control mode. Note for instance the consistently shorter completion times shown in figure 5 for both

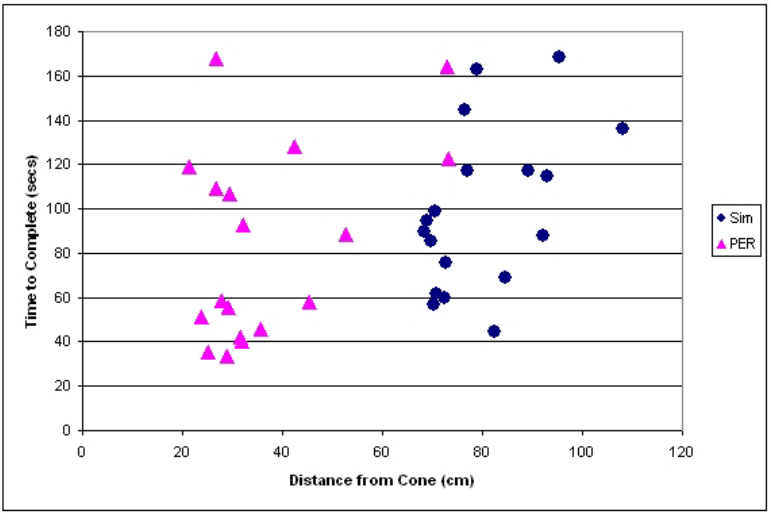

Fig. 6. Distribution of the stopping distances of the PER robot from the cone

actual and simulated Pioneers. Idle times, however, were much closer between the simulated PER and Pioneer than between the simulations and the simulated platforms. These substantially longer pauses between actions in controlling the real robot occurred despite matching frame rates although slight differences in response lag may have played a factor. Despite the difference in length of pauses completion times remain very close between the robot and the simulation. The average number of commands were also very similar between the simulation and the PER for control mode and environment except for straight travel over rocks in command mode where PER participants issued more than twice as many commands as those in the simulation or direct operation modes. A similar pattern occurs for forward distance traveled with close performance between simulation and PER for all conditions but straight travel over rocks, only now it is the teleoperated simulation that is higher.

5 Conclusions

In this paper we have presented the latest developments concerning the USARsim simulation environment, a natural candidate to meet the demands of researchers involved both in simulation and with real robots. Initial validation results show an appealing correspondence between experiences gained with USARsim and the corresponding real robots. Further benefits from USARsim can be obtained using MOAST, a framework that aids the development of autonomous robots. USARsim software and MOAST can be obtained for free from sourceforge.net/projects/usarsim and souceforge.net/projects/moast, respectively. As our library of models and validation data expands we hope to begin

incorporating more rugged and realistic robots, tasks and environments. Accurate modeling tracked robots which will be made possible by the release of UnrealEngine3 would be a major step in this direction. The open source model adopted for the development of these software foster the active involvement of multiple developers and already gained quite some popularity.

References

1. Kaminka, G., Veloso, M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A., Scholer, A., Tejada, S.: GameBots: A flexible test bed for multiagent team research. Comm. of the Association for Computing Machinery 45(1), 43–45 (2002)
2. Albus, J.: 4-D/RCS Reference Model Architecture for Unmanned Ground Vehicles. In: Proc. IEEE Int. Conf. on Robotics and Automation, pp. 3260–3265 (2000)
3. Asada, M., Kaminka, G.A.: An overview of RoboCup 2002 Fukuoka/Busan. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002. LNCS (LNAI), vol. 2752, pp. 1–7. Springer, Heidelberg (2003)
4. Carpin, S., Wang, J., Lewis, M., Birk, A., Jacoff, A.: High fidelity tools for rescue robotics: Results and perspectives. In: Robocup 2005 Symposium (2005)
5. Kitano, H., Tadokoro, S.: Robocup rescue: a grand challenge for multiagent and intelligent systems. AI Magazine (1), 39–52 (2001)
6. Lima, P., Custódio, L.: RoboCup 2004 Overview. In: Nardi, D., Riedmiller, M., Sammut, C., Santos-Victor, J. (eds.) RoboCup 2004. LNCS (LNAI), vol. 3276, pp. 1–17. Springer, Heidelberg (2005)
7. Nourbakhsh, I., Sycara, K., Koes, M., Young, M., Lewis, M., Burion, S.: Human-Robot Teaming for Search and Rescue, IEEE Pervasive Computing, pp. 72–78 (2005)
8. Shackleford, W.P., Proctor, F.M., Michaloski, J.L.: The Neutral Message Language: A model and Method for Message Passing in Heterogeneous Environments. In: Proceedings of the 2000 World Automation Conference (2000)
9. Tomoiki, T.: RoboCupRescue Simulation league. In: Kaminka, G.A., Lima, P.U., Rojas, R. (eds.) RoboCup 2002. LNCS (LNAI), vol. 2752, pp. 477–481. Springer, Heidelberg (2003)
10. Wang, J., Lewis, M., Hughes, S., Koes, M., Carpin, S.: Validating USARsim for use in HRI Research. In: Proceedings of the Human Factors and Ergonomics Society 49th Annual Meeting, pp. 457–461 (2005)