RSARSim: A Toolkit for Evaluating HRI in Robotic Search and Rescue Tasks

Bennie Lewis and Gita Sukthankar
School of Electrical Engineering and Computer Science
University of Central Florida, Orlando FL 32816-2362
BennieLewis@knights.ucf.edu, gitars@eecs.ucf.edu

ABSTRACT
A well-designed human-robot interface (HRI) is important for the success of search and rescue operations in urban environments. Controlling multiple robotic agents increases the complexity of the human operator's control task since the human operator's attention has to be divided between several robots. Without a good user interface, adding more robots to the system will not necessarily increase the area that the robots can cover within a bounded period of time. In this paper, we describe our search and rescue simulation toolkit, RSARSim (Robotic Search and Rescue Simulation), for evaluating multi-robot control paradigms. RSARSim allows users to prototype example multi-robot teams using Microsoft Robotics Developers Studio and to experiment with possible user interfaces. Using RSARSim, we developed and evaluated two possible user interfaces for controlling a Robocup Rescue team---one using an Xbox 360 gamepad controller and the other based on a keyboard control paradigm.

Keywords
Simulation, Urban Search and Rescue, Human-Robot Interfaces, Multi-Robot Control

1. INTRODUCTION
Controlling a team of USAR (Urban Search and Rescue) robots is a challenging task for a human operator since the operator must make intelligent control decisions while being simultaneously presented with a barrage of visual information from the sensors of multiple robots. Although increasing the number of robots can expand the area that a USAR team can cover within a bounded period of time, a poor human-robot interface will ultimately compromise the performance of the robotic team. The Robocup Rescue League [5] was introduced to provide testbeds, datasets, and a reference problem for researchers interested in studying multi-robot systems for urban search and rescue. Robocup Rescue teams are evaluated using a scoring metric that rewards teams for their ability to locate concealed victims and accurately map the environment. Although there are many research challenges in creating a team with effective sensing, mapping, and mobility, current competition rules state that only one person is allowed at the operator station at any time, regardless of the number of robots on the team. Hence, having a good human-robot interface and a well-trained operator are critical to the success of most Robocup Rescue teams.

Having the capability to prototype human-robot interfaces, experiment with different control paradigms, and train an operator to rapidly find victims with a specific user interface is a valuable resource for Robocup Rescue developers. In this paper, we describe the simulation toolkit that we have implemented, RSARSim (Robotic Search and Rescue Simulation) and are currently using for simulating multi-robot systems and evaluating user interfaces. RSARSim was designed to be used with Microsoft Robotics Developer Studio 2008, a flexible and powerful Windows-based environment that allows developers to create robotics applications for a variety of hardware platforms. Although there are other publicly available USAR simulation systems, ours is the only one currently available for Microsoft Robotics Developer Studio.

In this paper, we describe the implementation and usage of RSARSim for evaluating human-robot interfaces in an urban search and rescue domain. Section 2 provides some background on the capabilities of Microsoft Robotics Developers Studio. Section 3 gives a detailed comparison of USAR simulation software packages. In Section 4, we present an overview of RSARSim and its usage for Robocup Rescue. Section 5 describes the software services paradigm and Section 6 gives an example of the services used to simulate a Pioneer 3DX robot. We conclude the paper with an illustration of how we used RSARSim to evaluate two HRI paradigms for multi-robot control and discuss related work in this area.

2. MICROSOFT ROBOTICS STUDIO
Microsoft Robotics Developers Studio (MSRDS) includes a lightweight, service-oriented runtime, a set of visual authoring and simulation tools, along with introductory tutorials and sample code [1]. It was built be useful to a broad range of potential developers (academic, hobbyist, and commercial) and to be extendable to many robotic platforms.

MSRDS supports the .NET framework, enabling the use of Microsoft Visual Studio to design and develop robot applications. The developer can use any of the programming languages that are supported by the .NET framework, including Visual C#, Visual C++, Visual Basic .NET and scripting languages such as Perl and Python. For less experienced developers, MSRDS comes with a graphical programming language, Visual Programming Language (VPL). VPL offers a visual interface that allows beginners to use drag-and-drop techniques to build robotic programs. Additionally, VPL is useful to expert programmers who want to quickly prototype an application since many programs written with VPL can be directly converted into Visual C# applications.

The Visual Simulation Environment (VSE) tool that is included with MSRDS offers programmers a way to get involved with robotics development without any robotics hardware. VSE is powered by the NVIDIA PhysX engine and uses Microsoft XNA and DirectX for 3-D rendering. The Visual Simulation Environment manages sets of entities representing physical objects such as robots, sensors, and walls. *Services* are used to move the robots around and gather data from sensors; orchestration services are used to coordinate a group of services. Graphic scenes can be rendered in one of four modes---visual, physics, wireframe, or a combination of the three modes. Visual mode is a representation of what the robot would look like in a

real world environment. Microsoft Robotics Studio also includes support for packages to add other services. Those currently available include a Soccer Simulation and a Sumo Competition by Microsoft, as well as a community-developed Maze Simulator.

3. USAR SIMULATOR SOFTWARE

Before developing our simulator, RSARSim, we evaluated several other USAR simulators on their suitability for prototyping and evaluating human-robot interfaces. Initially, we chose to use Gridworld Search and Rescue [13], an educational Java-based search and rescue simulator, to implement an autonomous multi-robot team. Gridworld Search and Rescue was developed as AI courseware (as the name suggests it uses a gridworld representation) and was not suitable for researching realistic multi-robot sensor and control problems.

The USAR Virtual Robot Competition was introduced in 2006 to allow groups that couldn't afford to build and transport a team of physical robots to participate in the Robocup Rescue competition. The Virtual Robot Competition currently uses Unreal Engine, a commercial game engine built by Epic Games, for simulation and visualization. USARSim includes a collection of robot models and classes to instantiate commonly used sensors and actuators for the USAR competition. Robot controllers can connect to USARSim over a text-based socket interface (Gamebots).

Although users can create their own controllers, USARSim also interfaces with MOAST (Mobility Open Architecture Simulation and Tools) [9], a software framework developed by NIST for creating multi-robot controllers. MOAST uses a layered architecture with five levels (echelons): servo, primitive, autonomous mobility, vehicle, and section (team). Each echelon except the team one, provides a sensor processing function, a world modeling function, and a behavior generation function. Modules communicate using NML (the Neutral Messaging Language).

During our development experiences with USARSim and MOAST in 2008, we found Unreal Engine to be unstable and resource-intensive to use, particularly for simulating large Ackerman-steered vehicles. Running MOAST along with USARSim required the use of a second Linux system. Our developers experienced frequent problems with the path planning provided by the MOAST architecture; occasionally the planner never completed its search, depending on the initial starting conditions, and yielded robots that would stubbornly refuse to move forward.

Driven by a desire to create a better debugging and development environment, we started investigating Microsoft Robotics Developers Studio and developed our own custom urban search and rescue toolkit, RSARSim. RSARSim allows users to leverage the development tools provided by MSRDS (the .NET development framework and the Visual Programming Language) and provides a customized USAR simulation using Microsoft's Visual Simulation Environment, freeing the user from the external dependency of Unreal Engine. We believe that our environment is more accessible to beginning users than USARSim while still providing a powerful, extendable development environment for experienced programmers. Microsoft plans to release a rescue environment for their RoboChamps competition but preliminary information indicates that the software will be customized for their simulated robot and set of competition rules.

4. RSARSIM

The Robotic Search and Rescue Simulation Toolkit 1.0 (RSARSim 1.0) was designed to be used in conjunction with MSRDS to further research and development in Human Robot Interaction (HRI) paradigms for the search and rescue domain. RSARSim (implemented using the Visual Simulation Environment) makes it easy for developers to implement their algorithms and evaluate their techniques on human subjects without a multi-robot system. RSARSim 1.0 is fully compatible with all versions of MSRDS editions including the free express edition and can be downloaded at http://ial.eecs.ucf.edu/Projects/RSARSim.

Additionally this simulator can serve a prototyping and practice environment for teams competing in the physical Robocup Rescue competition, since code developed in MSRDS can be made to be fully compatible with the physical robot hardware. 3D models and entities for the rescue level (shown in Figure 1) are specified in a Visual C# file, RescueLevel.cs; this file is called by RobotInterface.cs file to start the simulated environment in our toolkit.

The environment is divided into different color zones similar to the RoboCup Rescue levels. The walls in the RSARSim have colored lines that correspond with the autonomy/mobility zones used in the Robocup Rescue competition: yellow, orange, and red. For instance, the yellow zone requires that the robots search this area fully autonomously; failure to do so will result in no points when a victim is found, but the victim will still count as being found. The red and orange zones pose mobility challenges, but allow users to teleoperate the robots. In our environment, walls with blue lines on them can be used to denote an upcoming zone change. This "blue zone" gives the operator time to switch from teleoperation to autonomous. Victims are represented using person shaped models distributed to the locations specified in the RescueLevel.cs file.

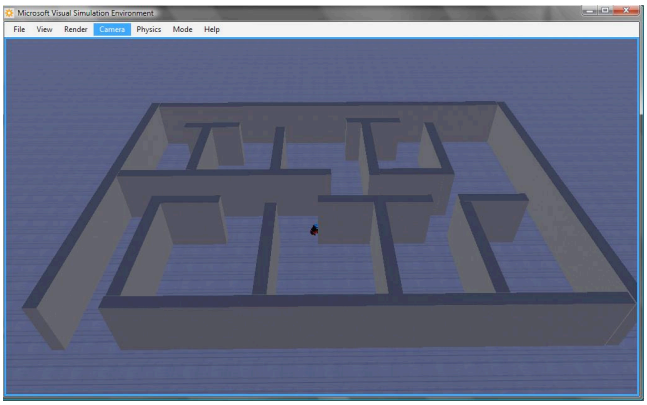

Figure 1 : An urban rescue scenario simulated using RSARSim 1.0. The user is controlling a single Pioneer robot.

5. SOFTWARE SERVICES

Services are the essential building blocks for robotic applications created with Microsoft Robotics Developer Studio 2008 [1, 8]. Every web-based service contains the code required to carry out one or more functions, such as reading data from a single sensor or transferring an output signal to an actuator. The service can also be used to communicate with an additional service or external software. A robotics application consists of several services that

work together to accomplish a common task operating the robot. MSRDS allows twenty services to run on a single host. Microsoft's Decentralized Software Services (DSS) is a NET-based runtime environment that sits on top of the Concurrency and Coordination Runtime (CCR). Decentralized Software Services provides a small state-oriented service model, which combines the concept of representational state transfer (REST) with a system level approach. DSS services are exposed as resources that are available both programmatically and for UI management. A crucial design goal of DSS is to couple performance with simplicity and robustness; this makes DSS for the most part suited for creating applications as compositions of services regardless of whether these services are running within the same node or across the network. DSS uses the Decentralized Software Services Protocol and HTTP as the foundation for interacting with services. DSSP is a SOAP based protocol that provides a fresh symmetric state transfer application model, with support for state manipulation and an event model determined by state changes. The DSS runtime provides a hosting environment with built-in support for service composition, security, monitoring, and logging both within a single node and across the network. Services can be written either in Visual Studio or using Microsoft's Visual Programming Language. Additionally, the DSS Manifest Editor provides a graphical environment for running DSS applications on a single node or across the network.

6. EXAMPLE ROBOT: PIONEER 3DX

MSDRS can be used to simulate a wide range of robotic hardware. Our RSARSim example scenario features a simulation version of the Pioneer 3DX (Figure 2), with bumper sensors, a webcam and a SICK Laser Range Finder. The Pioneer 3DX is a differential drive robot which can be modeled using the MSRDS service called SimulatedDifferentialDrive that implements the abstract Drive contract. The Drive contract has operations to position the left and right wheel speeds individually, alternate the entire drive by a particular angle, and to drive a specified distance. The drive service sends notification when its state changes; events include changes in wheel speed and drive enabled status. The user can configure a Pioneer 3DX with a front and rear bumper ring consisting of multiple contact sensors using the SimulatedBumper service that implements the abstract Contact Sensor contract. The service sends a notification when a bumper is pressed or released. The SICK Laser Range Finder is simulated using a service called SimulatedLRF that sends a notification for each scan.

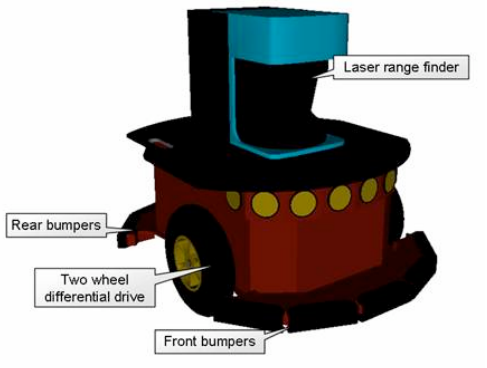

Figure 2: Simulated Pioneer 3DX robot [8]

7. EVALUATING HRI FOR USAR

Using RSARSim, we were able to prototype two possible user interfaces for multi-robot control suitable for human teleoperation of a Robocup Rescue team. In the simpler user interface, the user controls the team of four simulated Pioneer robots using the standard first-person shooter controls on a keyboard. In the second condition, the human operator navigates the robot through the RSARSim environment via a gamepad base approach using an Xbox 360 controller (Figure 5).

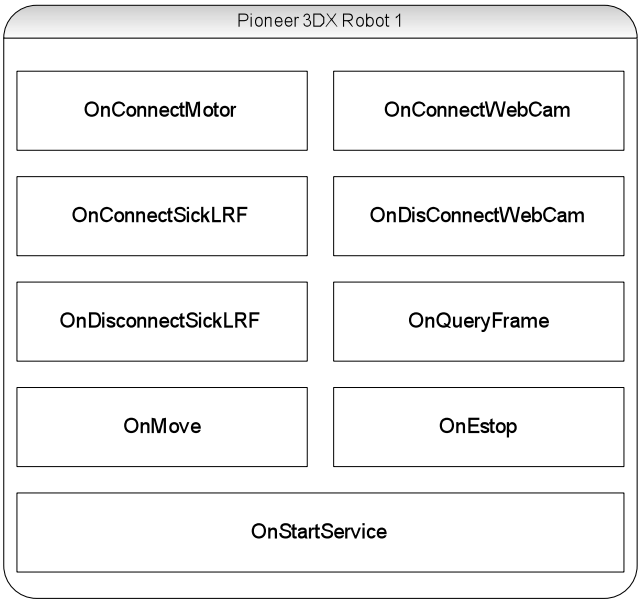

Figure 3: Services used to operate the Pioneer 3DX Robot

The current version of the implementation has sixteen individual services that it uses.

- OnStartService establishes the startup connection.
- OnLoad and OnClose services are used to handle the loading and closing of the rescue robot control panel.
- The OnChangeJoyStick service is used to handle the connection of Xbox 360 controller.
- The OnConnectMotor service is used to handle the connection of the drive function.
- OnMove and OnEstop services manage the start and stop of the motor of the four Pioneer 3DX robots.
- The OnConnectSickLRF and OnDisConnectSickLRF services manage the connection and disconnection of the laser range finder of the four Pioneer 3DX robots.
- OnConnectWebCam1-4 services manage the webcam connection for the four Pioneer 3DX robots.
- OndisconnectWebcam handles the disconnection of all cameras and the OnQueryFrame service executes the frame updates for individual cameras.

Since the predefined webcam service was designed for a single robot, we developed a different webcam service for RSARSim, that inherits from the MSRDS predefined webcam service. Our webcam service gets the latest frame and uploads the image to the corresponding image box on the rescue robot control panel. Since each robot has its own camera, our webcam service generates four

services to handle all the cameras. When the webcam service is called, it automatically determines the correct service for the specified robot.

Figure 5: Xbox 360 Controller. The human operator can switch between robots using the colored buttons (top right).

Our game controller service also inherits some traits from the predefined controller service included with MSRDS. The rescue robot control panel, the button configuration for the gamepad, and the keyboard controller are specified in the DriveControl.cs file. The layout of the gamepad was based on the layout of the Xbox 360 Controller. To connect the rescue robot control panel (Figure 4) to VSE, the operator must press the start button on the controller. The four colored buttons each represent a robot. If pressed, the button allows the human to teleoperate that robot, and the control panel displays the corresponding sensor information. To drive that robot the operator presses the RB button and steers the robot using the left analog stick; pressing the select button disconnects the laser range finder service. To control the robot using the keyboard, the user selects the robot drive service using the list box on the rescue robot control panel and drives the robot using the W,A,S,D keys .

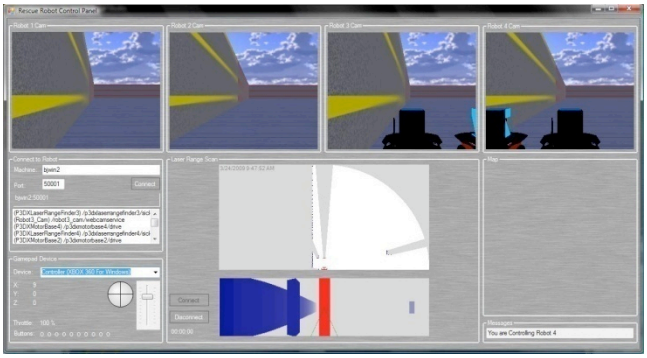

Figure 4: The RSARSim robot control panel displays: 1) 2D and 3D information from the laser range finder (middle); 2) webcam images from the robots (top).

We evaluated an initial group of human subjects on their ability to locate victims while teleoperating 4 simulated Pioneer 3DX robots using two different low-level teleoperation paradigms: 1) a keyboard based user interface and 2) the Xbox 360 gamepad controller. Preliminary results indicate that the gamepad controller is preferred by the majority of subjects and results in a higher number of victims found. Additionally, we are currently evaluating different options for tasking the unmanaged robots that aren't being teleoperated by the user.

8. IMPROVING USAR PERFORMANCE

There are a number of possible approaches for improving the overall performance of a USAR team in addition to refining the human-robot interaction paradigm. For instance, Wirth and Markov [2] describe an exploration transform approach for rapidly finding good paths to frontier cells in the map; it was successfully used during the RoboCup World Championship 2007 by the team that achieved the Best in Class Autonomy Award. Accurately detecting relevant environmental features is an important aspect of real-world USAR tasks; Gossow and Pellenz [3] proposed a technique for danger sign detection in rescue environments that was used in the RoboCup 2008 World Championship. Many USAR situations challenge conventional multi-agent planning techniques since they involve heterogeneous agents, joint tasks, and are subject to additional constraints. To address these issues, Koes, Nourbakhsh, and Sycara [7] developed COCoA, the Constraint Optimization Coordination Architecture, and demonstrated a mixed-integer linear programming optimization procedure for identifying good multi-robot task allocations.

There are many promising techniques for improving human-robot interaction in USAR teams. For instance, Pathak and Birk [4] created an integrated hardware and software framework that used fast lightweight robots and an optimized GUI to improve teleoperation. Their design was tested at the RoboCup 2006 World Championship. Two general approaches for improving human-robot interactions in USAR are: 1) improving visualization of the environment to reduce the cognitive load on the human operator and 2) creating adjustably autonomous robots that can operate effectively when the operator's attention is elsewhere. Ricks, Nielsen, and Goodrich [6] present an ecological interface paradigm that combines video, map, and robot pose information into a 3-D mixed-reality display. The results from their user studies show that the 3-D interface improves robot control, map-building speed, robustness in the attendance of delay, robustness to disturbing sets of information, awareness of the camera orientation with respect to the robot, and the ability to perform search tasks while navigating the robot. Intuitively, the participants favored the 3-D interface to the 2-D interface and felt that they did better, were less frustrated, and better able to anticipate how the robot would react to their commands. In the area of adjustable autonomy, Wang, Lewis, and Scerri [14] presented results on the use of teamwork proxies (implemented in Machinetta) for controlling robots performing a search and rescue task. Instrumenting the robots with these teamwork proxies enabled them to collaborate more effectively and operate autonomously even when the human operator's attention was elsewhere. We believe that our work on low-level control using the gamepad controller complments these other approaches for improving HRI in USAR.

9. CONCLUSION AND FUTURE WORK

In this paper, we introduce RSARSim (the Robotic Search and Rescue Simulation Toolkit) and describe how it can be used to prototype USAR multi-robot teams in combination with Microsoft Robotics Developers Studio. Combining RSARSim and the Visual Simulation Environment produces a robotic prototyping environment that is both powerful and easy to use. In this paper, we show how to use RSARSim and MSRDS to simulate a team of Pioneer 3DX robots for Robocup Rescue. Using RSARSim, we were able to rapidly evaluate the comparative effectiveness of two

different HRI paradigms for multi-robot control. For future work we intend to expand the functionality of the RSARSim toolkit to include more visualizations and teleoperation options for the USAR domain.

ACKNOWLEGMENTS

Thanks to David C. Lyle for his work with MOAST and Paul Scerri for helping us with USARSim. Eric Eaton and Marie desJardins were kind enough to provide us with the code for the Gridworld Search and Rescue simulator.

10. REFERENCES

[1] S. Morgan, Programming Microsoft Robotics Studio, Microsoft Press, 2008

[2] S. Wirth and S. Markov. "Exploration transform: a stable exploring algorithm for robots in rescue environments", Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics, 2007

[3] D. Gossow, J. Pellenz, and D. Paulus. "Danger sign detection using color histograms and SURF matching", Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics, 2008

[4] K. Pathak, A Birk, S. Schwertfeger, I. Delchef, and S. Markov. "Fully autonomous operations of a Jacobs Rugbot in the RoboCup Rescue Robot League 2006" Proceedings of the IEEE International Workshop on Safety, Security and Rescue Robotics, 2007

[5] Robocup Rescue, http://www.robocuprescue.org/, Retrieved Feb 2009

[6] B. Ricks, C. W. Nielsen, and M. A. Goodrich, "Ecological displays for robot interaction: a new perspective," Proceedings of Intelligent Robots and Systems (IROS), 2004.

[7] M. Koes, I. Nourbakhsh, and K. Sycara, "Constraint optimization coordination architecture for search and rescue robotics", in Proceedings of International Conference on Robotics and Automation (ICRA), 2006

[8] Microsoft Robotics Developer Studio, http://msdn.microsoft.com/en-us/robotics/default.aspx/ Retrieved Feb 2009

[9] C. Scrapper, S. Balakirsky, and E. Messina, "MOAST and USARSim---a combined framework for the development and testing of autonomous systems", Proceedings of the SPIE Defense and Security Symposium, 2006

[10] P, Velagapudi, P. Kwaki, P. Scerri, Robocup Rescue – Virtual Robots Team Steel, Pittsburgh PA, USA http://athiri.cimds.ri.cmu.edu/twiki/pub/UsarSim/WebHome/ steel-tdp-2008.pdf

[11] J. Casper and R. R. Murphy, "Human–robot interactions during the robot-assisted urban search and rescue response at the world trade center," IEEE Trans. Systems, Man, and Cybernetics. B, vol. 33, no. 3, pp. 367–385, Jun. 2003.

[12] S. Balakirsky, S. Carpin, A. Kleiner, M. Lewis, A. Visser, J. Wang, V. Ziparo, "Towards heterogeneous robot teams for disaster mitigation: results and performance metrics from RoboCup Rescue". Journal of Field Robotics, 2007.

[13] Eric Eaton. 'Gridworld Search and Rescue: a project framework for a course in artificial intelligence". In the Proceedings of the AAAI-08 AI Education Colloquium, July 2008

[14] J. Wang, M. Lewis, and P. Scerri, "Cooperating robots for search and rescue", Proceedings of the AAMAS 1st International Workshop on Agent Technology for Disaster Management, 2006