Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

A GAME ENGINE BASED SIMULATION OF THE NIST URBAN SEARCH & RESCUE ARENAS

Jijun Wang, Michael Lewis, and Jeffrey Gennari
School of Information Sciences
University of Pittsburgh
Pittsburgh, PA, U.S.A.

ABSTRACT

We are developing interactive simulations of the National Institute of Standards and Technology (NIST) Reference Test Facility for Autonomous Mobile Robots (Urban Search and Rescue). The NIST USAR Test Facility is a standardized disaster environment consisting of three scenarios of progressive difficulty: Yellow, Orange, and Red arenas. The USAR task focuses on robot behaviors, and physical interaction with standardized but disorderly rubble filled environments. The simulation will be used to test and evaluate designs for teleoperation interfaces and robot sensing and cooperation that will subsequently be incorporated into experimental robots. This paper describes our novel simulation approach using an inexpensive game engine to rapidly construct a visually and dynamically accurate simulation for both individual robots and robot teams.

1 INTRODUCTION

Large-scale coordination tasks in hazardous, uncertain, and time stressed environments are becoming increasingly important for fire, rescue, and military operations. Substituting robots for people in the most dangerous activities could greatly reduce the risk to human life and even allow new and more hazardous tasks to be undertaken. Because such emergencies are relatively rare and demand full focus on the immediate problems there is little opportunity to insert and experiment with robotic assistants (Murphy 2003).

1.1 NIST Arenas

The National Institute of Standards' (NIST) Reference Test Facility for Autonomous Mobile Robots for Urban Search and Rescue (USAR) (Jacoff, et al. 2001) is an attempt to replicate the challenges of such environments in a safe and reproducible way. The NIST USAR Test Facility is a standardized disaster environment consisting of three scenarios: Yellow, Orange, and Red physical arenas of progressing difficulty shown in Figure 1. The USAR task focuses on robot behaviors, and physical interaction with standardized but disorderly rubble filled environments.

The Yellow Arena resembles an office environment with a flat floor, perpendicular walls, and few obstacles. The challenges in the Yellow Arena are predominately perceptual. There are mirrors, transparent Lucite obstacles, venetian blinds, and large areas completely darkened by tarps. Success in the Yellow Arena depends on reliable redundant sensing and places little demand on locomotion.

The Orange Arena presents challenges of both sorts. It is constructed in two levels separated by difficult to navigate stairs and a ramp. Some of the floor is littered with paper while another area is strewn with dowels and small sections of pipe. Walls of some of the rooms are painted in optical illusion inducing stripes and patterns to confuse image processing and venetian blinds are again used as apparent obstacles. A negative obstacle (drop off) is introduced on the platform in the form of an open ventilation shaft. (Negative obstacles present a significant problem in robotics because they are often not apparent from an image and are more difficult to sense than 'positive' obstacles that reflect signals.) Successfully navigating the Orange Arena requires both reasonably robust sensing and locomotion able to handle stairs and some surface irregularities.

The Red Arena eschews perceptual difficulties and places maximal demand on locomotion. The design resembles an actual rubble pile with mounds of cement blocks and slabs, chicken wire, and other debris. The terrain is so irregular that it becomes difficult even for ruggedized robots to traverse without becoming entangled with rubbish, stuck in crevices, or rolling over.

Despite the improvement of documented reference tasks over idiosyncratic rubble piles, developing and comparing robots and robot teams remains a difficult task. Very few researchers have access to the permanent arenas in Maryland, California, and Japan and portable arenas are only available to competitors for short periods during major

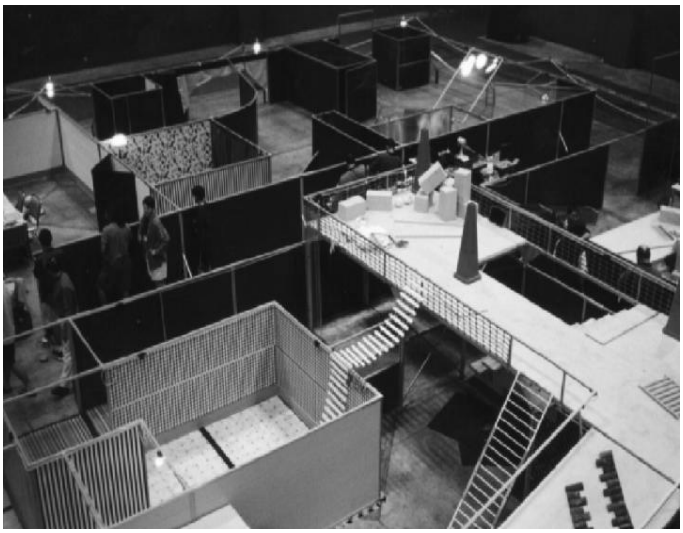

Figure 1: Orange (near) and Yellow Arenas
(NIST photograph)

meetings such as the International Joint Conference on Artificial Intelligence (IJCAI). Because researchers much wait so long between discovering flaws at one conference, implementing fixes, and testing then at the next conference, progress is slowed significantly.

1.2 Simulation Desiderata

One solution to the lack of accessible reference environments will be to develop simulations that allow researchers to test some aspects of their solutions without requiring complete implementation or access to a physical arena. The success of this approach will depend on:

- Expense and availability of simulation hardware and software to USAR robotics community
- Ease of programming to reflect targeted aspects of design
- Fidelity of simulation w.r.t. aspects of design to be tested

We have identified four characteristics of USAR robots, their tasks, and environments that need to be accommodated within a general purpose simulation:

1. *Simulated video feed*- for teleoperation and visual search and identification

To date all robots used in actual USAR operations have been teleoperated. The most crucial feature of their human-computer interface is the video feed from the robot's camera. This imagery must be used both to navigate the robot through an unknown environment and to locate and identify victims. Casper (2002) reports that perceptual errors and confusions were by far the greatest problems in robotic rescue attempts at the World Trade Center site. Fidelity in simulation of a video feed requires an accurate model of both surfaces and (visual) textures of the arena and control over camera FOV (field of view) and attitude (tilt and pan)

2. *Simulated robot dynamics*- for teleoperation and autonomous control

Experiments with either manual control or automatic control algorithms need an accurate model of robot dynamics. Ideally we would like to be able to tell from simulation whether or not a robot could climb stairs or might get stuck in chicken wire. While this level of realism may be difficult to attain an approximation which differentiates easy from difficult to traverse areas and models an increasing error in heading when navigating rough terrain would capture many crucial features.

3. *Sensor simulation*- for autonomous control and fused displays

Accurate simulations of sensors are needed to simulate robot behavior. While a human teleoperator can easily extract 3D information from a video display, this remains a difficult problem for machine vision. For expendable robots of the sorts likely to be used in USAR ranging sensors such as sonar, flir, or ladar are likely candidates and need to be modeled.

4. *Multiple entity simulation*- to allow interaction and cooperation among teams of robots.

Because the size and complexity of USAR tasks require multiple robots it is essential that simulations do as well.

2 USING GAME ENGINES FOR SIMULATION

The cost of developing ever more realistic games has grown so huge that even game developers can no longer rely on recouping their entire investment from a single game. This has led to the emergence of *game engines*—modular simulation code—written for a specific game but general enough to be used for a family of similar games. This separability of function from content is what now allows game code to be used for more general simulation (Lewis and Jacobson 2002).

The game's engine refers to the collection of modules of simulation code that do not directly specify the game's behavior (game logic) or game's environment (level data). The engine includes modules handling input, output (3D rendering, 2D drawing, sound), networking and generic physics and dynamics. The *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, display parameters, networking, and the base or atomic-level actions for animations and can be modified using a game-specific scripting language. 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.

2.1 The Unreal Engine

Our simulations of the NIST USAR arenas are based on the updated version of the Unreal Engine released by Epic Games with Unreal Tournament 2003. The simulation is written as a combination of *levels*, describing the 3-D layout of the arenas and *modifications*, scripts redefining the simulation's behavior. The engine to run the simulation can be inexpensively obtained by buying the game. The Unreal Engine is an excellent platform for rapid prototyping because it provides a sophisticated graphical development environment and a variety of specialized tools including the *Karma* physics engine and a skeletal animation system which simplify the detailed tasks of modeling physical processes. In this paper we discuss the ways in which we have used these tools to create a USAR simulation meeting our earlier desiderata.

2.1.1 Unreal Client-Server Architecture

The client-server model used in Unreal is the "generalized client-server model". The server controls the interaction among clients and the authoritative state of the simulation. On the client side, the client sends data to the server about its actions. It then displays to the user changes it has been given by the server. The client locally maintains a subset of the simulation state, which can be used to predict subsequent states. It executes the same code as the server but according to its local state. Thus, the client can approximately predict the next simulation state. The prediction technology can increase visible detail while lowering bandwidth usage, eliminate perceived latency in client movement and decrease the amount of data that needs to be exchanged between the clients and server. The client gets the simulation state from the server through "replication". In Unreal, replication deals with how information is sent from the server to the client and vice versa. Only a subset of the simulation state that affects a particular client is sent to it.

2.1.2 GameBots Modification

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 player's machine) and 'bots' (short for robots) simulated players running simple reactive programs. Gamebots is a modification to the Unreal Tournament game that allows bots to be controlled through a normal TCP/IP socket (Kaminka et al. 2002). Gamebots talks to the game engine directly, and opens its own networking sockets. A protocol for interacting with Unreal Tournament is defined in (Gamebots 2003). With a simple text-based TCP/IP protocol Gamebots can be used to create and manipulate bots in an Unreal Tournament instance. 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 in the same fashion as an actual one.

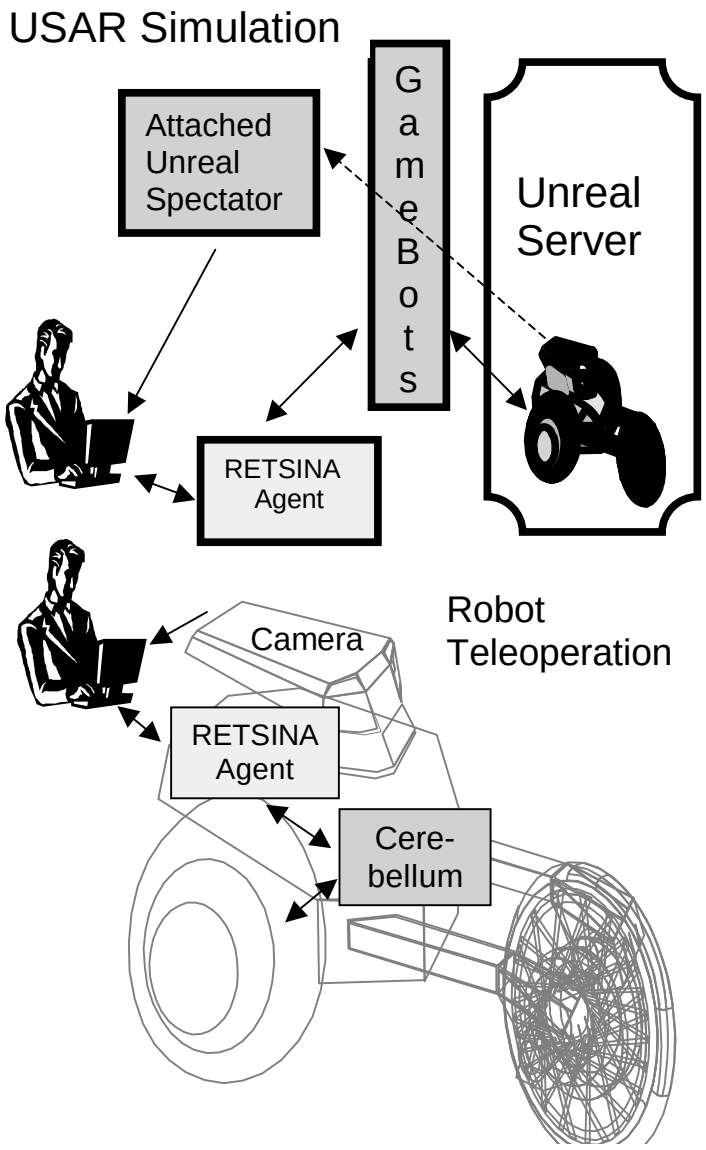

Figure 2: USAR simulation & teleoperation architectures

2.1.3 USAR Simulation Architecture

One of the client options in Unreal is the 'spectate' mode. As a spectator, the client's viewpoint (camera location and orientation from which the simulation is viewed) can 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. In the larger project both simulated and actual robots will be controlled through RETSINA agents (Sycara et al. 1996), modular agents with communication, planning, and execution monitoring capabilities. Under this architecture planning, cooperation, communication, and control are the same for actual and simulated robots. Figure 2 illustrates this arrangement.

3 MODELING THE ORANGE ARENA

The Orange Arena is a simulated collapsed building. The robot in this scene needs to move around in the building, try to avoid the obstacles and find the victims. Simulation objects in the arena are divided into three categories: static geometric objects, dynamic geometric objects and environmental objects.

1. *Static geometric objects*
   These objects are part of the building that is unmovable. They affect how the robot moves around in the building. The material of the object may affect the robot's perception. For example, the glass may affect the robot's perception of distance; the texture of a wall may affect the robot's judgment of target.
2. *Dynamic geometric objects*
   This kind of objects can change their own states. They may be bricks, rubbles, pipes, victims etc. They have more complex interaction relationship with the robot than the static geometric objects. When they interact with the robot, they may change their states such as position, gesture etc.
3. *Environmental objects*
   The environmental objects describe the ambient conditions that make up the environment. They include lighting, sound, and other intangible features.

The arena model has three layers corresponding to the types of objects: a geometric layer, a dynamic layer and an environment layer.

The geometry layer includes all the static geometric objects. This layer was built from Pro-Engineer solid models provided by NIST. The model was simplified using NuGraf and exported in 3D Studio format. The simplified model was then imported into the Unreal Tournament Editor to provide static geometry for the simulation.

Digital photographs and other data collected by a project team which visited the permanent installation at NIST were used to embellish the bare geometry of the Orange Arena to produce a simulation model difficult to distinguish visually from the original. The realistic model was created by mixing static textures such as wallpaper, wood surface, or brick surface with dynamic textures capable of changing according to their lighting, for example, glass, mirrors and the guardrail.

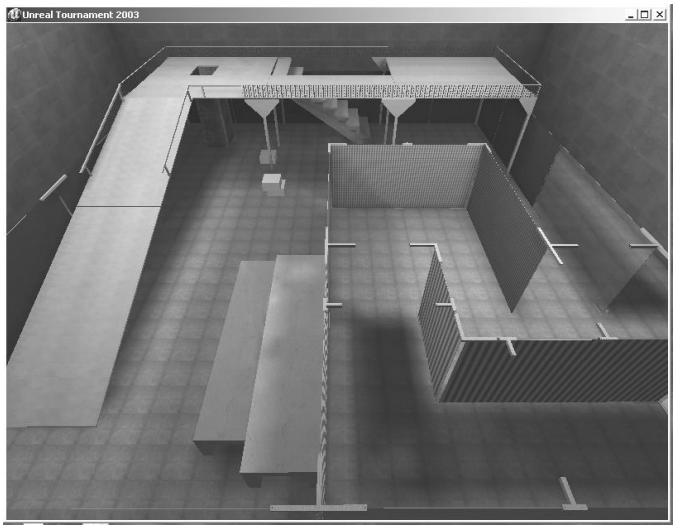

Figure 3: Simulated Orange Arena without rubble

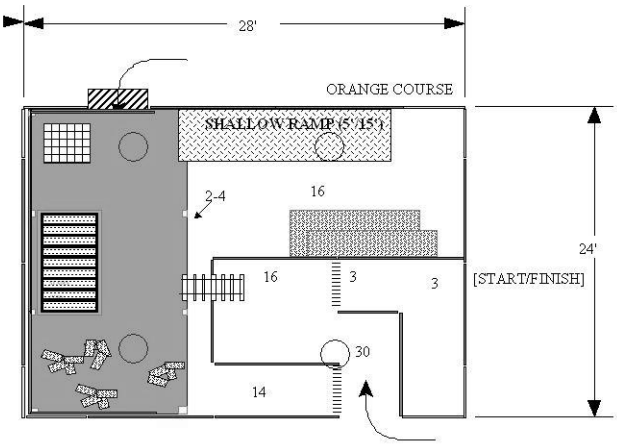

Figure 4: Lighting levels in Orange Arena

The dynamic layer is a layer of dynamic objects that include Unreal classes written in Unreal Script to simulate interaction with these objects.. The interactions are modeled using the *Karma* physics engine which will be introduced in the next section.

Two classes of dynamic objects were developed for this simulation: obstacle and victim. The obstacle class extends the *decoration* class of Unreal Tournament. According to the mass of an object that collides with it, the obstacle makes an appropriate response. The victim class is extended from the Unreal Tournament *xIntroPawn* class. With the skeletal modeling system, the victim can move its hand or arm to simulate the gesture of asking for help. Figure 2 is the picture of the obstacles and victims.

The environmental layer is presently limited to lighting effects and specifies light sources to reproduce ambient lighting levels and contrasts (15:1) resembling those found in the arena (figure 4). The resulting model faithfully reproduces both the geometry of the Orange Arena and its appearance through a simulated robot mounted camera.

Figure 5: Victim and fencing from real arena above with similar scene from simulation below

4 MODELING DYNAMICS

In the real world, objects interact with each other according to the laws of physics. Because difficulties in teleoperation and locomotion are significant problems in the USAR domain it is important to model these aspects of the domain as accurately as possible. The current release of the Unreal engine as well as Sony Playstation and the Microsoft Xbox all use the *Karma* physics engine to simulate interactions among solid objects such as crates, tires or bones, as well as different joints, motors or springs that make up mechanical objects.

4.1 Karma engine

The *Karma* engine is a rigid-body physics engine developed by MathEngine. It provides physics modeling, rigid-body dynamics with constraints and collision detection. Using a variety of computational strategies to simplify, speed up, and exclude non interacting objects it achieves animation level speed without sacrificing physical fidelity. Each simulated object has its own mass and inertia, obeys Newton's laws of motion, and interacts with other objects according to mass, inertia, friction, restitution, and gravity.

Every object has kinematic attributes that describe its position and movement, such as: position of the center of mass, orientation of the body, acceleration/velocity of the center of mass and angular acceleration/velocity, which describes the change of orientation. Forces and torques are the dynamic attributes used in Karma. Constraints are used to describe the restriction on the motion of an object. There are two types of constraints: joints and contacts. Joint attaches two objects and restricts one or more of the degrees of freedom between them. 14 joint types such as *Ball And Socket*, *Cone Limit*, *Hinge*, and *Car Wheel Joint* are provided in *Karma*. With joints, two or more rigid bodies can construct a multi-rigid object such as four wheeled vehicles or human bodies. Contacts limit how an object can move. For example, when a stone falls onto the floor, it will strike the floor and rebound back. Karma uses collision detection to detect whether two bodies are in contact and supports collisions between geometries of a variety of types.

4.2 Robot Dynamics

Simulating robot dynamics is greatly simplified by the Unreal engine's *vehicle* class which uses the *Karma* engine's *Car Wheel Constraint* joint A *vehicle* is made of a chassis, one or more wheels and the joints that connect the wheels to the chassis. The wheel rotates about its rolling or hinge axis; the chassis travels along the suspension direction and the wheels steer about the steering axis. The vehicle is driven by a motor whose output torque is provided by interpolating along its Torque-SpinSpeed curve.

Our initial arena robot is a *vehicle* class object mirroring the design of actual robots being built by other researchers on our project. The robot is driven by two wheels each with its own motor. The steering axis of the vehicle class has been locked to cause the simulated robot to turn by driving the two wheels with different spin speeds as in the actual robots' design.

Within Unreal there are two ways to simulate physical events: scripted behavior (animation), and bespoke solutions. Scripted behaviors control the movement of an object according to a predefined sequence of events. Bespoke solutions generate movements by applying the appropriate mathematical equations. These two approaches can be used together or switched according to context. Most of our robot simulation depends on bespoke solutions controlled by the Karma engine. However, in certain situations such as crossing a floor strewn several inches deep with paper it would be prohibitively expensive to model the geometry of individual sheets so scripted behavior is used instead.
To solve this problem Unreal Scripts were used to randomly change the floor friction to simulate slippage as the robot crosses the paper.

5 MODELING SENSORS

While simulating a video feed and robot dynamics allows us to investigate teleoperation and robot design it cannot model autonomous behavior that requires input from the environment. There has been much work on modeling sensors for virtual environments and the Unreal engine has many of the features needed to generate sensor data. Simulating sensors has little to do with their actual operation but instead involves degrading the perfect knowledge available from the simulation to resemble that available through noisy and imperfect sensors. The challenge is to the mimic the forms of distortion found for different classes of sensors, for example, blind spots due to specular reflections for sonar.

5.1 Proximity Sensors

The term proximity sensor refers to sensors that provide location data on objects relative to the sensing body. One of the oldest and most well known proximity sensors is sonar. Sonar operates by transmitting a pulse wave out into the environment. As the wave collides with objects it is reflected back to the transmitter. Based on the received signal the transmitter can estimate the distance to the foreign object. Every entity in Unreal has a vector representing its location as a triple: X-coordinate, Y-coordinate, and Z-coordinate. The *Trace* function will trace a line through the environment and return a reference to the first object the line collides with be it a wall, another entity, or some other obstruction. The location data returned can be manipulated in much the same way sonar data is. Sonar can be simulated by restricting the distance and accuracy of Trace data provided to the robot controller.

5.2 Laser based sensors

Laser based sensors use lasers to scan an object and assimilate the data gathered into a three dimensional image. An example of a laser based sensor that is becoming popular is laser radar (LADAR). LADAR provides detailed ranging data which is sometimes enhanced to produce near photographic quality imagery. LADAR is much better adapted to automatic target recognition (ATR) than intensity based imaging because the location and shapes of objects are represented unambiguously. LADAR based ATR can be simulated using the *Trace* function in much the same way as sonar. In this case, however, entity types for classes having ATR templates can be returned from a confusion matrix to simulate inaccuracies in sensing and recognition. LADAR imagery can be simulated for human viewing by applying a grayscale filter to visible imagery. While the resulting images resemble LADAR imagery they do not preserve its other characteristics.

5.3 Thermal sensors

Thermal sensors measure temperature differences in the environment and convert that information into a presentable format. Forward looking infrared (FLIR) is a widely used form of thermal sensing that generates images showing thermal differences. However, FLIR is problematic to simulate as it requires input from many variables, not all of which are present in our simulation. These variables include solar radiation, weather, internal heat, and the thermodynamics of physical materials all of which must be considered in order to accurately predict temperature. For robot use, FLIR is less capable than LADAR because it provides even less unambiguous detail than visual imagery. For robots using FLIR to locate and move toward sources of heat which may signal potential victims, this functionality can easily be simulated using the Trace function with suitable degradations for distance and noise. Displaying FLIR or fused imagery incorporating FLIR is more difficult and might require retexturing the entire arena to reflect the scene as viewed through FLIR.

Sensors have not yet been added to our USAR simulation because of the wide variation in their characteristics. We expect several sonar units to be installed on the robots being built for use in a safeguarded (robot self protection) operation mode and will add them to the simulation when they are selected.

6 PRELIMINARY USES

Portions of the USAR simulation have already been used in a series of teleoperation experiments involving camera control (Hughes et al. 2003) and gravity referenced attitude displays (Lewis et al. 2003). The full USAR simulation was publicly demonstrated at the First Robocup American Open held at Carnegie Mellon University April 30- May 4, 2003. In conjunction with regularly scheduled exhibitions by USAR teams in the Orange Arena, attendees were allowed to search for victims in the simulation using the same interface that controlled the "corky" team's robots. The simulation is currently being used to evaluate proposed changes to this interface.

7 DISCUSSION

Until recently accurate interactive virtual environment simulations were expensive, time consuming, and difficult to construct. The USAR simulation described in the paper, by contrast was built in less than three months and already meets the requirements we had laid out for it. While several specialized graphics packages, 3D Studio Max and NuGraf were used to speed development, the lion's share of work was done using tools provided with a fifty dollar video game on a conventional personal computer.

The simulation architecture described in section 2.1.3 is well suited for the USAR robotics community because it allows researchers to test the aspects of physical or algorithmic design in which they are interested without requiring other supporting implementation. The simulation is already being used for human factors research in teleoperation and perceptual search where it is particularly powerful due to the excellence and control over graphics provided by a game engine. As development proceeds we hope to provide user friendly tools to allow researchers to assemble new designs and program the needed behaviors with less effort. For other uses (teleoperation, perceptual search, autonomous and team behaviors) the simulation is already extremely easy to set up and use.

ACKNOWLEDGMENTS

This project is supported by NSF grant NSF-ITR-0205526. Katia Sycara and Illah Nourkbaksh of Carnegie Mellon University are co-pi's on the project. Mary Berna, Alexander Gutierrez, Terence Keegan, Kevin Oishi, Binoy Shah, Steven Shamlian, Mark Yong, and Josh Young assisted in data collection.

REFERENCES

Casper, J. 2002. Human-Robot Interactions during the Robot-Assisted Urban Search and Rescue Response at the World Trade Center, MS Thesis, Computer Science and Engineering, USF, Apr. 2002.

Gamebots 2003. Network API. http://www.cs.cmu.edu/-galk/GameBots/WEB/docapi.html.

Hughes, S., Manojlovich, J., Lewis, M., and Gennari, J. 2003, Camera Control and Decoupled Motion for Teleoperation. to appear in Proceedings of the 2003 IEEE International Conference on Systems, Man, & Cybernetics.

Jacoff, A., Messina, E., Evans, J. 2001. Experiences in deploying test arenas for autonomous mobile robots, Proceedings of the 2001 Performance Metrics for Intelligent Systems (PerMIS) Workshop, Mexico City, Mexico.

Kaminka, G., Veloso, M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A., Scholer, A., and Tejada, S. 2002. GameBots:A flexible test bed for multiagent team research, Communications of the Association for Computing Machinery(CACM),NY:ACM45(1),43-45.

Lewis, M. & Jacobson, J. 2002. Game Engines in Research. Communications of the Association for Computing Machinery (CACM), NY: ACM 45(1), 27-48.

Lewis, M., Wang, J., Manojlovich, J., Hughes, S., and Liu, X. 2003. Experiments with Attitude: Attitude Displays for Teleoperation to appear in Proceedings of the 2003 IEEE International Conference on Systems, Man, & Cybernetics

Murphy, R. 2003. Gaps in Rescue Robotics, presentation to IEEE Workshop on Safety, Security, and Rescue Robotics, USF, Tampa, FL, Feb 19, 2003.

Sycara, K., Decker, K., Pannu, A., Williamson, M. and Zeng, D. 1996. Distributed Intelligent Agents. *IEEE-Expert*; vol 11; number 6; 36-45; 1996

AUTHOR BIOGRAPHY

JIJUN WANG is a Ph.D. student in the School of Information Sciences at the University of Pittsburgh. He has B.S. and M.S. degrees in Electo-Mechanical Engineering from Tsinghua University of China. Before entering the University of Pittsburgh, he spent 3 years in the IT industry. <jiw1+@pitt.edu>

MICHAEL LEWIS is an Associate Professor in the School of Information Sciences at the University of Pittsburgh. He received his Ph.D. and M.S. degrees from the Georgia Institute of Technology. His current research involves information fusion and human control of mixed-initiative systems. <ml@sis.pitt.edu>

JEFFREY GENNARI is a MSIS student in the School of Information Sciences at the University of Pittsburgh where he also received his BSIS degree.
<jgennari@mail.sis.pitt.edu>