A SUPERVISORY CONTROL INTERFACE FOR LARGE MOBILE ROBOT TEAMS

William D. Smart and Timothy M. Blakely

Department of Computer Science and Engineering
Washington University in St. Louis
One Brookings Drive
St. Louis, MO 63130
United States
{wds,tmb1}@cse.wustl.edu

There is a pressing need for supervisory control interfaces that allow single human operators to effectively control large numbers of robots or unmanned vehicles. Although increasing levels of autonomy make it possible to control robots at higher-and-higher levels, we must still be able to respond to unforeseen problems with low-level intervention. In this paper, we present RIDE, a supervisory control interface for large teams of mobile robots. RIDE allows task-level control of robot teams, and includes a sliding autonomy mechanism that allows robots to request help from a supervising human when they cannot complete their assigned task.

I. INTRODUCTION

There is a pressing need for supervisory control interfaces that allow single human operators to effectively control large numbers of robots or unmanned vehicles. The increasing levels of autonomy being demonstrated by these systems allow us to control them at higher levels of abstraction, moving away from direct tele-operation, and toward task-level control. However, this autonomy is not perfect, and there is still a need to be able to supply lower-level commands to these systems when the autonomous behaviors are unable to cope.

Effective interfaces for the control of large teams of autonomous systems should allow both high-level supervisory control and more direct, low-level control of the systems. The interface should integrate the idea of sliding autonomy, where both the human operator and the unmanned systems are allowed to alter the level of autonomy. Human supervisors can raise or lower the level of autonomy, based on the current circumstances. The unmanned system can lower the level, essentially asking for help, when it detects a failure to complete the current task.

As the level of autonomy changes, the amount and type of information required by the human supervisor will also vary. When directly controlling the systems, visualizations of low-level sensor readings might be appropriate. However, as the level of autonomy and number of robots increases, the supervisor will need higher and higher levels of abstraction in the information displayed. A good interface should allow the supervisor to easily configure the level of detail shown, to provide the most relevant visualization.

The requirements for such a supervisory control interface are almost exactly the same as those for Real-Time Strategy (RTS) computer games. Developers of these games have spent the last two decades developing natural tools for directing large teams of heterogeneous agents operating in increasingly realistic virtual worlds. These games are typically resource allocation games, where players control heterogeneous collections of units (humans, vehicles, buildings, and weapons) to achieve some victory condition (such as amassing a certain amount of a particular resource, or defeating all enemy units). Orders can be given to the units at the task level (from a discrete set, such as “mine gold”, “attack enemy”, or “build wall”), and also at the tactical level (choosing from discrete behaviors, such as “patrol area”, “follow waypoints”, or “move in formation”). Players generally provide direction at the task level, and only intervene when the assigned tasks cannot be completed successfully, either due to the failure of an autonomous behavior or because of some unforeseen circumstances.

In this paper, we describe a prototype supervisory control interface for large teams of mobile robots that incorporates elements from RTS game interfaces. Human operators can task subsets of the robot team, based on capabilities of the individual systems, and can manipulate the levels of autonomy and visualization detail as the situation requires. We describe our implemented system, and present some preliminary evidence to show its practical utility. We begin, however, with a description of the features common in RTS games, for the reader unfamiliar with the genre.

Fig. 1. Microsoft's Age of Empires II real-time strategy game interface[1].

II. REAL-TIME STRATEGY GAME INTERFACES

Figure 1 shows a screenshot from Microsoft's Age of Empires II, a popular RTS resource-management game, representative of the current generation of the genre. Players must manage up to 200 units, operating over a large area, to achieve a variety of victory conditions. Up to four players can play against (or in cooperation with) each other over a network.

The main area of the interface shows an iconic, isometric view of the world. Units, terrain features, and game information (such as navigation way-points) are displayed here. The user can select single units or groups with keyboard and mouse commands. These units are then tasked by selecting from the context-sensitive task menu at the bottom left. A bar with status information appears along the top of the main window.

When one or more units are selected, a list of possible tasks is displayed in the bottom left window, based on the capabilities of the selected units. Clicking on one of the task icons assigns that task to all of the currently selected units. Details of the units, including current status, designation, and capabilities are also displayed in an information window.

The main display is a zoomed-in view of the larger area over which the game is played. Keyboard and mouse commands can move this view around. A smaller display in the bottom right shows the relationship between the zoomed-in view and the rest of the (bounded) world.[†] When important events happen out of view of the player, they are announced on this larger-scale map. This gives a heightened sense of situational awareness, and allows quick navigation to other parts of the world. The map also contains summary information about certain objects in the world, such as unit dispositions (the red and blue dots). Clicking on the large-scale map moves the zoomed-in view to that location in the world.

Once tasked, the units act autonomously, alerting the player when they are finished (and become idle), when they are unable to complete a task, or when some exceptional situation occurs, such as sighting enemy troops. The player can then re-task the units to deal with the situation. As long as these exceptions happen relatively infrequently, they are easily dealt with. The combination of large-scale and small-scale displays helps the player maintain situational awareness, and the iconic representation of the world allows a quick assessment of the current situation when a problem arises.

It is worth noting here that "real-time" in the context of video games means that players move their units at the same time, as opposed to taking discrete turns, as in traditional table-based wargames. This meaning has nothing to do with the very different definition of "real-time" in the computer science and robotics literatures.

III. RTS INTERFACES FOR MOBILE ROBOTS

Figure 2 shows the prototype version of RIDE, the Robot Interactive Display Environment, our supervisory control interface for large teams of mobile robots. The main window shows an iconic view of the world, just as in RTS games. The ground plane is shown as a 1m grid. Additional GUI elements allow control over the camera position, display information about the selected robots, and allow selected robots to be tasked by the operator.

The interface is modal, allowing the user to either control the camera position and view direction, or to select and task the robots. The current mode is displayed to the right of the main visualization window. The camera position and view direction is controlled using the keyboard and mouse, allowing the user to view the world from any angle, including a traditional isometric viewpoint. The camera can also be attached to a particular robot, giving a "robot eye view" of the world. The speed of camera movement is controlled by the slider in the upper right of the interface.

In selection mode, individual robots can be selected by directly clicking on them in the viewing window. When a robot is selected, it is highlighted, as shown in figure 2, where the five robots near the middle window are selected. Information about currently selected robots also appears in the information window, to the right of the viewing window. Multiple robots can be selected by holding down the control key while clicking on them individually, or by using the mouse to drag a bounding box around the required systems.

[†] This is shown as a white rectangle in the figure, which might not reproduce well in this paper.

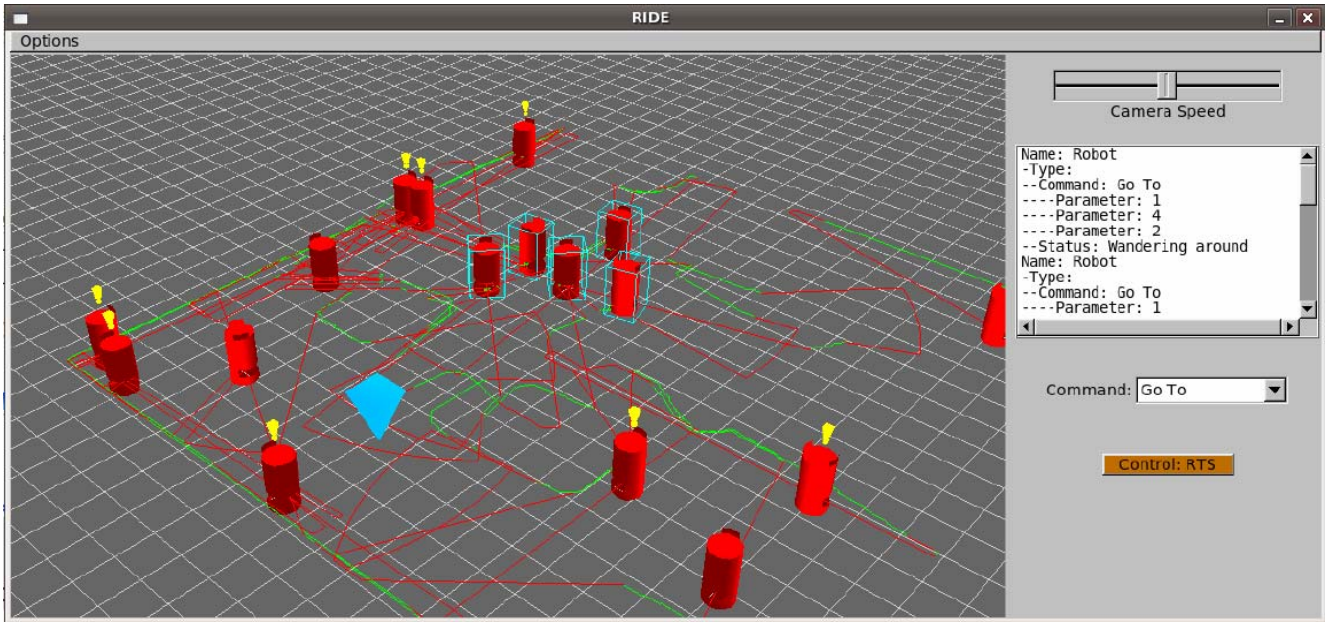

Fig. 2. RIDE, the Robot Interactive Display Environment.

When one or more robots are selected, the possible tasks are displayed in a pull-down menu on the bottom right of the interface. After the user selects a task from this menu, the task parameters can then be entered via the viewing window. For example, in figure 2, five robots are selected, and the user has chosen the "go to" task. The parameter to this task is a location in the world, which the user denotes by directly clicking on a position in the viewing window. This position is shown as a blue inverted pyramid in the figure. Once the task is selected and the parameters (if any) are given, the high-level commands are sent to the individual robots. Each of the robots is assumed to have behaviors that are capable of completing the task successfully.

The interface can visualize both raw sensor data from the robots, and higher-level internal data structures such as maps. All of the data visualizations can be toggled on and off by the user, as needed. This has proven to be extremely useful since, as shown in figure 2, displaying sensor data from a large number of robots can make for a very cluttered image. Data visualization settings can be chosen from the options menu in the toolbar.

RIDE implements a shared autonomy system by allowing the robots to request help from the supervisory. This is done by displaying a large yellow "!" above the robot in the interface, as shown in figure 2. This can be accompanied by an audible warning, to alert the supervisor.

III.A. Current Implementation Status

RIDE currently supports a subset of the sensors and behaviors available on our robot platforms. We are currently working on adding to this set but, at the time of writing, we support the following features.

III.A.1. Robot and Sensor Visualization

We currently support two popular research robot platforms: the iRobot B21r and the ActivMedia Pioneer III. It also supports visualization of the basic sensors available for these systems: SICK laser range-finder, sonar, and bump sensors. These visualizations are augmented to make them more meaningful to a human operator. For example, figure 2 shows a number of B21r robots, with SICK PLS laser range-finder visualizations enabled. Each PLS device returns 180 radial distance measurements, spaced one degree apart. If two laser contact points are within a small distance of each other, they are joined by a bright green line in the visualization. Otherwise, they are joined by a thin red line. As can be seen from the figure, this tends to pick out objects (walls in this case), allowing the operator to see them much more clearly (without the computational cost of maintaining a real shared map).

III.A.2. Data Visualizations

We support the display of learned occupancy grid maps, as shown in figure 3. Maps can either be visualized

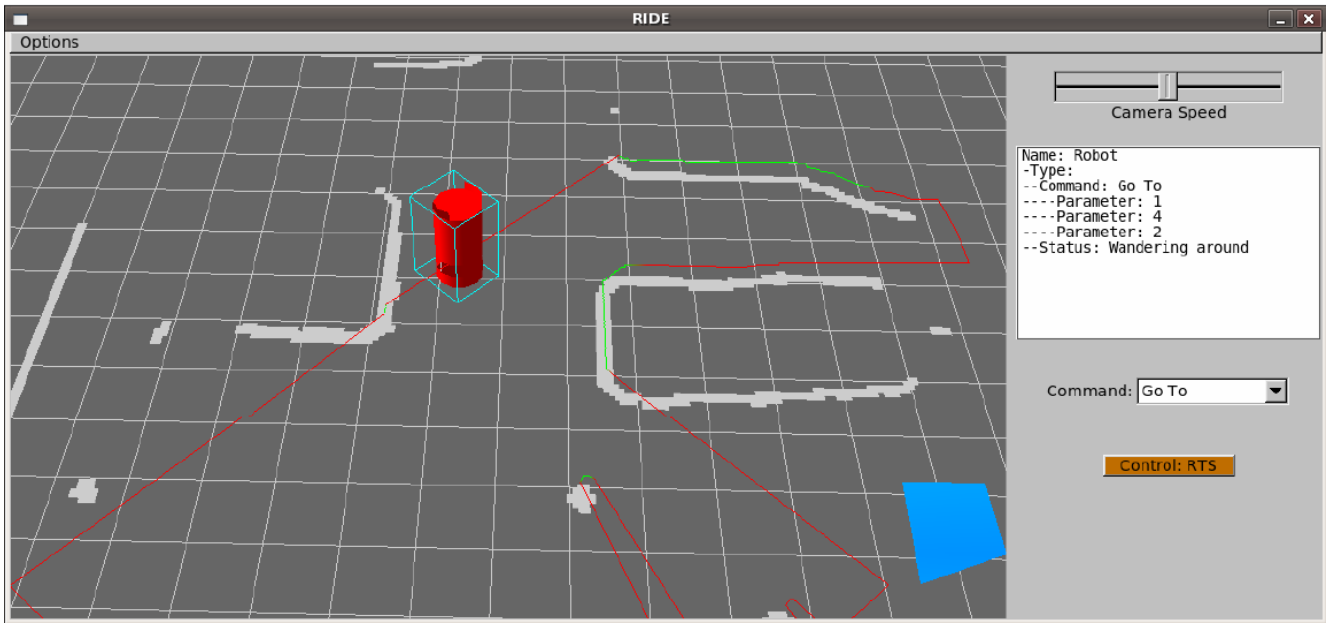

Fig. 3. Two-dimensional map visualization with a single robot. The robot has been selected (blue enclosing lines), is driving to a waypoint (blue inverted pyramid), and is displaying the data from a scanning laser range-finder (red lines).

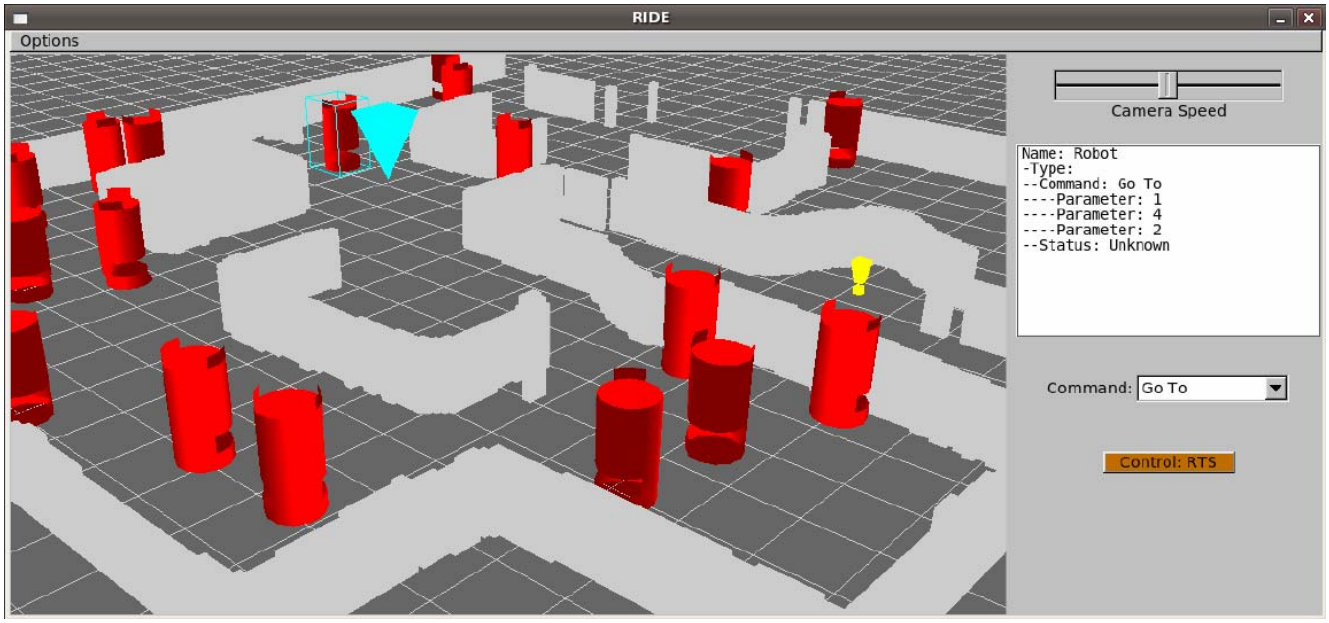

Fig. 4. Three-dimensional map visualization with several robots. One robot is selected, close to a waypoint. Another robot is in need of help, and is displaying a yellow "!".

two-dimensionally, as in figure 3, or three-dimensionally, looking like walls, as shown in figure 4. Notice also that, in this figure, there is a robot asking for assistance on the right of the main viewing window.

III.A.3. Tasks

At the time of writing, we support two tasks: go to a particular point, and follow a set of waypoints. These direct the selected robots to drive to a particular place in the world. There is no coordination between the robots.

We are also currently adding support for perimeter-following, and formation-keeping.

III.B. Implementation Details

RIDE is implemented in C++ using FLTK, a multi-platform graphical toolkit that allows it to run on a wide variety of platforms[2]. The rendering of the main viewing window is done using OpenGL, which allows us to implement advanced perspective, lighting, and texture features easily. It also allows us to efficiently display a large number of objects, since the OpenGL system can take advantage of hardware accelerations on the graphics card.

New objects in the world view (robots, sensors, and data visualizations) are easily added to the system using the C++ inheritance system, and by specifying how they are drawn in OpenGL.

The interface communicates with the robots over a standard TCP/IP socket, sending tasks and receiving sensor data and requests for assistance. The robots themselves are programmed with the Player[3] robot control framework. It is worth underscoring here that the interface itself offers no additional guarantees of performance or task-completion than does the software on the robots. It does not explicitly monitor the robots and their behavior, but is only a mechanism for a skilled operator to interact with many robots at once.

IV. INITIAL VALIDATION RESULTS

In this section, we present the results of some initial validation trials with the RIDE interface that illustrate its potential utility. The test domain is a collaborative mapping task, involving eighteen robots. The robots are programmed to randomly wander, unless directed to drive to a waypoint. While driving towards a waypoint, the robots do not perform path planning, or sophisticated obstacle avoidance.

Our rationale behind such primitive robot behavior is simple: we want the robots to get stuck often, and to have trouble making progress without intermittent, but frequent help. This will exaggerate the benefits of our interface somewhat but will, we believe, show that a single user can successfully control a large number of robots constantly seeking attention. This simple test case is also meant to be a surrogate for other, more complex scenarios. As the worlds in which the robots must operate become more complex, the autonomous software that they employ must become more sophisticated. However, we argue, that the ability of the software to deal with the world will always lag behind the ability of the world to thwart the robot (at least for the time-being). Our interface is designed precisely for these situations, regardless of the absolute complexities of the world and the robots.

We performed two experiments, using the Stage simulator[3], and used the area successfully mapped as our metric of success. In the first experiment, the eighteen robots wandered randomly about, building a map without human intervention, for five minutes. The map learned during the first experiment is shown in figure 5. We purposefully start the robots in an area that is difficult to escape from by executing a random walk to ensure that they will get stuck early and often. It could be argued that this is stacking the deck in our evaluation, since we are purposefully putting the robots in a situation that they will be hard-pressed to escape from. However, it could also be argued that this is just the type of worst-case scenario that we need to test our interface on. RIDE will have little utility, other than as a visualization tool, in environments where the robots need little assistance, and can operate autonomously for long periods of time.

In the second experiment, the human supervisor responded to requests for help from robots that found themselves stuck. The supervisor selects the robot in need of assistance, and tasks it to drive to a nearby waypoint, moving it away from the area in which it got stuck. The operator only did this for robots that recognized that they had become stuck (detected by remaining in more-or-less the same place for more than 10 seconds). In cases where several robots near each other request help, the operator can select them all, and task them with driving to the same waypoint. Again, we ran the experiment for five minutes. The resultant map is shown in figure 6. Notice the different scales on the displayed maps. Not surprisingly, the robots that benefited from supervisory help explored and mapped a much larger area of the world. Notice that most of the robots in figure 6 are requesting help (displaying a yellow "!"), because of their poor obstacle avoidance abilities. Although it is hard to draw any hard conclusions from such a simple experiment, we believe that it demonstrates that a single human operator can successfully provide supervisory control to a moderately large number of robot systems.

V. RELATED WORK

Although there are a number of interfaces designed for the direct teleoperation of a single robot system, there are surprisingly few for the supervisory control of a large team of robots. However, we are not the first to propose an RTS interface for robot control. Jones and Snyder[4], describe a system that is very similar to ours, although it is designed for the control of a small number of free-flying space robots. Our system differs from theirs in our extensive use of sensor and data visualizations, and in our use of a sliding autonomy system that allows the robots to request assistance.

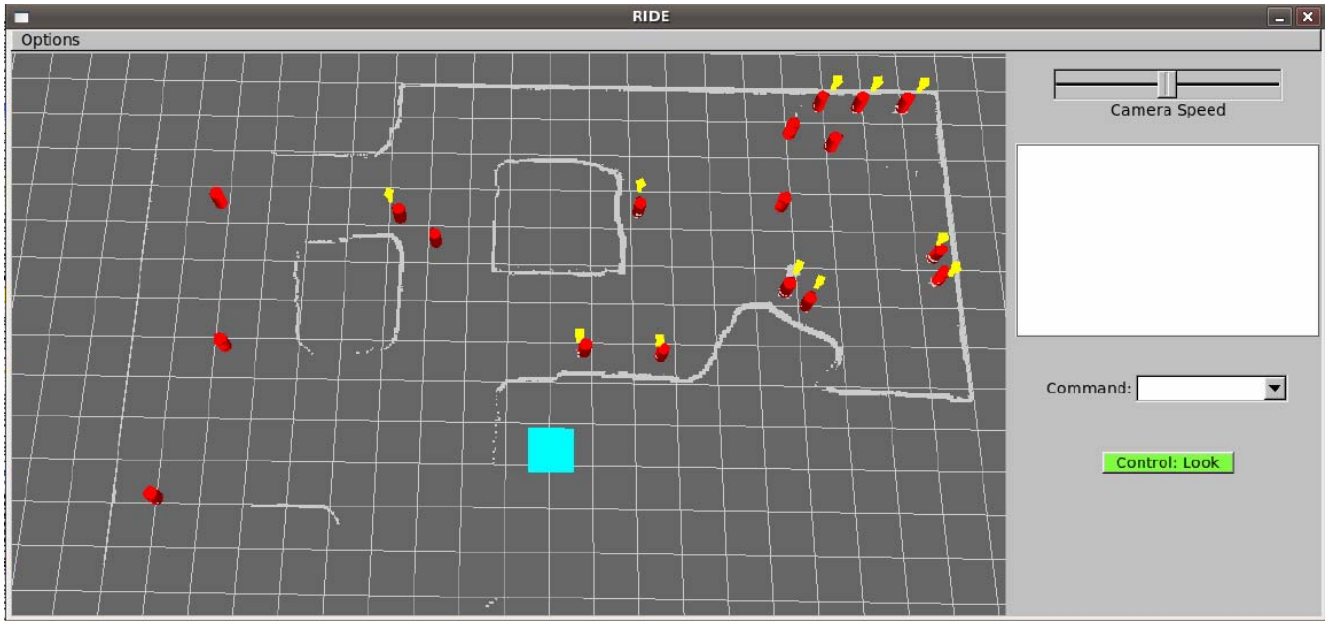

Fig. 5. The map learned without human intervention. The robots start in the upper right-hand corner of the map, but many get suck there (yellow "!"s).

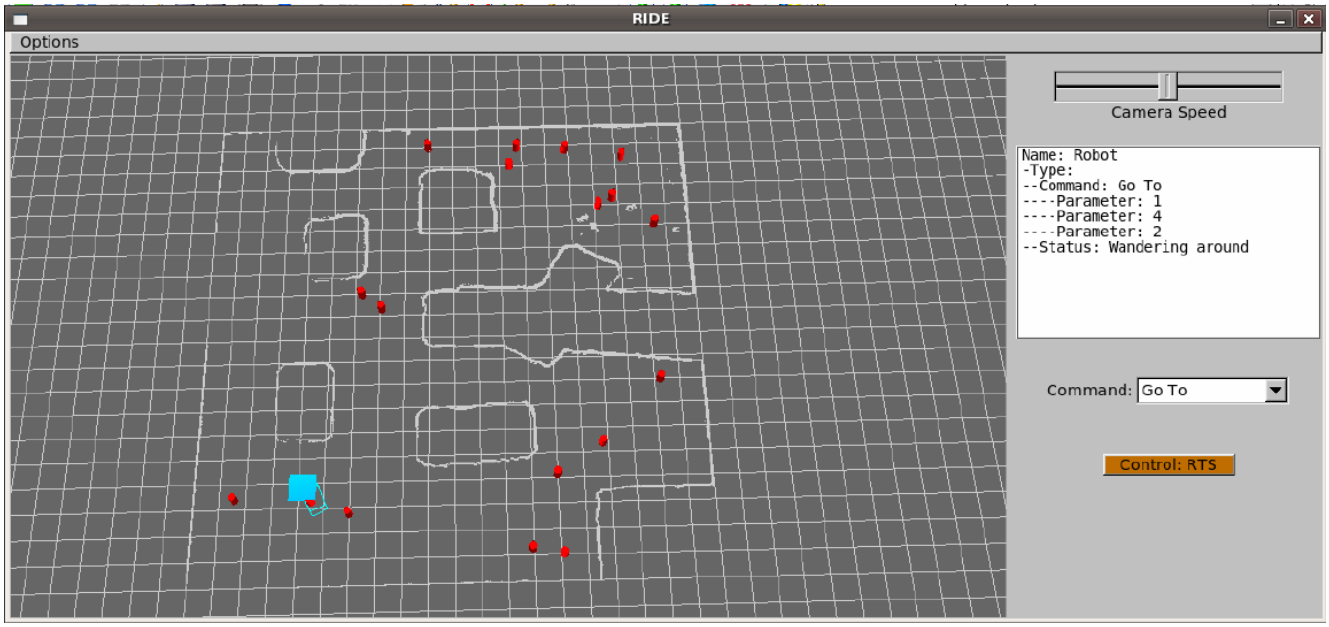

Fig. 6. The map learned with supervisory assistance. Notice the difference in scale, compare to figure 5. No robots are currently stuck, although one is being given a waypoint (blue inverted pyramid, lower left-hand corner)

Parasuraman, Galster, and Miller describe a task-level control interface called Playbook[5], and evaluate its effectiveness on a simulated unmanned vehicle control task. Subjects controlled six simulated vehicles under a range of conditions. The interface was a two-dimensional representation of the world, where the only sensor visualization was a representation of the robot's field of view.

A number of robot simulations take advantage of first-person computer games technology (for example, the USAR simulation[6], but typically do not take advantage of the associated user interfaces. The game-based simulator described by Faust, Simon, and Smart[7], on the other hand,

allows humans to directly control human avatars, using a traditional first-person game interface, but this is still essentially a direct tele-operation interface.

VI. DISCUSSION AND CURRENT WORK

In this paper, we presented the prototype of a Real-Time Strategy game interface for the supervisory control of large teams of mobile robots. The interface allows task-level control of a large number of robot systems by a single operator, visualizations of sensor data with varying levels of detail, and a basic sliding autonomy mechanism.

Although our initial studies have yielded very encouraging results, the interface needs to be evaluated more rigorously. It is still unclear exactly how many systems a single operator can effectively control under realistic conditions. In particular, if we have more self-sufficient robots, performing more complex tasks, will the operator be able to successfully maintain situational awareness as he is force to switch his attention from one robot (and task) to another.

Our current implementation includes only a subset of our robots' sensors and behaviors. We are currently working on adding in visualizations for the remaining sensors, and tasking for the additional behaviors. In particular, we are currently working on the addition of perimeter-patrolling and area-search behaviors, and the ability to group robots into formations.

We are also actively working on the graphical interface. In particular, we are implementing a large-scale view of the world, as shown in figure 1, to improve the operator's situational awareness. This is especially important in cases where a robot requests assistance from the supervisor. We are also working on making the interface more aesthetically appealing, and easier to use with, for example, icons rather than drop-down menus.

One interesting direction that we hope to explore is the control of mixed robot and human teams using the RIDE interface. Humans will carry a portable computer equipped with a GPS unit and headphone. They will appear icons in the interface, just as the robots do. When tasks are assigned to the human units, they will be translated into verbal commands, such as "go to waypoint 23", or "follow the robot in front of you", and sent to the appropriate portable computer.

REFERENCES

1. Microsoft Games. Age of Empires II: Age of Kings. http://www.microsoft.com/games/age2
2. The Fast Light Toolkit (FLTK). http://www.fltk.org/
3. B. P. Gerkey, R. T. Vaughan, K. Støy, A. Howard, G. S. Sukhatme, and M. J. Matarić, "Most Valuable Player: A Robot Device Server for Distributed Control," *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1226-1231 (2001).
4. H. Jones and M. Snyder, "Supervisory Control of Multiple Robots Based on a Real-Time Strategy Game Interaction Paradigm," *IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 1, pp. 383-388 (2001).
5. R. Parasuraman, S. Gastler, and C. Miller, "Human Control of Multiple Robots in the RoboFlag Simulation Environment," *IEEE International Conference on Systems, Man, and Cybernetics*, Vol. 4, pp. 3232-3237 (2003).
6. J. Wang, M. Lewis, and J. Gennari, "USAR: A Game Based Simulation for Teleoperation." *47th Annual Meeting of the Human Factors and Ergonomics Society* (2003).
7. J. Faust, C. Simon, and W. D. Smart, "A Video Game-Based Mobile Robot Simulation Environment," IEEE/RSJ *International Conference on Intelligent Robots and Systems (IROS)*, pp. 3749-3754 (2006).