RAVE: A Real and Virtual Environment for Multiple Mobile Robot Systems

Kevin Dixon John Dolan Wesley Huang Christiaan Paredis Pradeep Khosla

Institute for Complex Engineered Systems
Carnegie Mellon University

Prepared for the Proceedings of the *1999 IEEE/RSJ International Conference on Robotics and Systems* (IROS '99)

Abstract

To focus on the research issues surrounding collaborative behavior in multiple mobile-robotic systems, a great amount of low-level infrastructure is required. To facilitate our on-going research into multi-robot systems, we have developed RAVE, a software framework that provides a Real And Virtual Environment for running and managing multiple heterogeneous mobile-robot systems. This framework simplifies the implementation and development of collaborative robotic systems by providing the following capabilities: the ability to run systems off-line in simulation, user-interfaces for observing and commanding simulated and real robots, transparent transference of simulated robot programs to real robots, the ability to have simulated robots interact with real robots, and the ability to place virtual sensors on real robots to augment or experiment with their performance.

1 Introduction

Methods for collaboration among multiple heterogeneous mobile robots are a subject of ongoing research [6]. Developing a multiple-robot system requires many support capabilities such as communications, user interfaces, and support for simulation.

In order to focus on algorithms and architectures for collaborative behavior, we have created RAVE, a framework that provides a Real And Virtual Environment for running multiple mobile-robot systems. RAVE provides a unique set of capabilities to facilitate development of such systems. To allow multiple-robot systems to be developed and tested in simulation and then seamlessly transferred to real robots, RAVE allows any robot program to be run on either a real or simulated robot. Through RAVE's simulation capabilities, virtual sensors can be used on both real and simulated robots. Virtual sensors can model real counterparts or can provide sensing capabilities not available on robots in the test-bed. For example, this is useful to determine if infrared proximity sensors would be helpful for the test-bed robot, or if another sonar should be placed in the rear of the robot for more robustness, etc. Investigating sensor configurations in software rather than hardware saves an immense amount of time. RAVE also allows virtual obstacles to be added to the world model. This is necessary for running a system in simulation, but it also allows real robots to operate in virtual or partially virtual environments. These simulation capabilities allow real and simulated robots to operate simultaneously and interact with each other. This is particularly useful when the number of real robots available is limited. RAVE provides a communications package that allows system components to communicate with each other across different computers on a network; RAVE can run entirely on a single computer or can be distributed over many computers.

There are three different types of graphical user interfaces (GUIs) in RAVE, each of which provides different capabilities: an *Observer* can only view the state of the system, a *Commander* can interact with the system by controlling one or more robot teams, and a *Super-User* has the greatest privileges and is able to control the execution of a system run. The GUIs can be run remotely over the World-Wide Web, allowing users to be geographically dispersed.

After reviewing related work in multiple-robot architectures and simulators in Section 2, we describe the overall structure of RAVE in Section 3, some of its details in Section 4, and some applications in Section 5.

2 Related work

Researchers have proposed and developed a variety of multi-robot architectures [2, 7, 9, 11, 12, 13, 14]. These have largely concentrated on methods of cooperative motion and task planning, and placed themselves somewhere along a deliberative-reactive continuum. Fully deliberative systems exercise centralized control using a detailed world model, whereas fully reactive systems expect overall system properties and behaviors to emerge from individual robot behaviors that are not explicitly coordinated with one another. We are not aware of fully deliberative, centralized multi-robot architectures in the literature, probably due to their brittleness in the face of dynamic, unpredictable environments. Several architectures attempt to merge the advantages of the deliberative and reactive paradigms into hybrid systems [2, 9, 15].

Reactive, or fully decentralized, architectures include ALLIANCE [13], CEBOT [7, 14], and ACTRESS [3, 8]. The primary goals of ALLIANCE and its extension L-ALLIANCE, which automates the tuning of behavior control parameters, are to achieve fault-tolerance and adaptivity without sacrificing efficiency. Simulation is used to test subtask allocation strategies, but does not appear to be integrated tightly into the overall architecture when running real robots. Work using the CEBOT and ACTRESS architectures has concentrated on efficient communications strategies and cooperative strategies for accomplishing various tasks.

Hybrid architectures include GOFER [9], AuRA [2], and SIMNET [4]. GOFER combines traditional centralized AI planning techniques with the ability of individual robots to form their own plans to react to changing circumstances. Under AuRA, individual robots behave reactively, but also receive centralized guidance as appropriate. The DARPA SIMNET is a military trainer which allows hundreds of soldiers to train together in a fully virtual environment populated with a large number of simulated autonomous vehicles.

In conjunction with its work on AuRA, Georgia Tech has developed the MissionLab toolset [10], which provides a mission-planning GUI and the ability to run the same code on real or simulated robots. The XRDev simulator created by Nomadic Technologies also allows the execution of the same code on real or virtual Nomadic Technologies robots.

Work on multi-robot conceptual and software frameworks has focused more on architectures for cooperation (reactive, deliberative, hybrid, etc.) than on a general infrastructure which can accommodate a wide variety of architectures.

3 Basic framework

The three main components of RAVE are: libraries for robot programs, information servers, and a set of user interfaces.

3.1 Robot Libraries

Robot programs are linked to libraries that provide a standard interface to real or simulated robots. These libraries form a layer of abstraction that separates the high-level decision software from the low-level interaction with device drivers. This provides the facility for real robots to sense virtual obstacles, the placement of virtual sensors on real robots, and the interaction between real and simulated robots. This layer of abstraction also allows the direct transferring of programs from simulated to real robots. Transferring of programs between real and simulated robots is accomplished by dynamically loading the appropriate drivers at execution. These drivers are determined by parsing the configuration file, which will be discussed further in Section 4.5.

Robots can be endowed with virtual sensors which are not augmentations of a real sensor counterpart. For instance, if the user of RAVE had acquired a mobile robot chassis but had not yet determined the sensor suite, the user could place any number of virtual sensors in appropriate (or completely inappropriate) locations and evaluate the performance of the robot. This process can be repeated until the user is satisfied with the locations and types of sensors to be used. This can save much time and energy as compared to experimentally determining the sensor suite in hardware. Also, RAVE offers users the flexibility to write a virtual sensor driver which has no real world analogue, such as a sensor that detects the velocities of other robots, or an indoor GPS sensor.

When virtual obstacles are injected into the world model of a real robot, several issues arise. First is the issue of sensor fusion. Detecting virtual obstacles requires RAVE to have an appropriate model of the real sensor being augmented. Once the virtual component of the sensor computes its reading, this result is then fused with the result from the real component of the sensor and a single reading is returned (refer to Figure 1).

RAVE allows for interaction between real and simulated robots. This is useful when only a limited number of real robots is available or when the abilities of the real robots are not satisfactory or appropriate. For example, predator-prey scenarios can be executed where the prey might be a simulated robot. Since the prey is simulated, its speed, size, and other attributes can be easily changed to determine how well the predator strategy handles these changes.

3.2 Information Servers

The two main servers in RAVE are the Environment Manager and the GUI Server. These servers maintain the state of the world, i.e., the positions of all robots and any virtual obstacles. They distribute this information to their clients — the Environment Manager to all robots, the GUI Server to all user interfaces. Thus, one key assumption in RAVE is that robots can report their state to these servers in a timely manner. The required frequency of these updates depends

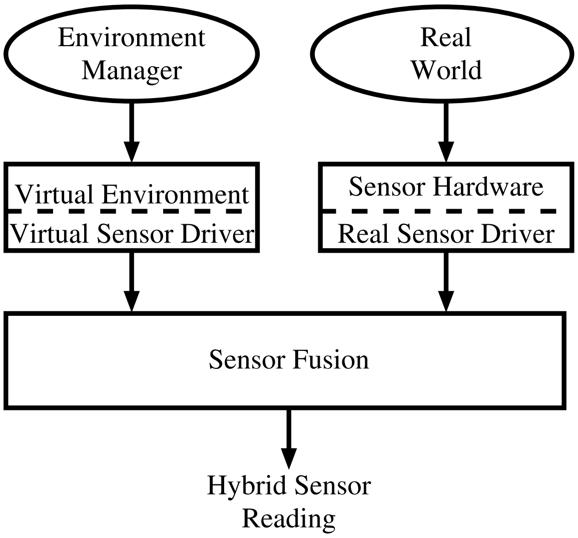

Figure 1: Diagram of a hybrid real and virtual sensor.

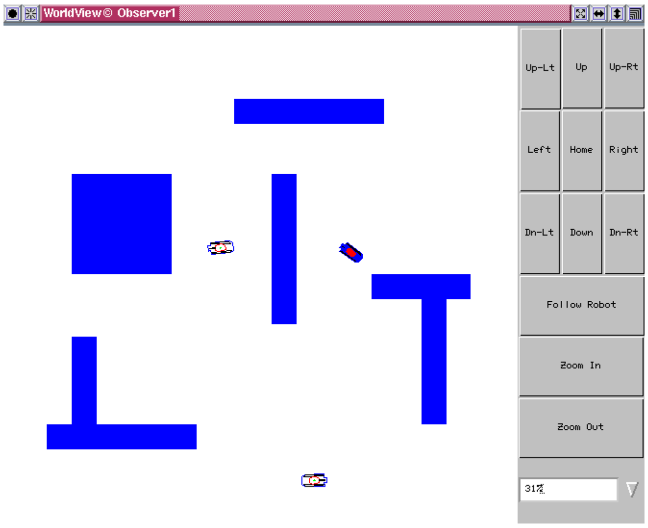

Figure 2: The World View panel.

on the time and distance scales of a particular application.

The Environment Manager distributes information to both real and simulated robots for the computation of virtual sensor readings. When the state of the world model changes, the changes are reported to the Environment Manager; when virtual obstacles are added to the world model by the Super-User or the position of a robot changes, both events are sent to the Environment Manager. Currently, the Environment Manager supports arbitrary convex polygonal obstacles and represents robots (both real and simulated) in a similar fashion. Both obstacles and robots are stored in two dimensions.

The GUI Server sends information to the various user interfaces so that they can display the current state of the system consisting of all robots and obstacles. This information might include positions of virtual obstacles, positions of robots, sensory information recorded by robots, etc..

There is no direct communication between the Environment Manager and the GUI Server. If a system does not have robots with virtual sensors, then the Environment Manager need not be executed. Similarly, if there are no GUIs observing the system then the GUI Server need not be executed.

3.3 User Interfaces

A user interacts with RAVE through one of three types of graphical user interfaces: an Observer, a Commander, or the Super-User. The Observer can only view the state of the system. In addition to viewing the state of the system, the Commander can send commands to one or more teams of robots. In addition to the privileges of the other GUIs, the Super-User has the ability to control and modify not only robots, but also the environment of a system. The Super-User also sets the access rights to information that the other user interfaces have available. The GUI Server is responsible for enforcing these permissions. In general, a given user will only be permitted to see certain classes of information. For example, a Commander GUI might only be able to see the position of robots it commands, or an Observer GUI might not be able to see the position of virtual obstacles. GUIs may be run locally or remotely over the World-Wide Web, allowing users to be geographically dispersed. The RAVE architecture allows any number of GUIs to be connected to a given system. Furthermore, the GUI executables are platform-independent, giving users the ability to command, control, and observe a system from a variety of computing environments.

Because of the overlapping functionality of the three user interfaces, they are structured as a modular collection of components, each of which provides a panel to the user or provides some additional capabilities to an existing panel. The components of an Observer GUI are a subset of those of a Commander GUI, and the components of a Commander GUI are a subset of those of the Super-User GUI.

The panels available in an Observer GUI are:

- World View panel — displays the location of robots and virtual obstacles in the environment (see Figure 2)
- Sensor Display panel — displays sensor readings from robots
- Map panel — displays the local maps from individual robots or a global map from a "map server"
- Chat panel — allows the user to send messages to other GUIs

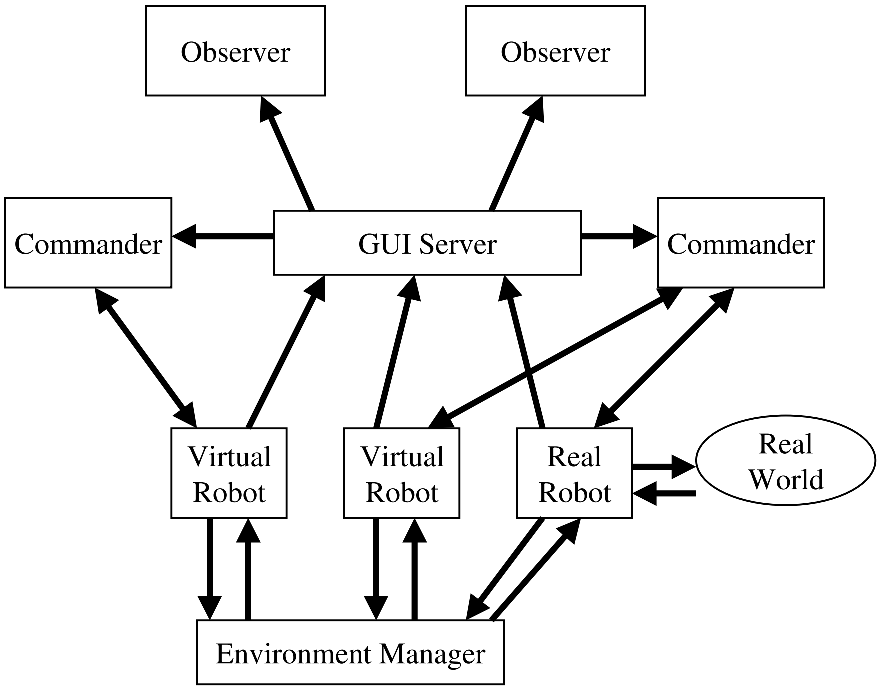

Figure 3: Overall architecture of the RAVE framework

A Commander GUI has one additional panel:

- Joystick panel — provides a joystick interface to control robots

and provides additional capabilities on the following panels:

- World View panel — allows the user to send a goal point to a robot
- Chat panel — allows the user to send messages to the robots it commands

The Super-User GUI additionally contains the following panel:

- Super-User panel — allows the user to select and execute a scenario script and to issue super-user commands

The Super-User panel is a front end to a "Super-User process" that does the work of starting up the robots, GUIs, and servers and sending messages to control the execution of the system. This process interprets a "scenario script" that a user constructs to specify what robots and GUIs should be started, which Commander GUIs control which robots, access privileges, and general coordination among system components.

Because of the modularity of the GUIs, additional functionality can be added to the GUIs by creating new components. Components are dynamically loaded at run-time, based on a configuration file.

3.4 Overall Architecture

Figure 3 shows the overall architecture of the RAVE framework for an example with three robots, two simulated and one real. Each robot provides regular position updates to the Environment Manager, which in turn periodically broadcasts the state of the world to all the robots. These updates are needed by any robot with at least one virtual sensor. Since both real and simulated robots may have virtual sensors, both may need this information to compute virtual sensor readings. Robots regularly report their position and some sensor data to the GUI Server, which in turn periodically broadcasts this information to the user interfaces. Here, we have illustrated two Observers, which can only view a system run, and two Commanders, which can also control the robots. Control commands are sent directly to the robots; the two robots on the right are controlled by one Commander, the leftmost by a second Commander. This Commander GUI can be used regardless of whether the robots are real or simulated, and regardless of whether they communicate their state to an Environment Manager. This allows RAVE to serve as a simulation and development environment as well as a command and control interface for real applications.

3.5 Computational Load

The entire RAVE architecture is implemented in a distributed modular fashion. All servers, GUIs, and robots may be executed on any number of machines. This allows low-end workstations the ability to execute RAVE modules, though a single Pentium II 333MHz machine with 256MB of RAM was used to run all RAVE modules (including the robots) for the collaborative mapping application in Section 5.

4 Robots

In this section, we discuss the standard interface to robots and robot configuration. We have taken an abstract view of a robot in order to provide a standard interface that can be used for a variety of different robot platforms. The four main components of this structure are illustrated in Figure 4 and include sensors and sensor processing, the decision process, controllers and actuators, and communicators and transmitters.

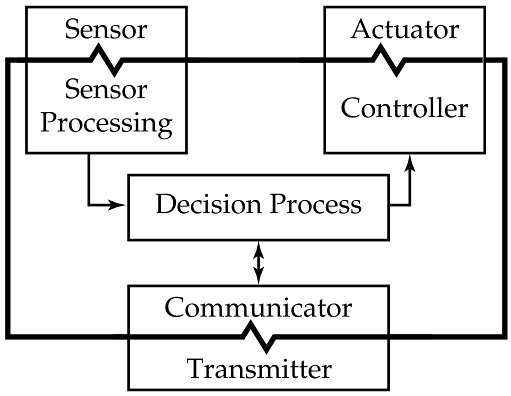

Figure 4: Abstract view of a robot.

4.1 Sensors and Sensor Processing

Real or virtual sensors may be attached to a robot. Real sensors sense the real world, while virtual sensors sense only the virtual world. For every sensor used, whether real or virtual, a sensor driver is needed. Real sensor drivers direct queries to hardware, while virtual sensor drivers make the queries into a virtual environment database that is periodically updated by the Environment Manager. At this point, the virtual sensor driver can inject noise into the sensor reading (e.g., statistical noise). Real and virtual sensors can be fused together in a type of hybrid sensor. In essence, a third sensor driver is executed which is responsible for fusing readings from the real and virtual components of the sensor. All this takes place without the robot realizing whether the reading came from a real, virtual, or fused hybrid sensor. Some processing of raw sensor data (like filtering) may be closely associated with a sensor and can effectively be viewed as a part of the sensor itself.

4.2 The Decision Process

RAVE defines the decision process as the entity that takes sensor readings as input and generates controller commands as output. RAVE does not require that the author of a robot program be confined to using a specific architecture to implement the decision process. Since the decision process only deals with two abstract interfaces, the decision process is the same for a real or simulated robot. Several architectures have been tested and evaluated in RAVE, including [1, 5].

4.3 Controllers and Actuators

The controller is an object that takes commands from the decision process as input and issues commands that the actuator driver will understand as output. Since the controller only deals with two abstract interfaces, the controller driver is the same for a real or simulated robot. Actuators are devices that the robot uses to influence its environment, real or virtual. For instance, this may be a motor, or a servoing mechanism on a pan-tilt camera apparatus. Again, for every actuator used, real or virtual, an actuator driver is needed. On real robots, these actuator drivers redirect actuation commands to hardware. On simulated robots, the virtual actuator driver will cause the robot to update its position (whether the robot as a whole, or perhaps the azimuth of the pan-tilt apparatus). It is at this level that the dynamics of simulated robots are modelled. A simulated robot may be modelled as a second-order system whose wheels occasionally slip, or without any dynamics may at all, depending on the driver used.

4.4 Communicators and Transmitters

A communicator is an object that takes messages from the decision process as input and issues messages that the transmitter will understand as output. A communicator allows the robot to send a message under some message model, for example, limited-range broadcast communication or point-to-point communication. Like a controller, a communicator deals only with two abstract interfaces and is independent of the robot being real or simulated. The transmitter is a device, such as an Ethernet or RF link, that enables the robot to send messages to other system components.

Figure 5: Our little demons. Tanks are on the left, Millibots on the right.

The author of a robot program implements only the decision process; the controller, actuator, communicator, transmitter, and sensor drivers are provided by RAVE through the robot libraries (discussed in Section 3.1).

4.5 Robot configuration

A robot's sensor suite, as well as the type of controller and communicator, is determined at run-time. When a robot program goes through its initialization phase, a configuration file is used to set up the appropriate drivers. Robot programs can be written generally so they are able to work with a variety of sensor configurations. It is then easy to test the effect of various sensor suites on robot performance. A typical robot configuration file would physically define the robot itself, contain controller parameters, definitions of various sensors to be used, and the positions of those sensors.

5 Applications

The non-restrictive architecture of RAVE has allowed its use in many domains. Presently, we use RAVE in a laboratory setting with a group of heterogeneous robots, outdoors with a homogeneous group of two robots, and novel interfaces have been developed to facilitate the interaction with teams of robots.

RAVE has been particularly useful in the development of software for the Millibot project (see Figure 5). In this project, we are developing a heterogeneous team of small mobile robots (6x6x6 cm) for performing surveillance tasks. Due to the small size, each Millibot has only limited sensing and computation capabilities. Still, as a team, they are able to accomplish interesting tasks such as exploration and mapping of unknown environments. To overcome their limited computational power, we have extended the RAVE framework to enable proxy-computation. Within RAVE, a virtual robot (running on a retrofitted model tank-robot that acts as the mothership for the Millibots) is the coordinator for the team. This team leader communicates with each of the Millibots over a low-bandwidth RF communication link, and acts as a proxy relaying messages from and to the commander over a more powerful wireless Ethernet link. At the same time, the team leader acts as a computation proxy for computationally-intensive tasks that cannot be handled by the individual Millibots. For instance, for each Millibot, the team leader executes an extended Kalman filter to determine the position and orientation of the Millibot. The team leader also performs the high-level coordination between the Millibots in the team. RAVE provided the basic functionality that allowed us to rapidly create this hierarchical framework of robot teams in which heterogeneous robots collaborate, guided by team leaders that can in turn be part of other robot teams.

Figure 6: One of our larger, more dangerous beasts.

RAVE has also been used to command and control a group of two homogeneous All-Terrain Vehicles (ATVs) in an outdoor environment (see Figure 6). RAVE is used to control the steering, gearing, throttle, and braking remotely using wireless Ethernet. Autonomous navigation was also demonstrated by entering GPS waypoints to the ATV via RAVE. To simplify command and control of robots in outdoor environments we implemented RAVE on a hand-held, pen-based wearable computer. A gesture-based interface has also been developed for RAVE. This allows users of RAVE to control teams robots by using natural, intuitive hand motions. Users can thus interact with robots (both real and simulated) on a "hands-free" basis.

Collaborative mapping is another application where the capabilities of RAVE have proved invaluable. We are developing and testing our mapping algorithms on a homogeneous group of three retrofitted model tank-robots (20x10 cm), as shown in Figure 5, named "Patton", "Rommel", and "De Gaulle". In this system, each robot builds a local map based on sonar readings as it is driven around by a Commander. The Commander can request a local map from any of its robots or from a "Map Server" (which is specific to this application). The Map Server queries each of the robots for its local map and generates a global map

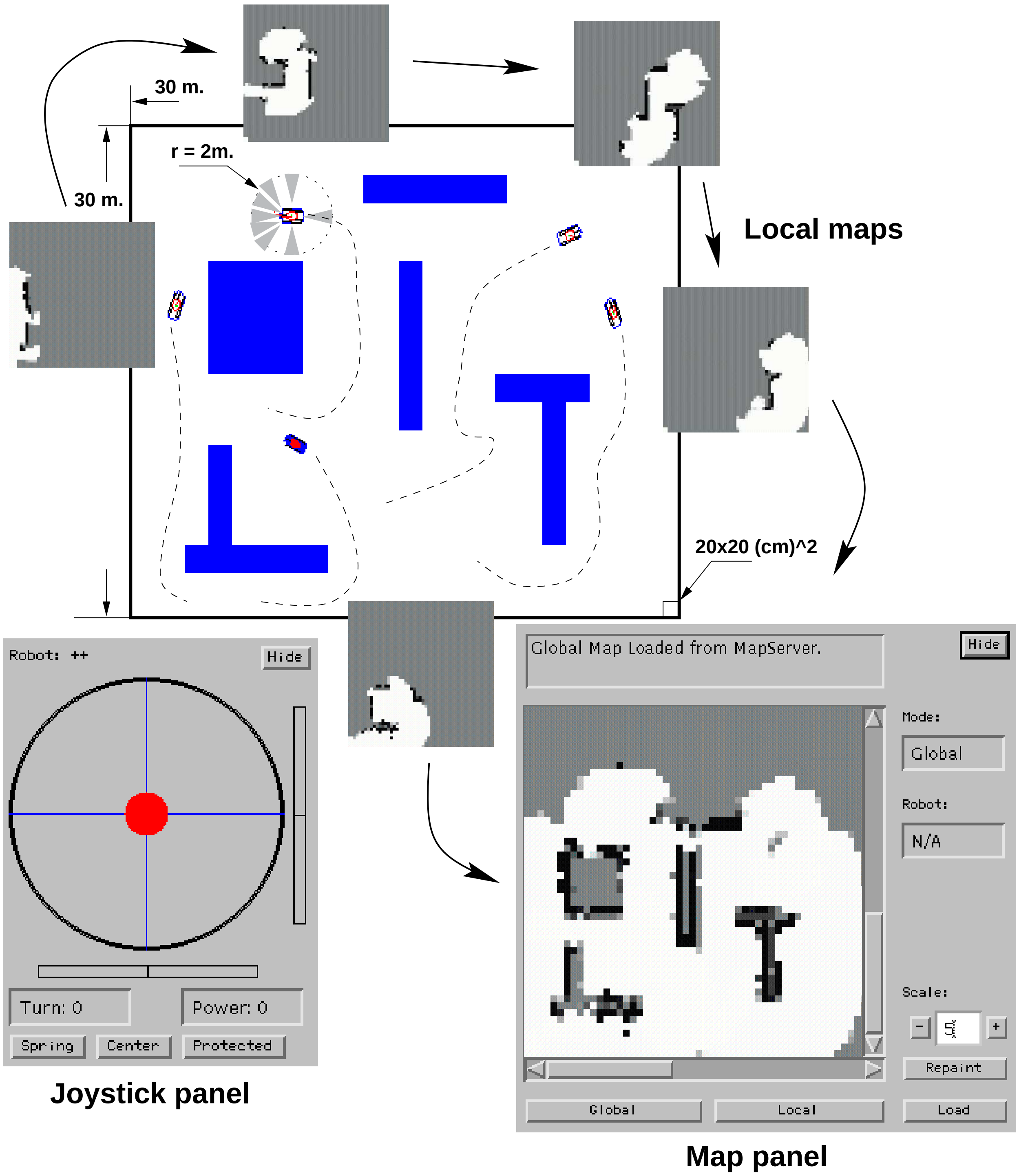

Figure 7: Overview of collaborative mapping.

using a Bayesian update method (illustrated in Figure 7).

6 Conclusions and Future Work

We have created a software framework that provides the infrastructure required to explore collaborative behavior in systems of multiple heterogeneous mobile robots. The key features of this framework are: the ability of the same robot program to run on either a real or a simulated robot, thus facilitating the transfer of a system developed or tuned in simulation to real robots; the provision of virtual sensors that can be used on both real and simulated robots; the ability to add virtual obstacles to the world model; the capability to allow real and simulated robots to operate simultaneously and interact with each other; support for multiple users to command individual robots or teams of robots via a local network or the World-Wide Web; and transparency of distributing computational processes over multiple computers.

Further work on RAVE will include the transition from 2-D to 3-D world modeling, the creation of complex virtual sensors such as cameras, and the extension of simulation management capabilities to include automated capture and analysis of runs.

Acknowledgments

We would like to thank Patrick Chiu, Elliott Delaye, Chris Diehl, Bob Grabowski, Mike Haefele, John Hampshire, Soshi Iba, Jeffrey Lam, Sam Listopad, Ted Pham, Ben Pugliese, Jesus Salido, Mahesh Saptharishi, Poj Tangamchit, and Ashitey Trebi-Ollennu for their contributions to RAVE.

This work was supported in part by DARPA/ETO under contract DABT63-97-1-0003, by the Charles Stark Draper Laboratory under IR&D contract 1005, and by the Institute for Complex Engineered Systems at Carnegie Mellon University. We also thank Intel for providing part of the computing hardware.

References

[1] R.C. Arkin. "Motor Schema-Based Mobile Robot Navigation", *The International Journal of Robotics Research*, 92-112, 1989.

[2] R.C. Arkin and T.R. Balch. "AuRA: Principles and Practice in Review", *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2), 1997.

[3] H. Asama, A. Matsumoto, and Y. Ishida. "Design of an Autonomous and Distributed Robot System: ACTRESS", In *Proc. of the IEEE/RSJ Int'l Workshop on Intelligent Robots and Systems*, 283-90, 1989.

[4] D.L. Brock, D.J. Montana, and A.Z. Ceranowicz. "Coordination and Control of Multiple Autonomous Vehicles", *IEEE Int. Conf. on Robotics and Automation*, 2725-30, 1992.

[5] R.A. Brooks. "A Robust Layered Control System for a Mobile Robot", In *IEEE Journal of Robotics and Automation*, 14-23, 1986.

[6] Y.U. Cao, A.S. Fukunaga, and A.B. Kahng. "Cooperative Mobile Robotics: Antecedents and Directions", *Autonomous Robots*, 4:1-23, 1997.

[7] T. Fukuda and S. Nakagawa. "A Dynamically Reconfigurable Robotic System (Concept of a System and Optimal Configurations)", In *Int. Conf. on Industrial Electronics, Control, and Instrumentation*, 588-95, 1987.

[8] Y. Ishida. "Functional Complement by Cooperation of Multiple Autonomous Robots", In *IEEE Int. Conf. on Robotics and Automation*, 2476-81, 1994.

[9] C. LePape. "A Combination of Centralized and Distributed Methods for Multi-Agent Planning and Scheduling", In *IEEE Int. Conf. on Robotics and Automation*, 488-93, 1990.

[10] D. MacKenzie, R. Arkin, and J. Cameron. "Multiagent Mission Specification and Execution", *Autonomous Robots*, 4(1):29-52, 1997.

[11] M. Mataric. "Interaction and Intelligent Behavior", *MIT AI Lab Technical Report*, AI-TR-1495, August 1994.

[12] F.R. Noreils. "An Architecture for Cooperative and Autonomous Mobile Robots", In *IEEE Int. Conf. on Robotics and Automation*, 2703-10, 1992.

[13] L. Parker. "ALLIANCE: An Architecture for Fault-Tolerant, Cooperative Control of Heterogeneous Mobile Robots", In *Proc. of the IEEE/RSJ Int'l Conf. on Intelligent Robots and Systems*, 776-83, 1994.

[14] T. Ueyama and T. Fukuda. "Self-Organization of Cellular Robots Using Random Walk with Simple Rules", In *IEEE Int. Conf. on Robotics and Automation*, 595-600, 1993.

[15] M. Watanabe, K. Onoguchi, I. Kweon, and Y. Kuno. "Architecture of Behavior-based Mobile Robot in Dynamic Environment", In *IEEE Int. Conf. on Robotics and Automation*, 2711-18, 1992.