Technical Report

Department of Computer Science
and Engineering
University of Minnesota
4-192 EECS Building
200 Union Street SE
Minneapolis, MN 55455-0159 USA

TR 01-031

Performance of a Distributed Robotic System Using Shared Communications Channels

Paul E. Rybski, Sascha A. Stoeter, Maria Gini, Dean Hougen, and Nikos Papanikolopoulos

July 26, 2001

Performance of a Distributed Robotic System Using Shared Communications Channels

Paul E. Rybski, Sascha A. Stoeter, Maria Gini,
Dean F. Hougen, and Nikolaos Papanikolopoulos*

Abstract

We have designed and built a set of miniature robots, called Scouts. In addition, we have developed a distributed software system to control them. This paper addresses the fundamental choices we made in the design of the control software, describes experimental results in a surveillance task, and analyzes the factors that affect robot performance.

Space and power limitations on the Scouts severely restrict the computational power of their on-board computer, which can only handle low-level control operations and data communication on an RF data link. We use a proxy-processing scheme, in which the robots are dependent on remote computers for their computing needs. While this allows the distributed robotics system to be autonomous, the fact that the robots behaviors are executed on remote computers introduces an additional complication – the behavior controller has to receive sensor data and send commands to the robots using RF communication channels. Because the capacity of the communication system is limited, the robots must share bandwidth.

We have developed a process management/scheduling system that is capable of handling high demand when controlling a group of robots. The resource allocation system dynamically assigns resources to each robot in an attempt to maximize the utilization of the available resources while still maintaining a priori behavior priorities.

We present experimental results on a surveillance task in which multiple robots patrol an area and watch for motion. We discuss how the limited communication bandwidth affects robot performance in accomplishing the task and analyze how performance depends on the number of robots that share the bandwidth.

1 Introduction

Controlling a group of miniature mobile robots in a coordinated fashion can be a very challenging task. The small size of miniature robots greatly limits the kinds of on-board computers and sensor processing systems they can use. One way to overcome these limitations is to use a robust communications link between the robot and a more powerful off-board

*Center for Distributed Robotics, Department of Computer Science and Engineering, University of Minnesota, Minneapolis, MN 55455.

processor. Unfortunately, the robots' small size also limits the bandwidth of the communications system that they can employ. In many robotic implementations, this problem can be addressed with large capacity communications hardware (such as a wireless Ethernet). However, many robotic systems of interest cannot use high-capacity communications links because of size, power, or computational bandwidth limitations. In these cases, addressing system issues such as process scheduling becomes critical for effective operation.

We describe a case study of a group of extremely small robots which must use very low capacity RF communications systems due to their small size. The size limitations of these robots also restrict the amount of on-board computational power they can carry, forcing them to rely on off-board decision processes. Thus, all the sensor data are broadcast to a remote computer or to a larger robot, and actuator commands are relayed back to the miniature robots. The operation of these robots is completely dependent on the RF communications links they employ. In order to handle high demand for this low capacity communications system, a novel process management/scheduling system has been developed.

A surveillance task in which the robots patrol an area and watch for motion is also described. The resource allocation system dynamically assigns resources to each of the robot's control processes in an attempt to use as much of the available bandwidth as possible while maintaining other constraints (such as process priorities).

2 Miniature Robotic Systems

We have developed a set of extremely small robotic systems, called Scouts [1], which are designed for reconnaissance and surveillance tasks. The Scout robot, shown in Figure 1, is a cylindrical robot 11.5 cm in length and 4 cm in diameter. The Scouts can transmit video from a small camera to a remote source for further processing. They can transmit and receive digital commands over a separate communications link that uses an ad-hoc packetized communications protocol.

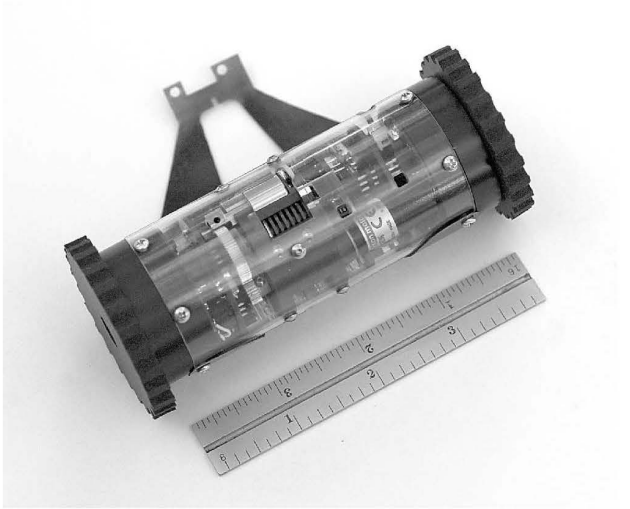


Figure 1: The Scout robot shown next to a ruler (in inches) for scale.

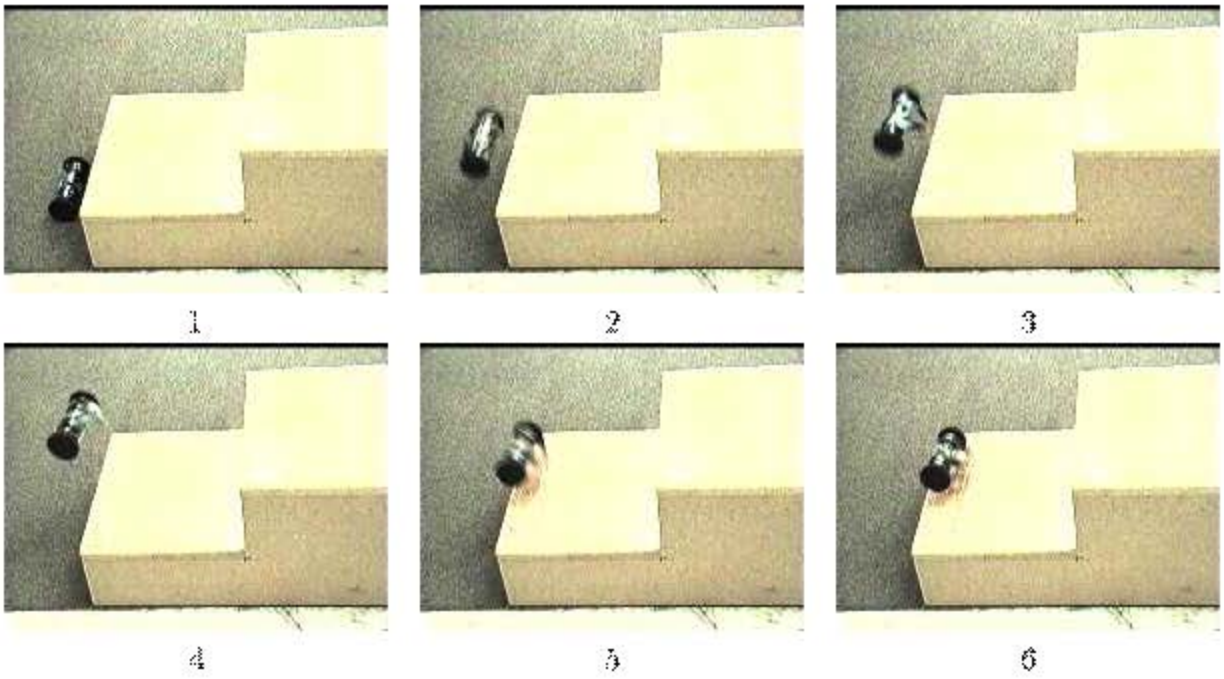


Figure 2: A Scout using its spring to jump up a stair.

Scouts locomote in two ways. They can use their wheels to travel over smooth surfaces (even climbing a 20 degree slope) and they are capable of jumping over objects 30cm in height using their spring-loaded tails. Figure 2 shows the Scout jumping up a step.

Due to the Scout's limited volume and power constraints, the two on-board computers are only powerful enough to handle communications and actuator controls. There is very little memory for any high-level decision process and no ability to process video. In order for the Scouts to accomplish anything useful, they must be paired with an off-board computer or a human teleoperator.

Scouts receive command packets transmitted by a radio hooked up to a remote computer. Each Scout has a unique network ID, allowing a single radio frequency to carry commands for multiple robots. By interleaving packets destined for the different robots, multiple Scouts can be controlled simultaneously.

Video data is broadcast over a fixed-frequency analog radio link and must be captured by a video receiver and fed into a framegrabber for digitizing. Because the video is a continuous analog stream, only one robot can broadcast on a given frequency at a time. Signals from multiple robots transmitting on the same frequency disrupt each other and become useless.

The RF limitations of the Scout pose a couple of fundamental difficulties when trying to control several Scouts. First, the command radio has a fixed bandwidth. This limits the number of commands it can transmit per second, and therefore limits the number of Scouts that can be controlled simultaneously. Currently, our inter-robot communications network operates on a single carrier frequency, with a command throughput of 20-30 packets per second.

Second, the absence of many independent analog video channels reduces the number of robots that can simultaneously transmit video. There are generally not enough commercial

frequencies available to allow for a large number of interference-free simultaneous analog transmissions. Only two different video frequencies are available with the current Scout hardware. As a result, video from more than 2 robots can be captured only by interleaving the time each robot's transmitter is on. Thus, an automated scheduling system is required to make sure that the robots share the limited communications resources and do not interfere with each other's transmissions. Sharing the bandwidth among different robots affects the performance of the Scouts as we will see in the description of our experimental results in Section 6.

3 Software Architecture

The decision processes that control the actions of the Scouts need to be able to connect to all the individual resources that are necessary to control the physical hardware. When multiple robots are used, this is not always a trivial task. To address the challenge of controlling multiple Scouts, we have designed a software architecture [2], illustrated in Figure 3, which is capable of connecting groups of decision processes with resource controllers that have the responsibility of managing the physical resources in the system.

This distributed software architecture dynamically coordinates hardware resources transparently across a network of computers and shares them between client processes. The architecture includes various types of user interfaces for robot teleoperation and various sensor interpretation algorithms for autonomous control. The architecture is designed to be extremely modular, allowing for rapid addition of behaviors and resources to create new missions. The system is composed of four distinct subsystems: the Mission Control, the Resource Pool, the User Interface, and the Backbone.

3.1 Mission Control

All behaviors and decision processes are contained and managed from within the *Mission Control* subsystem. Here, a mission can be built out of discrete behaviors, started, observed, and stopped by a single human operator. Behaviors are organized in a hierarchical fashion, where "parent" nodes spawn off "children" to do various tasks. Behaviors are given priorities which are used to determine how to allocate access to resources.

3.2 Resource Pool

In order for behaviors to do anything useful, they must make connections to components in the *Resource Pool* subsystem. This subsystem controls access to robotic hardware and other computational resources through processes called *Resource Controllers* (RCs). Every physical resource is given its own RC to manage it. If a behavior or another decision process needs use that particular resource, it must be granted access to the appropriate RC.

Some physical hardware can only be managed by having simultaneous access to groups of RCs. This grouping is handled by a second layer consisting of components called *Aggregate Resource Controllers* (ARCs). Every ARC is an abstract representation of the group of RCs that it manages, as shown in Figure 4. An ARC provides a specialized interface into

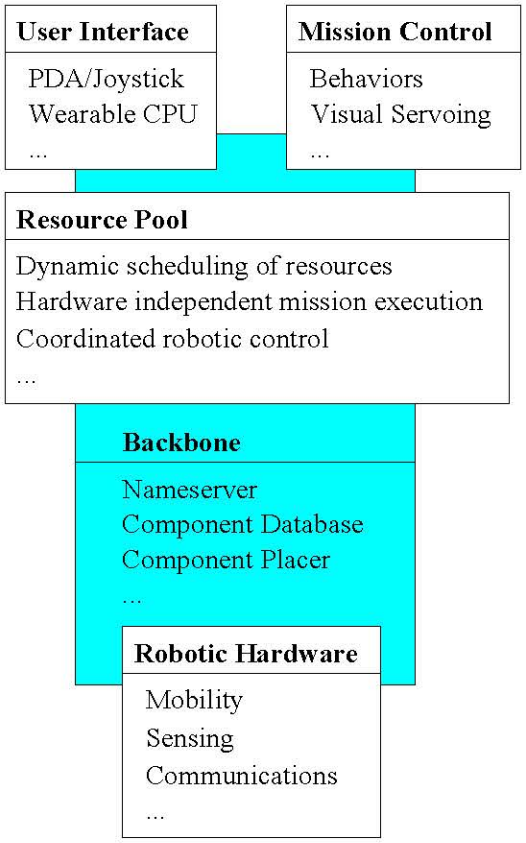


Figure 3: A layered system architecture for control of multiple robots in both teleoperated and autonomous modes.

the group of RCs that it manages. This frees behaviors from the effort of managing all of the commands to each of the specific RCs that it needs to operate. In fact, behaviors are never allowed to interface with the RCs directly.

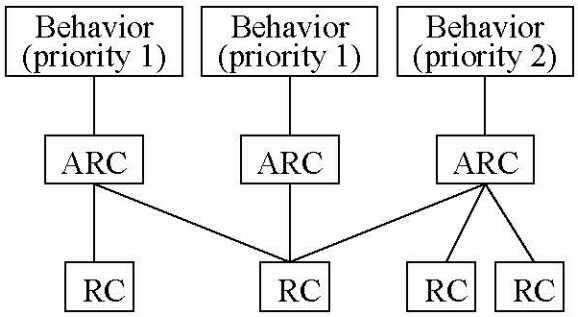


Figure 4: Example of how behaviors access resource controllers.

The central component which oversees the distribution and access to the ARCs and RCs is the *Resource Controller Manager*, which uses a centralized real-time resource scheduler

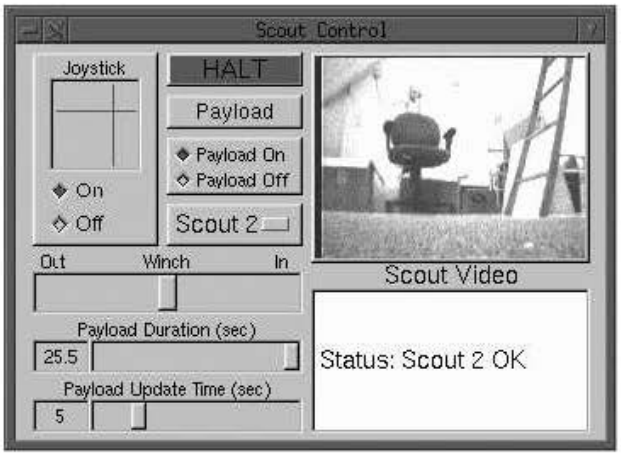


Figure 5: Scout user interface console.

to allocate resources according to the current demand of all running decision processes. Behaviors send scheduling requests to the Resource Controller Manager in order to secure runtime for their ARCs. Each query is parameterized by the specific set of RCs requested. The Resource Controller Manager takes each scheduling request and determines whether the behavior can have access to the ARC and RCs, or whether it will have to wait until those resources are freed when a higher-priority behavior is finished with them.

3.3 User Interface

Direct human control of the resources in the system is provided through the *User Interface* (UI) subsystem. This system allows a human operator to connect to a Scout robot directly and command it from a graphical console, as shown in Figure 5. A user interface console can be started on any machine on the local network, allowing multiple users to request control of the resources simultaneously. Like a behavior, a UI component gets access to its ARCs and RCs by sending a scheduling request to the Resource Controller Manager. UI components have priorities that are set when they are invoked, allowing them to have priority over all currently running autonomous behaviors. This allows a human to immediately take control over an autonomously controlled robot, adjust its position and actions, and then release it back to the behavior that was controlling it before.

3.4 Backbone

Tying all these subsystems together in a seamless fashion across a network of computers is a CORBA-based [3] group of core services called the *Backbone*. The Backbone is responsible for starting and stopping components as well as keeping track of which components are running which computers. An XML description file contains the configuration of how an instance of the architecture is defined. This file describes what machines on the local network are accessible and what core services must be activated to facilitate the mission. The first service to be started is the CORBA nameserver, which is the central communications hub

for all connected components (behaviors, UIs, ARCs, RCs, and core services). The next component to be started is the *Component Database*. This component manages a list of all possible component types and information about their type and multiplicity. For instance, it allows behaviors to be instantiated multiple times as part of a mission, but does not allow multiple core services to start (which may wreak havoc in the system otherwise). The next service to start up is the *Component Placer*. This component is responsible for starting all non-core service components and is the central service with which all other components interface for this purpose. A *Component Creator* service is started on each computer that takes part in the execution of the mission. The *Component Placer* instructs the *Component Creators* to start components as needed. The next system to be started is the the *Static Dependency Database*. This component maintains a database of all the hardware that is currently available for use. This information is provided by a description file that is updated immediately before the mission is started. Finally, the Resource Controller Manager is started.

4 Dynamic Resource Allocation

When a decision process wants to control a Scout, it must first get access to an appropriate ARC. Several kinds of ARCs are available, each requiring different resources to operate. These include ARCs which are capable of controlling only the actuators on the Scouts and do not require the use of the camera, ARCs which drive the Scouts and broadcast video data (to be viewed on a monitor by a human for instance), and ARCs which move the scout, broadcast data, and capture it on a workstation for processing.

The decision process chooses which ARC best fits its needs. When invoked for the first time, each ARC must be told which specific RCs to use. This information is either decided ahead of time (hard-coded into the behavior), or obtained from a database.

4.1 An Example of ARCs and RCs

In order for a process to control a single Scout robot, several physical resources are required. The first resource, a robot which is not currently in use by another process, must be selected. Another resource, a command radio which has the capacity to handle the demands of the process, is also needed (Refer to Section 4.3 for a discussion about the radio's capacity.) If the Scout robot is to transmit video, exclusive access to a fixed video frequency is mandatory. Without exclusive access, the interference caused by simultaneous video transmissions on the same frequency renders any signal processing hardware/algorithms useless. Finally, to process the video, a framegrabber connected to a tuned video receiver is required. Each instance of these four resources is managed by its own RC.

Figure 6 illustrates the interconnections between the components in the system. In this example, a behavior tree is responsible for controlling two robots and a user interface teleoperation console lets a user control a third. Each component has its own ARC which attempts to gain access to the appropriate resources. There are three Scout robots, all of which share a single video frequency. A single video receiver is attached to a video processing card, and a Scout command radio is attached to a serial port. The workstation that contains

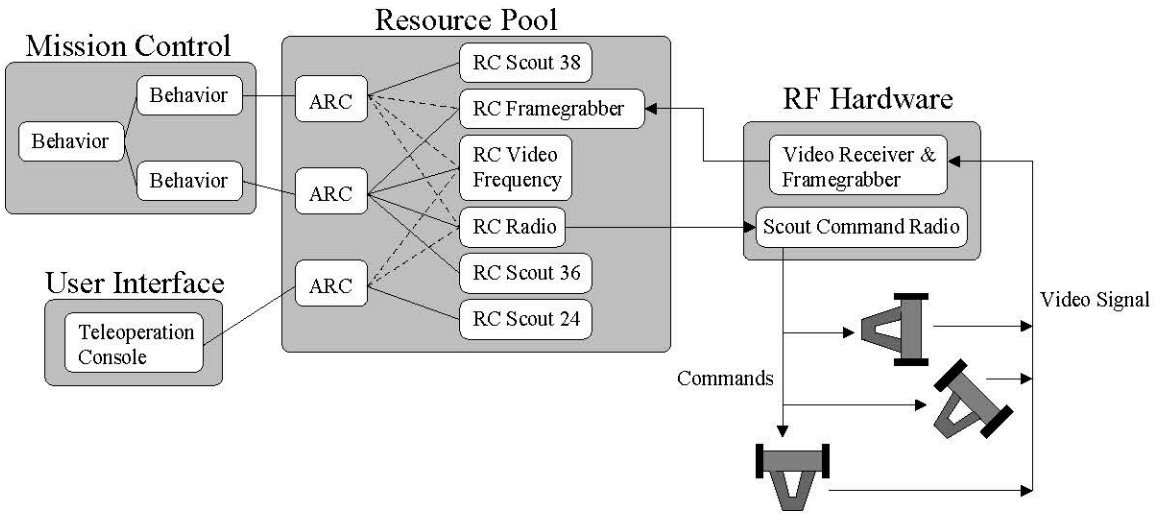


Figure 6: An instance of the architecture. Three Scouts are controlled by a combination of behaviors and a teleoperation console. All three share the same video frequency, so only one robot can be controlled at a given time. Solid lines indicate active connections (where data can flow between components) while dashed lines indicate connections that are not currently active but may be active later during the scheduling process.

the video card does not have to be the same machine that controls the radio. The ARCs belonging to the behaviors must share the video frequency and framegrabber RCs. The ARC owned by the teleoperation console does not need the framegrabber but still needs control of the video frequency to operate. In this situation, only one of the three ARCs will be able to send commands to its robot at a time and thus the ARCs must have their access scheduled by the Resource Controller Manager.

4.2 The Resource Scheduler

Access to RCs must be scheduled when there are not enough RCs to satisfy the requirements of the ARCs. The Resource Controller Manager maintains a master schedule of all active ARCs and grants access to each of their RCs when it is their turn to run. When requesting access to a set of RCs, an ARC must specify a minimum amount of time that it must run to get any useful work done. This value, which is generally on the order of seconds to minutes, is called the minimum runtime value.

The Resource Controller Manager's scheduling algorithm tries to grant simultaneous access to as many ARCs as possible. ARCs are divided into sets depending on the RCs they request. ARCs that ask for independent sets of RCs are put into different groups. These groups will run in parallel with each other since they do not interact in any way. The ARCs that have some RCs in common are examined to determine which ARCs can operate in parallel and which are mutually exclusive. ARCs which request a non-sharable RC cannot run at the same time and must break their total operating time into slices. ARCs which have a sharable RC in common may be able to run simultaneously, assuming that the capacity requests for that sharable RC do not exceed its total capacity.

ARCs that have higher priorities are given precedence over ARCs with lower priorities. The Resource Controller Manager attempts to generate a schedule of running ARCs which allows all ARCs of the highest possible priority to run as often as they are able. If any ARCs of a lower priority can run at the same time as these higher priority ARCs without increasing the wait time of any of the higher-priority ARCs, they are allowed to do so. Lower priority tasks that cannot be so scheduled must wait (possibly indefinitely) for the higher priority tasks to complete.

Once the ARC schedule has been constructed, the Resource Controller Manager invokes its schedule manager to execute it. The schedule manager takes care of notifying the RCs when their run time is completed or when they should start while each ARC runs for its requested minimum time. When a context switch occurs, the schedule manager instructs each of the running RCs to disconnect from their current ARCs and connect to the next set to be scheduled. The ARC signals its controlling process (behavior or UI) that a context switch has occurred and will not pass any messages to its RCs until they are available for work again.

4.3 Sharable Resources

Sharable RCs, such as the Scout radio, have to manage their own schedules to ensure that each of the ARCs using them is given a chance to send packets to their robot at their requested rate. When requesting access to a sharable RC, an ARC must specify a usage parameter which defines how often it will make requests and, if relevant, what kinds of requests will be made. In order to streamline the scheduling process, commands sent to sharable RCs must have a constant interval between invocation. In addition, each request must complete before the next request for that command is made. However, because the CPU load of any given computer will vary depending on how many components are running on it, the run-time of any given request may vary. Given the first two constraints, and some assumptions on the validity of the third, a rate monotonic algorithm (RMA) [4] is used to schedule access.

Requests with higher frequencies are given a precedence over requests with lower frequencies. Once again, however, the user-set priorities must be maintained when the scheduler computes the schedule. Thus, higher user-set priority ARCs have precedence over lower user-set priority ARCs regardless of the frequency of the requests. This can cause a disruption in the way requests are handled by the scheduling algorithm and may produce a schedule which is suboptimal in its usage of the RCs. Only when all of the higher-priority RCs have been scheduled will the lower priority RCs be allowed access. If all of the ARCs cannot be scheduled by the sharable RC, the responsibility for handling requests is given to the Resource Controller Manager.

Once the requests for access have been granted, the ARCs can use them in any way they see fit. Until they make a request for a specific command to be sent to the radio, for instance, the timeslices devoted to those ARCs are empty and the radio does nothing. There are two types of command requests that an ARC can make. The first type, the single command, is executed only once and it must be re-submitted each time the ARC requires it. The second type, the repeated command, only has to be submitted once since it will be repeated by the

RC as often as possible. ARCs can elect to replace an instance of a repeated command with a single command if they so choose.

As illustrated in Figure 7, several ARCs have been granted access to a radio. Both ARC1 and ARC2 have inserted a single execution command into the first slots allocated to them. After that single command has been executed, ARC1 will have a repeated command executed from that point on, and ARC2 will do nothing. ARC3 has requested a repeated command for its only timeslot. Slot 8 has not been filled with anything since all the ARCs have a command executed on their requested period.

1	ARC1 - Single
2	ARC2 - Single
3	ARC1 - Repeated
4	ARC3 - Repeated
5	ARC1 - Repeated
6	ARC2 - Empty
7	ARC1 - Repeated
8	Empty Slot

Figure 7: A typical radio schedule showing what timeslots are available to the scheduled ARCs and the types of commands with which those slots have been filled.

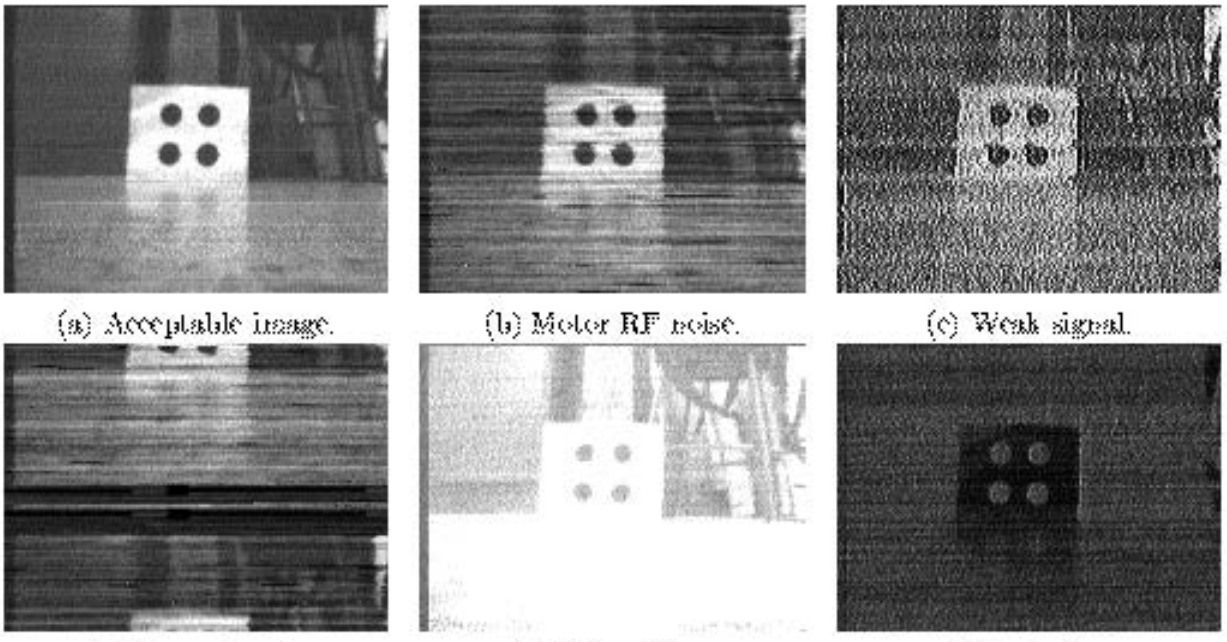
5 A Distributed Surveillance Task

The Scouts are used in a distributed surveillance task where they are deployed into an area and watch for motion. This is useful in situations where it is impractical to place fixed cameras because of difficulties relating to power, portability, or even the safety of the operator. In this task, the Scouts can either be deployed into their environment by a human or another robot, or they can autonomously find their way into useful areas.

Several simple behaviors have been implemented to do the task. All the behaviors use the video camera, which currently is the only environmental sensor available to the Scout. Using the video camera presents several problems. One problem is the Scout's proximity to the floor which severely restricts the area it can view.

Another problem occurs because the video is broadcast over an RF link to a workstation for processing. The quality of the received video often degrades due to noise injected into the system from the Scout's motors, multi-path reflections caused by the presence of obstacles around the robot, and weak signal strength caused by proximity to ground and excess distance between transmitter and receiver. Figure 8 illustrates how noise can affect the quality and clarity of returned images. Finding ways to address this problem is extremely important as any perceptual system which operates in the real world must be able to recognize and correct for corrupted sensor data if it is to operate correctly [5].

In earlier work, we used a simple frame averaging algorithm to reduce the effects of noise [6]. This approach only dealt with the problem of spurious horizontal lines and white



(a) Acceptable image. (b) Motor RF noise. (c) Weak signal.

(d) Loss of sync. (e) Saturation. (f) Inversion.

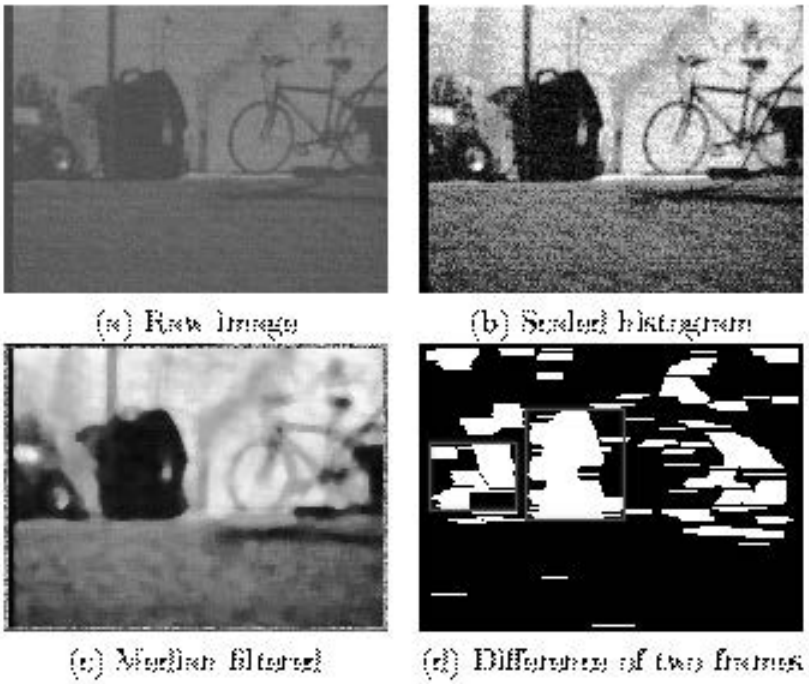
Figure 8: Effects of RF noise in Scout vision.

noise (Figures 8(b) and 8(c), respectively). If the image becomes saturated/inverted (Figures 8(e) and 8(f), respectively), or if vertical synchronization was lost (Figure 8(d)), averaging only compounded the problem.

Currently, the grayscale histogram of the Scout vision is scaled (Figure 9(b)) in order to maximize the contrasts between light and dark areas. However, this has the side effect of highlighting RF noise. To compensate, we apply a 5×5 median filter (Figure 9(c)) over the image to smooth out the data. The median filter is faster than applying a Gaussian convolution mask and does a fairly good job of removing much of the most common noise. Finally, depending on the image processing algorithms being used, several heuristic filters have been implemented which remove data that is corrupted by RF noise. These filters were generated by hand after analyzing how the vision is corrupted by RF noise. There is no universally-applicable way to clean up a bad frame of vision. Quite often, if the transmitted vision from a Scout is severely corrupted, the only hope of cleaning it up is to reposition the Scout such that there is less interference between the transmitter and receiver. Figure 9(d) shows the result of performing frame differencing and connected region extraction. This image is generated by rotating the Scout counterclockwise in place. The white regions are connected regions that are different between the two images. The two grey rectangles highlight blobs that are considered to be caused by the Scout's motion. All of the other blobs could have been caused by random RF noise (decided by their shape and area) and are thus ignored.

The behaviors we have implemented for this task are:

Locate-Goal. Determining the location of the darkest (or lightest) area of the room is accomplished by spinning the Scout in a circle and checking the mean value of the pixels



(a) Raw image (b) Scaled histogram

(c) Median filtered (d) Difference of two frames

Figure 9: Scout image processing algorithms.

in the image. The simplest scan is accomplished in a number of discrete movements. The Scout captures an image, rotates for half a second, takes a new image, and subtracts the new image from the old one. A large difference in the images indicates the Scout moved. There are instances where this approach can fail, however: if the transmitted image quality is so low that motion in the image cannot be distinguished from noise, and if the robot is operating in an area of very low light or uniform color, there may not be enough detail in the images to generate significant differences. Scaling the histogram, as described earlier, helps to increase the contrast between different objects in the image, allowing the objects to stand out when the Scout moves.

Drive-Toward-Goal. Identifying a dark area to move toward is a simple matter of analyzing a strip in the image along the horizon and determining the horizontal position of the darkest area. The Scout computes the darkest region and tries to servo in that direction. The Scout will stop when its camera is either pressed up against a dark object, or if it is in shadows. If either of these two methods fail, this behavior will time out and quit after a minute or two of operation. Scout motion in this behavior is continuous and the Scout does not check its movements by frame differencing because it does not move very quickly. The difference between subsequent frames captured during forward motion is often very minimal, making it difficult for the Scout to detect its own motion.

Detect-Motion. Detecting moving objects is accomplished using frame differencing. The Scout stays still and subtracts sequential images in the video stream and determines whether the scene changes at all (caused by movement in the image.) RF noise can also cause a great deal of perceived motion between frames. This is filtered out by

analyzing the shapes and sizes of the blobs and ignoring blobs that are caused by noise. Currently, a hand-tuned filter is used for this decision process.

Handle-Collisions. If the Scout drives into an obstacle, all motion in the image frame will stop. If no motion is detected after the Scout attempts to move, it will invoke this behavior and start moving in random directions in an attempt to free itself. In addition to freeing the Scout from an object that it has driven into, this random motion has an additional benefit. If the Scout is in a position where the video reception quality is extremely poor, the static in the image will prevent the Scout from detecting any motion (regardless of whether it is hung up on an object). Moving the Scout changes the orientation of the antenna which might help improve reception.

6 Experimental Results

The Scouts' ability to accomplish the surveillance task was examined with a series of experimental runs. These experiments were designed to test the individual and team performances of the Scouts and the controlling architecture in a number of different situations and group configurations. In particular, we were interested in evaluating:

- the effectiveness of the vision-based behaviors for navigating the Scouts to useful positions;
- how the scheduling system handles multiple Scouts all trying to use the limited bandwidth RF video channels to detect motion;
- the performance of the robotic team given a set of specific constraints for the system and the environment.

Three experiments were run to evaluate the Scouts' performance.

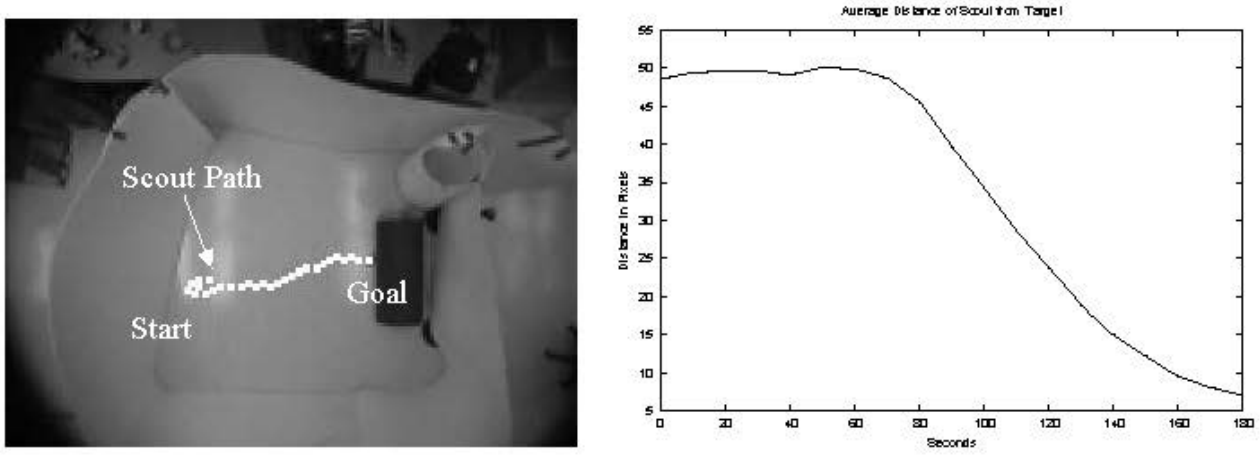
A. Visual Servoing. A Scout has to locate a dark area and move to it.

B. Hiding and Viewing a Room. One or more Scout robots have to hide in a dark area in a room and turn to view a bright area.

C. Detecting Motion. One or more Scout robots have to detect a moving object.

6.1 Visual Servoing

An initial experiment was done to determine how well the Scout could locate and move to an area, using the image from its camera. The environment consisted of a roughly $2.5\,m \times 3\,m$ enclosure with uniformly-colored walls and a $1\,m \times 0.5\,m$ black rectangle towards one side of the enclosure as the target for the Scout. The Scout was started 1.5 m away from the center of the target. This experiment was designed to examine the Scout's Locate-Goal and Drive-Towards-Goal behaviors in an analytical fashion.



(a) Top view of Experiment A.

(b) Average distance of robot to target.

Figure 10: Experiment A: Validation of the Locate-Goal and Drive-Toward-Goal behaviors.

Nine trials were run to see how long it would take the Scout to locate the black target object and move to it. A camera was mounted on the ceiling of the room and was used to view the progress of the Scout from above; this image was not used by the Scout. A simple tracking algorithm was used to automatically chart the progress of the Scout as it moved toward the target. Figure 10(a) shows the view from the overhead camera as well as a superimposed plot of the path that the Scout took to reach its objective during one of its nine trials. In each case, the Scout successfully located the target and moved to it.

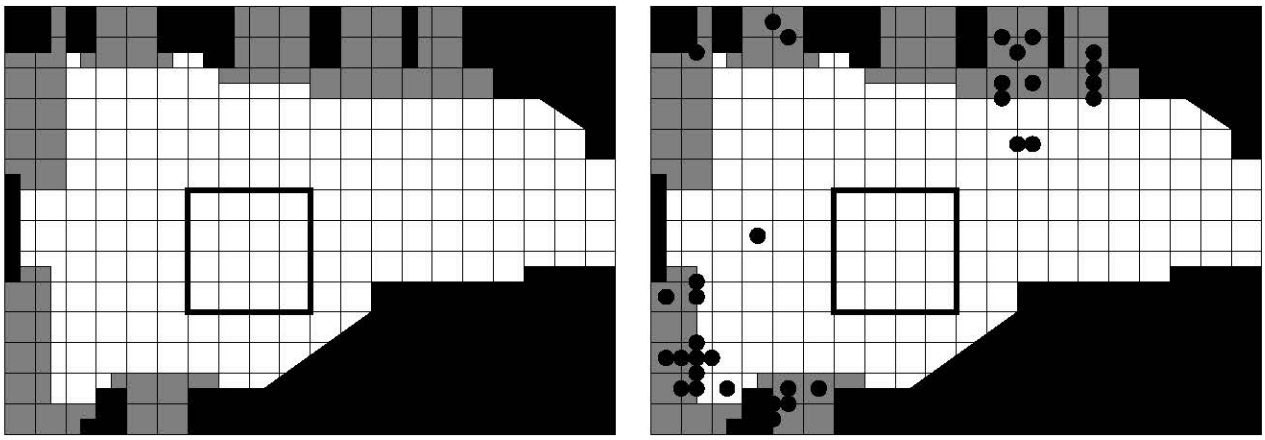
Figure 10(b) shows a plot of average distance from the Scout to the target vs. time for all of these trials. In the first 70-80 seconds, the Scout used its Locate-Goal behavior to find the dark spot. Once it found it, the Scout used its Drive-Toward-Goal behavior until it came in contact with the goal, somewhere between 150 and 160 seconds after the start of the trial.

6.2 Hiding and Viewing a Room

To test the ability of the Scouts to operate in a more real-world environment, a test course was set up in our lab using chairs, lab benches, cabinets, boxes, and miscellaneous other materials. The goal of each Scout in these experiments was to find a suitable dark hiding place, move there, and turn around to face a lighted area of the room.

As shown in Figure 11, the environment was 20 feet long and 14 feet wide and had a number of secluded areas in which the Scout could hide. The test course had roughly 145 square feet of open space, 86 square feet of obstructed space, and 49 square feet of potential hiding places. The Scouts were started at the center of one of the 16 tiles in the center of the room and were pointed at one of 8 possible orientations. Both the position index and orientation were chosen from a uniform random distribution.

The hiding and viewing experiment was divided into three cases, each using a different number of Scouts or communication channels. Within each case, ten trials were run. The stopping positions and orientations of the Scouts from the end of the trials were used later



(a) View of the room. (b) Positions the Scouts took.

Figure 11: A grid layout of the 20 ft by 14 ft room where Experiments B and C were conducted. The white squares indicate open space, the gray areas indicate areas where the Scout could hide itself (underneath tables and benches), and the black areas indicate obstacles/obstructions. The black square outline in the center of the open area shows the area where the Scouts were started. In (b) the positions the Scouts found for themselves in the Hiding and Viewing experiment are represented as dots.

for the detect motion experiment (Experiment C).

In the first case of the hiding and viewing experiment, a single Scout on a single video frequency was used. This case was selected to serve as a baseline with which to compare two different cases, each using two Scouts. The second case used two Scouts that had to share a single video frequency using the scheduler. The third case used two Scouts, each on its own video frequency.

For the second case, in which two Scouts shared a single video frequency, only one Scout could have access to the video frequency at any given time. Because of this, only one behavior could move a Scout at any given time. Access to the video frequency was allocated and scheduled by the Resource Controller Manager and the amount of time that each behavior could use the video frequency was specified by the behavior (the minimum runtime value) at mission startup. For these experiments, each Scout's behavior requested ten second intervals in which it could make use of the video frequency before access would be given to the other Scout's behavior.

When a behavior was granted access to the video frequency and could control its Scout, it had to wait three seconds before actually being able to make use of the video signal. This was done because the Scout video transmitter requires 2-3 seconds of warm-up time before the image stabilizes. Thus, Scouts effectively had seven seconds of useful work time when they were granted access to all of their resources.

Figure 11 shows the hiding places found for all trials of all cases. Over all the trials, the Scouts were able to hide themselves 90% of the time. In the remaining 10% of the time, the Scouts reached a 60 second time-out on the Drive-Toward-Goal behavior, and stopped out in the open where they could be more easily seen and bumped into. This time-out was required because the Scouts are unable to determine with confidence if progress is being

made in moving towards a hiding position. This time-out was also encountered on some successful hiding trials, as the Scout continued to try to progress to a darker hiding position, even after reaching cover. For this reason, the non-hiding trials are not considered outliers in the time data.

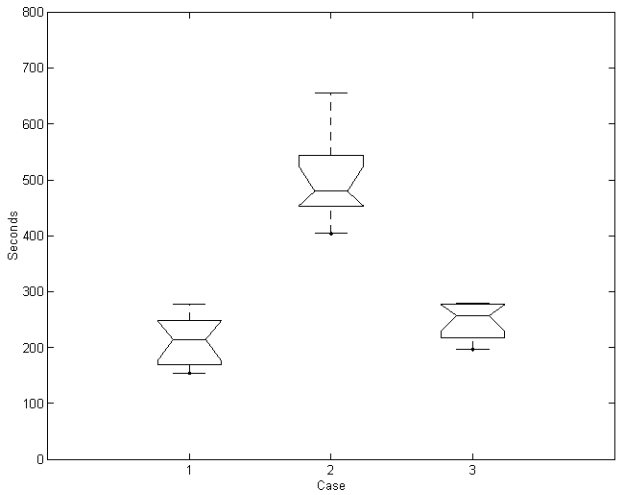


Figure 12: Experiment B. Hiding and Viewing a Room. The average time (in seconds) that it took the Scouts to complete each trial, shown for the three different cases: (1) one Scout, (2) two Scouts on a single frequency and (3) two Scouts on two different frequencies. The times are plotted using a box representation where the center line is the median value, the top and bottom lines of the box represent the upper and lower quartile values respectively, and the lines at the top and bottom of each plot represent the rest of the distribution. The notches in the box represent an estimate of the distribution's mean.

Once the Scouts had positioned themselves in the environment, they attempted to orient themselves to view a lighted area of the room. Figure 12 shows the times for the Scouts to reach their final poses (positions and orientations) in each trial of each case of the experiment. In cases 2 and 3, there are two Scouts per trial, so each value plotted for those cases is the average time it took for the Scouts to reach their final poses. As can be seen from this figure, two Scouts on a single video frequency (case 2) took longer to reach their final poses than did a single Scout (case 1). This is to be expected—the Scouts are time-multiplexing the video frequency resource. There is also a somewhat greater average time for two Scouts on two different video frequencies (case 3) to reach their final poses than there is for the single scout case (for case 1, mean = 212.50, σ = 44.55; for case 3, mean = 247.50, σ = 30.62), however, these differences are not statistically significant at the 95% confidence level (two-tailed, two-sample t test, $p = 0.0555$).

One interpretation of these results is that one Scout is better than two on the same frequency (as the task is accomplished more quickly by one) and that one Scout and two on different frequencies are approximately equal on this task. However, this ignores the fact that in cases 2 and 3, two Scouts are being introduced into the environment and would likely be able to accomplish more than the single Scout from case 1.

Nonetheless, even if two Scouts could accomplish twice as much as one after reaching

their final poses, one Scout is still better, on average, than two on the same frequency, as the time for case 2 is significantly greater than twice the time for case 1. This is because up to 30% of the time in case 2 is lost waiting for the video transmitter to warm up. For this reason, deploying two Scouts sequentially would often make more sense than deploying them in parallel, if the Scouts must share the video frequency. An instant-on transmitter would eliminate this advantage for sequential deployment.

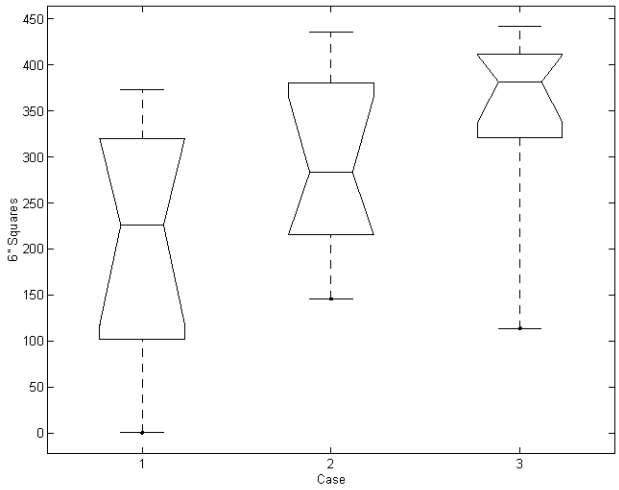


Figure 13: Experiment B. Hiding and Viewing a Room. The total areas (in 6" squares) that the Scouts were able to view with their cameras. (1) one Scout, (2) two Scouts on a single frequency, and (3) two Scouts on two different frequencies.

In contrast, if two Scouts could accomplish twice as much as one after reaching their final poses, then two Scouts on different frequencies are much better than one, as the time per Scout is much less in case 3 than case 1. However, the assumption that two Scouts are twice as good as one is unlikely to hold true.

Since the overall mission of the Scouts is surveillance, one measure of Scout performance after deployment is open area viewed. Figure 13 shows the total area viewed by the Scouts for each case. Considering the area viewed, two Scouts on different frequencies are again seen to be better than one for this task, as the area viewed is larger in case 3 than in case 1 (for case 1, mean = 203.95, σ = 124.57; for case 3, mean = 348.70, σ = 100.89)—this difference is significant (one-tailed, two-sample t test, $p = 0.0053$).

6.3 Detecting Motion

A final experiment was run to test the Scouts' detect motion abilities. Four different cases were tested, including a single Scout using a single video frequency, two Scouts sharing a single video frequency, two Scouts using two different video frequencies, and four Scouts sharing two different video frequencies.

For each of the four cases, the Scouts were placed in ten different positions in the environment. These positions were the same as the hiding positions obtained in the previous experiment. In the case using four Scouts, for which no hiding experiment was run, the

positions were randomly sampled with replacement from the results of the other hiding experiments. In each position, five individual motion detection trials were run, bringing the total number of individual trials to two hundred.

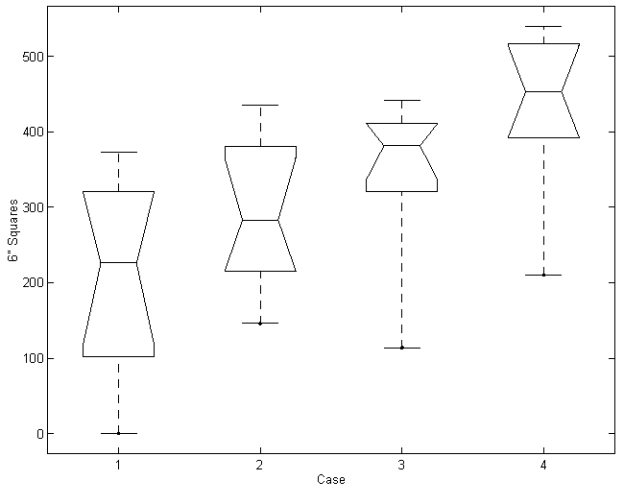


Figure 14: Experiment C. Detecting Motion. The areas (in 6" squares) that the Scouts were able to view with their cameras. (1) one Scout, (2) two Scouts on a single frequency, (3) two Scouts on two different frequencies and (4) four Scouts on two different frequencies.

Figure 14 shows how the total areas viewed by the Scouts in each of the four cases. The area viewed in the four Scout case was significantly greater (at the 95% confidence level) than the areas viewed in the other cases, but not by a factor of four over that viewed by one Scout nor by a factor of two over that viewed by two Scouts. The size and configuration of the environment was such that there was usually a great deal of overlap in the areas viewed by individual Scouts. Redundancy was probably not as useful in this environment (two or three Scouts might have sufficed), but would probably be more effective in larger or more segmented environments.

The moving target the Scouts had to detect was a Pioneer 1 mobile robot [7]. A Pioneer was chosen for its ability to repeatedly travel over a path at a constant speed. This reduced some of the variability between experiments that the use of human subjects might have caused.[1] The Pioneer entered the room from the right and made its way over to the left. The Pioneer moved at a speed of approximately 570 mm/s and traversed the length of the room in 8.5 seconds on average. Once it had moved 16 feet into the room, it turned around and moved back out again. With a 4 second average turn time, the average time the Pioneer was in the room was 21 seconds.

Figure 15 illustrates the fields of view seen by two Scouts and the area of the Pioneer's path that they cover. While the views of these Scouts do not overlap, there was a large amount of overlap in some of the other placements of Scouts.

To evaluate the motion detection abilities of the Scout cases and to determine the effect of having to share a video frequency, the actual time that the Pioneer was seen was compared

[1] Additional experimentation showed that the Scouts were at least as good at detecting human motion as they were at detecting the Pioneer.

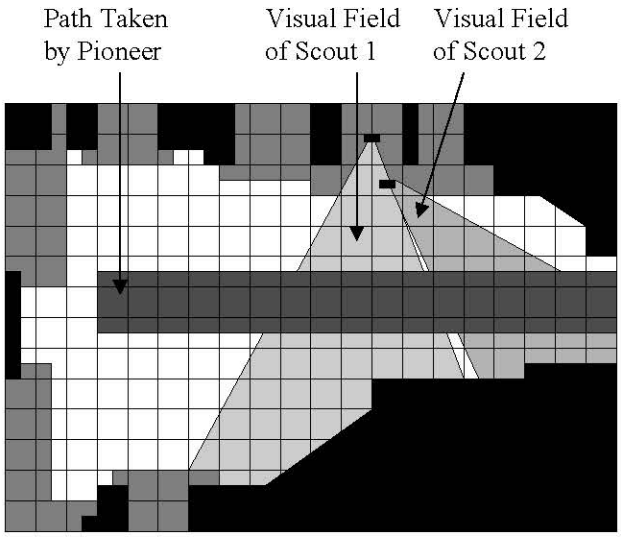


Figure 15: Example Scout placement in the 20 ft. by 14 ft. room. In this instance, there are two scouts that view the path of the Pioneer robot, shown in dark gray in the middle of the room. The fields of view do not happen to overlap between these two Scouts.

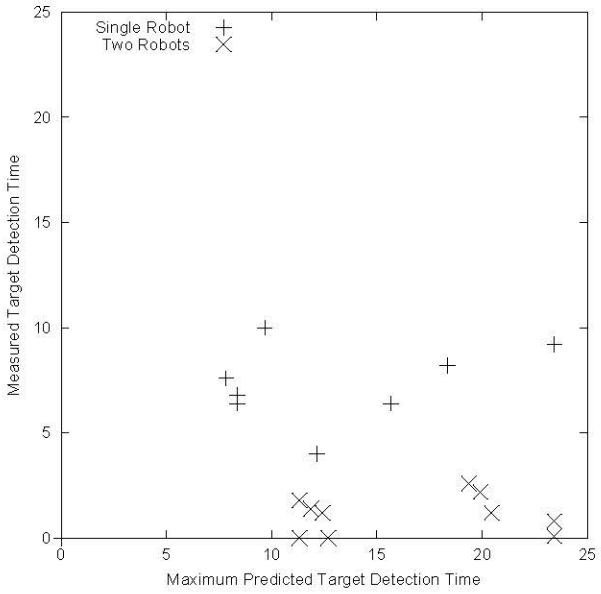


Figure 16: Single frequency cases. The horizontal axis represents the maximum possible time the Pioneer could be detected by the Scouts and the vertical axis represents the time it actually was. The closer these two values are to one another, the better the performance.

to the potential time that the Pioneer could have been seen given the Scout positions.

Figure 16 shows the plot of the cases with a single frequency using one and two Scouts. As can be seen, the one Scout case had a much higher success rate than the two Scout case. This was expected because the robots in the two Scout case were not able to view the entire area at once. Since they had to share a video frequency, they had to take turns

observing their respective fields of view. Since the Pioneer was moving relatively quickly (over half a meter a second), it would be missed by a Scout's camera if the Scout did not have access to the video frequency at that time. In the two Scout case, the video frequency was swapped between the robots every eight seconds. The minimum runtime value was set to eight seconds and the delay which allowed the camera to warm up was set to four seconds (necessary because the detect motion behavior is more sensitive to noise than the navigation behaviors). This value was chosen to give the Scouts a low latency between observations but would not lower the duty cycle (percentage of time active to inactive) beyond 50%.

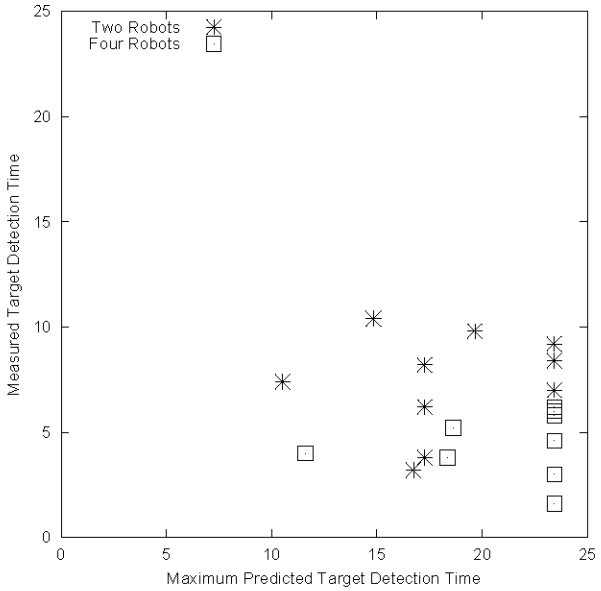


Figure 17: Double frequency cases. The horizontal axis represents the maximum possible time the Pioneer could be detected by the Scouts and the vertical axis represents the time it actually was. The closer these two values are to one another, the better the overall performance.

In the two Scout with two frequency case, there was no need to do any scheduling of the video transmissions as each Scout had full access to its own video channel. In the four Scout with two frequency case, each video channel was shared by two Scouts and the video frequency was swapped between the Scouts once every eight seconds (like the two Scouts with one frequency case). The plots of the actual time the Pioneer was detected compared to the potential time the Pioneer could have been detected for these two experiments is shown in Figure 17.

In this plot, the data points for the four Scout experiments are clustered in the lower right corner of the graph. This indicates that in most experiments, the entire path of the Pioneer was in the possible viewing area of a Scout. However, one of the reasons that the Pioneer was not detected for the full duration of its travel was because of the nature of the detect motion algorithm. If the Pioneer moved perpendicularly across the Scout's field of view, the Scout was able to detect it quite easily. If the Pioneer moved parallel to the optical axis of the Scout's field of view, the Scout would have a very difficult time detecting it due

to the relatively little change in detected motion between one frame and the next. Motion would be detected only when the Pioneer came very close to the Scout.

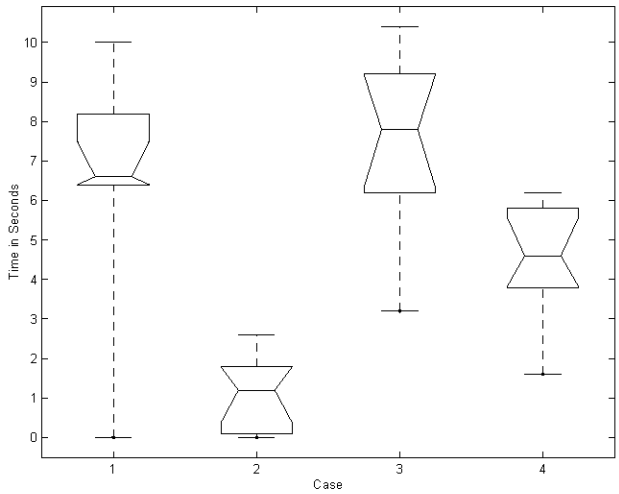


Figure 18: Experiment C. Detecting Motion. The actual time (in seconds) that the Scouts detected the motion of the Pioneer with their cameras. (1) one Scout, (2) two Scouts on a single frequency, (3) two Scouts on two different frequencies and (4) four Scouts on two different frequencies.

To make more explicit the effect of sharing the same frequency, Figure 18 shows the performance of the Scouts in the same experiments shown earlier.

The area traversed by the Pioneer that was visible to the Scouts and the amount of time the Pioneer was visible were different across experiments. This was caused by the fact that the Scouts did not always hide in the best positions for viewing the Pioneer. In some experiments, one Scout was facing the wall instead of facing the open area, and so it did not contribute to the detection task at all. In other cases, two Scouts were very close with viewing areas almost completely overlapping.

Figure 19 and Figure 20 show respectively the area traversed by the Pioneer that was in the field of view of the Scouts and the time the Pioneer was in the field of view of the Scouts for the different experiments. This gives an indication of the complexity of the task. The smaller the area and the shorter the time, the smaller is the opportunity for the Scout(s) to detect the Pioneer even when there is no swapping because a single frequency is used. The figures also illustrate the advantages of using a larger number of Scouts. Both the viewable area traversed by the Pioneer and the time that the Pioneer was in view have higher means and smaller variances when more Scouts were used. This provides a justification for the use of more Scouts than strictly needed to cover the area. Given the chance the Scouts will not hide in good places, using more Scouts reduces the variability in the results and provides more opportunities for the detection of motion.

However, we should caution that the differences were not always statistically significant at the 95% confidence level. In particular, four robots were found to be significantly better than one in these measures but four robots were not found to be significantly better than two on different frequencies for either measure and two robots on the same frequency were

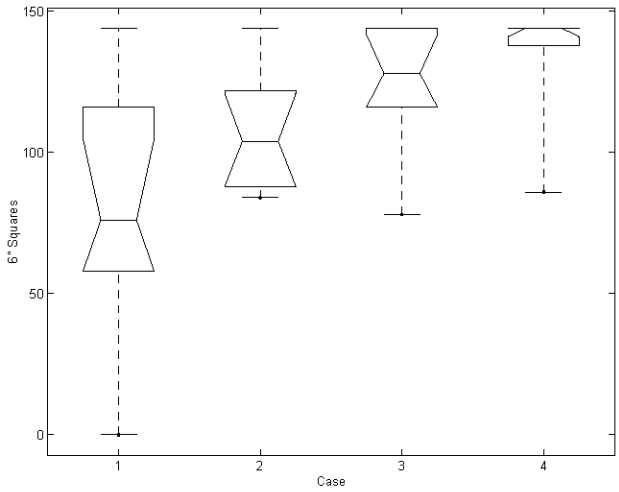


Figure 19: Experiment C. Detecting Motion. The areas (in 6" squares) traversed by the Pioneer that the Scouts were able to view with their cameras. (1) one Scout, (2) two Scouts on a single frequency, (3) two Scouts on two different frequencies and (4) four Scouts on two different frequencies.

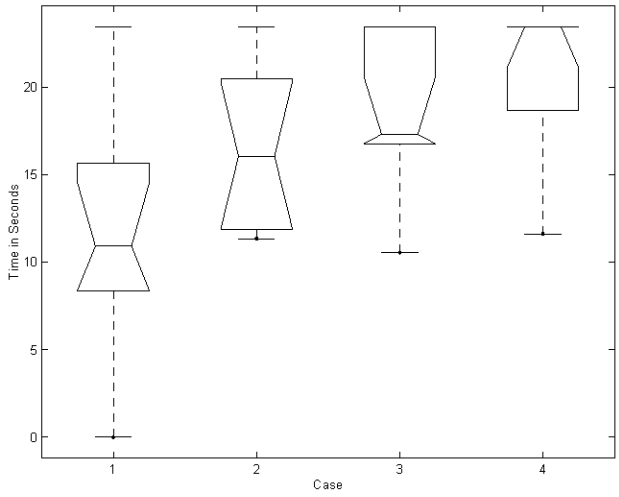


Figure 20: Experiment C. Detecting Motion. The potential time (in seconds) that the Scouts could have been seen the Pioneer with their cameras. This is calculated as the amount of time the Pioneer was in the field of view of a Scout even if the Scout wasn't active at the time. (1) one Scout, (2) two Scouts on a single frequency, (3) two Scouts on two different frequencies and (4) four Scouts on two different frequencies.

not found to be significantly better than one for Pioneer path area viewed. This is due to the overall better placement of two Scouts using two different frequencies than two Scouts on the same frequency. If the two robot results (cases 2 and 3) are pooled to give a larger sample size, then two Scouts are significantly better on these measures than one, and four are significantly better than two. Pooling these results is likely justified, as the differences between their means are nowhere near significantly different, but we cannot rule out the

slight possibility that these results are real effects of the differences in robot interactions in these two cases, rather than simple random noise.

7 Related Work

Automatic security and surveillance systems using cameras and other sensors are becoming more common. These typically use sensors in fixed locations, either connected ad hoc or, increasingly, through the shared communication lines of "intelligent buildings" [8, 9]. These may be portable to allow for rapid deployment [10] but still require human intervention to reposition when necessary. This shortcoming is exacerbated in cases in which the surveillance team does not have full control of the area to be investigated. Static sensors have another disadvantage–they do not provide adaptability to changes in the environment or in the task. In case of poor data quality, for instance, we might want the agent to move closer to its target in order to sense it better.

Mobile robotics can overcome these problems by giving the sensor wheels and autonomy. Robotics research for security applications has traditionally focused on single, large, independent robots designed to replace a single human security guard as he makes his rounds [11]. Such systems are now available commercially and are in place, for example, in factory, warehouse, and hospital settings [12, 13], and research continues along these lines [14, 15, 16]. However, the single mobile agent is unable to be in many places at once—one of the reasons why security systems were initially developed. Further, large mobile robots are unable to conceal themselves, which they may need to do in hostile or covert operations. They may also be too large to explore tight areas.

Multiple robots often can do tasks that a single robot would not be able to do or can do them faster, as described in the extensive surveys by Cao *et al.* [17] and C. Weisbin *et al.* [18]. The tasks traditionally studied with multiple robots are foraging [19], which involves searching and retrieving items from a given area; formation marching [20], which involves moving while maintaining a fixed pattern; map making [21]; searching an area [22] and janitorial service [23], where robots have to clean a room in an unfamiliar building by emptying the garbage, dusting the furniture, and cleaning the floor.

Multiple mobile robots for security have recently been investigated [24]. In this case, the robots were meant to augment human security guards and fixed sensor systems in a known and semi-tailored environment.

To control a group of robots, the software architecture must allow for distributed operations and facilitate allocation and use of resources. Multiple architectures have been proposed to support fault-tolerant execution of plans for single and multiple robot systems. Examples span from support for high-level mission specification [25] and task planning [26] to situated control [27], fault-tolerant control [28], control of combinations of real and virtual robots [29], and robust execution of distributed plans [30]. The architecture we presented provides support for distribution of resources across robots, use of shared resources, and integration in a seamless way with autonomous and human-supervised control.

Most existing multi-robot systems have either sufficient computing power on board or have a central controller to do vision processing for all of them. For instance, the robots used for RoboCup [31] fall into either of those two cases: the robots used in the large league do

all the computations on board, while the robots used in the small league rely on an overhead camera to track the actions of all the robots at once. Our robots are very small yet they cover a wide area, so we are forced to rely heavily on the communications system both for proxy-processing and for image processing.

Recently there has been a significant interest in miniature robots. Constructing robots that are small, easily deployable, and yet can do useful work and operate reliably over long period of times has proven to be very difficult. Many problems suggest the use of miniature robots [32]. Most miniature robots have wheels [33, 34, 35], others can jump [36], roll [37], fly [38], or swim [39]. Because of their limitations, their use has been limited to research laboratories or to specialized environments, such as the small league of RoboCup.

Due to the small size, most miniature robots have to use proxy processing, as in Inaba *et al.* [40], and communicate via a wireless link with the unit where the computation is done. This becomes a problem when the bandwidth is limited, as in the case of our Scout robots. Because of their limited size, not only is all processing for the Scout is done off-board but also the communication is also done only using RF on a few communications channels. This limits severely the ability to control multiple robots at once.

The problem we address in this paper is assessing the effects of limited communication bandwidth on the execution of tasks more than what strategies will guarantee an optimal allocation of the resources. A wide body of literature exists in the area of real-time scheduling algorithms [41]. We plan on experimenting with additional scheduling and negotiation algorithms for resource allocation. For instance, a promising method for negotiation based on quality of service has been proposed [42] in the context of automated flight control.

8 Summary and Future Work

Visual behaviors for simple autonomous operations of a group of Scout robots have been presented. Experimental results illustrating the ability of the Scout to position itself in a location ideal for detecting motion (such as in a reconnaissance task) and the ability to detect motion have also been shown. Future work is planned to allow the Scouts to make use of additional sensor interpretation algorithms for more complex environmental navigation. Ultimately, we hope to have the Scouts construct a rudimentary topological map of their surroundings, allowing other robots or humans to benefit from their explorations.

We have also presented some important system issues related to the control of multiple robots over a low bandwidth communications channel. We have described a distributed software control architecture designed to address these issues. An essential feature of the architecture is the ability to dynamically schedule access to physical resources, such as communication channels, framegrabbers, etc. that have to be shared by multiple robots.

We have demonstrated how the communications bottleneck affects the overall performance of the robots. We have also demonstrated initial results of how our system degrades under increased load. The next step is to add more intelligence into the behaviors which will allow them to dynamically adjust their requested runtimes to react to the situation. Additionally, we are examining other kinds of RF communications hardware to see whether we are able to increase the number of video channels, which is the main limiting factor in our system. The difficulty lies in the Scout's extremely small size and power supply. Very

few transmitters exist that meet the requirements of our hardware. We believe that a combination of intelligent scheduling and more flexible hardware will allow a larger number of Scout robots to operate simultaneously in an effective manner.

Acknowledgements

Material based upon work supported in part by the Defense Advanced Research Projects Agency, Microsystems Technology Office (Distributed Robotics), ARPA Order No. G155, Program Code No. 8H20, issued by DARPA/CMD under Contract #MDA972-98-C-0008 and in part upon work supported by the Doctoral Dissertation Fellowship program at the University of Minnesota.

References

[1] P. E. Rybski, N. Papanikolopoulos, S. A. Stoeter, D. G. Krantz, K. B. Yesin, M. Gini, R. Voyles, D. F. Hougen, B. Nelson, and M. D. Erickson, "Enlisting rangers and scouts for reconnaissance and surveillance," *IEEE Robotics and Automation Magazine*, vol. 7, no. 4, pp. 14–24, Dec. 2000.

[2] S. A. Stoeter, P. E. Rybski, M. D. Erickson, M. Gini, D. F. Hougen, D. G. Krantz, N. Papanikolopoulos, and M. Wyman, "A robot team for exploration and surveillance: Design and architecture," in *The Sixth International Conference on Intelligent Autonomous Systems*, Venice, Italy, July 2000, pp. 767–774.

[3] Object Management Group, *The Common Object Request Broker: Architecture and Specification*, Object Management Group, 1998.

[4] C. L. Liu and J. W. Layland, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, pp. 46–61, 1973.

[5] R. Hoffman and E. Krotkov, "Terrain mapping for outdoor robots: Robust perception for walking in the grass," in *Proc. of the IEEE Int'l Conference on Robotics and Automation*, 1993, pp. 529–533.

[6] P. E. Rybski, S. A. Stoeter, M. D. Erickson, M. Gini, D. F. Hougen, and N. Papanikolopoulos, "A team of robotic agents for surveillance," in *Proc. of the Int'l Conf. on Autonomous Agents*, Barcelona, Spain, June 2000, pp. 9–16.

[7] ActivMedia, Inc., Peterborough, NH, *Pioneer Operation Manual v2*, 1998.

[8] J. Porteous, "Intelligent buildings and their effect on the security industry," in *IEEE Int'l Carnahan Conference on Security Technology*, Larry D. Sanson, Ed., Sanderstead, Surrey, England, Oct. 1995, pp. 186–188.

[9] B. Horling, R. Vincent, R. Mailler, J. Shen, R. Becker, K. Rawlins, and V. Lesser, "Distributed sensor network for real time tracking," in *Proc. of the Int'l Conf. on Autonomous Agents*, June 2001.

[10] D. Pritchard, R. White, D. Adams, E. Krause, E. Fox, M. Ladd, R. Heintzleman, P. Sprauer, and J. MacEachin, "Test and evaluation of panoramic imaging security sensor for force protection and facility security," in *IEEE Int'l Carnahan Conference on Security Technology*, Alexandria, VA, Oct. 1998, pp. 190–195, Larry D. Sanson, ed.

[11] T. Kajiwara, J. Yamaguchi, Y. Kanayama, S. Yuta, and J. Iijima, "Development of a mobile robot for security guard," in *Proceedings of the 15th Int'l Symposium on Industrial Robots*, Tokyo, Japan, 1985, vol. 1, pp. 271–278.

[12] A. Kochan, "HelpMate to ease hospital delivery and collection tasks, and assist with security," *Industrial Robot*, vol. 24, no. 3, pp. 226–228, 1997.

[13] C. Pellerin, "Twenty-first century sentries," *Industrial Robot*, vol. 20, no. 2, pp. 15–17, 1993.

[14] R. Dillmann, M. Kaiser, F. Wallner, and P. Weckesser, "PRIAMOS: An advanced mobile system for service, inspection, and surveillance tasks," in *Modelling and Planning for Sensor Based Intelligent Robot Systems*, vol. 21 of *Series in Machine Perception and Artificial Intelligence*. World Scientific, Singapore, 1995.

[15] A. Osipov, V. Kemurdjian, and B. Safonov, "Mobile robots for security," in *Proceedings of the 1996 2nd Specialty Conference on Robotics for Challenging Environments, RCE-II*, Albuquerque, NM, 1996, pp. 290–295.

[16] M. Saitoh, Y. Takahashi, A. Sankaranarayanan, H. Ohmachi, and K. Marukawa, "Mobile robot testbed with manipulator for security guard application," in *Proc. of the IEEE Int'l Conference on Robotics and Automation*, Nagoya, Japan, 1995, vol. 3, pp. 2518–2523.

[17] Y. Cao, A. Fukunaga, and A. Kahng, "Cooperative mobile robotics: Antecedents and directions," *Autonomous Robots*, vol. 4, no. 1, pp. 7–27, 1997.

[18] C. Weisbin, J. Blitch, D. Lavery, E. Krotkov, C. Shoemaker, L. Matthies, and G. Rodriguez, "Miniature robots for space and military missions," *IEEE Robotics and Automation Magazine*, vol. 6, no. 3, pp. 9–18, Sept. 1999.

[19] M. Matarić, "Behavior-based control: Examples from navigation, learning, and group behavior," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 9, no. 2–3, pp. 323–336, 1997.

[20] T. R. Balch and R. C. Arkin, "Behavior-based formation control for multiagent robot teams," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 6, pp. 926–939, Dec. 1998.

[21] W. Burgard, M. Moors, D. Fox, R. Simmons, and S. Thrun, "Collaborative multi-robot exploration," in *Proc. of the IEEE Int'l Conference on Robotics and Automation*, Apr. 2000, pp. 476–481.

[22] Daisuke Kurabayashi, Jun Ota, Tamio Arai, and Eiichi Yoshida, "An algorithm of dividing a work area to multiple mobile robots," in *Proc. of the IEEE Int'l Conference on Robotics and Automation*, 1995, pp. 286–291.

[23] L. E. Parker, "On the design of behavior-based multi-robot teams," *Journal of Advanced Robotics*, vol. 10, no. 6, pp. 547–578, 1996.

[24] H. R. Everett and D. W. Gage, "From laboratory to warehouse: Security robots meet the real world," *Int'l Journal of Robotics Research*, vol. 18, no. 7, pp. 760–768, July 1999.

[25] D. MacKenzie, R. C. Arkin, and R. Cameron, "Specification and execution of multiagent missions," *Autonomous Robots*, vol. 4, no. 1, pp. 29–57, Jan. 1997.

[26] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, "Multi-robot cooperation in the MARTHA project," *IEEE Robotics and Automation Magazine*, vol. 5, no. 1, pp. 36–47, Mar. 1998.

[27] B. Werger and M. J. Mataric, "From insect to internet: Situated control for networked robot teams," *Annals of Mathematics and Artificial Intelligence*, vol. Fall, 2000.

[28] L. E. Parker, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Transactions on Robotics and Automation*, vol. 14, no. 2, pp. 220–240, Apr. 1998.

[29] K. Dixon, J. Dolan, W. Huang, C. Paredis, and P. Khosla, "Rave: A real and virtual environment for multiple mobile robot systems," in *Proc. Int'l Conf on Intelligent Robots and Systems (IROS'99)*, 1999.

[30] E. M. Atkins, T. F. Abdelzaher, K. G. Shin, and E. H. Durfee, "Planning and resource allocation for hard real-time, fault-tolerant plan execution," *Autonomous Agents and Multi-Agent Systems*, vol. 4, no. 1/2, Mar. 2001.

[31] H. Kitano, "Special issue – RoboCup: The first step," *Artificial Intelligence*, vol. 110, no. 2, June 1999.

[32] D. Gage, "Minimum-resource distributed navigation and mapping," in *SPIE Mobile Robots XV, (SPIE Proceedings Volume 4195)*, Nov. 2000.

[33] G. Caprari, P. Balmer, R. Piguet, and R Siegwart, "The autonomous micro robot ALICE: A platform for scientific and commercial applications," in *Proc. of 1998 Int'l Symposium on Micromechatronics and Human Science (MHS'98)*, Nov. 1998.

[34] R. Grabowski, L. E. Navarro-Serment, C. J. J. Paredis, and P. Khosla, "Heterogeneous teams of modular robots for mapping and exploration," *Autonomous Robots*, vol. 8, no. 3, pp. 293–308, 2000.

[35] F. Mondada, E. Franzi, and P. Ienne, "Mobile robot miniaturisation: A tool for investigation in control algorithms," in *Experimental Robotics III, Proc. of the 3rd Int'l Symposium on Experimental Robotics*, Kyoto, Japan, Oct. 1993, pp. 501–513, Springer Verlag, London.

[36] A. Halme, T. Schönberg, and Y. Wang, "Motion control of a spherical mobile robot," in *4th Int'l Workshop on Advanced Motion Control*, 1996.

[37] B. Chemel, E. Mutschler, and H. Schempf, "Cyclops: Miniature robotic reconnaissance system," in *Proc. of the IEEE Int'l Conference on Robotics and Automation*, 1999, pp. 2298–2303.

[38] A. S. Wu, A. C. Schultz, and A. Agah, "Evolving control for distributed micro air vehicles," in *Proc. IEEE Int'l Symposium on Computational Intelligence in Robotics and Automation*, Monterey, CA, Nov. 1999.

[39] T. Fukuda, A. Kawamoto, and K. Shimojima, "Acquisition of swimming motion by RBF fuzzy neuro with unsupervised learning," in *ALIFE V*, 1996, pp. 31–37.

[40] M. Inaba, S. Kagami, F. Kanechiro, K. Takeda, O. Tetsushi, and H. Inoue, "Vision-based adaptive and interactive behaviors in mechanical animals using the remote-brained approach," *Robotics and Autonomous Systems*, vol. 17, pp. 35–52, 1996.

[41] J. Stankovic, M. Spuri, K. Ramamritham, and G. Buttazzo, *Deadline Scheduling For Real-Time Systems: EDF and Related Algorithms*, Kluwer Academic Publishers, Boston, 1998.

[42] T. Abdelzaher, E. Atkins, and K. Shin, "QoS negotiation in real-time systems and its application to automated flight control," *IEEE Trans. Computers*, vol. 49, no. 11, pp. 1155–1169, Nov. 2000.