Effects of Limited Bandwidth Communications Channels on the Control of Multiple Robots *

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

Center for Distributed Robotics
Department of Computer Science and Engineering
University of Minnesota, Minneapolis, MN USA

Abstract

We describe a distributed software system for controlling a group of miniature robots using a very low capacity communication system. Space and power limitations on the miniature robots drastically restrict the capacity of the communication system and require sharing bandwidth and other resources among the robots. We have developed a process management/scheduling system that 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 describe a surveillance task in which the robots patrol an area and watch for motion, and present experimental results.*

1 Introduction

A restricted communication pipeline can be a limiting factor in the control of a distributed robotic system. In some robotic implementations, this problem can be addressed with large capacity communications hardware (such as a wireless Ethernet). Other robotic systems of interest cannot use high-capacity communications because of size, power or computational bandwidth limitations. In these cases, addressing systems issues such as process scheduling becomes critical.

We describe a case study of a group of miniature robots which must use very low capacity RF communications systems due to their small size. The size limitations of these robots require them to rely on off-board processing. Thus, these robots are completely dependent on their RF systems in order to operate. 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 assigns resources to each control process 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 miniature robotic systems, called Scouts [6], which are designed for surveillance tasks. A Scout, shown in Figure 1, is a cylindrical robot 11 cm in length and 4 cm in diameter.

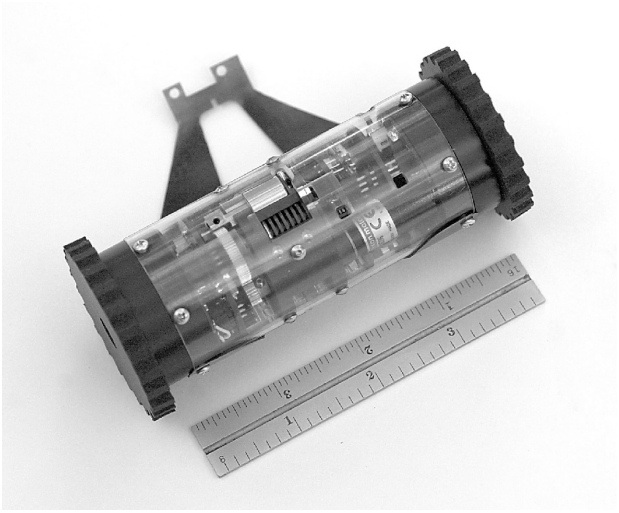

Figure 1: The Scout robot shown next to a ruler (in inches) for scale.

*This material is based upon work supported 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.

†Corresponding author

Scouts can roll over smooth surfaces (up to a 20 degree slope), and are capable of jumping over objects 30 cm in height by using their spring-loaded tails. The Scouts can transmit video from their cameras and communicate digital information on separate RF channels.

Due to the highly restrictive space and power constraints of the robot, its two on-board CPUs 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 captured by its camera. In order for the Scouts to do anything useful, they must be paired with an off-board decision process, such as a workstation or a human teleoperator.

A fixed-frequency command radio is used to transmit command packets to the Scouts. 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. If the radio throughput is high enough, real-time performance can be achieved.

The Scouts broadcast video data over a fixed-frequency analog radio link which must be captured by a video receiver and fed into a digitizer. Because the video is a continuous analog stream, only one robot can transmit on a given frequency at a time. Signals from multiple robots interfere with each other and become useless. Due to space and power restrictions on the current Scout design, we only have two different video frequencies, limiting the simultaneous broadcasting of video. Video from multiple robots can be captured by interleaving the time which each robot's transmitter is on.

The RF limitations of the Scout pose several fundamental difficulties when trying to control several of them simultaneously. 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.

3 Software Architecture

Decision processes such as behaviors or planning algorithms need access to all of the individual resources that are necessary to move the physical robots. To address this, we have designed a software architecture which is capable of connecting groups of decision processes with the physical resources in the system [14]. The system is broken down into 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. Behaviors are organized in a hierarchical fashion, where "parent" nodes spawn off "children" to do various tasks. Priorities are assigned to behaviors to determine how to allocate resources.

3.2 Resource Pool

The *Resource Pool* 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. Access to this RC must be granted before a behavior can use that resource.

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 processes called AGGREGATE RESOURCE CONTROLLERS (or ARCs). For the sake of consistency, all RCs can only be accessed through the use of ARCs, even if an ARC handles only a single RC.

3.3 User Interface

Direct human control of the resources in the system is provided through the *User Interface* (UI) subsystem. Using a UI console, a human can take control over an autonomously controlled Scout and then release it back to the behavior that was controlling it before.

3.4 Backbone

Tying all of these subsystems together in a seamless fashion across a network of computers is a CORBA-based [9] group of core services called the *Backbone*. The Backbone also oversees the distribution and access to the ARCs and RCs through the use of the RESOURCE CONTROLLER MANAGER.

4 Dynamic Resource Allocation

When instantiated by a behavior or UI console, each ARC must be told which specific RCs to use. This information is either decided ahead of time (hard-coded into the behavior) or can be obtained from an on-line database.

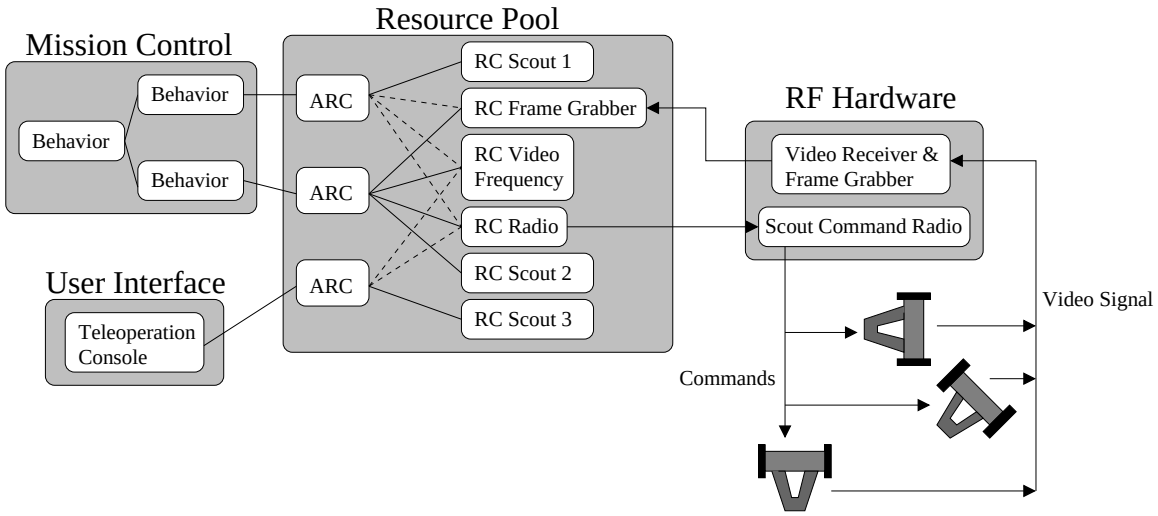

Figure 2: An Instance of the process management/scheduling system. Three Scouts are controlled by a combination of behaviors and a teleoperation console. All three share the same video frequency, so only one robot can be controlled at a given time. Solid lines indicate active connections (where data can flow between components) while dashed lines indicate connections that are not currently active but may be active later during the scheduling process. Components from the Backbone, such as the RESOURCE CONTROLLER MANAGER, are not shown.

4.1 Examples of ARCs and RCs

In order for a process to control a single Scout robot, several physical resources are required. First, a robot must be selected which is not currently in use by another process. Second, a command radio is needed which has enough capacity to handle the demands of the process (refer to Section 4.3 for a discussion about sharable RCs). Third, if the Scout robot is to transmit video, exclusive access to a fixed video frequency must be obtained. Finally, to process the video, a frame grabber attached to a tuned video receiver is required. Each instance of these four resources is managed by its own RC.

Figure 2 illustrates the interconnections between the various components in the system. In this example, a behavior tree is responsible for controlling two robots and a UI teleoperation console lets a human control a third. Each component has its own ARC which attempts to gain access to the appropriate resources. In this example, 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 ARCs belonging to the behaviors must share the video frequency and frame grabber RCs. The ARC owned by the teleoperation console does not need the frame grabber 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 must have their access scheduled by the RESOURCE CONTROLLER MANAGER.

4.2 The Resource Controller Manager

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 (generally on the order of seconds to minutes).

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 of them 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 operating time into slices. ARCs which have a sharable RC in common may be able to run simultaneously, assuming that the total bandwidth requirements for that sharable RC do not exceed its total capacity.

ARCs that have higher priorities are given precedence over ARCs with lower priorities. The RE-

SOURCE 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. This may be changed in future versions if it detracts from the overall usability of the system. Trying to maintain this priority structure can work against the system's goal of maximizing resource utilization. However, in the design specification for this architecture, maintaining the priority structure of the behaviors was deemed more important than overall system throughput.

4.3 Sharable Resources

Sharable RCs, such as the Scout radio, have to manage their own schedule to ensure that each of the ARCs using them can do so at their requested bandwidth. In order to streamline the scheduling process, commands sent to sharable RCs must have a constant interval between invocation. In addition, each request must be completed before the next request 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 simple rate monotonic algorithm (RMA) [7] is used.

RMA generates optimal schedules by giving precedence to processes with higher request frequencies. Once again, the user-set priorities must be maintained when schedule is computed. Thus, higher user-set priority ARCs have precedence over lower user-set priority ARCs regardless of the frequency of the requests. This often produces a schedule which is suboptimal in its usage of the RC, but which maintains the user-set priority relationships.

5 A Distributed Surveillance Task

We are developing a distributed surveillance task in which the Scouts are deployed into an area to 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 are assumed to already be deployed into the environment by a human or another robot [11]. Their cameras are aimed at specific locations where motion is likely to happen (such as by a doorway or in a hallway). The environment is assumed to be large enough that a single Scout cannot view all the area at once. In each experiment, 20 trials were run.

5.1 Experimental Results

To illustrate how the system works and to show how the low-bandwidth communications channels can affect the robots, a set of experiments is described in which several Scouts perform a simple distributed motion-detection task.

As shown in Figure 3, Scout 1 is placed to watch the entrance to the room and Scout 2 is placed to watch much of the inside of the room.

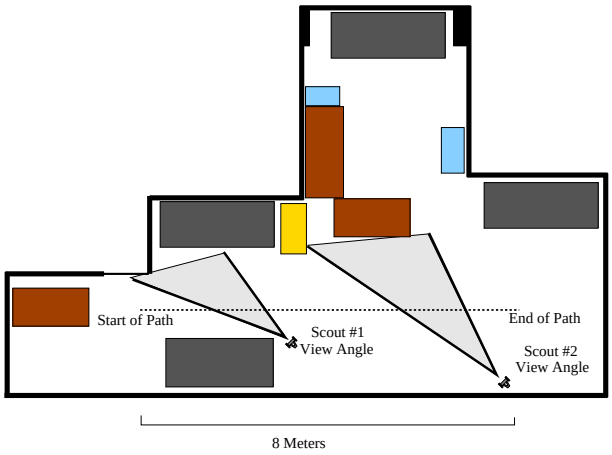

Figure 3: Top view of the room where the motion detection experiment took place. The square objects are tables and other pieces of furniture. The field of views of both Scout robots are shown as wedges and the path the moving object took through the room is shown as a dotted line.

The viewpoints of both Scout cameras intersect the main thoroughfare through the room. Anything traveling through this area would have to travel through both areas of coverage. In these experiments, both robots share the same video frequency, and so each robot's video transmission is scheduled in a simple round-robin fashion. Each robot's behavior is allowed to run for five seconds in order to capture a good number of clean frames. At the end of five seconds, the behavior is swapped out (shutting off the Scout's video), allowing the other behavior to capture video.

In the experimental setup, a Pioneer 1 mobile robot [2] enters the room from the left and makes its way to the right. The Pioneer was chosen for its ability to repeatedly travel over a path at an identical speed*.

*Several quick experiments showed that the system was just

Four different sets of experiments are run where the system load is varied between each experiment. In the first experiment, two Scouts are used to detect motion (referred to as the active Scouts). In the second experiment, a user requests a teleoperation console to access a third Scout while the first two active Scouts are doing their surveillance task. The system dynamically adjusts the new schedule to allow the new request and must reduce the available resource time for the first two Scouts. In the third experiment, two extra Scout teleoperation requests are made and in the fourth experiment, three extra Scout teleoperation requests are made. In each experiment, the Pioneer 1 moves past the first two Scouts (which do not change their positions between experiments) twenty times. Figure 4 shows the percentage of times the Pioneer was detected and missed by the Scouts.

Active Scouts	Extra Scouts	%Motion Detected	%Motion Missed
2	0	80%	20%
2	1	75%	25%
2	2	60%	40%
2	3	50%	50%

Figure 4: Experimental results showing the effectiveness of the system under increasingly heavy load.

As the system load increases, the ability of the system to function properly decreases. This was expected as the RESOURCE CONTROLLER MANAGER has no knowledge about what kinds of tasks the ARCs and RCs are doing and thus cannot alter the priorities to adjust to increased system load. This is beyond the scope of its design as its primary function is to avoid resource request conflicts among the decision processes.

One interpretation of these results is that the system is incapable of handling any reasonably large number of Scout robots (say larger than 8 or 9). These particular results show performance for a single environment, single position of robots, and single rate at which objects move about in the environment. The more accurate interpretation of these results is that if all of these robots are to share a single video channel and command radio, then the system will not be able to function very effectively as the number of robots increases. This is not a deficiency of the system but rather an indication that higher bandwidth communications channels are required. For instance, by adding a second video channel, the effectiveness of the system will increase. Based on some preliminary experiments, adding a second video channel in this particular experimental setup would have boosted the percentage of detected motion for the two robots (assuming they were on different channels) to 100%. Increasing the number of robots which were not assisting the first two in this task would once again decrease the overall performance. An exact measure of the performance depends on multiple variables, such as the positions of the robots in the environment, the rate at which the target moves, how many video frequencies are used and how many robots use a particular video frequency. A more thorough analysis of how each of these variables affects the performance of the system is beyond the scope of this paper but is included in [12].

as good at detecting human motion as it was for detecting the Pioneer.

6 Related Work

To control a large 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 [8] and task planning [3] to situated control [15], fault-tolerant control [10, 5], and robust execution of distributed plans [4]. Much remains to be done before a general architecture is developed that is applicable to heterogeneous robots and tasks and supports real-time operations under a wide spectrum of conditions with graceful degradation. The architecture we presented provides support for distribution of resources across robots and use of shared resources, and integrates in a seamless way autonomous and human-supervised control.

Limited communication bandwidth is a serious problem when robots have to transmit large amounts of data, such as live video, and when many robots need to share the bandwidth, as in the examples we presented. A wide body of literature exists in the area of real-time scheduling algorithms [13]. We plan on experimenting with various scheduling and negotiation algorithms for resource allocation in the context of our architecture. For instance, a promising method for negotiation for real-time systems has been proposed [1] in the context of automated flight control. The method is based on quality of service and allows users to specify a spectrum of quality of service requests. The problem we address in this paper is assessing the effects of limited communication bandwidth on the execution

of tasks more than strategies for guaranteeing optimal allocation of resources.

7 Conclusions

We have presented some important systems issues related to the control of multiple robots over a low bandwidth communications channel. We have described a distributed software control architecture designed to addresses these issues.

We have demonstrated how the communications bottleneck affects the overall performance of the robots and demonstrated initial results of how our system starts to degrade under increased load. The next step is to add more intelligence into the behaviors which will allow them to dynamically adjust their requested run-times to react to their situations. Additionally, we are examining other kinds of RF communications hardware to see whether we are able to increase the number of open video channels, which is the main limiting factor in our system.

8 Acknowledgements

This research is supported in part by the Doctoral Dissertation Fellowship program at the University of Minnesota.

References

[1] T. Abdelzaher, E. Atkins, and K. Shin. QoS negotiation in real-time systems and its application to automated flight control. *IEEE Trans. Computers*, 49(11):1155–1169, Nov. 2000.

[2] ActivMedia, Inc., Peterborough, NH. *Pioneer Operation Manual 2nd Ed.*, 1998.

[3] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert. Multi-robot cooperation in the MARTHA project. *IEEE Robotics and Automation Magazine*, 5(1):36–47, Mar. 1998.

[4] 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*, 4(1/2), Mar. 2001.

[5] 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.

[6] D. F. Hougen, J. C. Bonney, J. R. Budenske, M. Dvorak, M. Gini, D. G. Krantz, F. Malver, B. Nelson, N. Papanikolopoulos, P. E. Rybski, S. A. Stoeter, R. Voyles, and K. B. Yesin. Reconfigurable robots for distributed robotics. In *Government Microcircuit Applications Conference*, pages 72–75, Anaheim, CA, Mar. 2000.

[7] 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*, 20(1):46–61, 1973.

[8] D. MacKenzie, R. C. Arkin, and R. Cameron. Specification and execution of multiagent missions. *Autonomous Robots*, 4(1):29–57, Jan. 1997.

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

[10] L. E. Parker. ALLIANCE: An architecture for fault tolerant multirobot cooperation. *IEEE Transactions on Robotics and Automation*, 14(2):220–240, Apr. 1998.

[11] 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*, pages 9–16, Barcelona, Spain, June 2000.

[12] P. E. Rybski, S. A. Stoeter, M. Gini, D. F. Hougen, and N. Papanikolopoulos. Performance of a distributed robotic system using shared communications channels. Technical Report 01-031, University of Minnesota Computer Science and Engineering Department, 2001.

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

[14] 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*, pages 767–774, Venice, Italy, July 2000.

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