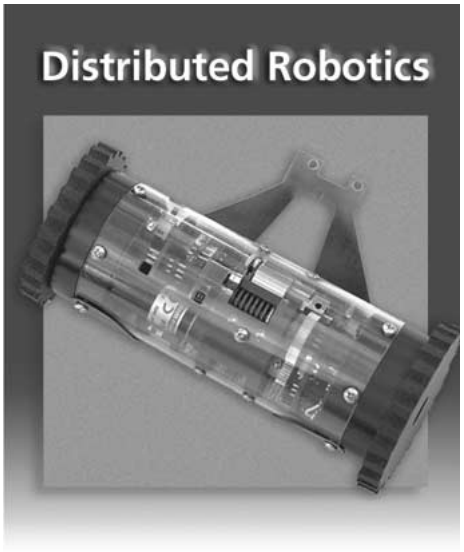

Sharing Control

Presenting a Framework for the Operation and Coordination of Multiple Miniature Robots

By PAUL E. RYBSKI, SASCHA A. STOETER, NIKOLAOS P. PAPANIKOLOPOULOS, IAN BURT, TOM DAHLIN, MARIA GINI, DEAN F. HOUGEN, DONALD G. KRANTZ, and FLORENT NAGEOTTE

Remote surveillance and reconnaissance applications can be greatly facilitated through the use of a small semi-autonomous robotic sensor platform. A human operator equipped with a wearable computer interface can direct a small mobile robotic platform into a dangerous or difficult to reach area and gather information from within. Sensor information is returned to the human through a head-mounted display, and commands are sent from a hand-held controller. Simply teleoperating the robotic platform can be useful in many situations, but the operator's attention must be completely dedicated to controlling the robot. This may be difficult in an emergency situation because of the high likelihood of distractions. To assist the human operator, the robot can be given more autonomy, allowing it to interpret higher-level commands from the human. If a task requires the use of multiple robots, a human operator cannot be expected to simultaneously control them all directly. Instead, the robots must be able to perform useful tasks when the operator's attention is called elsewhere.

This article introduces a layered system that has been developed to facilitate this kind of multimodal control (Fig. 1). This system includes user interfaces (UIs) for teleoperation clients and robust sensor interpretation algorithms for autonomous control clients. A distributed software control architecture dynamically coordinates hardware resources and shares them between the various clients, allowing for simultaneous control of multiple robots.

The article starts with related work and continues with a description of the updated Scout robot (the hardware focus of the system) and the various designs and implementations of the Scout UIs. The distributed control architecture, along with a dynamic resource allocation scheme, are described, followed by a novel mobile robot visual servoing scheme. Finally, some conclusions and future work are presented.

Related Work

Constructing robots that are small and easily deployable, yet able to do useful work and operate reliably over long periods of time has proven to be quite difficult. Our Scout robots promise to be among the first miniature robots ready for field exploration.

To control a large group of robots, the software architecture must allow for distributed operations and facilitate allocation and use of resources. Architectures have been proposed to support fault-tolerant control of multiple robots [1], mission specification [2], and high-level task planning [3]. Much remains to be done before a general architecture is developed that is applicable to heterogeneous robots and different tasks. The architecture we presented provides support for distribution of resources across robots and the use of shared resources and integrates, in a seamless way, autonomous and human-supervised control.

Robots for Surveillance and Reconnaissance

For surveillance and reconnaissance applications, a robot that is small enough to avoid detection and can access hard to reach areas is desirable. Such a robot should be able to transmit environmental information to a remote location for additional analysis. The robot should also be able to receive remote instructions to help guide its search and gather as much useful information as possible.

1070-9932/02$17.00©2002IEEE

The Scout robot [4], shown in Fig. 2, is an innovative miniature robot developed specifically to address these requirements. The robot's small size (11-cm long and 4-cm wide) and light weight (approximately 200 g) allows it to be easily carried by a human or another robot.

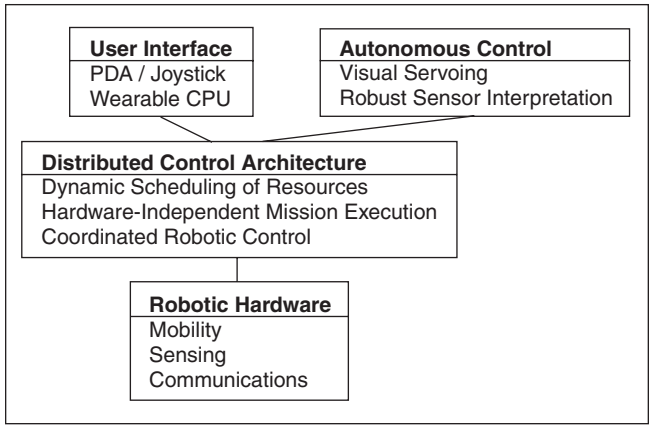

Figure 1. *A layered system architecture for control of multiple robots in both teleoperated and autonomous modes.*

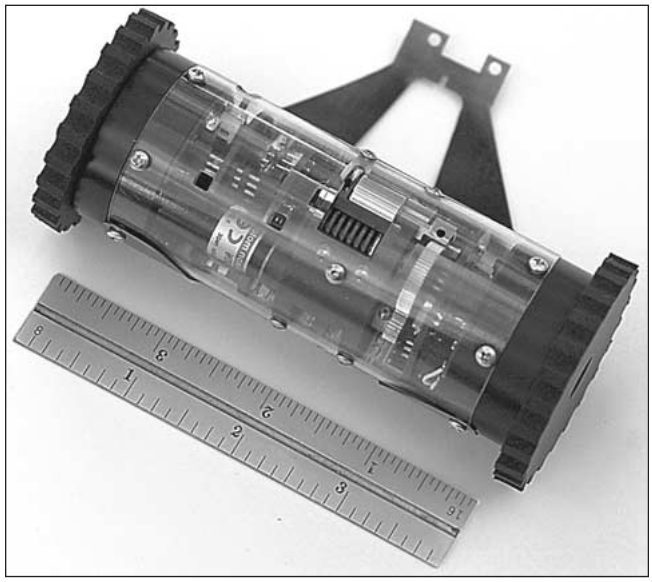

Figure 2. *The Scout robot shown next to a ruler (in) for scale.*

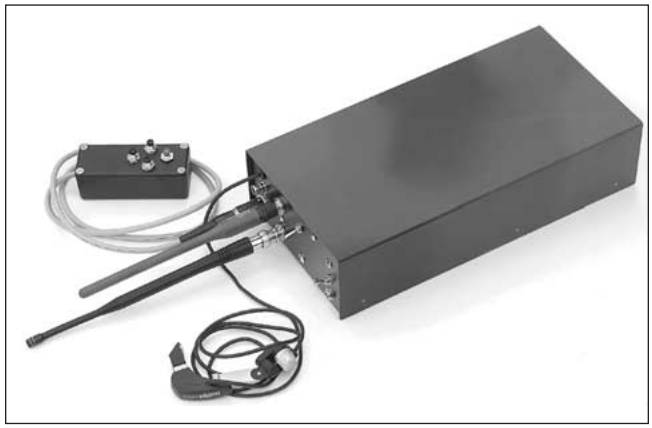

Figure 3. *The wearable Scout controller.*

Scouts carry a small camera that broadcasts environmental information over an analog radio-frequency (RF) transmitter. Scouts can also transmit and receive digital data over a separate RF modem using a custom network protocol. Due to their small size, Scouts have extremely limited onboard computational power, requiring the full capacity of their two CPUs for network communications and actuator control. Motion commands are transmitted from a remote source, where they are received and executed by the robot.

User Interfaces

Traditionally, our robot control interfaces have run on a Linux workstation or laptop, requiring the X Window System to operate. However, having to transport a computer, a Scout radio, and a video receiver can be extremely cumbersome and highly impractical for field use.

To address the issue of portability, a wearable Scout package (Fig. 3) has been developed that contains a Scout radio, a video receiver, and an internal battery pack. The user can send commands to the Scout robot through the use of a hand-held control pendant or joystick interface. The joystick is plugged into a UI CPU board that interprets the input from the controller and commands the data radio CPU to send a command packet to the correct Scout. Video data returned from the Scout is captured on the video receiver board and fed into a head-mounted display controller. A clip-on video display is attached to glasses worn by the user, allowing the video to be observed.

The wearable radio's control pendant requires only a single hand to control basic Scout functions. If more flexible control is desired, an interface running on a commercial PDA (Fig. 4) can be used. The PDA interface is connected to the wearable Scout controller and takes the place of the control pendant to provide a more versatile interface and data visualization system.

To allow for both autonomous and teleoperated control of a Scout from a wearable controller, a wearable PC with a framegrabber must be connected to the system. As shown in Fig. 5, the PC sits between the UI and the wearable controller. The PC receives inputs from the joystick/PDA/pendant and sends the appropriate commands to the radio. Received video is captured by a framegrabber and presented to the user through the clip-on display. This captured video can also be used by autonomous Scout control programs running on the PC. The addition of wireless Ethernet would allow the PC to communicate with other computers on the network and make use of the machine-independent nature of the software control architecture described in the next section.

Distributed Control Architecture

In order to support simultaneous control of multiple robots in both autonomous and teleoperated capacities, a distributed control architecture emphasizing the reusability of software components and the efficient utilization of resources has been designed [5]. This architecture runs on a set of networked computers and can control an arbitrary number of robots. The

software architecture consists of four subsystems: mission control, resource pool, UI, and backbone.

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 can be organized in a hierarchical fashion, where "parent" nodes spawn off "children" to do various tasks. In this way, a task can be decomposed into specific subtasks, where each subtask is addressed by a specific behavior and its children. When a child behavior finishes its task, it returns status information back to its parent, which decides what actions to take from that point.

Behaviors are given priorities that are used to determine how to allocate access to resources. Because behaviors at different levels of the tree describe different levels of detail of a task, it only makes sense to compare priorities between behaviors that share the same immediate parent. To mediate disputes between behaviors that are more distantly related, the tree is traversed to find the nearest ancestors of both behaviors that share the same parent. Only then can a decision be made about which high-level subtask is currently more important to complete.

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, such as robots or radios, and other computational resources, such as framegrabbers, via resource controllers (or RCs). Every physical resource is given its own RC to manage it. If a behavior or another decision process needs to make use of 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 processes called aggregate resource controllers (ARCs). Every ARC is an abstract representation of the group of RCs that it manages, as shown in Fig. 6. An ARC provides a specialized interface into 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. In fact, behaviors are never allowed to interface to the RCs directly. They request an ARC that allows the behavior to control the resources controlled by it.

The central component that oversees the distribution and access to the ARCs and RCs is the RC manager that runs a centralized real-time schedule that dynamically adjusts itself to the current demand of all running decision processes. Behaviors send scheduling requests to the RC manager in order to secure runtime for their ARCs. Each query is parameterized by the specific set of RCs that the ARC needs. The RC manager takes each scheduling request and determines whether

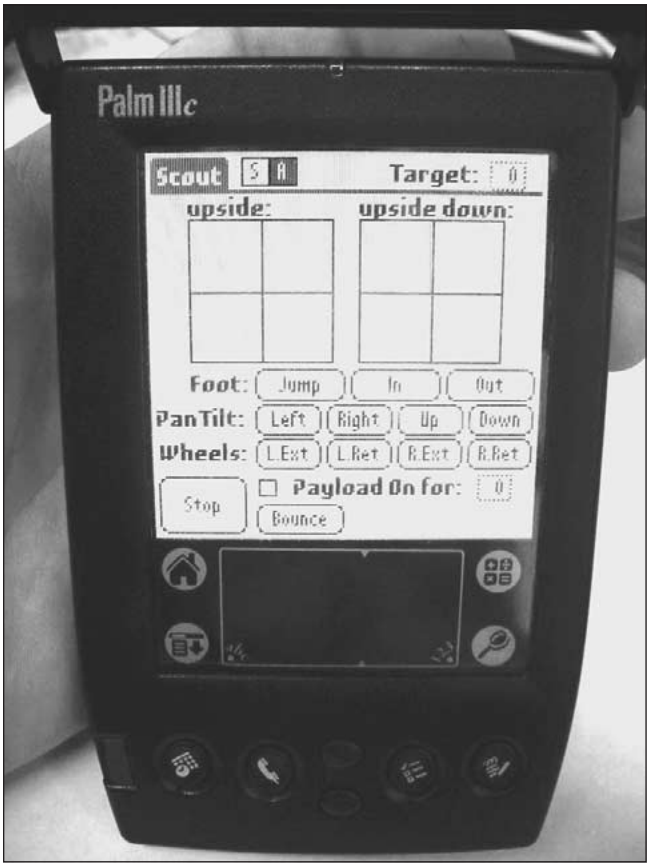

Figure 4. *PDA interface.*

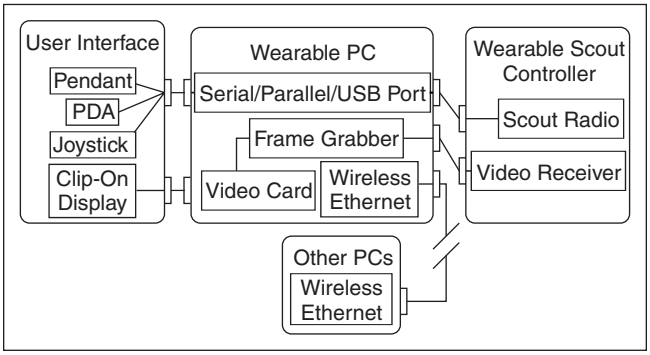

Figure 5. *Connecting a wearable PC to the UI and Scout radios.*

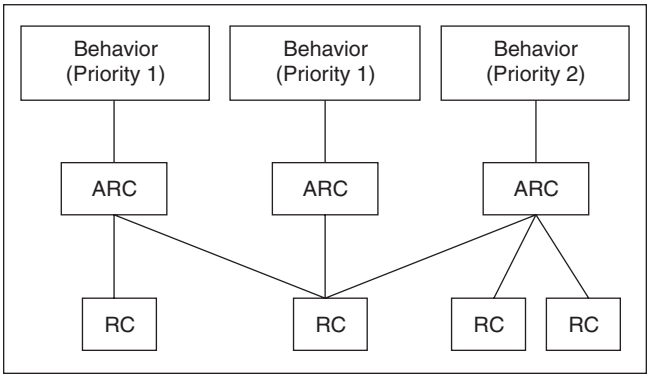

Figure 6. *Example of how behaviors access RCs.*

To control a large group of robots, the software architecture must allow for distributed operations and facilitate allocation and use of resources.

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.

User Interface

Direct human control of the resources in the system is provided through the UI subsystem. This system allows a human operator to connect to a Scout robot directly and command it from a graphical console. A UI 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 RC 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 release it back to the behavior that was controlling it before. Alternatively, very low-priority UI components can be run that simply monitor the status of the system and provide occasional feedback.

The UI subsystem serves two purposes: mission design and mission execution. In the mission design stage, it supports building complex behaviors from existing behaviors. During mission execution, an operator can view mission status, alter system parameters, and control robots from teleoperation consoles.

Backbone

The backbone subsystem provides services that enable the other subsystems to work together. These services are capable of dynamically starting and stopping components, providing seamless communications across a variable number of networked machines, and trying to balance the system load over these machines. All intercomponent communication is handled transparently by common objects request broker architecture (CORBA) [6].

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 otherwise wreak havoc in the system). The next service to start up is the component placer. This component is responsible for starting all noncore service components and is the central service that all other components interface with for this purpose. A component creator service is started on each computer that is taking 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 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 RC manager is started.

Examples of ARCs and RCs

In order for a process to control a single Scout robot, several physical resources are required. First, a robot that is not currently in use by another process must be selected. Second, a command radio that has enough capacity to handle the demands of the process is needed. If the Scout robot is to transmit video, exclusive access to a fixed video frequency is mandatory. Without exclusive access, the interference between the transmitted video from multiple Scouts would render any signal processing hardware/algorithms useless. Finally, to process the video, a framegrabber and tuned video receiver are required. Each of these four resources is managed by its own RC.

Several kinds of ARCs are available, each requiring different resources to operate. These include ARCs that are capable of controlling only the actuators on the Scouts and do not require the use of the camera, ARCs that drive the Scouts and broadcast video data (to be viewed on a monitor by a human, for example), and ARCs that move the Scout, broadcast data, and capture it on a workstation for processing.

The RC Scheduler

Access to RCs must be scheduled when there are not enough RCs to satisfy the requirements of the ARCs. The RC manager maintains a master schedule of all active ARCs and grants each of them access to their RCs when it is their turn. 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). The RC manager uses a scheduling algorithm that tries to grant simultaneous access to as many ARCs as possible.

ARCs are divided into groups of equal priority. All ARCs of a higher priority must be either completed or running (not waiting on RCs) before an ARC of a lower priority can be added to the schedule. While this may not make full use of all available system resources, it does maintain the hierarchy of priority defined in the behavior manager.

When activated, the ARC contacts the RC manager and requests access to its set of RCs. The RC manager examines the ARC's request, checks that the RCs requested are running

(and starts them if they are not), and then attempts to add the ARC to the schedule.

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 that request an RC that is nonsharable cannot run at the same time and must break their operating time into slices. ARCs that have a sharable RC in common may be able to run simultaneously, assuming that the capacity requests for that sharable RC do not exceed the total possible capacity.

ARCs that have higher priorities are given precedence over ARCs with lower priorities. The RC manager attempts to generate a schedule of running ARCs that allow 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.

Once the ARC schedule has been constructed, the RC manager signals its schedule manager to implement it. The schedule manager takes care of signaling context switches as each ARC runs for its requested minimum time. When a context switch happens, 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.

Sharable Resources

Sharable RCs, such as the Scout radio, have to manage their own schedule to ensure that each of their ARCs is given a chance to send packets to their robot at the rate they request. When requesting access to a sharable RC, an ARC must specify a usage parameter that 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 be periodic and 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 [7] is used to schedule access.

Requests with higher frequencies are given a higher priority over requests with lower frequencies. This allows for optimal scheduling of the requests. However, higher priority ARCs have precedence over lower-priority ARCs, regardless of the frequency of the request. This can cause a disruption in the way requests are handled by the scheduling algorithm and may produce a schedule that is suboptimal in its usage of the RC. However, the relative priorities of ARCs takes precedence over optimal usage of the resources. Only when all of the higher-priority RCs have been scheduled will the lower priority RCs be allowed access. This enforces the optimality of the schedule. All other ARC requests must be scheduled by the RC manager.

For surveillance and reconnaissance applications, a robot that is small enough to avoid detection and can access hard-to-reach areas is desirable.

Visual Servoing

One of the challenges that must be addressed when writing autonomous behaviors for the Scout is how to handle signal corruption and noise in the image returned from the analog RF link. Noise creeps into the system from the Scout's motors, multipath reflections caused by the presence of obstacles around the robot, and weak signal strength caused by excess distance between transmitter and receiver. Finding ways to address this problem is extremely important, as any perceptual system that operates in the real world must be able to recognize and correct for corrupted sensor data if it expects to operate correctly.

In previous work [8], a simple frame-averaging algorithm was used to reduce the effects of noise. This approach only dealt with the problem of spurious horizontal lines and white noise. The algorithms that were used to navigate the Scout only used patches of light and dark areas as features.

Generally, the environments the Scouts are expected to operate in will be noisy, cluttered, and highly unstructured.

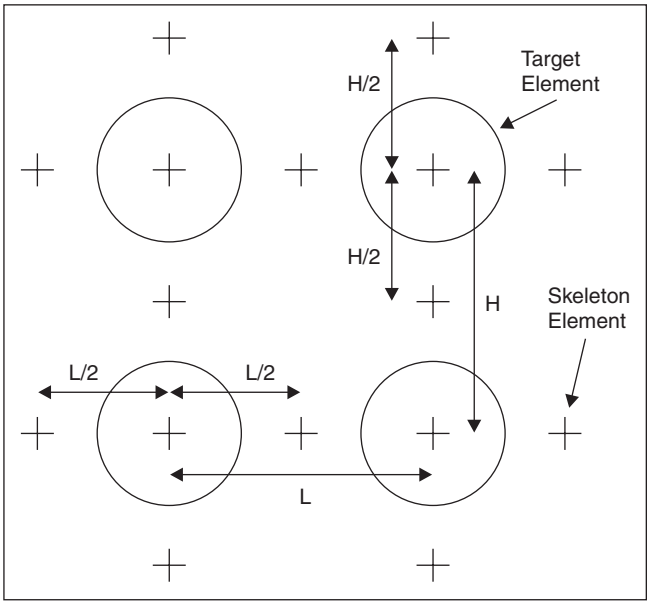

Figure 7. *Elements of the target and skeleton models.*

However, some aspects of the robot's immediate surroundings, such as a home base or deployment platform, could be marked with an easy-to-detect landmark.

The navigation procedure consists of a target acquisition phase and a tracking phase. In the target acquisition phase, the Scout rotates in place to find a target. Grayscale images digitized from the Scout's camera are converted to binary images by applying a fixed threshold to them. The image is searched for a pattern that is consistent with the known target geometry. If no such pattern is found, the Scout rotates a few degrees and tries again.

Once a likely target is found, the Scout enters the tracking phase. Two models are kept of the positions of the blobs in the image. The first model consists of a subimage containing the pixel values of the target. A second model consists of a subsampled set of pixels from the original image. This set consists of the pixel values at the centers of the blobs as well as the pixel values to the top, left, right, and bottom of each blob. This pixel set is referred to as the skeleton model of the target.

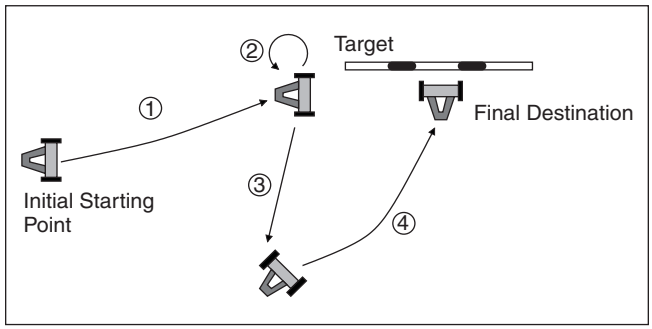

Figure 8. *Sample Scout trajectory executed with two approaches. After a forward movement (1) that doesn't reach the desired position, the Scout rotates left (2) before moving backwards (3) to give it a better approach (4) to the target.*

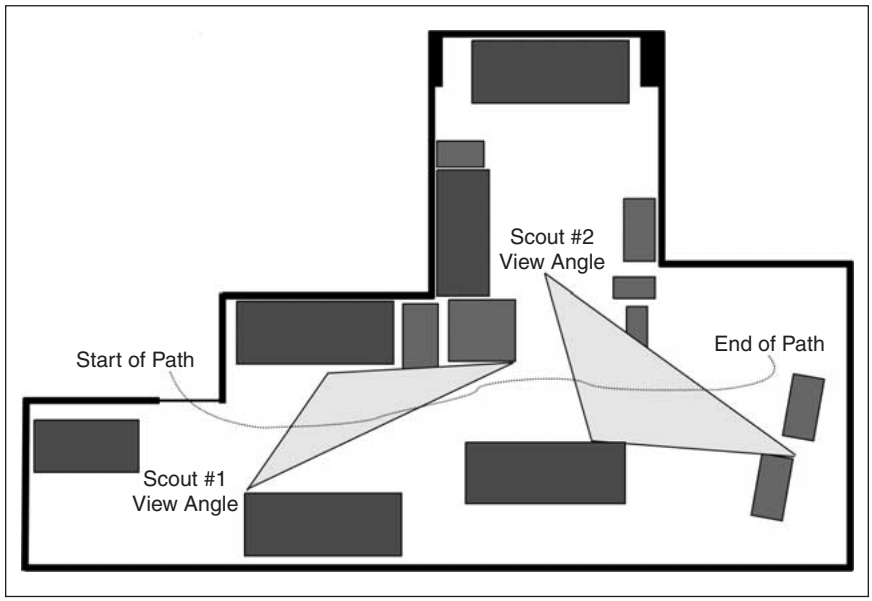

Figure 9 . *Top view of the room where the human motion detection experiment took place. The square objects are tables and other pieces of furniture. The field of views of both Scout robots are shown as wedges and the path the human took through the room is described as a line.*

As shown in Fig. 7, the primary target model is represented as a set of four circles that correspond to the position in the image that the target was last seen. The "+"s show the positions of the pixels of the skeleton. The skeleton for a four-blob target consists of only 17 pixels.

In the tracking phase, the Scout moves towards the target. Because the camera's field of view is so narrow, the Scout can easily lose sight of the target if it rotates too quickly. Thus, it is imperative that the Scout relocate the target blobs in each successive frame. To achieve real-time performance, new frames are acquired as close to a rate of 30 frames/s as possible. This tracking speed is made possible by only correlating the skeleton with the image. If a match is found, the target model is updated by extracting the target's new position from the image. If no match is found, the Scout must stop moving and use the last valid target model to locate the target (a more time-consuming operation). The Scout must return to the target acquisition phase if the best target model matches less than 70% of the target in the image.

A sample run of the algorithm is illustrated in Fig. 8. In this run, the Scout starts to the left of the target. Its goal is to reach a point on the zero angle line that will line the robot up with the target. Currently, experiments have been tried that start the robot upwards of 2 m away from the target. Barring battery or communications failures and given uniform lighting conditions, the robot is able to reach the target each time.

Distributed Surveillance Task

We are interested in a distributed surveillance task in which the Scouts are deployed into an area to watch for human motion. This is a useful task in situations where it is impractical to place fixed cameras because of the difficulties of placing them relating to power, portability, or even the safety of the operator (such as in hazardous situations).

With this scenario in mind, a set of experiments has been run to determine the effectiveness of the Scouts in a motion-detection task. In such a scenario, the Scouts would be remotely teleoperated to a position that provides the best overall view into an area. Once at this position, the Scouts are released by the teleoperation process and accessed by a behavior that analyzes the incoming frames of data and searches for moving objects.

Motion is detected by an image differencing algorithm that employs a two-step filtering process to reduce the chances of false positives. The first filter computes a weighted average of the incoming video data, which effectively smoothes out random white noise. Each new frame is subtracted from this averaged image, and a threshold is applied to

the pixels. This selects all of the pixels that have changed from one frame to the next. These pixels are grouped into blobs using a connected region extraction algorithm.

Some additional domain knowledge is used to classify whether a blob of pixels represents motion or just noise. The framegrabber that digitizes the incoming images has a very hard time dealing with a badly corrupted video signal. If the signal is bad enough, only a partial image will be returned. Generally, this kind of noise causes whole rows of pixels in the image to appear drastically different from the same pixels in clean images. Thus, any blob that exhibits characteristics like this will also be filtered out.

For the experiments, two Scout robots were placed into positions that allowed them to view a large section of a well-traveled path. As shown in Fig. 9, the likely path that a human would take through this area intersects with the lines of sight of both robots. Both robots share the same video frequency (due to limited hardware), so they had to be scheduled in a round-robin fashion. Because they could not operate in parallel, they were each able to make full use of the capacity of the radio. The minimum runtime for each robot was set to 5 s in order to allow enough time for a good number of clean frames to be captured. At the end of 5 s, a running behavior would be swapped out (the video on the Scout shut down), allowing another behavior to start.

In the experimental setup, a human enters the room from the left and makes his/her way to the right. It generally takes 7-8 s to walk this distance and then another 7-8 s to walk back. Of 22 trials that were run, the Scouts were able to correctly recognize the motion of the passing human approximately 81% of the time. In some of these trials, only one robot was able to make the proper identification. This occurred when the human walked by one Scout while its controlling behavior was swapped out of the currently running schedule. In 19% of the cases, the human was able to walk by the Scouts and not be detected. This occurred either because of high noise in the video signal or because each robot's controlling behavior process was swapped out at the time the human passed by.

These results illustrate the primary difficulty that arises when dealing with the coordination of multiple robots over a fixed capacity communications channel. To illustrate the effects of additional load on the performance of the system, an additional set of behaviors was loaded into the mission to run in parallel with the motion-detecting behaviors. These two behaviors were also given a 5 s minimum runtime to simulate the effects of more robots being run elsewhere. The overall detection rate of the system for the same task described above dropped to less than 57%.

Conclusion

We have presented a robotic system for reconnaissance and surveillance applications that is designed to operate in a semi-autonomous fashion. A human operator is able to remotely direct the robot to explore unknown areas as well as to allow the robot to do some of the tasks autonomously (such as returning to marked pickup area). A set of different kinds of UIs have been described that allow humans to control one or more Scouts, depending on the complexity of the mission. A software architecture has been presented that allows distributed communication to an arbitrary number of robots by teleoperation and autonomous control clients.

We plan to explore new communications technologies in order to increase the range and data rate of the RF communications link.

Future work that will include more advanced sensor interpretation and spatial reasoning techniques will expand on the Scout's autonomous capabilities. The software control architecture is also being expanded to allow more types of hardware resources, such as larger robots, to be controlled. Finally, we plan to explore new communications technologies in order to increase the range and data rate of the RF communications link.

Acknowledgments

This material is based upon work supported by the Defense Advanced Research Projects Agency, Microsystems Technology Office (Distributed Robotics), ARPA Order G155, Program Code 8H20, issued by DARPA/CMD under Contract MDA972-98-C-0008. This work has also been supported in part by the Microsoft Corporation and the Idaho National Engineering and Environmental Laboratory (INEEL).

Keywords

Multiple robots, mobile robots, distributed software architecture, resource allocation.

References

[1] L.E. Parker, "ALLIANCE: An architecture for fault tolerant multirobot cooperation," *IEEE Trans. Robot. Automat.*, vol. 14, pp. 220-240, Apr. 1998.

[2] 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.

[3] R. Alami, S. Fleury, M. Herrb, F. Ingrand, and F. Robert, "Multi-robot cooperation in the MARTHA project," *IEEE Robot. Automat. Mag.*, vol. 5, pp. 36-47, Mar. 1998.

[4] 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 *Proc. Government Microcircuit Applications Conf.*, Anaheim, CA, 2000, pp. 72-75.

[5] 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 *Proc. Sixth Int. Conf. Intelligent Autonomous Systems*, Venice, Italy, 2000, pp. 767-774.

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

[7] C.L. Liu and J.W. Layland, "On the complexity of fixed-priority scheduling of periodic, real-time tasks," *J. Assoc. Comput. Mach.*, vol. 20, no. 1, pp. 46-61, 1973.

[8] 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. Int. Conf. Autonomous Agents*, Barcelona, Spain, 2000, pp. 9-16.

Paul E. Rybski received an interdisciplinary B.A. in math/computer science with an emphasis in cognitive science in 1995 from Lawrence University in Appleton, Wisconsin, USA. He received his M.S. in computer and information sciences in 2000 at the University of Minnesota and is currently pursuing a Ph.D. in computer science with a minor in cognitive science at the same institution. His research interests include behavior-based control, distributed robotic teams, and robotic navigation/localization.

Sascha A. Stoeter obtained his M.S. in computer and information sciences in 1997 from the University of Minnesota. Before entering the Ph.D. program in Minnesota, he was a research assistant at the Institute for Robotics and Process Control in Braunschweig, Germany.

Nikolaos P. Papanikolopoulos received the Diploma degree in electrical and computer engineering from the National Technical University of Athens, Athens, Greece, in 1987 and the M.S.E.E. and the Ph.D. in electrical and computer engineering from Carnegie Mellon University in Pittsburgh, Pennsylvania, USA in 1987 and 1992, respectively. He is currently a professor in the Department of Computer Science and Engineering at the University of Minnesota and the director for the Center for Distributed Robotics. His research interests include robotics, computer vision, sensors for transportation applications, and control. He has authored or coauthored more than 140 journal and conference papers in the above areas (35 refereed journal papers). He was a McKnight Land-Grant Professor at the University of Minnesota from 1995-1997 and has received the NSF Research Initiation and Early Career Development Awards. He was also awarded the Faculty Creativity Award from the University of Minnesota. Finally, he has received grants from DARPA, Sandia National Laboratories, NSF, INEEL, Microsoft, USDOT, MN/DOT, Honeywell, and 3M.

Ian Burt has a B.S. degree in mechanical engineering and is currently working towards his M.S. in the Department of Mechanical Engineering through the University of Minnesota. He was worked in the Center for Distributed Robotics for the past two years. He has also competed in eight robotic combat tournaments across the United States (including Battlebots) over the last two years.

Tom Dahlin is an electronics design consultant specializing in microprocessor-based sensor and motor control applications. He graduated from Northern Michigan University in 1979 with a B.S. in math/computer science. He now works for the 3M company in St. Paul, Minnesota, USA, as a senior engineering specialist in the corporate research and development group. Prior employment includes design engineering positions with Honeywell and Stratasys.

Maria Gini is a professor at the Department of Computer Science and Engineering of the University of Minnesota. She has received the Continuing Education and Extension Distinguished Teaching Award (1995), the Morse-Alumni Distinguished Teaching Professor of Computer Science (1987), the Outstanding Professor Award (1986 and 1993), the Fullbright-Hays Fellowship (1979), and the NATO Fellowship (1976). She was the editorial program cochair of the International Conference on Autonomous Agents (Agents' 2000) in Barcelona, Spain, May 2000. She was also a member of the Advisory Board of IJCAI-99. She is on the editorial board of *Autonomous Robots* and *Integrated Computer-Aided Engineering*. Finally, she is member of the Executive Council of the AAAI Special Interest Group on Manufacturing.

Dean F. Hougen received his B.S. in computer science from Iowa State University in 1988 with minors in mathematics and philosophy. He received his Ph.D. from the University of Minnesota in 1998, also in computer science, with a graduate minor in cognitive science. After serving as an assistant professor in the Department of Computer Science and Engineering and associate director of the Center for Distributed Robotics, both at the University of Minnesota, Dr. Hougen moved to the School of Computer Science at the University of Oklahoma, where he has founded the Robotic Intelligence and Machine Learning Laboratory. His research includes distributed heterogeneous robotic systems, learning (reinforcement, connectionist, and memetic) in real robots, and evolutionary computation.

Donald G. Krantz received his Ph.D. in computer and information sciences from the University of Minnesota. He is currently vice president of the Advanced Systems Division of MTS Systems Corporation. Dr. Krantz was previously a principal investigator and program manager at MTS. MTS designs and builds complex computer-controlled systems to simulate physical phenomena for testing and research (www.mts.com). Prior to joining MTS, Dr. Krantz was an Alliant Techsystems Fellow and a Honeywell Fellow.

Florent Nageotte graduated from the Ecole Nationale Superieure de Physique (engineering school) and from the university of Strasbourg, France, in 2000. He is a Ph.D. student in LSIIT (UMR CNRS/ULP 7005), Strasbourg. His research interests include visual servoing and surgical robotics, especially the use of visual servoing to improve robotized laparoscopic surgery.

Address for correspondence: Nikolaos Papanikolopoulos, Center for Distributed Robotics, Department of Computer Science and Engineering, University of Minnestoa, Minneapolis, MN 55455 USA. E-mail: npapas@cs.umn.edu. URL: http://distrob.cs.umn.edu.