Cooperative Search and Rescue with a Team of Mobile Robots

James S. Jennings
EECS Department
Tulane University
New Orleans, LA 70118
jennings@eecs.tulane.edu

Greg Whelan
Robotics Institute
Carnegie Mellon University
Pittsburgh, PA 15213
whelan@cs.cmu.edu

William F. Evans
EECS Department
Tulane University
New Orleans, LA 70118
evans@eecs.tulane.edu

Abstract

We present an implemented algorithm for a distributed team of autonomous mobile robots to search for an object. When one robot finds it, they all gather around it, and then manipulate ("rescue") it. The algorithm exploits parallelism, with all robots searching concurrently, and also teamwork, because the manipulation is performed cooperatively. Our algorithm is fully distributed; the robots communicate with each other, and there is no central server or supervisor. Applications include hazardous waste cleanup, bomb detection and removal, materials delivery, and eventually the rescue of survivors of accidents or disasters.

The search and rescue program was written using MOVER, *a programming system for distributed tasks.* MOVER *is quite general, allowing arbitrary synchronization of tasks among workstations and robots. The system provides high-level programming constructs for task distribution across robots. Finally,* MOVER *encourages code re-use because the task distribution mechanism can synchronize any set of procedures (without rewriting), allowing the programmer of a distributed task to access libraries of robot software written for single-robot tasks.*

Keywords: Multi-robot systems, Mobile robots, Software systems, Algorithms for Cooperation.

1 Introduction

Much recent work in mobile robotics focuses on achieving cooperation among a team of mobile robots. A cooperative solution to a task is usually implemented as a set of programs, each of which runs on a member of the robot team. The programs are intended to cause the team to collectively solve the (global) task. For some tasks, communication between robots appears necessary; "search and rescue" is such a task. In a "search and rescue" task, many robots search in parallel for a single object. When one robot discovers the object, it signals the others, at which time as many robots as are necessary to manipulate the object meet at the object's location. The robots then cooperatively manipulate the object to a specified goal location and orientation. (They "rescue" it.)

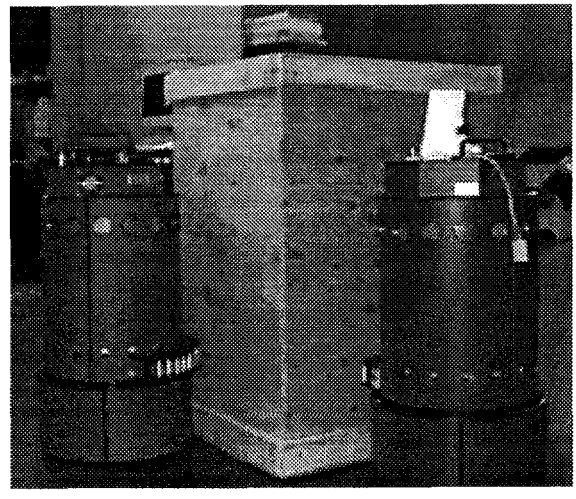

Figure 1: Two of our mobile robots, Ernst and Moseley, near a large box which might be the object for which they are searching, and which they would then retrieve.

We have automated such a task on a team of Tulane Mobile Robots, which can cooperatively search out and manipulate objects too large for a single robot to handle alone. These are typically pieces of furniture, or large boxes such as the one shown in Figure 1. Our search and rescue program has the following features:

- The program is fully distributed across the robot

0-7803-4160-0-7/97 $10.00 © 1997 IEEE

team – there is no central server or coordinator required to execute the task. Looking ahead to the development of fault-tolerant distributed robot programs, we seek to avoid dependence on a central server or controller.

- The program which enables the cooperation by coordinating the various actions of the robots in the team is quite short, about 17 lines (see Figure 3).

- The implementation is modular in that the synchronization of the subtasks does not depend on specific properties of the algorithms chosen for search, object recognition, navigation, or manipulation. Consequently, new algorithms may be substituted without rewriting the other algorithms, or the search and rescue program itself.

The coordination of several robots in a search and rescue task is more complex than in tasks such as foraging, which is parallel but individual. That is, although the robots work simulataneously in foraging, they do not purposefully interact with each other. By contrast, a coordinated manipulation algorithm requires a high degree of interaction among the members of a robot team. Perhaps more significantly, there is a critical transition in search and rescue tasks between *search*, which is a foraging-like task, and *rescue*, a closely cooperative manipulation task. Our search and rescue implementation makes use of a simple task control mechanism to achieve this transition.

MOVER's task-control mechanism is a pair of procedures called `distributed-or` and `distributed-and` which do *not* rely on a central server or controller. A task written using these procedures is fully distributed across all of the robots in the task, which then communicate only with each other. Also, MOVER is not restricted to distributing tasks using only these mechanisms. New forms of synchronization and communication may be added by users of the system.

In this paper we describe the MOVER system in its current stage of development (Section 3), and we note three examples of its use. The primary example is a search and rescue task (Section 4). In another application MOVER is used to train a mobile robot's neural network using a previously-trained robot (Section 5). In that experiment, the training robot learns to recognize a physical location by following another robot to that location. Finally, we note that MOVER is used in the Tulane EECS Robotics Laboratory for single-robot experiments as well, such as active model-based recognition of objects [JE96, JR93].

2 Related Work

Our work is inspired in particular by two powerful architectures for task control on a single robot, TCA [Sim94] and ESL [Gat96].

2.1 TCA and ESL

The *Task Control Architecture (TCA)* is designed for building task-level control systems for mobile robots. This general-purpose architecture allows building distributed, concurrent robot programs by providing tools for communication, task decomposition, task sequencing, resource management, execution monitoring, and exception handling. Like TCA, MOVER is built using remote procedure calls, but MOVER does not need a central server for task control.

The *Executive Support Language (ESL)* [Gat96] is another task control system which inspired us. ESL is a language designed to support the construction of reactive control mechanisms for autonomous robots and spacecraft. Some of ESL's built-in mechanisms are similar to the constructs we have created for managing tasks across multiple robots in MOVER. For example, note the similarity between ESL's `or-parallel` (a construct which executes a set of tasks in parallel until one has completed), and the Mover system's `distributed-or` (which performs the same function in a fully-distributed system). See Section 4.2, below, for a description of task control in MOVER. As with TCA, ESL was designed for a single-robot system in which the presence of a central server or controller for sub-task scheduling is both reasonable and effective.

2.2 Kali Scheme

MOVER is based on Kali-Scheme [CJK95], a distributed implementation of the Scheme language [CR92] which runs under most Unix operating systems on networked computers. Because Kali-Scheme is not platform-dependent, our distributed programs can include processes which execute on desktop workstations, laptops, or high-speed compute servers, as well as on robots. (The Tulane Mobile Robots are based on the RWI B14 architecture, with on-board networked Pentium PC's running the Linux operating system.)

Kali-Scheme was designed to "engender a number of new abstractions and paradigms for distributed computing" [CJK95]. We expect that the field of distributed robotics will require new abstractions, and so we have chosen Kali-Scheme particularly because of its support for user-level control of distributed processes. Its proxies and remote procedure calls are used

to implement MOVER's distributed task control mechanisms.

2.3 Architectures for Cooperation

Because our research goal is to enable cooperation specifically for manipulation tasks, we do not review here systems which allow mobile robots only to arrange themselves in either static or moving formations. Instead, we focus on architectures for cooperative manipulation.

Without Communication Between Robots

In previous work [BJ95, RDJ95, DJR94, DJR93], we demonstrated algorithms with which teams of mobile robots could cooperatively reposition and reorient large objects without communication. A key feature of this work is that the robots in the task need little if any *a priori* information about the geometric shape or size of the object they are manipulating.

The goal of [KZ96] is similar: to control multiple robots without using a centralized server, and without communication, for a manipulation task. A set of 10 robots (running identical algorithms) simultaneously try to find a box and push it to a goal. The method requires the programmer to design sensing routines which provide cues for action selection.

Another architecure for multi-robot systems is Alliance [Par94], in which the aim is to provide fault-tolerant accomplishment of a global goal by a set of robots. Although most of the demonstrations of Alliance were performed in simulation, two actual (physical) robots using the architecture were able to push a box despite the failure of one of the robots. The system does not appear to have the ability to synchronize actions of the robots. Alliance achieves fault-tolerance by relying on the ability of each robot to accurately detect the effects of the actions of all of the others through sensing.

With Communication Between Robots

Discussing issues of communication in multi-robot tasks is [ABN93], in which task solutions with and without communication are compared. In order to propose a concrete notion of the *type of communication* used in a multi-robot algorithm, three categories are suggested: no communication, state communication, and arbitrary communication. The second is an interesting category, in which robots broadcast a representation of their internal state, and all are able to receive such broadcasts from others.

Instead of asking what type and how much communication is needed to solve a task, we take a different tack and ask first how we will distribute the task across the robots: *Will all robots search in parallel? Must the manipulation of the object be delayed until all of the robots have arrived at the object? Etc.* Then we produce a working program demonstrating a solution (see Figure 3). Finally, with a working solution in hand, we can examine its performance (see Section 4.4 and robustness, and hopefully derive better algorithms as a result. By starting with the high-level abstraction of remote procedure calls, we avoid dealing directly with low-level communication issues.

Developing a model of cooperation called "task-sharing," [MNS95] demonstrates box-pushing with two robots which manipulate a box to a specified goal location. Although the box is small enough for one robot to accomplish the task, two robots perform the task more efficiently. The goal is marked so as to be directly sensed by the robots. The robots send explicit messages to each other; some of these messages are tokens that pass control between the robots so that only one of them is moving at a time. Messages are processed as sensory data, with programs written in a behavioral style. The technique requires each robot to process sensory data sent to it by the other.

The Actress architecture for multi-robot cooperation (see e.g. [AOI+91]) also uses message-based communication between the robots in the task, but a rule system governs the actions taken as the result of receiving a message, instead of the behaviors of [MNS95]. The task is fully distributed with no central controller or server, but programs must be written to take explicit action upon receipt of various types of messages from other robots.

3 The MOVER System

The goals of our research program may be summarized as follows:

1. to enable robots to perform more tasks autonomously;
2. to enable robots to work cooperatively, in teams;
3. to allow heterogeneous teams to cooperate; and
4. to encourage re-use of robot programs for more rapid development of robust solutions to new tasks.

Our previous experiments in minimalism [BBD+95] shared some of these broad goals, but focused on deriving minimal resource configurations for distributed manipulation tasks. Here we take a wider view. We propose an experimental agenda in which *we derive a*

distributed solution to a task using whatever resources are available, such as reliable communication links between robots. Efficiency and fault-tolerance, while extremely important, momentarily become secondary concerns. As builders of robot programs, we consider it essential to begin with a working solution for a problem, and then transform it, e.g. to increase efficiency. Transformations can be applied to a working program which preserve its correctness.[1]

Features of MOVER include:

- There is no explicit passing of messages, tokens, or internal state. Instead, robots make remote procedure calls which cause the other robots to (typically) alter what they do.
- The task distribution mechanism is *secure*, meaning that robots and computers *which are not in the team* cannot interfere with the task by calling procedures on the robots in the team.
- The task distribution mechanism is also *safe*, meaning that the remote procedure calls cannot effect other than the desired activity. (This implies, for example, that one robot cannot call a procedure which makes another robot drive off of a cliff.)[2]
- MOVER is built out of a distributed implementation of the Scheme language, giving two noteworthy benefits:
  1. We did not build a new programming language in order to program distributed robot tasks.
  2. We did not introduce ad hoc primitives to an existing language, a process which inevitably complicates the semantics of the language and makes programs more difficult to write and debug.
- MOVER encourages the re-use of robot control programs. The procedures incorporated into MOVER are drawn from a library of robust procedures for navigation, active acquisition of object models [JE96, JR93], model-based recognition [ACH+91], and cooperative manipulation [BJ95, DJR93, DJR94]. Most of these procedures are written in Scheme. Some are written in C. Multi-robot experiments performed using MOVER at Carnegie Mellon University used many TCA library procedures written in C (see Section 5).
- New task-control mechanisms may be added to MOVER by users of the system, making it a useful testbed for exploring multi-agent (robot and computer) architectures.
- Support for heterogeneous teams of robots is built-in. The Tulane Mobile Robots (Ernst, Moseley, and Elvis) are almost identical, but we routinely use workstations in our distributed robot applications in order to offload heavy computation, interact with users, and display graphical output.
- Load-balancing and thread (process) migration are supported by Kali-Scheme, and so they are part of MOVER. These facilities provide the user with a great deal of control over tasks, and could be used to increase performance, or potentially to recover from catastrophic failures.[3]
- MOVER requires reliable point-to-point communication channels between robots. We propose that with the current level of affordable technology, maintaining such a connection by radio (e.g. radio-modem or wireless ethernet) is feasible. Reliability comes from standard layered network protocols, which do not assume a reliable low-level channel. Our programs currently stall if they need communications and network access is unavailable, and they automatically resume when access is restored.

4 The Search and Rescue Task

4.1 Limitations of the Current Implementation

We have developed and implemented a program which performs the distribtuted search and rescue task as described in Section 1 in order to demonstrate the MOVER system.

A robot team which can solve a search and rescue task requires the following capabilities:

- navigation and localization;
- search;
- object recognition;
- communication with other members of the team; and
- cooperative manipulation of large objects.

[1] We thank Dan Friedman (Computer Science Dept., Indiana University) for exposing us to the raw power of correctness-preserving transformations.

[2] Safety and security derive mainly from Scheme's lexical scoping, and that Scheme procedures and continuations are first-class objects. Kali-Scheme supports the communication of these higher-order objects which are used by MOVER.

[3] We do *not* address fault tolerance in this paper; we plan to address this issue in a subsequent publication.

In order to illustrate the MOVER system, we employed very simple search, recognition, navigation, and manipulation algorithms.[4] They are:

Search: Our robots search by wandering in randomly-chosen directions until they hit an object. With a geometric map of the environment, a robust deterministic method could be used, which would allow a negative conclusion to the search if the target object was not in the room or on this floor of the building.

Recognition: Currently, the object to rescue is recognized by the robot's ability to move it with a fixed amount of motor torque. Thus, immovable objects are ignored, and the first movable object is rescued. Vision-based object recognition could be added, or reliable shape-based recognition which we have already implemented [JE96].

Navigation: Robots in the team currently use dead-reckoning to keep track of their global position, and an online (without a map) navigation method to arrive at an object found by another robot. Path planning and a true localization system such as that of [Bro95] are planned.

Manipulation: Without a geometric map of our building, planning the manipulation of the object to the goal pose is not feasible. Thus, our robots currently move the object relative to its current position, instead of to a global goal position. The method used is the Pusher/Steerer of [BJ95, Bro95], which takes as input a path to be followed. The relative motion we have implemented serves to demonstrate the task transitions from search to manipulation, to completion. To be added is a manipulation planner which would generate a plan with which the robots could move the object to the global goal location.

Finally, we note that at the time of this writing, the robots trail an ethernet cable behind them. Soon they will have wireless network communication. MOVER has been used in a wireless system, however – see Section 5 below.

4.2 Performing the Task

Our current experimental system consists of two autonomous robots, Ernst and Moseley, each with onboard computers, sonar and tactile (bumper) sensors. Communication is via an ethernet network and can occur between robots and workstations. Typically a task is started by entering a program on a workstation, which automatically distributes the task to the desired set of robots. (See Figure 2.)

[4] Soon we expect to report experimental results in which more powerful routines from our library are employed instead of the simple routines shown here.

The program that allows the robots to synchronize their actions during the search and rescue task is shown in Figure 3. This is the complete program that a user would enter at a workstation in order for the robots to autonomously locate the object and retrieve (rescue) it. The program is written in the Scheme language and uses the `distributed-or` and `distributed-and` features of MOVER. An explanation of these features is included in the following discussion of how the search and rescue program of Figure 3 is interpreted:

1. The `distributed-or` procedure is given three arguments: the *operation* to be performed on all robots, the *continuation* of the task when the first robot finishes its operation, and a list of robots in the team. In search and rescue, the *OR operation* is a call to the search procedure, which is given as an argument a procedure for recognizing the desired object. See Figure 2*(1,2,3)*.

2. The search phase of the task stops when one robot has found the object. Figure 2*(4)*.

3. When the search phase is complete, the `distributed-or` procedure calls its second argument, the *OR continuation*, which is a procedure that specifies what each robot is to do after the search is complete. The *OR continuation* is a procedure of one argument, `results`, which contains the results of the search from the robot which found the object. See Figure 2*(5)*.

4. The results of the search phase include two pieces of data: `uid`, the unique identifier of the robot which found the object, and `object-location`, indicating where the object was found.

5. All robots except the one which found the object need to arrive at the object for the manipulation to proceed, and they execute `goto-object` to do so. This is the *AND operation*. See Figure 2*(6)*.

6. Only after *all* robots have arrived at the object, the `distribtuted-and` procedure calls the *AND continuation*, which specifies that the robot which found the object should execute `(pusher)` and the other robots (in our example, just one other) should execute `(steerer)`. See Figure 2*(7,8)*.

7. The *AND continuation* is the last operation to be performed in this task, and so the task concludes when the pushing and steering robots complete the manipulation. See Figure 2*(9)*.

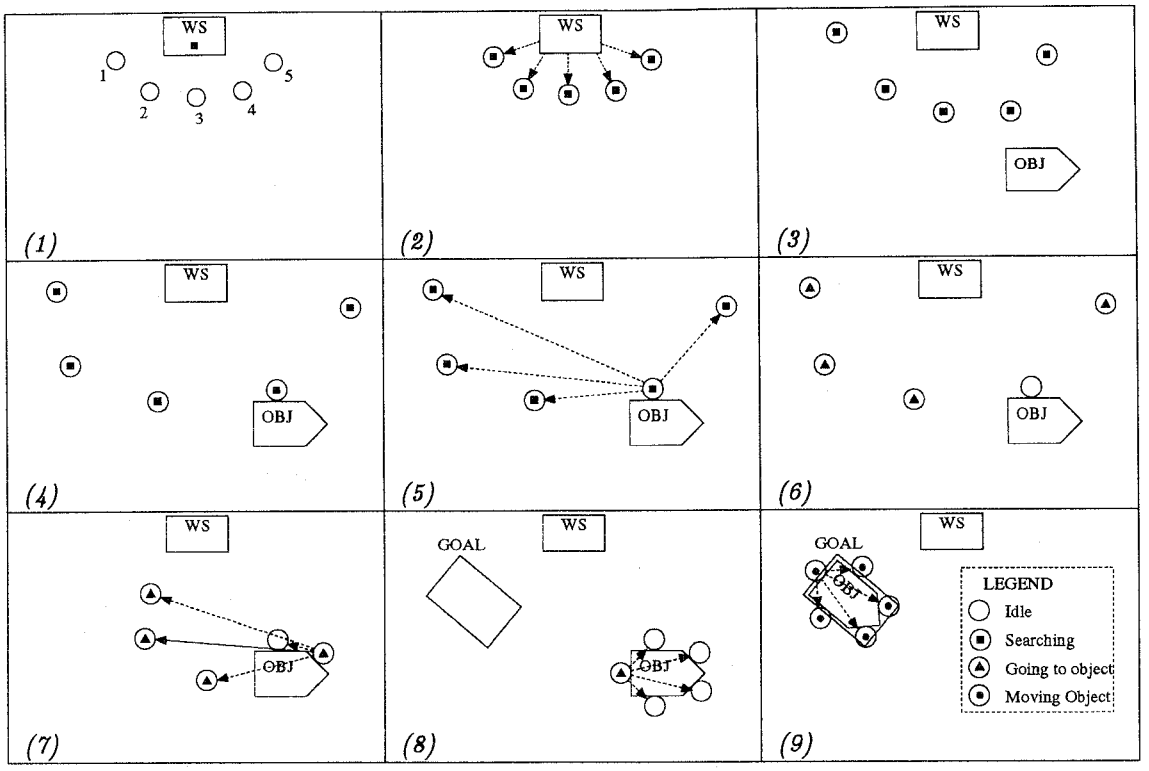

Figure 2: The search and rescue task in full generality. *(1) A workstation (WS) and a team of 5 idle robots (circles). (2) The user at the work-station enters a program for performing search and rescue (or loads it). The program is distributed to all of the robots in the task. (3) All robots have started searching for the house-shaped object (OBJ). (4) One of the robots has found the house-shaped object. The others are still searching and are as yet unaware that the object has been found. (5) The robot that found the object notifies the other robots. (6) The other robots head towards the object. (7) One of the robots arrives at the object, and informs all the other robots that it has arrived at the object. (8) The last robot arrives at the object and notifes the others, as all of the other robots have done. Cooperative manipulation begins now. (9) The robots have manipulated the object to the goal, and the search and rescue task is complete.*

4.3 Procedures Used

The search and rescue procedure is built using the following procedures from our code library:

(`search` *predicate terminator*) Search with a random walk until an object is found which passes the *predicate* test. In our experiments, this predicate checks to see if an object is movable by the robot using a medium amount of motor torque. The *terminator* parameter is a procedure which is called periodically to determine if the search should be stopped for any reason.

(`goto-object` *location*) Go to a point near *location*, and then perform a cautious approach to arrive at *location*, which includes a desired final orientation (heading). Currently, this procedure attempts to follow a straight-line trajectory.

(`pusher`) Push the object along a trajectory specified by the `steerer` robot. Because we are using only two robots at this time, the manipulation algorithm we have chosen is the Pusher/Steerer method of [BJ95], in which one robot pushes the object and the other steers, keeping the object pinned between them.

(`steerer`) Steer the object along the desired trajectory to the goal. In our initial tests, this trajectory is fixed, and specified relative to the object's location. This procedure will be replaced by one which plans for the manipulation of the object to a specified goal location.

4.4 Analysis of Communication

An analysis of the communication requirements of a task with n robots and k steps (points at which the robots must be synchronized either by `distributed-and` or `distributed-or`) reveals a worst case of $O(kn^2)$ separate communications between robots in the task, a bound which is achievable if all k

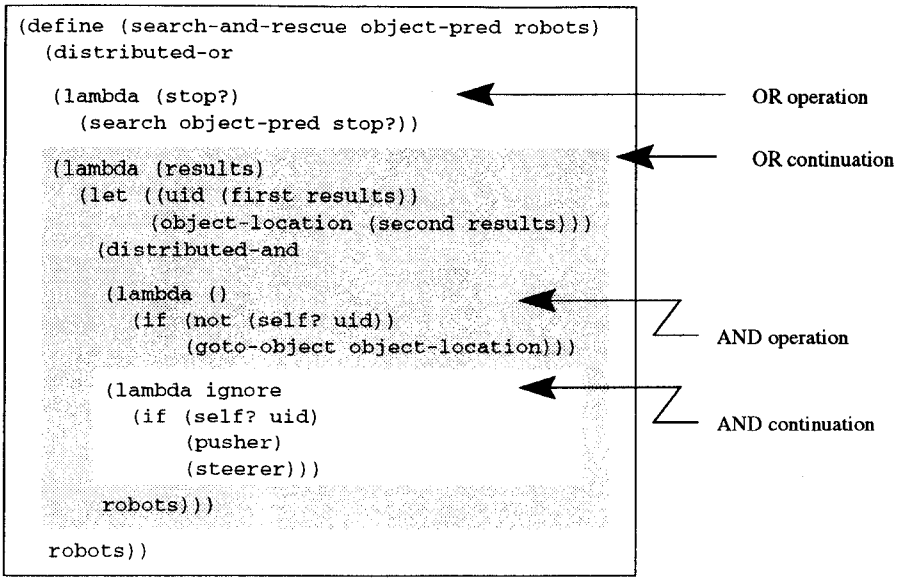

Figure 3: The search and rescue program. In Scheme, the keyword `lambda` indicates a procedure. The `distributed-or` and `distributed-and` procedures take 3 arguments: an operation, a continuation, and the list of robots in the task. The *operation* is a procedure which is executed on every robot. The *continuation* is a procedure which specifies how the task continues after *operation* is complete. In `distributed-or` an *operation* is complete as soon as one robot has finished it. In `distributed-and`, an operation is complete only when all robots have finished it. In this figure, the *OR continuation* is highlighted by shading, and the *AND continuation* is in white. Note that the *OR continuation* encodes how the task continues after the search, and thus includes the entire `distributed-and`. See Section 4.2.

steps use `distributed-and`, which causes each robot to communicate with all of the others. For many tasks, however, the behavior of `distributed-and` could be modified such that all robots inform exactly one designated robot[5] when they complete their operations. Also, note that a single `distributed-or` requires just $O(n)$ communications, one to each of the other robots by the robot which finished its operation first. Thus, the expected number of communications in a given task will be considerably less than in the worst case.

5 Neural Net Training

A prototype of MOVER is in use for the high-level control of a multi-robot task in which one mobile robot with the capability of identifying doors (entrances to rooms) automatically trains another robot to identify the same doors. During training, the trainee robot follows the trainer robot to each door that the trainer can recognize. At each door, the trainee records sensory data to be used in subsequent training of a neural network. Utilizing this data to train the trainee's neural-network is currently in progress.

These experiments are being carried out at the Carnegie Mellon Learning Robot Laboratory using robots which are programmed individually using TCA [Sim94]. MOVER was extended using Kali-Scheme's foreign function interface to call TCA procedures for sensing, motion, planning, etc. MOVER synchronized the steps in the training process, with communication between the robots taking place over a wireless ethernet.

Planned is a system written in MOVER for automating a more general transfer of other knowledge between multiple robots by a similar training process.

6 Conclusion

We have presented a system, MOVER, which supports fully distributed, safe, and secure cooperative robot programming, including high-level task control in the form of `distributed-or` and `distributed-and`. The first demonstration of these distributed task control mechanisms was the search and rescue program shown in Figure 3. Although the library procedures (such as `search`) called by this program are limited individually, each can be easily replaced by a more powerful

[5]The designated robot could be chosen as the task is executed, and could vary from step to step.

procedure without altering the others, or the search and rescue program itself. Indeed, the search and rescue program is passed an argument which is a procedure for recognizing an object. A new object can be rescued by calling search and rescue with a different recognition procedure. Thus we can exploit the re-use of code and stimulate the rapid development of new solutions to cooperative robot tasks.

7 Acknowledgements

We are grateful to the developers of Kali-Scheme, and especially to Richard Kelsey for much assistance. We thank Eric Beuscher and Joshua Allen for hacking robots and graphics for us.

This paper describes research done in the Robotics Laboratory of the Department of Electrical Engineering and Computer Science at Tulane University. Support for our robotics research was provided in part by the Louisiana Education Quality Support Fund under Contract Number LEQSF-RD-A-27 (1996).

References

[ABN93] R.C. Arkin, T. Balch, and E. Nitz. Communication of behavioral state in multi-agent retrieval tasks. In *Proc. of the 1993 IEEE International Conference on Robotics and Automation*, volume 2, pages 588–594, Atlanta, Ga, 1993.

[ACH+91] E. Arkin, L. P. Chew, D. P. Huttenlocher, K. Kedem, and J. S. B. Mitchell. An efficiently computable metric for comparing polygon shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **13**(3):209–216, 1991.

[AOI+91] H. Asama, K. Ozaki, H. Itakura, A. Matsumoto, Y. Ishida, and I. Endo. Collision avoidance among multiple mobile robots based on rules and communication. In *Proc. of IEEE Conf. on Intelligent Robot Systems*, pages 1215–1220, Osaka, Japan, 1991.

[BBD+95] K. Bohringer, R. Brown, B. Donald, J. Jennings, and D. Rus. Distributed robotic manipulation: Experiments in minimalism. In *Proceedings of the International Symposium on Experimental Robotics*, 1995.

[BJ95] R. Brown and J. Jennings. Manipulation by a pusher/steerer. In *Proc. of IEEE Conf. on Intelligent Robot Systems*, Pittsburgh, PA, August 1995.

[Bro95] R. G. Brown. *Algorithms for Mobile Robot Localization and Building Flexible, Robust, Easy to Use Mobile Robots.* PhD thesis, Cornell University, Ithaca, NY, 1995.

[CJK95] H. Cejtin, S. Jagannathan, and R. Kelsey. Higher-order distributed objects. *ACM Transactions on Programming Languages and Systems*, September 1995.

[CR92] W. Clinger and J. Rees. Revised[4] report on the algorithmic language scheme. Technical report, Cornell University Department of Computer Science, 1992.

[DJR93] B. R. Donald, James S. Jennings, and D. Rus. Experimental information invariants for cooperating autonomous mobile robots. In *Proceedings of the 1993 International Joint Conference on Artificial Intelligence, Workshop on Dynamically Interacting Robots*, Chambery, Fr., 1993.

[DJR94] B. R. Donald, James S. Jennings, and D. Rus. Analyzing teams of cooperating mobile robots. In *Proceedings of the 1994 IEEE International Conference on Robotics and Automation*, pages 1896–1903, San Diego, CA, 1994.

[Gat96] E. Gat. Esl: A language for supporting robust plan execution in embedded autonomous agents. *AAAI Fall Symposium*, 1996.

[JE96] J. Jennings and W. Evans. Active object recognition by mobile robots using a metric on geometric shape. In *unpublished manuscript*, Tulane University Computer Science Department, 1996.

[JR93] J. Jennings and D. Rus. Active model acquisition for near-sensorless manipulation with mobile robots. In *IASTED International Conference on Robotics and Manufacturing*, pages 179–184, Oxford, England, September 1993.

[KZ96] C.R. Kube and H. Zhang. The use of perceptual cues in multi-robot box-pushing. In *Proc. IEEE Int. Conf. on Robotics and Automation*, pages 2085–2090, Minneapolis, MN, 1996.

[MNS95] M.J. Mataric, M. Nilsson, and K.T. Simsarian. Cooperative multi-robot box-pushing. In *Proc. of IEEE Conf. on Intelligent Robot Systems*, Pittsburgh, PA, 1995.

[Par94] L.E. Parker. *Heterogeneous Multi-Robot Cooperation.* PhD thesis, Massachusetts Institute of Technology, 1994.

[RDJ95] D. Rus, B. Donald, and J. Jennings. Moving furniture with mobile robots. In *Proceedings of Intelligent Robot Systems*, Pittsburgh, PA, August 1995.

[Sim94] R. Simmons. Structured control for autonomous robots. *IEEE Transactions on Robotics and Automation, 10(1)*, Feb 1994.