Dartmouth College Undergraduate Theses   Theses and Dissertations

6-5-1998

Multiscouting: Guiding distributed manipulation with multiple mobile sensors

Michael G. Ross
Dartmouth College

Recommended Citation

Multiscouting:
Guiding distributed manipulation with multiple mobile sensors

Michael G. Ross
Senior Honors Thesis
Advisor: Assistant Professor Daniela Rus
Dartmouth College

Abstract

This thesis investigates the use of multiple mobile sensors to guide the motion of a distributed manipulation system. In our system, multiple robots cooperatively place a large object at a goal in a dynamic, unstructured, unmapped environment. We take the system developed in [Rus, Kabir, Kotay, Soutter 1996], which employs a single mobile sensor for navigational tasks, and extend it to allow the use of multiple mobile sensors. This allows the system to perform successful manipulations in a larger class of spaces than was possible in the single scout model. We focus on the development of a negotiation protocol that enables multiple scouts to cooperatively plan system motion. This algorithm enhances the previous' system's scalability and adds greater fault-tolerance. Two alternate algorithms for cooperation: a modification of negotiation and a bidding protocol, are also discussed. Finally, an implementation of the negotiation protocol is described and experimental data produced by the implementation is analyzed.

1: Introduction

1.1: Cooperation

Cooperation is one of the most fascinating features found in natural systems. Despite fiercely competitive natural selection, humans have evolved complex cooperative structures over millions of years that enable us to survive in the world far more successfully than we would individually. We are hardly unique in this respect – wolves hunt together in packs, lions live in prides, and even some insects, most notably bees and ants, are organized into large colonies in which each member has a particular role to play. Considering the relative dearth of cooperative technology, one could say that this is one of the natural behaviors that computer scientists and engineers have been least successful in emulating.

There are many potential applications for cooperative computing, stretching across fields as diverse as robotics, bio-technology, and astronomy. Many problems that lend themselves to matrix-based representations can be spread across multiple CPUs and solved far more quickly in parallel than they could on a single processor. In other applications, no single system has the resources required to complete a task alone and simple parallelism is insufficient. On an automobile assembly line, any particular robot is incapable of building a car by itself because the job requires a range of specialized equipment – the machines must cooperate and combine their skills in complex ways to produce the desired results. In other cases, such as the pursuit-evasion problem discussed in [Guibas, Latombe, LaValle, Lin, Motwani 1997], multiple, cooperating robots are

necessary to provide widely distributed, simultaneous sensing capabilities that cannot be achieved by a single machine. Multiple robots can also be used to find and/or manipulate an object more efficiently, as in the cooperative search and rescue work in [Jennings, Whelan, Evans 1997].

Distributed, coordinated robotics presents a range of technical and algorithmic challenges. First, most distributed robotics applications require system elements to communicate in order to coordinate their activities. Explicit message-passing over networks or via radio modems is often unreliable and relatively slow compared to the rate at which local sensor processing occurs. Missed messages can lead to failures in cooperation protocols and the extra time required for network operations can negatively impact system responsiveness.

Responsiveness is also crucial, especially to systems in a dynamic environment that may have to react to changes quickly to avoid failure or destruction. Heavy communication requirements can reduce responsiveness, as mentioned above, but they are not the only factor. A complex task whose control algorithms require extensive sensor processing and interpretation prior to each decision may be especially susceptible to failure if its environment changes more quickly than it can adapt.

Just like other distributed applications, distributed robotic systems should be scalable and fault-tolerant. If a large problem needs to be solved, algorithms should be designed so that the number of cooperating modules can be increased easily and so that increasing the number of system elements makes larger problems more tractable. Conversely, fault-tolerance requires that if system elements become inoperable the

remaining modules continue their common task as well as possible. This ability to recover requires that every piece be engineered with as few assumptions about the behavior of the rest of the system as possible. It is extremely difficult for a robot system to handle the complexities of a non-discrete environment, its own sensor and mechanical errors, and the potential errors in other parts of the system reliably.

1.2: Cooperative Manipulation

In distributed manipulation systems we are interested in solving the placement problem, in which an object is moved from its initial location to a specified goal. This problem is related to the task of navigation, but is different because it requires that all navigational plans executed by the powered system elements take their effects on the external, manipulated object into account. Since the interactions between manipulators and objects are physical and involve frictional forces and other sources of error, placing objects correctly is a much more difficult problem than simply finding a path through an environment.

The cooperative, distributed placement of objects has a large range of potential applications. In a warehousing application, for instance, teams of small robots could handle small object placement individually or in pairs, but combine their efforts on larger jobs. Our challenge is to develop system architectures and algorithms that allow cooperation to take place. It would be highly desirable to develop a general framework for all cooperative tasks, but that goal is too ambitious at the moment. Even if we restrict ourselves to just manipulation problems, the centralized, general motion planning problem

in three dimensions has been shown to be PSPACE-hard[1] in [Reif 1979] and has been

proven to require doubly exponential time on nondeterministic machines in [Canny, Reif

1987]. Therefore, there is little hope of developing a distributed, cooperative system

capable of solving all the classes of manipulation and navigation problems given our

current computing tools. Our research focuses on developing algorithms for more specific

tasks that may have tractable solutions.

1.3: Goals

Distributed placement has two components, positioning and planning. A

manipulation system must be able to physically position objects in an environment and it

must be able to plan a series of positioning actions that result in safe movement of the

manipulated object from its initial position to the specified goal. The development of

planning, or navigation, algorithms for cooperative manipulation systems is the focus of

this thesis.

This thesis investigates cooperative manipulation systems that function in

unmapped environments. The only algorithms that we will consider will be on-line. Off-

line algorithms, which require maps, plan a complete path before manipulation begins,

while on-line algorithms make choices in real-time, as the task proceeds. The off-line

solution to motion planning with uncertainty was proposed by [Lozano-Peréz, Mason,

Taylor 1984]. The placement problem can be considered an instance of the motion

---

[1] PSPACE-hard problems require a polynomial amount of space to solve, even on nondeterministic Turing

planning problem in which the planning and movement functions are completely external to the object being moved. We are interested in on-line solutions for unmapped, dynamic spaces, so off-line algorithms that require maps are not applicable and methods that use real-time sensor data are necessary.

One method of manipulating large objects in the placement problem is by pushing them along paths to the goal location. When pushing, however, the size and shape of the manipulated object may impair the ability of the manipulator robots to adequately sense their environment in order to perform on-line navigation. One solution to this potential problem was explored in [Rus, Kabir, Kotay, Soutter 1996]. In this work, a heterogeneous system of robots was implemented that assigned all navigational functions to a small scout robot, while two large "mover" robots were tasked with manipulating the given object, in this case a couch or a large box. The mover robots place objects by pushing them, but the object's bulk usually prevents the movers from sensing the environment and avoiding obstacles. Therefore, the scout robot functions as a mobile sensor for the system and positions itself alongside the system, traveling a parallel course and handling all navigational functions, as shown in Figure 1-1. The goal of this system, which the work presented in this thesis extends, is to move a large object across a room.

---

machines.

Figure 1-1: Rus, et. al.'s, original distributed manipulation system
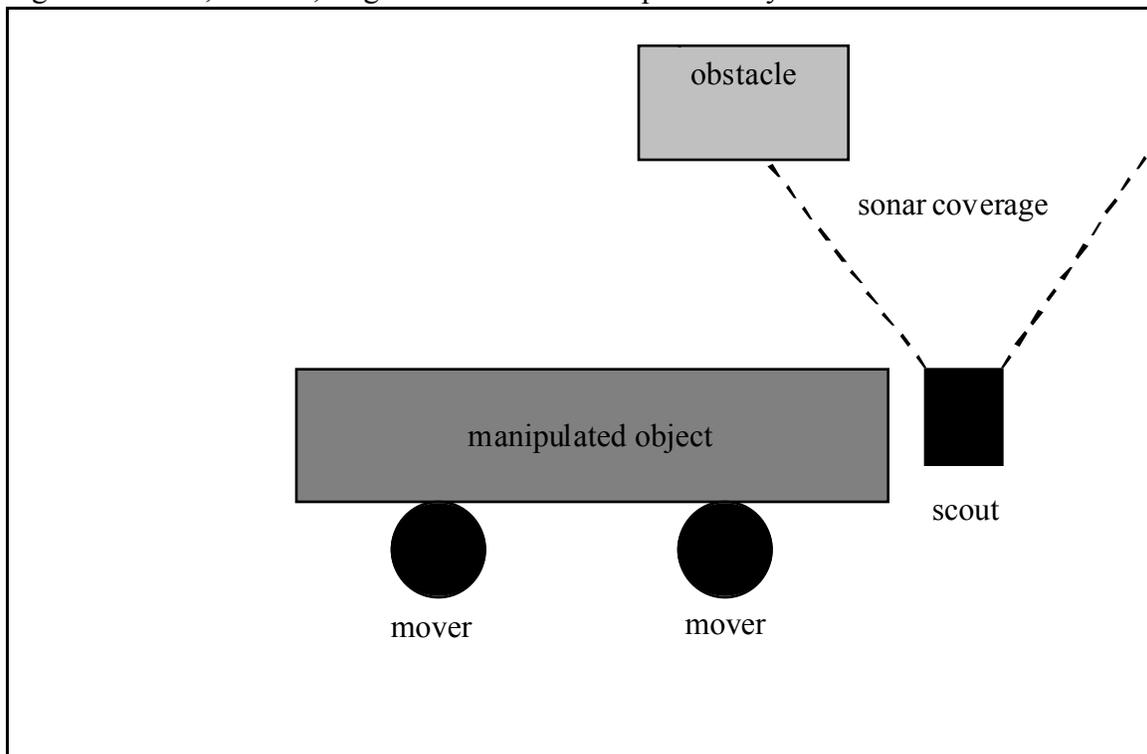
This previous work developed an algorithm that used only one mobile sensor for navigation and the implementation of this algorithm only treated obstacles on the right hand side of the object and its path. Our goal is to generalize this scouting system and examine algorithms that allow the system to successfully solve the placement problem in environments in which obstacles may occur anywhere around the manipulation system. We also wish to make navigation scalable and fault-tolerant.

Although the system extensions developed in this thesis handle a wider variety of obstacle configurations, they are restricted to spaces in which every pair of obstacles is separated by enough space so that the system may pass between them. Because the system is non-holonomic and cannot backtrack effectively, this assumption is necessary

to prevent the possibility that it may become stuck at a dead-end in an environment that does contain a path to the goal.

The project described in this paper is an effort to extend the original scouting system's capabilities by distributing the navigational functions among multiple mobile sensors.

1.4: Related Work

This work is inspired by research in cooperation, distributed robotics and systems, and motion planning. There is a widespread recognition that cooperation is an important area of research and many interesting approaches have emerged. [Mataric 1995] examines the use of biologically based local control laws, such as safe-wandering, aggregating, and dispersing, that can be combined to produce complex group behaviors such as flocking. A variation on this approach is discussed in [Shibata, Ohkawa, Tanie 1996], which describes cooperation as an emergent behavior resulting from the local "emotions" of participating robots. An emotion hierarchy is developed which correlates robot emotions to their local states and encourages adjustments that produce globally desirable behavior. [Parker 1992] provides a good overview of global and local control laws, comparing emergent cooperation to globally imposed cooperation, and concluding that a mix of local and global control and knowledge is necessary to achieve optimal results.

[Cai, Fukuda, Arai, Ishihara 1996] examines the issue of cooperative navigation for a Distributed Autonomous Robotic System (DARS), specifically the Cellular Robotic

System (CEBOT) model. They describe a method in which each element of the system can improve its preplanning for movement by collecting sensor information from other elements of the system. Other cooperative tasks that have been investigated include search and rescue in [Jennings, Whelan, Evans 1997] and pursuit-evasion systems in [Guibas, Latombe, LaValle, Lin, Motwani 1997].

Some researchers have created architectures for single robots that are implemented in terms of cooperating, subsystems. [Brooks, Connell 1987] promotes a subsumptionist approach to robot design which seeks to produce higher order behaviors as a result of interacting sub-behaviors. Work on the concept of intelligent cooperating subsystems in [Harmon, Gage, Aviles, Bianchini 1984] provides useful terminology that can be adapted to multi-robot distributed systems such as our own. Harmon places all messages between distributed robot systems into two categories: reports, which "maintain the consistency between the distributed parts of the world model," and plans, which control actions and the model's state. The primary differences between the two algorithms discussed in the subsequent sections can be described using Harmon's terms. The first algorithm for multiscouting cooperation, negotiation, requires both plans and reports to pass between the scouts, while the second algorithm, bidding, requires only plans and relies more heavily on sensor data to provide information on the current state of the scouts' cooperation.

Our work on distributed mobile sensing builds on the single-scout system proposed in [Rus, Kabir, Kotay, Soutter 1996]. The distributed manipulation algorithms for the mover robots were developed in [Rus 1993], [Donald, Jennings, Rus 1993],

[Donald, Jennings, Rus 1994a], [Donald, Jennings, Rus 1994b], [Donald, Jennings, Rus 1996], and [Rus, Donald, Jennings 1993]. Distributed, cooperative tumbling of objects is proposed and investigated in [Sawasaki, Inoue 1996].

There has been a great deal of research in distributed fault-tolerance, much of it the area of operating systems. Recovery after system components fail is investigated in the bully algorithm proposed in [Garcia-Molina 1982].

1.5: The Single-Scout Distributed Manipulation System

In this section we review the algorithm and experimental setup presented in [Rus, Kabir, Kotay, Soutter 1996]. This thesis builds on and extends this work and uses a similar experimental design.

Experimental Setup: The system employs two mover robots, Bonnie and Clyde, which are nearly identical RWI B14 robots. Each is equipped with a row of contact sensors, a set of discrete sonars, and infrared detectors. Using a distributed algorithm coded in Scheme, Bonnie and Clyde are able to cooperatively move large objects, such as a couch or a large, rectangular box. They require no explicit communication to perform this task and no specialized knowledge, except knowing if they are the right or the left robot in the system. When their manipulation processes are running and initialized, Bonnie and Clyde push the object they are manipulating from behind and can respond to four basic commands - "right," "left," "straight," and "stop." The system has been developed so that, given the right sequence of commands, Bonnie and Clyde should be able to push their object across a room, or along a given path, avoiding obstacles. At the moment, the

system is unable to reverse course in any reasonable or reliable manner, but modifications to the mover software may make this possible in the future.

The scout robot, or mobile sensor, is named "Jerry," an RWI Pioneer robot that is controlled via a radio modem. The scout software, originally developed in C, has, in the years since the original publication of the work, been rewritten in C++ and implemented as a simple class hierarchy, which will be discussed later. Also, the scout's hardware has been enhanced with the addition of a rotating sonar.

The Single-Scout Algorithm: Figure 1-1 shows an example of the scouting system's configuration. The scout robot sits to the right side of the object being pushed. Using his left-side sonars, which include the rotating sonar, Jerry attempts to steer a course parallel to the side of the object. Simultaneously, the scout scans ahead with his front sonar sensors, attempting to detect potential obstacles. When one is detected, the scout sends an appropriate command to the pushers, which are linked to the same Ethernet as the computer running the scout software.

Figure 1-2: The single-scout distributed manipulation algorithm

```
while (not at the goal)
        scout monitors system and obstacles
                if (obstacles exist)
                        order avoidance
                else
                        order straighten out
                issue command
        movers execute command
                (straight, left, right, or stop)
```

Jerry is unable to detect obstacles that block the left half of the system, as seen in

Figure 1-3. Because of this, the system may crash into such an obstacle – an

unrecoverable failure, since backtracking is impossible. There are two potential solutions

to this problem.

Figure 1-3: The original, single-scout system misses a dangerous obstacle
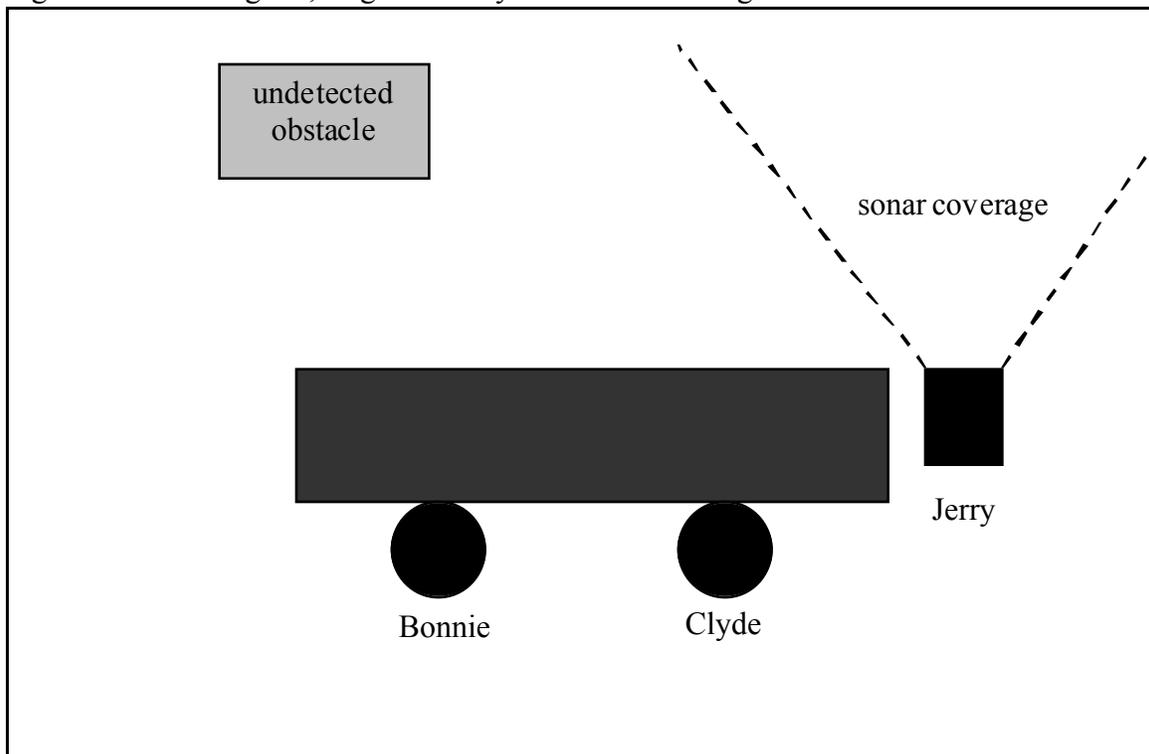
One possible solution is to give Jerry more flexibility and allow him to move in front of the system, rather than alongside it. This would eliminate the positioning constraints that prevent him from detecting obstacles on the left side of the system. Given Jerry's hardware limitations, specifically his complete lack of rear sonar sensors, this would most likely require the development of a two-mode manipulation system. First, the mobile sensor would scout a path, while the mover system remained stationary, and then the mobile sensor would return to the system and guide it through that step.

A different approach is to extend the system's sensing abilities by employing multiple scouts. If a second scout existed on the other side of the system, it would be able to detect most, if not all, of the obstacles that Jerry cannot, and order the system to avoid them. *Multiscouting* is defined as the simultaneous use of multiple mobile sensors to guide

a manipulation system. Designing and implementing multiscouting extensions to the original project is the task of this thesis.

1.5: Multiscouting

The multiscouting extensions to the manipulation system have been designed with two goals in mind. First, in keeping with the goals of the project's first iteration, we would like to minimize communications requirements. Communication processing can negatively impact system responsiveness and unreliable communications have been found to be a primary source of error in previous distributed manipulation work. Secondly, navigation should not require a central coordinating process. Our goal is to produce a uniform system in which the mobile sensors all execute the same algorithm asynchronously and in parallel.

Two algorithms have been developed to achieve these goals, *negotiation* and *bidding*. In negotiation, each scout detects obstacles by checking the front sonar sensors, exactly as if they were operating in a single scout system. However, instead of commanding the pushers to carry out orders directly, the scouts negotiate with each other for control over the manipulation system. At regular intervals, the scouts consult over the network, exchanging (command, priority) pairs. Suggestions that the system move "straight" receive the lowest priority, "stop" suggestions have the highest priority, and the priorities for "left" and "right," are assigned using a formula based on the distance between the system and the object it is attempting to avoid. When the scouts negotiate,

the highest priority suggestion is transmitted as a command to the movers and both scouts are informed of which command was executed.

The bidding algorithm differs from negotiation in two respects. First, scouts send (command, priority) pairs directly to the mover robots and engage in no communication with their peers. At any given time, the pushers execute the highest priority command they are currently receiving. Command priorities are calculated in the same way as they are in negotiation algorithm. Secondly, no explicit reports are allowed. The movers do not inform the scouts as to what command is currently being executed and from which scout that command originated. This forces bidding scouts to rely on their sensors much more than negotiating scouts do. Without explicit knowledge of what command the pushers are attempting to execute, a bidding scout must be able to detect whether its desires are currently being obeyed or ignored using only sensor data. The bidding approach is discussed more fully in Section 5.

Although each of these algorithms has its benefits and drawbacks, this thesis will focus on discussion and implementation of the negotiation algorithm. Distributed robotics algorithms are very difficult to implement and debug. During the two terms devoted to this thesis, I only had time to work in detail with one algorithm. If one considers a range of possible algorithms, stretching from centralized control over all scouts to maximum distribution and minimal inter-scout communication, negotiation occupies a middle position, while bidding is closer to the distributed systems extreme. The bidding algorithm evolved from a consideration of how to remove communication from the negotiation

method. Therefore, it made sense to begin by implementing negotiation fully and only proceed to bidding once negotiation was soundly established.

The remaining parts of this thesis are divided up as follows. Section 2 discusses the negotiation algorithm, describing it and providing a proof of its correctness for a subset of possible obstacle configurations. Section 3 examines the implementation of the algorithm and the practical challenges involved, while Section 4 analyzes the performance of the system. Section 5 looks at two potential alternative algorithms, a variation on negotiation and the bidding method. Finally, Section 6 concludes the thesis with a discussion of potential future research areas.

2: A Negotiation Algorithm

We are interested in solving the placement problem with a distributed team of mobile robots. This problem can be decomposed into two parts: distributed sensing for computing motion directions and maneuvering the object along those directions. These two tasks are executed together repeatedly until the destination is reached. This section develops an algorithm for distributed sensing. Specifically, a negotiation protocol is developed that allows multiple mobile sensor scouts to agree on a correct path for the system.

Each scout only has access to a limited field of view, but together they can cover an area at least as wide as the manipulation system. The goal of the negotiation algorithm is to fuse the scouts' information into globally correct decisions. The system does not construct a map, but instead uses sensor values and geometric information about the system size and shape to decide if the system should stop, turn left, turn right, or move straight.

The algorithms developed here are designed to solve the on-line manipulation case, which is distinct from the off-line case. In off-line manipulation, a map indicating the locations of all the obstacles is available to the manipulators. In our placement problem, no map is available and the obstacles may be dynamic, so we are required to detect obstacles and plan paths as the manipulation occurs. The system places objects in a two-dimensional environment by pushing them from their initial location to desired goals. Enough mobile sensors must be available and must be positioned so that, in combination, they can continuously monitor the space in front of the manipulation system.

2.1: Algorithmic Description

In this section we develop the negotiation algorithm for distributed sensing and decision making. The algorithm can utilize n mobile sensors, but here we treat the n = 2 case, which provides the minimal number of sensors to cover the necessary field of view for our system. By placing one scout on each side of our experimental system we achieved continuous sensor coverage of the region in front of the system. The system is minimalist in flavor, but has all the resources necessary to solve our placement problem. For simplicity of exposition, we continue to present our algorithm for n = 2, but all analyses are done for arbitrary values of n. We generalize the system to n scouts in Section 2.2.

The two scouts maintain a course parallel to and alongside of the manipulated object. They concomitantly monitor the manipulated object and the region in front of it. Each scout has knowledge of its global angle, which is its current angle relative to its angle at system start time. For purposes of this system, it is assumed that the scouts, and the rest of the system, start on the vector that the system will attempt to steer. Our goal was the development of an algorithm that would enable the system to manipulate an object across a room along a chosen vector. As will be discussed in Section 5, this algorithm can be adapted to move along a chosen path as well.

Figure 2-1: Negotiation Algorithm for Cooperative Manipulation

```
while (not at the goal)
        scouts monitor system and obstacles
                if (obstacles exist)
                        negotiate avoidance
                else
                        negotiate straighten out
                issue command
        movers execute command
                (straight, left, right, or stop)
```

Figure 2-1 gives a top-level view of the algorithm. For each iteration, the scouts

monitor the object being manipulated with side-mounted sensors and scan ahead for

obstacles. After examining their sensor data, the scouts communicate with each other to

decide how to proceed. If no obstacles are detected and the system is heading towards its

objective, the scouts will decide that the system should proceed straight. If obstacles are

detected or if the system is not on course to the goal, the scouts will communicate to

determine the best motion direction for avoiding the obstacle or returning to the desired

course. If one scout loses track of the system and needs to reacquire it, the negotiation

will result in a decision to stop the system while the lost scout searches for it.

The key contribution of this thesis is the algorithm used by the scouts to

determine the best path for the system. This is a difficult problem because if both scouts

detect obstacles they may wish to impose contradictory motion directions. For example,

if one scout detects an obstacle blocking the left half of the system and its partner detects

an obstacle blocking the right half, the first scout will want to turn right, while the second

will see a left turn as being most desirable. A priority scheme is necessary to ensure that the scouts always select the globally optimal decision.

The priority algorithm for negotiation is described in Figure 2-2. It consists of four simple rules. The first is that if either scout asks the system to stop, that command takes priority over all others. Stops are ordered when a scout has lost track of the manipulation system and needs to regain contact. Because sensor coverage will be significantly impaired by the loss of a scout, allowing scouts the opportunity to find the system if they become lost takes priority over all other tasks.

The second priority rule is that a "straight" command has the lowest priority. A scout will only order straight movement if it detects no obstacles and believes that the system is headed on a direct path to its destination. If its partner detects an obstacle or believes a course adjustment is necessary, its commands take precedence.

If both scouts order turns, we need to decide which turn is more important. Each scout assigns a priority to every attempt at obstacle avoidance. The priority of the turn is equal to (maximum sonar range – distance to the obstacle). The system gives highest priority to avoiding the obstacles that are nearest to it. Avoiding imminent crashes takes priority over avoiding future crashes. Turns designed only to return the system to its original motion vector receive a priority of 0.

In order to make communication more efficient, our system designates one scout as responsible for sending negotiation results to the pusher system. This scout is termed the "master," but the only difference between it and its partner is the responsibility for communicating negotiation results. Each scout runs the priority algorithm independently

and has access to its partner's suggestions. In order to prevent system deadlock, any ties

generated by the priority algorithm are broken in favor of the master.

Figure 2-2: Negotiation priorities

1) A "stop" takes precedence over any other command.

2) A "straight" has the lowest precedence (and, therefore, is only executed if both scouts suggest it).

3) Otherwise, precedence is established by examining the priority of the suggestion. The highest priority suggestion wins.

4) If a tie exists, it is broken in favor of the master's suggestion.

2.2: Generalization to n-Scouts

The algorithm's successful operation depends on two conditions. First, the mobile

sensors must be numerous enough and correctly positioned so that they can detect

obstacles across the width of the entire system. Secondly, they must be able to measure

how far any obstacle they detect is from the system. So long as each scout has knowledge

of the system geometry and its location relative to that geometry, it can calculate the

distance between the system and any object it senses. Therefore, a system with n-scouts

that satisfies these conditions can operate using the same algorithms as a two scout

system and the proofs of algorithmic correctness developed in Section 2.2 apply to the n-

scout case. Because each scout executes the same algorithm, we can increase the number

of scouts by duplicating the program with parameters adjusted for each new robot in the system.

2.3: Analysis and Proofs of Correctness

In this section we analyze the negotiation protocol and demonstrate that the multiscouting system is efficient, fault-tolerant, and produces correct motion commands for the placement problem.

2.3.1: Computational Complexity

Theorem 2.1: The negotiation algorithm requires $O(n^2)$ messages per decision, where n is the number of cooperating mobile sensors in the system.

Proof:

In the algorithm described above, each mobile sensor must communicate a (command, priority) pair representing its suggestion for the next move to each cooperating mobile sensor, so that all the scouts are capable of calculating the result of the negotiation.

With n scouts, therefore, each decision requires (n-1) messages from each scout. Therefore, a total of (n-1) * n messages are passed per decision. Since (n-1) * n = $n^2 - n$, this is $O(n^2)$ messages.

Theorem 2.2: There is a protocol equivalent to the negotiation algorithm that requires $O(n)$ messages per decision.

Proof:

In the original protocol, each scout calculated negotiation priorities independently. But if all mobile sensors send their (command, priority) pairs to a central, designated coordinator, that coordinator can compare all the commands and inform each scout of the result.

In this case, a decision would require (n-1) (command, priority) pairs – one sent from each cooperating scout to the coordinating scout – and then (n-1) messages from the coordinator to inform the other mobile sensors of the result. This is a total of 2(n-1) messages per decision, for O(n) growth.

2.3.2: Fault-tolerance

The algorithm is designed so that if one scout fails the remaining scouts will continue to navigate the system as best they can. Every scout executes an identical algorithm and detects failure in its partners by means of a communications timeout. If a partner fails to negotiate within a given time period, other scouts assume that partner is dead and continue as before, excluding the dead scout from future rounds of negotiation. The system will still plan the placement successfully if all the obstacles it encounters fall within the sensor coverage provided by the remaining scouts.

If only one scout remains after a failure, that scout will proceed using the single-scout algorithm proposed in [Rus, Kabir, Kotay, Soutter 1996], and extended by myself and Christine Alvarado in subsequent work. All scouts maintain network connections to the movers, so that if it is the master scout who fails, a slave can switch into master mode

and handle communication itself, using a distributed coordinator election algorithm such as the bully algorithm developed in [Garcia-Molina 1982].

In some cases, it may be desirable for the active mobile sensors to reposition themselves after a failure removes one of their partners from the system. In the two-scout system, for example, the system could better handle failure if the remaining scout moved ahead of the system to increase its field of view.

2.3.3: Correctness

An on-line placement system must avoid multiple errors. First, it must not deadlock in any situation. Deadlock is avoided in our algorithm by the use of a priority system that deterministically decides upon a command to be executed no matter what suggestions the scouts submit. Even if two commands have the same priority, a tie-breaking mechanism exists.

Secondly, the system must compute a path that leads to the goal and not reach dead-ends. Unlike an off-line system that can use a map to discover any paths that exist, it is possible that an on-line placement system can become stuck, even in an environment that contains a solution path. One possibility is illustrated by the following example. In Figure 2-3, the system can avoid all obstacles easily by turning right. However, without a map, the system has no reason to favor right over left if it encounters the first obstacle while the others are too far away to be detected. So, the system turns left to avoid the first obstacle and is then trapped. In a holonomic system, it would be easy to backtrack and try another route. Our manipulation system, however, is non-holonomic and,

therefore, incapable of effective backtracking. Therefore, it would be helpless in a

situation such as the one described.

Figure 2-3: An on-line system fails when a path was possible
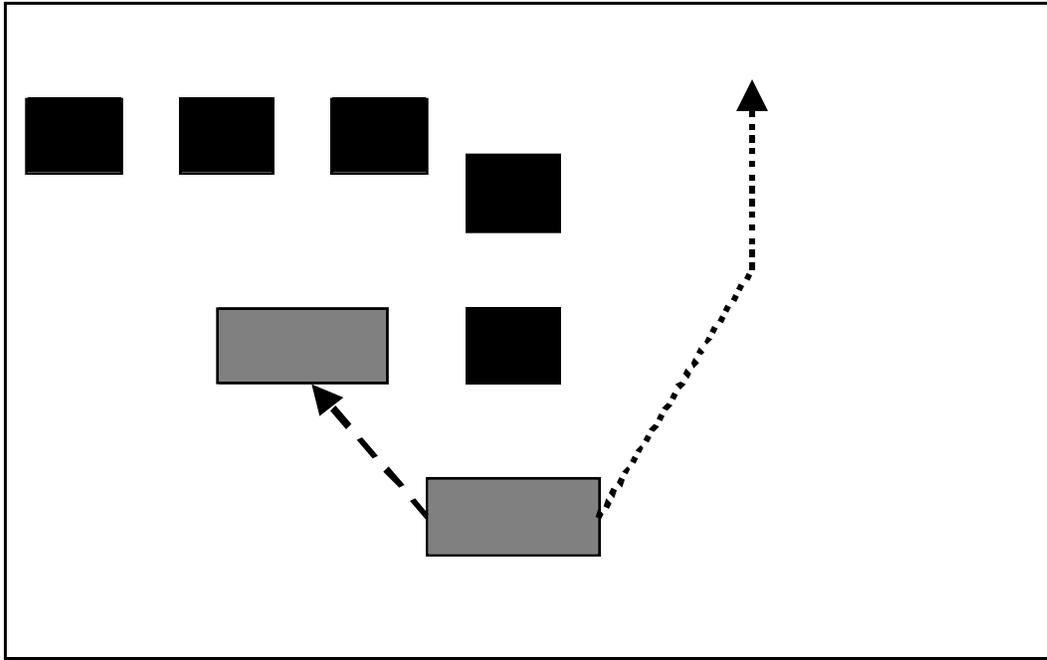
Since any on-line system with limited sensor capabilities is susceptible to traps of

the type illustrated in Figure 2-3, we constrain our problem space to only include

placement problems in which any pair of obstacles is separated by enough space for the

manipulation system to pass between them. The proofs that follow demonstrate that the

algorithm can correctly solve the placement problem for this class of obstacle-spaces.

Theorem 2-3 demonstrates that a centralized algorithm for placement in these

spaces exists and Theorem 2-4 demonstrates that the distributed algorithm described in

Section 2.1 reduces to the centralized method. The centralized method assumes that all

sensor data is global and is processed by a single algorithm. Its three fundamental rules are

outlined in Figure 2-4. As noted in Section 1, the placement problem we are attempting to

solve here requires passage from one side of a room to the other, so the system need not follow a specific path. It is sufficient that its net vector of motion move it to the other side of the room.

In the centralized algorithm, the system starts moving along an original vector of motion that points directly across the room. Odometry is available to store the global angle of this vector. In the absence of any obstacles, the system should turn back to the original vector and proceed straight along it. If one obstacle is detected, the system turns to avoid it, passes by it, and returns to its original vector. If multiple obstacles are detected, the closest one is avoided first in one direction, then the second-nearest obstacle is avoided by turning in the opposite direction. This results in the system passing between the nearest two obstacles, which is always possible because of the constraints we have placed on the obstacle space. If several turns are executed, the original vector is recovered by comparing the current global angle of the system with the stored global angle for the original motion vector.

In the context of this algorithm, a correct path is one that leads us from one side of the room to the opposite side. It is not the shortest possible path to the goal. If we used off-line methods and had access to a map, it would be possible to calculate the most efficient path, but in our on-line system we can only make decisions based on limited, local information and we cannot backtrack to investigate alternate routes. Therefore, correctness in this context is only measured by the fact that the algorithm is guaranteed to eventually reach the goal.

Figure 2-4: Centralized algorithm for placement problem navigation

Case 1: In the absence of obstacles, steer the system back to its original vector.

Case 2: If one obstacle is detected, change the system's vector so that the system will pass to one side of it. Once the obstacle is avoided, straighten out to the original vector.

Case 3: If multiple obstacles are detected, avoid the closest one first, then the next closest one by turning in the opposite direction. This will result in the system passing between the obstacles.

Theorem 2-3: The centralized navigation algorithm finds the correct path when the environment is such that the pairwise distance between any two obstacles is at least as wide as the width of the manipulation system.

Proof:

This algorithm uses a greedy search strategy – make the minimal avoidance necessary at any one time and then try to return to the original vector once the system is free to do so. After any step is completed, the algorithm will have returned to the original vector.

The algorithm's three cases are described in Figure 2-4. The net vectors resulting from each of these cases indicate forwards movement, which is defined as any movement that has some component along the original vector. No turns will have an angle of more than 90 degrees because the system's center of rotation must always be at one end. The only situation that could require a 90 degree rotation would be avoiding an object whose nearest point to the system is the center of rotation, and in that case the system could and

would turn in the opposite direction – right-hand obstacles are avoided to the left, and vice-versa. Figure 2-5 shows that the maximum turn necessary, which is required for an object directly ahead of the system's center, is less than 90 degrees in magnitude.

Figure 2-5: The maximum avoidance turn is fewer than 90 degrees
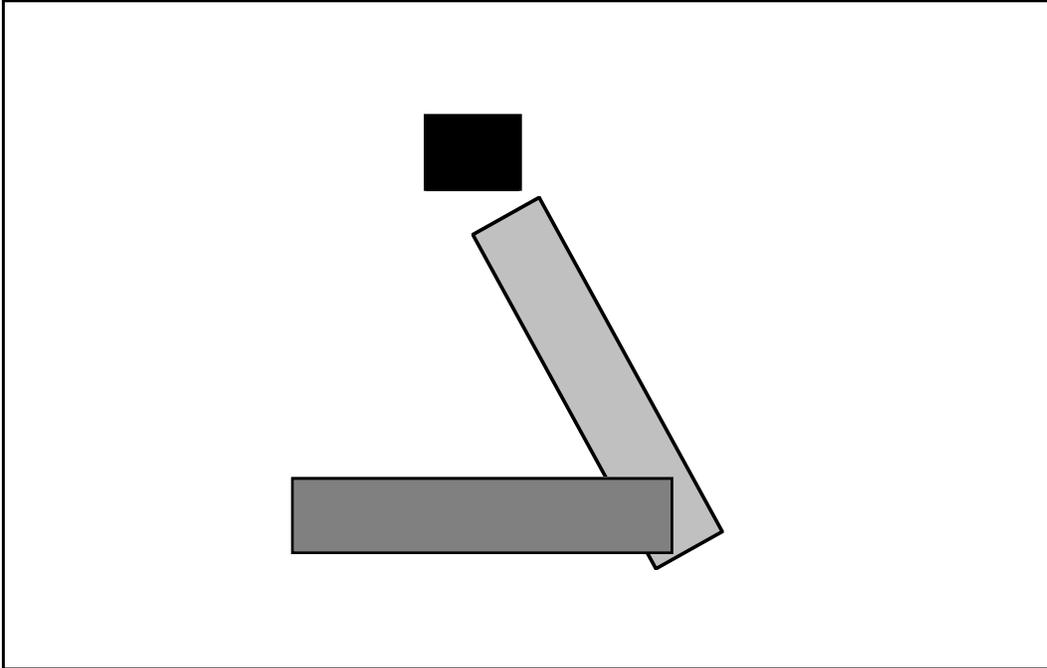
In Case 1, the system travels straight and maintains its motion vector. In Case 2, a single obstacle is avoided with a turn of fewer than 90 degrees and then the algorithm returns to the pre-avoidance vector. The net vector indicates forward motion because we never turned more than 90 degrees away from the original vector. In Case 3, we avoid obstacles by passing between them, avoiding the nearest one in one direction, then the next in the opposite direction. Since neither turn can have a magnitude of more than 90 degrees and because they are in opposite directions, the net vector has a sub-90 degree deviation from the original vector, so the system still has a net vector with a component that indicates the same direction as our original vector.

In all three cases, the system will have made some progress in the direction of the original vector. Because all obstacles are separated by enough space to allow the system to pass between them, all three cases can be executed in the circumstances described. We will, therefore, make progress towards our goal in all three cases described in Figure 2-4 and we will avoid all obstacles. Therefore, the algorithm solves our placement problem for the restrictions given.

Since the centralized algorithm is correct, the distributed algorithm will also be correct if it produces identical behavior as the centralized algorithm in each of that algorithm's three cases. The details of the distributed algorithm are discussed in Section 2.1. The priority system for conducting negotiations developed in that section result in the system always selecting the decisions that the centralized algorithm requires.

Theorem 2-4: The distributed negotiation-based navigation algorithm for the distributed system described in Section 2-1 is equivalent to the centralized method described in Figure 2-4 and, therefore, correct for all spaces that the centralized method is correct for.
Proof:

Consider the Cartesian coordinate system in which the front of the rectangle containing all of the system elements is a line segment along the x-axis, centered at (0,0). The length of the entire system, including all the robots, is 2L, and the width, similarly defined, is 2W.

Figure 2-6: Coordinate system
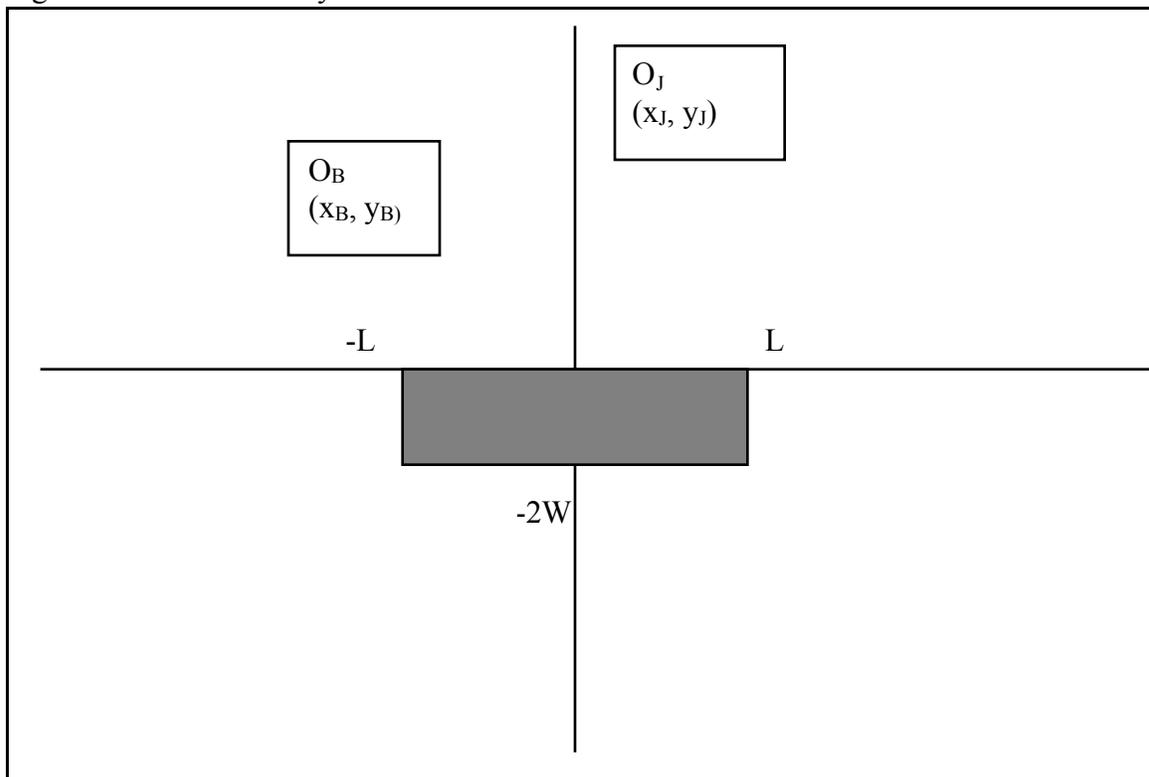
We will refer to the scout positioned along the left side of the system as "Ben" and the right-positioned scout as "Jerry". Assume that Ben is located at approximately (-L, 0) and Jerry is located at approximately (L, 0).

Case 1: If no obstacles are detected, the distributed algorithm will, like the centralized algorithm, direct the system straight along the original vector.

If neither Ben nor Jerry detect any obstacles, they will attempt to straighten out, as is required by their scouting algorithm. In our implementation, if one of them detects that the system can be straightened by ordering a turn that it will be on the outside of (a right turn for Ben, a left turn for Jerry) and there are no other obstacles, that scout will

order the system back onto the original vector while his partner suggests continuing

straight. Since straight has the lowest priority, the system will return to its original

vector. Therefore, this case reduces to the first part of the centralized algorithm.

Case 2: If one obstacle is detected, the distributed algorithm will, like the centralized

algorithm, direct the system to avoid the obstacle and straighten it out after the avoidance

is complete.

 If only one scout detects an obstacle, it will suggest that the system turn to avoid

it, while its partner will suggest returning to the starting vector. Turning has a higher

priority than continuing straight, so this situation correctly corresponds to the second

part of the centralized algorithm. After avoidance is complete, the straightening out

behavior of Case 1 returns the system to the original vector.

Case 3: If two obstacles are detected, the distributed algorithm will, like the centralized

algorithm, direct the system to pass between them by first avoiding the nearest obstacle

in one direction and then avoiding the next obstacle by turning in the opposite direction.

After all avoidances are complete, the system will return to its original vector.

 In this case both scouts detect obstacles and suggest avoidance maneuvers. If their

object-following algorithms are operating correctly, Ben and Jerry will be positioned at (-

L, 0) and (L, 0) respectively, within an error E that is negligible when compared to L and

W. Without loss of generality, it can be assumed that O_B, the object Ben is trying to

avoid, is in quadrant II of the graph, while O_J, the object Jerry has detected, is in quadrant

I. If a scout detects an obstacle that is in its partner's quadrant, that scout will

undoubtedly have also detected it, since it is nearer to it. Therefore, the scouting

algorithms can be restricted to detecting objects in their own quadrants without any loss

of overall sensor performance.

Given the assumption that O_B is in II and O_J is in I, we need to decide which of

them is "closest" to the system. The definition of distance that makes the most sense for

this is the obstacle's distance from the line segment that forms the front of the system

because this indicates how near the system is to crashing. Each scout detects its obstacle

as a distance reported on a sonar receiver that points at a particular angle away from the

system. These distances are denoted D_B and D_J, respectively. Since the scouts should

ignore any obstacles that are outside the system's current path, Ben will only discover

obstacles with x-coordinates ranging from $-L$ to 0. Similarly, Jerry only discovers

obstacles that exist in the x-range 0 to L. So, consider that Ben's sonar reading comes from

an angle of A_B, towards the y-axis, while Jerry's sonar reading comes from an angle of A_J,

also towards the y-axis.

Figure 2-7: Obstacle – system distances
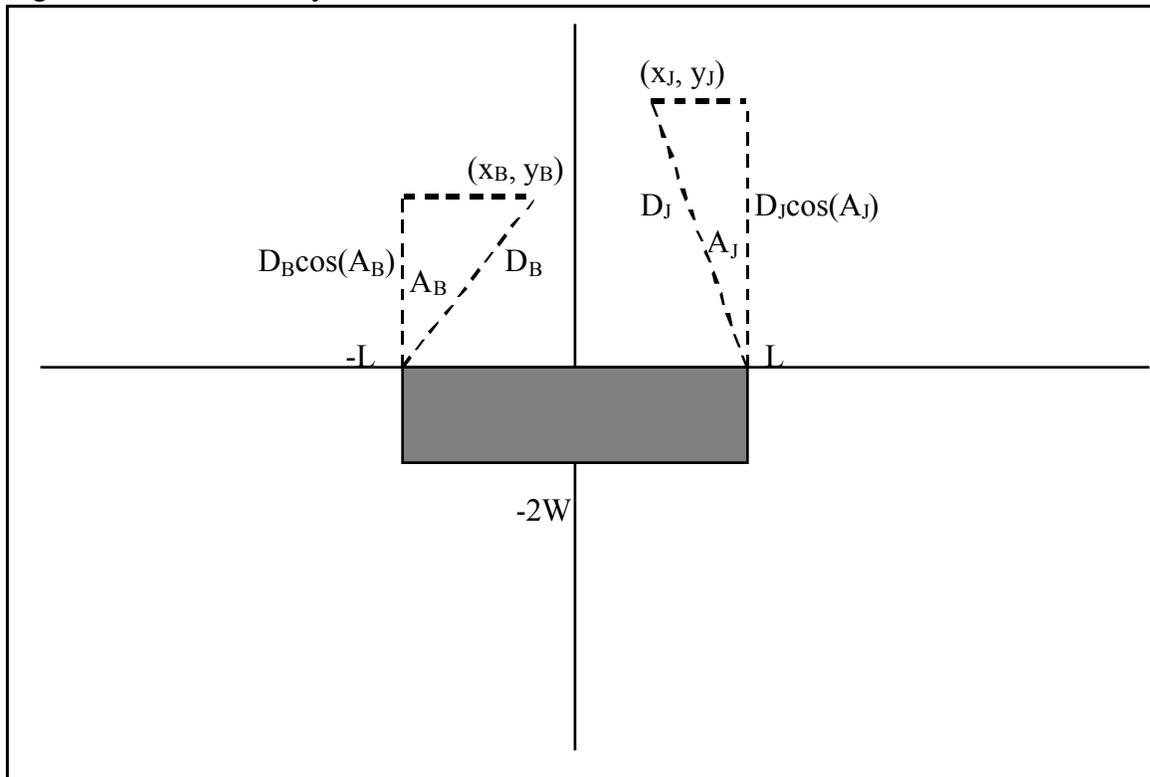
In that case, it is a simple matter to calculate the distance between the front of the system and each obstacle. Because the front of the system is along the x-axis and centered at (0, 0), the distance between it and O_J is the y-component of D_J, D_{Jy}. Similarly, the distance between the system's front and O_B is the y-component of D_B, D_{By}. As demonstrated in Figure 2-6, $D_{Jy} = D_J\cos(A_J)$ and $D_{By} = D_B\cos(A_B)$. If the scouts use these distance calculations, the system will always attempt to avoid the closest object first.[2]

For this distributed case to perfectly correspond to the centralized behavior, it is necessary that the system avoid the second obstacle by turning against the direction of

---

[2] In the implementation, the cosine factor is ignored because the sizes of the obstacles used for testing and the limitations enforced by the robot lab's size and configuration make it insignificant. We can only create

the first avoidance turn. Consider the following situation with respect to Ben, the

leftmost scout, which generalizes to any avoidance situation. At some point in the past,

Ben has detected an obstacle that was closer to the system than any other and therefore

ordered a turn to avoid it. As a result, the system has made the minimal possible right

turn to avoid the obstacle and traveled past it. Assume that Ben's detection algorithm

operates under the following two constraints – after ordering the minimal angle turn

possible, he will not attempt to avoid any further obstacles until the system's x-axis

passes the obstacle that it is currently avoiding. Secondly, assume that Ben's maximum

sonar range is 2L, the length of the system. This will not limit his effectiveness as a scout

because he only needs to discover any obstacle O_B in time for the system to avoid it. The

maximum detection distance would be required for an obstacle on the y-axis. With a

detection distance of 2L, Ben, positioned at (-L, 0), would be able to cover that axis up to

distance a, where $L^2 + a^2 = 4L^2$ by the Pythagorean theorem, resulting in a $= \sqrt{3}\,L$. The

system rotates around one fixed end, so a turn to avoid an object along the y-axis requires

the right end of the box to remain fixed and the left end to swing such that the box, in the

pre-turn coordinate system, has one end on the y-axis and the other still at (L, 0). The

right triangle thus formed has a hypotenuse of 2L (the system length), one side of length

L (along the x-axis), and the other side of length $\sqrt{3}\,L$ (along the y-axis), which allows the

system to fit, since O_B was detected at $\sqrt{3}\,L$. Since any other obstacle can be avoided

with less advance warning than one on the y-axis and since the y-axis location provides

---

test conditions that involve extremely low values for A_B or A_J. In a complete implementation, however, the

the minimum possible advance warning, restricting the sonar range to 2L will not impact system effectiveness.

Therefore, if the system has just avoided an obstacle, Ben resides at one side of obstacle O_B. His maximum detection range is 2L, but according to the constraints of the problem, no other obstacle can be within 2L of O_B because the system cannot pass through a pair of obstacles separated by less than 2L + E, where E is a positive factor that allows for small positional uncertainties as the system maneuvers. Therefore, Ben sees no other obstacles. The only possibilities are that no objects are seen, in which case the system straightens out with a leftwards turn, or that Jerry detects an obstacle and avoids it by ordering a leftwards turn. Ben will remain out of sonar contact with any obstacle until the system is straight or has avoided an obstacle by turning against the turn he has just directed. In the first case, the system has returned to its original vector. In the second, the system will attempt to return to its original vector once Jerry has finished his avoidance since he will also, by similar logic, finish out of contact with any obstacle. If Jerry has turned past the original vector, we already straightened out at some time in the past. If Jerry has not turned past the original vector, the straightening out turn will be a left turn and, similarly, any new obstacle detected will be avoided by continuing left so that the original vector will eventually be reached.

Therefore, just in the centralized algorithm, the distributed algorithm avoids multiple obstacles by passing between the two nearest obstacles – turning one way to

---

cosine term is necessary to ensure proper behavior in every circumstance.

avoid the first and in the opposite direction to avoid the second, and eventually returning to the original vector.

The distributed algorithm's behavior corresponds exactly to that required by the centralized algorithm in each of the three cases. Therefore, the distributed negotiation algorithm is equivalent to the centralized version and, therefore, is correct for navigating any finite field of obstacles in which all pairs of obstacles are separated by at least $2L + E$.

3: Implementation of the Negotiation Algorithm

### 3.1: Hardware

The scouting hardware consists of two Real World Interfaces Pioneer mobile robots, named Ben and Jerry. Each is equipped with a roughly semi-circular set of discrete sonar sensors wrapped around their front end. Jerry, who was used in the original single-scout system, is equipped with an additional left-side-mounted, servo-directed, sonar. Ben has a small video camera, which was remounted on his right side as a rotating-sonar substitute. Ben's image processing is controlled by a Cognachrome vision board that can perform simple color-tracking tasks. The vision system is capable of learning a color range and detecting the largest contiguous blob of that range. All control of the scout robots is performed by off-board workstations running the FreeBSD Unix operating system, and the lowest level of interaction between the robots and their off-board processes is handled by the Saphira 6.1e C libraries provided by RWI.
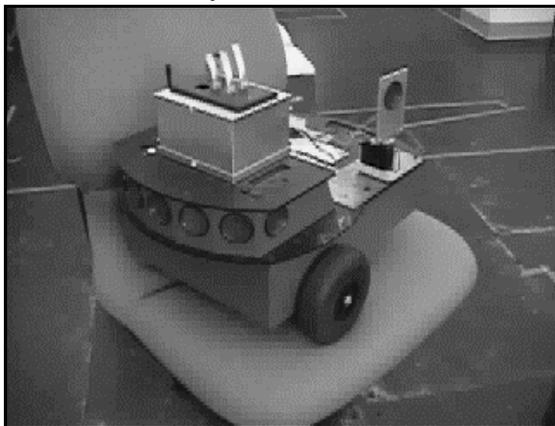
Picture 3-1: Jerry   Picture 3-2: Ben
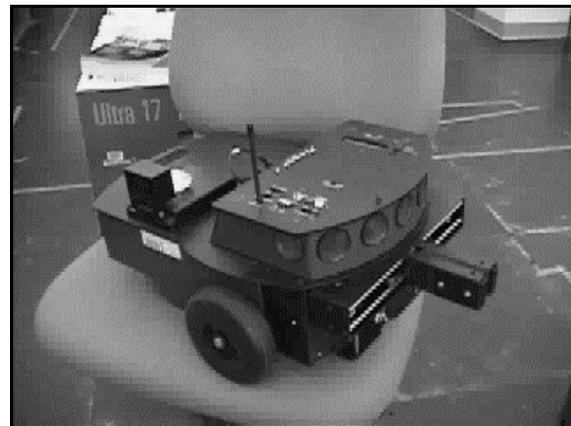
The mover system consists of two B14 robots developed by Real World Interfaces, named Bonnie and Clyde. These robots have multiple sonar and infrared

sensors, but these are not used in our project, since the scouts are responsible for all

environment sensing. The only sensors we employ are a row of contact sensors that each

robot uses to determine its contact and angle with the manipulated object. The mover

robots are controlled by onboard computers running the Linux operating system and

Scheme programs that implement their manipulation algorithms.

Picture 3-3: Bonnie and Clyde and the simulated couch

3.2: Software

Figure 3-1: Multiscouting software
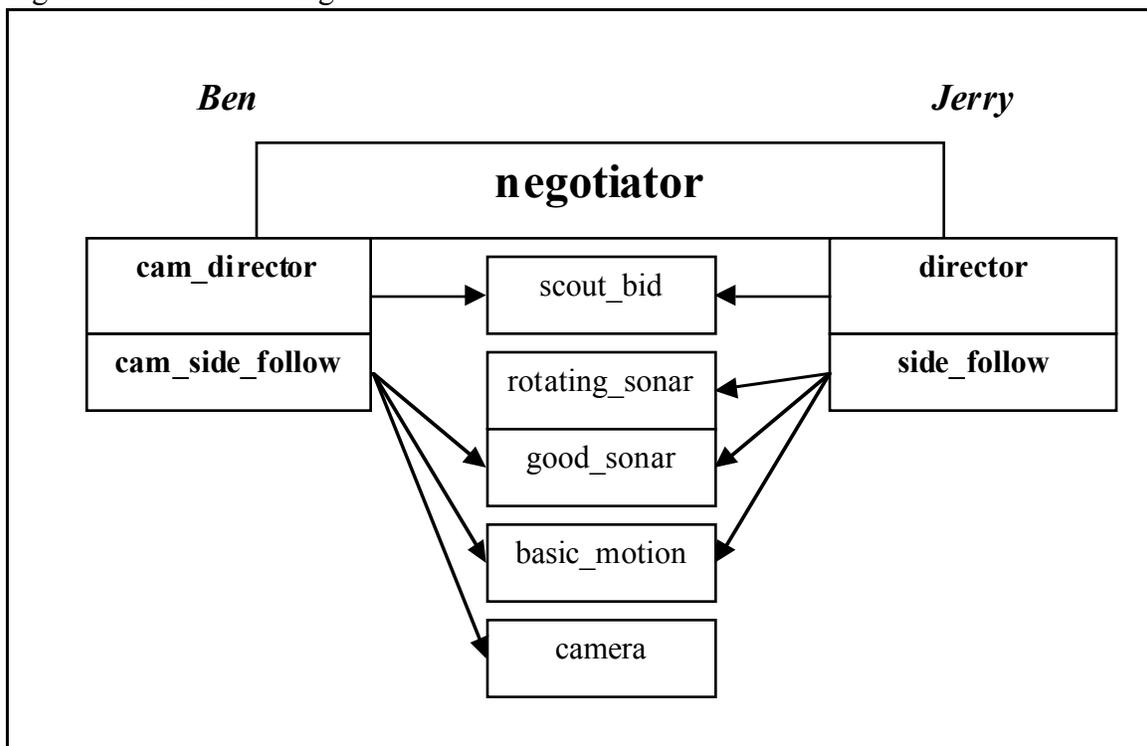
3.2.1: Implementation History

The multiscouting system was implemented as a hierarchy of C++ classes, shown

in Figure 3-1.[3] I developed the basic_motion, good_sonar, rotating_sonar, side_follow,

and director classes with Christine Alvarado. They encode the original, single-scout

algorithm described in [Rus, Kabir, Kotay, Soutter 1996]. These original classes were

designed for use with Jerry. For this thesis, the camera, cam_director, and

cam_side_follow classes were developed to make Ben capable of executing the single-

---

[3] Source code for the system is available at ftp://ftp.cs.dartmouth.edu/TR/TR98-332.code.tar.Z

scout algorithm. Then, the scout_bid and negotiator classes were created to implement the

multiscout, negotiation algorithm.

3.2.2: Negotiation Implementation

Each scout runs an independent process and only communicates with its partner

over the Ethernet at the negotiator level. The lowest level of classes – basic_motion and

good_sonar – provide low-level control of the Pioneer scout's hardware. The

basic_motion class provides simple movement methods and good_sonar provides both an

interface to the sonars and a mechanism for filtering out bad readings (sudden, non-

repeated spikes in the sonar data). The rotating_sonar class is derived from good_sonar

and adds servo-control for the rotating rear sonar and a simple algorithm capable of

scanning for the minimum reading. Objects of these classes are included in the side_follow

class, which implements the manipulated-object following behavior on Jerry. The director

class, which is derived from the side_follow class implements the single-scout algorithm

for monitoring the forward sonar array and commanding the pushers to execute the

movements necessary to avoid obstacles.

The cam_side_follow and cam_director classes use Ben's camera in place of

Jerry's rotating sonar – a substitution that resulted in some algorithmic changes. Most of

the modifications were in the cam_side_follow class, to take into account that the camera,

unlike the rotating sonar, could not provide angle data, given the limitations of the

Cognachrome processing algorithms. In order to control the vision system, a camera class

was implemented that can extract color-tracking information from the camera and process it into the distance and location information necessary for following a manipulated object.

The negotiator class replaces single-scout commands with multiscout cooperation. This class is capable of carrying out all the inter-scout communication tasks required for multiscouting. Derived from the director class, it overrides the private diretor::order_pushers () method (or, in the case of Ben, the cam_director::order_pushers () method) with a member function that conducts the negotiation algorithm whenever one of the scouts gives an order. To support negotiation, a scout_bid class containing the suggested command, its priority, and a Boolean value indicating if the command came from the scout or its partner was also implemented.

These additions required some changes to the director and cam_director classes. We modified the order_pushers () member functions to return the scout_bid of the successful command and modified all calls to order_pushers () to handle situations in which the transmitted command loses to a partner's suggestion. These changes were engineered so that the director code remained independent of other modules. Thus, director objects retained their functionality even if they were not negotiators. This provides support for fault-tolerance because it enables the single-scout algorithm to take control if all partners fail and leave a single scout in control.

Another implementation goal is to maximize the efficiency of communication and to minimize its effect on the other code modules. One issue is the effect of communication delays on the side_follow algorithm. Because it is crucial that the scouts keep up with all system movements in a timely manner, all network reads are made non-blocking. This

allows a scout to continue executing side_follow methods while it waits for its partner to submit a scout_bid.

The negotiator::order_pushers () method sends a scout_bid to the partner scout and waits for its scout_bid. After both bids have been exchanged, each scout compares them independently, using the precedence rules described in Section 2. Then, the master scout, who is charged with communicating the result of the negotiation to the movers, passes the winning command to its director (or cam_director, as the case may be) order_pushers () member function.

3.3: Implementation Challenges

Apart from the algorithmic challenges, the cooperative navigation project had to overcome many practical obstacles.

3.3.1: Heterogeneity

The original scouting system was implemented on our first Pioneer robot, Jerry. Jerry was a prototype model of the Pioneer series, modified to include a side-mounted, servo-controlled, repositionable sonar. When it came time to add a second scout, we had to adapt the algorithms and code originally developed for Jerry's hardware for Ben, a much newer model Pioneer which lacked a rotating side sonar and offered, in its place, a video camera and a Cognachrome vision processing board.

Before any significant implementation of the algorithms could be attempted, Ben's systems needed to be modified so that they could at least approximate the functionality

of Jerry's. First, the camera was moved from the front of the robot to its right side, to mirror the positioning of Jerry's rotating side sonar. Next, software had to be developed to process camera data into distance data. Unfortunately, due to the limitations of the Cognachrome vision board, this required significant amounts of environmental engineering.

Picture 3-4 shows that the box-sides the scouts must follow along are visually complex objects. First, there is a large strip of highly reflective duct tape across the area that must be tracked. According to [Wright, Sargent, Witty, Brown 1996], the system performs poorly when tracking objects with highly reflective surfaces and glossy colors. Secondly, apart from the difficulties caused by the reflective tape, there are four distinct colors on the unmodified box: purple, blue, white, and black. Although purple and white predominate, there are no areas with large patches of unmixed color. Because the system training algorithm needs to sample color areas from the tracked object at different angles, distances, and lighting conditions to be effective, the system will always pick up at least two of the colors on the box while the training is proceeding. Unfortunately, all of the box colors, especially white and black, are present in the background environment. Experiments demonstrated that after the system was trained to track an object in a wide variety of angle and lighting conditions it became very susceptible to picking up the background and treating it as the object to be tracked unless the tracking colors were sharply differentiated from the background colors.

The solution to this problem was covering the side of the manipulated object that Ben tracks with black construction paper and then placing a bright, fluorescent pink square in the middle of the black field, as shown in Picture 3-5. This provided the camera

system with a regularly shaped object with a distinctive color and sharply defined edges to track. After proper training, the system was able to return data indicating the tracking box's x-coordinate in Ben's field of vision (indicating how far forwards he was with respect to the manipulated object) and the observed area of the square, which is used to determine his distance away from the manipulated object.[4]

Picture 3-4: Normal box-side

Picture 3-5: Modified for vision-tracking
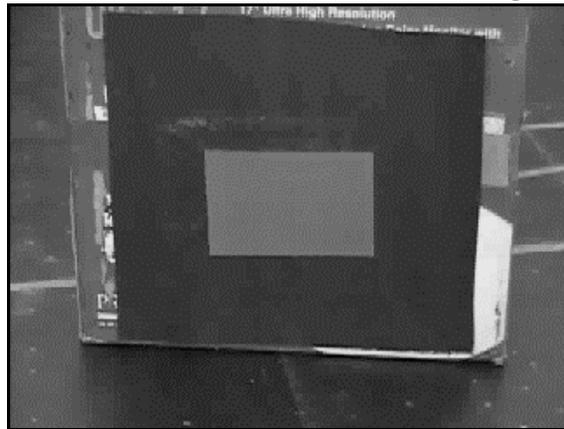
Even after the implementation of camera software that allowed the camera to give data similar to that provided by Jerry's rotating sonar, problems still remained. Although the camera could provide distances which, in fact, were more accurate than sonar distance measurements in most cases, it could not rotate to stay parallel to the manipulated object at all times. The knowledge of the rotating sonar's current angle forms an important part of the side-follow algorithm employed by Jerry and is especially useful in trying to decide

---

[4] Because the camera only offers monoscopic vision, the only method of approximating the distance between Ben and the manipulated object was to measure the area the color blob tracking functions returned at a distance of 30 cm and, using this as a base, derive all other distances by simply calculating the inverse square proportions with respect to this original measurement. Of course, this is not the most accurate way of measuring distance – changes in viewing angle can cause area changes even if no distance changes take place. However, for Ben's range of motion with respect to the box and taking into account the wide

what to do if the front, fixed, side sonar loses contact with the manipulated obstacle. On the other hand, the camera offered advantages of its own, since it provided information about Ben's position relative to the manipulated object that Jerry's sonar could not because the camera could see exactly how far ahead or behind Ben was running.

Unfortunately, the end result was the development a of new variation of the side-follow and director algorithms adapted to the camera's strengths and weaknesses. From a software engineering standpoint, this was undesirable because any changes to the algorithm that applied to both scouts now had to be made in two different locations. An attempt was made to recombine the algorithms into a common source file, differentiated by C++ preprocessor directives, but this was abandoned because it severely impacted the readability of the code. In the future, it would certainly be valuable to redesign the code so that a common source file for both scouts was feasible.

3.3.2: Uncertainty & Error

The distributed manipulation system is subject to many different sources of error: the sensors are inaccurate, the communication is unreliable, and the Saphira software controlling the robot is not robust. It would have been extremely useful to have access to a method of placement planning, such as the motion-planning algorithm of [Lozano-Peréz, Mason, Taylor 1984], that precisely accounted for error in all planning. However, since our task is dynamic and on-line this algorithm and others that rely on pre-existing maps

---

viewing angle of the camera (which results in relatively low area distortions due to viewing angle) the area-

do not apply directly. Instead, we adjusted the system implementation in an attempt to minimize the effects of these errors on performance.

We accounted for sensor inaccuracy by never basing any decision on a single sensor reading. Every piece of sonar data the planning software received was an average of three readings, with unusually high or low distances filtered out of the pre-average data set. The camera information was similarly filtered and averaged. We attempted to counter unreliable communications between the scout robots and the workstations running the planning algorithms by repeating some commands twice in order to make sure they were received. The communication between the scout processes and the mover robots was also unreliable, so the scouting system was designed to repeatedly send commands to the pushers until the desired results were achieved. Finally, we attempted to make the algorithms robust enough to recover if erroneous readings or communications resulted in unexpected behavior. If the scouts lost track of the system due to missed commands or bad sensor data, algorithms were available that enabled them to scan for the system and reliably reposition themselves alongside it. We also countered error by making use of each scout's knowledge of what command the pushers were currently executing. This enabled each scout to better anticipate system movement and avoid possible misinterpretation of sensor data.

---

based distance approximations were more than adequate to my purposes.

4: Experimental Results

4.1: Experimental Environment

We would like to measure the performance of a distributed manipulation system by observing its actions in a wide variety of obstacle configurations and its behavior over long paths. For this we would need a large environment, empty of permanently located obstacles. However, the debugging of the system was a challenging task and took a long time. Little time was left for large-scale experiments, so our tests were confined to the environment of the Robotics Laboratory. Unfortunately, the Robotics Laboratory environment makes thorough testing impossible for a system as large as ours. Because of the limitations of the lab space, the system has only been measured in its ability to avoid two obstacles and straighten out afterwards, and even then the configuration of the room and the presence of a large support pillar near its center severely limited the testing possibilities. One of the obstacles avoided always has to be the pillar, and there is only enough space in the lab to place another obstacle to the pillar's left. Still, even in this limited environment we can test the some of the basic steps of the multiscouting implementation.

The following table summarizes a series of tests that were run in the final two weeks of the project.[5] The multiscouting code was frozen throughout the tests. Each test placed two large boxes in the system's path, separated by enough room for the system to

---

[5] In the future, we hope to find a large, open space, such as a gymnasium, in which more extensive tests could be conducted. In particular, we would like to test a greater variety of obstacle configurations and the system's performance over long paths.

pass through them. In roughly half of the tests, the boxes were set up so that Ben was

responsible for the first avoidance and Jerry for the second. This order was reversed for

the other tests. During each run we recorded if the system avoided the first obstacle, if it

avoided the second obstacle, and if it straightened out successfully. In both cases,

avoidance of the second obstacle also included some straightening-out behavior because

the initial turns always rotated the system such that the second obstacle would not be

detected until the straightening behavior caused the movers and scouts to swing in the

direction opposite to the first turn. The number of attempts at each task is different

because in most cases a failure at an earlier task completely prevented an attempt at a

later task. Figures 4-1 and 4-2 describe the test environment in each case and indicate the

approximate path that a successful run would take in each instance.

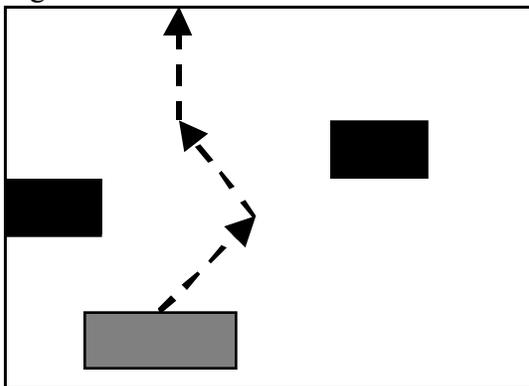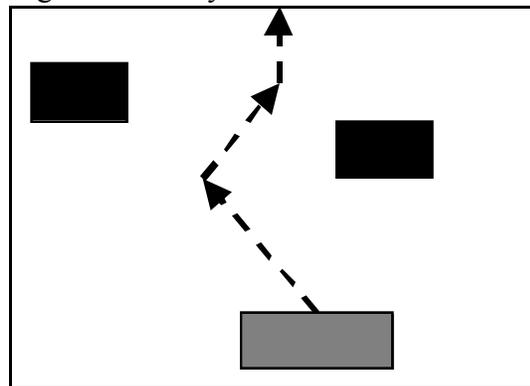Figure 4-1: Ben avoids first   Figure 4-2: Jerry avoids first

All tests were conducted using a simulated couch, constructed by joining three

large computer monitor boxes together with duct tape, as the manipulated object. Both

the movers' and scouts' software is best suited for handling rectangular manipulated

objects. Because sonar readings are most reliable with simple, rectangular objects (and

because our lab contained an abundance of them), large computer boxes and the

rectangular support post were used as obstacles in all tests.

4.2: Experimental Results

Table 4-1: Testing results

Total Runs: 77	1st Avoidance: 53/77 = 69%	2nd Avoidance: 13/53 = 23%	Straightening: 0/13 = 0%
Ben Avoids First: 40	24/40 = 60%	13/24 = 54%	0/13 = 0%
Jerry Avoids First: 37	29/37 = 78%	0/29 = 0%	0/0

The test results reveal that the system is not as reliable as it needs to be for

practical use. Avoidance of the second obstacle in cases in which Jerry controlled the first

avoidance proved especially troublesome, as was straightening out after avoiding the two

obstacles. The algorithm is sound, as demonstrated in Section 2, but the implementation

needs significant work. In order to provide some information about the weaknesses and

strengths of various system components, we kept track of the apparent causes of each

system failure throughout the tests. Of course it is difficult to ascribe the failures of a

group of robots to any one component with complete accuracy, but this at least provides

an estimate of navigational reliability.

The pusher system, which this project sought to leave "as is" as much as

possible, appeared to be responsible for many failures. Although the manipulation

algorithms are very robust, the mover robots are slow to respond to commands and react

very slowly. The pusher implementation also assumes that the manipulated object is a

rigid body. Our simulated couch, constructed of cardboard boxes, flexed and bent considerably at times and the pushers' rigid-body assumptions did not match the system's behavior in these cases. If we subtract out those failures that appeared to be primarily caused by the mover robots, the test results are as follows.

Table 4-2: Mover errors removed

Total Runs: 77	1st Avoidance: 53/61 = 87%	2nd Avoidance: 13/37 = 35%	Straightening: 0/11 = 0%
Ben Avoids First: 40	24/29 = 83%	13/16 = 81%	0/11 = 0%
Jerry Avoids First: 37	29/32 = 91%	0/21 = 0%	0/0

Although this simple subtraction of pusher-failures incorrectly takes away cases in which the scouts would have failed even if the pushers had succeeded, it provides an approximation of the system's potential performance with perfect manipulation. The navigation system's two weaknesses are in the second avoidance, if Jerry ordered the first one, and in straightening out. All of the cases of failure to straighten out were caused because either the system deadlocked unexpectedly, due to an undiscovered coding error (6 cases), or because one or both of the scouts became lost or stuck behind one of the obstacles during the avoidance process. The second avoidance failures not due to mover error were also overwhelmingly due to failures of the side-following behavior.

Ben was especially unsuccessful in directing second turns. Many of the failures were due to the fact that the camera software had difficulty interpreting the results produced by a high rate of rotation on the target. Ben was often well out of position by

the time of the second turn because correctly adjusting to high rotation rates was a necessary skill during the first turn, which he was on the inside of. More sophisticated camera algorithms are necessary to prevent this from occurring.

Many failures resulted from scouts becoming stuck behind obstacles or crashing into obstacles or the manipulation system. Some of these incidents could have been avoided if the experimental setup had allowed for more than the bare minimum of space between the two obstacles. In the future, testing should either use a smaller manipulated object or a larger space to help eliminate this problem. A larger space would also allow us to test the system's long-term performance and see how well it performs in an environment with more than two obstacles.

Scout side-following failures often appeared to be caused by slow responses to environmental changes. The scouts did not adjust to new sonar or camera data in the multiscouting system as rapidly as they had in the single-scout system. This is undoubtedly due to the overhead imposed by the negotiation layer, which resulted in the sensor-processing methods being invoked less frequently than they had been in the single scout implementation. In future implementations, it may be possible to reduce the negotiation overhead by running the processes controlling the scouts on the same machine. This would eliminate the need for using an Ethernet connection for interprocess communication and increase the speed of negotiation. It would require a workstation with multiple serial output ports because each scout requires a separate radio modem. We could also attempt a multithreaded implementation of the system. Each behavior would

exist as a preemptively scheduled thread and receive a more equitable share of processing time than it does in the current, cooperatively scheduled system.

If the system is to improve in its reliability, the mover implementation needs to be made more responsive and better able to handle objects that have some flexibility, the side-following behavior of the scouts must be made more reliable and responsive, and the imbalance in scout effectiveness needs to be addressed.

5: Alternative Algorithms

There are other approaches to distributed sensing for distributed manipulation systems. In the course of developing the negotiation algorithm, two alternate approaches were also investigated. The first is a variation of negotiation that operates without any system element requiring knowledge of the global system angle or the system's original motion vector. Instead of returning to the original vector with a single turn after an avoidance, each scout is responsible for undoing any avoidance movements that resulted from its suggestions. This is called "distributed straightening" in contrast to the straightening out based on global knowledge employed in the implemented version of negotiation. This variation was not implemented because it is less efficient than the original negotiation algorithm, requires a great deal more overhead, and is not well-suited to a fault-tolerant implementation. The other alternative algorithm is bidding. Bidding employs an identical priority scheme as negotiation, but requires the mover robots to perform all the comparisons between scout suggestions. The bidding algorithm was not implemented because time constraints prevented it, but it is worth discussing because it offers some advantages in communications and disadvantages in sensing when compared to negotiation.

5.1: Negotiation With Distributed Straightening

This algorithm is identical to the original negotiation algorithm, except that the scouts do not attempt to straighten out to the original vector in the absence of obstacles. Instead, in the absence of obstacles, a scout attempts to undo the deviation caused by its

previous moves. Each scout maintains a stack of the moves it needs to execute so that the system can return to its original vector. After completing a turn, the scout responsible for it pushes the undo information onto the stack. The burden for straightening out the system is, therefore, distributed – each scout only has local responsibilities, not global ones.

This system has several drawbacks, however, that lead us to decide that giving each scout a knowledge of the global original vector and attempting to return to that vector in one step was the better approach. First, it requires a great deal of overhead. The maintenance of the undo-stack for each scout requires careful monitoring of where each avoidance maneuver begins and ends, and the straightening-out code must account for the possibility that moves may be interrupted unexpectedly by higher priority commands from the partner scout. This required far more extensive modification to the director code than one-step straightening out. Secondly, distributed straightening requires a great deal more physical space than global straightening and can result in a great deal of wasted movement. If one scout makes a 20 degree turn right and his partner follows with a 40 degree turn to the left, distributed straightening requires turning 20 degrees to the left and 40 degrees to the right to succeed. In the implemented algorithm, only one turn of 20 degrees is required to achieve the same objective, utilizing less space and less time.

Most importantly, from the standpoint of robustness, if one of the scouts fails a system based on distributed straightening would be unable to recover to the original vector unless each scout had a full copy of their partner's undo stack, kept precisely

synchronized. This erases any advantages gained from using a distributed system, since all the data needs to be globalized in order to achieve fault-tolerance.

Finally, if someone desired a system that returns to its original path rather than to its original vector, it is much simpler to use the current global deflection from the original path to calculate a return vector. To calculate a return to the original path using distributed straightening, it is necessary to reverse the moves in the opposite order they were performed in, as is demonstrated by the following proof and diagrammed in Figure 5-1.

Figure 5-1: Results of an out-of-order, distributed attempt to return to the path
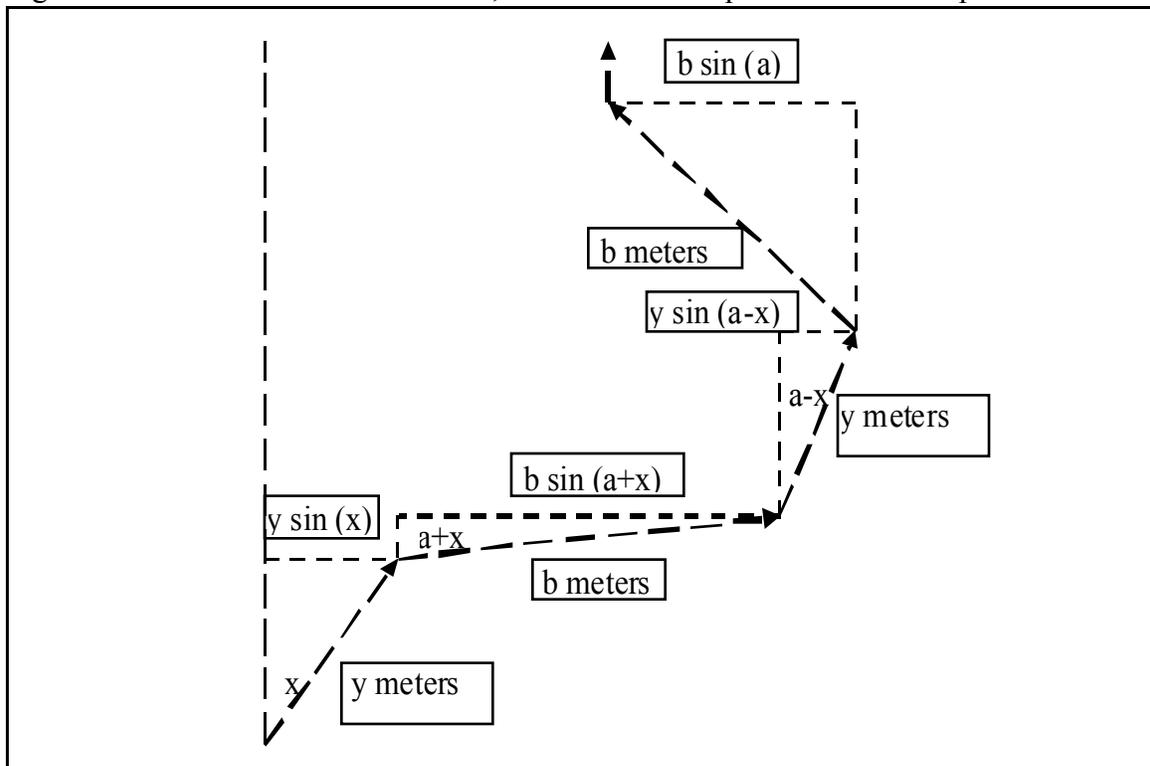
Theorem 5-1: A system based on distributed straightening-out must undo all moves in the opposite order they were executed in to return to its original path.

Proof:

1) Scout A orders a turn. The system moves along a net vector with angle x and magnitude y meters. Assume that the original course is at global angle 0 and positive degrees indicate rightwards turns. The system is now y sin (x) meters to the right of where it started.

2) At this point, Scout B sees an obstacle that also must be avoided, so it orders a turn. The system travels alongside a vector offset a degrees from the current vector (where a > x) and of magnitude b meters. The system is now b sin (a + x) meters to the right of where it was at the end of Step 1, or a total of y sin (x) + b sin (a + x) meters to the right of the path.

3) Now, the system has turned to a+x degrees and each scout has one move to undo.

4) Scout A undoes its move first.

5) So, the system turns (-2x) degrees and then moves y meters. If a > x, however, the system is still traveling to the right while it moves y meters. Then the system turns x degrees and Scout A has undone its most recent move. As a result, we moved at angle (a - x), getting a further y sin (a - x) degrees to the right. The system is now y sin (x) + b sin (a + x) + y sin (a - x) meters off its path.

6) The system now points at a degrees and is further to the right than it was before Scout B executed its undo move. Now, Scout B turns (-2a) degrees (global angle -a), travels b meters, and turns a degrees to come to rest at global angle 0. However, this move was only designed to counter the original rightwards drift produced by the move executed in Step 2. Therefore, we are not on the path. We only moved b sin (a) left.

left-right deviation = y sin (a - x) + b sin (a + x) + y sin (x) - b sin (a).

= y sin (a - x) + y sin (x) + [b sin (a + x) - b sin (a)]

= y sin (a - x) + y sin (x) + [positive value - since (pi/2) > (a + x) > (a)]

-> we are still to the right of the path

This problem can be solved. If we undo moves in the order that they are made, we will not have this problem. If all the undo's are timestamped in the negotiation process, we can structure negotiation so they occur in the following pattern.

<moves>-><straight>

<straight>-><move x><straight><undo x> | <empty>

The key to returning to the path is guaranteeing that all left-right motion cancels out. If we turn x degrees off of a path and move y meters, we will have traveled y sin (x) meters to the right. So long as we can return to this left-right position and orientation, all we need is to turn (-2x) degrees and travel y meters again we will move y sin (x) meters to the left and only need to straighten out to 0 (turn x degrees) to be back on course. Because we do not care where we are along the path, the distance traveled in between move and unmove is irrelevant, so long as we return to the move's path before we run it's unmove.

However, the necessity of timestamping adds to the complexity of any implementation of this already wasteful, inefficient method of achieving a result that can be achieved with a globalized return-to-path method.

Therefore, global straightening-out is preferable to the distributed method with respect to efficiency in time and space, efficiency in communication, and extensibility to a future return-to-path negotiation algorithm.

5.2: Bidding

As described in the introduction, there is an alternative approach to the negotiation algorithm – the bidding method. The bidding method operates in exactly the same way as multiscout negotiation with one major exception. Instead of sending (command, priority) pairs to each other, bidding scouts communicate them to the mover robots directly. Each mover robot stores the most recent command from each scout and acts upon whichever has the highest priority. The scouts receive no response from the pushers as to which bid was successful. We expect the bidding algorithm to have performance similar to the negotiation algorithm. The movers should execute roughly the same moves, since the priority algorithm is unchanged. The only difference is that it is now executed by the movers and not by the scouts.

Bidding trades negotiation's reliance on communications for reliance on sensing. Bidding requires $O(n)$ communications with respect to scouts without requiring any one mover or scout process to be designated as central or using a round-robin communication scheme. Also, bidding should require far less overhead in the scouting code. Because there

is no need to wait for a counter-suggestion from another process, a bidding implementation in our current system would allow scouts to spend far more time on side_follow and director tasks than negotiation does.

Bidding's disadvantage is its reliance on sensors. Because the scouts have no direct knowledge of the results of bidding, they need to derive that information from their sensors. Ideally, the task of following alongside the manipulated object would be completely independent of the task of directing the mover system, but in reality the two must be linked because of the limitations of our hardware. If a scout is on the inside of a turn, for instance, the normal side_follow methods result in scout crashes or the loss of sensor contact with the rest of the system. In order to be successful when implemented with the limited sensor data provided by two sonars, or a sonar and a camera, the side_follow algorithms need to make assumptions about the speed and behavior of manipulated objects, and in all iterations of the distributed manipulation project it has been found necessary to use the knowledge of what command is currently being executed to supplement and, in some cases, override these assumptions.

Deriving knowledge from the sensors is, however, very difficult. Scouts and pushers continually make small adjustments relative to one another and relative to their original vector. Their sensors are not accurate enough and their environment is far too complex to allow for precise movement. Therefore, it is very difficult to interpret sonar data that shows that a particular scout is misaligned with the rest of the manipulation system. A small misalignment could mean that the movers are responding to a turn bid and that the scout should change its motion plans accordingly. It could also mean that the

movers are merely pushing the manipulated object irregularly or that the scout has wandered off-course due to bad sensor or odometry readings. Differentiating between these cases on the current hardware may not be impossible, but it is extremely difficult. It will either require the development of sophisticated filtering algorithms or refinements to the side_follow methods that enable them to function with fewer assumptions about the sorts of motion that a pushed object is likely to undergo. We think that the implementation of the bidding algorithm and work on the refinements necessary to make it reliable are worthwhile tasks for future researchers in this area.

6: Conclusion & Future Directions

Algorithmically, the negotiation algorithm for multiscouting was found to be efficient, robust, and provably correct for all spaces in which each pair of obstacles is separated by enough free space for the system to pass between them. The algorithm was implemented as an extension to an earlier cooperative manipulation system and tested. The test results indicate that the negotiation algorithm requires a more robust implementation of the side-following behavior and sensor interpretation algorithms than is presently available.

An alternative version of the negotiation algorithm that operated without global knowledge of the "straight" system path was analyzed and found to provide inferior performance, robustness, and extensibility. The bidding algorithm was proposed as an alternative algorithm that requires better sensor performance than is currently available in the scouting system.

The theory of negotiation-based multiscouting is far more robust than the actual system proved to be, but the test results do suggest improvements that could lead to reliable performance. First, the scouting system should be fully multithreaded in any future implementation. Each additional layer reduced the responsiveness and effectiveness of previous layers because of the need to explicitly call lower-level methods often enough to produce the correct side-following and directing behaviors, while still allowing enough CPU time for negotiation. If side-following, directing, and negotiating could each be assigned to a preemptively scheduled thread, communicating via shared objects and

semaphores, responsiveness should improve and the code for each level could be greatly simplified.

The unreliable performance of the mover system was a leading cause of system failure. Any future system needs a much more fault-tolerant implementation of the pushing algorithms. The scouting algorithms, particularly side-following, also need much improvement. Although the scheduling problems discussed above are probably a main cause for their failure, the algorithms need to deal much more intelligently with a host of situations, particularly when they are extremely close to obstacles or the manipulated object.

Algorithmic analysis suggests that the negotiation algorithm should be suitable for use in any distributed manipulation system that requires an on-line, real-time navigational component, so long as the set of obstacles is finite and the objects are separated by enough space so that the system can pass through them. If the mover system was extended to allow backtracking, it should be possible to develop a system that can escape from local traps and navigate successfully in all situations that contain potential paths.

An extension that is possible using the existing manipulation system would be a navigational system that returned to its original path, rather than merely to its original vector, once all obstacles had been avoided. Some of the algorithmic details of such a system was discussed in Section 5, as is the reason why the current version of negotiation is better suited to such an extension than one that more fully distributes the task of straightening out the system.

Another interesting approach to future work would be to combine the real-time planning offered by the negotiation system with the path searching algorithms developed in [Alvarado 1998]. These algorithms could be used to search out paths for the system to travel through in spaces that contain regions in which some obstacles are not separated by enough space to allow the system to pass through them. Then, while the system navigates the path mapped out by Alvarado's algorithm, any dynamic changes to the environment that occurred after the initial planning phase could be handled by the multiscout, real-time negotiation algorithm. A system that employed such a hybrid algorithm would be able to handle a class of spaces larger than either algorithm could solve separately.

Two useful generalizations to the negotiation and manipulation system would be one that could handle n scouts, given certain information about the positional relationships between each scout and the rest of the system, and one capable of handling non-rectangular manipulated objects. Before either of these is attempted, however, more work should be done to improve system reliability in scout and mover performance, particularly focusing on improving the methods for interpreting sensor data.

The algorithmic analysis and initial implementation of multiscouting suggest that it is a worthwhile area of research, an important extension of the cooperative manipulation work developed in [Rus, Kotay, Kabir, Soutter 1996], and opens up a number of interesting areas for future work.

References:

[Alvarado 1998] C. Alvarado, "Distributed Route Planning Using Partial Map-Building," in Computer Science Technical Report PCS-TR98-336, Department of Computer Science, Dartmouth College, 1998.

[Brooks, Connell 1987] R. Brooks, J. Connell, "Asynchronous distributed control system for a mobile robot," in *Proceedings of the International Society for Optical Engineering Mobile Robots*, 1987.

[Cai, Fukuda, Arai, Ishihara 1996] A. Cai, T. Fukuda, F. Arai, H. Ishihara, "Cooperative Path Planning and Navigation Based on Distributed Sensing," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[Canny, Reif 1987] J. Canny, J. Reif, "New lower bound techniques for robot motion planning problems," in *IEEE Foundations of Computer Science*, 1987.

[Donald, Jennings, Rus 1993] B. Donald, J. Jennings, D. Rus, "Towards a Theory of Information Invariants for Cooperating Mobile Robots," in *Proceedings of the International Symposium on Robotics Research*, 1993.

[Donald, Jennings, Rus 1994a] B. Donald, J. Jennings, D. Rus, "Information invariants for distributed manipulation," in *The First Workshop on the Algorithmic Foundations of Robotics*, eds. K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, pages 431-459, 1994.

[Donald, Jennings, Rus 1994b] B. Donald, J. Jennings, D. Rus, "Analyzing teams of cooperating robots," in *Proceedings of the International Conference on Robotics and Automation*, 1994.

[Donald, Jennings, Rus 1996] B. Donald, J. Jennings, D. Rus, "Information Invariants for Distributed Manipulation," in *International Journal of Robotics Research*, 1996.

[Garcia-Molina 1982] H. Garcia-Molina, "Elections in a Distributed Computing System," in *IEEE Transactions on Computers*, January 1982.

[Guibas, Latombe, LaValle, Lin, Motwani 1997] L. Guibas, J. Latombe, S. LaValle, D. Lin, R. Motwani, "Visibility-Based Pursuit-Evasion in a Polygonal Environment," in *Algorithms and Data Structures*, 1997.

[Harmon, Gage, Aviles, Bianchini 1984] S. Harmon, D. Gage, W. Aviles, G. Bianchini, "Coordination of Intelligent Subsystems in Complex Robots," in *Proceedings of the First Conference on Artificial Intelligence Applications*, 1984.

[Jennings, Whelan, Evans 1997] J. Jennings, G. Whelan, W. Evans, "Cooperative Search and Rescue with a Team of Mobile Robots," in *Proceedings of the International Conference on Advanced Robotics*, 1997.

[Lozano-Peréz, Mason, Taylor 1984] T. Lozano-Peréz, M. Mason, R. Taylor, "Automatic synthesis of fine-motion strategies for robots," in *International Journal of Robotics Research*, 1984.

[Mataric 1995] M. Mataric, "Designing and Understanding Adaptive Group Behavior," in *Adaptive Behavior*, December 1995.

[Parker 1992] L. Parker, "Local versus Global Control Laws for Cooperative Agent Teams," A.I. Memo No. 1357, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, 1992.

[Reif 1979] "Complexity of the mover's problem and generalizations," in *Proceedings of the 20th IEEE Symposium on Foundations of Computer Science*, 1979

[Rus 1993] D. Rus, "Coordinated rotations of polygons," in *Proceedings of the Symposium on Intelligent Robot Systems*, 1993.

[Rus, Donald, Jennings 1993] D. Rus, B. Donald, J. Jennings, "Moving furniture with mobile robots," in *Proceedings of the Symposium on Intelligent Robot Systems*, 1993.

[Rus, Kabir, Kotay, Soutter 1996] D. Rus, A. Kabir, K. Kotay, M. Soutter, "Guiding Distributed Manipulation with Mobile Sensors," in *Proceedings of the IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems Proceedings*, 1996.

[Sawasaki, Inoue 1996] N. Sawasaki, H. Inoue, "Cooperative Manipulation by Autonomous Intelligent Robots," in *JSME International Journal*, 1996.

[Shibata, Ohkawa, Tanie 1996] T. Shibata, K. Ohkawa, K. Tanie, "Spontaneous Behavior of Robots for Cooperation - Emotionally Intelligent Robot System," in *Proceedings of the IEEE International Conference on Robotics and Automation*, 1996.

[Wright, Sargent, Witty, Brown 1996] A. Wright, R. Sargent, C. Witty, J. Brown, *Cognachrome Vision System User's Guide*, Edition 2.0, Newton Research Labs: 1996.