6-5-1998

Distributed Route Planning Using Partial Map Building

Christine J. Alvarado
Dartmouth College

Distributed Route Planning Using Partial Map-Building

Christine Alvarado
Senior Honors Thesis (Advisor: Daniela Rus)
Dartmouth College
Computer Science Technical Report PCS-TR98-336
June 5, 1998

Abstract:

Our goal is to manipulate and guide an object across an unknown environment toward a goal in a known location in space. Our tools include a system of manipulation robots, which are "blind" and one mobile scout robot who relies on a series of sonar sensors for information about the environment. Previous solutions to this problem have taken a simultaneous guiding and manipulating approach, moving the whole system under the scout's guidance. My approach, however, presents a separate scouting algorithm that can return a series of coordinates through which the manipulation system can safely pass to reach the goal in a static environment. This new approach produces more optimal paths to the goal, as well as evading the concern of what actions to take should the entire system reach a dead end. In this paper I will present both the algorithm and the experimental results I obtained when I built the scouting system.

1 Introduction

The work presented in this thesis is based on the desire to solve the moving problem with a distributed team of robots. We wish to be able to move an inanimate object across a field of obstacles, and we would like to do this work on-line, that is, without a prior map of the obstacle field. As motivation, one can imagine trying to position exploratory equipment in a novel environment, for example, underwater, or on other planets. As in the aforementioned paper, in this case we would also like the robots' behavior to mimic the coordinated series of guiding and following behaviors that human manipulators would employ. The only difference is that the task of scouting has been separated from the task of moving the objects. Scouting can be described as using mobile sensors for distributed object placement. This thesis focuses on the details of getting a robot scout to perform the necessary steps in seeking out and returning a safe path from a starting point to the goal point, if such a path exists.

We have separated the problem into the separate tasks of (1) scouting, and (2) manipulation. We were motivated by (1) maximizing usefulness of our heterogeneous system of robots and (2) minimizing the errors which occur during changes in our manipulation system. Our team of robots consists of two RWI B14 robots and an RWI Pioneer robot. The two B14 robots are adept at pushing large objects, but have no sensory information about the world in front of them. In this sense they are totally blind, relying one hundred percent on the external motion commands. The Pioneer scout, on the other hand, is equipped with eight sonar sensors, seven with fixed direction, and one rotating sensor. Five of the fixed sensors are positioned at varying angles in front of him, one faces directly to the scout's left, and the last faces directly to the scout's right. The rotating sonar is positioned so that at the zero angle it faces directly to the scout's left, and can rotate approximately forty-five degrees to the scouts rear, and forty-five degrees to the scout's front. Thus, although the scout does have a few blind spots, it is designed to detect a large area in a 180-degree arc in front of it. When we consider the features of the robots, we see

neither type of robot is capable of performing the entire task by itself. The B14 robots must be restricted to object manipulation, while the Pioneer is restricted to scouting.

A key issue that arises is how the scout and the manipulators interact. In our first system, the scout followed along side the object being manipulated (in this case, a box), both scouting and directing simultaneously. [RKKS96] This set up was efficient, as long as no dead ends were encountered, since the system could move toward the goal as a whole. When obstacles were encountered, the scout would simply direct the manipulators using the commands *turn right*, *turn left*, *move straight*, and *stop*.

However, there were several problems with the set-up. First, it was difficult for the scout to stay positioned along the side of the box using only sonar sensors while the whole system was moving. Because sonar sensors have a low angle tolerance threshold, when the system began to turn the scout would often lose its sonar reading and have to stop the forward progress of the system to relocate the side of the box. In addition, when the system made right hand turns and the box was pushed into the scout, the scout would often collide with the box. Second, because the scout had to follow along the right side of the box, it simply could not detect objects in front of the system which were far enough on the left hand side. This problem could not be remedied without moving the scout away from its position on the right side of the box. But the scout's position was a constraint to that system and could not be changed. Third, if the system ever did run into a dead end situation, the manipulators were all but incapable of backtracking.

To solve these problems, I chose to separate the scouting from the manipulation. That is, the Pioneer robot acted as a mobile sensor in the system, exploring the environment to find a path while the B14 manipulation robots remained stationery. When the scout searches for a path on its own, it no longer has to keep track of the rest of the system, it can view the environment from any position, and it can backtrack relatively easily. My challenge was to construct an algorithm that exploited these new freedoms. The scout needed to explore the uncharted world enough to return a path that the manipulators could

safely follow blindly to reach the goal. There exist many off-line path-planning algorithms for similar tasks. If one can determine the visibility graph of an environment for a mobile object, one can easily compute the shortest safe path from the start to the goal. However, because we would like to be able to do this work on-line, most of the off-line path planning algorithms would not apply to our situation. At first glance, then, an existing on-line algorithm may appear reasonable. However, simply using the right hand rule to maneuver around obstacles, while it would indeed succeed in getting the scout to the goal, would not necessarily result in optimal performance. First, the scout may in the worst case have to circumnavigate every obstacle to determine the optimal path from start to goal. Second, complications arise when we consider that the scout must take into account the dimensions of the object it is guiding. Any path which it chooses must also be wide enough at all points for the team of manipulators and the object.

My algorithm combines an off-line algorithm for finding the shortest path to a goal with an on-line exploration and mapping of an unknown environment. An on-line algorithm refers to the algorithm which must be used in a situation where the system has no a priori knowledge of its environment; to use an off-line algorithm, the system must have a complete map of its environment which it can explore before it makes its first move. For an off-line visibility graph algorithm to be successful, we must know two things about the environment and system: a map of the obstacles in the environment (with obstacles usually represented by their edges), and the dimensions of the object which must traverse the environment.

The dimensions of the object are known, so we are simply missing the environment information. If we can use the scout to locate the corners of all the obstacles that exist between the object and the goal, we could easily determine an optimal path from start to goal for the object. However, mapping an entire environment is a difficult and time-consuming process. In the best case, we only need to map the corners along the path to the

goal, but unfortunately, we cannot know where these corners lie. [1] But now we have at least reduced the problem from mapping the whole environment, to possibly finding a limited number of corners, so long as a path exists from start to goal.

A high level description of the algorithm is as follows:

1 The scout does a sonar sweep of the area in front of him to locate the corners of the obstacles which are in range.

2 The scout chooses the corner which minimizes its estimated cost to the goal.

3 The scout proceeds to that corner, adds it to its stored map of the space, and repeats steps 1-3.

4 If the scout reaches the goal, it returns the path it took to get there, adjusted for the size of the object that it guides; if it cannot reach the goal, it returns failure.

This way, the scout explores the corners in an organized way, so as to explore the fewest number while finding an optimal path to the goal. My contributions to the system as a whole include fully developing and implementing this novel approach, establishing efficiency bounds on the algorithm and proving its usefulness through several tests runs.

The outline for this paper will be as follows. First I will discuss several works related to this problem. Then I will discuss in detail my own algorithm and the theory related to it. Next I will describe the restrictions introduced in the physical system and I will note how the experimental system differs from the theoretical system. Finally I will discuss experimental results and discuss extensions to both the experimental and theoretical systems.

2 Related Work

The work in this thesis has been inspired primarily by previous work by Rus, discussed in the introduction to this paper [RKKS96]. Both work on the previous system

---

[1] The *estimated path cost* can be described for each corner in the environment as the length of the distance traveled to reach that corner, plus the straight line distance from that corner to the goal.

[RKKS96], as well as the work presented here has been inspired by related work done in distributed manipulation, on-line navigation algorithms, and map-making algorithms.

2.1 Distributed Robotics

In the area of distributed robotics, important work on an initial solution to the problem presented in this paper is presented in [RKKS96]. Another distributed manipulation algorithm in the form of a search and rescue algorithm is presented by Jennings, Whelan, and Evans. [JWE97] In their search, a team of robots fans out in several directions to attempt to locate a lost object. When one robot finds the object, it notifies the others and they all converge to manipulate the object towards the goal. This fanned out search is embodied in my algorithm as an initial sonar sweep. The best point from the sweep is chosen and all efforts converge on this point as the scout moves to it. From there the sweep repeats.

Another approach to distributed robotics can be found in [KH98]. Kurazume and Hirose examine ways to use teams of robots to solve the positioning problem in indoor environments.

2.2 On-line Navigation

Another class of algorithms is interested on-line exploration and navigation. Hoffmann, Icking, Klein and Kriegel describe a competitive on-line strategy for determining a watchman tour through a polygon. [HIKK] Important work done by Deng, Kameda, and Papadimitriou explores a competitive on-line algorithm for exploring an unknown environment. [DKP91] While these works did not prove to be directly applicable to my own, they were indispensable in that they establish a method a relating the efficiency of an on-line algorithm to the corresponding off-line solution to the same problem. This "competitive ratio" strategy will come in useful in arguing that the path found from start to goal is related by a constant value to the shortest path an off-line algorithm could have

found. Any on-line algorithm, in order to be considered noteworthy, must have this constant relation to the existing off-line algorithm.

Other aspects of on-line navigation are also important in the work I will present and related work has been done by others. Jean Claude Latombe examines solutions for Nonholonomic motion planning [BL93][ALMR97], which can be related to the work presented here because the team of pushing robots form a Nonholonomic, system of manipulators; that is, the manipulation robots are not connected an must coordinated their pushing so as to move the object they are manipulating. Latombe also explores navigation under visibility constraints. [GGLLLMT97] This field is important to my research since I am developing a method of separating the visibility of a mobile system from its manipulation. Finally, work by Jon Kleinberg explores in depth the localization problem for mobile robots. [K94]

2.3 Map Making

My work is involved with making a limited map of the environment so as to determine a path to the goal, so the final area which we explore in this section will be map making algorithms. Work done by Sebastian Thrun looks at different ways to map indoor environments using mobile robots. [BCFHLSST98],[T98] Work done by Choset and Burdick on mapping using the Generalized Voronoi Graph (GVG) [CB95a,b] provides a way to determine a path through the obstacles in a two-dimensional space. The path returned by this algorithm would certainly be useful in our work. Since the path returned by this method is equidistant from all obstacles it is passing between, it would appear that we could simply examine this graph to find a path to the goal wide enough for the object which we are guiding. This scheme may indeed work; however, I chose not to use it because of the time-consuming nature of computing the GVG. As we noted earlier, it is

not necessary that we plot a graph of the whole space. We save time by optimizing the order in which we explore the space.

3 The Scouting Method and its Analysis

3.1 The algorithm

To begin, I will give the reader a general feel for how the scout determines the best path to the goal. I will examine each of these steps in more detail in later sections.

The scout executes the following algorithm in planning a path to the goal for the pushers to follow:

```
While (not at goal) and (searchable corners exist)
    Sonar sweep to find corners in sight limit
    Adjust points detected to account for the object size
    If corner not already explored
        Add corner to partial map of space
    Repeat
        Choose most optimal point to explore
        Proceed to chosen point
        If not reached safely
                Delete point from tree
    Until point reached safely
```

This algorithm is recursive. At each point the scout scans the area in front of it, localizes the corners of the objects within a given range, finds the most optimal corner, heads to it and repeats the process. If the scout encounters problems on its way, it abandons its current path of exploration and tries another. For a graphical representation of the algorithm, see figure 1. The algorithm is discussed in more detail in section 3.3.

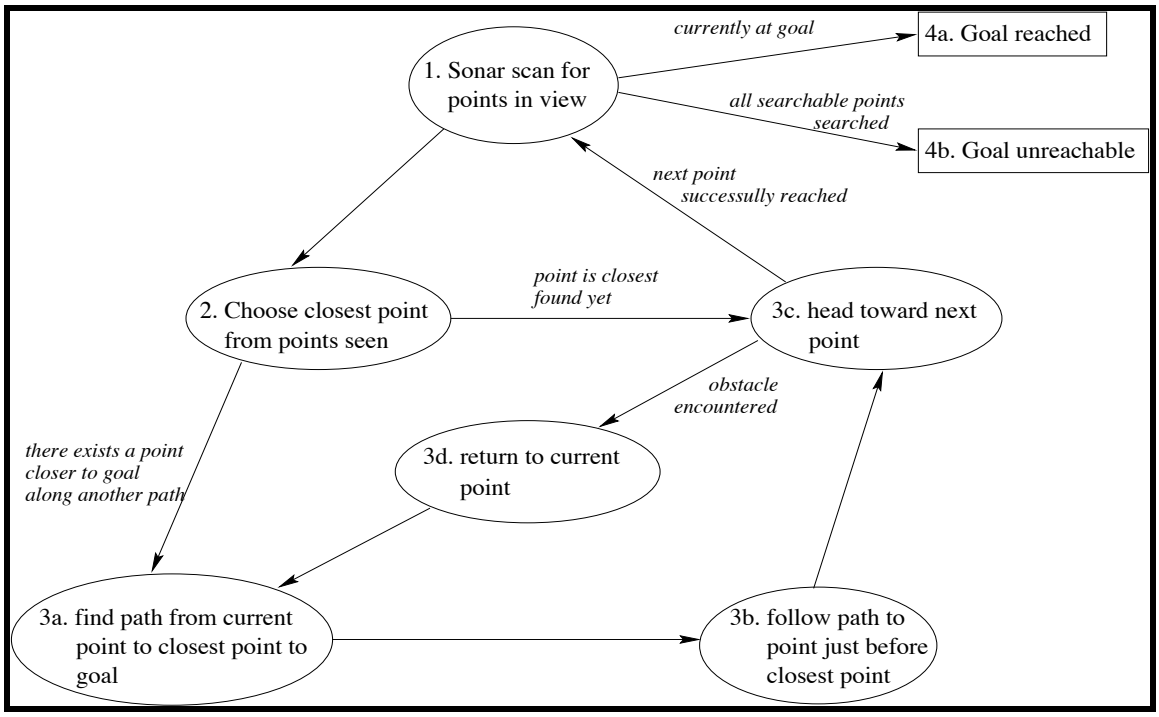


Figure 1: *Graphical representation of the steps of the algorithm. Progress proceeds along arrows. Conditions for following arrows are given in italics next to each path.*

3.2 Motivation

One of the most important parts of any algorithm is that it be logically designed and more suitable to a specific task than any other algorithm that exists. I argue that my algorithm is well suited to the search task it was designed for, and I will attempt to argue this point by justifying some of the points whose presence may not immediately be obvious.

One argument that could be made is that this algorithm is overly complicated. Why not just have the scout head toward the goal while avoiding obstacles? The combinations of these two restrictions should provide a smooth "pull" toward the goal, while leading the scout down a safe path. It is not really necessary to stop and sweep at every point along the way.

This point can be addressed by observing that this setup provides necessary checkpoints that facilitate the construction of the tree and simple path to the goal. Consider a scout whose only task was to head on a safe path to the goal. Simply reaching the goal would mean that the scout had indeed found such a path as long as the scout had verified the path as it found it. However, how, then would the scout convey information about this path to the manipulation system? Any curves would have to be represented by complicated equations, and the scout would indubitably find curves if it weren't specifically keeping track of points. In addition, if the scout ever got stuck and had to backtrack, it would not have a clear reference point to which it could return. The current system not only provides a clear path to the goal through a series of concrete points, but it is conceptually simple, for all practical purposes simply building a search tree physically across the search space.

A further advantage to getting a rough location of corners before moving to them is that it provides a level of optimization before the scout moves. And since the scout knows approximately where it is trying to go, it can take a straight-line course to that point. Since the goal is always to move to the point which optimizes the estimated path length to the goal, the next point can be chosen easily before it is explored. This process of looking one step ahead simulates the off-line visibility graph approach instead of making the scout blindly explore the area in front of it.

3.3 Scouting in detail

Since designing an algorithm for any real system is quite complicated, we must now examine the algorithm in more detail. Through a more detailed examination of the algorithm, I will give the reader an image of the method which the scout uses to navigate through and search its environment.

Following the algorithm presented above, I have presented here a detailed description of the major steps of one iteration through the algorithm:

1. *The sweep*: The scout turns to face the goal, and then does a sonar sweep of the area in front of it. It records the distances and angles where readings are less than its sight limit, thus determining the points in which contain an obstacle. This sweep can be through any angle; increasing the angle swept by the sonar increases the possibility that the scout will find a path to the goal.
2. *Corner localization and adjustment for object size*: Next the scout must determine which points can be considered corners of obstacles. We do this simply by making sure the point's Euclidean distance from at least one of its neighbors is above some threshold. Since the scout must take into account that it is guiding an object that has some known dimension, this threshold should be the dimension of the object. Eliminating paths that are too narrow at this point will in effect prune unsuitable branches from the tree before they are expanded. Clearly, because the sonar sweep does not give totally accurate information about the environment, some paths that are thought to be acceptable at this point, may turn out later to be unsuitable.
3. *Tree construction*: The scout loads into the search tree the points which were found to be acceptable corners to explore as children of the current node.
4. *Exploration choice*: The scout chooses the next point in the tree to explore. It does this by optimizing the direction and area it chooses to explore. If there is a straight path to the goal (which it detects during the sweep step), the scout heads straight toward the goal until objects are detected at some threshold distance in front of the scout. Otherwise the scout chooses the point in the tree which would minimize the path length to the goal (i.e. the distance of the next point to from the goal plus the path length traveled to reach the point). To locate the point, the scout performs one of the following actions:

If the point is a child of the current node: the scout turns in the direction of the point, and heads toward it.

If the point is in a different branch of the tree: the scout must navigate back through the tree to reach the next point. In this case, the scout chooses the path it must traverse by examining the child lists of the current point and the point it is trying to reach, and then navigates the path it finds.

5. *Detection of unsuitable paths*: Because sonar readings are not always accurate, the scout must be aware that the path it has chosen may not be wide enough for the object it is guiding. While heading to the next point, the scout may find an obstacle it did not detect earlier which blocks this section of the path. When it finds such unseen obstacles, it deletes the point it was trying to reach from the tree, returns to the point it had been exploring from, and repeats step 4.
6. *Advancement in the tree*: If the scout reaches the next point successfully, the point becomes the current point from which to explore and it repeats steps 1 through 5 until it reaches the goal. If it has explored each point in the tree and has not detected any new points then the search has failed.

3.4 The storage structures

The method that the scout uses to find a path to the goal is a recursive one, which involves both building and searching a tree that represents the environment at once. To fully discuss how the scout finds and returns the path to the goal, we must first examine the data structures the scout uses to store the information about the environment around it.

The main data structure is a tree, with each node holding an x and y position, relative to the start, the point (0,0). The nodes also hold several other facts about the environment, such as the goal position (and thus the distance to the goal), the current path length, and the point through which the object would pass to safely reach the goal, and the most important element: the node's array of children. Each node contains an array of child nodes in the path tree. This array contains the points that can be seen, hence explored, from the current node. Through this representation, the path tree becomes a schematic map of the

environment, as well as a search tree that gets built as each point is expanded. As the algorithm searches forward, it builds nodes onto the tree. When it finally reaches the goal, all it must do is return the path from start to goal through the tree, and this will be the safe path from start to goal. To facilitate traversal of the tree (since each node only has pointer to its children, not to its parents), we store in each node a list of integers, the *child list*, which represents the child path that can be traversed from the starting node to reach that particular node. For example, consider a child list (1, 3, 2). The first node in the path to this node is the starting node, the next is the second child of the start node (because the list starts at 0), the next is that node's forth child, and the final node, would be that node's third child.

It is interesting to examine the setup of this path tree. Because we construct the path tree by sweeping from left right, the children each node are arranged in a spatial manor, with few exceptions. That is, the array of any node's children not only represents which corners can be seen from its position, but it also represents these corners in the order in which they can be seen from left to right in space.

Now that we have a useful data structure that will virtually construct the path from start to finish for us, we must find an efficient was of loading this data structure so that the path we end up with is the most efficient path we can determine.

3.5 Assumptions

While ideally we would like for an algorithm to work under any conditions in any environment, when one is working with a real-time system in an unknown environment, obviously this idea is not a realistic possibility. While I tried to keep the conditions as unrestricted as possible, I have placed a few restrictions on the environment.

My first assumption is that the obstacles are convex or concave polygon objects. The obstacles must all have flat faces which, must be just large enough to have distinct corners, and in practice, must be at least as long as the scout, so it can position itself along such an

edge. The objects must all have corners due to the dependence of the algorithm on locating corners to build the map to the goal. The length restriction arises from the scout's use of sonar sensors determine corners by positioning itself parallel to an edge of the obstacle, which will be discussed when I discuss the implementation of the system. In theory, the algorithm could work for round objects by approximating them with polygons as long as the radius of curvature was large enough. But because we would use this approximation, for the purposes of our discussion of the algorithm and its verification, we shall assume that we only have flat-sided obstacles.

Our second assumption is on the sensory system. Sonar sensors both restrict the system as well as offer advantages, differing from a camera in several ways. For example, they can provide important depth information a camera cannot, but they cannot recognize objects. Perhaps the greatest restrictions a sonar sensor inflicts are angle sensitivity, and range. To illustrate this point, let's consider the things that can occur when we try to use a sonar sensor to locate an object in space. If the object is too far away, the reading will be a large number, but not one large enough to represent the distance of the obstacle. If the object is just a little closer, we get a reading which may be correct, but which is indistinguishable from the incorrect readings. Thus there exists some threshold distance at which our readings become garbage. We will call this distance the scout's *sight limit* and only consider readings smaller than this threshold. Any other readings can be considered to be too far away to be relevant. However, this assumption can also be problematic when we introduce the sonar's angle restrictions. Let's consider what happens when the object we are trying to locate is within the sight limit. If the angle from which the sonar is taking a reading is within some critical range, the sonar sensor will return a correct reading. However, if the sensor is at too great an angle to the face, the sensor will return a reading that is larger than the sight threshold. Thus, we cannot always be guaranteed that when we get a large reading that there is really nothing there, and we must make sure we account for this in our algorithm, to avoid potential problems.

Our third assumption is a restriction on the size of the object which the scout can guide following this algorithm. Because the scout must be able to detect a corridor which is too small for the object to pass, the object which the scout guides can be at most as long as the scout's sight limit. To find the path, the scout will often trace along the edges of obstacles while checking out of its other side sonar that there is enough room for object it is guiding. If the object were longer than the scout's sight limit, the scout would have to move off its path to make sure the path was wide enough. This deviation from the scout's exploration procedure would complicate the process considerably.

Finally, our fourth assumption is on the size of the obstacles in the environment. While I will not restrict the size of the environment, I will require that the obstacles themselves must be finite. In other words, the scout will never encounter an obstacle which it cannot navigate around. This restriction, however, turns out to be a bit arbitrary, as we can construct a series of obstacles in such a way as to lead the scout infinitely away from the goal. This distinction will be discussed when we verify the correctness and completeness of the algorithm.

3.5 Correctness and competitive measures

In this section I will prove several properties about my algorithm, including its termination, correctness, its running time and a competitive ratio for the length of the path which it returns.

3.5.1 Termination

To prove that the algorithm terminates, we must break up the possible cases and prove that each of these cases will lead to termination

Theorem 1: The search algorithm will terminate in a space containing finite obstacles if restrictions are set on the angle of the search or on the distance away from the goal the scout may search.

Proof:

Case 1: A path to the goal is found

This is a trivial case in which the scout simply needs to recognize the fact that it has reached the goal point, and halt.

Case 2: No path to the goal is found

This case is considerably more complicated, and involves the scout realizing that it has exhausted all possible paths to the goal and quitting. We examine this case by breaking the problem into two possible alternative algorithms that will affect the halting power of the algorithm.

Before we can examine the algorithm itself, we must recall the restrictions on the environment and the obstacles. For the purpose of the proof we will not assume any boundaries on the environment or on the number of obstacles. We will however, assume that all obstacles are finite for now. We also assume that all obstacles have flat edges, and thus have corners, since the algorithm is based on these corners.

First, let's consider an algorithm that behaves as described in section 3.3, but, when sweeping for points, sweeps out an angle of just less than 90 degrees, 45 degrees on either side of the angle from the current position to the goal. This will restrict the scout to search always "towards" the goal.

First we shall prove a couple of lemmas which will be useful in proving this case.

Lemma 1: In a sweep which is less than 45 degrees on either side of the goal angle, if the scout is further than its sight limit from the goal, each point found which is a possible next point is closer to the goal than the scout's current point. If the scout is within its

sight limit from the goal, then each point detected will be less than the sight limit away from the goal.

Proof:

Draw a line from the goal (G) to the current point (C), and trace out the circle formed using G as the center and GC as the radius. (See figure 2) Then, if GC is greater than the sight limit for the scout, the line which has an angle of 45 degrees with the line GC (on either side) is at least sight limit times the square root of two long. This distance is too far for the scout to see, thus any line that has an angle of 45 degrees or less with the line GC, and whose length is less than the sight limit of the scout will fall within the circle. Therefore, any point chosen must be closer to the goal than the current point.

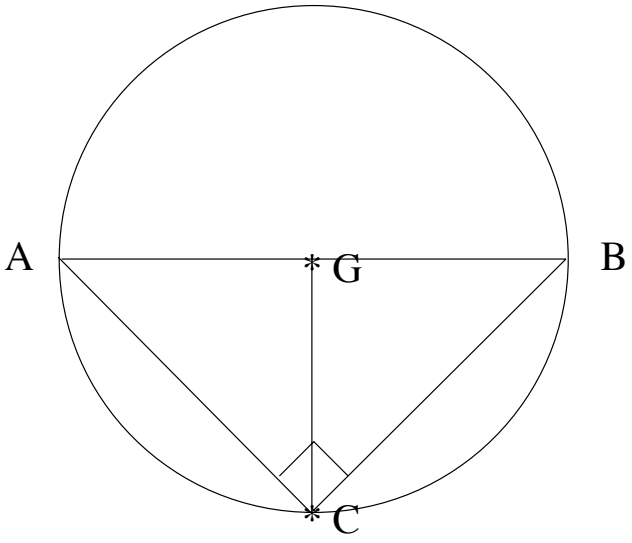
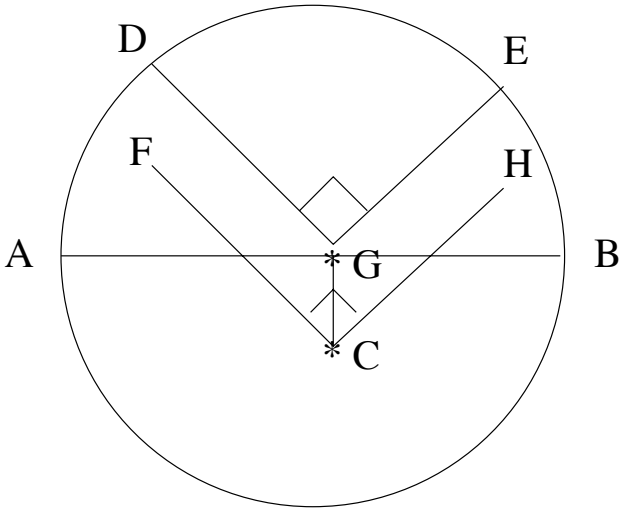


Figure 2: *Point C represents the scout's current position. Point G represents the goal position. Since the scout can only detect points within the triangle ACB, it is clear that any point the scout detects will be closer to the goal than the scout's current position.*

If the scout is closer than the sight limit from the goal, consider the circle centered at G whose radius is the sight limit of the scout. Assume the scout is at any point within this circle. Recall that the scout must face toward the goal, and may only search in a 90-degree angle. It can easily be seen in figure 3 that the scout cannot see any point outside the circle. Therefore all points which the scout detects must be less than the sight limit away from the goal.



|GD| = sightlimit
|GC| < sightlimit
|CF| = |GD| = sightlimit

Figure 3: *In this diagram, the set of lines FCH exactly matches the set of lines DGE. Since the lengths of DG and EG are equal to the radius of the circle (thus, the sight limit of the scout), when they are moved back to point C, they will not reach the edge of the circle. Therefore, the scout cannot see any point that is further than its sight limit away from the goal.*

From the initial restriction and Lemma 1 we obtain the following lemma:

Lemma 2: In the condition where the scout sweeps out a search space of 90 degrees, there exists a finite number of corners which the scout will explore on its way to the goal.

Proof:

By lemma 1 each new point the scout finds will be either closer than the point before, or within a distance of the sight limit from the goal. So choose the initial

distance from the goal, or the sight limit of the scout, whichever is larger, as the radius of a circle with center at the goal. This circle partitions off a finite space which the scout will explore. Therefore, within this finite space there must exist a finite number of corners.

Proof of the 90-degree case is now easily seen. Since the scout does not explore each point more than once, the scout will explore each point in a finite list, and then halt when it has searched them all.

Unfortunately, extending the proof to the general case fails. It is not the case that the scout will always search towards the goal. And, since we have put no restrictions on the environment, the scout could therefore end up searching endlessly away from the goal. Thus, in the general case, when there is no path to the goal, the only way to guarantee that the algorithm will terminate is to restrict the distance away from the goal that the scout may search. Once this distance is bounded, the algorithm becomes much like the case presented previously: a finite search space with a finite number of points to be searched. By applying this restriction and lemma 2, we have proven theorem 1.

3.5.2 Completeness

Obviously it is not enough that the algorithm terminate. The algorithm must also find a path to the goal if one exists.

Again the two cases must be considered separately. As we saw above, only the 90-degree case guarantees that the algorithm will terminate in its pure form. However, this termination comes with a price: the 90-degree case is less powerful in path finding than the unrestricted case. Thus, for the 90-degree case we can only prove a restricted version of path finding to be true, and this restriction reduces the possibilities for obstacle arrangements considerably. We must assume that all obstacles, when adjusted for the size of the object to be guided, are seen as convex. Otherwise we could easily conceive of a

situation where the scout enters into a "cave" and is trapped because of the fact that it cannot explore points which are further away from the current point.

Another thing we must note is when we say that the scout has “found” a point, we mean that the scout has found a path to that point, since the definition of the algorithm restricts the scout to exploring only along the actual path which it will return. So, to prove that the scout find a path, we simply need to prove that it can find the goal, and its route is assumed to be along a valid path. In other words, we will not worry about the length of the path which it finds at this moment.

Theorem 2: In the 90 degree sweep case with strictly convex obstacles, the scout will find a path to the goal if and only if a path exists which always "flows" towards the goal, that is, each point on the path is closer to the goal than the point before it was.

Proof:

By lemma 1 the scout finds points which are each closer to the goal than that current point when the scout is more than its sight distance away from the goal. So, clearly it is the case that the scout will not find the path if a path does not exist or if that path leads away from the goal. We must only prove that if such a path exists, the scout will find it.

We know that a path to the goal exists. Call this path P, with points p_0, p_1, p_2,..., p_g, where p_0 is the start point, p_g is the goal point and the rest of the points are listed in order. Let P be any arbitrary path that may exist to the goal from p_0 (since the idea is just that the scout finds a path, one is just as good as another). Now we can prove by induction that the scout will find each of the points in the path in the correct order. The base case is one in which the path consists only of two points: p_0 and p_g. In this case, there are no obstacles between the scout and the goal and the scout will head toward the goal until it eventually reaches it and halts.

Now consider the case where there is just one obstacle between the scout and the goal. The fact that the scout must find the point p_1 can easily by seen by contradiction.

Say the scout does not find p_1. Then it must have found the goal, because if it hadn't it would only have halted by searching each point in the finite space, thus finding p_1. But it could not have found a path to the goal, because in this case the shortest existing path must go through p_1. Then, by applying the base case, the scout finds p_g from p_1. For the general case, assume the scout will find a path of length up to n, and prove that the scout can find a path of length n+1. In the case where the shortest path has n+1 steps, we know that the scout will find the path from p_1 to the goal by the inductive assumption. Now we just need to prove that the scout finds p_1 before it finds any of the other points. This can easily be seen to be the case, since if the scout found any other point first, it would then find a path from that point to the goal (because length < n), and this would violate the fact that the shortest path goes through p_1. Therefore, it must find p_1 before it finds p_2 (although of course it may find other points between p_0 and p_1).

Generalizing the algorithm to the 360 degree case we may remove the restrictions on the environment (i.e. for convex or concave obstacles) to obtain the following:

Theorem 3: In the 360-degree case, the algorithm will always find a path if one exists.

Proof:

Since we have no restrictions on the area of the space in which we are searching, we cannot at any point guarantee that we will find a point simply because we have done an exhaustive search of the area, as we did in the 90 degree case.

Lemma 3: The 360-degree algorithm will find all the points on a path to the goal that are along the visibility graph if such a path exists. Furthermore, the algorithm will find these points in order, thus finding the path to the goal.

Proof:

To prove this lemma we must do two things: show that the scout will find each point on the path, and show that it finds these points in the correct order, i.e., the order in which they occur along the path to the goal.

We can prove the first part with an inductive reasoning must like we used in the above case. If there are no obstacles, the scout clearly finds the goal. As we did above, assume that the scout can find paths of length n, and prove the scout can find a path of length n+1. We know that, once we make it to p_1, the scout will find a path from p_1 to p_g, since this is a path to the goal of length n. So what we need to prove is that the scout will find and explore p_1. However, unlike the 90-degree case, we cannot assume that the scout will find p_1 through an exhaustive search of the area, since the area in which the scout searches is no longer bounded.

To begin, we must first show that the algorithm will detect point p_1. If p_1 is close to the scout it will be seen directly. If however, it is out of the scout's sight, we have two possibilities for what happens next. One, the scout has not seen any obstacles blocking its path to the goal, and it moves toward the goal and toward detecting point p_1. Two, it does not detect point p_1, but instead moves toward another detected obstacle. If this is the case, this other detected obstacle must form a wall, which eventually leads the scout back to point p_1. If these obstacles did not form a wall, there would exist a more direct path to the goal, which we are assuming does not exist.

Now that we have seen that the scout will find p_1, we can show simply that the scout will explore it. If the scout is not exploring p_1, it must be exploring some other point closer to the goal. There are a finite number of points closer than p_1 to the goal, so once all those points are explored, p_1 will be explored.

Going back to the theorem, if a path to the goal exists, it will exist along the visibility graph, and thus by lemma 3 the scout will find such a path.

3.5.3 Search time

The time the scout takes to find a path is widely variable, and depends mostly on the speed at which the scout can go from point to point. In the worst case, the scout would bounce back and forth exploring several paths. While it will not explore a point more than once, if a series of paths flip-flop distances from the goal as each new point is expanded, the scout will go back and forth between the paths, causing the scout to move a lot for the amount of exploration it does. While I will not go into detail in this analysis, some of the more difficult geographies are shown in figure 4.

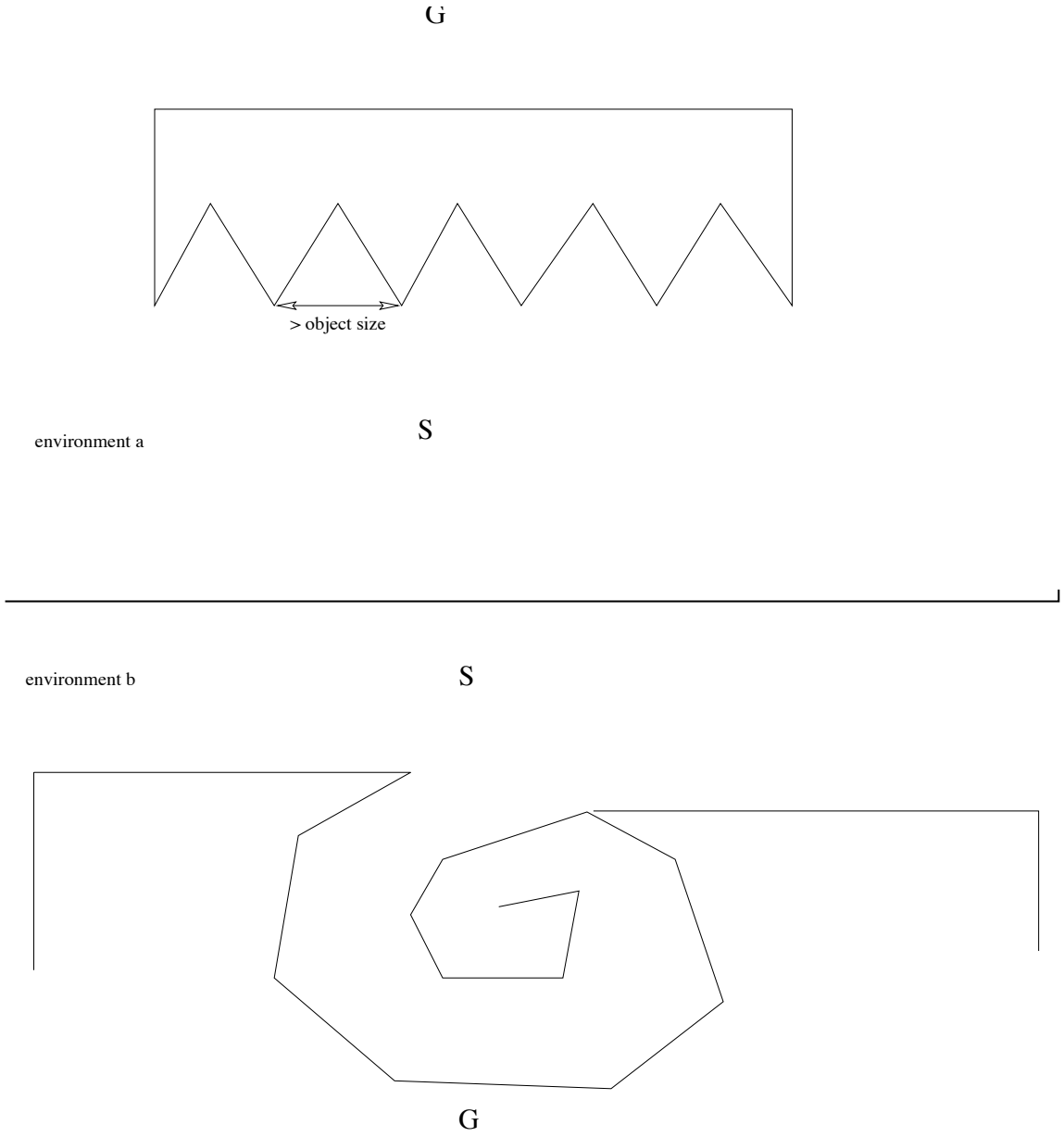


Figure 4: *Examples of complex environments. Environment a: because the width of the opening is greater than the width of the object the scout is guiding, the scout will believe that there is enough room to pass to reach the goal. It will have to explore each trap separately to realize it is a dead-end. The more traps which are present on the obstacle, the more difficult it is for the scout to explore. Environment b: A spiral provides a challenge for the scout as it does in any on-line algorithm. The scout is forced to explore all the way through to the end of the spiral before realizing it is a dead end.*

3.5.4 Path length

Now that we have shown that a restricted version of the algorithm will always terminate, and that the generalized algorithm will find a path if one exists, we can begin to argue about the length of the path this algorithm will find.

Theorem 4: The path found by the scout can be related with a constant factor to the shortest path along the visibility graph found by an off-line algorithm. Let L represent the width of the obstacle which from the side which the visibility graph approaches it. Each segment in the path found by the scout will either fall exactly on the corresponding segment of the visibility graph path, or will be a one of a pair which go with a set of segments, S, in the visibility graph path. In the case where the segment in the scout's graph is one of a pair of segments, the total length of these two segments will differ by less than 2L from the total length of the segments in S.

To get the idea of what this theorem is stating, see figure 5 for an illustration of some example cases.

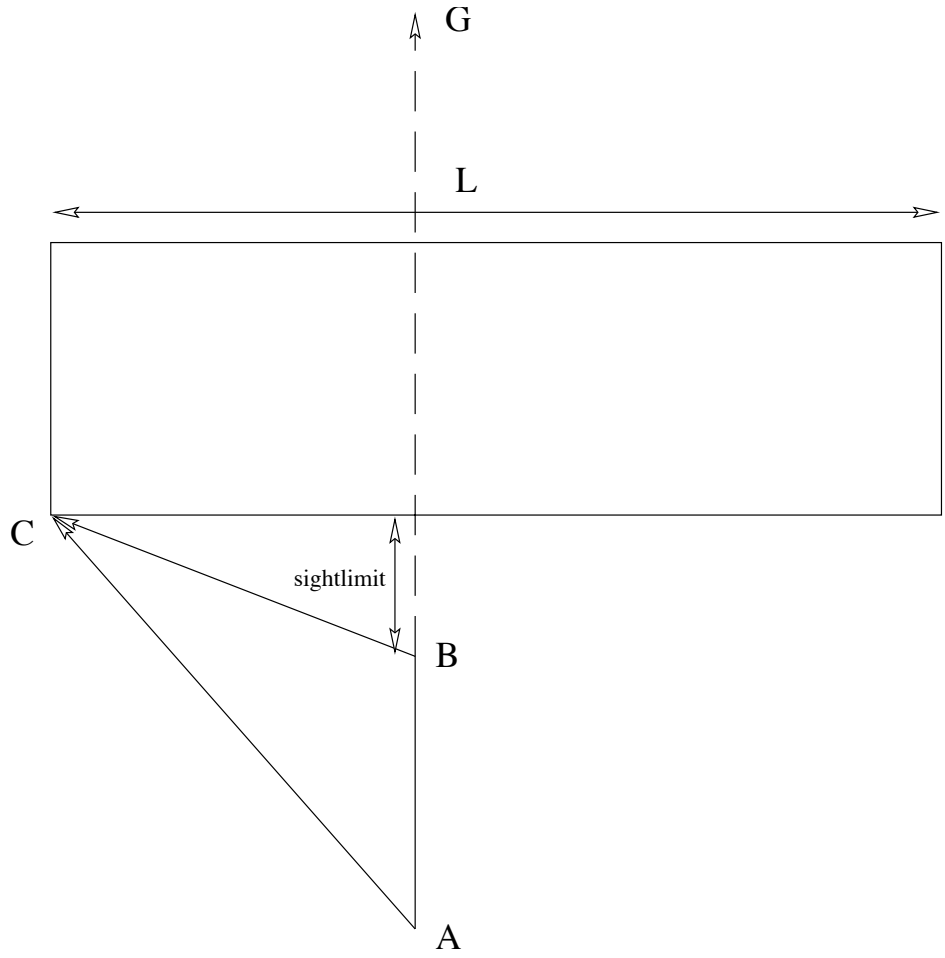


Figure 5: *How the scout's path may differ from the optimal path. If the distance from A to the obstacle is greater than the scout's sight limit, the scout will head along a straight-line path toward the goal (segment AB) before turning to rejoin the optimal path (segment BC). The optimal path is represented with segment AC. The combined lengths of AB and BC will be less than the length of AC plus 2L.*

Proof:

To begin this proof, we will first establish a few necessary facts.

Lemma 4: A greedy search based on the total estimated path length (distance traveled to reach the point in question plus that point's distance from the goal) will yield the shortest path to the goal in two-dimensional space.

Proof:

This is simply A* search, which is proven to always yield the shortest path with an admissible heuristic. Because we know that the distance of a point from the goal must be at least as great as any path from that point to the goal (i.e. the shortest distance between two points is a straight line), this heuristic is clearly admissible.

Although we are working in three-dimensional space, we only allow two degrees of freedom; thus, we have effectively reduced our environment to two dimensions.

Lemma 5: Consider an edge from point p_n to point p_{n+1} in the shortest path from start to goal (which is on the visibility graph), where p_{n+1} is not the goal point. Then, if the point p_n is a point on the path that the scout returns, and if there is no apparent straight-line path to the goal from p_n, point p_{n+1} will be the next point on the scout's path if and only if the distance between p_n and p_{n+1} is less than the sight limit of the scout.

Proof:

First we prove the forward case: if the distance between the two points is less than the sight limit of the scout, then there exists an edge from p_n to p_{n+1} on the path returned by the scout. We can verify this statement through contradiction. Assume that these two points did not form an edge in the visibility graph. Then it must be the case that there is an edge from p_n to p_k where p_k is not p_{n+1}. We know that if p_k was both detected and chosen as the next point to explore from point p_n, it must be the point that minimizes the total path estimation that can be detected from p_n. But by lemma 4 it can be seen that if p_{n+1} is on the path to the goal, then it must have the shortest path length estimation of any point that can be reached from p_n that is along a path to the goal. We can conclude from this observation that point p_k must either be the same point at p_{n+1} or not lie along a path to the goal. Both of these statements provide contradictions.

Suppose, however, that the algorithm does not choose to explore any point that can be detected from p_n. In this case, there must be an unblocked straight path to the goal

that is longer than the sight limit of the scout, which cannot exist due to the restrictions of the lemma.

Now we can prove the backward case: If the path the scout returns has an edge between p_n and p_{n+1}, then the distance between these two points must be less than the sight limit of the scout. This fact is easily seen. If the scout cannot detect p_{n+1} from point p_n, it simply cannot be the next point the scout explores from, as the scout either chooses a point it detects, or heads in a straight-line towards the goal.

We know that since the algorithm starts its search for the goal on the visibility graph, in particular, at the start node. If the next node on the optimal path to the goal is within the sight limit of the scout, the algorithm will find this point as the next point in its path, by lemma 5. The algorithm will continue to find the points along the optimal path so long as they are always close enough to the previous point, and there is no apparent straight-line path to the goal. To finish our proof of theorem 5, however, we must examine what happens when the next optimal point from a given point is out of the scout's sight limit.

Lemma 6: Consider an edge from p_n to p_{n+1} in the optimal path to the goal. If this edge is longer than the scout's sight limit, then there exists either an apparent straight-line path to the goal or an apparent corner corresponding to the point p_{n+1} which the scout will find. (See figure 6)

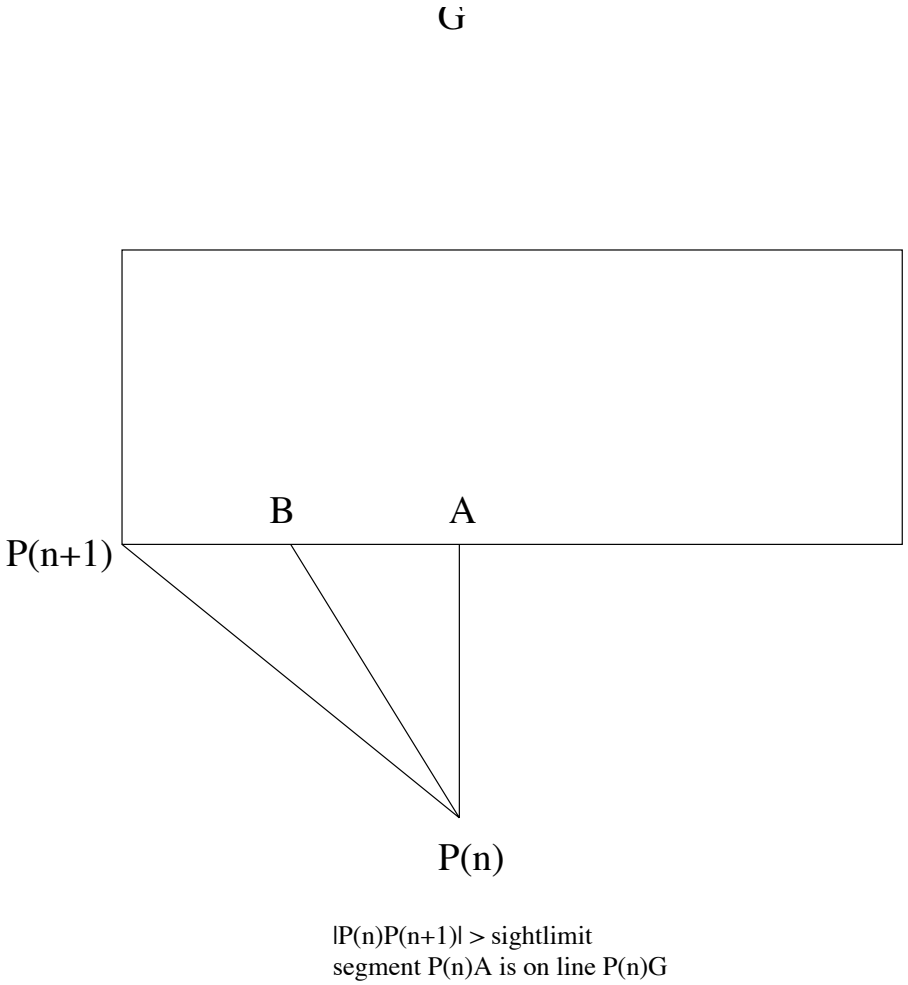


Figure 6: *If the distance from P(n) to A is greater than the scout's sight limit, it will detect a straight-line path to the goal. Otherwise there must exist some point B between P(n+1) and A such that the distance from P(n) to B is equal to the sight limit of the scout. B will be the point that the scout detects as the corner of the obstacle, or the pseudo-corner.*

Proof:

Before we prove this lemma, we must qualify our statement and recall the way the algorithm progresses to corners. Recall that the scout does an initial sweep for corners, and then proceeds to what it thinks are corners. They may, however, not be the actual corners of the obstacles, in which case the scout does another sweep, and continues to

look for corners. It does, however, list these *pseudo-corners*[2] as points in its path to the goal.

If the point p_{n+1} is on the optimal path to the goal, it must be the corner of an obstacle or the goal point. If it is the goal point, then there must be a straight-line path from the point p_n to the goal. If it is not the goal, then it must be the corner of an obstacle that either blocks the straight-line path to the goal, or does not. If the obstacle does not, then there exists a straight-line path toward the goal. If the obstacle does block the scout's direct path to the goal, then it must be within the sight limit of the scout, in which case the scout would detect it as an edge. When the scout detects an edge it will always detect boundaries to the edge because of the limitations of the sonar sensors. These boundaries will be the pseudo-corners mentioned above.

Now, to finish proving the theorem, we must show that when the scout's path deviates from the optimal path, it will eventually regain the optimal path. Furthermore, we must show that the length by which the path is increased each time it deviates from the optimal path is less than 2L, where 2L is the length of the side where the scout's path reencounters the optimal path. We can show that the scout's path will reencounter the optimal path, and using lemma 6, we can establish a correlation between the edges in the scout's path and the edges in the optimal path. Once we have established this correlation, we can then examine it to determine the difference in length.

Clearly, the scout will reencounter the optimal path at the goal if not before. Now we must make some assertions about how must longer the scout's path is than the optimal path. I will examine several scenarios, and eventually show that in the worst case scenario the scout only loses the path for one edge. In other words, two edges in the scout's path correspond to only one edge in the optimal path.

---

[2] We define pseudo-corners as points that the scout perceives as corners from its sonar sweep, but are not actually corners of obstacles.

To begin, let's examine how the scout can possibly regain the path to the goal to gather insight on how the scout's path deviates from the optimal path. By lemma 6 we know that if the scout does not find the next point in the optimal path, it will either proceed along a straight-line path toward the goal, or explore one of a set of points detected from the current point. This set of points will include the pseudo-corner corresponding to the next point in the optimal path.

If a straight-line path to the goal is detected, the scout will move along this path until detecting an object at some threshold distance in front of it. We will assume that the scout started on optimal path to the goal. Clearly, when it takes the straight-line path, it is not necessarily heading to the next point on the optimal path. But, as the scout travels in a straight-line path toward the goal, it will eventually encounter an obstacle in its path. At this point it must navigate around the obstacle by locating the obstacle's corners.

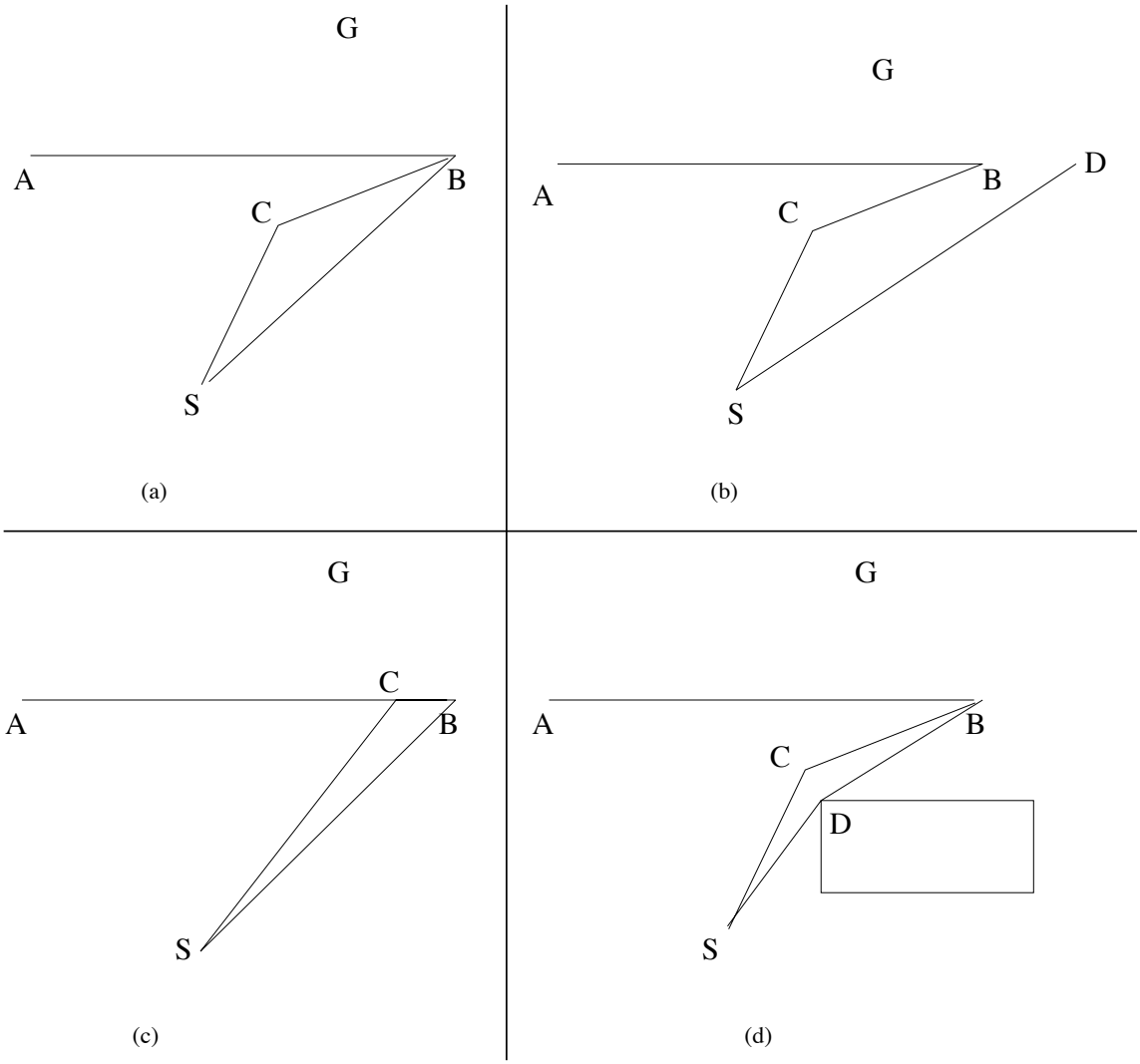


Figure 7: ***(a)*** *The scout's path (SCB) and the optimal path (SB) around the obstacle with width AB.* ***(b)*** *If the optimal path swings wide of the obstacle (SD) but the scout stays close, the difference between the two path length is reduced (|SD| > |SB|).* ***(c)*** *If the distance from S to the obstacle is less than the sight limit of the scout, the scout will find a pseudo-corner (C). The path in part s through this pseudo-corner will be shorter than the path in part a on the straight-line path to the goal.* ***(d)*** *If the scout misses more than one segment of the optimal path (SDB), the difference between the scout's path and the optimal path is again reduced, as the length of SDB is greater than the length of SB.*

When we examine the possible scenarios for the optimal path, we see that in the worst case the scout rejoins the optimal path at the outermost corner of the obstacle in its way. Consider figure 7, part a. Point S represents the location where the scout deviates from the optimal path. The line AB represents the dimensions of the object in the scout's

way. And point G represents the goal, which is shown in order to establish the straight-line path from S toward the goal. Now, we note that both the scout and the optimal path must somehow navigate around the obstacle in order to reach the goal taking the shortest route possible. The scout begins its path by heading straight toward the goal. It eventually will reach the obstacle and navigate around it, heading either to the left or the right. The optimal path must also head around the obstacle. The optimal path around the obstacle must be shorter than the path the scout chooses around the obstacle, so the optimal path may swing, slightly wide of the obstacle's edge, but not too wide.

In the worst case, then, the difference between the length of the optimal path and the length of the scout's path caused by this deviation will be no more than the difference between the sum of length of sides SC and CB, and the length of side SB.[3]

Before we examine how large this difference could possibly be, we must note that by lemma 6, the scout either had to detect a straight-line path to the goal, or had to detect pseudo-corners. We quickly note that the exploration and choice of these pseudo-corners allows for a more optimal approximation of the optimal path (see figure 7c). Thus, examining the straight-line path deviation will give us a worst-case bound on the length of the scout's path relative to the optimal path.

If the scout makes a straight-line deviation from the optimal path, it can skip over any number of segments in the optimal path. However, we see clearly that the more segments it skips, the less the difference between the scout's path and the optimal path (figure 7d). So, to finish our proof of theorem 5, we must only examine the triangle SCB mentioned above. Referring to figure 7a, I will show that the maximum difference between the sum of the lengths of SC and CB and the length of the optimal path edge (SB) is two times the length of segment AB, or twice the width of the obstacle which is between point S and the goal.

---

[3] We note in figure 7b that if the scout swings wide it causes the rest of the path to be longer, and lessens the difference between the scout's path and the optimal path.

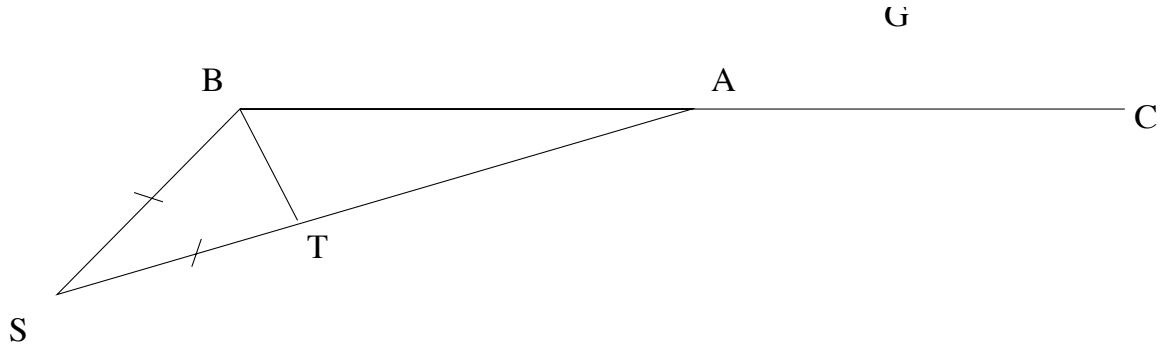


Figure 8: *Worst case scout's path (SAB) could go to the end of the obstacle (BC). The optimal path goes from S to B.*

In the worse case, consider figure 8. Here the scout heads far past the optimal point while on its straight-line path to the goal, along segment SC. Because the scout does not detect that there is an obstacle blocking the goal, it actually moves away from the point to which it will return. Once it detects the obstacle, it marks a point in the path (point C), and maneuvers to avoid the obstacle, eventually reaching point B. Here the angle SBC is very large, indicating that the scout has planned a path that deviates greatly from the optimal path (SB). In the very worst case, the scout travels until it can detect the obstacle, or in other words, until the end of the obstacle. If it reaches the end of the obstacle, but does not choose to backtrack, we know that the path that it has found is shorter than a path that would have been found had it chosen to backtrack to point B. So we can get a worst case by looking at the case in which it chooses to backtrack to point B.

The extra length that the scout travels in the very worst case is 2L, where L is the length of this obstacle. If we subtract the length of SB from SC, we find that the segment TC is shorter than BC because SBC is less than 180 degrees. Thus, the total extra length the scout travels is less than two times BC, which is less than two times BA, or L.

By simply applying this theorem, we could infer a worst case bound of two times the perimeter of every obstacle in the environment. However, because of the nature of our search, we can combine our search algorithm and theorem 5 to prove a tighter bound.

Theorem 6: In the worst case, the length of the path which this algorithm returns will be no more than the sum of two times the width of each obstacle in the environment plus the length of the optimal path.

Proof:

This can be seen rather simply. As we noted before, whenever the scout's path deviates from the optimal path, both paths will eventually have to navigate around an obstacle in their path. Each time this happens, we get a maximum deviation noted in theorem 5, of two times the width of the obstacle. If we consider the algorithm, it is clear that the optimal path will never navigate around the same object more than once. So, when sum up all the deviations across all the environment, we get a maximum deviation of the sum of two times the width of each obstacle in the environment.

4 The Experiment and its Results

4.1 System design

4.1.1 Implementation

I implemented the system with a number of C++ classes. I build the code on basic classes which Michael Ross and myself wrote for the previous implementation. The basic classes upon which I built included classes to control the robot's motion (class *basic_motion*), the robot's sensor readings (class *good_sonar*), and the robot's rotating sonar (class *rotating_sonar*). These base classes were designed to account for some of the errors inherent in the hardware and existing in the basic software provided with the scout robot. The basic_motion class provides a clean way to give the scout motion commands, as well as slowing down the movements he performs to make them more accurate. In

particular it has implemented a slight pause after each rotation command to ensure that the scout has finished his turn before he attempts his next maneuver. The good_sonar class attempts to account for erroneous readings the sonar can sometimes return. It takes three successive readings from one given sensor. If the three readings are sufficiently close together, the scout assumes they are correct and returns the average of the three. If they are not close enough together, the scout discards them and takes three more readings. Finally, the rotating_sonar class uses the robust reading algorithm of good_sonar while providing a clean way to control the rotation of the sonar.

The two main classes I added are the class to encode the storage structure, discussed in section 3.4 (class *pathTree*), and the class to control the exploration of the environment (class *pathFinder*). PathFinder used class path tree to store the information it gathered from its surrounding environment. Class pathFinder itself had three main components--sweeping, loading, and moving –corresponding to three major sections of the algorithm. The sweep section controlled the sonar sweep. The load section controlled the scout's loading of the points detected into the path tree and the choice of the next point to explore. The move section controlled the scout's motion and checked for problems as the scout proceeded to its next point. Figure 9 illustrates roughly how all the classes outlined here work together in the implemented system. A listing of all the code is available at *ftp://ftp.cs.dartmouth.edu/TR/TR98-336.code.tar.Z.*

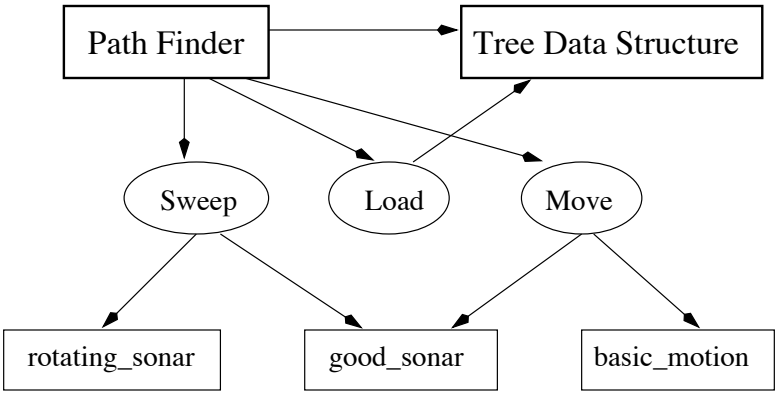


Figure 9: *Diagram illustrating the relationships between the various classes. The circles represent subsections of the pathFinder class. Connections show which classes use and depend on which others.*

4.1.2 Experimental platform

I implemented my system in the Dartmouth College Robotics Lab, using an RWI Pioneer robot as my scout. The scout is equipped with 8 sonar sensors – 7 fixed position and 1 rotating. Five of its fixed sonar sensors are located in front of the scout and are used for detecting obstacles as the scout moves forward along a path. Two of the fixed sonar sensors are located one on either side of the scout. I have used these side facing sensors to detect when the scout enters a corridor that is too narrow for the system that the scout is guiding to pass. Finally, the scout has its one rotating sonar located on its left-hand side, about 20 cm to the rear of the robot from its fixed side sonar. Because its rotating sonar is located on his left, each time it performs a sweep, it must rotate 90 degrees to the right, and then rotate back when it is done with its sweep.

The platform limits several variables within the system. The rotating sonar has an angle range of slightly over 90 degrees, so I used 90 degrees as the sweep angle for the scout. Recall that with a 90-degree sweep angle the scout will always terminate its search, but will not find any path that leads away from the goal. Its sonar sensors have a range of about two and a half meters, but, when confused, can sometimes return a value of just under two meters. I considered readings of a meter and a half and under to be valid. Thus, in my

experiments, the scout's sight limit was 1.5 meters, implying that in my experiments Jerry could not plan a path for an object that is more then 1.5 meters in diameter. The robot lab contains a tile floor on which the robots can move which measures about 6 meters by 6 meters. All of my obstacles were rectangular, as I used cardboard boxes.

4.2 Limitations to the system

Implementing any theoretical system always has its difficulties, and this system was no exception. There were many limitations to the physical system that caused my implementation to stray from the theoretical model.

To begin with, as I have mentioned before, the sonar sensors have some inherent limitations. They give approximate readings only, assuming that the reading that they are receiving comes from a face that is exactly perpendicular to their angle of measurement. This assumption causes incorrect readings when we do our initial sonar sweep. While the sensor will get at least one correct reading on any face which it contacts, it will have contact with any face over a small range of angle positions. Because it gets readings at these different angles, it assumes that the face is always exactly facing the sonar sensor, which causes the points that the sensor detects to take on a curved nature, as the sonar rotates, instead of being represented by a flat face. Also, if the face it too large, the sensors we are using then not to get readings as far out as the corners, as the angle is too great by the time the sensor has reached the corner. Finally, the readings the sensors receive are not always accurate, and often the system receives data points that are not indicative of the edges of obstacles.

The odometry also induces errors into the system. When the scout moves, it does not always exactly keep track of how far it has moved, but rather, is off by a small amount. While this difference does not cause errors in the tests that I performed because they were not over a sufficiently large environment, in an unlimited environment these differences

could begin to add up and cause problems. We will discuss ways later in the paper to reduce errors caused by faulty odometry readings.

Perhaps the greatest difference between the theoretical system and the physical system is the actual navigation and orientation of the scout. In theory, the scout can scan for its corners, proceed to the chosen corner, and repeat the search. However, because of the limitations mentioned above, the scout cannot simply move blindly to a corner it detects. For starters, the scout itself must maintain a safe distance away from the obstacle. And because the scout needs to maintain this safety distance, it must determine to which side of the corner it should proceed by choosing the side which exists in free space, not blocked by the box. Orientation of this manner is exceedingly difficult using only sonar sensors. A fellow researcher and I have developed a robust algorithm for edge localization using two side sonar sensors, which I will discuss in the following section.

Finally, the scout's speed and accuracy in locating points in space limits the systems ability to strictly adhere to the optimal algorithm. If the scout had an unbounded amount of time to search, or if the scout moved much faster than the movers, then it would be to our advantage to optimize the path the scout returns using the algorithm presented in the previous section of this paper. However, in the system with which I am working, the scout moves only minimally faster than the movers do. Therefore, it is to my advantage to modify the algorithm slightly to prevent the scout from continually abandoning its search down one portion of the tree to search down apparently more optimal branches. I claim that the time the manipulation system loses by following a slightly longer path is easily recovered in the scouting portion of the algorithm through the scout's saved backtracking.

4.2 Deviations from the Theoretical System

Now that I have mentioned the motivations behind straying from the theoretical system, I will elaborate on the specific aspects of the implementation which differ from the theoretical model.

4.2.1 Corner localization

Let's consider what actions the scout must take when it locates a corner that it chooses to explore. After the scout does its initial sweep, it chooses the next best corner to explore, and then heads to that corner. In theory these corners may be real corners, if such corners are within range, or they may be the pseudo-corners mentioned earlier. When the scout detects such pseudo-corners in the theoretical model, it can treat them as real corners, proceed to them, and repeat the sweep process.

In the physical system, such pseudo-corners occur frequently, both because the corners are further than the scout's sight limit and because of angle limitations of the sonar mentioned above. In order to repeat the sonar sweep process, the scout must find a corner and move slightly away from it. So, in contrast to the theoretical system, the scout cannot simply treat these pseudo-corners as real corners, because they are blocked on both sides by the obstacle. The scout must find the real corner associated with the perceived corner.

The scout finds this corner is as follows:

If the scout is approaching the edge of the obstacle at a shallow angle:

If the angle at which the scout is approaching is shallow enough (i.e. the scout detects the obstacle only with one of its side sonar sensors), then the scout is virtually heading parallel to the side of the obstacle. In this case, the scout simply adjusts its position so it is exactly parallel to the face of the obstacle as it follows along its edge. It can accomplish this straightening out by keeping track of its distance from the obstacle's face. If the scout is moving toward the obstacle, it turns away. It the scout is moving away from the obstacle, it turns towards it. If its distance from the obstacle does not change significantly over time, it is heading parallel to the obstacle's face, and continues to do so until it no longer detects the obstacle with its side sonar. The scout has then reached the corner of the obstacle.

If the scout is approaching the edge of the obstacle at a large angle:

In this case, the scout is on a collision course with the obstacle. It must stop before crashing into it, and consider its position along the obstacle's face. The first thing the scout does is rotate 90 degrees to the right to prepare itself to locate the edge of the obstacle.[4] It then moves forward or backward a small distance depending on whether it thinks it is on the obstacle's left side (backwards) or right side (forwards). If the scout cannot determine which side of the obstacle it is on (i.e. it has readings from all its sensors, or only the middle one), it does not move forward or backward, but will most likely locate the right corner of the face. The reason for this assertion and for the moves will become clear after I discuss the edge location algorithm.

Now the scout is ready to locate the side of the box. To position itself parallel to the side of the box the scout performs the following steps:

1. The scout does a sweep with the rear sonar and keeps track of the angle with the lowest reading.
2. The scout turns so as to make the angle at which the scout determined the lowest reading parallel to the face.
3. The scout backs up until the fixed side sonar has contact with the face.
4. The scout sweeps with the rear sonar again. If the lowest reading was found within a 5-degree angle of the scouts heading, the scout decides it is parallel to the face. Otherwise the scout repeats steps 1 through 4.

Once the scout has positioned itself parallel to the edge of the box, it moves forward until it loses contact with the box, having found the corner.

We can now examine why the scout chooses to move forward or backward to locate the corner. If the scout locates the edge of the obstacle that it approached will find the right hand corner of the box, due to the fact that the sonar sensors are on the scout's left side. However, because the edge finding algorithm involves backing up, the algorithm will cause the scout to back around the right hand corner if it starts sufficiently close to it. Thus, by

[4] Recall that there are two sonar sensors on the scout's left side, one fixed and one rotating. The scout will use both in its process positioning itself parallel to the side.

backing up, we force the scout closer to the left corner of the face. If the scout was originally sufficiently close it will back around the corner, find the face which is to just to the left of original face. When the scout then navigates off the edge of that face, it will have found the left-hand corner of the original face.

Obviously by this method of corner localization, the scout will find a corner with slightly different coordinates that the one which it detected in its sonar sweep. Because the scout has an internal system of coordinates, it can correct for this position change by simply modifying the coordinates in the node it is exploring to match its current coordinates. Another piece of information which must be loaded at this point which was not mentioned in the theoretical system is point through which the manipulators must pass, that is, the point in the path taking into account the dimensions of the object to be moved. The scout has a good idea of which side of its current position the obstacle is located, so it can determine the modified coordinate by simply calculating the point in space which is half the object's width away from the obstacle. Then, when the scout moves, it still will still have the means to return the modified point, even though it no longer has the means to calculate that point.[5]

4.2.2 Path exploration

A second major difference between the physical system and the theoretical system is that the scout will not backtrack to attempt to find a more optimal path. That is, once the scout begins to explore a path, it will continue to explore along that branch of the tree unless it reaches a dead end. In the optimal system, the scout would always explore the node in the tree that had the minimum estimated path to the goal. This exploration could, in the worst case, cause the scout to explore a different branch of the tree every time it explored a new node. In the physical world, the scout has to move back through the nodes that it has already explored. This movement theoretically should not be a problem, since it

[5] This is due to the fact that the data structure simply stores the coordinates of the corner, not how that corner relates to the obstacle.

has exact coordinates for points it has already explored and since it has already determined that there are no obstacles blocking the path between the coordinates. However, the scout still requires time to retrace its steps, and since our odometry system is less than ideal, we would like to minimize the scout's unnecessary movement. As a side note, if we were to implement this backtracking part of the algorithm, it would be a simple task to determine the path from any node to another in the tree since we have each node's child list from the start of the tree. So, to determine the path from any node to another, we simply compare their child lists. The node in the lists where they diverge represents the place in the tree where the nodes' branches diverge. We can then easily determine how to navigate up the branch of the first node and down the branch of the second.

4.3 Experimental runs

My goal was to test the system in a variety of different cases, starting very simple and progressing to more difficult environments. In the previous system on which I worked solving the same task, the system was able to successfully able to avoid an obstacle detected on the right about 50% of the time. [RKKS96] Any result which improves on this percentage would be an improvement to the system, although I also realize that my task is only concerned with the scouting section of the system and avoids all the errors induced by the manipulation process.

As I mentioned at the beginning of this paper, a major advantage to separating the scout from the manipulation process is that the scout can more thoroughly explore the environment to avoid obstacles. In particular, the scout is able to avoid obstacles that would not have been detected by the previous system in which the scout was rooted alongside the object being moved. I was able to test my scout on obstacles that were in all locations along the path to the goal-- directly in front, to the left, to the right.

To run the experiments, the scout was placed at one end of the floor with an arrangement of boxes in its path toward the goal. The scout was then told where the goal was located, and set free to plan its path toward the goal.

4.4 Results and analysis

I tested the system over 56 total runs, involving a variety of cases. All of the cases for which I tested contained a valid path which always "flowed" toward the goal, which the scout theoretically been able to find. As we noted above, the scout cannot find any path that heads away from the goal since it uses only a ninety-degree sweep angle. The cases were broken up into *simple* cases (n=36) in which the scout was only presented with spaces which were big enough for the object it was guiding to fit through, and *complex* cases (n=18), in which the scout had to recognize gaps which were big enough for it to fit through, but not big enough for the object to fit though. Furthermore, in each of the simple cases, one main obstacle blocked the scout from the goal, and it could be oriented squarely (*square*, n=29) or at an angle (*angled*, n=7) relative to the starting direction of the scout.

The scout performed quite well in most of the test cases. Overall, the scout successfully navigated and found a valid path around the box on 37 of its 54 tries. This is a 69% success rate, which improves on the previous 50% rate. If we look at individual cases the number improves for certain cases. In the simple straight case, the scout successfully planned a path in 22 of 29 tries, or 76% of the time. In the angled case, the success rate drops to 57%, or 4 of 7 tries. And for the complex case the success rate is 56%, or 10 of 18 attempts. See figure 10 for a complete table of the results.

	number of runs	number of successes	percent success
Simple	36	27	75%
straight	29	22	76%
angled	7	4	57%
Complex	18	10	56%
Total	54	37	69%

Figure 10: *Results from test runs.*

To get a real feel for what the system does well and what it does not do well we must examine the reasons for failure and the behavior under specific circumstances more closely. In the simple case, the reasons for the scout's failure included missed commands (2 cases) and failed location of a corner of the box (8 cases). A missed command cannot be easily accounted for in my software, but must be improved upon in the communications software. However, it is the missed corners that are the biggest cause of failure in the simple system. The scout misses corners when it either fails to pinpoint them using its corner location system, as described in section 4.2 above, or when it repeatedly locates the same corner using its corner location system. This repeated location of the same corner can occur because the sonar sweep fails to identify a corner which is far enough away from the current point. Failed corner location turned out to be more of a problem for the angled case, because as the scout faced the goal it had a more shallow angle with the obstacle, and thus could not get a precise reading on the corner location from its initial sweep.

In the complex case the scout encountered difficulties that it did not encounter in the simple system, as well as some of the same problems. When the scout had to deal with gaps that were too small for the obstacle to fit through, it not only failed to locate the

corners it was supposed to locate (1 case), but it also got stuck in the gap between the obstacles which the object could not fit through (6 cases), and incorrectly determined that a path to the goal did not exist (2 cases). The scout got stuck in the gap when it falsely identified the corners which made up the gap as valid corners to the path, and then could not back up enough to locate the valid corners after realizing that the corner which it had found was not valid. However, on a good note, the scout never chose a point on the path toward the goal as being valid when it was not (i.e. the scout never planned a path through a corridor which was too narrow for the object to fit through).

Looking at a couple specific behaviors of the scout, we see patterns worth noting. First, the scout tended to choose a path around the right hand side of the box, even when the left-hand side was slightly closer. This behavior is due to the fact that the scout's rotating sonar is on its left-hand side; thus, when the scout found an edge, it would more likely be facing the right corner of the box. However, a simple argument by symmetry shows that if the scout had an additional rotating sonar on its right side it would choose the left side as often as it chose the right side of the box. Furthermore, if it had two rotating sonar sensors, it would be better at finding the side of the path that corresponds to the shorter path. Second, the system is extremely robust. Even if the scout fails to find a corner on its first try (for example, it looses contact with the box premature to reaching the corner, and assumes this lost contact indicates a corner) it will almost always find the corner on its next sweep and corner localization.

Although my system performed quite well in the cases on which I tested it, it has two major limitations. First, because of the small sweep angle, the scout is unable to locate paths that , at any point, head away from the goal. The rotating sonar only had a sweep angle of 90 degrees, and, while the scout could have rotated its own position to sweep the entire 360-degree field, this would have taken considerable time and risked introducing errors in odometry caused by excess rotation. The other limitation involves the scouts inability to locate points accurately enough to determine which points are the same.

Because the scout always finds moves to a point just off the corner of the obstacle, and then adjusts its point, the point which the sonar sensor detects will rarely, if ever, correspond to the point in space which the scout finds. Thus, in practice, since the scout cannot tell which points it has explored and which it has not, its search time is not bound by the number of corners in the space. The scout could search a space for an undetermined amount of time looking for a path to the goal when none exists.

5 Extensions

Although the system performed quite well in practice, it could be improved through various means, ranging from simple to more complex.

The first extension to the scouting system would be to add the manipulation system to move the object along the path that the scout returns. This should be a simple task. It requires the manipulators to have an internal odometry system. They could then blindly turn toward the next point in the path and head to it, since the scout has already located a clear path. The addition of the manipulators must be performed for the system to be complete, since the scouting system was clearly designed with the manipulation system in mind.

Another more interesting extension involves the use of the manipulators to help the scout maintain its position information. As I mentioned earlier, one of the limitations to the system is that it tends to lose its position the more it moves. It is inspired by the work of Hirose and Kurazume at the Tokyo Institute of Technology. They describe a way to use a team of robots to solve the position identification problem, “Cooperative Positioning System (CPS)”,[KH98] by first moving one of the two robots while using the other as a landmark, then moving the second, using the first as a landmark. CPS has the advantages of being more reliable than dead reckoning, and can be used indoors and in unknown environments [KH98], which are conditions we would like to meet in our system. The difference between our system and the one described by Kurazume and Hirose is that in

our system only one of the robots (the scout) is mobile while exploring the environment. Since their system depends on the robots being able to detect each other, when the scout wanders away from the manipulation system, it will not be able to use them as a landmark. Also, because the scout relies on sonar data it cannot necessarily tell the manipulation robots from the other landmarks in the environment. Finally, because the manipulators are supposed to be blind (if they were not we would not have a separate scouting in the first place), they cannot detect the scout to get a reading on its position.

If a minimum number of these restrictions can be overcome, the scout might gain critical dependency about its location. This added accuracy would allow the scout more freedom to move around its environment. It could then backtrack, as called for by the original algorithm, and find a more optimal path to the goal.

6 Acknowledgements

This work was done entirely in the Dartmouth College robotics laboratory, headed by Daniela Rus. I would like to thank Professor Rus particularly for her guidance. I would also like to thank Keith Kotay for his countless hours spent maintaining the equipment in the lab so that I could do my research. In addition, much of the background work for this paper was done in conjunction with Michael Ross, whom I would like to especially thank for the many hours we spent together developing some of the crucial background for my current work.

References:

[ALMR97] P.K. Agarwal, J.C. Latombe, R. Motwani, and P. Raghavan, Nonholonomic Path Planning for Pushing a Disk Among Obstacles, Proc. 1997 IEEE International Conference on Robotics and Automation.

[BCHLSST98] W. Burgard, A.B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Schulz, W. Steiner, S. Thrun, 1998. The Interactive Museum Tour-Guide Robot. To appear at AAAI-98.

[BLA93] J. Barraquand and J.C. Latombe. Nonholonomic Multibody Mobile Robots: Controllability and Motion Planning in the Presence of Obstacles, Algorithmica, 10(2-3-4):121-155, 1993

[CB95a] H. Choset, J Burdick. Sensor Based Planning, part I: The generalized voronoi graph, In Proc. IEEE Int. Conf. On Robotics and Automation, 1995.

[CB95b] H. Choset, J Burdick. Sensor Based Planning, part II: Incremental construction of the generalized voronoi graph, In Proc. IEEE Int. Conf. On Robotics and Animation, 1995.

[DKP91] X. Deng, T. Kameda, C. Papadimitriou, How to Learn an Unknown Environment (Extended Abstract), 1991.

[GGLLLMT97] H.H. Gonzalez-Banos, L.J. Guibas, J.C. Latombe, S.M. LaValle, D. Lin, R. Motwani, and C. Tomasi, Motion Planning with Visibility Constraints: Building Autonomous Observers. The 8th Int. Symp. of Robotics Research, Hayama, Japan, October 3-7, 1997.

[HIKK] F. Hoffmann, C. Icking, R. Klein, K. Kriegel, The Polygon Exploration Problem: A New Strategy and a New Analysis Technique.

[JKT97] J. Jennings, C. Kirkwood-Watts, C. Tanis, Distributed Map-making Using Online Generalized Voronoi Graphs, Dept. of Electrical Engineering and Computer Science, Tulane University, 1997.

[JWE97] J. Jennings, G. Whelan, W. Evans, Cooperative Search and Rescue with a Team of Mobile Robots, In IEEE Int. Conf. On Robotics and Automation, Albuquerque, NM, 1997.

[K94] J. Kleinberg. The localization problem for mobile robots. Proc. 35th IEEE Symposium on Foundations of Computer Science, 1994.

[KH98] R. Kurazume, S. Hirose, Study on Cooperative Positioning System – Optimum Moving Strategies for CPS-III, Tokyo Institute of Technology, Tokyo, Japan, 1998.

[RKKS96] D.Rus, A. Kabir, K. Kotay, M. Soutter, Guiding Distributed Manipulation with Mobile Sensors, Department of Computer Science, Dartmouth College, 1996.

[T98] S. Thrun, 1998. Learning Metric-Topological Maps for Indoor Mobile Robot Navigation, AI Journal 99(1), 21--71.

[TFL94] H. Takeda, C. Facchinetti, and J.C. Latombe, Planning the Motions of a Mobile Robot in a Sensory Uncertainty Field, IEEE Transactions on Pattern Analysis and Machine Intelligence, 16(10):1002-1017, 1994.