Path Relaxation: Path Planning for a Mobile Robot

Charles E. Thorpe and Larry H. Matthies

Carnegie-Mellon University

Abstract. Path Relaxation is a method of planning safe paths around obstacles for mobile robots. It works in two steps: a global grid search that finds a rough path, followed by a local relaxation step that adjusts each node on the path to lower the overall path cost. The representation used by Path Relaxation allows an explicit tradeoff among length of path, clearance away from obstacles, and distance traveled through unmapped areas.

1. Introduction

Path Relaxation is a two-step path-planning process for mobile robots. It finds a safe path for a robot to traverse a field of obstacles and arrive at its destination. The first step of path relaxation finds a preliminary path on an eight-connected grid of points. The second step adjusts, or "relaxes", the position of each preliminary path point to improve the path.

One advantage of path relaxation is that it allows many different factors to be considered in choosing a path. Typical path planning algorithms evaluate the cost of alternative paths solely on the basis of path length. The cost function used by Path Relaxation, in contrast, also includes how close the path comes to objects (the further away, the lower the cost) and penalties for traveling through areas out of the field of view. The effect is to produce paths that neither clip the corners of obstacles nor make wide deviations around isolated objects, and that prefer to stay in mapped terrain unless a path through unmapped regions is substantially shorter. Other factors, such as sharpness of corners or visibility of landmarks, could also be added for a particular robot or mission.

Path Relaxation is part of Fido, the vision and navigation system of the CMU Rover mobile robot. [7] The Rover, under Fido's control, navigates solely by stereo vision. It picks about 40 points to track, finds them in a pair of stereo images, and calculates their 3D positions relative to the Rover. The Rover then moves about half a meter, takes a new pair of pictures, finds the 40 tracked points in each of the new pictures and recalculates their positions. The apparent change in position of those points gives the actual change in the robot's position.

Fido's world model is not suitable for most existing path-planning algorithms. They usually assume a completely known world model, with planar-faced objects. Fido's world model, on the other hand, contains only the 40 points it is tracking. For each point, the model records its position, the uncertainty in that position, and the appearance of a small patch of the image around that point. Furthermore, Fido only knows about what it has seen: points that have never been within its field of view are not listed in the world model. Also, the vision system may fail to track points correctly, so there may be phantom objects in the world model that have been seen once but are no longer being tracked. All this indicates the need for a data structure that can represent uncertainty and inaccuracy, and for algorithms that can use such data.

Section 2 of this paper outlines the constraints available to Fido's path planner. Section 3 discusses some common types of path planners, and shows how they are inadequate for our application. The Path Relaxation algorithm is explained in detail in Section 4, and some additions to the basic scheme are presented in Section 5. Finally, Section 6 discusses shortcomings of Path Relaxation and some possible extensions.

2. Constraints

An intelligent path planner needs to bring lots of information to bear on the problem. This section discusses some of the information useful for mobile robot path planning, and shows how the constraints for mobile robot paths differ from those for manipulator trajectories.

Low dimensionality. A ground-based robot vehicle is constrained to three degrees of freedom: x and y position and orientation. In particular, the CMU Rover has a circular cross-section, so for path planning the orientation does not matter. This makes path planning only a 2D problem, as compared to a 6 dimensional problem for a typical manipulator.

Imprecise control. Even under the best of circumstances, a mobile robot is not likely to be very accurate: perhaps a few inches, compared to a few thousandths of an inch for manipulators. The implication for path planning is that it is much less important to worry about exact fits for mobile robot paths. If the robot could, theoretically, just barely fit through a certain opening, then in practice that's probably not a good way to go. Computational resources are better spent exploring alternate paths rather than worrying about highly accurate motion calculations.

Cumulative error. Errors in a dead-reckoning system tend to accumulate: a small error in heading, for instance, can give rise to a large error in position as the vehicle moves. The only way to reduce error is to periodically measure position against some global standard, which can be time-consuming. The Rover, for example, does its measurement by stereo vision, taking a few minutes to compute its exact position. So a slightly longer path that stays farther away from obstacles, and allows longer motion between stops for measurement, may take less time to travel than a shorter path that requires more frequent stops. In contrast, a manipulator can reach a location with approximately the same error regardless of what path is taken to arrive there. There is no cumulative error, and no time spent in reorientation.

Unknown areas. Robot manipulator trajectory planners usually know about all the obstacles. The Rover knows only about those that it has seen. This leaves unknown areas outside its field of view and behind obstacles. It is usually preferable to plan a path that traverses only known empty regions, but if that path is much longer than the shortest path it may be worth looking at the unknown regions.

Fuzzy objects. Not only do typical manipulator path-planners know about all the objects, they know precisely where each object is. This information might come, for instance, from the CAD system that designed the robot workstation. Mobile robots, on the other hand, usually sense the world as they go. Fido, instead of having precise bounds for objects, knows only about fuzzy points. The location of a

0197-7385/84/0000-0576 $1.00 © 1984 IEEE

point is only known to the precision of the stereo vision system, and the extent of an object beyond the point is entirely unknown.

In summary, a good system for mobile robot path planning will be quite different from a manipulator path planner. Mobile robot path planners need to handle uncertainty in the sensed world model and errors in path execution. They do not have to worry about high dimensionality or extremely high accuracy. Section 3 of this paper discusses some existing path planning algorithms and their shortcomings. Section 4 then presents the algorithms used by Path Relaxation, and shows how they address these problems.

3. Approaches to Path Planning

This section outlines several approaches to path planning and some of the drawbacks of each approach. All of these methods except the potential fields approach abstract the search space to a graph of possible paths. This graph is then searched by some standard search technique, such as breadth-first or A* [8], and the shortest path is returned. The important thing to note in the following is the information made explicit by each representation and the information thrown away.

Free Space methods. [2, 3, 9] One type of path planner explicitly deals with the space between obstacles. Paths are forced to run down the middle of the corridors between obstacles, for instance on the Voronoi diagram of the free space. Free space algorithms suffer from two related problems, both resulting from a data abstraction that throws away too much information. The first problem is that paths always run down the middle of corridors. In a narrow space, this is desirable, since it allows the maximum possible robot error without hitting an object. But in some cases paths may go much further out of their way than necessary. The second problem is that the algorithms do not use clearance information. The shortest path is always selected, even if it involves much closer tolerances than a slightly longer path.

Vertex Graphs. [5, 10, 6] Another class of algorithms is based on a graph connecting pairs of vertices. For each pair of vertices, if the line between them does not intersect any obstacle, that line is added to the graph of possible paths. Vertex graph algorithms suffer from the "too close" problem: in their concern for the shortest possible path, they find paths that clip the corners of obstacles and even run along the edges of some objects. It is, of course, possible to build in a margin of error by growing the obstacles by an extra amount; this may, however, block some paths.

Both free space and vertex graph methods throw away too much information too soon. All obstacles are modeled as polygons, all paths are considered either open or blocked, and the shortest path is always best. There is no mechanism for trading a slightly longer path for more clearance, or for making local path adjustments. There is also no clean way to deal with unmapped regions, other than to close them off entirely.

The **Potential Fields** [1, 4] approach tries to make those tradeoffs explicit. Conceptually, it turns the robot into a marble, tilts the floor towards the goal, and watches to see which way the marble rolls. Obstacles are represented as hills with sloping sides, so the marble will roll a prudent distance away from them but not too far, and will seek the passes between adjacent hills. The problem with potential field paths is that they can get caught in dead ends: once the marble rolls into a box canyon, the algorithm has to invoke special-case mechanisms to cut off that route, backtrack, and start again. Moreover, the path with the lowest threshold might turn out to be a long and winding road, while a path that must climb a small ridge at the start and then has an easy run to the goal might never be investigated.

Another approach that could explicitly represent the conflicts between short paths and obstacle avoidance is the **Regular Grid** method. This covers the world with a regular grid of points, each connected with its 4 or 8 neighbors to form a graph. In existing regular grid implementations, the only information stored at a node is whether it is inside an object or not. Then the graph is searched, and the shortest grid path returned. This straightforward grid search has many of the same "too close" problems as the vertex graph approaches.

4. Path Relaxation

Path Relaxation combines the best features of grid search and potential fields. Using the rolling marble analogy, the first step is a global grid search that finds a good valley for the path to follow. The second step is a local relaxation step, similar to the potential field approach, that moves the nodes in the path to the bottom of the valley in which they lie. The terrain (cost function) consists of a gradual slope towards the goal, hills with sloping sides for obstacles, and plateaus for unexplored regions. The height of the hills has to do with the confidence that there really is an object there. Hill diameter depends on robot precision: a more precise robot can drive closer to an object, so the hills will be tall and narrow, while a less accurate vehicle will need more clearance, requiring wide, gradually tapering hillsides.

This section first presents results on how large the grid size can be without missing paths. It next discusses the mechanism for assigning cost to the nodes and searching the grid. Finally, it presents the relaxation step that adjusts the positions of path nodes.

Grid Size. How large can a grid be and still not miss any possible paths? That depends on the number of dimensions of the problem, on the connectivity of the grid, and on the size of the vehicle. It also depends on the vehicle's shape: in this section, we discuss the simplest case, which is a vehicle with a circular cross-section.

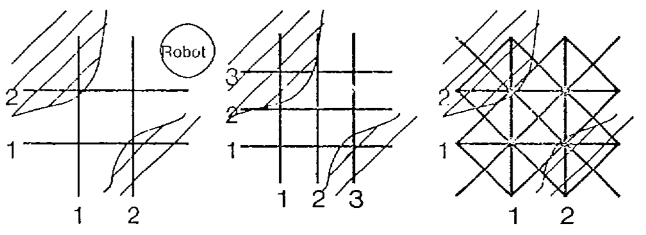


Figure 1: Grid Size Problems

The area to be traversed can be covered with a grid in which each node is connected to either its four or its eight nearest neighbors. For a four-connected grid, if the spacing were r, there would be a chance of missing diagonal paths. At left in Figure 1, for instance, there is enough room for the robot to move from (1,1) to (2,2), yet both nodes (1,2) and node (2,1) are blocked. To guarantee that no paths are missed, the grid spacing must be reduced to r * sqrt(2) / 2, as in the center of Figure 1. That is the largest size allowable that guarantees that if diagonally opposite nodes are covered, there is not enough room between them for the robot to safely pass. Note that the converse is not necessarily true: just because there is a clear grid path does not guarantee that the robot will fit. At this stage, the important thing is to find all possible paths, rather than to find only possible paths.

If the grid is eight-connected, as in the right of Figure 1, (each node connected to its diagonal, as well as orthogonal, neighbors), the problem with diagonal paths disappears. The grid spacing can be a full r, while guaranteeing that if there is a path it will be found.

Grid Search. Once the grid size has been fixed, the next step is to assign costs to paths on the grid and then to search for the best path along the grid from the start to the goal. "Best", in this case, has three conflicting requirements: shorter path length, greater margin away from obstacles, and less distance in uncharted areas. These three are explicitly balanced by the way path costs are calculated. A path's cost is the sum of the costs of the nodes through which it passes, each multiplied by the distance to the adjacent nodes. (In a 4-connected graph all lengths are the same, but in an 8-connected graph we have to distinguish between orthogonal and diagonal links.) The node costs consist of three parts to explicitly represent the three conflicting criteria.

1. Cost for distance. Each node starts out with a cost of one unit, for length traveled.

2. Cost for near objects. Each object near a node adds to that node's cost. The nearer the obstacle, the more cost it adds. The exact slope of the cost function will depend on the accuracy of the vehicle (a more accurate vehicle can afford to come closer to objects), and the vehicle's speed (a faster vehicle can afford to go farther out of its way), among other factors.

3. Cost for within or near an unmapped region. The cost for traveling in an unmapped region will depend on the vehicle's mission. If this is primarily an exploration trip, for example, the cost might be relatively low. There is also a cost added for being near an unmapped region, using the same sort of function of distance as is used for obstacles. This provides a buffer to keep paths from coming too close to potentially unmapped hazards.

The first step of Path Relaxation is to set up the grid and read in the list of obstacles and the vehicle's current position and field of view. The system can then calculate the cost at each node, based on the distances to nearby obstacles and whether that node is within the field of view. The next step is to create links from each node to its 8 neighbors. The start and goal locations do not necessarily lie on grid points, so special nodes need to be created for them and linked into the graph. Links that pass through an obstacle, or between two obstacles with too little clearance for the vehicle, can be detected and deleted at this stage.

The system then searches this graph for the minimum-cost path from the start to the goal. The search itself is a standard A* [8] search. The estimated total cost of a path, used by A* to pick which node to expand next, is the sum of the cost so far plus the straight-line distance from the current location to the goal. This has the effect, in regions of equal cost, of finding the path that most closely approximates the straight-line path to the goal.

The path found is guaranteed to be the lowest-cost path on the grid, but this is not necessarily the overall optimal path. First of all, even in areas with no obstacles the grid path may be longer than a straight-line path simply because it has to follow grid lines. For a 4-connected grid, the worst case is diagonal lines, where the grid path is sqrt(2) times as long as the straight-line path. For an 8-connected grid, the equivalent worst case is a path that goes equal distances forward and diagonally. This gives a path about 1.08 times as long as the straight-line path. In cases where the path curves around several obstacles, the extra path length can be even more significant. Secondly, if the grid path goes between two obstacles, it may be non-optimal because a node is placed closer to one obstacle than to the other. A node placed exactly half way between the two obstacles would, for most types of cost functions, have a lower cost. The placement of the node that minimizes the overall path cost will depend both on node cost and on path length, but in any case is unlikely to be exactly on a grid point. If the grid path is topologically equivalent to the optimal path (i.e. goes on the same side of each object), the grid path can be iteratively improved to approximate the optimal path (see Section 5). But if the grid path at any point goes on the "wrong" side of an obstacle, then no amount of local adjustment will yield the optimal path. The chance of going on the wrong side of an obstacle is related to the size of the grid and the shape of the cost vs. distance function. For a given grid size and cost function, it is possible to put a limit on how much worse the path found could possibly be than the optimal path. If the result is too imprecise, the grid size can be decreased until the additional computation time is no longer worth the improved path.

A few details on the shape of the cost function deserve mention. Many different cost functions will work, but some shapes are harder to handle properly. The first shape we tried was linear. This had the advantage of being easy to calculate quickly, but gave problems when two objects were close together. The sum of the costs from two nearby objects was equal to a linear function of the sum of the distances to the objects. This creates ellipses of equal cost, including the degenerate ellipse on the line between the two objects. In that case, there was no reason for the path to pick a spot midway between the objects, as we had (incorrectly) expected. Instead, the only change in cost came from changing distance, so the path went wherever it had to to minimize path length. In our first attempt to remedy the situation we replaced the linear slope with an exponentially decaying value. This had the desired effect of creating a saddle between the two peaks, and forcing the path towards the midpoint between the objects. The problem with exponentials is that they never reach zero. For a linear function, there was a quick test to see if a given object was close enough to a given point to have any influence. If it was too far away, the function did not have to be evaluated. For the exponential cost function, on the other hand, the cost function had to be calculated for every obstacle for each point. We tried cutting off the size of the exponential, but this left a small ridge at the extremum of the function, and paths tended to run in nice circular arcs along those ridges. A good compromise, and the function we finally settled on, is a cubic function that ranges from 0 at some maximum distance, set by the user, to the obstacle's maximum cost at 0 distance. This has both the advantages of having a good saddle between neighboring obstacles and of being easy to compute and bounded in a local area.

Relaxation. Grid search finds an approximate path; the next step is an optimization step that fine-tunes the location of each node on the path to minimize the total cost. One way to do this would be to precisely define the cost of the path by a set of non-linear equations and solve them simultaneously to analytically determine the optimal position of each node. This approach is not, in general, computationally feasible. The approach used here is a relaxation method. Each node's position is adjusted in turn, using only local information to minimize the cost of the path sections on either side of that node. Since moving one node may affect the cost of its neighbors, the entire procedure is repeated until no node moves farther than some small amount.

Node motion has to be restricted. If nodes were allowed to move in any direction, they would all end up at low cost points, with many nodes bunched together and a few long links between them. This would not give a very good picture of the actual cost along the path. So in order to keep the nodes spread out, a node's motion is restricted to be perpendicular to a line between the preceding and following nodes. Furthermore, at any one step a node is allowed to move no more than one unit.

As a node moves, all three factors of cost are affected: distance traveled (from the preceding node, via this node, to the next node), proximity to objects, and relationship to unmapped regions. The combination of these factors makes it difficult to directly solve for minimum cost node position. Instead, a binary search is used to find that position to whatever accuracy is desired.

The relaxation step has the effect of turning jagged lines into straight ones where possible, of finding the "saddle" in the cost function between two objects, and of curving around isolated objects. It also does the "right thing" at region boundaries. The least cost path crossing a border between different cost regions will follow the same path as a ray of light refracting at a boundary between media with different transmission velocities. The relaxed path will approach that path.

5. Additions to the Basic Scheme

One extension we have tried is to vary the costs of individual obstacles. The current vision system sometimes reports phantom objects, and sometimes loses real objects that it had been tracking correctly. The solution to this is to "fade" objects by decreasing their cost each step that they are within the field of view but not tracked by the vision module.

Another extension implemented is to re-use existing paths whenever possible. At any one step, the vehicle will usually move only a fraction of the length of the planned path. If no new objects are seen during that step, and if the vehicle is not too far off its planned course, it is possible to use the rest of the path as planned. Only if new objects have been seen that block the planned path is it necessary to replan from scratch.

The relaxation step can be greatly speeded up if it runs in parallel on several computers. Although an actual parallel implementation has not yet been done, a simulation has been written and tested.

6. Remaining Work

Path Relaxation would be easy to extend to higher dimensions. It could be used, for example, for a 3D search to be used by underwater vehicles maneuvering through a drilling platform. Another use for higher-dimensional searches would be to include rotations for asymmetric vehicles. Yet another application would be to model moving obstacles; then the third dimension becomes time, with the cost of a grid point having to do with distance to all objects at that time. This would have a slightly different flavor than the other higher-dimensional extensions; it is possible to go both directions in x, y, z, and theta, but only one direction in the time dimension.

Another possible extension has to do with smoothing out sharp corners. All wheels on the Rover steer, so it can follow a path with sharp corners if necessary. Many other vehicles, are not so maneuverable; they may steer like a car, with a minimum possible turning radius. In order to accommodate those vehicles, it would be necessary to restrict both the graph search and relaxation steps. A related problem is to use a smoothly curved path rather than a series of linear segments.

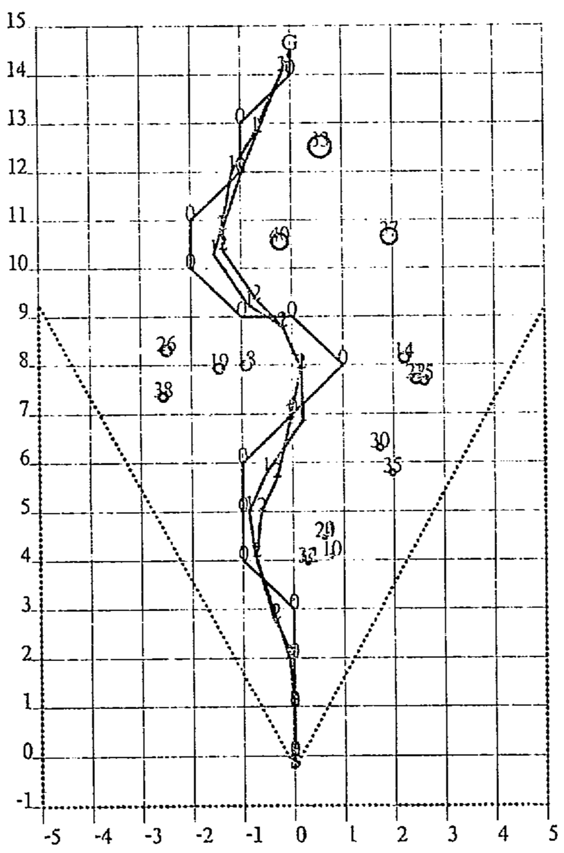
An interesting direction to pursue is multiple-precision grids. This could make it possible to spend more effort working on precise motion through cluttered areas, and less time on wide open spaces.

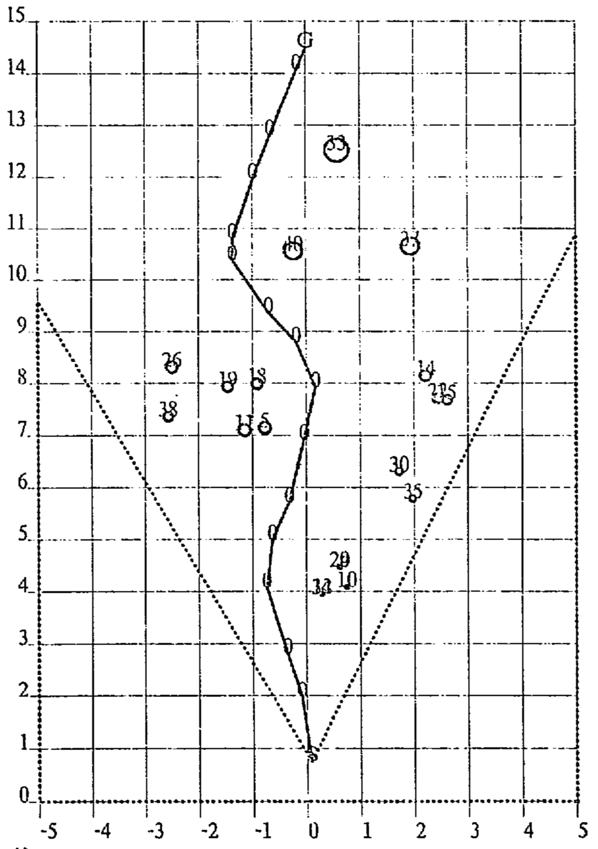
Path relaxation, as well as almost all existing path planners, deals only with geometric information. A large part of a robot's world knowledge, however, may be in partially symbolic form. For example, a map assembled by the vehicle itself may have very precise local patches, each measured from one robot location. The relations between patches, though, will probably be much less precise, since they depend on robot motion from one step to the next. Using such a mixture of constraints is a hard problem.

Acknowledgements Thanks to Hans Moravec and Rich Wallace, both of the Mobile Robot Laboratory, for advice and encouragement. This research was partially supported by Office of Naval Research contract N00014-81-K-0503.

References

1. J. Randolph Andrews. Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator. Master Th., MIT, 1983.
2. Rodney Brooks. Solving the Find-Path Problem by Representing Free Space as Generalized Cones. AI Memo 674, Massachusetts Institute of Technology, May, 1982.
3. Georges Giralt, Ralph Sobek, and Raja Chatila. A Multi-Level Planning and Navigation System for a Mobile Robot; A First Approach to Hilare. Proceedings of IJCAI-6, August, 1979.
4. Oussama Khatib. Dynamic Control of Manipulators in Operational Space. Sixth CISM-IFToMM Congress on Theory of Machines and mechanisms, New Delhi, India, December, 1983.
5. Tomas Lozano-Perez and Michael A. Wesley. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles." *CACM 22*, 10 (October 1979).
6. Hans Moravec. Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover. Tech. Rept. CMU-RI-TR-3, Carnegie-Mellon Univesity Robotics Institute, September, 1980.
7. Hans Moravec. The CMU Rover. Proceedings of AAAI-82, August, 1982.
8. N. Nilsson. *Problem Solving Methods in Artificial Intelligence.* McGraw-Hill, 1971.
9. Colm O'Dunlaing, Micha Sharir, and Chee Yap. Retraction: a new approach to motion-planning. Courant Institute, November, 1982.
10. Alan M. Thompson. The Navigation System of the JPL Robot. Proceedings of IJCAI-5, 1977.

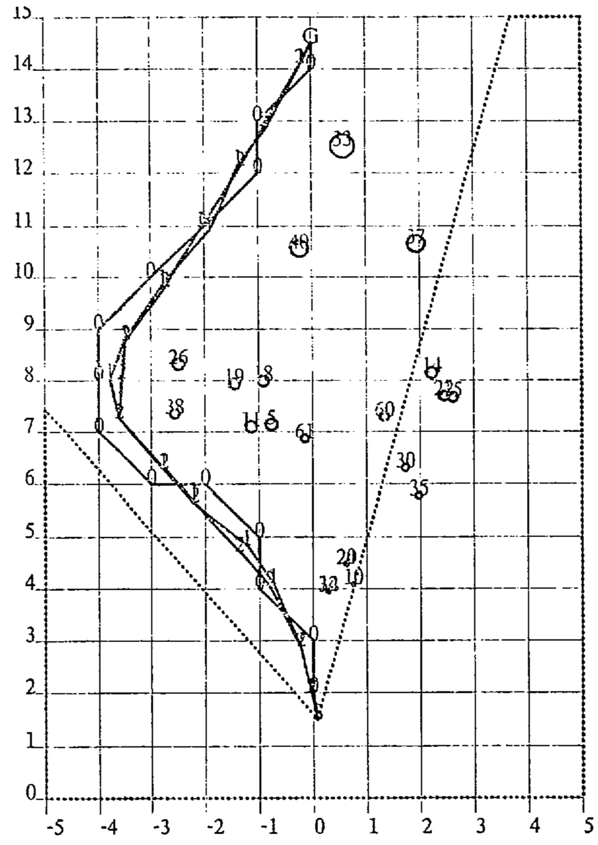


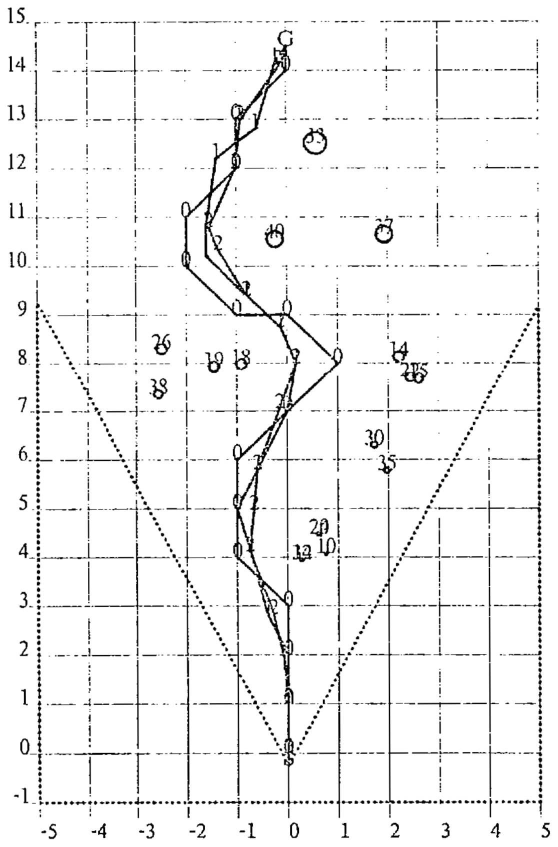


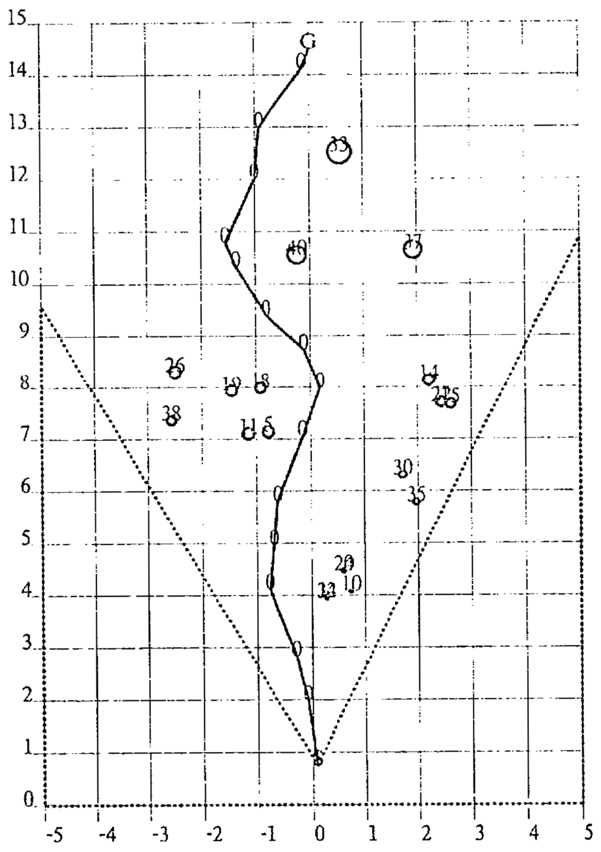
These figures for a set. The top left figure is a run from scratch, using real data extracted from the stored images by the Fido vision and navigation system. Objects are represented as little circles, where the size of the circle is the positional uncertainty of the stereo system. The numbers are not all consecutive, because some of the points being tracked are on the floor or are high off the ground, and therefore aren't obstacles. The dotted lines surround the area not in the field of view; this should extend to negative infinity. The start position of the robot is approximately (0, -.2) and the goal is (0, 14.5). The grid path found is marked by 0's. After one iteration of relaxation, the path is marked by 1's. After the second relaxation, the path is marked by 2's. The greatest change from 1 to 2 was less than .3 meters, the threshold, so the process stopped. The size of the "hills" in the cost function is 1 meter, which means that the robot will try to stay 1 meter away from obstacles unless that causes it to go too far out of its way.

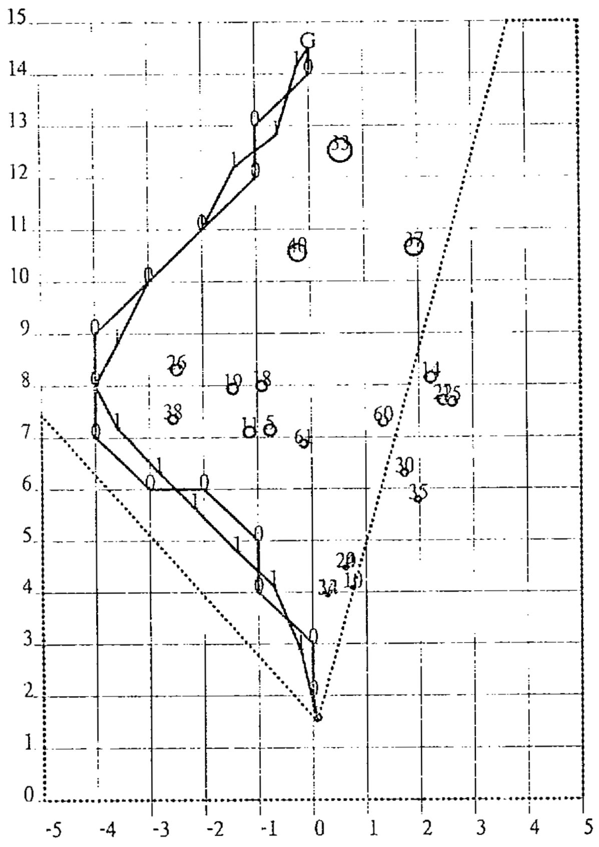
After the robot's first move, it tries to reuse the existing path. The robot is now at about (0.1, 0.8). It deletes the old start node and the nodes at (0,0) and (0,1), and links its current position into the path. In this case it had no new objects to add, so the path was still passable. It then uses the old path without having to plan from scratch, as show in the top right figure.

After the next step, there are some new objects added (see the bottom right figure). In particular, object #61 is too close to the old path, so a new one has to be planned. Again, the 0's are the grid path, and the 1's and 2's mark steps of the relaxation algorithm.









These figures are the same run as the figures on the previous page, except that the relaxation step is simulated to be in parallel. The biggest difference is near the end of the path in the top left figure, where the path is somewhat jagged. The path flips between one state on even-numbered iterations (0 and 2) and the opposite state on odd-numbered steps (1). This could be fixed with damping, at the expense of slower convergence on other parts of the path.