Three Findpath Problems

Richard S. Wallace
Department of Computer Science
Carnegie-Mellon Univeristy
Pittsburgh, PA 15213

Abstract

Three findpath problems are considered. First, the problem of finding a collision-free trajectory for a tentacle manipulator is examined. Second, a new findpath algorithm for a mobile robot rover is presented. This algorithm differs from earlier ones in its use of quantitative information about the uncertainty in the position of the robot to keep the robot away from obstacles without going too near them and without going too far out of its way to avoid them. Third, a method for coordinating two moving arms so that they avoid collisions with each other is presented. The two-arm findpath algorithm here is restricted to cases where coordinated collision-free trajectories can be found by controlling the velocities of the arms. Each of these findpath problems suggests a heuristic to find its solution and these are discussed at the end of the paper.

0. Introduction

With the maturation of the theory of Configuration Space[1] and the foundational work of Schwartz and Sharir on the 'Piano Movers' problem[2] it is tempting to conclude that the final word on the findpath problem has been said. Indeed for the case of a two-dimensional robot moving around fixed planar obstacles (or a three-dimensional robot who can be reduced to a two-dimensional one by projection) many algorithms for the findpath problem have been proposed [3,4,5,6,7,8,9], no one of which is obviously best for simple cases. But the theory of configuration space tells us that for an n-dof robot we must search an n dimensional Euclidean space for a collision-free path when the obstacles are fixed. Also, theoretical work on the findpath problem for linkages indicates that its algorithm is NP-complete[10]. So the issue for computer science becomes not solving the findpath problem in general but examining special cases one by one, to see if we can find an efficient solution for any.

Three findpath problems are examined here: for a planar tentacle manipulator; for a mobile robot, with the additional twist that we don't want the robot to come too near any obstacles nor to navigate too far away from them when avoiding them and; for two coordinated manipulator arms. Each solutions to a findpath problem discussed here suggests a heuristic that may be useful for solving other findpath problems. The heuristics are discussed at the end of the paper.

1. Tentacle Findpath Problem

Spiral functions (i.e. monotonically increasing polar functions of the form $r = f(\theta)$) appear to be reasonable candidates for modeling tentacles. The important consideration from the robot theoretical point of view is that the total length of such a manipulator is constant. The diagram in figure 1 illustrates a tentacle manipulator modeled by a logarithmic spiral (a function of the form $r = e^{a\theta}$). This tentacle robot can "spiral out" or "wind up" and also rotate at its shoulder about the origin. The parameters for this type of arm are its rotational orientation ϕ and a parameter a, which determines how much the manipulator has "spiraled out." For the findpath problem, the logarithmic spiral tentacle has the advantage that a closed form of its inverse kinematic solution is obtained easily [12].

The method used to solve the tentacle findpath problem is an approximation method, in which obstacles in the manipulator's position space are bounded by instances of a class of obstacles, called *tentacle obstacles*, which are simple to map into the robot's configuration space. The obstacles under consideration have four sides. The *left* side is the side first intersected by the tentacle if it is rotating clockwise around the origin. The *right* side is the side first intersected by the tentacle as it rotates counter-clockwise. The *near* side is the side first intersected by the tentacle as it holds its ϕ value constant and increases a, that is, the side nearest the origin The *far* side is the side of the obstacle farthest from the origin. For reasons described below, the near and far sides are always circular arcs and the right and left sides are particular polylines.

The near side of a tentacle obstacle is a circular arc between two angles ϕ_1 and ϕ_2 so that when $\phi_1 \leq \phi \leq \phi_2$ the tentacle parameter a must be sufficiently small to ensure that the point on the tentacle furthest from the origin is closer to the origin than the circular arc. In figure 2 the near sides of the two obstacles correspond with the sides of the configuration space obstacles nearest the ϕ axis and parallel to it. The relationship between a and the point on the tentacle furthest from the origin is non-linear but is bounded by a linear function which is asymptotically equal to the precise relation.

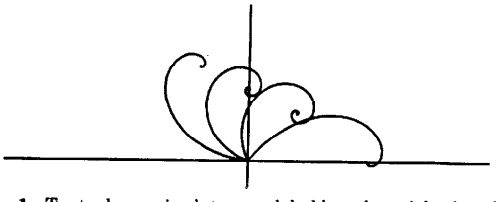

Figure 1. Tentacle manipulator modeled by a logarithmic spiral function shown in several configurations. These drawings were produced by a program which solves the inverse kinematics of this type of arm.

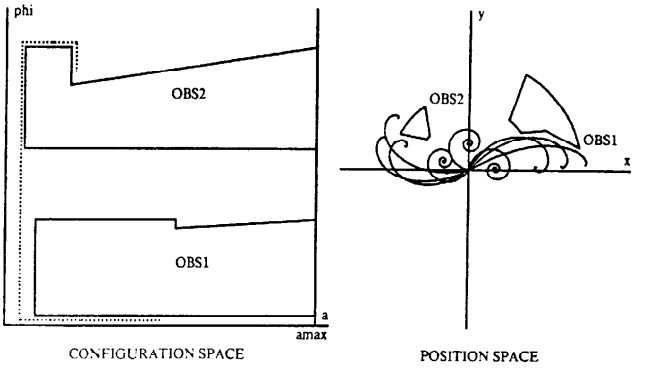

Figure 2. A collision-free path for a tentacle robot manipulator. Real obstacles in the tentacle's position space are bounded by *tentacle obstcales* that look like the ones marked OBS1 and OBS2 here. These obstacle map into the tentacle's configuration space as shown. In the configuration space the vertical boundary *amax* represents the maximum reach of the tentacle in any direction.

Figure 3 illustrates the area swept out by the tentacle as its a value varies while its ϕ value remains fixed. A polygon is shown which closely bounds this area. In the diagram shown call the boundary of the polygon above the x axis the *top* side and those below the *bottom* side. The right side of a tentacle obstacle is a rotated portion of the top side and the left side of a tentacle obstacle is a rotated portion of the right side. This can be seen by comparing the contours of the obstacles in figure 2 with the sides of the bounding polygon in figure 3. These contours correspond to the sides of the configuration space obstacles parallel to the a axis. In other words, they represent limits on the value of ϕ outside of which the manipulator is guaranteed to not intersect the obstacle, for any value of a.

Given the foregoing understanding of the left, right and near sides of a tentacle obstacle it is relatively simple to plan collision-free trajectories for the manipulator around these obstacles, because (ignoring for the moment the far sides) these obstacles map into the configuration space as rectangles (the obstacles in the configuration space of figure 2 without the V-shaped notches on top). There is, however, the additional problem of planning paths so that the manipulator may reach points on or beyond the top side of an obstacle. Examining the tentacle obstacles in figure 2 it can be seen that for a given value of ϕ there might be a range of values of a for which the hand is beyond the top of the obstacle but for which the arm does not intersect the obstacle. As a becomes sufficiently small or sufficiently large the arm will intersect the obstacle, however. These considerations give rise to the V-shaped notches in the configuration space obstacles. The exact relationship between ϕ and a for which the hand reaches beyond the obstacle and the arm doesn't intersect the obstacle is non-linear, but can be conservatively bounded by the linear function represented by the V-shape in figure 2.

A program to map tentacle obstacles into configuration space has been implemented in C. Using this program the collision-free trajectory illustrated in figure 2 was found.

2. "Not Too Near, Not Too Far" Findpath Problem

Considerable work has been done on the problem of finding a collision-free trajectory for a rigid two-dimensional robot moving around fixed obstacles in a plane. A new algorithm called the "Not too Near, Not too Far" algorithm has been developed and implemented. Where this algorithm differs from others previously developed is in its quantitative assessment of the uncertainty in the position of the rover and how that information is used to plan a path for the robot that takes it "not too near" obstacles but "not too far" from them either.

The robot rovers built in the Mobile Robotics Laboratory at C-MU can be viewed as rigid two-dimensional robots moving around in a plane if the robot and obstacles are projected onto the floor. Many algorithms have been developed to solve the findpath problem for this simple case. It can be observed, however, that they all share one or another basic flaw. A two-dimensional findpath algorithm may find a collision-free path for a robot but the path may not be suitable for a real robot either because it brings the robot so close to obstacles that the robot might actually hit them if there is any uncertainty about its position or, coversely, the robot may be sent far out of its way to avoid very small obstacles. The problem then becomes to write a findpath program which keeps the robot "not too near" to obstacles but "not too far" from them.

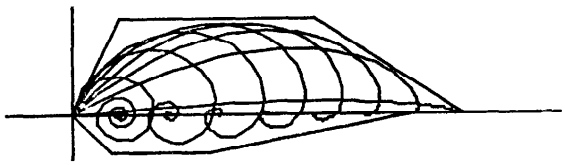

Figure 3. A polygon bounds the region swept out by the tentacle as ϕ is held constant (here $\phi = 0$) and a varies. The "top" sides of this polygon are used to construct the "right" sides of tentacle obstacles and the "bottom" sides are used to construct the "left" sides of tentacle obstacles.

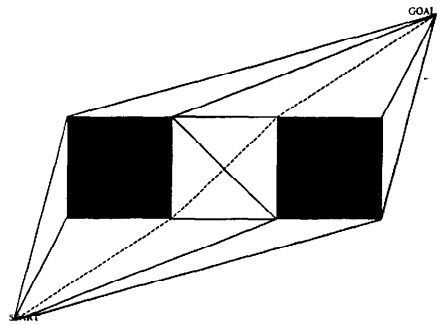

Figure 4. A Path found by a V-graph search algorithm.

One of the earliest findpath algorithms is the *visibility graph* or *V-graph* algorithm developed at SRI[5] for the robot Shakey in the late 1960's. The V-graph algorithm works like this: Given the obstacles are all fixed convex polygons and the robot is a moving point (if the robot is not a point, grow the obstacles by including within their walls all points less than or equal to the radius of the robot and shrink the robot to a point), construct the set of line segments linking all the vertices of the polygons with each other and with the start and goal positions. Delete from this set all segments which intersect polygons (but not the segments lying along the edges of polygons). The remaining segments form a V-graph, which may be searched for a collision-free path from start to goal. Figure 4 illustrates a path found by a V-graph algorithm.

The V-graph algorithm clearly has the property that the robot comes too near the obstacles, in fact it often must follow path right along the walls of obstacles. (Imagine walking through the corridors of a building while staying as close as possible to the walls). The obvious solution to this "too near" problem is to grow the obstacles. But what is a reasonable amount of "growth?" An answer to this question appears below.

A more recent findpath algorithm is the "freeway" algorithm developed by Rod Brooks[6]. In this algorithm generalized ribbons (roughly symmetric polygons with a well-defined axis) are fit to the free space between obstacles. Path-planning for a simple pointlike robot consists of finding the nearest generalized ribbon axis and following a connected set of the axes to a point on an axis nearest the goal. Figure 5 illustrates a collision-free path found by this algorithm.

The freeway algorithm has the unfortunate drawback that for cases where there are few obstacles and they are sparsely distributed the collision-free paths planned may take the robot far out of it's way. (Imagine walking through a gymnasium-sized room from one corner to its diagonal opposite but walking along a long L-shaped path to avoid a small box placed in the center.)

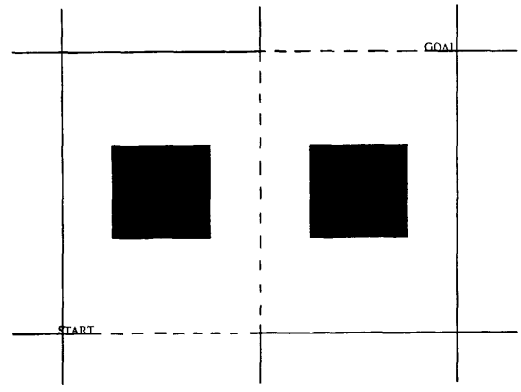

Figure 5. A Path found by a Freeway Algorithm.

The "Not too Near, Not too Far" algorithm has two parts. First, obstacles are grown into *virtual obstacles* such that the obstacles nearer the robot are grown less than those further away, taking into account the fact that the certainty in the position of the rover degrades with distance away from its start position. Second, a search program is applied to find a path around the virtual obstacles.

I assume that the robot is a circle capable of omnidirectional motion in the plane and that the obstacles are convex polygons. This type of robot was selected in part because the Pluto rover in the Mobile Robotics Laboratory is cylinder shaped and omnidirectional. When Pluto and obstacles are projected onto the floor, this algorithm models the situation exactly. It is assumed also that the uncertainty in the position of the robot increases according to a linear function of distance away from the robot's start position. Equivalently, we can say that the size of the robot grows as $R = kd + r$ where R is the radius of the grown robot, d is distance from the start position and r is the radius of the actual robot and k is some empirically selected constant. This assumption is of course only an approximation of the uncertainty in the position of the robot, which actually varies as a function of distance *traveled* by the robot rather than distance from the start. But distance traveled cannot be known before a path is selected so we assume that the uncertainty in the robot's position can at least be bounded by the linear function above. Over the short distances traveled by the robots in the Mobile Robotics Laboratory this assumption is certainly valid.

Given the linear-growth assumption we proceed to expand the obstacles and to shrink the robot. The robot transforms to a point. If an obstacle has an edge e and p is a point along e then we look for a point p^* along a perpendicular to e from p so that the distance from p to p^* is $R = kd + r$ where k and r are as above and d is the distance from the robot's start position to p^*. It is easy to see that unless the wall is aligned with a ray from the robot start position the transformed virtual obstacle wall will have a complicated shape. Fortunately, the virtual obstacle walls can be approximated in the following way: At each vertex of the original polygonal obstacle construct circles of radius $R = kd + r$ where k and r are again as before and d is now the distance from the start position to the vertex. The approximated walls of the virtual obstacle are the outside tangents between these circles. The transformation is illustrated in figure 6.

The second part of the algorithm involves searching for a collision-free path around the transformed obstacles. It is easily seen that if the shortest collision-free path lies along segments linking the start and goal positions with the obstacles such that the segments are tangent to the circles at the vertices of the obstacles (this can be proved with a string-tightening argument), or along a straight-line path from start to goal.

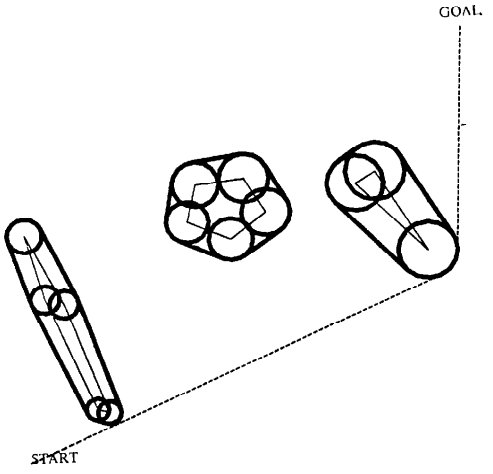

Figure 6. Transformation of an obstacle into a virtual obstacle by "Not too Near, Not too Far" algorithm. The original obstacle is the polygon whose vertices are the centers of the circles. The virtual obstacle is bounded by the circular arcs and tangential edges shown in bold. The dashed line indicates the path selected by the program.

The algorithm proceeds by constructing the tangential line segments and eliminating those which intersect obstacles. A search graph is constructed so that each node represents either a circle or the start or goal. Links in the graph represent tangential segments which may be followed from one circle to another. These tangential segments may be either edges of virtual obstacles or tangents between virtual obstacles. We assume that the robot won't "back up" along a circle, that is, when it reaches a circle by a tangent it will continue around the circle until it reaches a second tangent along which it can smoothly exit. Thus each circle is represented as two circles, a clockwise one and a counterclockwise one. From the clockwise circle the counterclockwise one may not be reached directly, but any other circle may be reached. The resultant search graph may be searched using any of a variety of conventional search procedures.

A version of the "not too near, not too far" algorithm has been implemented in C. The diagram in figure 6 was produced by this program.

3. Two-Arm Findpath Problem

Systems of multiple robot manipulators must be coordinated so that the arms are not always crashing into each other. The general findpath problem for multiple arms is computationally expensive, but in certain special cases very inexpensive solutions may be obtained. Developed here are some theoretical approaches to the multiarm collision avoidance problem, based on some trajectory planning work done by Kamal Kant at McGill University[11]. The ideas were used to implementing a simple two-arm path planner discussed below.

Kamal Kant has done some interesting theoretical work on the findpath problem for a mobile robot in an environment with moving obstacles. He considers the special case of this problem in which the mobile robot's path is fixed in advance, and the trajectories of moving obstacles are known. The parameter in this system is the *speed* of the robot.

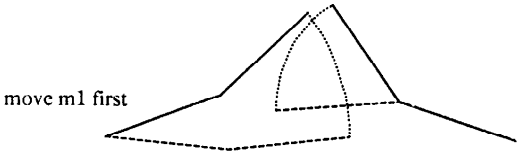

Figure 7(a). The trajectory of two planar manipulators. The manipulators start in configurations given by the solid lines and move to configurations given by the dashed lines. The dotted line follows the trajectory of the hand.

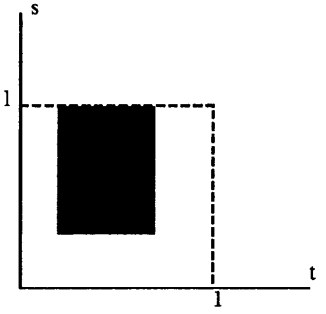

Figure 7(b). The $s \times t$ space constraint corresponding to figure 7(a).

It is easy to visualize the problem by considering a train moving along a fixed track which is being crossed by, say, people and autos. If the velocities of the people and autos are known then we can plan a velocity profile for the train so that it avoids collisions with them.

Applying this idea to the two-arm findpath problem, we begin by fixing the trajectories of each arm. For simplicity, we take these trajectories to be straight line segment paths in joint space (for a 2 joint manipulator such as the one illustrated in figure 7 there is a 2 dimensional joint space). Also, for one of the arms we assume a constant velocity. Thus the joint space path of one arm can be parameterized with a parameter t representing time, such that $0 \leq t \leq 1$. The other manipulator is parameterized with a parameter s, $0 \leq s \leq 1$ so that we can control the speed ds/dt of the second manipulator.

If we now construct the space which is $s \times t$ we can plan the velocity profile for the second manipulator. In figure 8(a) we see two fixed trajectories for each of two manipulators, m_1 and m_2. We fix the trajectory of m_1 and m_2 but allow ourselves to vary the speed of m_2. The two paths cross each other at some initial t_0 corresponding to some s_0 and overlap until t_1 which has a corresponding s_1. This constraint is represented as a rectangular obstacle in $s \times t$ space. The obstacle constrains path from $(s,t) = (0,0)$ to $(s,t) = (1,1)$ so that m_2 must move slowly so that it avoids m_1, or, as the program suggests, that m_1 must complete its motion first.

Unfortunately the problem of computing the actual values of t_0, t_1, s_0 and s_1 is very difficult. The problem can be approximated by considering canonical cases of obstacles in $s \times t$ space (i.e. box against s axis, box against t axis, box against no axis etc.). By classifying a particular problem into one of these canonical types we can find not the exact velocity of the second arm, but at least get an indication of which arm to move first.

A program in Lisp was implemented which plans collision-free paths for the simple two-arm system illustrated in figures 7 and 8. Figure 8 illustrates some of the results of this algorithm.

Of course, fixing the trajectory and varying the velocity of robot arms will not result in collision-free paths in all cases (see figure 8). But it is interesting to consider using an approach such as this as a front-end to some more complete findpath solver. In the cases where the $s \times t$ space approach works, it will find solutions very quickly.

4. Conclusions and Future Work

Three findpath problems were discussed here. Each problem suggested a heuristic for its solution. It may be possible to use these heuristics in the search for efficient solutions to other findpath problems. For a particular findpath problem, we could ask:

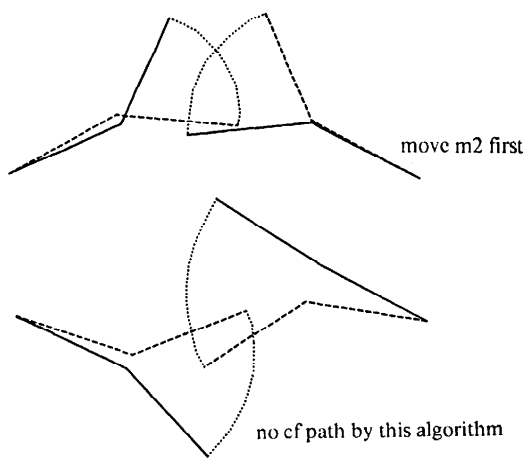

Figure 8. Some motion strategies for the two-arm path planning problem suggested by the program. See the description of figure 7.

1. Does the configuration space naturally generate any "interesting" obstacles? If so, can these obstacles be used to bound obstacles that occur in position space?

2. Can information about the uncertainty in the position of the robot be used to constrain the space in which we search for a cf path?

3. Can the dimensionality of the findpath problem be reduced by searching in spaces other than the actual configuration space of the robot? Can parameters such as the velocity of the robot be used to reduce the size of the search space?

In this article we reported cases in which these heuristics lead to efficient solutions to particular findpath problems. Using the first heuristic it was shown that the tentacle's configuration space contained some easily constructed obstacles which could be used to bound real obstacles in position space. In the rover "Not too Near, Not too Far" program we used the second heuristic to find a manageable search spacein which to search for real-world paths. The third heuristic suggested the solution to the constrained two-arm findpath problem, in which the trajectory paths of each robot is already selected and the speed of the arms is controlled to prevent collisions. More work needs to be done, however, to see if these heuristics are helpful in finding solutions to other findpath problems.

Future work on the particular findpath problems examined here includes solving tentacle findpath problems for more interesting tentacles, experimentally evaluating the performance of the "Not too Near, Not too Far" program on a real robot, and extending the two-arm solution to work for manipulators with polyhedral links.

Bibliography

[1] Lozano-Perez, Thomas *Automatic Planning of Manipulator Transfer Movements* MIT AI Memo 606, December, 1980.

[2] Schwartz, Jacob T. and Micha Sharir *On the 'Piano Movers' Problem I. The case of a Two-dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers* Computer Science Department, Courant Institute of Mathematical Sciences, Report No. 39. October, 1981.

[3] Moravec, Hans P. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover*, Stanford Artificial Intelligence Laboratory Memo AIM-340, September, 1980.

[4] Rowat, Peter Forbes *Representing Spatial Experience and Solving Spatial Planning Problems in a Simulated Robot Environment*, Ph. D. thesis, University of British Columbia, Department of Computer Science, October, 1979.

[5] Nilsson, N. J. and Raphael, B. "Preliminary Design of an Intelligent Robot", *Computer and Information Sciences* vol. 7 no. 13 pp. 235-259. 1967.

[6] Brooks, R. A. "Solving the Findpath Problem by Good Representation of Free Space" in AAAI-82, Proceedings of the National Conference on Artifical Intelligence, pp. 381-386. August, 1982.

[7] Thompson, A. M. "The Navigation System of the JPL Robot" in *Proceedings of IJCAI-5*, August, 1977.

[8] Udupa, Shriram M. "Collision Detection and Avoidance in Computer-Controlled Manipulators" in *Proceedings of IJCAI-5*, August, 1977.

[9] Thorpe, Charles E. *Path Relaxation: Path Planning for a Mobile Robot*, Department of Computer Science, Carnegie-Mellon University, in preparation, 1984.

[10] Hopcroft, J., Joseph, D., and Whitesides, S. "On the Movement of Robot Arms in 2-Dimensional Bounded Regions" in *23rd Annual Symposium on Foundations of Computer Science*, IEEE Computer Society, pp. 281-289. November, 1982.

[11] Kant, Kamal "Trajectory Planning Problems, I: Determining Velocity along a Fixed Path" in *CSCSI-84* (Proceedings of the Fifth National Conference of the Canadian Society for Computational Studies of Intelligence), May, 1984.

[12] Wallace, Richard S. "Three Findpath Problems", extended version of this paper. *forthcoming.*