PATH PLANNING USING A POTENTIAL FIELD REPRESENTATION

Yong Hwang
Narendra Ahuja

Coordinated Science Laboratories
Department of Electrical and Computer Engineering
University of Illinois
Urbana, IL 61801

ABSTRACT

Findpath problem is the problem of moving an object to the desired position and orientation while avoiding obstacles. We present an approach to this problem using a potential field representation of obstacles. A potential function similar to the electrostatic potential is assigned to each obstacle, and the topological structure of the free space is derived in the form of minimum potential valleys. A path specified by a subset of valley segments and associated object orientations, which minimizes a heuristic estimate of path length and the chance of collision, is selected as the initial guess of the solution. Then, the selected path as well as the orientation of the moving object along the path are modified to minimize the cost of the path, which is defined as a weighted sum of the path length, required orientation changes during the motion, and the chance of collision along the path. Findpath problems possessing three different levels of difficulty are identified. Path optimization is performed in up to three stages according to the level of difficulty of the problem. These three stages are addressed by three separate algorithms which are automatically selected. The performance of the algorithms is illustrated on a variety of two- and three-dimensional problems.

1.

1. INTRODUCTION

This paper presents a solution to the findpath problem, defined as follows. Given a space littered with obstacles and a moving object (MO), find a continuous path and orientation connecting the starting position/orientation and the goal position/orientation of MO. Existing algorithms can be broadly classified as being either complete or approximate. Complete algorithms either find a solution or prove that none exists. There appear to be only two such algorithms. In the configuration-space approach [3], MO is shrunk to a point and obstacles are grown correspondingly. A solution is found in the configuration space through search. The critical-curve approach [9] divides free space into regions, each associated with ranges of feasible orientations of MO. A solution is found by identifying a sequence of regions through which MO can move without assuming any infeasible orientation. Each of these algorithms takes a long time to find a solution. In the interest of generating fast algorithms many authors have proposed approximate methods which reduce the problem complexity by approximating the shapes of MO and obstacles by simple ones such as circles and rectangles in two dimensions. Two-dimensional (2D) algorithms based on free-space decomposition are reported in [1, 7, 10], and a three-dimensional (3D) algorithm using an octree representation is described in [4]. Most of these algorithms have two disadvantages. First, the allowed shapes are too restricted for the algorithms to be applicable in general cases. Second, they may fail to find a solution even if there is one.

The approach presented in this paper uses a potential field representation of the obstacles. Several motivations have lead to the use of such a representation. First, it is desired to find a solution path which requires MO to move along a smooth curve with smoothly varying orientation. Many of the existing algorithms use geometric representations of objects which often result in paths with sharp corners. Second, it is desired that the identification of an approximate solution path (e.g. its topological relationships to obstacles) be separated from the details of optimizing its cost. This would reduce computation time by limiting extensive computation to only those places where a solution is likely to exist. Third, the nature of a segment of the path through a given part of the free space should be determined only by the surrounding obstacles, thus, further reducing the computational complexity. Fourth, the allowed shapes of objects should not be greatly restricted. These factors have prompted us to use the potential field representation. The potential field approach can be used to obtain a global representation of the space and to find collision-free efficient paths. A continuous potential field gives a good indication of distance to and shapes of obstacles so that necessary changes in position and orientation of MO can be made in a smooth and continuous manner.

A potential field like representation has been used for related problems, but to a limited extent for path planning. Khatib [6] uses artificial potential repulsion to avoid obstacles locally. Use of potential in manipulator control can be found in [8]. In both of these algorithms, potential is used as a warning of obstacles - like a proximity sensor. In contrast, we use the potential field to plan global paths. Thorpe [12] uses a potential like cost function in designing an optimal path for a circular MO in 2D. Suh and Shin [11] have reported a potential based algorithm to find the optimal path for a point MO in 2D, and given a brief sketch for the 3D case. It is our goal to develop a potential based approach to global path planning for the findpath problem in 2D and 3D.

The potential field approach divides the problem into two stages. First, all topologically different paths between the starting and goal configurations are found, and a candidate path that is mostly likely to yield the shortest collision-free path for MO is selected. Second, three algorithms are used to modify the candidate path to derive the final path and the orientations of MO along the path. Section 2 describes a potential field function and estimation of the topological paths. Sections 3-5 describe three findpath algorithms to derive the optimal paths from the topological paths, and illustrate their performance on a variety of problems. Implementation details can be found in [5]. The last section evaluates our approach, highlights some salient features of the approach, and discusses possible future work.

2. POTENTIAL FIELD AND TOPOLOGICAL PATHS

A natural choice for potential is the Newtonian potential function. Unfortunately, however, an analytic expression of the Newtonian potential is not available even for an arbitrary polygon. A potential function with a simple closed form is described below. Let $g(x) \le 0$, $g \in C^m$, $x \in R^n$ be the set of inequalities describing a region where x denotes the location of a point. Then,

$$f(x) = \sum_{i=1}^{\text{no. of bound. seg.}} g_i(x) + |g_i(x)|$$

is zero inside the region and grows linearly as the distance from the region increases. If the potential function p is defined as $p(x) = [\delta + f(x)]^{-1}$, where δ is a small constant, p resembles the Newtonian potential; $p(x)$ has its maximum value of δ^{-1} inside the region and decreases as the inverse of the distance outside the region. The potential function of a triangular object in 2D is shown in Figure 2.1. When there are multiple obstacles present, the potential at any point is given by the maximum of the potentials due to individual obstacles. It is crucial to use maximum rather than the sum of potentials. When the sum is used as the combined potential, small local maxima of the potential function may appear in free space away from obstacles. It is desired to have local maxima of the potential function only in the regions obstacles are occupying so that all parts of MPV are topologically distinct. The uniqueness of MPV is proved in the following propositions.

Proposition 1. Local maxima of the potential function $P(x) = \mathit{Maximum}\ [p_j(x)]$, where j is an obstacle index, occur only on the obstacles.

proof: The free space can be partitioned into regions such that each region contains exactly one obstacle, and at each point in the region the potential due to the obstacle is greater than the potential due to each of the other obstacles. The resulting partition is similar to the Voronoi partition with a different (potential based, instead of Euclidian) measure of distance. Any local maximum must occur either within some region or at a point on the boundary of some region. It cannot occur within a region since the potential due to each obstacle decreases monotonically as the distance from the obstacle increases. It cannot occur at a point on the boundary either, since the boundaries consist of saddle points or local minima of the potential function.

Proposition 2. Any closed curve (surface) defined by the minimum potential valleys contains a local maximum of the potential function in its interior.

proof: Suppose on the contrary that there is in MPV a closed curve (surface) that does not contain a local maximum in its interior. Then the maximum occurs at a point on the curve (surface). But this is a contradiction since a local minimum of the potential along a line perpendicular to the MPV occurs on MPV.

2.1. Obtaining Topological Structure of Free Space

It is an integral part of our algorithm to classify all possible paths into a finite number of representative, or topologically distinct paths, and examine only these for deriving collision-free paths. MPV lie as far away from the obstacles as possible and thus are good candidates for collision-free, topological paths. This unifies the steps of detecting topological and optimal paths through the use of the same representation. The MPV are represented by a set of nodes, and they are generated as follows.

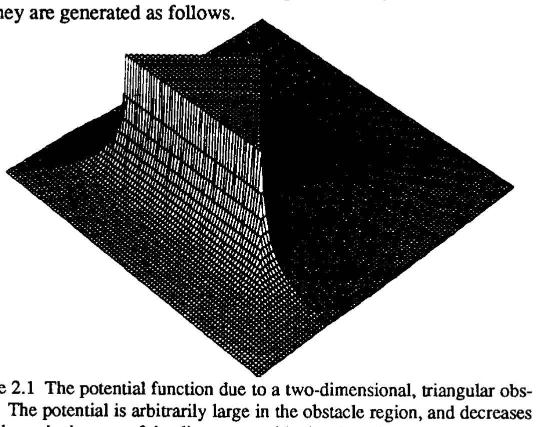

Figure 2.1 The potential function due to a two-dimensional, triangular obstacle. The potential is arbitrarily large in the obstacle region, and decreases roughly as the inverse of the distance outside the obstacle.

MPV in 2D consist of curves, and they are generated beginning from the start position of MO. Initially, a circle of maximum radius contained in the free space is drawn with its center at the start node. The potential along the circumference of the circle is computed at a discrete interval, and the points of locally minimum potential on the circle are labeled as neighbors of the start node. The neighbors are again treated as the start node to generate more neighbors. Successive nodes represents branches of MPV originating at the start node. The same procedure is carried out for the goal node.

MPV in 3D are surfaces, and the search for the local minima has to be done on a sphere rather than a circle. This is computationally expensive. Furthermore, saddle points of the potential on the sphere must be also included among nodes to cover all MPV. The 2D algorithm is modified to overcome these two problems. Beginning from the start and goal nodes, two spheres of maximum radii contained in the free space are drawn. A preset number of nodes, called son nodes, are placed evenly on each sphere. Rather than finding local minima of potential among these nodes, they are ordered by their distance to obstacles (MD). The son node with the largest MD is selected as a point on MPV, and all the son nodes that are within the distance MD from the selected node are deleted from further consideration. Among the remaining son nodes, the node with the largest MD is selected as a point in MPV. The selection is continued until there is no son node left. The same process as used for start and goal nodes is recursively applied to the selected nodes.

2.2. Search for the Best Path in Minimum Potential Valleys

An optimal candidate path selected from MPV should be of the minimum length and least likely to cause collisions between MO and the obstacles. The chance of collision is estimated by a cost function that uses only the width and length of MO. We define the cost function $C = \int w(x)\,|dx|$ where the weighing factor $w(x)$ is defined in Figure 2.2. If MD at x is smaller than one-half MO width, MO cannot go through x and the maximum possible weight is used at x. If MD is greater than one-half the longest dimension of MO, MO can go through x in any orientation. Only length of the path is important in this case, and a uniform weight of 1 is used. The curve for intermediate MD values is a part of a hyperbola, but any curve similar to this could be used. The above choice of $w(x)$ gave a good balance between chance of collision and the lengths of the paths for most of the findpath problems we considered. Once the cost function for each branch is determined, dynamic programming is used to find the minimum-cost path. This path is used by the algorithms developed in Sections 3-5 to determine the final collision-free path and orientations for MO.

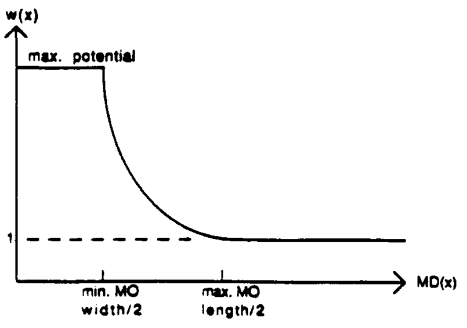

Figure 2.2 A weight function used in heuristics determining the best candidate path.

3. PARALLEL OPTIMIZATION ALGORITHM

The parallel optimization algorithm (POA) finds a path that minimizes a weighted sum of the path length and the total potential experienced by MO along the path. Minimizing the potential on MO favors object motion away from obstacles to avoid collisions, whereas minimization of the path length prevents MO from wandering deep into the free space. The objective functional to be minimized is

$$J = \int_{x_0,\theta_0}^{x_f,\theta_f} [(1+bP(x,\theta))(dx)^2 + a(d\theta)^2] \qquad (3.1)$$

where $P(x,\theta)$ is the total potential on MO at (x,θ). $P(x,\theta)$ generally doesn't have an analytic expression, and is approximated by a sum of the potentials computed at a set of points uniformly distributed over the surfaces of MO. Estimating $P(x,\theta)$ in this way places no restrictions on the shape of MO. The second term in the integrand penalizes the orientation change of MO, a being the relative weighting factor.

The optimal control formulation [2] is used with the gradient algorithm to minimize J. A numerical method starts with an initial guess of the solution and iteratively changes it to decrease the value of the functional J. We use the best candidate path selected from MPV as an initial guess for position of MO. The initial orientations of MO along the path are assigned such that the longest axis of MO is tangent to the initial path. This minimizes the space swept by MO during the motion, and thus the chance of collision on the average.

3.1. Examples

The performance on a range of 2D and 3D problems is described to bring out the capabilities and limitations of POA. Figure 3.1 shows the problem of moving a line segment around the corner of a hallway. The path and orientations shown in Figure 3.1a, which involve collisions, are used as the initial guess. The resulting solution shown in Figure 3.1b avoids the obstacles by making a proper turn at the corner. Figure 3.2 shows a case where intelligent maneuvering is necessary in order to move the L-shaped object through the narrow passage. Although the result is indeed locally optimal in the sense of (3.1), the resulting solution is not collision free.

Figure 3.3 demonstrates the effect of the potential term in the performance index J. In Figure 3.3a, the space between the top and bottom blocks is wide enough for a T-shaped object to go through to minimize the path length. When the space is narrow, the T-shaped object elects to go around all three blocks to minimize the potential. The next example demonstrates the ability of POA to handle initial guesses involving collisions. A rectangular board is to turn the corner of an L-shaped hallway, and the initial path is shown in Figures 3.4a. POA changes the orientation of MO to a semivertical position to make a turn, resulting in a collision-free path of minimum length.

The last example illustrates a case where POA with a simple heuristic to assign the initial orientation fails. Consider the problem of moving an H-shaped block through an H-shaped hole as shown in Figure 3.5. If the initial orientation is assigned so that the longest axis of the H-shaped block lies in the direction of motion, a successful path will not be found. This problem requires a more sophisticated method of assigning the initial orientation along the candidate path. It is therefore necessary to complement POA with other steps so that appropriate perturbations of configurations can be made. Such an algorithm is described in the next section.

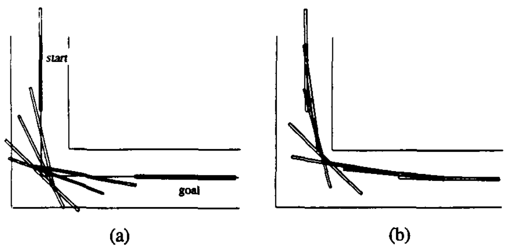

Figure 3.1 Problem of moving a bar around a corner of an L-shaped hallway.
(a) The initial path and orientations.
(b) A collision free path and orientations found by POA.

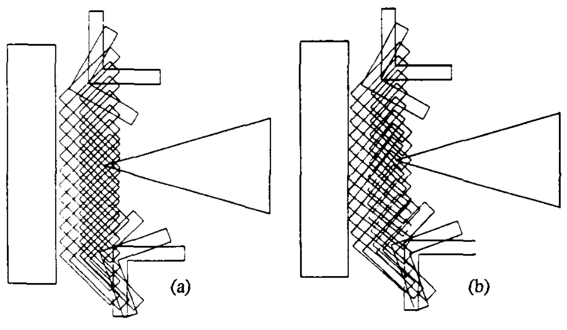

Figure 3.2 The problem of moving an L-shaped object through a narrow channel. An intelligent maneuvering of the moving object is necessary at the bottleneck.
(a) Initial guess of a solution.
(b) Result of optimization. POA fails on hard findpath problems.

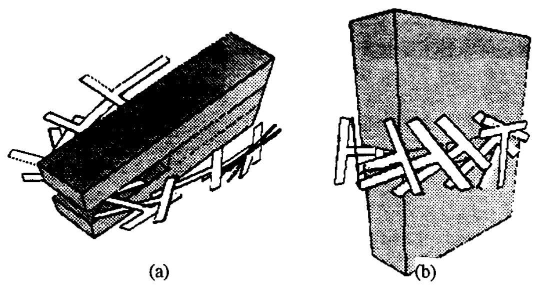

Figure 3.3 A T-shaped object is to move across the stack of three blocks.
(a) When the gap between the blocks is wide, the T-shaped object goes through the gap to minimize the path length.
(b) When the gap is narrow, the T-shaped object goes around all three blocks to minimize the potential.

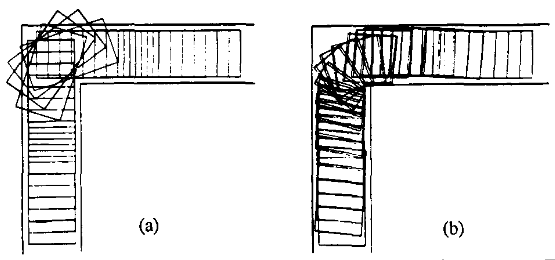

Figure 3.4 A long rectangular board turning a corner of a hallway. The board assumes a semivertical orientation while turning in the narrow corner space.
(a) Top view of the initial path and orientations.
(b) The result of POA viewed from the top.

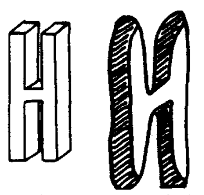
Figure 3.5 An H-shaped block is to go through the wall with an H-shaped hole. This problem requires spatial reasoning to determine the appropriate orientation of the moving object at the narrow region.

4. SERIAL OPTIMIZATION ALGORITHM

SOA first examines the initial path and identifies the parts of the path that may cause collisions. Then minimization of the total potential on MO is carried out in these collision regions locally. Since the minimization is done locally, SOA can take into consideration the shape of MO and the local geometry of the free space much more effectively. SOA divides the task into four steps. First, MO is moved along the candidate path and the collision regions are identified. Then collision-free configurations of MO in these regions are found. The next task is to connect a collision-free configuration of one of the collision regions with a collision-free configuration of an adjacent collision region. If the start configuration and the goal configuration can be connected through a sequence of collision-free configurations in the collision regions, then a solution of the findpath problem follows. If no such sequence is found, then the candidate path is assumed not to lead to a solution. Finally, POA is used to minimize the length of the collision-free path generated by SOA.

4.1. Identification of collision regions

After the best candidate path has been selected, SOA identifies collision regions, i.e., narrow parts of the free space where MO needs to be in specific configurations, different from the initial guesses, to avoid collisions. MO is moved along the initial path with its orientations assigned as in POA. The place in each collision region where MO overlaps with obstacles the most is called the collision center.

4.2. Feasible configurations in collision regions

There are usually an infinite number of collision-free configurations in each collision region, and they have to be grouped into a finite number of topologically distinct configurations. This is achieved by selecting only those configurations which yield locally minimum potentials on MO. In 2D, MO is rotated with its center (reference point) fixed at each collision center, and the orientations of locally minimum potentials are identified. Then for each such orientation the center of the MO is moved around the collision center to further lower the potential on MO. Some of these configurations may still involve collisions, and only those without collisions are considered as the feasible configurations. In 3D the three-dimensional orientation space makes it difficult to classify the collision-free orientations into topologically distinct classes. Without solving this problem in a general way, we use the following ad hoc method in 3D. The longest axis of MO is placed in a finite number of directions, and MO is rotated about the longest axis to find orientations of locally minimum potential. Then the position of the reference point is changed to further lower the potential. Finally, those orientations without collisions are selected as the feasible configurations.

4.3. Connecting feasible configurations

The findpath problem is now transformed into the problem of finding a connected sequence of feasible configurations, one from each hard region, from the start to goal configuration. This requires search. To minimize changes in the orientation of MO between adjacent configurations, those adjacent feasible configurations with the closest orientations are selected first. SOA tries to connect two adjacent configurations of MO by making two replicas of MO in the adjacent configurations move toward each other. In doing so, the initially chosen path from MPV must provide the MOs with the general direction to move. MO is allowed to move within 45 degrees from the initial candidate path. This restriction drives the two MOs toward each other, while allowing MOs to adjust the position and the orientation to minimize the potential locally. After one of the MOs moves one step, SOA checks whether the two configurations can be connected by moving MO in a straight line and changing the orientation at a uniform rate. If this succeeds, the two feasible configurations are connected. If this fails, one of the MOs takes another step closer to the other MO while minimizing the potential. The connecting algorithm signals failure when a MO collides with obstacles, or when the two configurations are still not connected even after the two MOs step through all the nodes on the path between the two configurations.

4.4. Parallel optimization of the result

A solution to a findpath problem found by SOA is not optimal with respect to the objective functional J given in equation (3.1). The result of SOA can be used as an initial path by POA to yield the final path of a minimum length with smoothly changing orientation of MO along the path.

4.5. Examples

All the examples presented in this section contain "narrow" bottlenecks in the free space around which intelligent maneuvering of MO is required. In all the examples the solution paths always lie near the candidate topological paths. The first example is the problem of moving an L-shaped object through three polygons, as depicted in Figure 4.1. When the L-shaped MO is moved along the candidate path, it collides with the obstacles between the two rectangles. SOA finds feasible configurations in this region and connects the start and goal configuration using one of these feasible configurations.

Figure 4.2 brings out several properties of SOA. SOA tries three topological paths before finding a collision-free path and orientation. These three paths are shown in Figure 4.2a. Path I is the shortest path in length but the L-shaped MO cannot make a turn in the space between the triangle and the two small squares. The next best path, Path II, can be used by SOA to transfer MO to the goal location but in a wrong orientation. Although the free space around the goal location is wide enough for MO to rotate, it amounts to a slight sidetracking form the candidate Path II. SOA does not allow sidetracking and abandons this path. SOA does find a collision-free path and orientation along the third topological path, Path III, and the solution is shown in Figure 4.2b. Figures 4.3 shows a bird coming out of its cage. It has to duck under the edge of the roof to come out.

The next example in Figure 4.4 shows a case on which SOA fails. One collision region occurs at the corner of the V-shaped channel, and two feasible configurations (FC1 and FC2) in the collision region are shown in Figure 4.4. The start configuration can be connected to FC1, but FC1 can not be connected to the goal configuration. Similarly, FC2 can be connected to the goal configuration, but the start configuration cannot be connected to FC2. A remedy for this problem is to realize that FC1 and FC2 can be connected by moving them forward into the sharp corner. This amounts to a sidetracking from the candidate path. Such problems are addressed in the next section.

Figure 4.5 shows a tall, skinny, triangular pyramid going through a small triangular opening and a narrow vertical slit. Figure 4.6 shows an L-shaped MO turning a corner in a hallway whose width is much smaller than the length of the legs of the L-shaped MO. The hard region is obviously at the corner. MO has to lower one of its legs to assume a horizontal position at the corner, and then raise the other leg to reach the goal configuration. Figure 4.7 shows

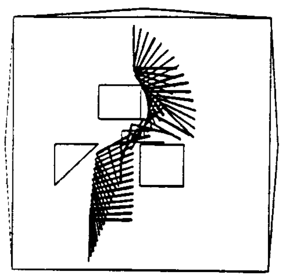
Figure 4.1 An L-shaped object goes through a tight space to minimize the path length. The L collides between the two rectangles when moved along the initial path. SOA finds feasible configurations in this region, and connects the start and goal using one of the feasible configurations.

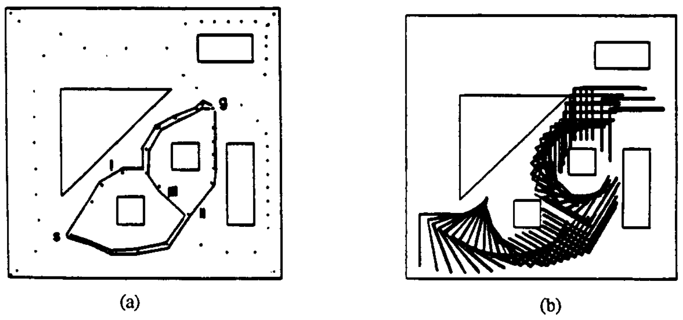

Figure 4.2 Moving an L through rectangular obstacles.
(a) Minimum potential valleys and three topologically distinct paths attempted by SOA.
(b) The solution derived by SOA from the initial guess III. Initial guesses I and II fail to yield a solution.

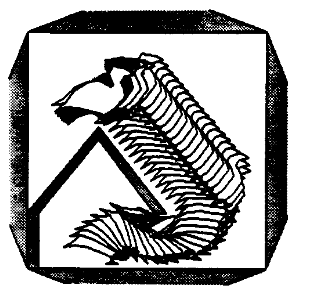
Figure 4.3 A bird has to duck under the roof to come out of the cage.

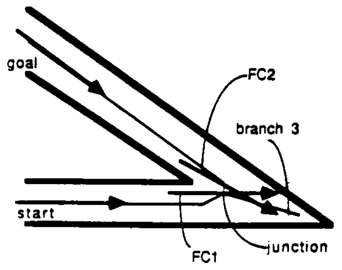

Figure 4.4 A problem for which SOA fails to find a solution.

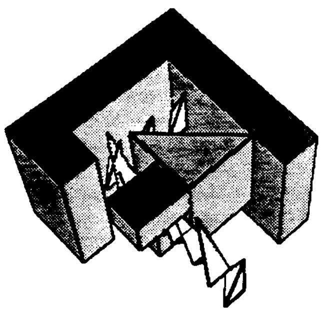
Figure 4.5 A tall tetrahedron tilts to go through a triangular opening, then stands vertically to make a turn.

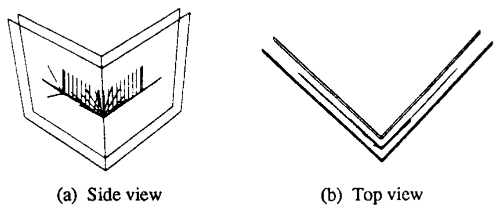

Figure 4.6 An L-shaped object turning the corner of a narrow hallway.

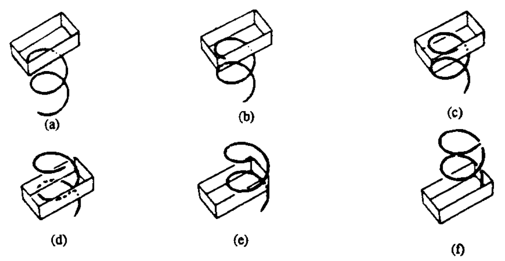

Figure 4.7 A mechanical spring has to rotate several times to go through a hole whose width is smaller than the diameter of the spring.

a helix-shaped mechanical spring going through a small opening in a wall whose width is smaller than the diameter of the spring. The only solution is to rotate MO to screw through the opening. This problem would probably have caused the existing algorithms a great deal of difficulty due to the particular shape of MO. Our method of representing MO with a grid of points on its surface makes the potential field approach tolerate MO of almost any shape.

5. SIDETRACKING ALGORITHM

This section deals with the hardest class of problems that we have considered. These problems require intelligent nonlocal maneuvering of MO. That is, MO may have to take into consideration the geometry of the free space far away from its current location in order to solve the findpath problem at the current location. Figure 5.3 illustrates such a situation. Even humans may exhibit noticeable delay in solving such findpath problems, possibly because a sequential search of the space may be hard to avoid. Let us return to the example in Figure 4.4. Figure 4.4 shows MPV, the start and goal configurations and the feasible configurations (FC1, FC2) in the collision region, which is at the sharp corner. The graphical representation of the problem is shown in Figure 5.1. The solid edges between two nodes mean that the two nodes (or configurations) are connected by SOA, and the dotted edges mean that the nodes cannot be connected. SOA tries to connect the nodes in Figure 5.1b in the order of the integer labels of the edges. SOA signals failure after attempting to connect the start node and FC2. It can be easily seen that the solution to this problem is to connect FC1 and FC2, and then connect FC2 and the goal configuration. Connecting FC1 and FC2 requires MO to move from these configurations toward the corner, following branch 3 of MPV in Figure 5.1a. Connecting two feasible configurations in one collision region involves sidetracking from the initial candidate path. If SOA is equipped with sidetracking capability to connect the feasible configurations in the same collision region, that would help solve the hardest class of the problems we have considered.

5.1. Sidetracking Algorithm

The purpose of sidetracking is to take an excursion from a point P along the initial candidate path only to change the configuration of MO when it is not possible to dc o locally at point P. This could be done by moving MO away from the region by following MPV connected to the region. Thus, the only places where MO is allowed to take excursions from the initial candidate path are the start node, the goal node and the junction nodes on the initial candidate path. It is therefore necessary to find feasible configurations at these places in addition to the collision regions. The regions corresponding to the start, goal and junction nodes will all be called junction regions. The graphical representation of the findpath problem in Figure 5.1b now becomes that shown in Figure 5.2a. Two additional nodes are placed at the top and bottom in order to distinguish the start and goal configuration from the feasible configurations at the start and goal nodes, and to define unique start and goal nodes.

Once the feasible configurations in the junction regions and in the collision regions are found, SOA is applied from the start node. For the example in Figure 5.1a, SOA signals failure after generating the graph in Figure 5.2a. A solid edge between two nodes means that the two nodes are connected by SOA, and a dotted edge means that the two nodes cannot be connected by SOA. The solid edges form a tree structure, which will be referred to as "forward tree." The nodes in the forward tree called "forward nodes," represent the configurations which can be reached from the start node. Upon the failure of forward SOA, STA applies SOA backwards from the goal node. This process spans a "backward tree," consisting of the "backward nodes." Whenever the backward tree reaches a node in a junction region that contains forward nodes, STA tries to connect the backward node with each of the forward nodes in the region via sidetracking. If one of the forward nodes is connected to the backward node, a solution is found. If none of the forward nodes can be connected to the backward node, then STA tries to use some of the nodes that are neither forward nodes nor backward nodes (NFNB nodes) as intermediate nodes in trying to connect the backward node to one of the forward nodes. If all of these attempts fail, then the backward tree is expanded until another backward node is generated in a junction region. This process of spanning the backward tree and sidetracking continues until a solution is found or the backward tree cannot be expanded any more.

Having determined the places to sidetrack and the two feasible configurations to be connected by sidetracking, it remains to determine the exact paths to be used for sidetracking. Since sidetracking is always done from the junction regions, the branches of MPV connected to the junction region provide excellent directions to sidetrack. In case there are more than one branch, the branch leading to a wide part of the free space in the shortest distance is selected first. In 2D, there are a small number of junctions from which to sidetrack. In 3D, however, most nodes have more than two neighbors, and thus are junction nodes. To limit the number of junctions to a reasonable level, only those nodes on the initial path with locally maximum number of neighbor nodes are used as junctions.

5.2. Examples

STA can solve much harder problems than both POA and SOA. In the problem in Figures 5.3, the arc needs to sidetrack first to the wide space at the right, then to the left of the T-shaped junction to reach the goal. Figure 5.4 shows the problem of moving a chair from one side of a desk to the other side. The chair is small enough to go underneath the desk, but has to sidetrack away from the desk so that the seat is between the drawers in the final configuration.

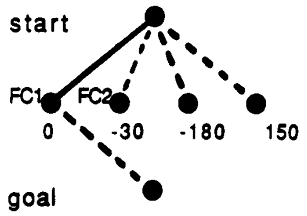

Figure 5.1 The graphical representation of the problem in Figure 4.4

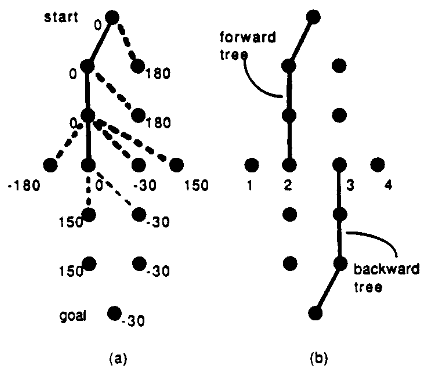

Figure 5.2 The graphical representation of the sidetracking algorithm.

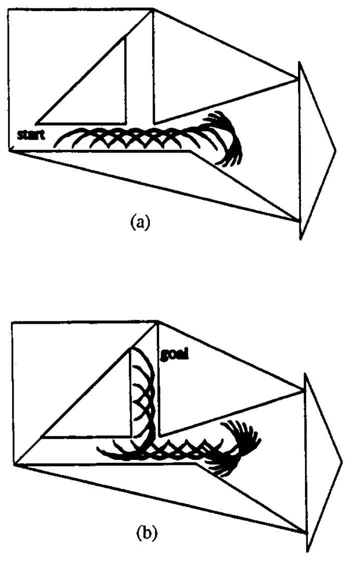

Figure 5.3 The arc needs to sidetrack twice; first, off the junction to the wide space (a), and then, on return, slightly to the left to reach the goal (b).

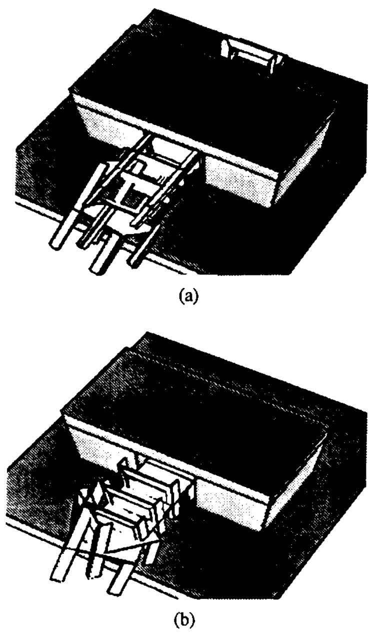

Figure 5.4 The problem of moving a chair from one side of a desk to the other side.