October 1988 UILU-ENG-88-2251

COORDINATED SCIENCE LABORATORY
College of Engineering

PATH PLANNING USING A POTENTIAL FIELD REPRESENTATION

Yong Koo Hwang
Narendra Ahuja

UNIVERSITY OF ILLINOIS AT URBANA-CHAMPAIGN

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS None
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) UILU-ENG-88-2251	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Coordinated Science Lab University of Illinois 6b. OFFICE SYMBOL (If applicable) N/A	7a. NAME OF MONITORING ORGANIZATION National Science Foundation Rockwell International
6c. ADDRESS (City, State, and ZIP Code) 1101 W. Springfield Ave. Urbana, IL 61801	7b. ADDRESS (City, State, and ZIP Code) Washington, D.C. 10552 (NSF) Thousand Oaks, CA 91360
8a. NAME OF FUNDING / SPONSORING ORGANIZATION Rockwell Int. & National Science Foundation 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER ECS 83-52408 (NSF)
8c. ADDRESS (City, State, and ZIP Code) Washington, D.C. 10552 Thousand Oaks, CA 91360	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.
11. TITLE (Include Security Classification) Path Planning Using a Potential Field Representation	
12. PERSONAL AUTHOR(S) Hwang, Y. K. and Ahuja, N.	
13a. TYPE OF REPORT Technical 13b. TIME COVERED FROM _____ TO _____	14. DATE OF REPORT (Year, Month, Day) 1988 October 15. PAGE COUNT 81
16. SUPPLEMENTARY NOTATION	

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	findpath problem, field representation, free space, parallel optimization algorithm, serial optimization

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

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 (MPV). A path specified by a subset of MPV 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 posessing three different levels of difficulty are identified. Path optimization is performed in upto three stages according to the level of difficulty of the problem. These three stages are addressed by three separate algorithms. The first algorithm, called the *parallel optimization algorithm*, solves the simplest of the findpath problems we have considered -- those in which the free space between the obstacles is relatively wide.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT ☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	21. ABSTRACT SECURITY CLASSIFICATION Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE (Include Area Code) / 22c. OFFICE SYMBOL

DD Form 1473, JUN 86 *Previous editions are obsolete.*

SECURITY CLASSIFICATION OF THIS PAGE

UNCLASSIFIED

19. Abstract (continued)

This algorithm fails to yield a collision-free path for problems having the second level of difficulty, where free space contains narrow regions and intelligent maneuvering of the moving object is required to avoid collisions in such regions. A second algorithm, called the *serial optimization algorithm*, finds feasible configurations of the moving object in collision regions, and solves the problem by dividing it into two subproblems around each collision region. These two algorithms still cannot solve problems which require nonlocal geometric analysis to derive local paths. Such problems require moving the object away from the initially guessed solution in order to achieve necessary changes of orientation. A *sidetracking algorithm* solves this third and the most difficult class of path planning problems considered in our work using a three-way divide-and-conquer around each collision region. The performance of the algorithms is illustrated on a variety of two- and three-dimensional problems.

Two principal motivations have lead to the use of the potential field representation. The first is to derive a path that leads to a smooth motion, i.e., the position and orientation of the moving object are smooth functions of time. The second is that any given part of the trajectory should be determined by only those obstacles with which the moving object could potentially collide while traversing that part of the trajectory. There are some further advantages of the use of the potential field representation. It avoids the combinatorial complexity of intersection detection which is necessary when geometric representations of obstacles are used. The computation of the final path is performed effectively through a multiresolution analysis of the free space to reduce the computational effort. This is accomplished through a decomposition of the problem into three coarse-to-fine computational steps: extracting all topologically distinct paths, selecting candidates for the best solution path, and performing optimization. The computation of the potential field is highly amenable to parallel and analog computing. The complexity of extending the approach to solve findpath problems in higher dimensional spaces is marginal.

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 (MPV). A path specified by a subset of MPV 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 posessing three different levels of difficulty are identified. Path optimization is performed in upto three stages according to the level of difficulty of the problem. These three stages are addressed by three separate algorithms. The first algorithm, called the *parallel optimization algorithm*, solves the simplest of the findpath problems we have considered -- those in which the free space between the obstacles is relatively wide. This algorithm fails to yield a collision-free path for problems having the second level of difficulty, where free space contains narrow regions and intelligent maneuvering of the moving object is required to avoid collisions in such regions. A second algorithm, called the *serial optimization algorithm*, finds feasible configurations of the moving object in collision regions, and solves the problem by dividing it into two subproblems around each collision region. These two algorithms still cannot solve problems which require nonlocal geometric analysis to derive local paths. Such problems require moving the object away from the initially guessed solution in order to achieve necessary changes of orientation. A *sidetracking algorithm* solves this third and the most difficult class of path planning problems considered in our work using a three-way divide-and-conquer around each collision region. The performance of the algorithms is illustrated on a variety of two- and three-dimensional problems.

Two principal motivations have lead to the use of the potential field representation. The first is to derive a path that leads to a smooth motion, i.e., the position and orientation of the moving object are smooth functions of time. The second is that any given part of the trajectory should be determined by only those obstacles with which the moving object could potentially collide while traversing that part of the trajectory. There are some further advantages of the use of the potential field representation. It avoids the combinatorial complexity of intersection detection which is necessary when geometric representations of obstacles are used. The computation of the final path is performed effectively through a multiresolution analysis of the free space to reduce the computational effort. This is accomplished through a decomposition of the problem into three coarse-to-fine computational steps: extracting all topologically distinct paths, selecting candidates for the best solution path, and performing optimization. The computation of the potential field is highly amenable to parallel and analog computing. The complexity of extending the approach to solve findpath problems in higher dimensional spaces is marginal.

The support of the National Science Foundation under grant ECS 8352408 and Rockwell International is gratefully acknowledged.

TABLE OF CONTENTS

1. INTRODUCTION

This report presents a solution to the findpath problem. There are many variations of the findpath problem. One of the simplest of them is the classical mover's 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. Humans encounter the findpath problem every day. When we go across a living room or move a newly bought television set into the house, we seem to solve the findpath problem rather easily and often unconsciously. Developing an algorithm that solves the findpath problem, however, is difficult. This is due partly to infinite variety of the shapes of MOs and obstacles, and their spatial configurations encountered in findpath problems. The findpath problem has been studied for many years by scholars in mathematics and engineering.

A successful findpath algorithm must coordinate the two fundamental aspects of the problem: appropriate representation of the space containing the obstacles and MO and derivation of a trajectory for MO and its orientation along the motion path.

1.1. Classification of the Findpath Problem

The findpath problem is usually classified according to the types of moving objects and the available obstacle information. If the configurations of all obstacles are known, optimal solutions can be found before MO starts to move. When the obstacle information is not complete, MO has to explore the environment while designing optimal paths. We will call the first case problems with complete information, and the latter problems with incomplete information. The findpath problem can be time varying or time invariant, according to whether the configurations of the obstacles or the shape of MO are functions of time or not. The findpath problem is said to be conformable if the shapes of the obstacles or moving object can be changed to some extent at the will of the problem solver. If the number of MO's is more than one, it becomes a multiple movers' problem. In some cases, moving objects have to perform additional tasks during the motion from the starting configuration to the goal configuration. Carrying a cup of coffee, for example, demands an extra effort so as not to spill the coffee. The motion of MO is restricted in such cases, and the corresponding findpath problems are said to be constrained. The classical mover's problem defined in Section 1.1 is a time invariant, nonconformable, single moving object and unconstrained findpath problem. Finding a trajectory of a linked robot arm in a changing environment falls in the category of a time varying, conformable, single moving object and unconstrained problem. Two people in an ice arena trying to reach each other is the case of a time varying, conformable, multiple moving object and constrained findpath problem. The complexity of the algorithms to solve these various problems is, as one can imagine, tremendous. The complexity increases rapidly as the number of degrees of freedom increases[1]. In general, the complexity of the findpath problem has been found to be exponential in the number of degrees of freedom, and polynomial in the number of obstacles. As will be discussed later, the only known complete polynomial algorithm for the classical mover's problem is that of Schwartz and Sharir [SCSH83a]. Unfortunately, the algorithm takes $O(n^{(2^{d+1})})$ time where n is the number of edges of the obstacles and d is the number of degrees of freedom. For the classical mover's problem in three dimensions where $d=6$, corresponding to the 3 translational and 3 rotational degrees of freedom, this becomes $O(n^{4096})$. As a result of this time complexity, many efforts have focused on developing approximate but fast algorithms. In the next section, a survey of the previous work on the findpath problem is presented.

1.2. Review of the Previous Work

Most of the past work on the findpath problem is devoted to the classical mover's problem or much simplified versions of it. Two exceptions are the papers by Schwartz and Sharir [SCSH83b] and Erdmann and Lozano-Pérez [ERLP86]. Schwartz and Sharir have considered the task of moving several circles among polygonal obstacles in the plane, and Erdmann and Lozano-Pérez have investigated the movements of multiple hinged objects in the plane. The work reported here concerns the classical mover's problem.

Findpath algorithms are classified as either complete or approximate. Complete methods are aimed at guaranteeing a solution if there is one or proving that there is no solution. These algorithms are useful for very hard findpath problems, e.g., when the space is cluttered densely with obstacles and MO is bulky, and it takes intelligent maneuvering to move an object to the destination. At present, there appear to be only two complete algorithms

[1]The degrees of freedom is defined as the number of variables necessary to completely specify the configuration of MO. For example, it requires seven numbers to specify the configuration of two bars connected at their ends by a hinge in three-dimensional space: three numbers to specify the position, three numbers to specify the orientation, and one to specify the angle between the links.

[DONA84, SCSH83a]. Approximate algorithms make simplifications at the representation level in order to generate fast algorithms. For example, objects can be represented by circles (spheres) or ellipses (ellipsoids) circumscribing them. Or, the motion of MO can be restricted to pure translation, or to a mutually exclusive temporal interleaving of translation and rotation. Approximate approaches are attractive for the findpath problems where the free space between obstacles is wide and generous avoidance of obstacles suffices without tight maneuvering through them. These algorithms are much faster than complete algorithms. In the next three subsections the two complete algorithms and various approximate algorithms are presented.

1.2.1. Configuration obstacle approach

The idea of a configuration obstacle first appeared in the paper by Lozano-Pérez and Wesley [LPWE79] in which the findpath problem in two dimensions is considered. In this approach, MO is shrunk to a point, and the obstacles are grown correspondingly as shown in Figure 1.1. The grown obstacles are called configuration obstacles because the obstacles have to be grown differently according to the orientation of MO. The original problem is then transformed to a point navigation problem if MO is not allowed to change orientation. This approach has been extended to a complete algorithm for the findpath problem in two dimensions by Brooks and Lozano-Pérez [BRLP83]. When MO is allowed to rotate, the configuration obstacles vary with the orientation of MO. Lozano-Pérez and Brooks introduced a *configuration space* in which the two-dimensional configuration obstacles are plotted against the orientation of MO. The configuration space is then divided into rectangloids, and each rectangloid is labeled either full, mixed or empty depending upon whether the rectangloid is completely, partially or not occupied by the configuration obstacles. Then the search is done on the rectangloids to find a connected sequence of empty rectangloids between the starting and goal position and orientation. When no such sequence is found, the rectangloids that are most likely to yield sequences of empty subrectangloids are divided. The process of searching and dividing continues until it finds a solution or the size of the divided rectangloids reaches a preset limit.

The generalization of this configuration space approach to three dimensions is carried out by Donald [DONA84]. Objects are assumed to be polyhedral. The dimensionality of the configuration space is six since the specifications of position and orientation require three numbers each. After the generation of the configuration space a six-dimensional lattice of points is thrown over the space. Each point in the lattice represents a small neighborhood in the configuration space. The search space is very large, and a conventional search strategy would take a long time to find a solution. Donald uses several heuristics to speed up the search. This algorithm is complete at a given resolution of the lattice, but it has not been proven that it is a polynomial time algorithm.

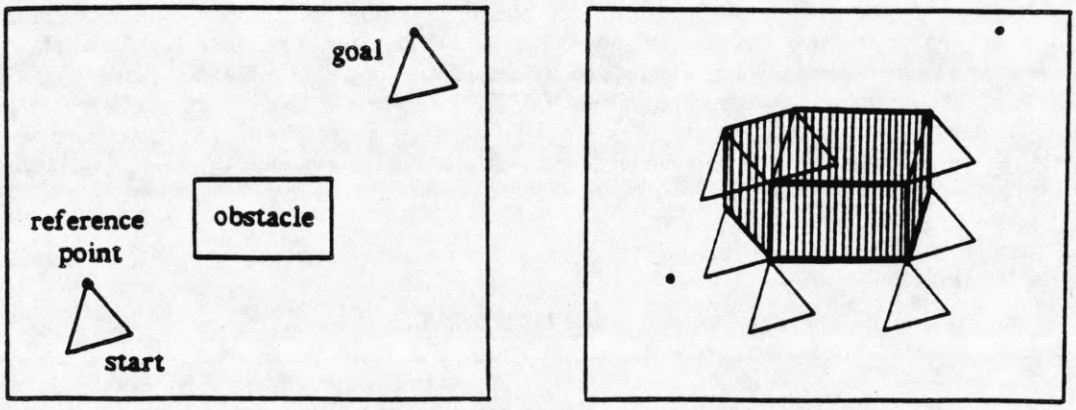


Figure 1.1 The original problem and the configuration obstacle (hatched region).

1.2.2. Critical curve approach

A resolution independent algorithm for two dimensions is developed by Schwartz and Sharir [SCSH83a] which is also complete. All objects are assumed to be polygons. The free space is partitioned into subregions called *non-critical* regions by *critical curves*. A critical curve is the curve that the reference point of MO traces while MO maintains a *critical* contact with obstacles. Critical contacts are defined in many ways, for example: a vertex or an edge of MO touches a vertex or an edge of an obstacle and the reference point is as far from the obstacle as possible, or MO touches two obstacles simultaneously. Figure 1.2 shows three types of critical curves traced by reference point P while the line segment PQ maintains a critical contact. Schwartz and Sharir show that for a polygonal object moving among polygonal obstacles in the plane, there are eight types of critical curves possible. These are generated by keeping in contact edges or vertices of MO with edges or vertices of obstacles while tracing the curves. If the reference point of MO lies in a certain noncritical region, the feasible orientations of MO can be expressed as a union of a finite number of open sets of angles associated with the noncritical region. Each open set is bounded by the angles at which MO is making contacts with some obstacles. The open sets are represented by the pair of edges/vertices in contact. For example, for line segment PQ in Figure 1.3 feasible orientations are given by (θ_1, θ_2) and (θ_3, θ_4). These open sets of angles are represented by the edges of the obstacles making contacts, namely, $[E_2, E_1]$ and $[E_1, E_2]$. The pairs of edges denoting the open sets of the feasible orientations stay the same as long as the reference point of MO stays in the same noncritical region. This implies that MO can move from a point to another in the same noncritical region if the orientations of MO at the two points are in the same open set. MO can move from one noncritical region to another if at a point on the boundary there exists between the two regions an orientation of MO feasible in both regions. Two regions are said to be connected if such a point is found. The connectivity of the regions is then represented in a graph called the connectivity graph, and the findpath problem is transformed to one of finding a sequence of connected noncritical regions in the connectivity graph. The time complexity of this algorithm is $O(n^5)$ in the plane. In [SCSH81], Schwartz and Sharir give a general framework for the findpath algorithm with arbitrary degrees of freedom. The time complexity of the algorithm is $O(n^{(2^{d+1})})$, where n denotes the total number of obstacle edges and d is the number of degrees of freedom. In three dimensions (d=6), this becomes $O(n^{4096})$. This algorithm serves as the existence proof of a polynomial time algorithm.

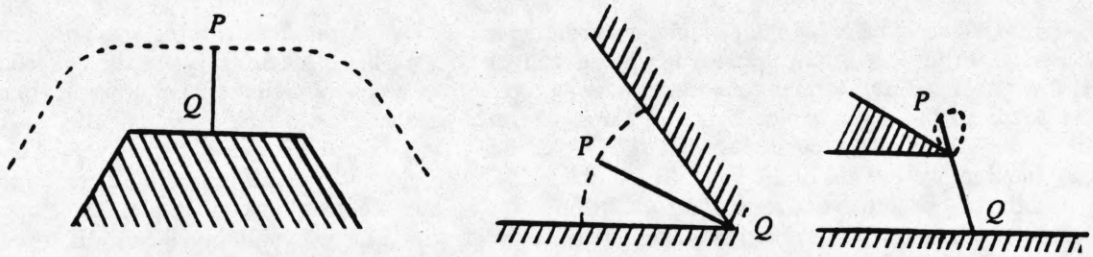
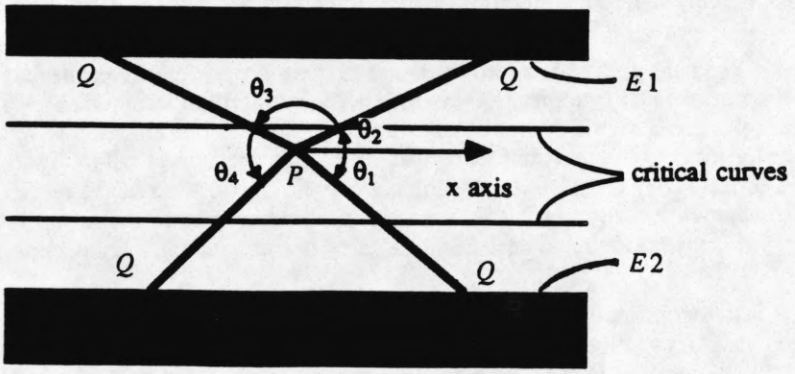


Figure 1.2 Critical curves
(a) A critical curve P traces while Q is touching an obstacle and P is farthest away from the obstacle.
(b) A critical curve P traces while Q is fixed at a vertex of an obstacle.
(c) P traces another critical curve while Q is touching an obstacle edge and an obstacle vertex is touching the segment PQ.



feasible orientation	edge representation
(θ_1, θ_2)	$[E2, E1]$
(θ_3, θ_4)	$[E1, E2]$

Figure 1.3 Open sets of angles denoting feasible orientations can be represented by the pairs of the edges of the obstacles the line segment PQ touches when the orientation is set to the bounding angles.

1.2.3. Approximate algorithms

There are numerous approximate algorithms for the findpath problem. Most of them are developed for two dimensions. Extension of the algorithms to three dimensions does not seem trivial except for a few algorithms. All the algorithms reviewed in this section are in two dimensions except that of Herman [HERM86]. Simplification of object shapes and restrictions on the motion are ways in which approximations are introduced. This subsection gives a brief survey of the approximate findpath algorithms.

Brooks [BROO83] represents two-dimensional (2D) free space as a union of possibly overlapping generalized cones. The algorithm translates a polygonal moving body along the axes or spines of the generalized cones and rotates it at the intersections of the generalized cones. This algorithm is fast and yields paths that avoid obstacles generously. Kuan and Brooks [KZBR85] later improved the quality of the paths found by representing the 2D free space as a union of generalized cones and convex polygons.

Nguyen [NGUY84] developed a fast heuristic algorithm for planning collision-free paths of a moving robot in a cluttered planar workspace. The free space is described as a network of linked cones. Feasible positions and orientations of MO within the cone are computed. Feasible path segments are derived by local experts which use adjacency information of linked cones to generate local paths. Five local experts are used, namely, traversing a free convex region, sliding along an edge, circumventing a corner and going through a star-shaped region. A* search is used to find global paths from local segments.

Moving a ladder or a line segment among iso-oriented rectangular obstacles is considered by Maddila [MADD86]. The problem of moving a ladder is decomposed into several local motion planning problems. The free space is divided into corridors and junctions. Corridors are the "hallways" between rectangular obstacles, and junctions are the areas where corridors meet. The movement of the ladder is either horizontal or vertical, and rotations are done at L-shaped junctions. Because of the iso-oriented nature of the rectangular obstacles, the maximum length of the ladder that can move through an L-shaped channel is easily calculated (see Figure 1.4). A weighted graph called motion graph is constructed from the solutions of the local subproblems. The weights represent the longest ladder that can be moved between the nodes of the motion graph. The algorithm finds a path in $O(nlogn)$ time, and is also capable of finding the longest length of the ladder movable between two positions in the free space.

Chew [CHEW85] reported an $O(n^2logn)$ time algorithm to solve the problem of moving a disc among nonintersecting polygons. In this algorithm the moving disc is convolved with the obstacles to yield the *convolution diagram*, and the problem is transformed into that of moving a point in the convolution diagram. The convolution diagram corresponds to the configuration space in Lozano-Pérez's approach. A visibility graph (also used by Lozano-Pérez [LPWE79]) is then generated consisting of those lines connecting the vertices of the convolved obstacles that do not intersect the interior of the convolved obstacles. The problem of translating an ellipse among elliptic obstacles is studied by Oommen and Reichstein [OORE86]. The ellipses are not allowed to rotate. The ellipse to be moved is transformed into a circle and the elliptic obstacles are grown correspondingly into larger ellipses. The grown obstacle ellipses are then replaced by circumscribing octagons. Octagons are shown to approximate the ellipses sufficiently closely. The search for a path is done on the visibility graph generated from the octagonal obstacles. This algorithm is of time complexity $O(n^2)$ where n is the number of the obstacles.

de Rezende et al. [RLWU85] studied the problem of finding a shortest path in the Manhattan or L_1 distance. The obstacles are iso-oriented rectangles and MO is a point. The essence of the algorithm is the fact that the shortest path between two points in the free space is monotone in either vertical or horizontal direction as illustrated in Figure 1.5. This algorithm is useful for planning bus routes in cities or finding paths for electrical connections on the circuit boards.

Singh and Wagh [SIWA86] decomposed the free space into maximal convex shapes called prime convex areas, which have an analogy to the prime implicants in logical expressions. Their algorithm is applicable to the findpath problems with iso-oriented rectangles and a moving point object. The free space is partitioned into rectangles using the edges of the rectangular obstacles, and the maximal rectangles of the free space are found. Figure 1.6 shows the maximal rectangles with bold edges. They associate binary logical expressions to each of the rectangles in the free space such that the maximal rectangles have the logical expressions corresponding to the prime implicants. The connectivity graph of the prime convex areas is constructed to carry out the search for the minimum length paths.

Herman [HERM86] developed a fast algorithm for collision-free motion planning for a robot in three-dimensional (3D) space. The robot and its swept volume paths are approximated by primitive shapes such as spheres or cylinders, and the search for a collision-free path is done on the octree representation of the free space. Three search techniques, hypothesize and test, hill climbing and A* search, are used to generate the solution. First, an octant closest to the goal location is selected among the neighboring octants containing the start location. Then the algorithm hypothesizes that a collision-free path is given by the straight line connecting the center of the

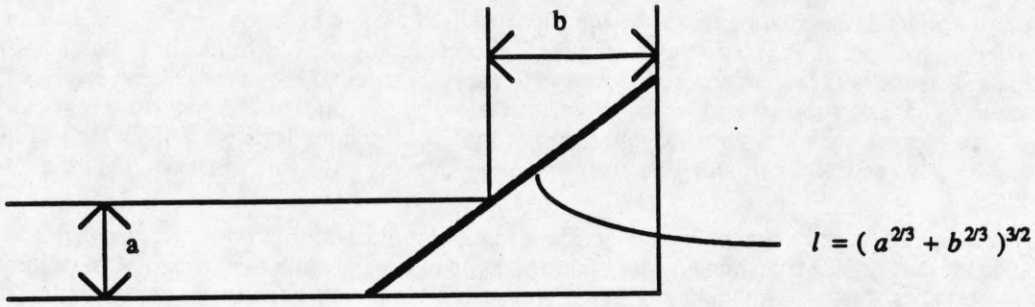


Figure 1.4 The maximum length of a line segment that can turn a corner.

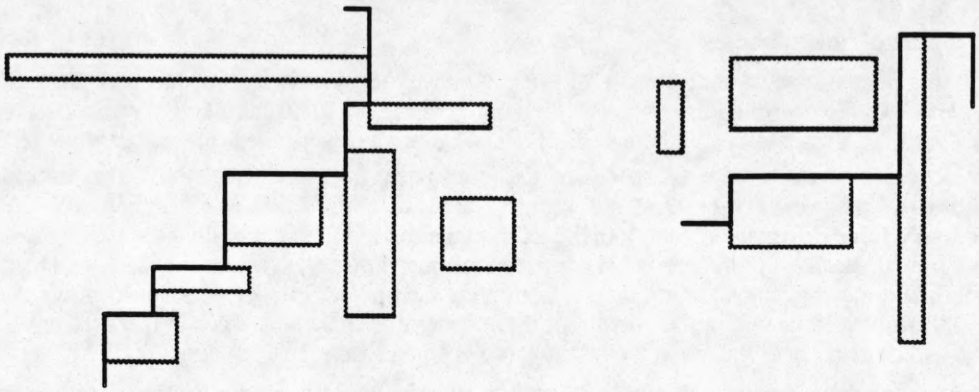


Figure 1.5 In Manhattan or L1 space, the shortest path is monotone in either vertical or horizontal direction.

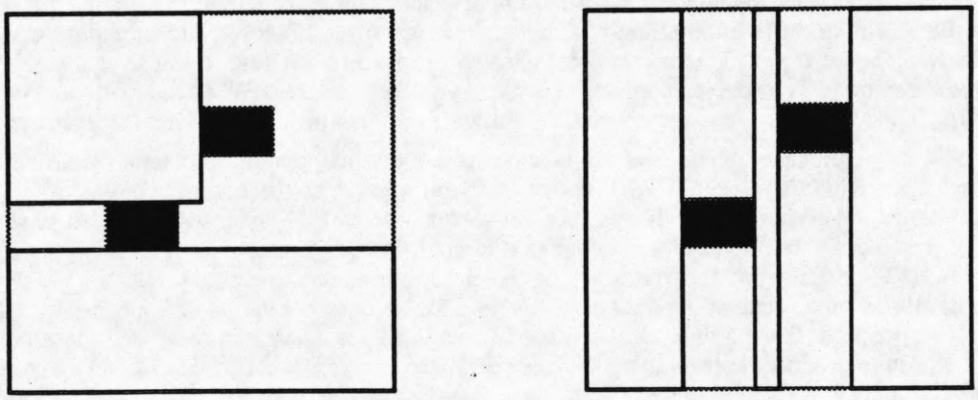


Figure 1.6 The rectangles drawn in solid lines represent some of the prime convex areas. The shaded rectangles are the obstacles.

selected octant and the center of the octant containing the start location. The algorithm then approximates with spheres and cylinders the volume swept by the robot while the robot is following the straight-line path and tests whether the swept volume intersects any of the octants occupied by the obstacles. If an intersection is found, the algorithm selects the next best neighboring octant of the start location. If the swept volume does not intersect any of the octants occupied by the obstacles, the robot moves along the straight-line path to the selected neighboring octants. This process continues until the robot reaches the goal location. It may happen that this hill climbing way of approaching the goal location may lead the robot to a dead end. A* search is used to get the robot out of such situations.

As stated earlier, these approximate algorithms make simplifications in object representation and search for a solution on various graphs containing information about the topology of the free space. Most of the fast algorithms represent obstacles in some configuration space, and search for a path is done on graphs similar to the visibility graph. They result in algorithms of time complexity of $O(nlogn)$ or $O(n^2)$.

The work on path planning in uncertain environments are restricted to point-robot navigation in 2D [CHAR85, CHAT85, KOCH85]. All of them use world maps that are updated as the robot explores the environment. Heuristic strategies are used to plan paths from the partial information of the world using the world map. They often avoid unknown terrains by assigning a large cost to such areas.

1.2.4. Path planning of linked bodies

Path planning of one type of conformable objects, namely, linked rigid bodies, has been studied extensively because of its application to manipulators. A typical manipulator has six joints, allowing the end effector to be positioned and oriented in 3D work space without restrictions. Although this problem has six degrees of freedom, the nonlinear transformation between the work space and the joint space[2] poses a difficulty. The research on manipulator trajectory (or path) planning can be classified into two areas. The first area is concerned with the problem of computing necessary force or torque at each joint so as to make the manipulator follow the nominal (or given) trajectory closely. Much research in this area is concerned with control theory. The second area is concerned with the design of the optimal trajectory to perform a task, e.g., moving the end effector to a certain location in the presence of obstacles. This problem falls within the domain of interest in this paper. The work in this area has lead to only a limited success. A brief survey of papers in the second area is given below.

Lozano-Pérez uses the idea of configuration space to plan paths for a general n-link manipulator [LOPE87]. The free space is represented in terms of the joint variables in a recursive fashion. First, all the other links except the first link are neglected, and the collision-free ranges of the first joint variable are found. Then for each of the collision-free ranges of the first joint variable, collision-free ranges of the second joint variable are found. This procedure is repeated until collision-free ranges of all the joint variables are found. The parts of free space that can be reached from each other are then represented by a connectivity graph of the collision-free regions in the joint space. Finally, A* search is used to find an optimal path.

Lumelsky has developed an algorithm for motion of planar 2-link manipulators among a priori unknown obstacles [LUME87]. The obstacle information is acquired from touch sensors located throughout the manipulator surface. The movements of the manipulator are restricted to those corresponding to linear changes in the joint variables, or those keeping parts of the manipulator in contact with the obstacles. When the manipulator hits an obstacle while traveling in the free space, it slides around the obstacles maintaining contacts with the obstacle.

The obstacle avoidance problem is formulated as an optimal control problem by Gilbert and Johnson [GIJO85] and Chen and Vidyasagar [CHVI87]. In their formulation of the optimal control problem, Gilbert and Johnson have added the constraint that the distance between any pair of the links and obstacles must be greater than some preset threshold. A fast algorithm is developed to compute the distance between two polyhedra. Chen and Vidyasagar [CHVI87] transform the obstacles in the 2D work space to obstacles in the joint space that are represented as ellipses. A couple of hundred of points are used to represent the set of joint variables that cause collisions in a 2D work space. Then a numerical method is used to fit a smallest ellipse around these points in the joint space. This ellipse represents the forbidden range of the joint variables, and is easily incorporated into the optimal control formulation.

---

[2]Work space is the 3D space in which the manipulator is operating, and the joint space is the space of joint angles between the links of the manipulator.

1.2.5. Artificial repulsion approach

The algorithms by Thorpe and Khatib make use of artificial potential repulsion as the means to avoid obstacles. Obstacles are assigned a potential function which is used to compute repulsive force when MO approaches them. Thorpe [THOR84] has used potential function to design an optimal path for a circular MO in 2D. In this algorithm a grid of points is drawn over the problem space, and each point on the grid is assigned a cost inversely proportional to the distance to the obstacles, which is estimated by a repulsive force generated by a potential function. Then A* search is used to compute the path along the grid with optimal length and cost. This path is then modified by a relaxation method to improve efficiency.

Khatib [KHAT85] has used an artificial potential repulsion approach to avoid imminent collisions among robot arms and obstacles. This algorithm is aimed at the local, short-term avoidance of obstacles in real time for moving robot arms rather than planning good global paths. Although the algorithm does not quite solve the findpath problem, the use of repulsion force makes this algorithm different. The repulsion force is generated by a fictitious potential field around each obstacle due to a potential assigned to it. When any link of the robot arm approaches an obstacle, a repulsive force pushes the link away from the obstacle. The potential function used, P, is a function of the minimum distance, D, between the link and the obstacle. P becomes a large value as D gets smaller, and becomes zero beyond a preset distance from the obstacle. The force on MO is calculated from the equation

$$F = -\frac{dP}{dD}\frac{dD}{dx}$$

where x is the position vector of MO. There is a higher-level trajectory planner to generate an approximate path needed to perform a particular task. Then appropriate torques at the joints of the robot to follow the path are computed. The force from the artificial potential field is then incorporated with the above torques to generate the final forces at the joints. This allows each link of the robot to follow the initially planned path as closely as desired while avoiding the obstacles. The role of the artificial potential field is not to plan the trajectory, but only to "bend" a given trajectory around obstacles. Khatib's algorithm is significant in that the local obstacle avoidance problem is implemented at low (control) level for real-time execution, instead of being incorporated as an integral part of the trajectory planner.

1.3. A Potential Field Approach

1.3.1. Motivation

Conventional approaches to findpath problems use volumes occupied by objects or the surfaces of objects to represent the objects. Some other approaches use shape primitives to represent the free space between objects rather than the objects themselves. In contrast, the potential field approach uses a scalar function to describe the objects and the free space. The main advantage of the potential field representation is that the potential field gives good guides as to which objects are near by and how MO should move to avoid collisions with them. More specifically, the negative gradient of the potential field is precisely the direction to move to avoid the obstacles. Other approaches may detect collisions easily, but may not find a way to avoid collisions as easily. The safety margin between MO and the obstacles is also an important factor in object manipulation. In many practical applications, a large safety margin is often preferred to a shorter path length. This may be either because MO can be moved faster in a wide free space, or the situation cannot tolerate even a slight collision. The local potential hills surrounding the obstacles can be used to control the safety margins. All the above advantages are offered by a single potential field, and yet no information about the obstacles and the free space between them is lost.

The potential field approach can, in fact, be used to obtain a global representation of the space and to find collision-free, efficient paths. In this paper we describe an algorithm that uses the potential field of the objects to estimate candidate, coarse global paths of MO, and then modifies these paths to improve their efficiency. Our algorithm solves 2D and 3D findpath problems for arbitrarily shaped MO's. First, we present a summary of our approach.

1.3.2. Overview of Potential Field Approach

The artificial potential approach is motivated by the electrostatic repulsion between like charges. Imagine that all the obstacles are composed of positively charged matter. If MO is also positively charged, the obstacle avoidance problem is resolved by the repulsive force. This force can be calculated as the negative gradient of a potential field. The potential field approach divides the problem into two stages. First, all possible 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 orientations of MO.

The best candidate path is found as follows. First, a scalar potential field is computed due to all charged objects. Minima of the local potential field are identified. These minima capture the topological structure of the free space. All distinct paths between the starting and goal configurations are identified along the valleys of potential minima. The minimum potential valleys (MPV) represent possible paths for point objects and serve as initial estimates from which the final paths/orientations of MO are derived. The best candidate path is found from MPV using a cost function and dynamic programming. The cost function is based on the length of the paths and the width of the free space along the paths. The cost goes to infinity when the width of the free space is smaller than the width of MO, and decreases as the width of the free space becomes larger. After the cost is assigned to all the paths in MPV, dynamic programming is used to compute the shortest path with the minimum cost. The selected path serves as the initial estimate of the solution, and is used by findpath algorithms to derive the final path and orientation of MO. If the initially chosen candidate path does not yield a solution, the parts of the candidate path where MO collides with some of the obstacles are assigned infinite costs. Dynamic programming is run again to find the next best candidate path, and this path is used by the findpath algorithms.

The final paths are desired to be short and collision-free, and they should minimize the change in object orientation as it moves along the path. In problems where the free space between obstacles is wide, the final path and orientation of MO can be obtained by simply changing the initial candidate path away from the obstacles MO is colliding with (Figure 1.7a). In other cases, the simple strategy of "running away" from the obstacles may not result in a collision-free path and orientations of MO. The initial estimate may have to be modified significantly to derive the final path, and the final path may require a complex change in the configuration of MO as it moves along the path (Figure 1.7b). In still other cases, the initial estimates may even have to be changed topologically to obtain the final path (Figure 1.7c). These three levels of difficulty of the findpath problem have led us to use three different optimization algorithms. Given an initial estimate, the final path can be derived by the parallel optimization algorithm (POA), or in more difficult cases by the serial optimization algorithm (SOA), or in the third case by the sidetracking algorithm (STA).

The parallel optimization algorithm employs a numerical method to minimize a weighted sum of the path length and the total potential experienced by MO along the path. The minimization of the potential pushes MO away from the obstacles, whereas the minimization of the path length keeps MO from wandering in wide parts of free space. This algorithm solves the easiest class of findpath problems. If some parts of the free space are so narrow that MO must approach the narrow spaces with specific orientations, POA does not yield a collision-free path and orientations. The serial optimization algorithm moves MO along the candidate path, and identifies the narrow parts of the free space. SOA then finds collision-free configurations of MO in these regions and tries to connect the start and the goal configuration through a sequence of intermediate collision-free configurations, one corresponding to each narrow region. Thus the global path is synthesized from the segments connecting successive intermediate configurations. The third and the hardest class of the findpath problems that we have considered requires excursions from the initially estimated paths to derive the final paths, which even humans need more than several seconds to solve (Figure 1.7c). The sidetracking algorithm allows MO to sidetrack from the candidate path to get proper changes of the orientation of MO. These three algorithms are tested on many examples, and they appear to obtain satisfactory paths in a variety of cases.

Section 2 describes a potential field function and estimation of the topological path. Sections 3-5 describe the three findpath algorithms to estimate final paths and illustrate their performance on a variety of problems. The description of each algorithm is separated from the implementation details, which are given in separate subsections.

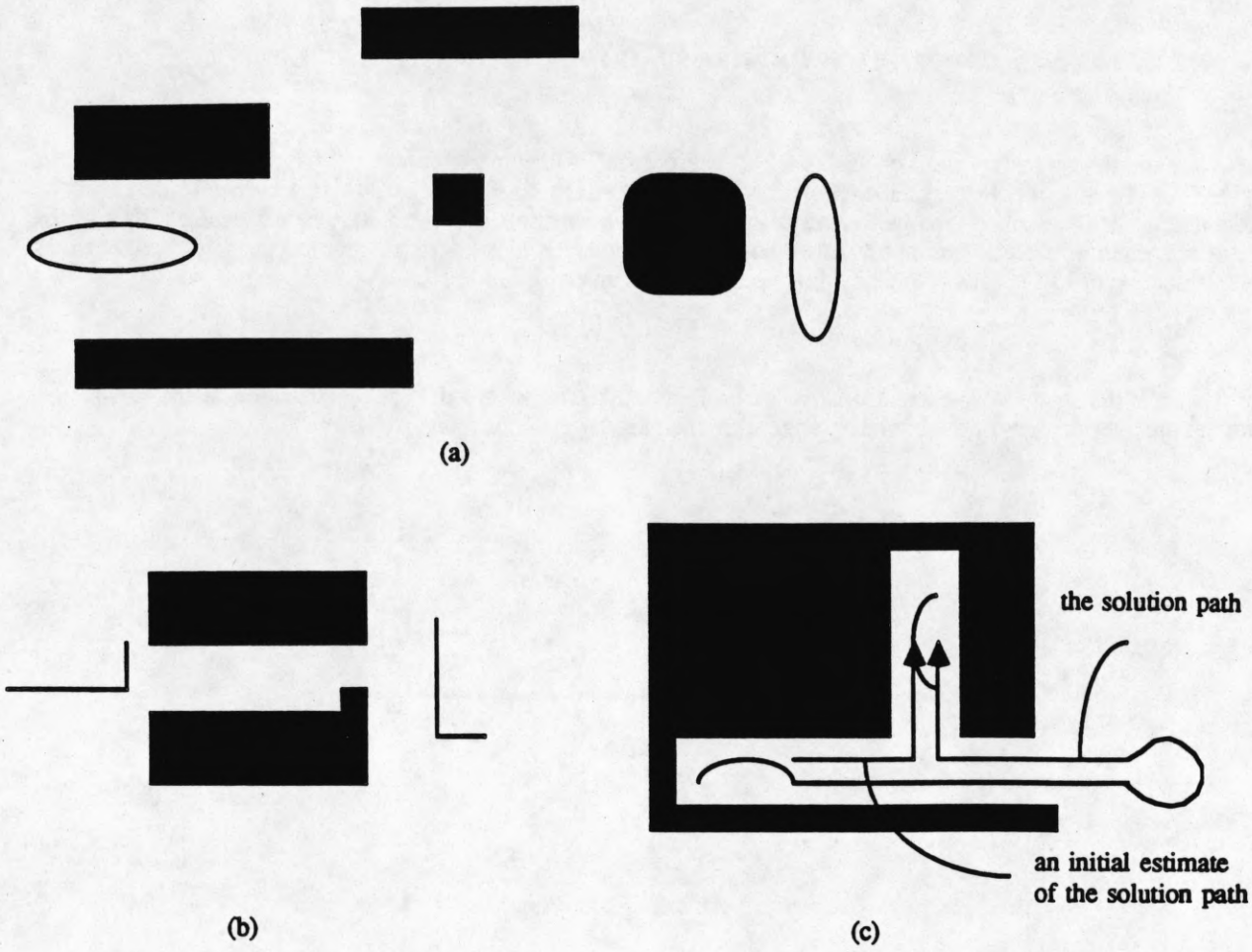


Figure 1.7 Three findpath problems with different levels of difficulty.

2. DESCRIPTION OF FREE SPACE WITH POTENTIAL FIELD

A natural way to represent obstacles for the purpose of collision avoidance is by specifying their surfaces. For a piecewise smooth object consisting of smooth faces separated by ridge edges, the surface can be described by the collection of faces since each face i can be specified by an equation, say, $g_i(x)=0$, where x denotes the position vector of a point in space. Thus an obstacle whose faces are analytic hypersurfaces, $g_i(x)=0$, may be represented as the intersection of half spaces defined by the hypersurfaces, namely, by

$$\cap(x \mid g_i(x)\leq 0).$$

For example, the shaded two-dimensional object in Figure 2.1 is represented as the intersection of the half-planes lying on the interior sides of the boundary edge segments, i.e., by the constraints

$$-x\leq 0 \quad \text{and} \quad -(x+y)\leq 0 \quad \text{and} \quad x^2+y^2-1\leq 0.$$

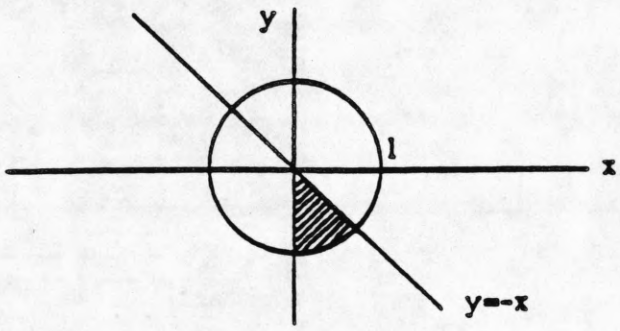


Figure 2.1 An example object.

From such surface representations, and analogous representations of obstacle volume, it is rather inconvenient to extract the information about the topology of the free space between the obstacles which is necessary for finding distinct paths. Transformation of this representation into a form so as to bring out the topological structure constitutes an integral and crucial part of the solution to the findpath problem[3]. This has been the motivation behind the use of direct representation of free space instead of obstacles. However, a geometric representation of free space by primitives leads to combinatorial complexity in intersection detection. Intersection of MO with a large number of primitives becomes necessary to derive collision-free paths.

In the approach presented in this paper, we propose to describe the topological structure of the free space by a scalar potential function. The negative gradient of the potential field yields a repulsion force between MO and the obstacles. The strength of the force is small far away from the obstacles, and grows monotonically towards infinity as MO approaches an obstacle. In order to generate the force with such magnitude profile, the potential function must monotonically decrease as the distance from the obstacles increases. The second derivative of the potential function must also be negative to make the magnitude of the gradient smaller as the distance from the obstacles increases.

2.1. Selection of a Potential Function

The potential field serves two purposes in the development of findpath algorithms. First, it is used to compute repulsive force which is used to avoid collisions. Second, and more important, the potential field brings out the topological structure of the free space between obstacles in the form of minimum valleys of the potential field. It serves to indicate the presence of obstacles, and as an analog representation of object shapes. The potential at any location is influenced by only the surrounding obstacles. This is precisely what is necessary since collision

[3]As described in the previous section this has been done in different ways in the existing findpath algorithms: Chatila [CHAT82] divides the free space between polygonal obstacles into convex polygons which are then stored in a connectivity graph, Kuan et al. [KZBR85] use convex polygons and generalized cones to describe the free space; Herman [HERM86] decomposes the problem space into an octree and marks each cell as free or occupied by the obstacles.

avoidance and local optimality characteristics of the path are determined from the configuration of nearby obstacles. Thus, such an analog representation circumvents the combinatorial complexity characteristic of representations of objects or the free space in terms of shape primitives, wherein the occurrence of collisions or their absence must be established by performing intersection tests against individual primitives separately. In this section we first describe how we choose a potential function. Then we describe how the potential function representation is used to obtain minimum potential valleys. For brevity, the potential field functions will be called potential functions in the rest of the paper.

There are many choices of the potential functions that meet the requirements mentioned above. The most common potential in physics is the Newtonian potential function. It is proportional to the inverse of the distance from the charge source. When the source is not a point, the potential $p(x)$ at location x due to a charged region R can be computed as a cumulative effect of charges dq present in small subregions around points x':

$$p(x) = \int_{x' \in R} dq / (x - x').$$

The Newtonian potential has many nice properties such as harmonicity[4]. Unfortunately, however, an analytic expression of the Newtonian potential is not available even for an arbitrary polygon, and the use of the numerical integration is inevitable. Since the computation of potential function will be performed extensively in the findpath algorithms, a potential function that has an analytic expression and is thus more efficient to compute is developed below.

2.2. A Simple Potential Function

The following derivation of a simple potential function is valid in both two and three dimensions. Consider a function which is zero inside a region and increases monotonically outside the region. Such a function is described in a paper by Comba [COMB68]. Let

$$g(x) \leq 0, \quad g \in C^m, \quad x \in R^n$$

be the set of inequalities describing a region, whether bounded or unbounded. Then the scalar function

$$f(x) = \sum g_i(x) + |g_i(x)|$$

where $\sum$ ranges over all boundary segments of a region, is zero inside the region and grows linearly as the distance from the region increases. Let us define a potential function p as

$$p(x) = [\delta + f(x)]^{-1} = \frac{1}{\delta + \sum_{i=1}^{\text{no. of bound. seg.}} (g_i(x) + |g_i(x)|)}.$$

The function p meets the requirements of the potential function needed for the findpath problem; $p(x)$ has its maximum value of δ^{-1} inside the region and decreases as the inverse of the distance outside the region[5]. The graph of the potential function of a triangular object is shown in Figure 2.2, and the level curves of the potential function for a rectangle are shown in Figure 2.3. 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.

---

[4]Harmonicity guarantees that local maxima of the total potential function occur only at the locations of the charge sources. This is a desirable property for the potential function since the local maxima would then always correspond to obstacles.

[5]The parameter δ has a small but nonzero value; the value $\delta = 0$ is avoided to keep p bounded for the purpose of computer representation.

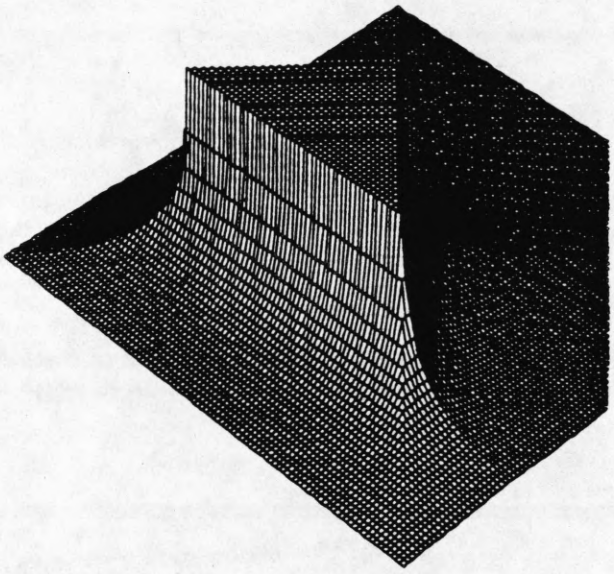


Figure 2.2 The potential function of a triangular object.

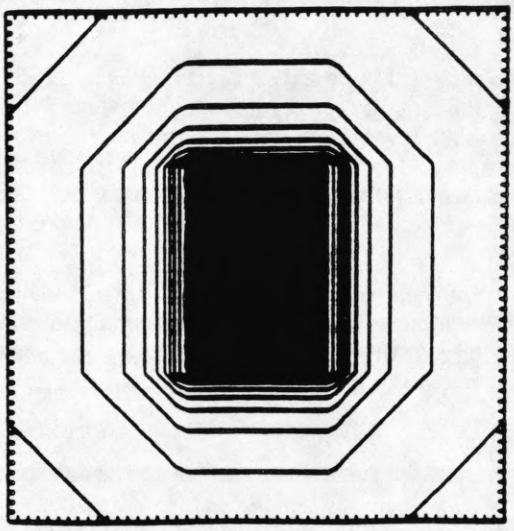


Figure 2.3 Level curves of the potential function due to a rectangular object.

Proposition 1. Local maxima of the potential function

$$p(x) = \underset{j=1}{\overset{\text{no. of obstacles}}{Maximum}} \left[\frac{1}{\delta + \sum_{i=1}^{\text{no. of bound. seg. of } j} (g_i(x) + |g_i(x)|)} \right]$$

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) contained in 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 perpendicularly intersecting MPV occurs on MPV.

The above two propositions guarantee that all curves (surfaces) in MPV are topologically distinct from one another. We now describe the use of this potential function to obtain the minimum potential valleys along with a summary of the algorithms we have developed for this purpose.

2.3. Obtaining Topological Structure of Free Space

There are infinitely many paths connecting the start and the goal positions in a given problem. It is an integral part of our algorithm to classify all possible paths into a finite number of representative, or topologically distinct paths[6], and examine only these for deriving all distinct efficient collision-free paths. Valleys of the potential function defined by the potential minima can be used as the topological paths. Minimum potential valleys (MPV) lie as far away from the obstacles as possible, and thus correspond closely to the medial axes[7] of the free space. The algorithms generating MPV differ significantly in two and three dimensions, and are discussed separately. In the following subsections, the points representing the MPV are called the nodes.

---

[6]In 2D two paths are said to be topologically equivalent if one can be deformed to another without crossing over the obstacles. Otherwise, they are said to be topologically distinct. In 3D a path can be deformed to a qualitatively different path without crossing over the obstacles. Thus in 3D, we will loosely define the term "topological distinct" as qualitatively different or simply very different.

[7]Paths similar to the medial axes of the free space may serve as topological paths. There are many methods for finding the the axes of the freeways between the obstacles. Voronoi diagram [DRYS79] can be used to generate such axes, or the spines of the generalized cones may be used. The availability of the potential function, however, offers an attractive alternative to the above methods, which unifies the steps of detecting topological and optimal paths through the use of the same representation.

2.3.1. Minimum potential valleys in two-dimensional space

MPV in 2D comprise an undirected graph consisting of junctions and branches. Junctions are the places where two or more branches meet. To obtain paths from the start to the goal nodes, we are of course interested in only that part of MPV which is connected or close to the start/goal nodes. Thus, it is not necessary to find all local minima of the potential function; rather the trajectories of the potential minima can be traversed starting at the start and the goal nodes to derive the MPV of interest. To extend the MPV beyond a node, all local minima of potential along a circle centered at the node are found, and they serve as successors of the original node along different branches of MPV emanating from the original node. Initially, a circle of maximum distance (MD) contained in the free space is drawn with its center at the start node (the algorithm computing MD is presented in Appendix). 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. These points are local maxima of distance from obstacles. The neighbors are again treated as the start node to generate more neighbors. Successive nodes are along medial axes of free space originating at the start node. The same procedure is carried out for the goal node. This process continues until all parts of the free space are filled with circles. The centers of the circles define MPV. The process of finding MPV is illustrated in Figure 2.4. Figures 2.5 a and b show MPV found by this algorithm for two examples.

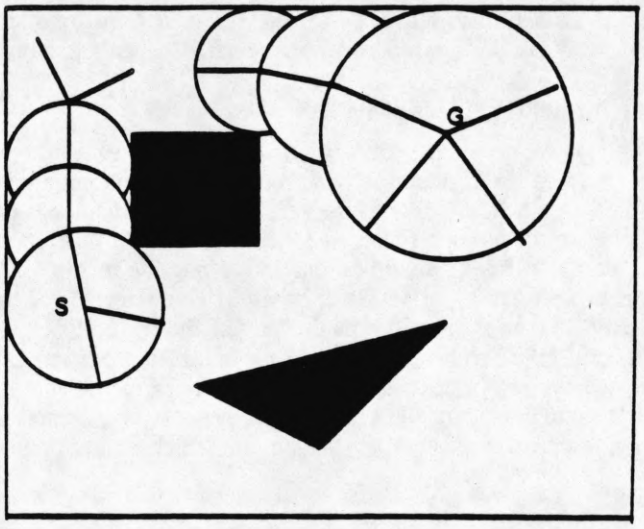


Figure 2.4 The process of generating the minimum potential valleys.

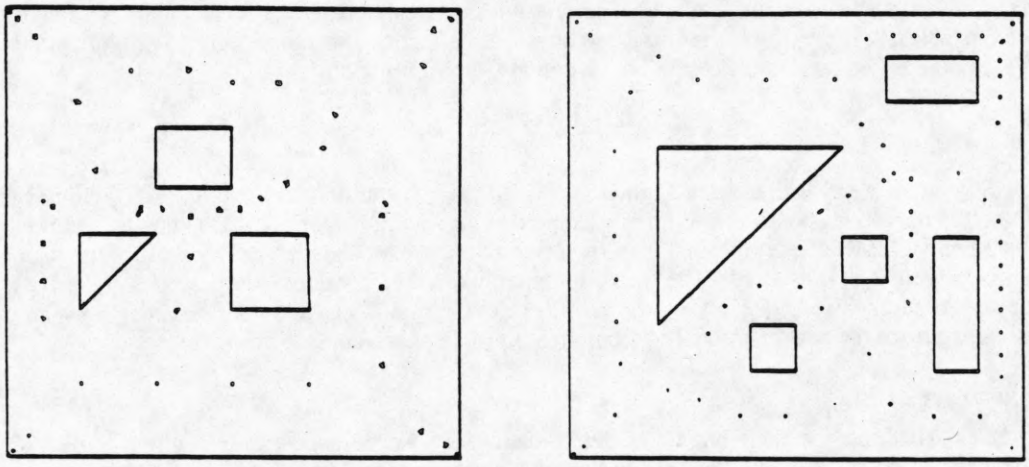


Figure 2.5 Minimum potential valleys of three findpath problems.

2.3.2. Minimum potential valleys in three-dimensional space

While MPV in 2D are curves, MPV in 3D are isopotential surfaces which can be thought of as many soap bubbles stuck together. For computational reasons, the MPV algorithm should generate a minimum number of points (nodes) on the minimum potential surfaces, without missing parts of MPV. In 2D, the sons of the current node are the nodes of locally minimum potential on the circle centered at the current node. In 3D, however, the search for the sons has to be done on a sphere rather than a circle, and this causes problems. Since a sphere is a two-dimensional manifold, points of locally minimum potential are harder to find. In principle, points of locally minimum potential must have the directional second derivative positive in all directions. It is computationally expensive to check this in all directions on the sphere. Furthermore, saddle points of the potential must be included among nodes to cover all minimum potential surfaces. These two problems prevent the generalization of a 2D MPV algorithm to 3D. A new algorithm, which is called "cover algorithm," is developed to solve these problems. This algorithm basically fills or covers the free space with overlapping spheres, and the centers of the spheres are marked as the points on MPV. Beginning from the start and goal positions, 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. The son nodes whose distance to the obstacles (MD) is greater than the width of MO are considered as possible nodes representing MPV. These son nodes are ordered by their 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. When this procedure terminates, MPV span the entire free space.

2.4. Search for the Best Path in Minimum Potential Valleys

The minimum potential valleys represent the topology of the free space in a condensed form. An infinite number of paths connecting the start and goal node have been reduced to a finite number of possibilities. The next task is to select an optimal candidate path from MPV in the sense that the path should be of minimum length and least likely to cause collisions between the moving object and the obstacles. At this juncture, we fall into a bit of a dilemma. To find out whether a collision-free path and orientations along the path exist in the vicinity of a candidate path, the findpath algorithms in Sections 3 and 4 have to be used. But before using the findpath algorithms, we have to determine an optimal candidate path in the sense of the least chance of collision. This problem forces us to estimate the chance of collision for a given path without using the complete collision-detection algorithms of following sections. In this section, we propose a cost function that estimates the chance of collision only from the width and length of MO and the potential along the path. Once the cost is assigned to all the points in MPV, dynamic programming is used to compute the shortest path with minimum chance of collisions.

2.4.1. Cost function

A cost function is used for each branch in MPV to denote the length and the difficulty of each branch to be traversed by MO. The cost function is made to be proportional to the length of the branch. The values of potential along the branch are used as a measure of the chance of collision. Lower potential means that the branch is farther away from the obstacles, and therefore, there is less chance of collisions and more room to minimize the length of the branch. Taking these two factors into account, we define the cost function

$$C = \int w(x)\, |dx|$$

where the weighing factor $w(x)$ is defined in Figure 2.6. 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 to indicate this fact. If MD is greater than one-half the longest dimension of MO, MO can go through x in any orientation. In such places, only length of the path is important, 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 $w(x)$ in Figure 2.6 gave a good balance between chance of collision and the lengths of the paths for most of the findpath problems we considered.

2.4.2. Dynamic programming

Once the cost function for each branch is determined, MPV represents a graph with positively weighted branches. Many algorithms are available for finding the shortest path in such a graph. Dijkstra's algorithm [MINI78] and dynamic programming [DYMC70] are some of the most efficient algorithms. Dynamic programming seems to work faster in the network of MPV since the shortest path between two nodes is generally given by the path using the smallest number of branches. A dynamic programming technique assigns to each junction the minimum cost to go to the goal node. For each branch of a junction, the cost to go to the goal node via that branch

is computed. Once this is done, the shortest path can be found by following the branch with the smallest cost to go from the start node until the goal node is reached. The resulting path will be the path of minimum length and minimum chance of collisions. This path is smoothed and then used by the findpath algorithms developed in the following sections to determine the final collision-free path and orientation for MO. It may well be the case that there is no solution along this path. Then the branch that causes the unavoidable collision is identified and assigned a large cost. The MPV graph is updated and the shortest path in the updated graph is found and supplied to the findpath algorithms. At present, dynamic programming is used again over the whole MPV. More efficient updating algorithms may be used. Figure 2.7 shows the shortest path found for a 3D findpath problem.

The MPV and the path selected by dynamic programming thus serve as coarse estimates of likely feasible and optimal paths. The true paths and MO orientations along the paths must still be computed. This is accomplished by one of three algorithms according to the difficulty of the problem. These algorithms are described in Sections 3-5.

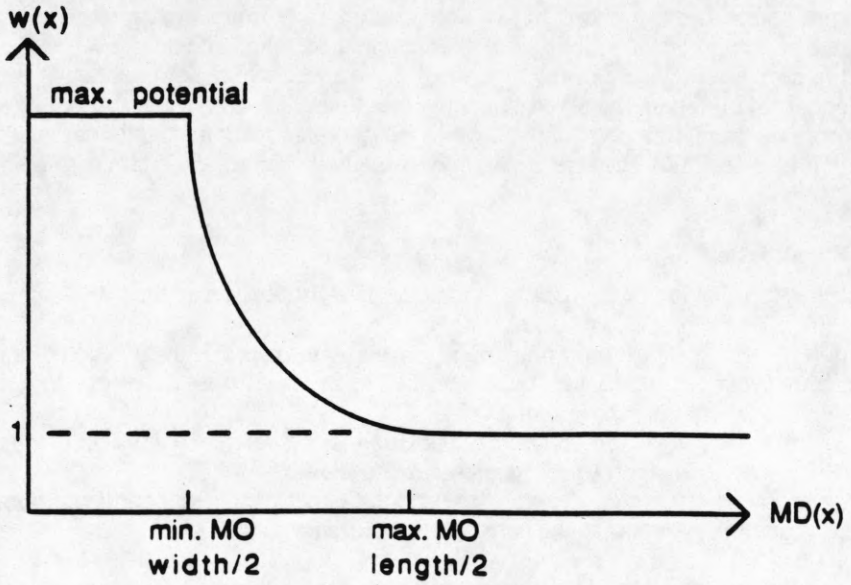


Figure 2.6 A weight function.

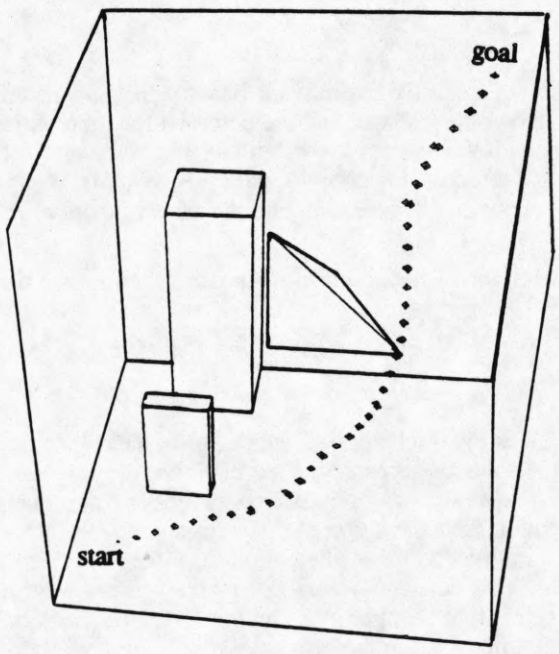


Figure 2.7 The minimum cost path chosen from the minimum potential valleys.

3. PARALLEL OPTIMIZATION ALGORITHM

The goal of the parallel optimization algorithm (POA) is to find 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

$$J=\int_{x_0}^{x_f} |dx| + P(x)|dx|, \quad x \in R^n \tag{3.1}$$

where $P(x)$ is the total potential experienced by MO at position x. A numerical method must be employed to minimize J in the space of continuous functions. A numerical method starts with an initial guess of the solution and iteratively changes it to decrease the value of the functional J. In most nonlinear optimization problems, the initial guess determines which local minimum the algorithm converges to and the convergence time. We use the best candidate path selected from the minimum potential valleys in the previous section as an initial guess to the numerical method. This algorithm is called POA because the numerical method changes the entire path a little per iteration in a parallel fashion.

3.1. Mathematical Formulation

This section gives a formulation of the parallel optimization algorithm, using the following notation:

reference point	: an arbitrary point on an object used to specify the position of the object
object coordinate frame	: a coordinate frame attached to an object with the origin located at the reference point
x	: a vector denoting the position of the reference point of an object with respect to the base coordinate frame
θ	: a vector denoting the orientation of the object coordinate frame with respect to the base coordinate frame

Thus, the configuration of an object is completely specified by the pair (x,θ).

The total potential on MO is then a function of (x,θ):

$$P(x,\theta) = \int_{x' \in R_{MO}(x,\theta)} p(x')dx'$$

where $R_{MO}(x,\theta)$ represents the region occupied by the moving object. In general, an analytic expression is not available for $P(x,\theta)$ since it depends upon object shape, and the potential function varies nonlinearly with position. In our algorithms $P(x,\theta)$ is approximated by a sum of the potentials computed at a set of points uniformly distributed over the entire moving object. It suffices, however, to add the potentials along the surface of MO since it makes no difference whether MO is hollow or filled. Estimating the object potential $P(x,\theta)$ in this way places no restrictions on the shape of MO.

Before we proceed with the formulation of the optimization process, we rewrite the objective functional J as

$$J = \int_{x_0,\theta_0}^{x_f,\theta_f} [(1+bP(x,\theta))(dx)^2 + a(d\theta)^2]. \tag{3.1.1}$$

The second term in the integrand penalizes the orientation change of MO, the constant a being the relative weighting factor. Although this term is not absolutely necessary, it ensures that the orientation of MO does not change wildly in the final solution found by the numerical optimization process. The constant b controls the relative weights given to the optimization of path length and the separation from obstacles. If b is large the resulting solution tends to stay away from obstacles in order to lower the potential. When b is small, MO travels very close to obstacles to minimize the path length. This constant yields a very convenient way to control the safety margin between MO and the obstacles. The differential changes in x and θ are replaced with their squares for two reasons. The first reason is the computational simplicity, since the distance formula in more than one dimension requires the calculation of square roots. The second reason is related to the way the findpath problem is formulated, and is

discussed later in this section.

There are many different but equivalent ways to mathematically formulate the path optimization problem. In this paper, we use the optimal control formulation [ATFA66, BRHO75]. Let v and ω be the time rates of change of the position x and the orientation vector θ, respectively. Thus v and ω are the control variables that determine the trajectory of MO. Then system equations are

$$\frac{dx}{dt} = v(t), \quad x(t_0)=x_0, x(t_f)=x_f, \quad \frac{d\theta}{dt} = \omega(t), \quad \theta(t_0)=\theta_0, \theta(t_f)=\theta_f. \tag{3.1.2}$$

The Hamiltonian of the system is defined as

$$H = [\,(1+bP(x,\theta))(v(t))^2 + a(\omega(t))^2\,] + y_1 v(t) + y_2 \omega(t)$$

where y_1 and y_2 are the adjoint variables. The necessary conditions for the optimality based on the first variations are

$$\nabla_y H = 0 : \quad \frac{dx}{dt} = v(t), \quad x(t_0) = x_0, \; x(t_f) = x_f \tag{3.1.3a}$$

$$\frac{d\theta}{dt} = \omega(t), \quad \theta(t_0) = \theta_0, \; \theta(t_f) = \theta_f$$

$$\frac{dy}{dt} = -\nabla_{x,\theta} H \; : \; \frac{dy_1}{dt} = -\frac{dP}{dx} v(t)^2 \tag{3.1.3b}$$

$$\frac{dy_2}{dt} = -\frac{dP}{d\theta} v(t)^2$$

$$\nabla_{v,\omega} H = 0 : \quad 2(1+bP(x,\theta))v(t) + y_1 = 0 \tag{3.1.3c}$$

$$2a\,\omega(t) + y_2 = 0$$

Equation (3.1.3a) is the system equation, which merely says that the rates of change of position are velocities. Equation (3.1.3b) is the adjoint equation. Introducing adjoint variables is a common technique in optimization algorithms. Equation (3.1.3c) is the stationary condition. In other words, it states that the objective functional must stay at the same value for arbitrary differential changes in the path and orientation if the path and orientation are optimal.

The reason that the differential changes in (3.1.1) are replaced with their squares is to make the above formulation nonsingular. Suppose a path minimizing (3.1) is found. Then another solution on the same path with a different speed profile also qualifies as a minimizing solution. This singular nature of the problem makes the implementation of second-order algorithms difficult, although only the first-order algorithm is used in this paper. It is also desirable to have a solution whose speed profile is as uniform as possible. When the squares of the differential changes are used as integrators, not only the problem formulation becomes nonsingular, but also the final solution of the numerical algorithm tends to have a uniform speed profile. The reason for this tendency is that although the total distance traveled is the same, the sum of the squares of the incremental distances is smaller for the case in which the increments are uniform.

Many numerical methods for the minimization of (3.1.1) subject to the differential constraints (3.1.3) exist [DYMC70]. The first-order gradient method requires less computation per iteration and converges rapidly initially, but shows slow convergence near the optimum. Second-order methods converge at faster rates once they get close to the optimum, but require much more computation per iteration, especially in high dimensional problems. The gradient method performs adequately for the findpath problem formulated above, and its steps are outlined in the appendix.

3.2. Parallel Optimization Algorithm in Two-Dimensional Space

3.2.1. Assignment of initial path and orientations

In the two-dimensional world, MO has two translational and one rotational degrees of freedom. The configuration can be represented by a three-dimensional vector. The first two components represent the position and the last component denotes the orientation of MO. Before the gradient method is used, the initial path of the reference point of MO and the orientation of MO along the path have to be determined. The initial path is selected as discussed in Section 2. The assignment of the initial orientation of MO along the initial path is the next important step in the algorithm. As the object moves along the path, a natural way to avoid collisions with the obstacles is to align the longest axis of the object shape with the direction of the path. Since the potential valleys correspond closely to the medial axes of the free space between the obstacles, setting the reference point of MO at the center of volume of the object will further minimize the chance of collisions. The two heuristics above are simple and effective in determining the initial orientation, and are used as the principle means of assigning the initial orientation. Once the initial path and orientations along the path are determined, the gradient method minimizes J starting from the initial path and orientations.

3.2.2. Examples

The results of running POA algorithm are presented below on several path planning problems, roughly in the order of increasing difficulty. We start out with the simplest MO, namely a point object, and then try a line segment and an L-shaped MO. The findpath problems are much harder when the shape of MO is concave, and the concavity has to be exploited to find a solution path and orientation. It turns out that a findpath problem in which an L-shaped MO is required to go through a bottleneck of the free space is complicated enough to make POA fail. The performance on a range of findpath problems is described to bring out the capabilities and the limitations of POA.

The easiest example considered is the problem of moving a point object through the space containing a single point obstacle, as depicted in Figure 3.1a. There are two topological paths possible in this case, and they are considered separately in Figures 3.1b and 3.1c. The dotted lines denote the initial guesses given to the optimization algorithm, and the solid lines are the results of the numerical optimization. The initial guesses actually used are the minimum potential valleys, but different initial paths are tried to test the convergence behavior of the optimization algorithm. The algorithm converges to paths that are close to the optimal paths for all initial guesses in Figures 3.1b and 3.1c in less than ten iterations. The starting value of ε in the gradient algorithm affects the rate of convergence of the algorithm. Small values of ε result in a slow convergence rate, whereas large values may make the algorithm jump over some of the obstacles. For example, if $\varepsilon = 5$ is used with the initial paths depicted in Figure 3.1b, the algorithm converges to the straight line path in Figure 3.1c. The path in Figure 3.1d is obtained as the optimal solution when the value of the parameter b in Equation (2.4.2) is increased from 1 to 5. The increased value of b makes the optimal paths stay farther away from the obstacle as the algorithm penalizes the potential to a greater degree.

A slightly harder problem than Example 3.1 is moving a bar-shaped object through polygonal obstacles. Figure 3.2a shows an example along with the minimum potential valleys and the initial object orientations. The results of optimizing each path are shown in Figure 3.2b, and these correspond closely to the ways people would actually move the object. The globally optimal path is path 1 in Figure 3.2b, which has the minimum path length among the locally optimal (topologically distinct) paths.

A more difficult problem occurs when the initial orientation of the moving object is such that it collides with some of the obstacles. Figure 3.3a shows one such situation. This physically impossible path is nonetheless used as the initial guess for the gradient algorithm. The resulting solution shown in Figure 3.3b avoids the obstacles by making a proper turn at the corner.

Figure 3.4 shows a situation similar to Figure 3.3, but it actually presents a much harder problem. Finding a path here requires intelligent maneuvering at the sharp corner, and the initial assignment of orientation according to the heuristics mentioned above results in a physically unrealizable path. Figure 3.5 also shows a case where intelligent maneuvering is necessary in order to move the L-shaped object through the narrow passage. When the path shown in Figure 3.5a, which has collisions, is given as an initial guess, the algorithm converges to the solution in Figure 3.5b. The resulting solution is indeed locally optimal in the sense of (2.4.2), but it amounts to saying that the algorithm can not find an acceptable solution to this findpath problem by modifying the given initial guess. The numerical method is guaranteed to yield an collision-free path of a minimum length once it is given a physically realizable path as the initial guess. This initial assignment of feasible orientations along the minimum potential valley plays a very important role in hard findpath problems, whereas simple heuristics suffice for relatively easy problems.

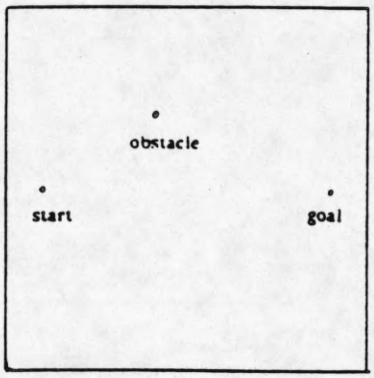


Figure 3.1a A simple point navigation problem.

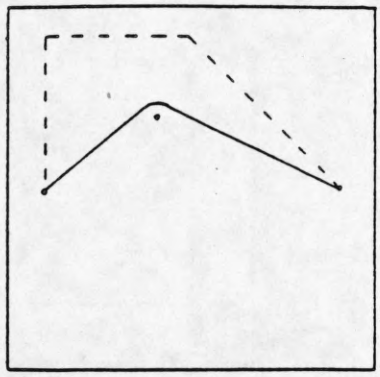


Figure 3.1b The initial guess (dotted line) and the optimized path (solid line) above the point obstacle.

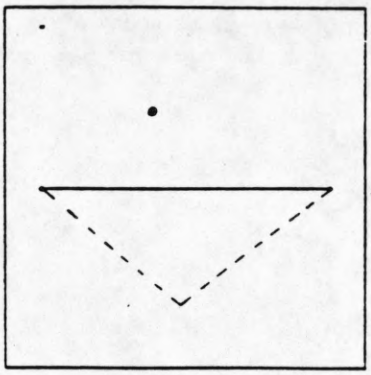


Figure 3.1c The initial guess (dotted line) and the optimized path (solid line) below the point obstacle.

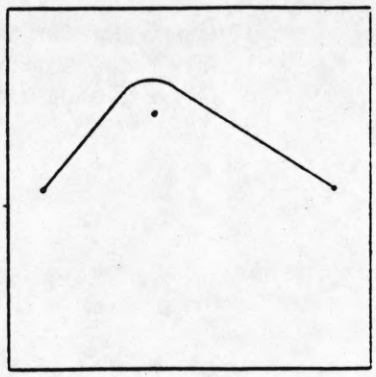


Figure 3.1d The effect of increased potential on the optimized path of Figure 3.1b.

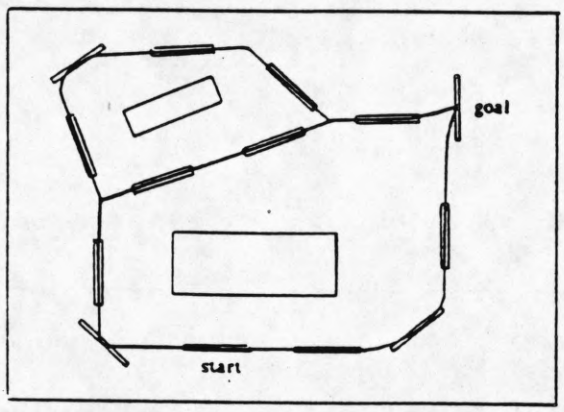


Figure 3.2a A bar moving through polygons. Initial paths are found from the minimum potential valleys.

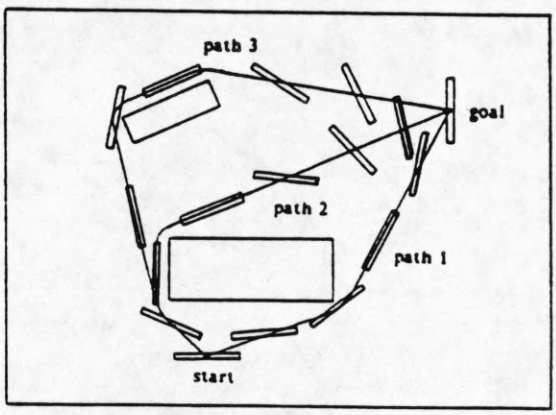


Figure 3.2b Optimized paths and orientations along the paths.

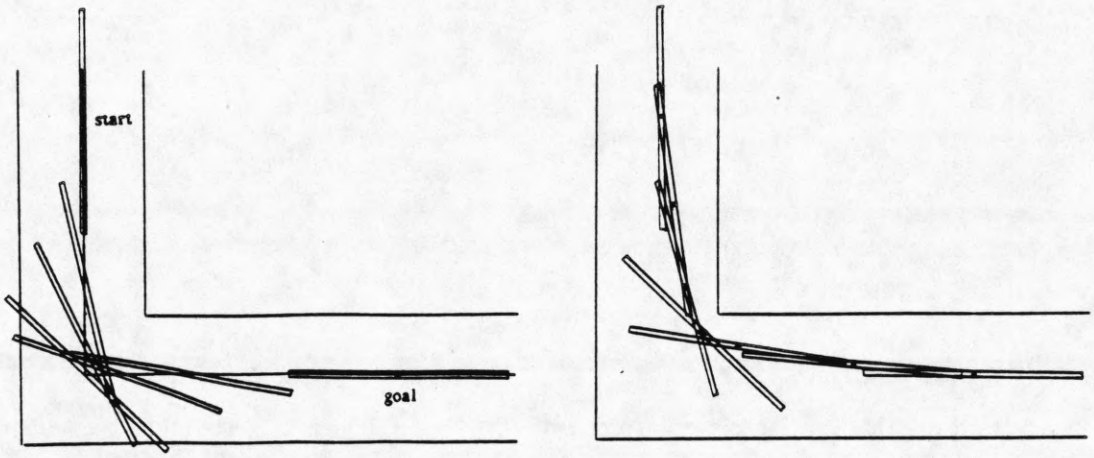


Figure 3.3 Problem of moving a bar around a corner of an L-shaped hallway.
(a) The initial path and orientaion.
(b) A collision free path found by POA.

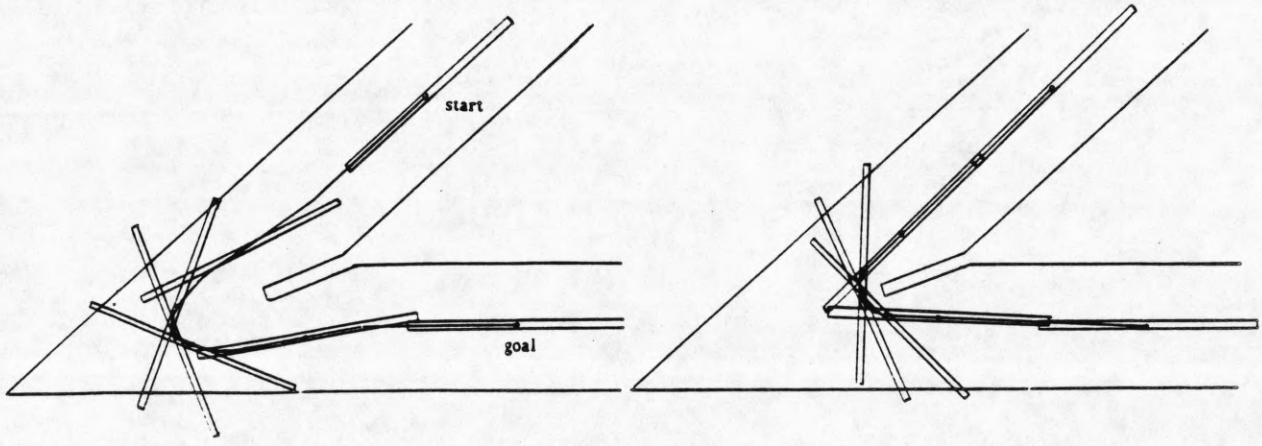


Figure 3.4 A findpath problem requiring intelligent maneuvering of the moving object at the corner.
(a) Initial guesses for the path and orientation are shown.
(b) The path found by POA. This physically impossible path and orientation mean that the algorithm cannot find a solution.

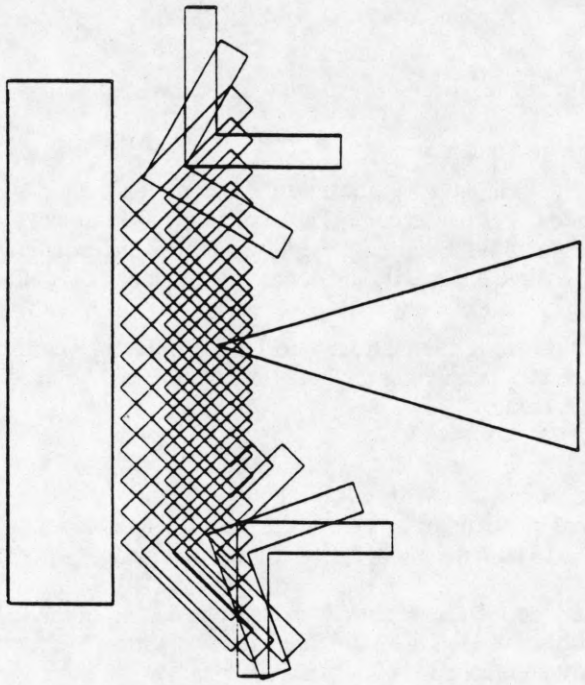


Figure 3.5a Initial guess of a solution to the problem of moving an L-shaped object through a narrow channel.

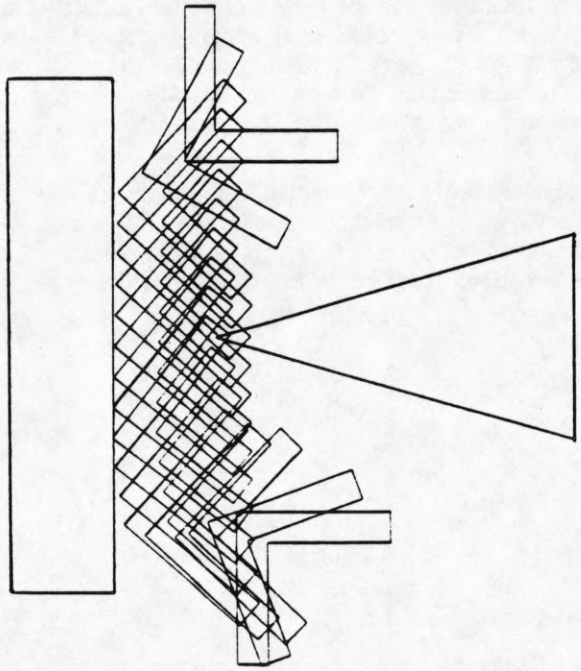


Figure 3.5b Result of optimization. POA fails on hard findpath problems.

3.3. Parallel Optimization Algorithm in the Three-dimensional Space

3.3.1. Assignments of the initial path and orientations

The findpath problem in three dimensions is significantly more complex than in two dimensions. There are three translational and three rotational degrees of freedom. The orientation of an object can be represented in Euler angles, quarternians or roll-pitch-yaw notation. Euler angle representation is more natural than others, and is used in this section. Three angles, θ, ϕ, ψ, completely specify the orientation of an object as illustrated in Figure 3.6. See [RIPA81] for more detailed discussion of the Euler angles.

An initial path and the orientation along the path have to be determined. The initial path comes from the MPV developed in Section 2. The heuristics assigning the initial orientation of MO need to be more sophisticated. Let us define the following terms for discussion.

Definition: Axes of a Moving Object
major axis: the axis along the longest extent of the object
second axis: the axis orthogonal to the major axis along the second longest extent of the object
minor axis: the axis orthogonal to both the major axis and the second axis

The choice of the axes may not be unique for an irregularly shaped object. As in the 2D case, it is reasonable to assign the major axis of the moving object to align with the direction of motion. The remaining degree of freedom in orientation can be selected such that the second axis lies in the plane tangent to the minimum potential surface at the given point. This alignment minimizes the total potential experienced by MO, and thus on the average, minimizes the chance of collisions. Once the initial path and orientations along the path are determined, the gradient method is used to minimize J.

3.3.2. Examples

The performance of POA in three dimensions is demonstrated through some examples illustrated in Figures 3.7 through 3.16, roughly in the order of increasing difficulty. In the first several examples, POA uses good initial guesses for the paths and orientations, which are derived by the procedure described above. In the next several examples, the initial paths and the orientations are intentionally assigned to cause collisions between MO and the obstacles. This is to test the robustness of POA to bad initial assignments of the paths and orientations. It may happen that the heuristics of choosing initial orientations may not lead to collision-free initial guesses. Through these examples, the ability of POA to recover from bad initial guesses are demonstrated. The last two examples show the cases where POA fails to find solutions.

Figure 3.7 shows a rectangular board turning a corner. The initial path and orientations are shown in Figures 3.7a and 3.7b. POA successfully finds the path of minimum length shown in Figures 3.7c and 3.7d. Notice the gradual change in the orientation of the board in Figure 3.6d. This occurs because we use squares of differential changes in position and orientation in Equation (3.3.1) rather than the magnitude of the differential changes. Sum of

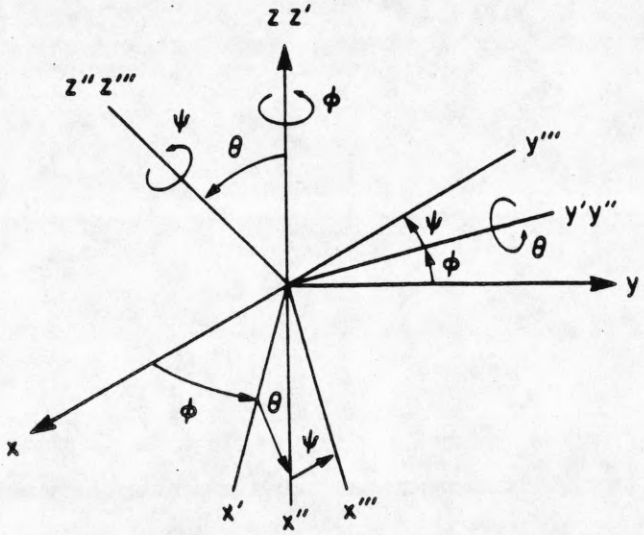


Figure 3.6 Euler angles, reprinted from [RIPA81].

the squares of the differential changes is smaller when the magnitude of the changes is uniform. A harder findpath problem is shown in Figure 3.8a with a guess of the solution drawn as a time lapse picture. Here the obstacles are a triangular block and a tall rectangular block behind it. A rectangular board is to move from behind the tall block to the left of the triangular block. The result of POA shown in Figures 3.8b and 3.8c is satisfactory. It took about 20 iterations, however, to converge to this solution since the initial path makes several sharp turns.

The next example in Figure 3.9 has three obstacles, two rectangular blocks of infinite heights and a rectangular block with the top surface higher than the initial and goal position of MO. A long rectangle avoids three blocks while minimizing the path length to reach the goal. Figure 3.10 shows an X-shaped object moving from the top of the smaller block to the side of the tall block. Figure 3.11 demonstrates the effect of the potential term in the performance index J. In Figure 3.11a, 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. A slightly harder 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 3.12a and b. POA successfully changes the position and the orientation of MO, resulting in a collision-free path of minimum length, as shown in Figures 3.12c and d. The next example is a big hoop trying to move around a corner of a cube, as shown in Figures 3.13a and b. The main thing to look for in this example is whether, to minimize the path length, the hoop will move such that the corner of the cube will poke through it. The result in Figures 3.13c and d shows that the hoop indeed hugs the corner while avoiding a collision.

In the following examples, bad initial guesses are used to test the robustness of POA. In the example in Figure 3.14, there is a narrow passage between the initial guess shown in Figures 3.14a and b and the final solution of the minimum path length. The obstacles are a rectangular block and a parabolic table with a small gap between them, and the moving object is a rectangular board. POA changes the initial guess iteratively through the narrow gap region, and converges to the path shown in Figures 3.14c and d. This shows that the narrow gap region, which is a local potential maximum in 3D space, is not a local maximum in the sense of J in (3.3.1) in the configuration space.

The next example is to move the same board through a rectangular hole. Notice in Figure 3.15a that the width of the hole is so narrow that the board hits the sides of the hole while moving according to the initial path and orientation. This initial orientation is not, of course, the result of the heuristics described earlier, but it is used as a bad initial guess to demonstrate the robustness of the algorithm. Figure 3.15b shows MO safely moving through the hole with minimum change of its orientation.

Figure 3.16 shows the same example as that in Figure 3.9, but initial guess used here is not the shortest path (the shortest path lies above the middle block. POA starts with the initial path shown in Figure 3.16a, and the gradient procedure converges to the nearest optimal path of Figure 3.16b. The globally optimal path is shown in Figure 3.9. This example shows that there is a local maximum of J in the configuration space between the path/orientations of Figure 3.16 and Figure 3.9. This local maximum prevents the gradient algorithm from converging from the initial guess in Figure 3.16a to the globally optimal solution. Figure 3.16c shows the same example, except the goal position is above the level of the top surface of the middle block. POA starts to raise the initial path to make MO rise at a uniform rate. Once the path reaches the top of the middle block, POA converges to the path shown in Figure 3.16d. The result is not, however, the way we would normally move the board from the starting to the goal position. These results show that POA always converges to a local minimum in the sense of J in (3.3.1) nearest to the initial guesses. Therefore, POA can tolerate bad initial guesses within the basin (or region) of attraction of the optimal solution.

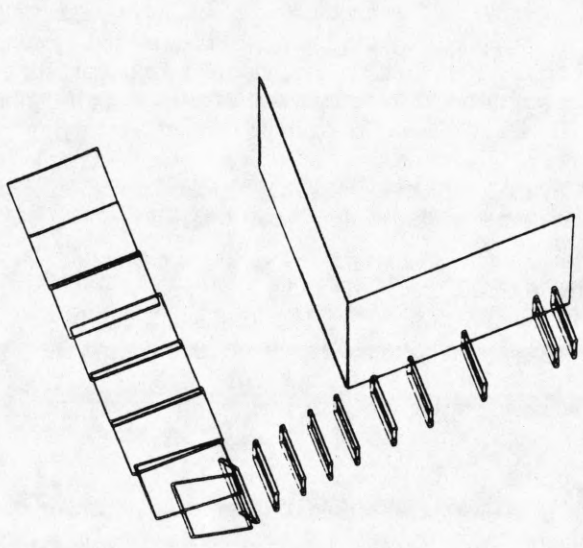
The last two examples are the cases where POA with a simple heuristic to assign the initial orientation fails. Consider the problem of moving an H-shaped block through an H-shaped hole, as shown in Figure 3.17. If the initial orientation is assigned so that the major axis of the H-shaped block lies in the direction of motion, a successful path will not be found. Another example is the task of hanging a hat on a pole, as shown in Figure 3.18. The hollow structure of the hat makes the assignment of the initial orientation a nontrivial task. These problems are still easier than the findpath problem in Figure 1.7c, but they require a more sophisticated method of assigning the initial orientation along the minimum potential valley.

3.4. Need for Spatial Reasoning

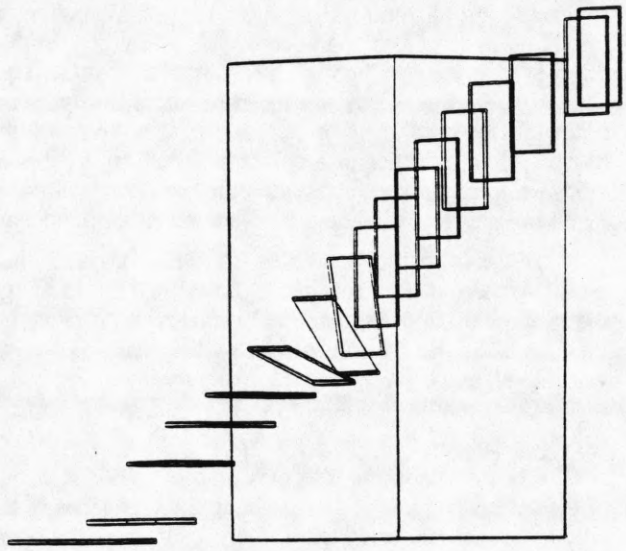
As seen from the examples above, a simple repulsive force between MO and the obstacles alone does not capture the complex shape interrelationships among objects that may have to be exploited to devise complex paths. The dividing line between *easy* and *hard* problems for POA could be described as follows. Suppose there is a stream of water flowing from the start to the goal location over the problem space. The obstacles are immovable barriers to the water flow, and MO is floating in the current. If MO is able to float from the start to the goal location

all by itself, then the particular problem is solvable by POA. There are many situations, however, in which MO is stuck between obstacles. A slight tipping or a change of orientation might get MO resume its journey to the goal position.

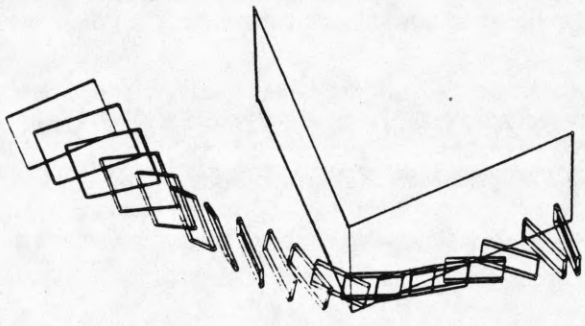
It is therefore necessary to complement POA with other steps so that appropriate perturbations of configurations can be made. This may be necessary only in the hard regions of the problem space requiring interlocking of shapes, while large parts of the problem space may be correctly handled by POA. Such spatial reasoning should examine a topological path and determine the necessary intermediate configurations along the path essential to avoid collisions. An algorithm that accomplishes this goal is described in Section 4. This algorithm can solve problems having the intermediate level of difficulty as illustrated in Figure 1.7b.



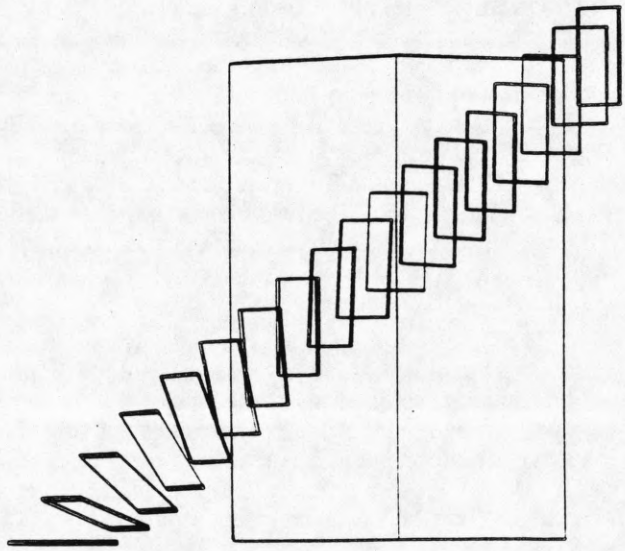
(a) The initial path and orientation.



(b) The side view of the path of (a).



(c) The optimal path found to move the board around the corner.



(d) The side view of the optimal path shown in (c).

Figure 3.7 A rectangular board turning around a corner.

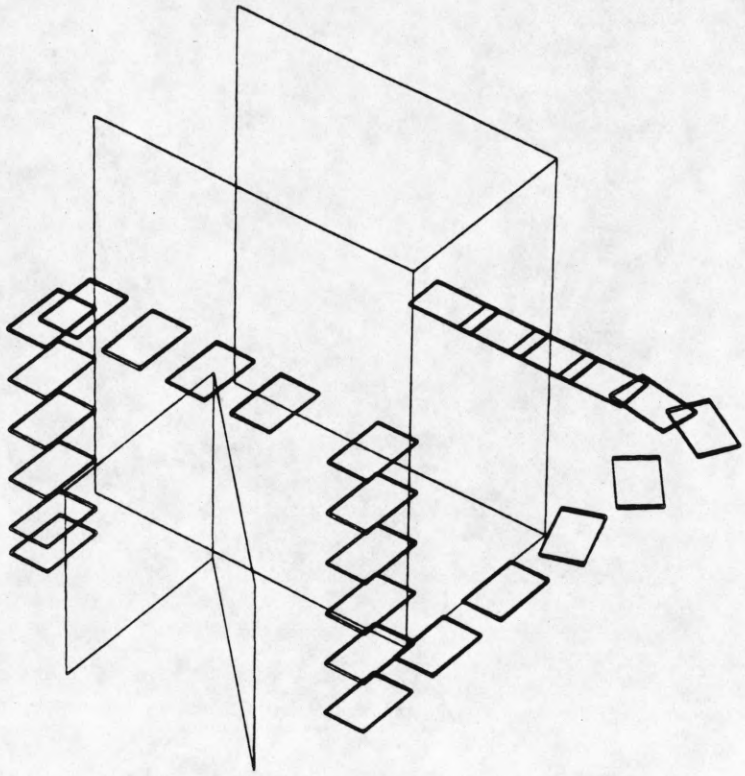


Figure 3.8a Initial path and orientation of the rectangular board.

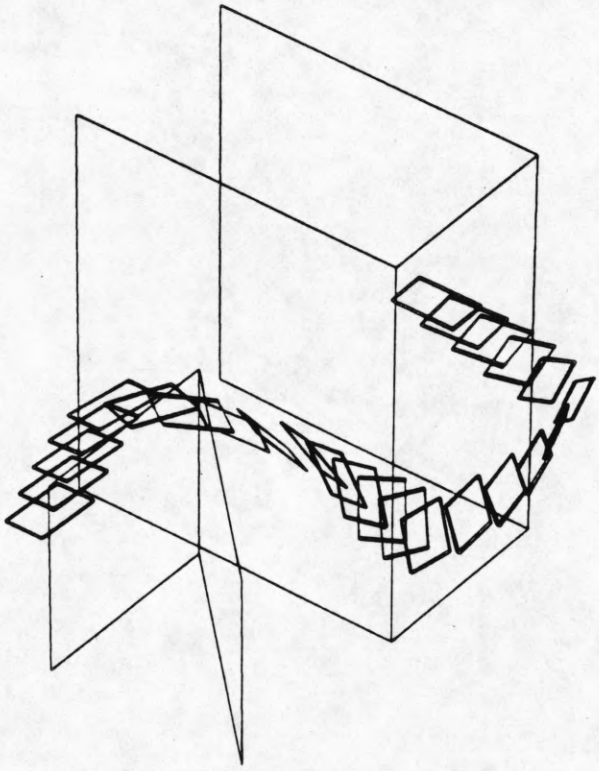


Figure 3.8b The result of optimization of the path in Figure 3.8a.

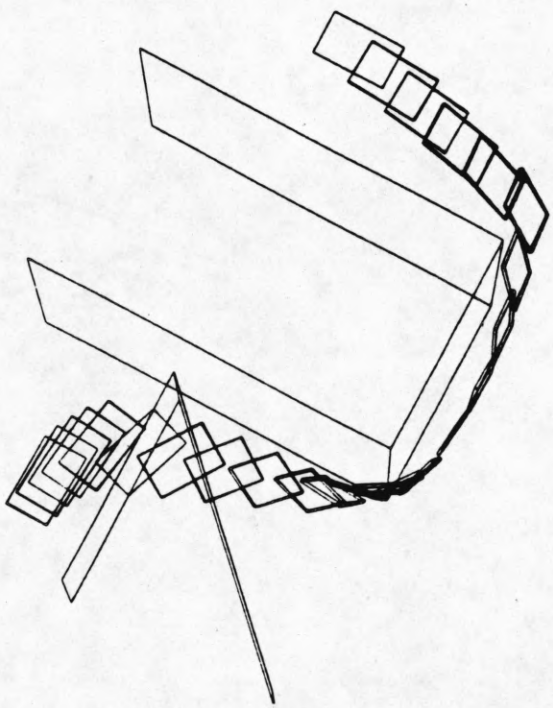


Figure 3.8c Top view of the path in Figure 3.8b.

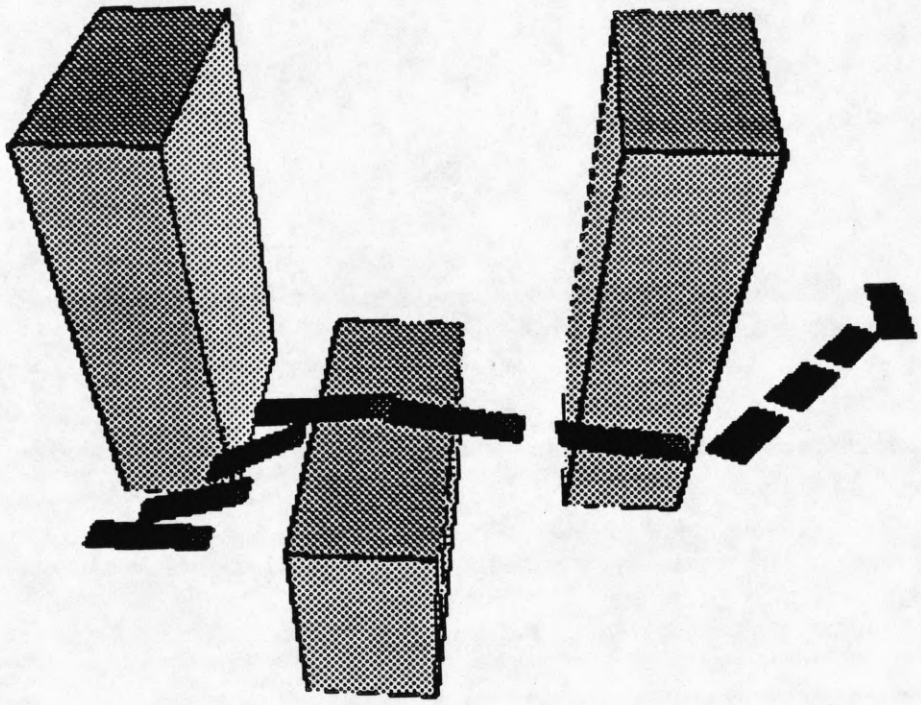


Figure 3.9 The optimal path avoiding collisions with three blocks.

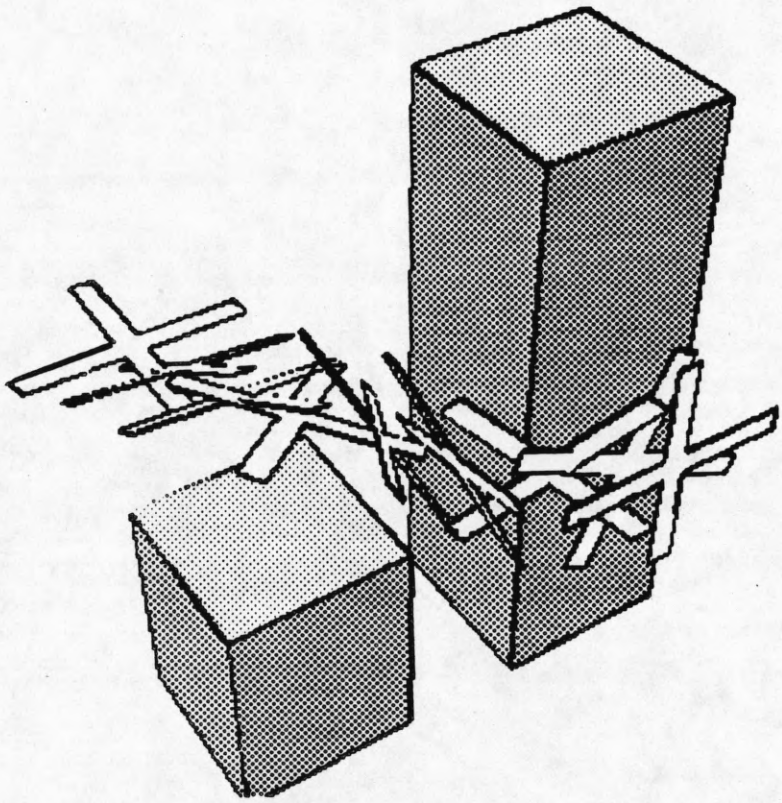


Figure 3.10 An X-shaped object moving from the top of the smaller block to the side of the tall block.

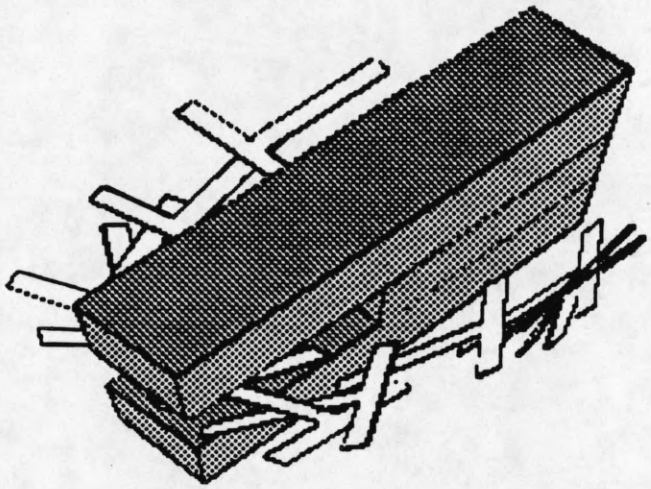


Figure 3.11a The gap between the blocks is wide enough for a T-shaped object to go through.

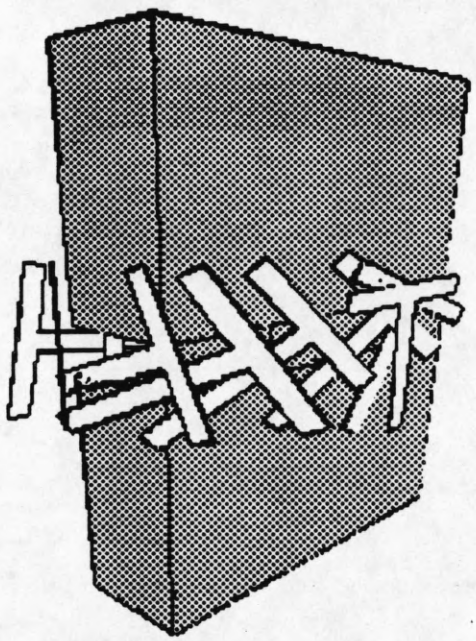
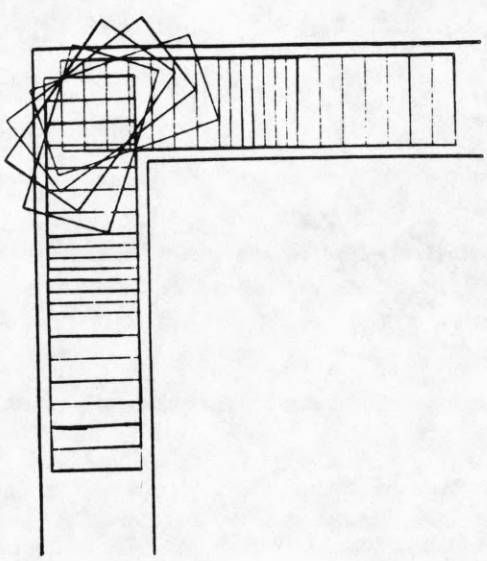
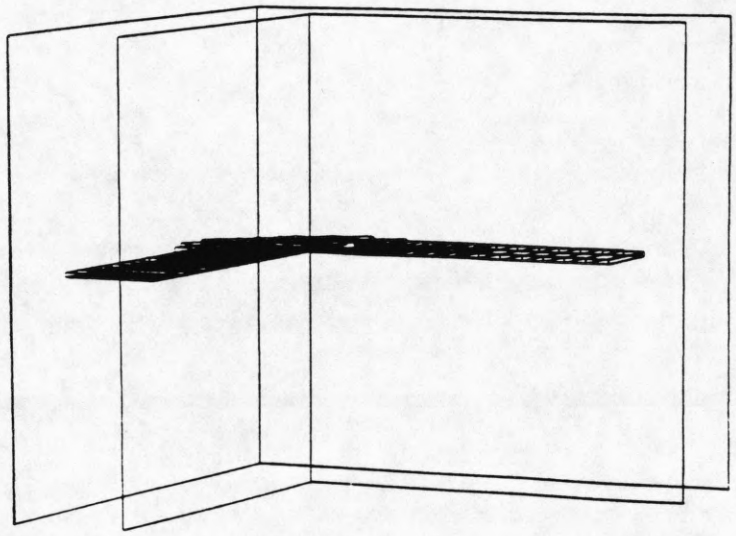


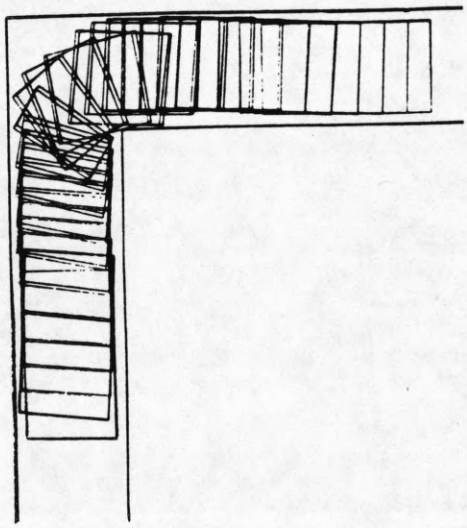
Figure 3.11b The gap between the blocks is so narrow that the T-shaped object goes around all three blocks to minimize the potential.



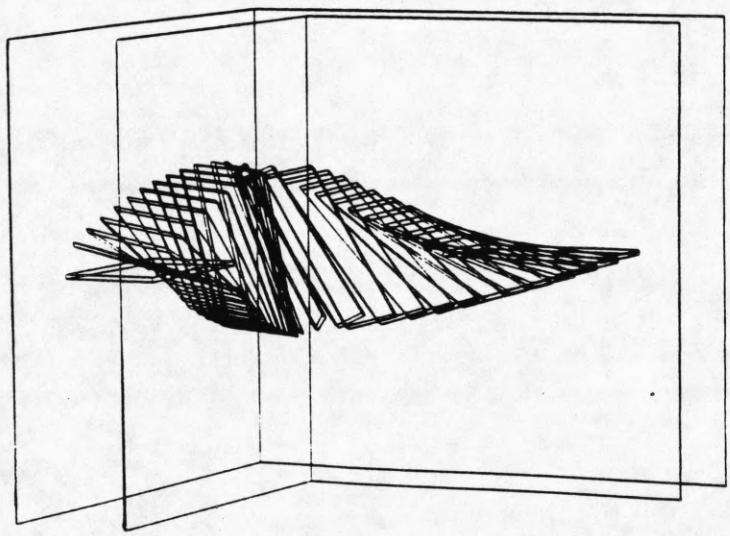
(a) The initial path and orientation.



(b) The side view of the initial path.

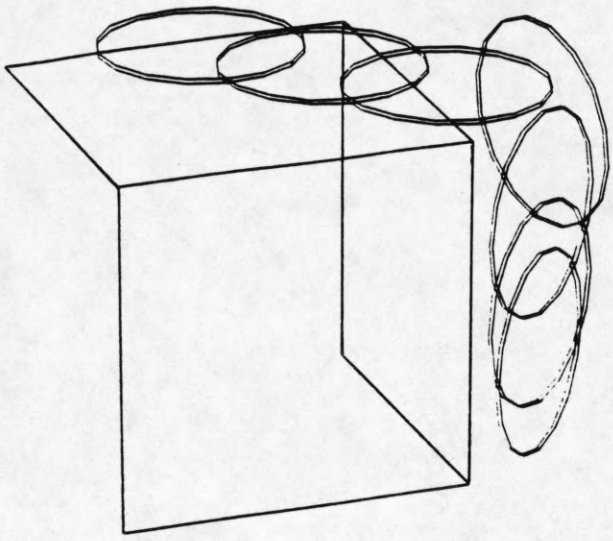


(c) The result of POA viewed from the top.

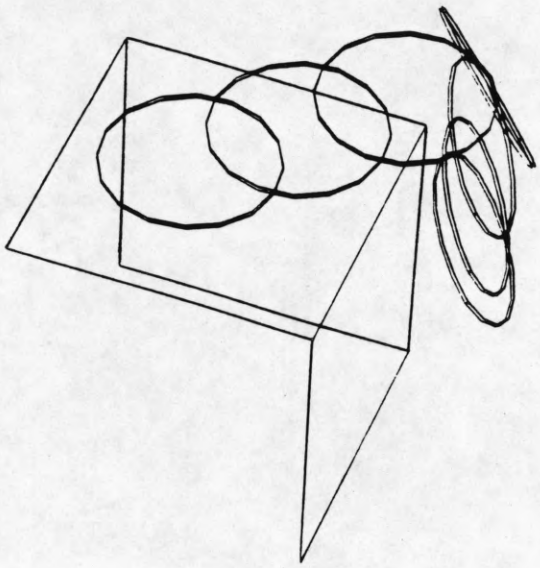


(d) The side view of the optimized path.

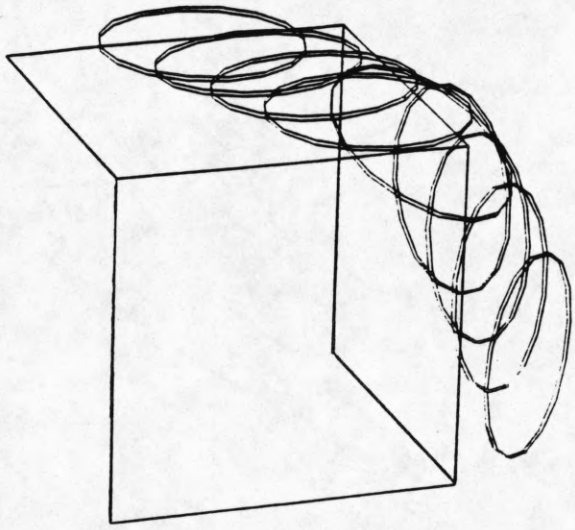
Figure 3.12 A rectangular board turning a corner of a hallway.



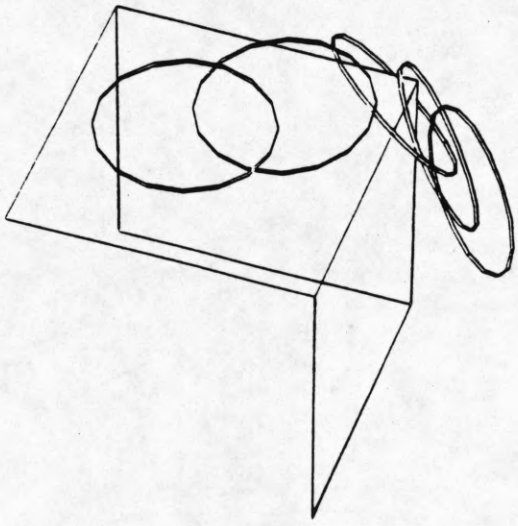
(a) The initial path and orientation.



(b) The initial path viewed from a different angle.

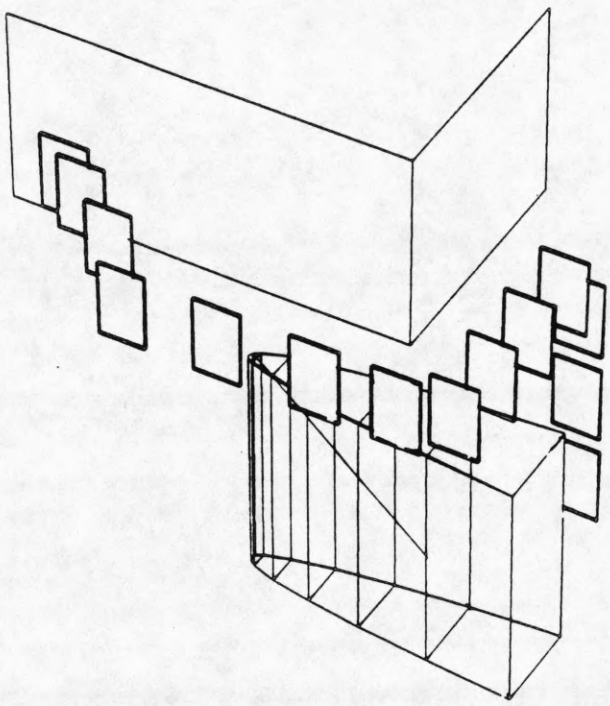


(c) The hoop hugs the corner of the cube to minimize the path length.

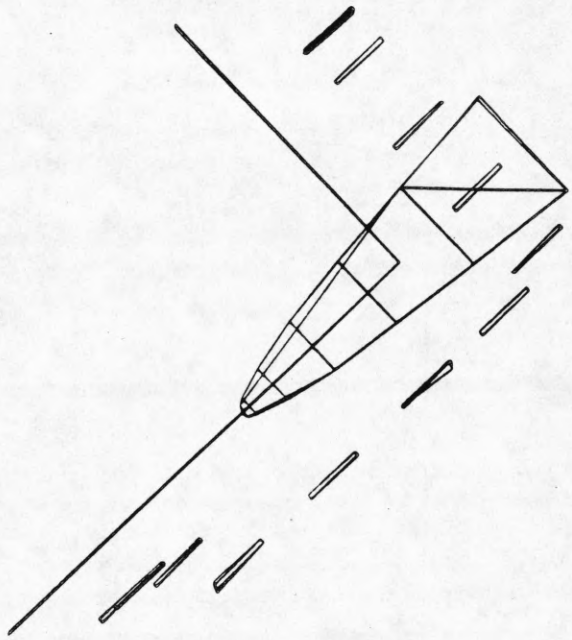


(d) The optimized path viewed from another angle.

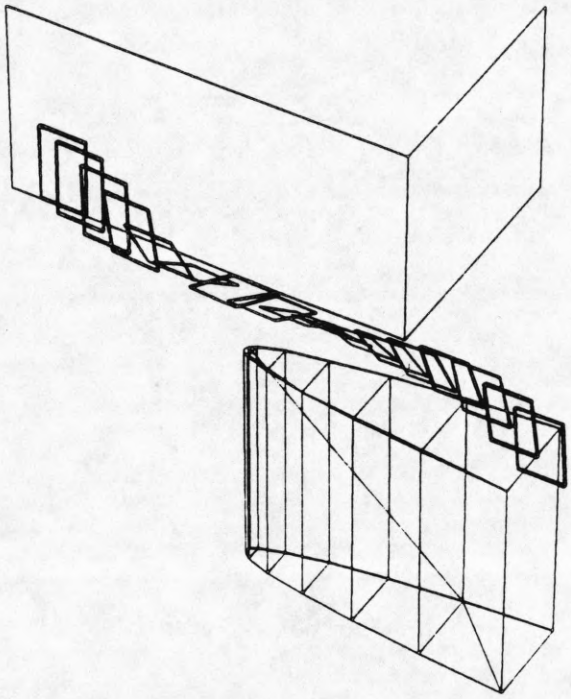
Figure 3.13 A large hoop turning around a corner of a cube.



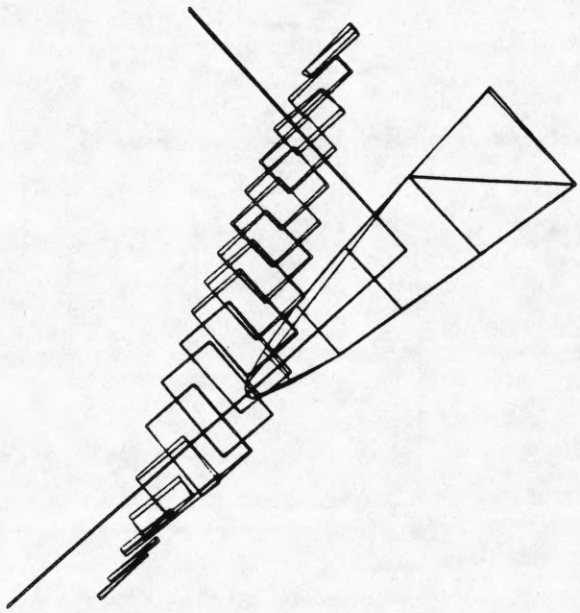
(a) The initial path and orientation.



(b) The initial path shown from the top.



(c) The optimized path and orientation of the board.



(d) The top view of the optimized path.

Figure 3.14 A board is to move from behind the parabolic table to the front rectangular block.

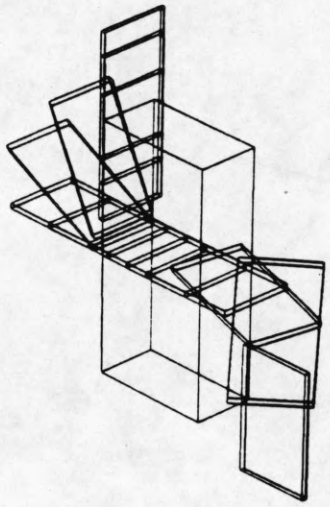


Figure 3.15a The initial path of a rectangular board moving through a rectangular opening.

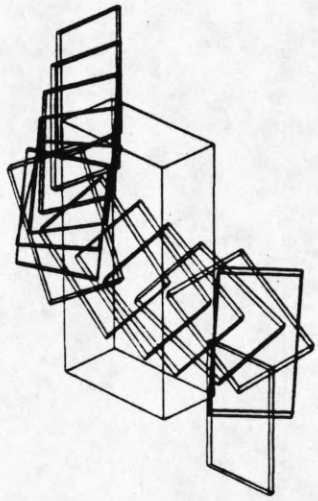


Figure 3.15b The collision free path and orientation of the board.

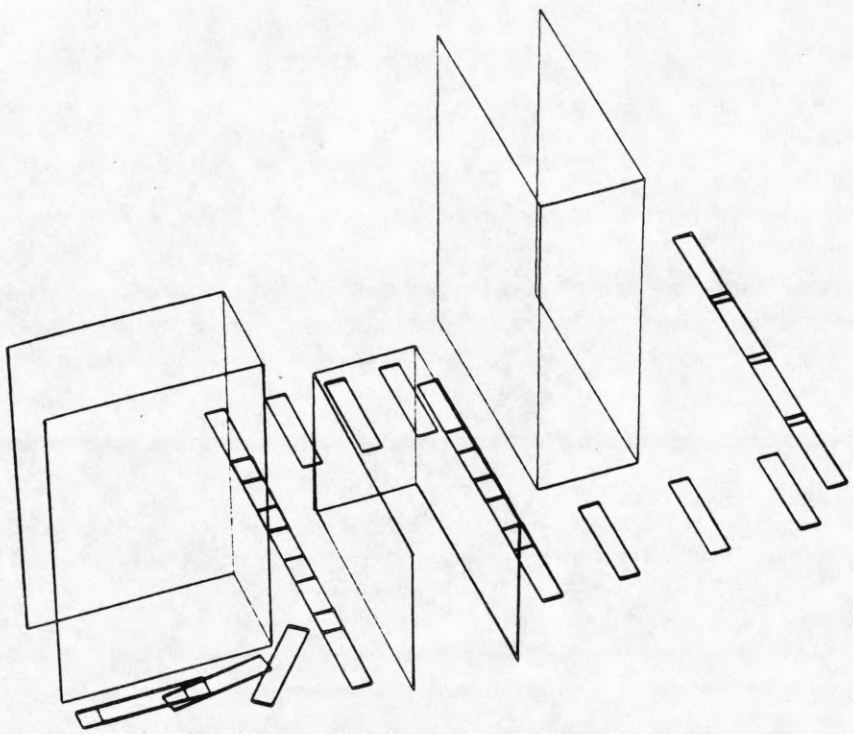


Figure 3.16a A suboptimal initial path for a long rectangular object. The optimal initial path lies above the short middle block. The suboptimal guess is used to study the performance of POA.

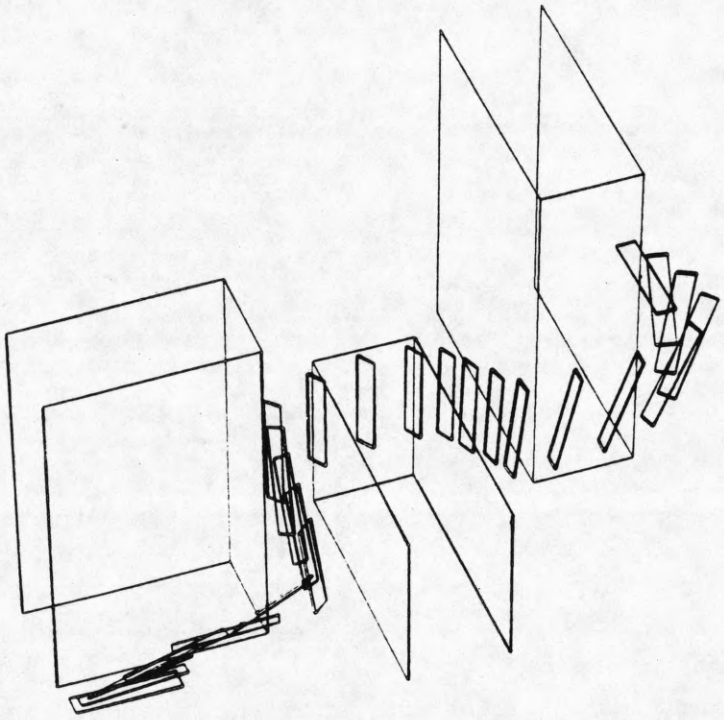


Figure 3.16b The moving object reorients vertically to move through the vertical gaps between the blocks. The final path is optimal in the vicinity of the initial guess, although the globally optimal path (Figure 3.9) is different. This example illustrates the importance of good initial guesses.

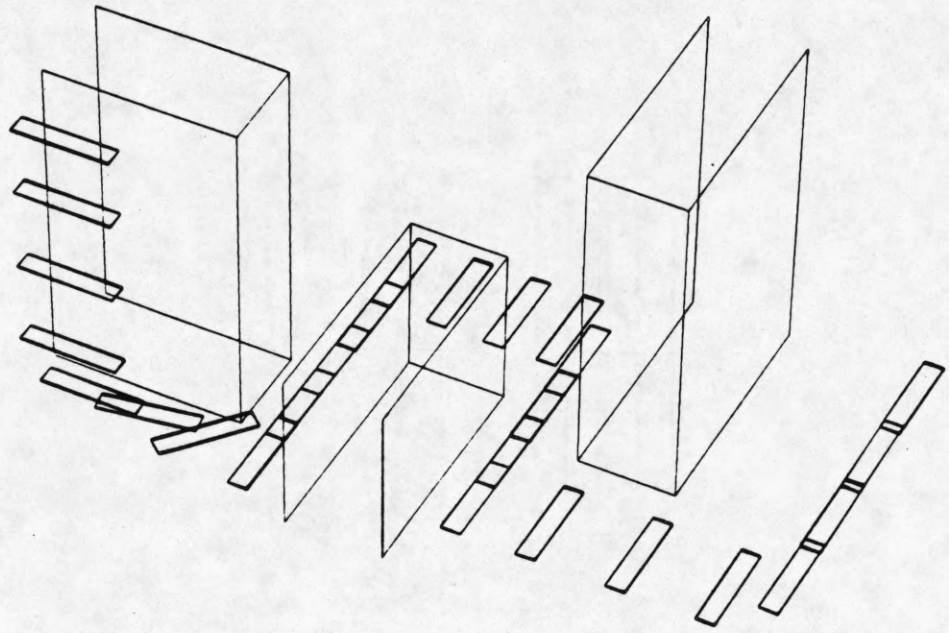


Figure 3.16c Same as the problem of Figure 3.12a except that the destination location is higher.

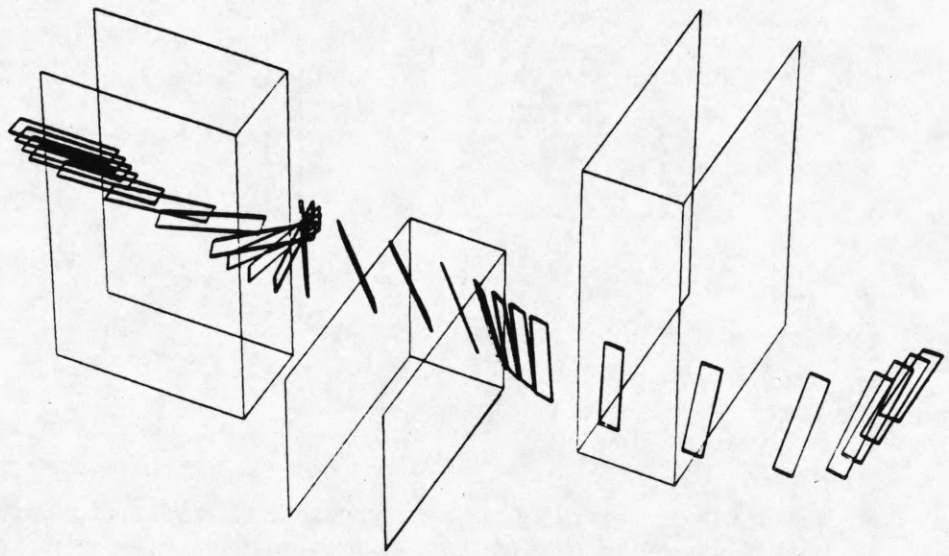


Figure 3.16d The result of optimization shows the effect of the inefficient initial guess.

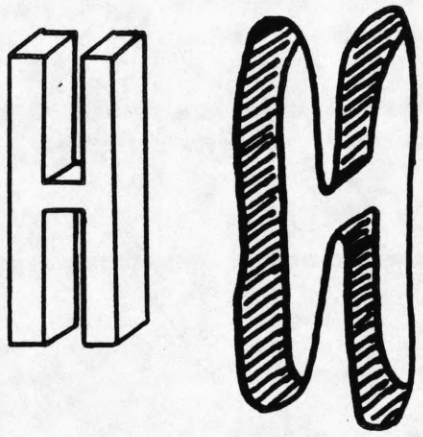


Figure 3.17 A findpath problem requiring spatial reasoning to determine the appropriate orientation of the moving object.

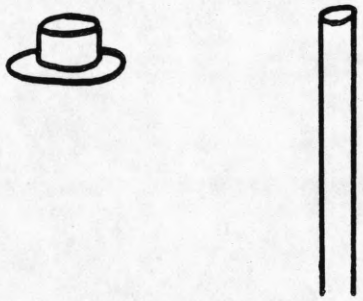


Figure 3.18 Problem of hanging a hat on the pole illustrates the need for a more sophisticated method of assigning the initial path for concave moving objects.

4. SERIAL OPTIMIZATION ALGORITHM

The parallel optimization algorithm can solve only "easy" findpath problems. When an intelligent maneuvering of MO is required to find a solution to a findpath problem, an algorithm which utilizes the shape of MO and the local geometry of the free space is needed. Such an algorithm, which we will call serial optimization algorithm (SOA), will be developed in this section. This algorithm differs from POA in the following. POA minimizes the weighted sum of the path length, and the potential experienced by MO over the entire path. POA may or may not converge to a collision-free path. In contrast, 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 likely 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 regions where collisions occur are identified. These regions will be called collision regions or "hard regions," since they occur in the narrow parts of the free space. Then collision-free configurations of MO in the hard regions are found. The next task is to connect a collision-free configuration of one of the hard regions with a collision-free configuration of an adjacent hard region. If the start configuration and the goal configuration can be connected through a sequence of collision-free configurations of the hard 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. Since SOA differs significantly between 2D and 3D cases, the two cases are described below separately.

The first two sections describe SOA for 2D and 3D. The next two sections show the results of SOA for 2D and 3D problems. As will be seen from the results, SOA can solve much harder problems than POA; specifically, it can solve problems of the intermediate level of difficulty (Figure 1.7).

4.1. Serial Optimization Algorithm in Two-Dimensional Space

4.1.1. Identification of collision regions

After the best candidate path has been selected, the next task is to identify narrow parts of the free space where MO needs to be in specific configurations, different from the initial guesses, to avoid collisions. The parts of the path where initial guesses about orientations of MO lead to collisions are called collision regions. The place in each collision region where MO overlaps with obstacles the most is called the collision center.

4.1.2. Feasible configurations in collision regions

The next task is to find collision-free configurations of MO at each collision center. There are usually an infinite number of such collision-free configurations, 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. MO is rotated with its center (reference point) fixed at each collision center, and the orientations of locally minimum potentials are stored for each collision center. Then for each stored orientation the center of the MO is moved around the collision center to further lower the total potential on MO. The results are topologically distinct configurations in which MO has locally minimum potential in the hard region. Some of these configurations may still make MO collide with the obstacles, and only those without collisions are considered as the feasible configurations. Figure 4.1 illustrates the initial path, the nodes with collisions and their collision centers, and the feasible configurations in the hard regions.

4.1.3. Connecting feasible configurations

We will say two configurations are connected if there is a continuous collision-free path of MO between them. 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. It is desirable to minimize changes in the orientation of MO between adjacent configurations, and thus adjacent feasible configurations with the closest orientations are selected first.

An algorithm then tries to connect the selected feasible configurations sequentially. This is the reason why this algorithm is called SOA. 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. The MOs follow the initial path while adjusting the position and the orientation to minimize the potential locally. After one of the MOs moves one step, the connecting

algorithm 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, then the two feasible configurations are connected before the algorithm steps through all the nodes between the two configurations. If this fails, one of the MOs takes another step closer to the other MO while minimizing the potential. This process of moving out of a high potential region into open space is easier than converging onto a configuration in a high potential region from a low potential region. This is because the MOs move out to wide parts of the free space, which allows two configurations of MO leaving adjacent collision regions to be connected via a straight line as early as possible. The connecting algorithm signals failure when 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.

The SOA is illustrated in Figure 4.2. Each level denotes one of the collision regions, and the nodes at each level represent topologically distinct, feasible configurations in the corresponding collision region which cannot be connected. The top node denotes the start configuration, and the bottom node denotes the goal configuration. Finding a collision-free path amounts to connecting the top and the bottom nodes using one node from each level. Figure 4.2b illustrates depth first and best first connecting algorithms. See Section 4.3.1 for implementation details.

4.1.4. 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 3.1.1. 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. Since the initial path given to POA is collision-free, the result of POA is guaranteed to be collision-free.

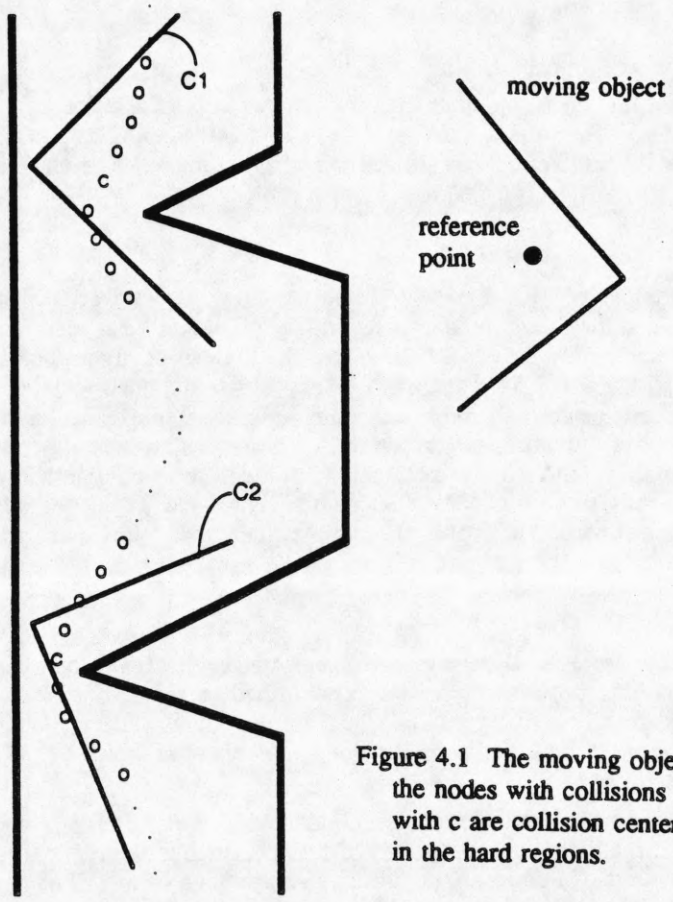


Figure 4.1 The moving object is moved along the initial candidate path, and the nodes with collisions are identified (denoted with o). Nodes denoted with c are collision centers. C1 and C2 are the feasible configurations in the hard regions.

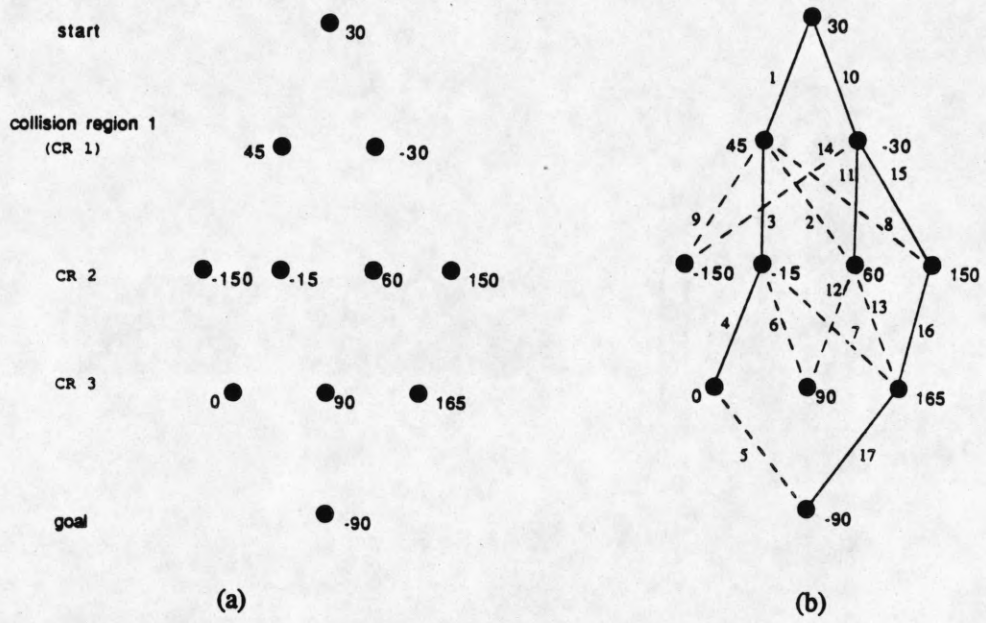


Figure 4.2 Graphical illustration of the serial optimization algorithm.
(a) Each level denotes a collision region (CR), and the nodes at a level represent the feasible configurations in the corresponding collision region.
(b) A solid line between two nodes denotes that the two nodes are connected by SOA, and a dotted line denotes that they cannot be connected by SOA. The connectivities are checked by SOA in the order of the numbers labeling the edges. The order shows the best first nature of SOA.

4.2. Serial Optimization Algorithm in the Three Dimensional Space

The 2D SOA can be extended to 3D in a straightforward manner. The 3D SOA consists of exactly the same four steps as 2D SOA, each step being slightly more involved due to the additional variables needed to specify the configuration of MO in 3D. Three variables are needed to specify the orientation of an object in 3D. Euler angles are a convenient way to specify the orientation, but the restrictions placed on the angles to make the representation unique cause problems, as will be discussed later. To avoid these problems, three orthonormal vectors specifying the major, second and minor axes of MO are used along with the Euler angles in the rest of this paper.

4.2.1. Identification of collision regions

After the best candidate path is selected, the orientations of MO along the path have to be assigned. MO is then moved along this path following the assigned orientations to determine narrow parts of the free space. As for POA, the longest or the major axis is aligned tangent to the path to minimize the chance of collisions in the initial movement. This heuristic assigns only two of the three Euler angles. As for POA, the third angle (ω in Figure 3.6) is determined by placing the second axis of MO in the direction of minimum potential.

4.2.2. Feasible configurations in collision regions

Finding feasible configurations in 3D is much more complicated than in 2D in the following sense. In 2D the orientation space is one-dimensional, and the orientations corresponding to local potential minima may represent topologically distinct orientations. For example, since the Y-shaped MO in Figure 4.3a cannot be rotated from orientation O1 to O2 without collision, O1 and O2 represent two distinct classes of collision-free orientations. In 3D, however, the three-dimensional orientation space makes it difficult to classify the collision-free orientations into topologically distinct classes. For example, orientations O1 and O2 of the bar MO in Figure 4.3b are very different, and yet it is possible to rotate the bar from O1 to O2 without collisions. Without solving this problem in a general way, we use the following ad hoc method to find collision-free orientations in 3D. The major axis of MO is placed in a finite number of directions, and MO is rotated about the major 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.

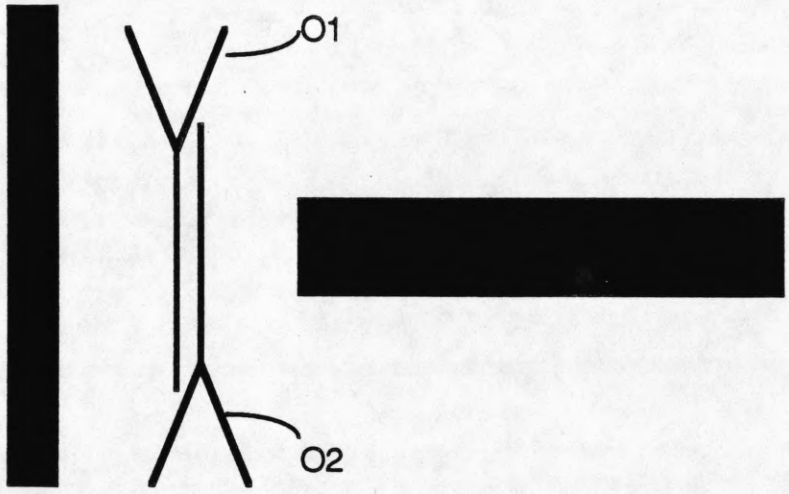


Figure 4.3a Topologically different orientations in two dimensions.

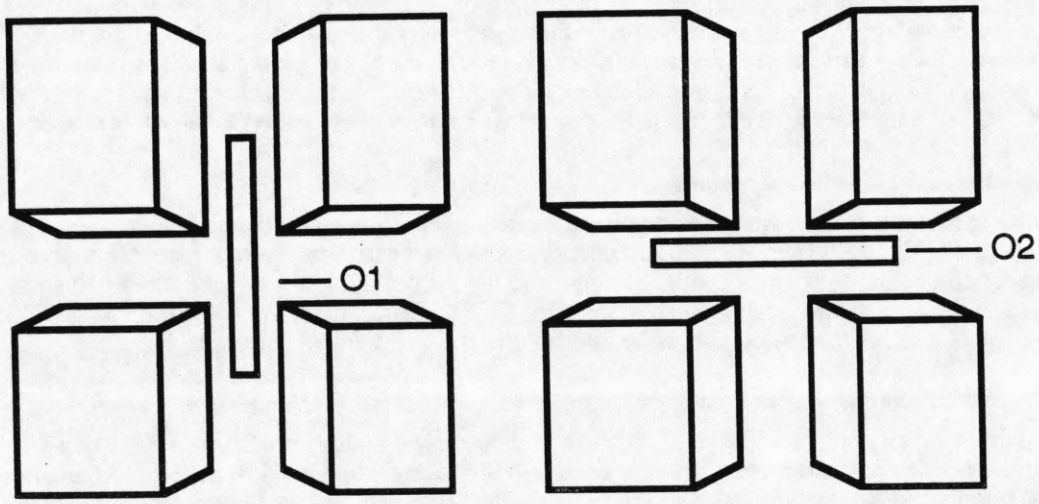


Figure 4.3b In 3D, the bar shown can move between two very different orientations without collisions by first standing vertically.

4.2.3. Connecting feasible configurations

After the feasible configurations of MO in each collision region are found, a solution to the findpath problem can be obtained by connecting a sequence of feasible configurations, one from each collision region. The 3D connecting algorithm is similar to the 2D connecting algorithm. The connecting algorithm selects from the neighboring collision regions a pair of feasible configurations having closest orientations. Once the two configurations to be connected are determined, the connecting algorithm places the MO in these two configurations and makes them move toward each other. MO in each configuration moves a step in the direction that results in the smallest potential on MO, but the movement is restricted within the 45 degrees of the direction toward the other configuration. MO then adjusts its orientation to lowers the potential on itself at the new location. This process is repeated until the two configurations are connected, or until they are shown to be unconnectable.

The remaining parts of 3D SOA are exactly the same as their counterparts in 2D SOA. MO is always moved from the configuration with the higher potential to that with the lower potential. POA can be used to optimize the solution found by SOA with respect to the objective functional in equation 3.1.1.

4.3. Examples of Findpath Problem in Two-Dimensional Space

SOA can solve far more difficult findpath problems than those solved by POA. All the examples presented in this section contain "narrow" bottlenecks in the free space around which intelligent maneuvering of MO is required to generate collision-free paths and orientations. The MOs used are a square, L shapes, a Z shape, an X shape, a T shape, a human foot and a bird. In all the examples here the solution paths always lie near the candidate topological paths. That is, MO does not have to go off the candidate path to generate the solution.

The first example is the problem of moving an L-shaped object through three polygons, as depicted in Figure 4.4. The MPV and the best candidate path for this problem are shown in Figures 2.5a and A.8d. 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.

The next example in Figure 4.5 contains the same obstacles as the previous example, but MO is a large square. The best candidate path in this example is different from that in the previous example because the cost function used to compute the best candidate path takes into account the width of MO. Since the width of MO is larger compared to the width of the space between the two rectangles, the path to the right of the rectangles has a lower cost. Another thing to note is that the algorithms in this thesis ignore the symmetry of MO and make a distinction between orientations in which MO appear to be the same. For this square MO example, the algorithm makes a distinction between four orientations in which the square looks the same, and this causes MO to rotate in the upper right, free space to get correct orientation.

The next example in Figure 4.6 is similar to the one presented earlier in Figure 1.3. Two bottlenecks are present between the start and the goal configurations, and MO has to approach these bottlenecks with proper orientations to go through. The example in Figure 4.7 appears to be a simple problem to humans, but requires the Z-shaped MO to move along a zigzag path.

Figure 4.8 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 4.8a. 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 4.8b.

Figure 4.9 shows an X-shaped MO going through a series of narrow bottlenecks by rotating itself. Figure 4.10 shows how one can change the word "CTA" into "CAT" by moving only one letter. Figures 4.11a and b show 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 4.12 shows a rigid foot putting on a shoe.

The next example in Figure 4.13 is the same as in Figure 3.4, on which POA fails. As it turns out, SOA cannot solve this problem either. 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 4.13. 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. For this reason, SOA fails to find a solution. 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 by the sidetracking algorithm discussed in the next section.

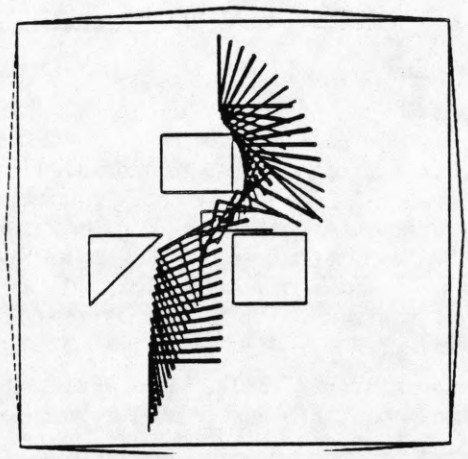


Figure 4.4 An L-shaped object goes through a tight space to minimize the path length. The L collides between the two rectangles when moved along the initial path. SOA finds feasible configurations in this region, and connects the start and goal using one of the feasible configurations.

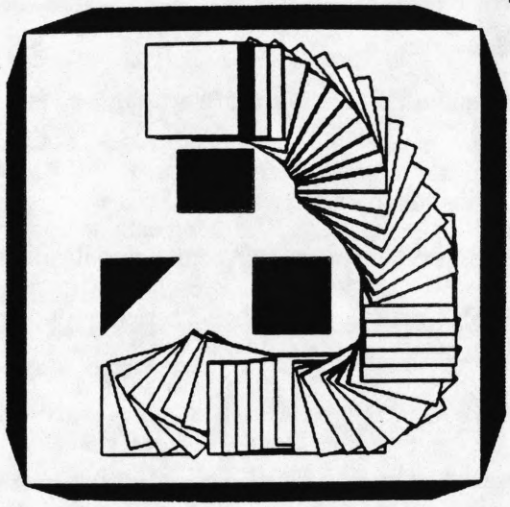


Figure 4.5 A large square object must use a wide free space at the right to reach the goal configuration.

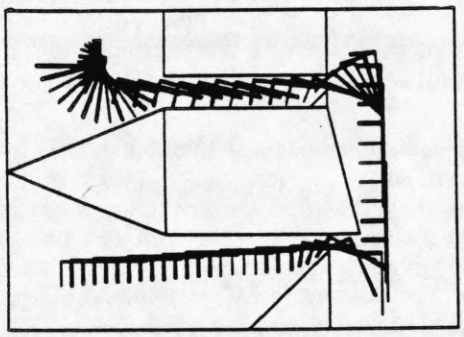


Figure 4.6 An L-shaped object goes through two bottlenecks.

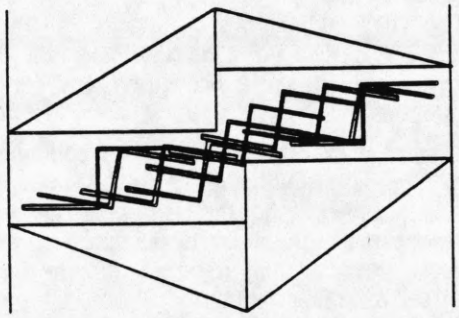


Figure 4.7 A Z-shaped object has to move in a zigzag pattern to reach the goal.

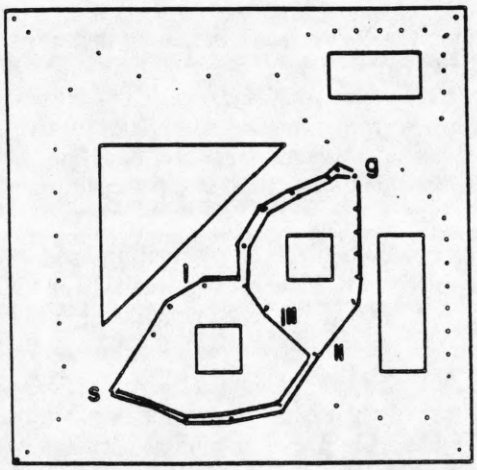


Figure 4.8a Minimum potential valleys and three topologically distinct path used by SOA.

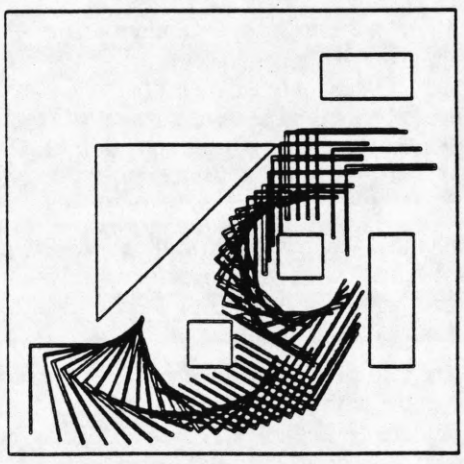


Figure 4.8b The solution obtained by SOA.

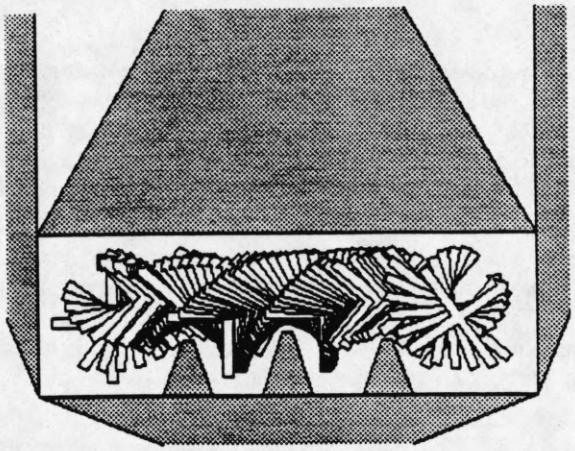
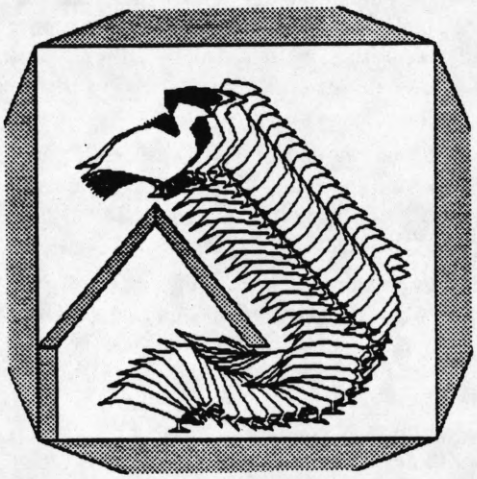


Figure 4.9 An X-shaped object goes through a series of rotations to go through three bottlenecks.

Figure 4.10 The problem of moving the letter T without touching the other letters.



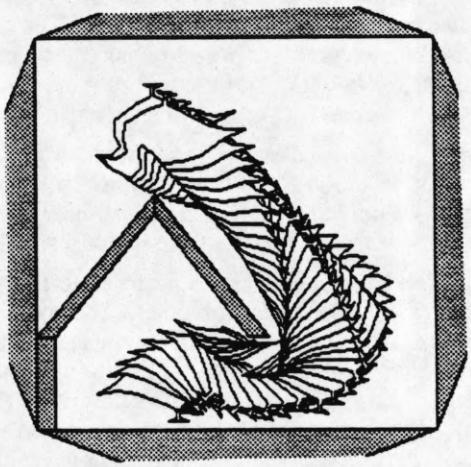


Figure 4.11 A bird has to duck under the roof to come out of the cage.

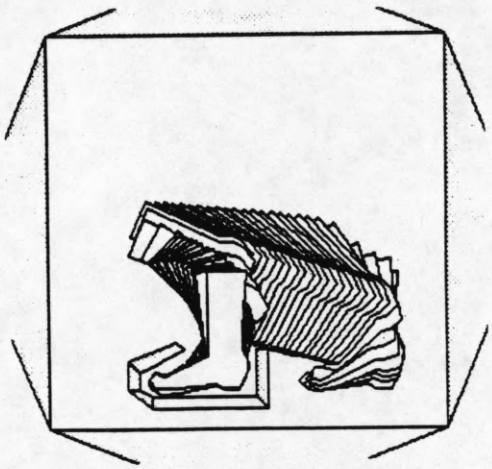


Figure 4.12 A rigid foot putting on a shoe.

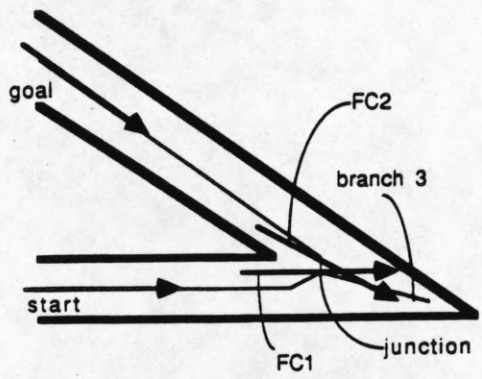


Figure 4.13 A problem for which SOA fails to find a solution.

4.4. Examples of Findpath Problems in Three-Dimensional Space

SOA is generalized to three dimensions and applied to several findpath problems. All of these examples have about the same level of difficulty as 2D SOA examples, in that the solution paths lie near the initial candidate paths and no sidetracking is necessary. The MOs in these examples are a rectangular board, an airplane, a tetrahedron, an L, a helix which is a part of a mechanical spring, a flower, and a grand piano. All the examples are such that the initial assignments of the paths and orientations cause collisions between MOs and the obstacles. The examples are presented roughly in the order of increasing difficulty.

Figure 4.14 shows a rectangular board going through narrow gaps between two pairs of rectangular boxes. A rectangular MO stands up to go through the vertical space between the first two boxes, and becomes horizontal to go through the horizontal slit between the second two boxes. This example has two collision regions. In the next example shown in Figure 4.15, an airplane is to go through an H-shaped hole. SOA successfully finds a feasible configuration at the wall and makes MO go through the wall without collisions.

In the example in Figure 4.16, MO is a tall, skinny, triangular pyramid. It is required to go through a small triangular opening and a narrow vertical slit. SOA successfully finds feasible configurations in these two places and yields a solution.

Figure 4.17 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 4.18 shows 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 particular findpath problem probably has caused most of 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.

Figure 4.19 shows a flower with an irregularly shaped root being pulled out of a vase with a small opening. Shown in isolation is the flower whose root first curves to the side, and then toward the front. The front part of the vase is not drawn. Figure 4.20 shows how to move a grand piano into the living room. Intelligent maneuvering is necessary at the doorway, whose width is smaller than the height of the piano.

The computation times to generate the solutions in the above examples are on the order of minutes for 2D examples on a SUN 260 computer. The short computation time of the 2D SOA enables us to use SOA before POA whenever there exists a collision after the initial assignment of the path and orientation of MO. Using the result of SOA makes POA converge to a locally optimal solution much faster than when the initial candidate path and orientation are used. The 3D examples described above take about 20 to 30 minutes. Most of the time is spent in generating MPV. Once the initial candidate path is given, SOA itself takes only several minutes. If there is a fast way to generate MPV, SOA in 3D space should run in the order of minutes. Above examples show the types of findpath problems SOA is capable and incapable of solving.

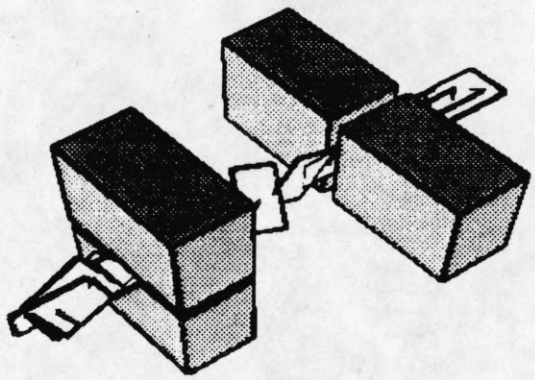


Figure 4.14 A rectangle goes through two narrow gaps to reach the goal.

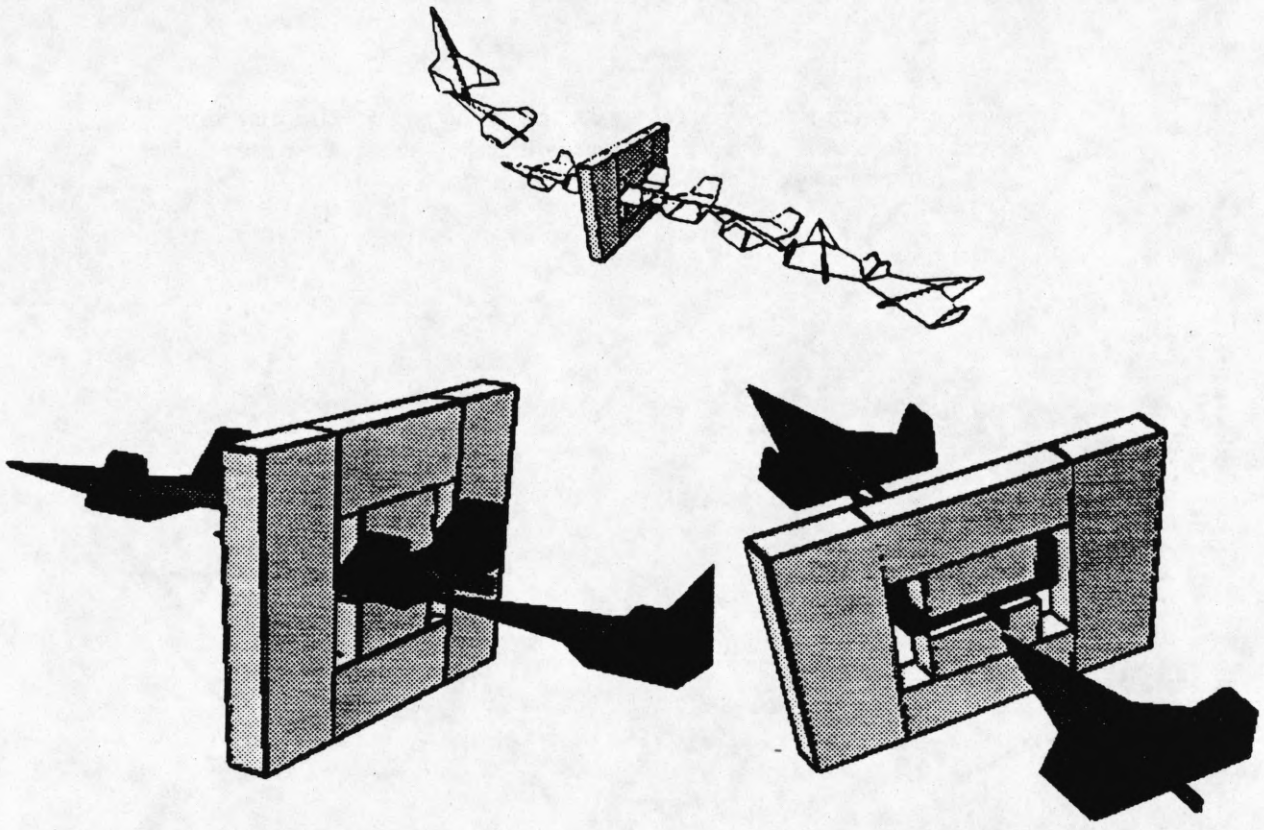


Figure 4.15 An airplane with two large vertical tail wings goes through a wall with H-shaped hole.

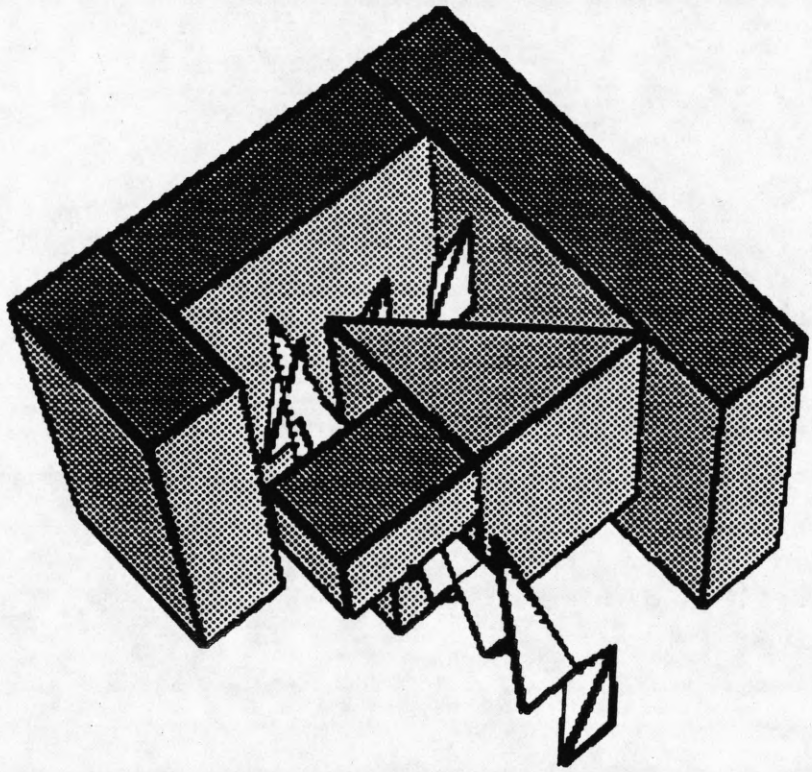


Figure 4.16 A tall skinny vertical tetrahedron tilts to go through a triangular opening, and then stands again vertically to make a turn at the narrow corner between the triangular block and the adjacent wall.

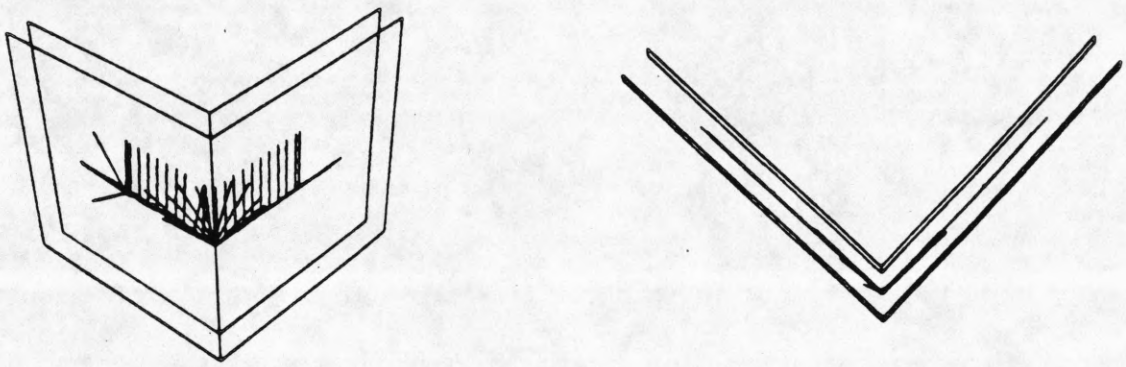


Figure 4.17 An L-shaped object turning the corner of a narrow hallway.

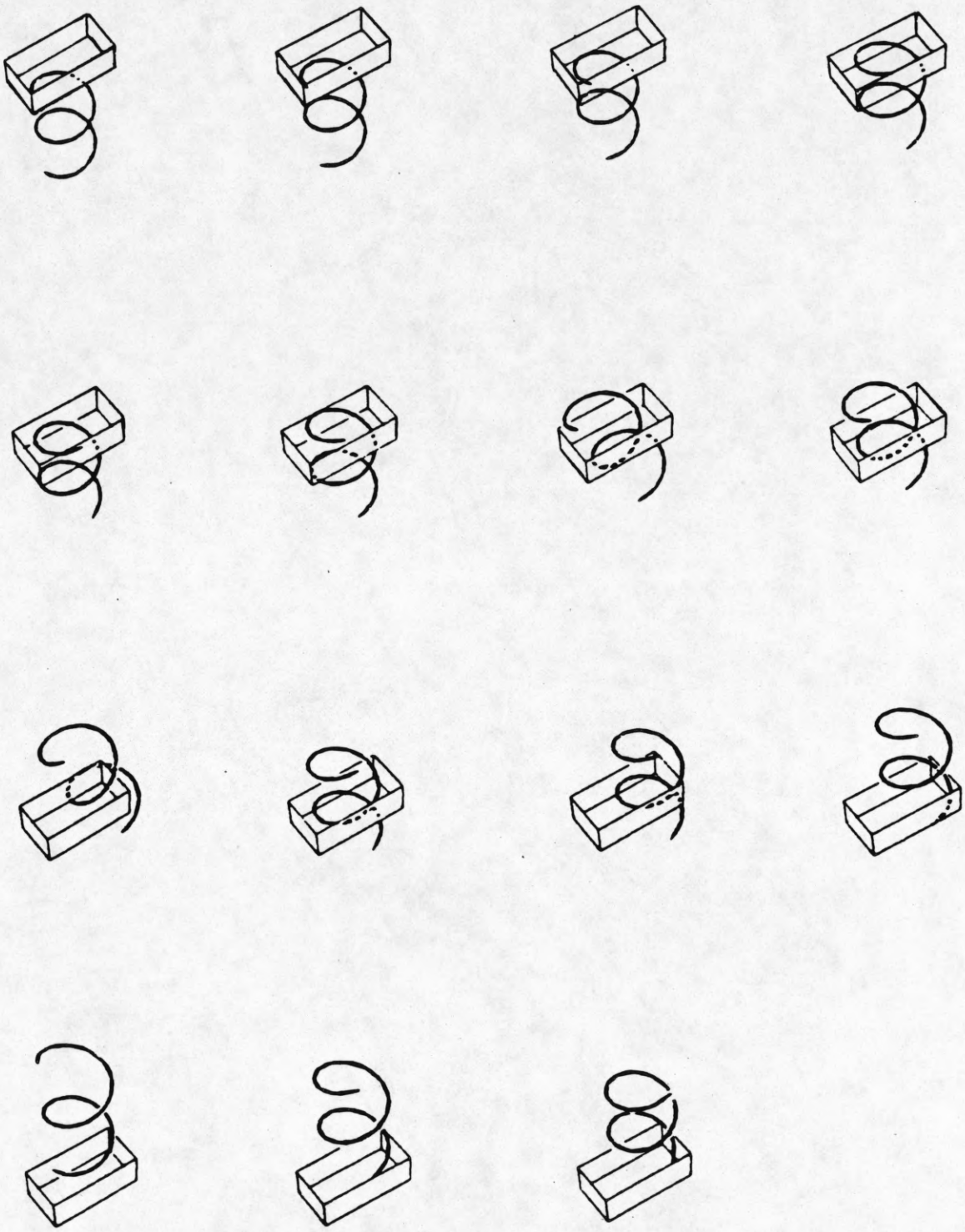


Figure 4.18 A mechanical spring has to rotate several times to go through a hole whose width is smaller than the diameter of the spring.

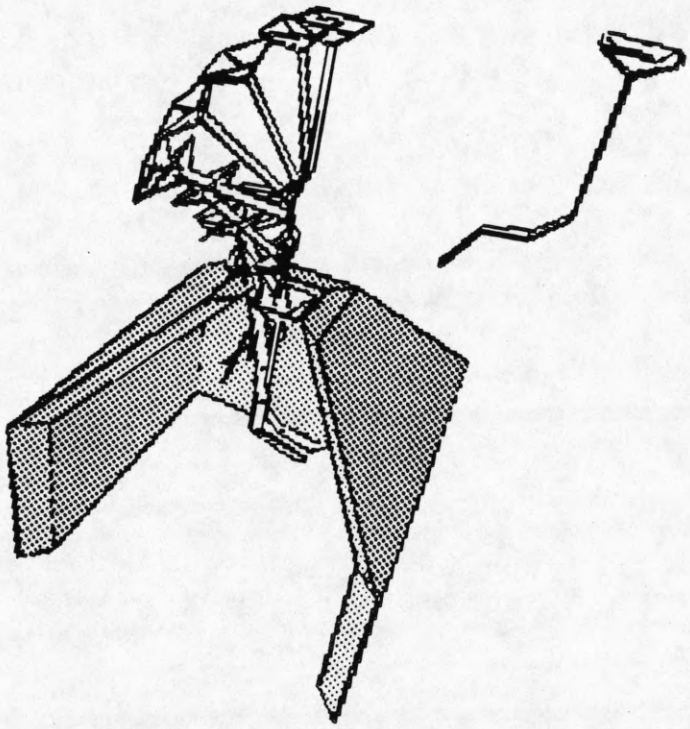


Figure 4.19 A flower with a crooked root coming out of a vase. The front part of the vase is not drawn.

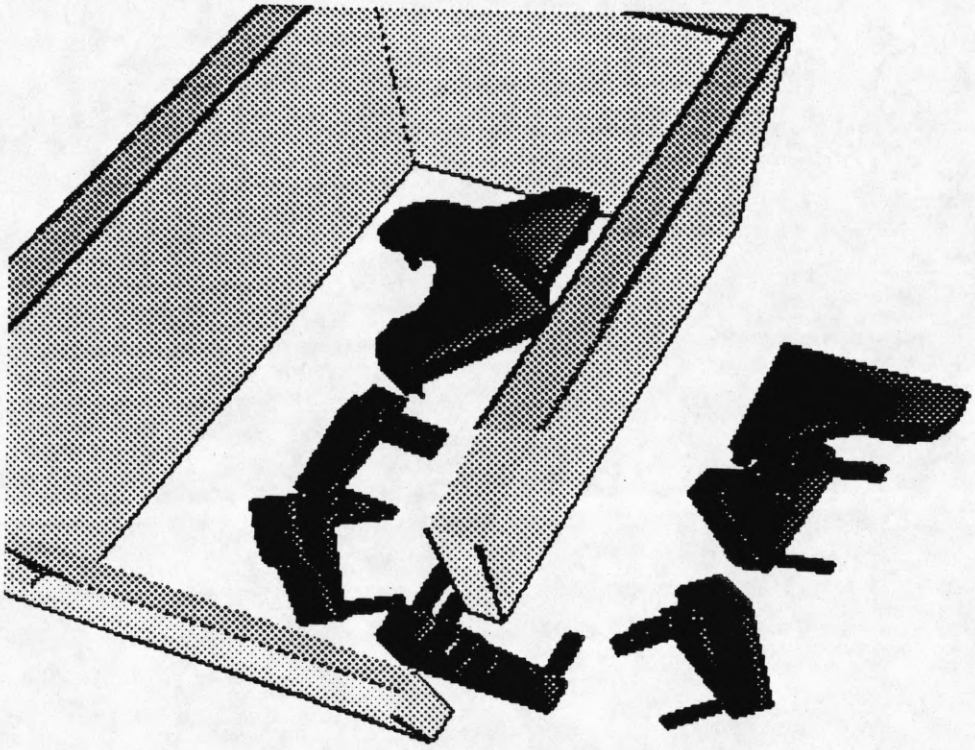


Figure 4.20 Moving a grandpiano into a living room.

5. SIDETRACKING ALGORITHM

This section deals with the hardest class of findpath 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 1.7c 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 last example of Section 4.4. Figure 5.1a 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 5.1b. 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 findpath 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 a 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 findpath problems we have considered.

The sidetracking algorithm (STA) works as follows. SOA is applied forward from the start node. When the forward SOA fails to find a solution, SOA is applied backwards from the goal node. Whenever the backward SOA connects the goal node to a feasible configuration in a collision region, containing another feasible configuration connected to the start node, STA tries to connect the two feasible configurations. To do so, STA moves MO away from the collision region along MPV. If STA succeeds in connecting them, a solution is found. Otherwise, backward SOA is continued until the goal node is connected to another feasible configuration in the collision region. The next two sections describe the details of STA for two- and three-dimensional problems. Sections 5.4 and 5.5 present the results for some 2D and 3D findpath problems that are solved by STA.

5.1. Sidetracking Algorithm in Two-Dimensional Space

Several questions must be answered to develop STA. First, it is necessary to determine where along the initial candidate path is sidetracking necessary. Once such locations are identified, the feasible configurations to be connected have to be selected. Thirdly, the direction or the path along which MO will sidetrack has to be determined. The following subsections discuss these three issues in detail. It is convenient to use an example to explain STA, and the findpath problem in Figure 5.1 is used for this purpose throughout this section.

5.1.1. Places for sidetracking

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 do so locally at point P. In other words, sidetracking establishes the connectivity of two feasible configurations in a region by taking excursion. This could be done by moving MO from the region along MPV connected to the region. There is no need, however, to connect the two feasible configurations in a collision region. This is because if MO acquires a feasible configuration FC1 in a collision region by first assuming another feasible configuration in the same region, and then coming back out of the region to get to FC1, MO could just go to FC1 in the first place. Thus, the only places of the initial candidate path from which MO is allowed to sidetrack 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. For brevity, the nodes or regions where feasible configurations are to be found will be called regions. In particular, 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.

5.1.2. Selection of feasible configurations to be connected by STA

Once the feasible configurations at the start node, the goal node, the junction nodes 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

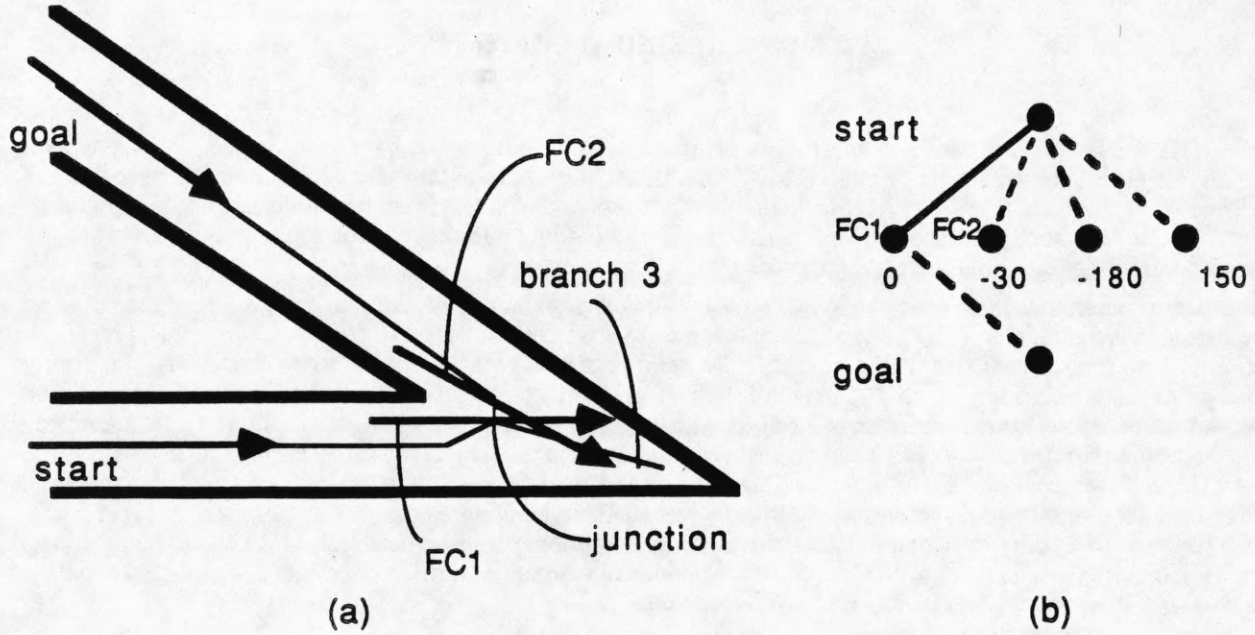


Figure 5.1 A findpath problem SOA cannot solve.
(a) The start and the goal configuration of MO (arrow) is shown along with the feasible configurations in the collision region. The thin lines denote the minimum potential valleys.
(b) The graphical representation of SOA.

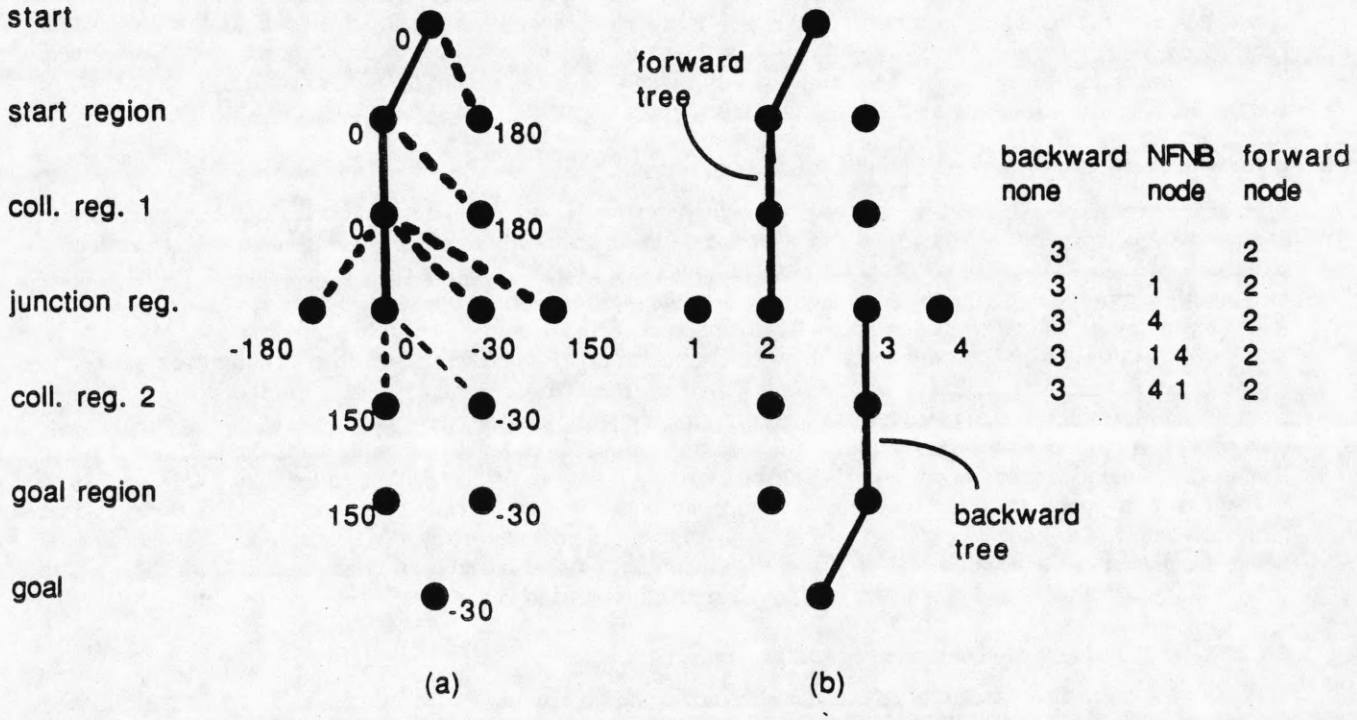


Figure 5.2 The graphical representation of sidetracking algorithm.

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 represent the configurations which can be reached from the start configuration using SOA, and these nodes will be called "forward nodes." Upon the failure of forward SOA, STA applies SOA backwards from the goal node. Since it is already known that the forward nodes cannot be connected to any of the nonforward nodes at the next level[8], backward SOA only needs to try to connect the nonforward nodes. This process spans a "backward tree," and the nodes connected to the goal configuration will be called "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, then a solution to the problem 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. That is, STA first tries to connect the backward node to one of the NFNB nodes and then tries to connect the NFNB node to one of the forward nodes. It may be necessary to use several NFNB nodes 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 to connect a backward node with a forward node continues until a solution is found or the backward tree cannot be expanded any more.

5.1.3. Direction of sidetracking

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. It needs to be determined which branch should be tried first. It is reasonable to assume that two feasible configurations may be best connected by following the branch which leads to a wide part of the free space at the shortest distance. The distance from a junction to a wide part of the free space or to the neighboring junction along each branch is computed by the algorithm in Section 2 that selects the best candidate paths among the paths in MPV. STA selects branches in the increasing order of distance.

5.2. Sidetracking Algorithm in Three-Dimensional Space

All parts of 2D STA generalize to 3D in a straightforward manner except two. In 2D STA, there are only a few junction nodes. Also, there are only a finite number of possible directions STA is to follow from a junction when sidetracking. On the contrary, in 3D most nodes have more than two adjacent nodes, and thus are junction nodes. The number of directions to sidetrack from a junction node is combinatorially explosive[9]. These two problems are handled in the following way.

Sidetracking is done at locations in free space where MO has locally maximal number of directions to move[10]. Such locations are determined in the following way. The number of neighbor nodes (also called a branching factor) is computed for each of the nodes along the candidate path. The nodes with a locally maximal branching factor are selected as the places to sidetrack[11]. At each such node, called the father node, all neighbor nodes are treated as possible successors along the sidetracking paths. For each neighbor, the corresponding path is generated in the following way: Let each successor node be called the current node. Among the nodes connected to the current node which are also on the opposite half space from the father node, the node with the minimum potential (or maximum distance to obstacles) is chosen as the new successor node. The current node now becomes the father node, and the successor node becomes the current node. This process continues until the successor node reaches a wide enough part of free space for MO to rotate freely, or the path becomes longer than a preset length.

---

[8] If connectable, forward SOA would have connected them.

[9] This is because MPV in 3D consists of two-dimensional surfaces, and there are uncountably many curves on the surfaces. Even if the surfaces are represented by a finite number of nodes, the number of curves is still combinatorially explosive.

[10] Junctions are such places in 2D.

[11] In the actual implementation, sidetracking is done at the nodes whose branching factor is greater than the branching factors of the adjacent nodes by more than some threshold. This keeps the places to sidetrack to a reasonable number.

5.3. Examples of Findpath Problem in Two-Dimensional Space

STA can solve much harder problems than both POA and SOA. We start out with the familiar example of moving a line segment around a sharp corner, which we have used extensively in the development of STA. As illustrated in Figure 5.3, the line segment goes into the dead end corner to make a proper turn. This is one of the easiest findpath problems solved by STA. The next two examples presented also require MO to sidetrack from the start and the goal region. The example in Figure 5.4 shows a car in a head-in parking position being reparked to a tail-in parking position. The car comes out to the left to change its orientation by 180 degrees. In the next example in Figure 5.5, MO has to sidetrack from the goal region. The arc-shaped MO starts out at the bottom and makes a successful turn at the corner only to arrive at the goal location in a wrong orientation. Upon failure of SOA, STA makes MO sidetrack to a wide free space at the top to change its orientation to the goal orientation.

An interesting findpath problem is shown in Figures 5.6 through 5.8. Depending on the initial and goal orientations of the arc-shaped MO, the problem presents one of the three different levels of difficulty. Figure 5.6 shows the easiest of the three. This is at the same level of difficulty as the first example in Figure 5.3. For the start and goal orientations shown in Figure 5.7, MO has to come out from the T-shaped junction to change its orientation. Figure 5.8 shows MO making two sidetrackings at the T junction. This problem is one of the hardest 2D findpath problems we have experimented with.

In the last 2D example in Figure 5.9, a fork-like MO successfully goes through a narrow space between two mushrooms. MO sidetracks to the upper-left part of the free space to change its orientation.

5.4. Examples of Findpath Problem in Three-Dimensional Space

The 3D sidetracking algorithm is used to solve several problems. MOs in the examples include a rectangular board, a submarine, a set of coordinate axes, and a chair.

Figure 5.10 shows a board coming out to a wide part of the free space to make a turn at the corner of a hallway. Figure 5.11 shows a similar example with an additional obstacle. The initial path given to STA is an L-shaped path that lies below the top surfaces of the three short obstacles. The board moves above the three short obstacles to change its orientation. The next example in Figure 5.12 shows a submarine moving through several polyhedral obstacles. It first goes beneath a rectangular block, goes through a triangular opening, and then sidetracks to change its orientation by 180 degrees.

Figure 5.13 shows an example with MO that is a wireframe of three mutually perpendicular branches. It resembles a set of Cartesian coordinate axes. Z-axis is along the axis of the rectangular free space in the middle of the four obstacles. In the starting configuration, x and y-axis are hugging the tallest obstacle. MO is to move to a configuration such that x and y-axes rest on the top surfaces of the two shortest obstacles. The reference point of MO lies at the intersection of x,y and z-axes. The initial path given to STA is the straight line connecting the positions of reference point at the starting and goal configurations. MO first moves up to rotate itself 180 degrees, then comes down to reach the goal configuration.

Figure 5.14 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.

As in the cases for SOA, STA takes on the order of minutes for 2D problems, and on the order of tens of minutes for 3D problems on a SUN 260 computer. As mentioned before, the generation of MPV takes a large portion of the computation time. A faster way to get the initial candidate path will speed up our algorithms a great deal.

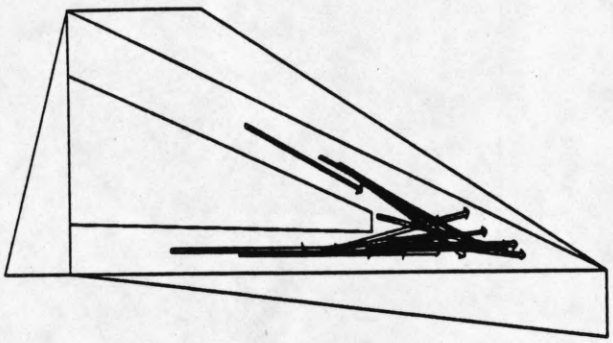


Figure 5.3 STA finds a solution to the problem shown in Figure 5.1

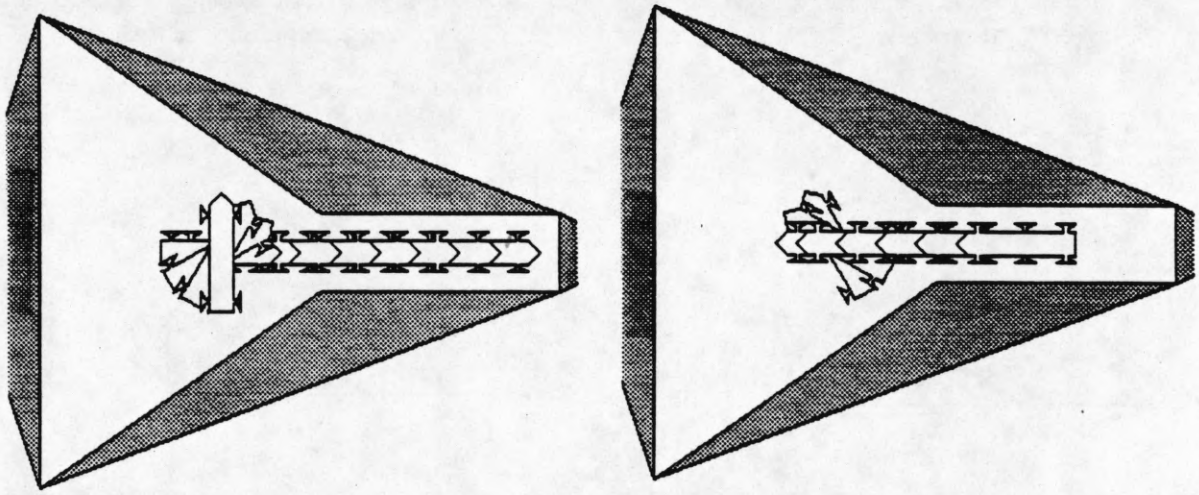


Figure 5.4 A car comes out to a wide space to go into an alley in tail-in position.

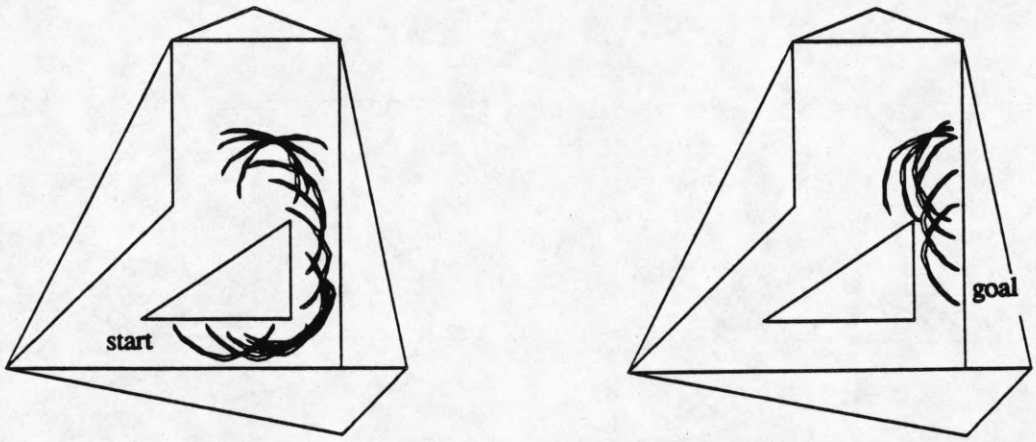


Figure 5.5 An arc-shaped object sidetracks from the goal configuration to change the orientation.

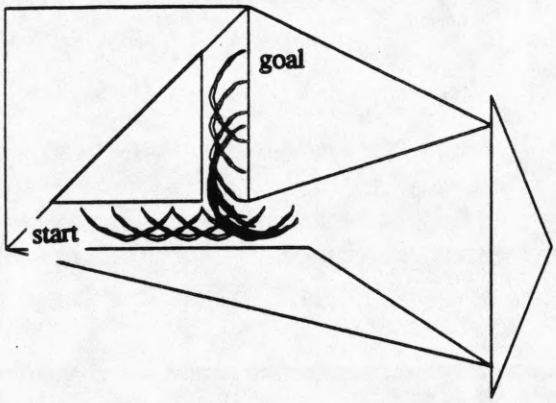


Figure 5.6 The arc needs only a slight sidetracking to reach the goal.

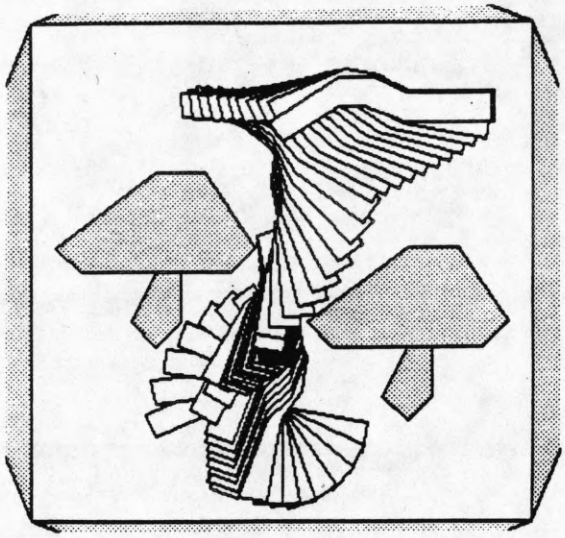


Figure 5.9 A fork-like object sidetracks to the upper-left part of the free space.

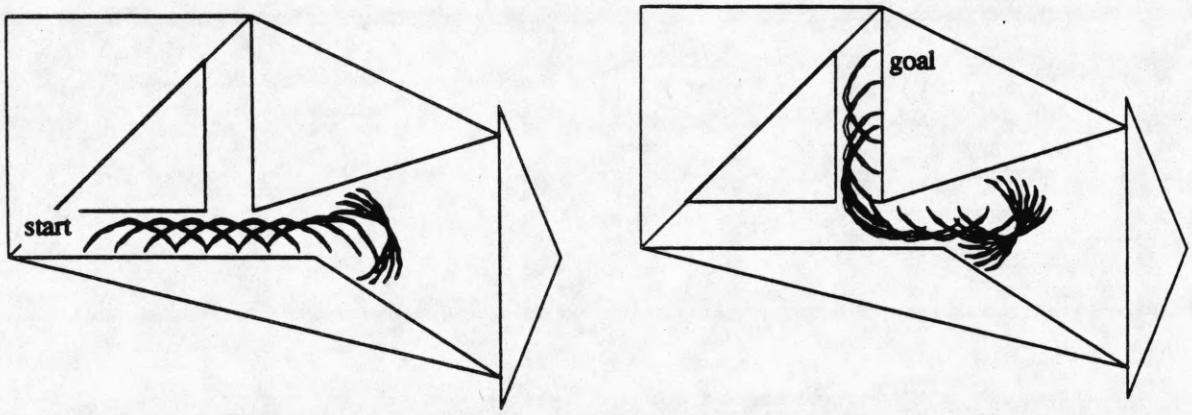


Figure 5.7 The arc has to sidetrack to the wide space to change its orientation.

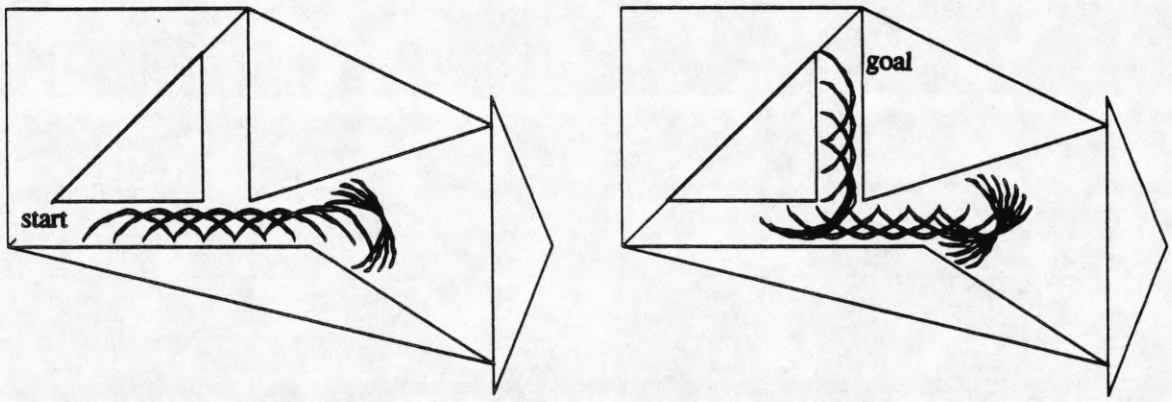


Figure 5.8 The arc needs to sidetrack twice to reach the goal.

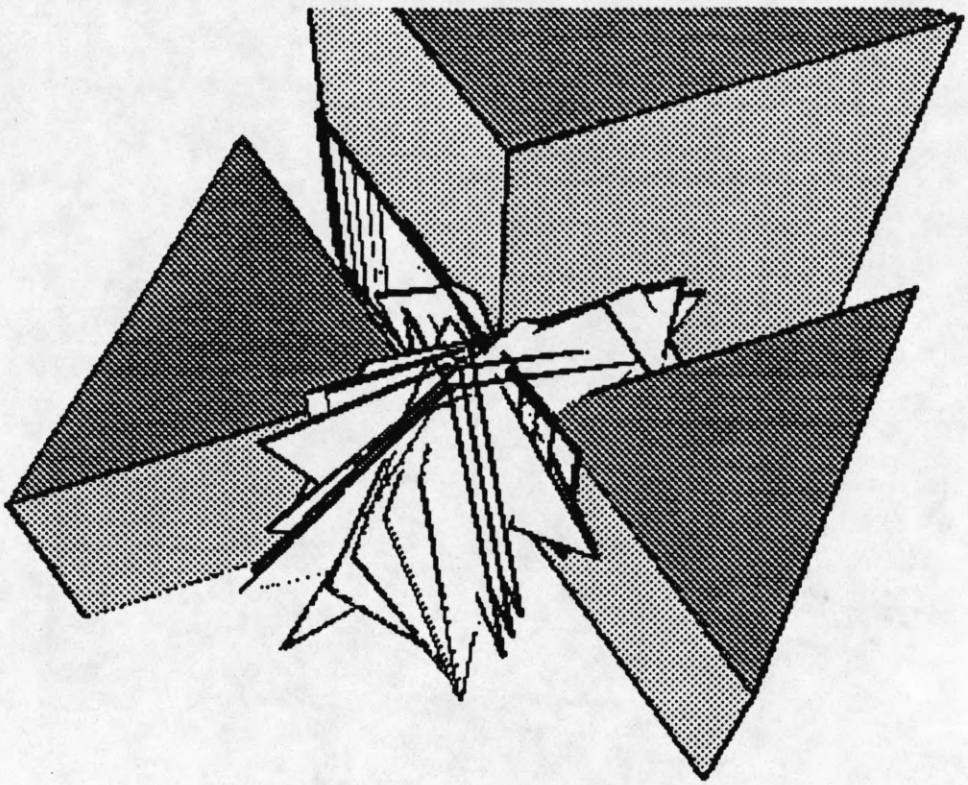


Figure 5.10 A square object can turn the corner by sidetracking to the wide space.

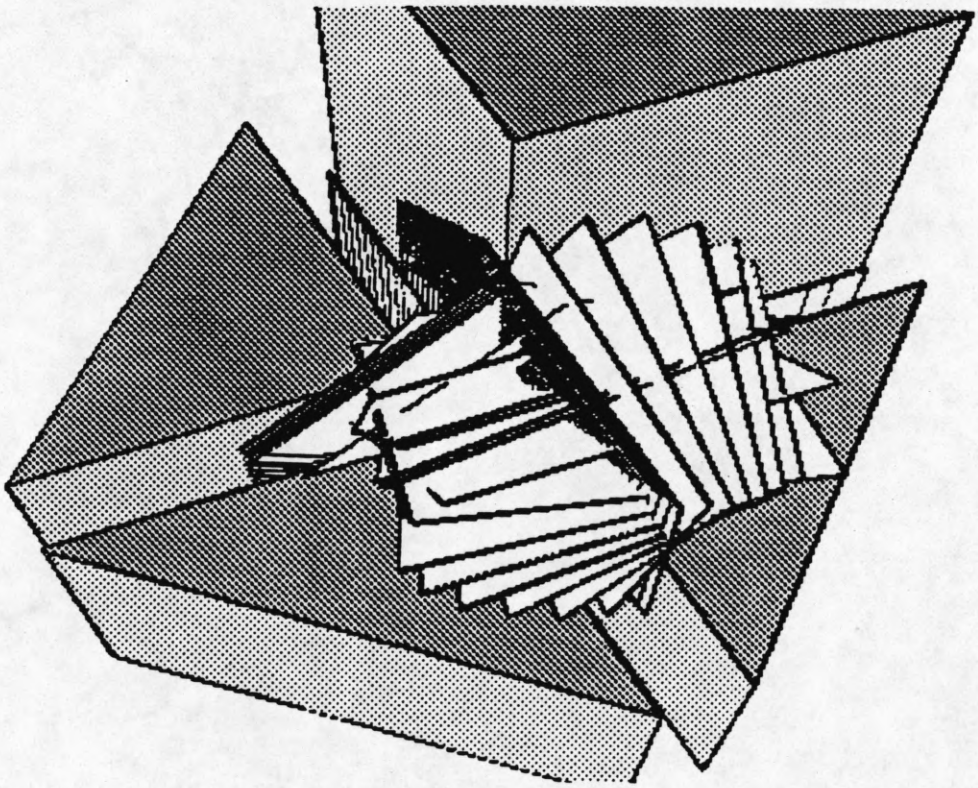


Figure 5.11 A square object sidetracks to the free space above to make a turn at the crossroad.

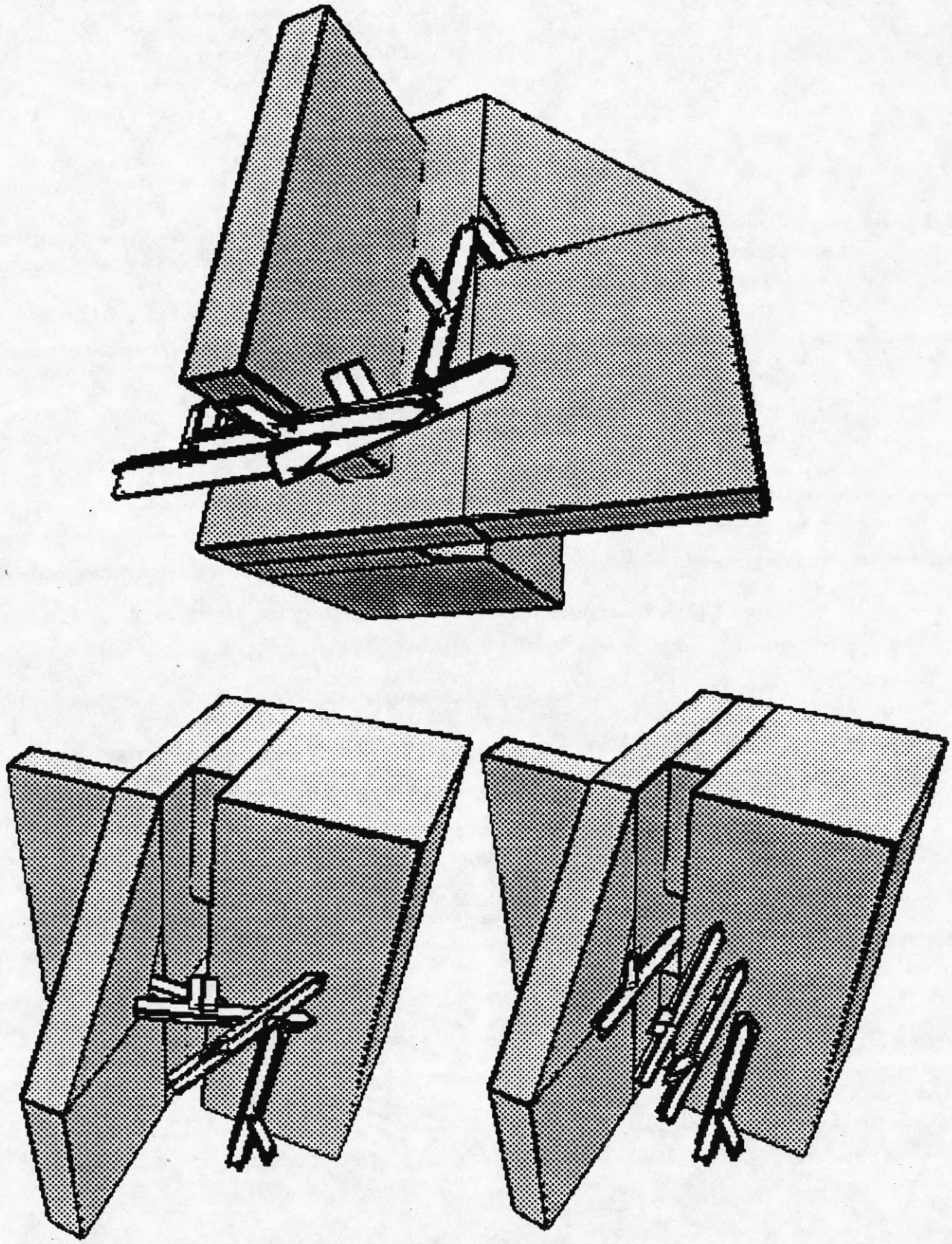


Figure 5.12 A submarine goes under a block, through a triangular opening, and then sidetracks to change its orientation.

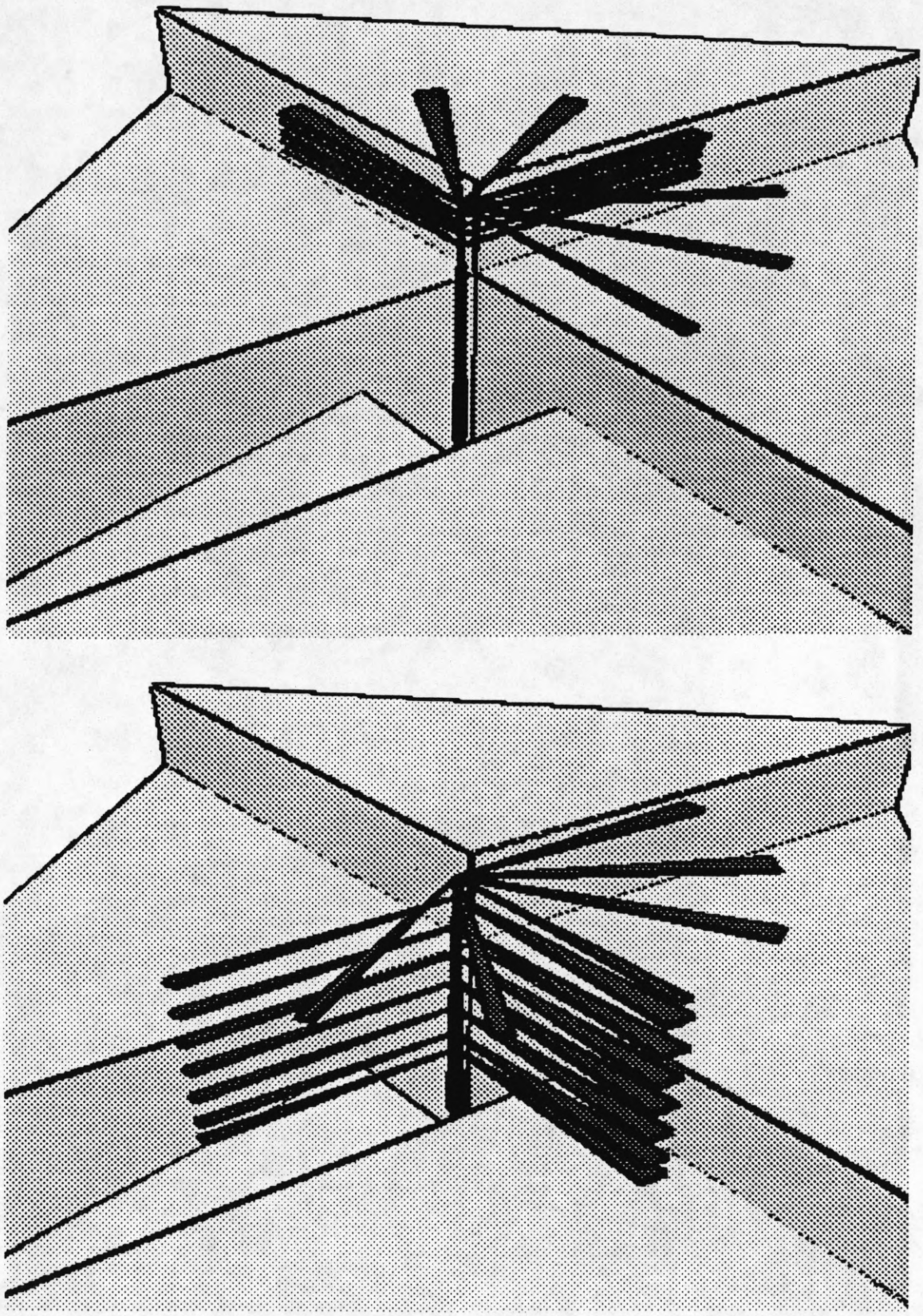


Figure 5.13 A coordinate axis rise above the tallest obstacle to change the orientation.

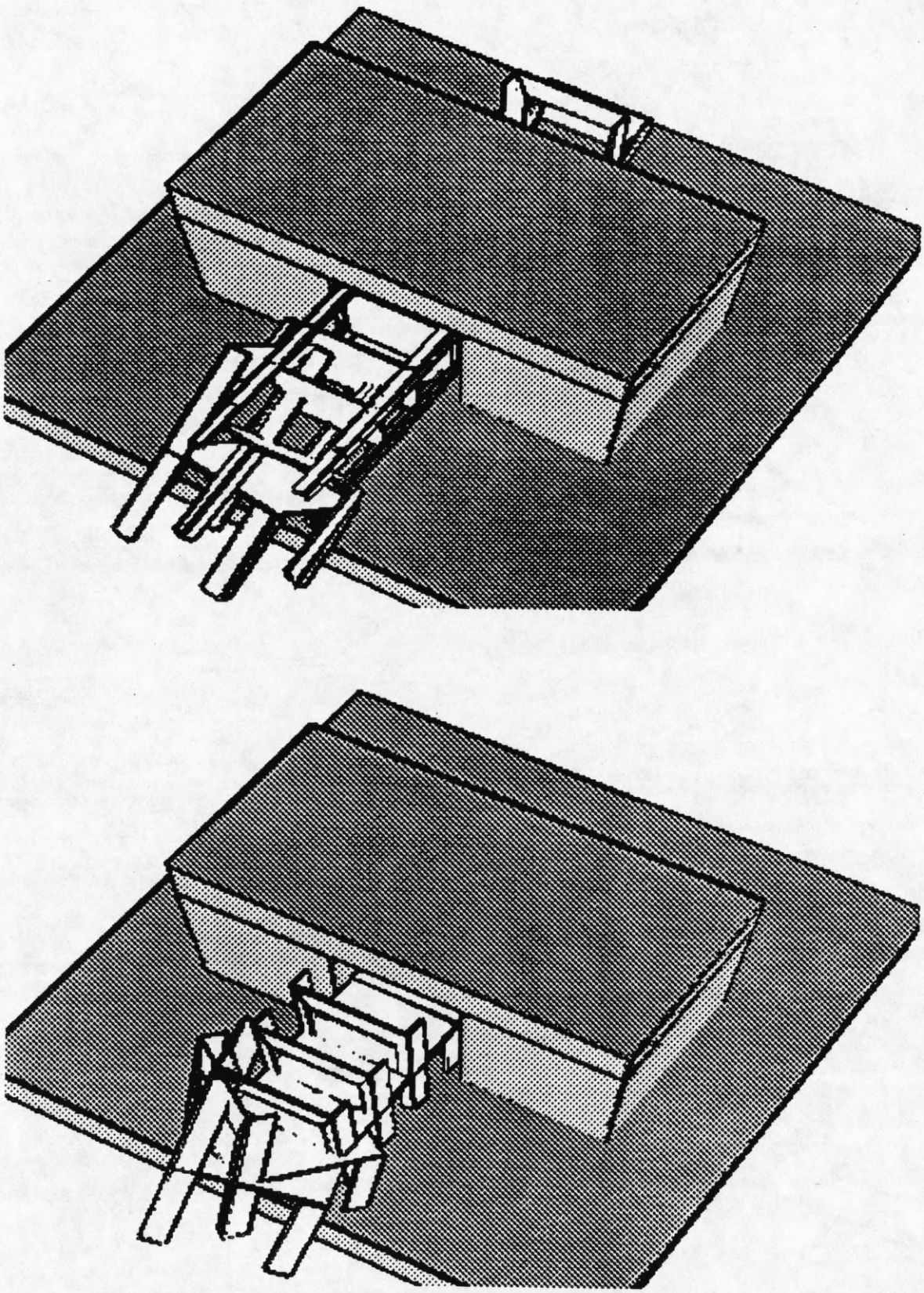


Figure 5.14 The problem of moving a chair from one side of a desk to the other side.

6. PERFORMANCE ANALYSIS AND FUTURE EXTENSIONS

We have presented an approach to findpath problem using a potential field representation. Three different levels of complexity of the findpath problem are addressed by three different stages of the algorithm. These three stages of the algorithm are described as separate algorithms in the previous sections, and their performance on a wide variety of problems is demonstrated. We have concentrated on building a general framework to solve the findpath problem rather than giving proofs of correctness of our algorithms or exact complexity analysis. In the interest of developing a complete algorithm, we have used heuristic solutions to some problems where rigorous solutions required extensive effort. Each such heuristic part, sometimes involving the use of ad hoc parameters, can be improved to make the algorithms more robust and efficient. We discuss some improvements of this kind later in this section and in Section 7.

Our approach does not restrict the shape of objects allowed. This is a significant advantage over other approximate algorithms, where objects are often limited to points or spheres. Our algorithms are much faster than the complete algorithms [DONA86, SCSH83a], but they fail to find solutions to certain problems. Exact characterization of such problems is difficult due to the heuristics used. Instead, we give a qualitative description of such problems along with examples in the next subsection. We have not been able to establish a polynomial-time bound for our algorithms, rather their complexity is measured by the actual computation time. Our algorithms take less than 5 minutes for all 2D examples presented, and 5 to 30 minutes for 3D examples on a SUN 3/260 computer. These times are quite short considering the computation time of hours reported for one implementation of of the configuration space approach [DONA86] and the theoretical estimates available for the algorithm of Schwartz and Sharir.

6.1. Limitations of the Current Algorithms

There are classes of findpath problems that give our potential field approach a great deal of difficulty. Two conditions must be met in order for our algorithms to succeed. First, MPV must provide good initial estimates for solution paths. In cases where the obstacles and MO are of complicated shapes and interlocked together, MPV between obstacles may not resemble a solution path at all. Second, rotation of MO about its reference point must produce important changes in the configuration of MO. Figures 6.1 and 2 show examples which violate the first and the second condition, respectively.

A maze problem is shown in Figure 6.1. If the walls are obstacles and the mouse is MO, our algorithms will solve the problem rather easily. On the other hand, if the mouse is assumed to be the obstacle and the collection of the walls to be MO, our algorithms will fail immediately. For the problem of unraveling two objects which have equally complicated shapes and are interlocked together, our algorithm will fail regardless of which object is considered as MO. In the example in Figure 6.2, the crucial parts of MO are its extremities. If the reference point of MO lies near its center, any rotation of MO about the reference point will not produce a movement needed to unhook MO from the nails. A common feature in these examples is that MO is very concave. Solving these problems will require a detailed study of crucial parts of MO, which can be used to synthesize the motion of the whole MO.

6.2. Possible Extensions

The changes discussed below for the various parts of our algorithms should help improve the performance from the point of view of both completeness and efficiency.

6.2.1. Generation of minimum potential valleys

Minimum potential valleys are generated by filling the free space with circles (spheres) and using their centers to represent the valleys. It takes a long time to fill narrow and long space with round shapes, especially in 3D. As it is implemented now, it takes on the order of minutes for 2D and tens of minutes for 3D to generate MPV. A new algorithm that uses elongated shape primitives such as ellipsoids or rectangloids as well as spheres will fill the free space much faster. Selecting appropriate shape primitives for each part of the free space will improve the performance of such an algorithm.

The current potential field approach generates all of MPV before determining an optimal candidate path and applying the findpath algorithms. It seems highly plausible to get an optimal candidate path once some paths connecting the start and goal positions are found while generating MPV. This allows the findpath algorithms to be applied at an early stage, and can reduce the computation time.

6.2.2. Parallel optimization algorithm

The gradient algorithm is used to minimize a weighted sum of the path length and the potential on MO over the entire path. When the path is long and passes near many obstacles, the the variation in the magnitude of the gradient is very large along the path. The gradient is large near the obstacles, and small in wide free space. Large gradient changes the path drastically during iteration, and often makes the gradient algorithm fail to converge. For this reason, the gradient has to be scaled down. However, the gradient algorithm then takes a long time to converge to an optimal path in wide free space. A remedy for this problem is to partition long paths into meaningful segments and apply the gradient algorithm to each segment separately. This would, however, leave unchanged the locations of the end points of each segment of the path. A different partition would be necessary to change the location of the end points of the first partition. This process of partitioning and minimization will have to be repeated several times before the final path and orientations are determined.

6.2.3. Serial optimization algorithm

SOA is nearly complete for 2D, whereas it needs to be improved in the following parts for 3D.

First, in the current implementation, feasible configurations of MO in narrow parts of the free space are found in a brute force way. MO is positioned such that its center coincides with the center of the collision region. The major axis of MO is placed in a finite number of directions, and MO is rotated about the axis to find the orientations

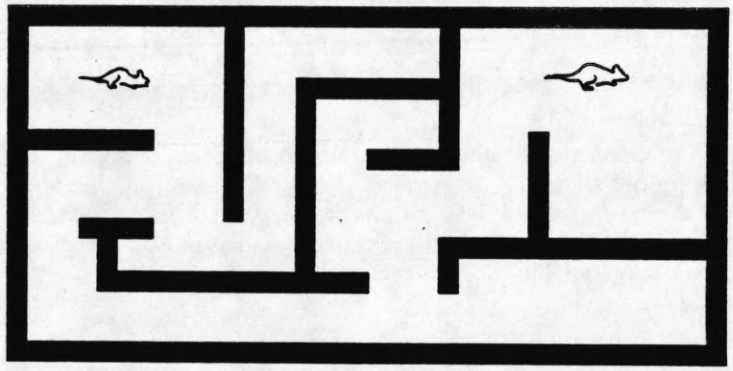


Figure 6.1 Dual nature of findpath problem. In this example, the walls can be considered as the obstacles and the mouse as the moving object, or vice versa.

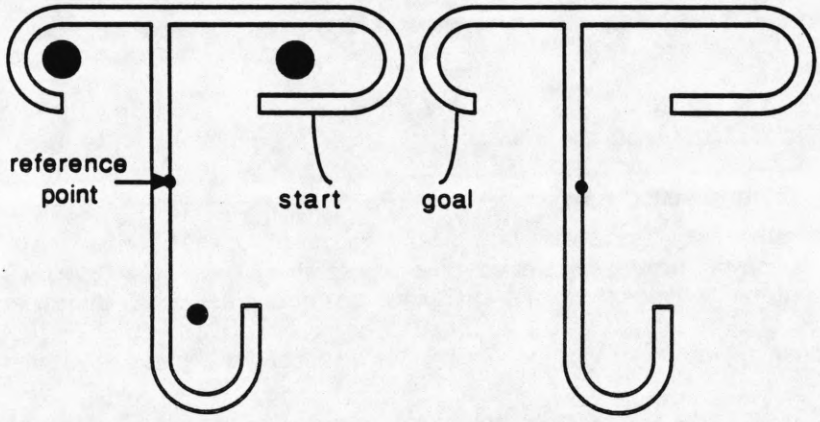


Figure 6.2 A problem illustrating that the rotation of MO about its reference point does not always produce the necessary movement to find a solution.

with locally minimum potential. Then the orientation and position of MO are allowed to change within some limits to further lower the potential. This method of finding feasible configurations has many disadvantages. First, it is computationally expensive. Even for a moderate number (about 20) of directions for the major axis and for a similar number of orientations about each direction, the potential on MO has to be computed a few hundred times. Second, the number and selection of the feasible configurations is determined relatively arbitrarily. The shape of the collision region is not taken into account in selecting the configurations. Consequently, while some of the selected configurations may be redundant, other useful configurations may be completely missed. For example, when the geometry of the narrow free space is relatively simple, all but a few feasible configurations may be topologically identical. But, when the shapes of MO and the free space are complicated, some of the crucial feasible configurations needed in finding a solution may not be selected. The problem of missing significant configurations may be avoided by increasing the number of orientations to be tested at the expense of increasing redundancy and computation time.

In the last stage of the connecting algorithm, the positions and orientations of two configurations to be connected are interpolated, and MO is tested against collisions in these intermediate configurations. If no collisions occur, the two configurations are considered connected. Strict interpolation of orientations, however, does not always yield an optimal way to change the orientation as illustrated in Figure 4.8. The current implementation allows the longest axis of MO to change within 45 degrees of the strictly interpolated orientation. When MO has to change its orientation in a narrow space of complicated shape, the current algorithm may well miss a sequence of collision-free orientations that will move MO into the desired orientation.

Enumerating all possible configurations and relative locations, which we have attempted in our algorithms as discussed above, defines a search based method of identifying all feasible configurations in a uniform sampling of the configuration space. Ideally, the sampling of the configuration space should be adaptive so that a single sample comes from a class of topologically equivalent samples. Thus, only topologically distinct configurations are chosen based on considerations of compatibility between the shapes of the collision region and MO. In our potential field representation, such a selection of configurations may be approximated by estimating the potential on the object as a function of the orientation and location changes in the vicinity of the collision region, and treating those configurations that lead to local potential minima as candidates for feasible configurations. The search for the minima may be made efficient by a coarse-to-fine computation of the potential on MO. The problem of connecting selected collision-free configurations is also simplified by the use of the potential function instead of uniform interpolation of intermediate configurations, as done in our current implementation. The selection of the intermediate configurations could be done to, for example, homogenize the change in potential between adjacent configurations.

An alternative to using the potential function for the necessary spatial reasoning, more direct shape representations may also be used. A method of representing objects in terms of canonical shape primitives forms may help find topologically distinct and meaningful feasible configurations. For interpolation also, the same canonical representation of objects could be used.

6.2.4. Sidetracking algorithm

Sidetracking algorithm is used to change the orientation of MO when it is not possible to do so locally. The paths along which MO sidetracks from a region are well defined in 2D. They are the branches of MPV connected to the region. In 3D, however, paths connected to the region are minimum potential surfaces, and the nodes on the surfaces are heavily interconnected. The number of possible paths is combinatorially explosive. The current implementation makes MO sidetrack following the path with minimum potential. Although the current implementation makes MO to go to a wide part of the free space quickly, MO may miss some of the paths with higher potential which still lead MO to a part of the free space wide enough to change its orientation. A better method to sidetrack would be to determine the part of the free space where the orientation of MO can be changed, and then move MO to that place. This will be much more efficient than the current implementation, in which MO sidetracks to follow a low potential path.

The solution found by the sidetracking algorithm may not be near optimal. The potential field approach first finds best candidate (topological) path and uses SOA to find a solution. When SOA fails, there are two choices. The next best candidate topological path can be found, and SOA can be applied to that path; or STA can be applied to the first candidate path. Since STA often increases the path length, it is hard to judge in advance which method would result in a shorter path. A possible way of resolving this problem is to define a measure that estimates the path length of the solution from STA and compare it with the path length of the second best candidate path. Then the program would shift back and forth between SOA and STA, whichever has a shorter estimated length of solution path. This requires storing of large amount of the intermediate results which may be used in the future.

7. SUMMARY AND CONCLUSIONS

We have presented an approach to findpath problem using a potential field representation. A potential field similar to the electrostatic potential is used to accomplish two things. First, the topological structure of the free space is extracted in the form of the minimum potential valleys. The valleys define a network, much like the Voronoi diagram for point obstacles, in which different paths represent topologically distinct connections. Second, the potential field is used to derive the most efficient collision-free path corresponding to a given topological path. Three levels of difficulty associated with findpath problems are identified. The first level problems are those in which the free space between obstacles is relatively wide, and MO does not have to undergo much twisting and turning to avoid obstacles. The parallel optimization algorithm solves such problems by finding the collision-free path and orientation that minimize path length and the potential on MO along the path. When MO must maneuver through tight spaces, a straight application of POA may not find a path even if there exists one - neither the initial estimate of the solution nor the optimization process may find collision-free object configurations in tight spaces. Such problems are said to possess the second level of difficulty. The serial optimization algorithm solves these problems by searching for safe object configurations in collision regions and constraining the global solution to go through these safe, intermediate configurations. Thus, a divide-and-conquer approach is used to formulate the overall problem as a set of the following subproblems: connect source to the first intermediate configuration, connect successive intermediate configurations, and finally connect the last intermediate configuration to the destination. Each of these problems is attempted by the POA.

It is possible that in a given region no collision-free configuration exists that can be connected to other configurations on each side. However, there may exist two different collision-free configurations in the same collision region, each of which connects to another configuration on only one of the two sides. Thus, a solution can be derived if the two different collision-free configurations can themselves be connected. This property characterizes the problems possessing the third level of difficulty. These problems are again solved by the divide and conquer approach, but a collision region here gives rise to three subproblems and not two as in case of SOA. The third subproblem is that of connecting the two different collision-free configurations in a single collision region. The sidetracking algorithm thus performs a three-way divide-and-conquer. To solve the third subproblem requires taking an excursion away from the initial estimate of the path defined by MPV, which leads to a solution which is topologically different from the initial estimate. Typically, STA moves MO away from the initial estimate into open space, changes its orientation there, and brings it back to the collision region to continue on. In contrast, the problems solved by both POA and SOA are such that the solution path lies near the initial estimate. Having applied the POA to a given problem, the passing of control to SOA and, if necessary, to STA, is achieved automatically.

Two principal motivations have lead to the approach presented in this paper. The first is to derive a smooth path that allows a smooth motion of MO, i.e., the path should be a smooth spatial curve and the object translation and rotation should be smooth functions of time. The second is that any given part of the trajectory should be determined by only those obstacles with which MO could potentially collide while on that part of the trajectory. Both of these goals are served by using the potential field to represent the obstacles. The potential at a given point is completely unaffected by obstacles behind other, closer obstacles; it is determined by the locations of the surrounding surfaces. Further, the potential field is spatially continuous and an analysis of local potential variation helps determine appropriate continuous changes desired in object location and orientation to obtain smooth motion. Effectively, the measure of continuous distance to the obstacles available through the potential field is used like the viscosity function of an anisotropic fluid medium through which MO must swim while minimizing the total resistance.

In comparison, those approaches that represent the obstacles directly in geometric terms (e.g., as polyhedra) must explicitly check for intersections with the primitives of the representation (e.g., faces and edges of polyhedra). Thus, the time complexity is determined by the combinatorics of intersection detection. A typical intermediate stage in the algorithms that use geometrical representations is to identify the safe regions in some way. Finding a smooth motion path constitutes a separate, independent step of traversing the safe regions. For example, consider the two complete algorithms which use geometric representations of free space. The configuration space approach generates all the boundary equations of the configuration obstacles[1] first, and then finds a collision-free path using standard search techniques. Similarly, the critical curve approach divides the free space into noncritical regions (see

[1]The configuration obstacles are the collection of the points in the space of position and orientation of MO that result in collisions between MO and the obstacles.

Section 1.2.2), and the search for a solution is done by checking the connectivity of the noncritical regions. In each case, the search typically yields paths with corners; both the geometrical path and the MO orientation along the path must be processed separately to make them smooth. Ideally the smoothing should be done to ensure compatibility between MO orientation and the nearby obstacle shapes. To do this requires further geometrical computation on the primitives of the representation. Both of these complete algorithms incur significant effort in analyzing those parts of the free space through which a solution path may never pass.

The potential field approach effectively performs a multiresolution analysis of the free space. The first step of extracting all topologically distinct paths between the source and the destination requires a relatively simple computation, that of computing the potential field. Then by performing a more complex computation of heuristics, it selects likely candidates for the best solution path. Finally, using the most expensive, iterative computation of applying a subset of POA, SOA and STA, the initial path is modified to minimize the cost. Such coarse-to-fine organization of computation - performing more complex computations on selected, smaller parts of the space - minimizes the total amount of computational effort. In our case, the initial candidate path is selected from MPV based on the length of the path and chance of collision. Simple heuristics are used to measure the chance of collision, and therefore, the final optimal path found from the best initial path may have a higher cost than the globally optimal solution. However, a solution obtained by the potential field approach is near optimal in the sense that it is of short length and has a reasonable safety margin between MO and the obstacles. If finding the most economical solution is the goal, then other competing initial topological paths must also be considered and the solution selected based on the optimized costs.

Two other consequences of using the potential field representation (and similar other continuous representations of obstacles) are worth noting. First, the computation of the representation is highly amenable to parallel and analog computing. Any fine grain parallel architecture or particularly an analog system can rather easily implement the computation of potential field. Considering that this takes typically about 70% of the total computation time required by the algorithms currently, such parallelization would lead to a significant improvement in performance. Second, extending a findpath algorithm so it solves problems in higher dimensional space involves marginal increase in complexity. For example, there are no serious basic differences between our 2D and 3D algorithms.

There are some obvious ways in which the computational efficiency of our current implementation could be improved. These ways were not paid attention to in our current implementation in the interest of completeness of the implementation. For example, the algorithms of finding MPV can be improved by filling the free space with additional shape primitives along with the sphere. Spatial reasoning can be incorporated into the methods of finding feasible configurations in collision regions. The structure of the potential field gives a great deal of information about the free space, especially in collision regions, and those configurations with locally minimum potential can be used as topologically distinct feasible configurations. Or the heuristics of selecting the best candidate path can be made dependent on the shape of MO in order to detect possible collisions at an earlier stage. All of these will significantly decrease the computation time of our algorithms.

The next line of research would be to extend the potential field approach to findpath problems with conformable objects, especially linked objects. Almost all general purpose manipulators consist of rigid subparts linked together, and trajectory planning of such bodies to perform various tasks is of great importance. In such problems, the mechanics of the linked objects is as important a factor in designing the trajectory as the topology of the free space. For the problems with flexible objects such as those made of rubber, the potential field approach seems to be quite appropriate; for example, a new type of potential function due to the bending of objects can be included in the total potential field to allow the distortion of the objects to some extent. Other future extensions include elimination of the limitations of our algorithm described in Section 6.1, and incorporation of the extensions listed in Section 6.2.

APPENDIX - IMPLEMENTATION DETAILS

A.1. The Potential Function

In the definition of the potential function due to an obstacle that we have used (Sec. 2), namely,

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

$g_i(x)$ must be expressed in a normalized form in order for the potential to decay at the same rate in all directions as the distance from the obstacle increases. One way to normalize is to have $g_i(x)$ equal to the minimum distance between the location x and the boundary segment corresponding to g_i. Failure to do so will result in level curves that are not uniformly spaced in all directions as shown in Figure A.1. The above definition of potential function makes it very sensitive to changes in obstacle shape. Consider the two 2D obstacles in Figures A.2a and b. Although they are almost identical, their level curves are drastically different, which is undesirable. One remedy to this problem is to weigh each boundary segment by its size. Thus, for polygonal obstacles the normalized $g_i(x)$ for each boundary segment can be multiplied by the length of the segment. This ensures the continuity of the potential function with respect to the shapes of obstacles. The modified potential function is then

$$p(x) = [\, \delta + \sum_{i=1}^{\text{no. of bound. seg.}} l_i (g_i(x) + |g_i(x)|) \,]^{-1}.$$

The potential function due to multiple obstacles is then maximum of the above $p(x)$ due to each obstacles. Figure A.2c shows the level curves of the modified potential function of the obstacle in Figure A.2b.

A.2. Algorithms For Finding Minimum Potential Valleys

A.2.1. Minimum potential valleys in two dimensions

MPV will generate all the topological paths except the outermost paths. Artificial obstacles are introduced around the whole problem space to generate the outermost paths. It is desirable to have the fewest points possible to represent the curves of MPV to reduce computation, e.g., in a polygonal approximation of MPV. Of course, the polygonal edges should not cross the obstacles. This is ensured by making the distance between adjacent vertices less than MD of one of the points.

In generating MPV a queue and two lists are kept. The queue contains the nodes whose adjacent nodes are yet to be found. It initially contains the start and the goal nodes. One list called a node list contains all the nodes generated. It also contains the start and the goal nodes initially. A second list is a temporary node list. The first element of the queue is called the current node. First, the MD of the current node is computed[13]. Then a circle of radius MD centered at the current node is drawn, and the potential values along the circle at discrete angular intervals are computed. The points of locally minimum potential along the circle are selected as the adjacent nodes of the current node. The adjacent nodes are called the sons of the current node, and the current node is called the father of the adjacent nodes. The current node is removed from the queue after its sons are generated.

If the current node has more than one son, the current node is marked as a junction node. The sons are then examined to see if they can be connected to any of the existing nodes in the node list other than their father. This is done by checking whether the distance between one of the sons and an existing node is less than the MD of the son. If there is an existing node that satisfies this, it is considered as an adjacent node of the son and a proper connection is made in the data structure. If there are more than one such existing nodes, than the node closest to the son is selected as the adjacent node of the son. This connecting process is done for each of the sons. The sons that are not connected to any of the existing nodes are added to the end of the queue, since their adjacent nodes are yet to be found. The sons are then added to the node list, and the next current node is taken from the front of the queue.

---

[13]Many algorithms are available for computing MD from a point to a polytope. There exist O(logN) algorithms in two dimensions and O(NlogN) in three dimensions [GIJO85], where N is the number of vertices of the polytope. As noted by Gilbert, however, the asymptotic behavior of the algorithm is less important than the multiplying constant. This is because the number of vertices of a typical obstacle in findpath problems are not many, usually less than 10. The distance algorithms are thus selected according to their complexities for small N. The minimum distance algorithms in two and three dimensions are described in Appendices 1 and 2.

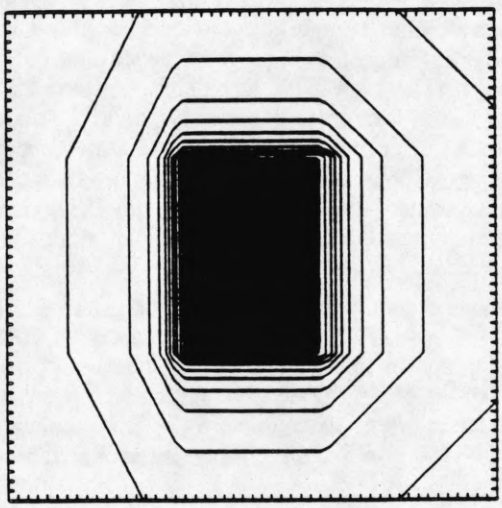


Figure A.1 Lopsided potential function due to unnormalized $g_i(x)$.

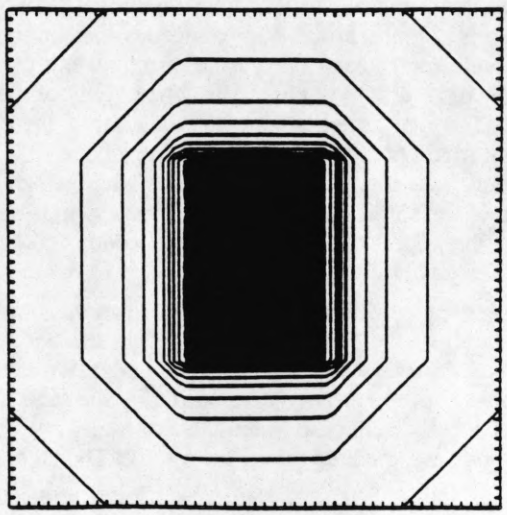


Figure A.2a

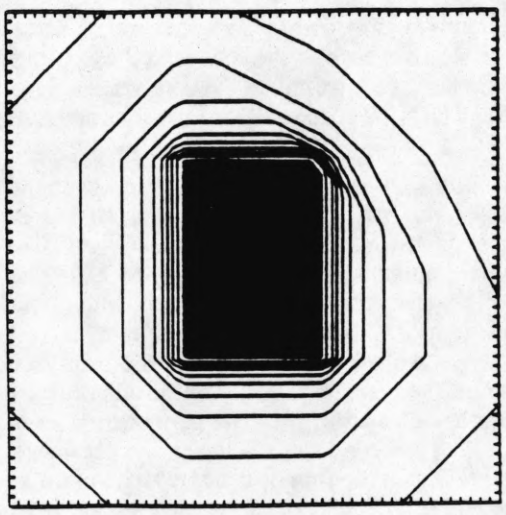


Figure A.2b

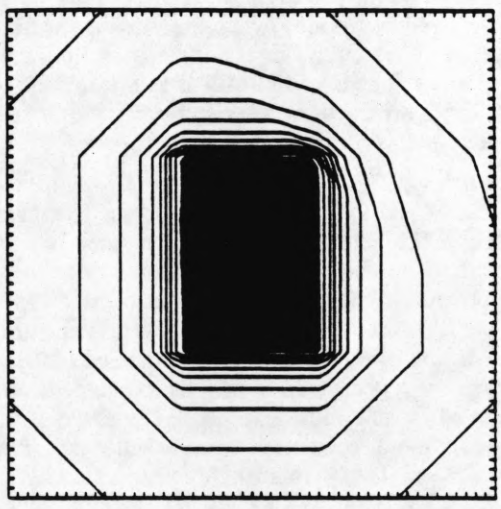


Figure A.2c

Figure A.2 Differences in potential functions of almost identical objects.
(a) Level curves of the potential function of a rectangular object.
(b) Level curves of the potential function of the same object with a chipped corner.
(c) Level curves of the modified potential function of the chipped rectangular object.

If the current node has only one son, then the son is examined to see if it can be connected to any of the nodes in the node list except its father. If so, the proper connection is made. If the son is not connected to any of the nodes in the node list, then it is put into the temporary node list and becomes the current node. The reason why the son is not added to the node list is the following. Since we invoke the connecting process after each generation of new nodes, it is computationally advantageous to keep the node list as small as possible. The nodes within a branch are never connected to each other except to their immediately adjacent nodes. When we are generating the nodes within a branch, it suffices to check the connectivity of the newly generated node with the existing nodes in the node list that is not on the same branch. It is reasonable, therefore, not to add the newly generated node to the node list, but to keep it in a separate list. The temporary node list will become longer and longer if the current node keeps on generating only one son. Such a case arises when we are generating MPV in a long channel. If more than one son is generated or the son is connected to an existing node in the node list, then the elements of the temporary node list are added to the node list. The next current node is taken from the front of the queue. This completes one cycle of the 2D MPV algorithm.

There are cases where this algorithm generates nodes in dead-end channels. Also, it is of no use to find MPV where the width of the free space is less than the width of MO. We stop generating the adjacent nodes of such nodes and mark the nodes as the dead end nodes. Another thing to note is that except for the start and goal node, adjacent nodes should not be generated in the direction from which the current node is generated. To prevent this, the points of locally minimum potentials are found only on the three quarters of the circle away from the direction the current node is generated (see Figure A.3). The algorithm to find MPV in two dimensions is summarized in Figure A.4.

A.2.2. Minimum potential valleys in three dimensions

In generating MPV in 3D, a queue and two lists are kept. The queue contains the nodes whose adjacent nodes are yet to be found. It initially contains the start and the goal node. One list called node list (NL) contains all the nodes generated. An additional node list is also needed. The nodes in the queue are called "boundary nodes," because they mark the boundary between the free space already filled with nodes and the free space yet to be filled with nodes. First, the boundary nodes are added to NL, and MD of each boundary node is computed. Then a sphere of radius MD of each node is drawn at each boundary node, and N points are uniformly placed on each of the spheres[14]. The N points on each of the spheres are called the sons of the boundary node at the center of the sphere, and the boundary node is called their father. All the sons are put in the node list called son list (SL). The next step is to remove from SL the nodes which are in the parts of the free space already filled with nodes, i.e., the free space MPV are already generated. This is done by checking the distance between each of the sons with the existing nodes in NL. If any of the sons is within MD of any one of the existing nodes except its father, the son is deleted from SL. What are left in SL are the son nodes in the free space yet to be filled with the spheres. Once the sons are generated, the queue is emptied.

Not all the sons left in SL can be used to represent MPV. The sons with large MDs are the ones far away from the obstacles and thus are good candidates to represent MPV. The sons chosen to represent MPV are called permanent sons. The selection of permanent sons is done as follows. MDs of the sons are computed. The sons with MD less than the width of MO are deleted since the nodes are impossible to be traversed by MO. The remaining sons are ordered in the descending order of MD. The first son is selected as a permanent son, and all the other sons within the distance $MD_{first\ son}$ from the first son are deleted from SL. Then the first of the remaining sons is selected, and all the other sons within the distance $MD_{selected\ son}$ from the selected son are deleted from SL. This selection process is repeated until only the permanent sons are left. The permanent sons are the adjacent nodes of their fathers in MPV, and thus appropriate connections are made in the data structure. The permanent sons located close together should also be considered as adjacent nodes in MPV. The connections between the permanent sons are made as follows. Two permanent sons separated by a distance less than the sum of their MDs are considered as adjacent nodes. The straight line connecting two such permanent sons does not cross the obstacles, but can approach an obstacle arbitrarily closely as shown in Figure A.5. An additional node is placed in the intersection of the two spheres of radius of respective MDs centered at the two permanent sons. MD of the additional node gives a good measure of how hard it is for the moving object to move between the two permanent sons. All the possible additional nodes are added to SL as permanent sons.

---

[14]Uniform quantization of a sphere can be achieved using the vertices of regular polyhedra. Let N be the number of quantizing points. If N = 4, 6, 8, 12 or 20, then the vertices of the corresponding regular polyhedron determine the locations of the quantizing points. For N = 14 or 32 the vertex locations together with the centers of faces of a cube or an icosahedron can be used. For most of the findpath problems, N = 14, 20 or 32 is sufficient.

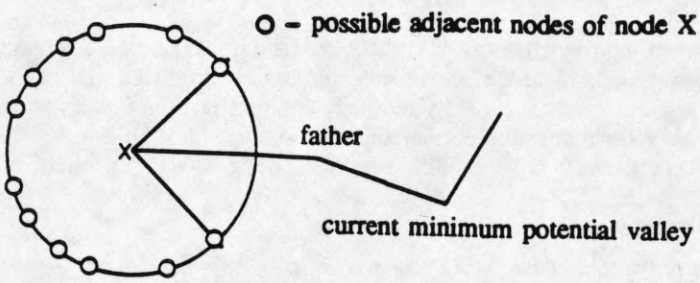


Figure A.3 Except for the start and goal nodes, adjacent nodes of a node are found away from the direction of its father.

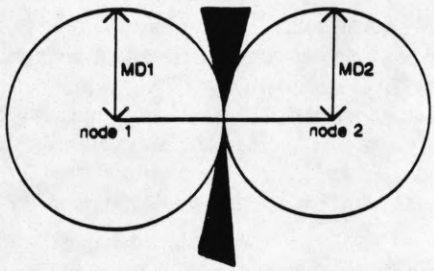


Figure A.5 An additional node is needed to represent how close the obstacles are from the line connecting two nodes.

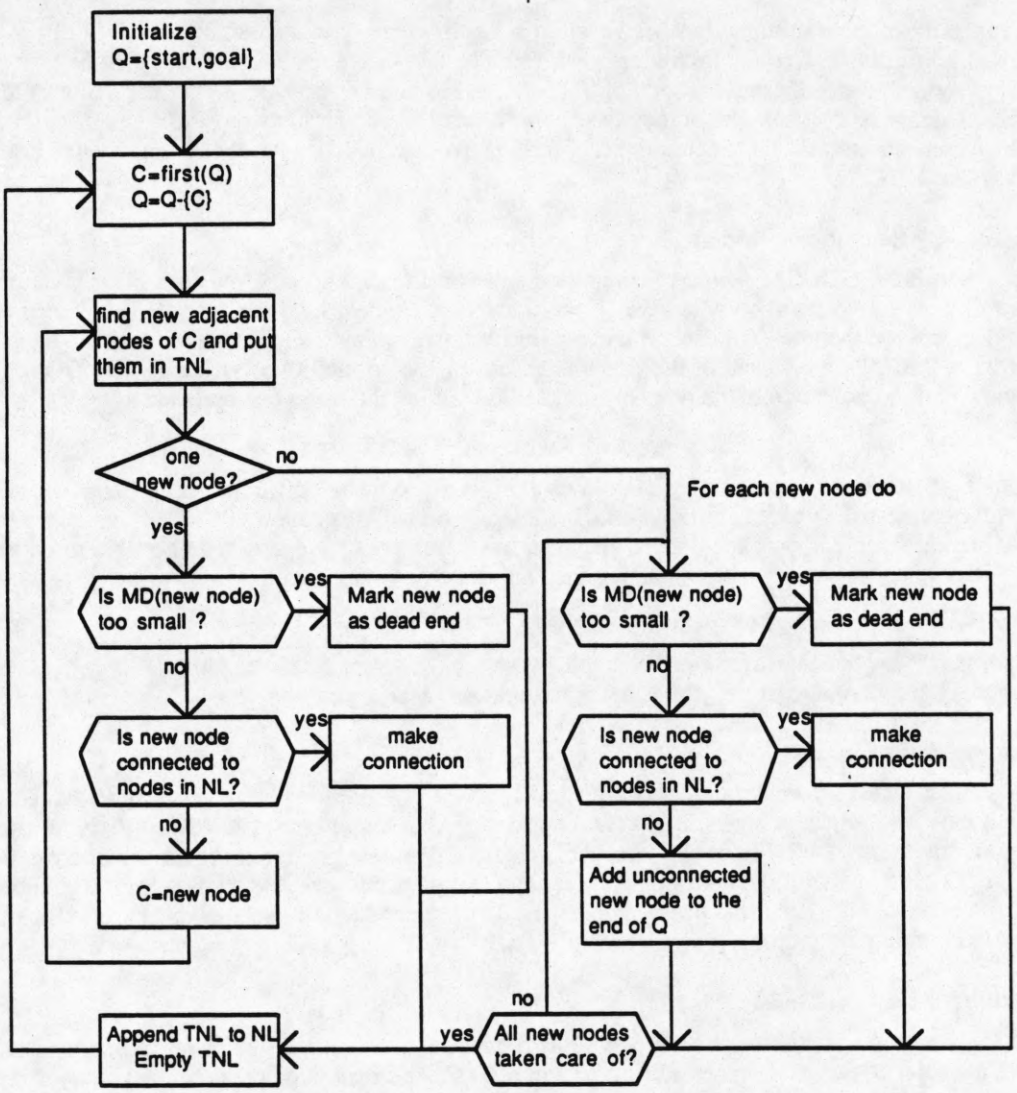


Figure A.4 Flow chart of the algorithm generating minimum potential valleys in 2D.
NL: node list, TNL: temporary node list, C: current node,
Q: the queue containing the nodes whose adjacent nodes are to be found.

Additional connections should be made between the permanent sons and the existing nodes in NL. A connection is made between a permanent son and an existing node if the distance between them is less than either of their MDs. It is sufficient to check the connections between the permanent sons and their fathers since the fathers are the boundary nodes. After all the necessary connections are made, the permanent sons are put into the queue. This completes one cycle of the 3D MPV algorithm. This process is repeated until no permanent sons are generated.

A.2.3. Minimum-Distance Algorithm in Two Dimensions

Let us assume that all obstacles are convex polygons. Then MD from a point x to an obstacle can be computed in the following way. Let $g_i(x)$ be the equations defining the edges of a polygonal obstacle, and $n_i(x)$ be the equations of the lines perpendicular to the edges passing through the vertices as shown in Figure A.6. First all $g_i(x)$'s are evaluated, which are negative on the obstacle side and positive on the other side. If all $g_i(x)$'s are nonpositive, then point x is on the obstacle and MD=0. If some of $g_i(x)$'s are positive, then $n_i(x)$'s corresponding to the positive $g_i(x)$'s are evaluated. For example, in Figure A.6 $g_1(x)>0$ and $g_2(x)>0$ and $n_i(x)$'s are evaluated from $n_2(x)$. By successively evaluating $n_i(x)$'s, the edge or vertex yielding the minimum distance from the obstacle to point x can be determined. For example, if x is in region R_1 then MD is the distance between x and edge 1. If x is in region R_{22} then MD is the distance between x and vertex 2.

The complexity of the minimum distance function is O(N) where N is the number of edges of the polygonal obstacle. This can be computed in the following way. First, N $g_i(x)$'s are evaluated. Less than half of $g_i(x)$'s are positive on the average. Since there are a total of 2N $n_i(x)$'s, the number of $n_i(x)$'s corresponding to the positive $g_i(x)$'s is on the average 1/2x2N, or N. Among the N $n_i(x)$'s corresponding to the positive $g_i(x)$'s half of them are expected to be evaluated before MD is determined. Thus the average number of equations evaluated are N $g_i(x)$'s plus N/2 $n_i(x)$'s, or 1.5N.

A.2.4. Minimum-Distance Algorithm in Three Dimensions

Gilbert's algorithm [GIJO85] is used to compute the distance from a point x to a convex polyhedron. A brief explanation of Gilbert's algorithm is given here. First, the vertex v* closest to x is found. This vertex yields MD only if the angles between the line segment xv* and the line segments joining v* to other vertices are all greater than or equal to $\pi/2$. If this fails, then each of the edges is tested for the possibility of yielding MD. Given two of the vertices, v_1 and v_2, of the polyhedron, the points, y, on the line joining them can be expressed as

$$y = \lambda_1 v_1 + \lambda_2 v_2, \quad \lambda_1 + \lambda_2 = 1.$$

The closest point, y*, on the line to x can be found by solving a set of linear equations. If λ_1* and λ_2* corresponding to y* are all nonnegative, y* lies on the line segment $v_1 v_2$ and MD is given by y*. If all the edges fail to give MD, a similar step is taken for each face of the polyhedron. Three points are selected at a time to make up a triangular face. The points, y, on the plane containing the face can be expressed as

$$y = \lambda_1 v_1 + \lambda_2 v_2 + \lambda_3 v_3, \quad \lambda_1 + \lambda_2 + \lambda_3 = 1.$$

The closest point, y*, on the line to x can be found by solving a set of linear equations. If λ_1*, λ_2* and λ_3* corresponding to y* are all nonnegative, y* lies in the triangle and MD is given by y*.

A.2.5. Dynamic programming

In 2D there are relatively small numbers of junctions (nodes adjacent to more than two other nodes). A string of nodes which have two adjacent nodes in 2D can be grouped as a single branch to reduce the time taken by the dynamic programming algorithm. Figure A.7 shows the result of dynamic programming for the findpath problem in Figure 2.5a. In 3D MPV, most of the nodes have more than two adjacent nodes. The overhead of grouping strings of nodes to form branches is greater than savings resulting from the reduction in the number of nodes. Therefore, the dynamic programming algorithm is run on the original MPV in 3D.

A.2.6. Smoothing the selected path

The above algorithms approximate MPV with polygonal lines consisting of the fewest possible vertices as the adjacent nodes are found on a circle (sphere) of maximum radius. As a result, the paths contain many sharp corners, and the distances between adjacent nodes are often too large to be used by the algorithms that generate optimal paths. A solution to the first problem is lowpass filtering of the selected optimal path. A simple second order lowpass filter is used. Let $x(i)$ i=0,...,n be the sequence of the position vectors along the selected path. Then $y(i)$ i=0,...,n given by

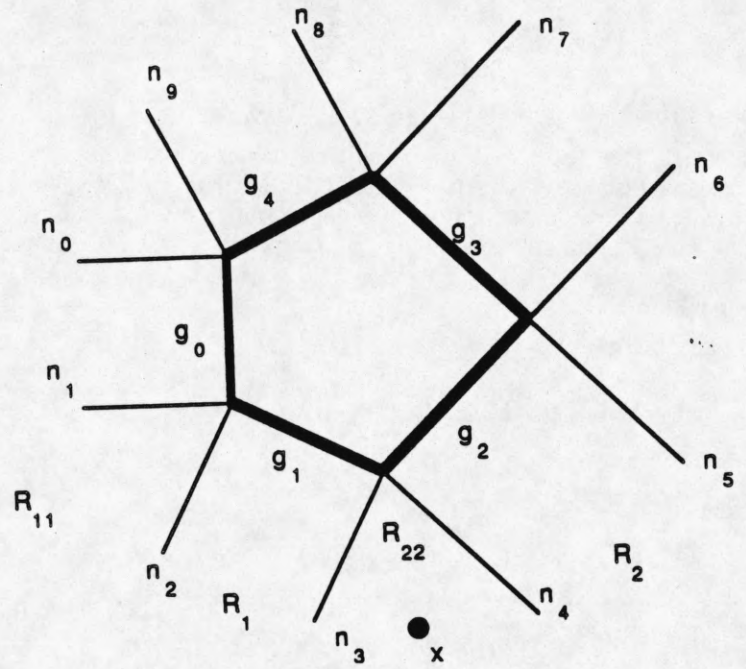


Figure A.6 Computing the minimum distance from a point X to a polygon.

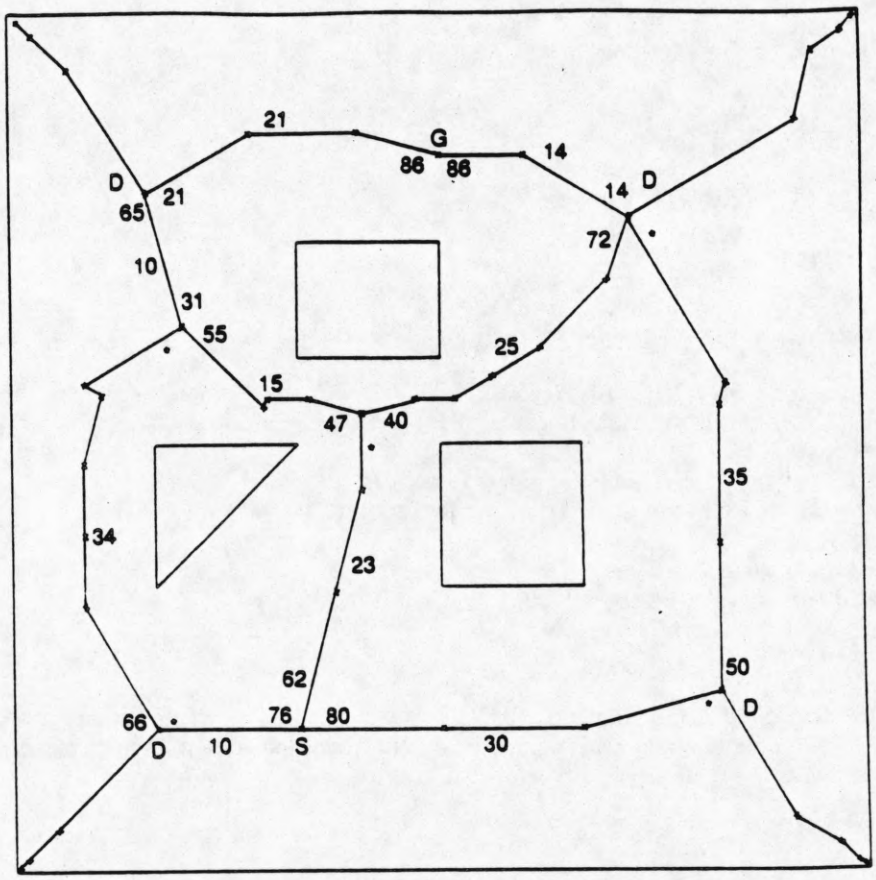


Figure A.7 Result of dynamic programming. The numbers in the middle of a branch is the cost of the branch, and the number at the end of a branch near a junction is the cost from the junction vertex to go to the goal via the branch. * denotes a branch leading back to the start node, and D represents a branch leading to a dead end. Both * and D are considered to have infinite costs.

$$y(i) = (x(i-1) + 2x(i) + x(i+1)) / 4, \quad y(0)=x(0), \quad i=1,...,n-1$$

yields a path without sharp corners. The distances between adjacent nodes are still large, and more nodes are created between the nodes to make the distances between adjacent nodes less than a preset value. The resulting path is lowpass filtered once more to yield a smooth path with high enough resolution for the findpath algorithms. Figures A.8a through d show various phases of the path-smoothing and interpolation for a 2D example.

A.3. Parallel Optimization Algorithm

The objective functional to be minimized is

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

and the Hamiltonian of the system is

$$H = [(1+bP(x,\theta))(v(t))^2 + a(\omega(t))^2] + y_1 v(t) + y_2 \omega(t). \tag{A.3.2}$$

The necessary conditions for the optimality are

$$\nabla_y H = 0 : \quad \frac{dx}{dt} = v(t), \quad x(t_0) = x_0, \; x(t_f) = x_f \tag{A.3.3a}$$

$$\frac{d\theta}{dt} = \omega(t), \quad \theta(t_0) = \theta_0, \; \theta(t_f) = \theta_f$$

$$\frac{dy}{dt} = -\nabla_{x,\theta} H \; : \; \frac{dy_1}{dt} = -\frac{dP}{dx} v(t)^2 \tag{A.3.3b}$$

$$\frac{dy_2}{dt} = -\frac{dP}{d\theta} v(t)^2$$

$$\nabla_{v,\omega} H = 0 : \quad 2(1+bP(x,\theta))v(t) + y_1 = 0 \tag{A.3.3c}$$

$$2a\omega(t) + y_2 = 0.$$

The steps of the gradient algorithm are as follows.

1. Guess initial solutions $v^0(t)$, $\omega^0(t)$. Set i=0, $J^0 = \infty$.
2. Integrate (A.3.3a) forward in time to get $x^i(t)$, $\theta^i(t)$. Compute J^i.
   If $J^i > J^{i-1}$, set $\varepsilon=\varepsilon/2$,
   set $v^{i+1}(t) = v^i(t) + \varepsilon \nabla_v H^i$, $\omega^{i+1}(t) = \omega^i(t) + \varepsilon \nabla_\omega H^i$, go to step 2.
3. Integrate (A.3.3b) backward in time from $y_f=0$ to get $y^i(t)$.
4. Compute $\nabla_{v,\omega} H^i$
   If $|\nabla_{v,\omega} H^i| < \delta$, go to step 5.
   If $|\nabla_{v,\omega} H^i| > \delta$, set $v^{i+1}(t)=v^i(t)-\varepsilon\nabla_v H^i$, $\omega^{i+1}(t)=\omega^i(t)-\varepsilon\nabla_\omega H^i$, i=i+1.
   Go to step 2.
5. Done.

Note that the computation of dP/dx in Equation (A.3.3b) has to be carried out numerically. Since P, the total potential experienced by the moving object, is computed by numerical integration, numerical differentiation is the only way to calculate dP/dx.

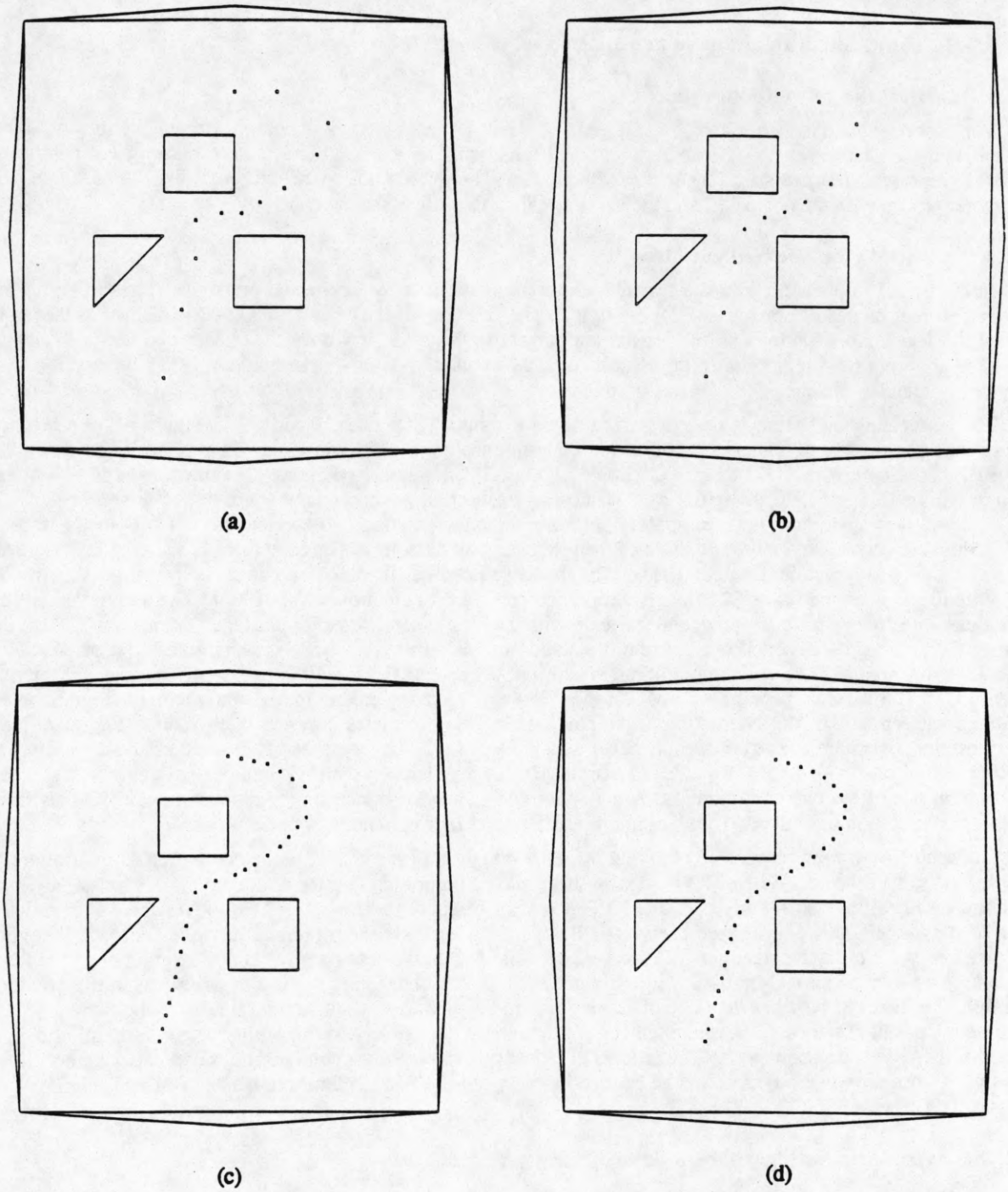


Figure A.8 Interpolation and smoothing of the initial candidate path.
(a) The minimum cost path chosen from the minimum potential valleys.
(b) The path after lowpass filtering.
(c) The path after interpolation.
(d) The path obtained after lowpass filtering the interpolated path three times.

A.4. Serial Optimization Algorithm

A.4.1. Serial optimization algorithm in two dimensions

A.4.1.1. Identification of collision regions

The best candidate path is given as a sequence of nodes. At each node it is determined whether or not MO is colliding with some obstacles. The nodes with collisions occur in groups. The middle node of each group is selected to represent the corresponding region of collision. The middle nodes are located at the places where MO has a large overlap with the obstacles, and they will be called the collision centers.

A.4.1.2. Connecting feasible configurations

In the actual implementation, the two feasible configurations to be connected are selected according to the similarity of their angles of deviation (AD) rather than the similarity of the orientations. AD is defined as the angle between the feasible orientation and the orientation tangent to the path. If we try to connect two feasible orientations of neighboring hard regions with the closest ADs, the resulting path and orientation of MO will have the least amount of orientation change.

The connecting algorithm is best explained by an example. Suppose C1 and C2 in Figure A.9 are the two feasible configurations to be connected. Note that the reference point of MO in C1 or C2 may not lie on the initial path. MO is first moved from C2 toward C1, since MO in C2 is at a lower potential. The movement of MO is restricted to within 45 degrees of the initial path. If no such restriction is imposed, MO will not be driven toward C1. Instead, it will wander around the free space just to minimize the potential. The orientation of MO in C1 is fixed and SOA tries to determine the next position of the reference point among the three nodes S1, S2 and S3. S2 is generated by moving MO parallel to the initial path by the same length as the internode distance in the initial path. S1 and S3 are the adjacent nodes of S2 in the direction perpendicular to the movement of MO. The total potentials on MO at these three nodes are computed, and the node with the least potential is chosen as the next node of C1. In the example, S2 will be the next node. Next, the orientation of MO at the new node is determined. Three candidate orientations (or more if more computation time is allowed) are selected, the middle one being the same orientation as that in C1. The the total potentials on MO at these three orientations are computed, and the one with the lowest potential is chosen as the MO orientation at the new node. This completes one step of the connecting algorithm. The connecting algorithm next tries to connect the new node and C2. This process of advancing one step and relaxing the potential continues until C1 and C2 are connected. It may happen that MO collides with an obstacle while moving from one of the two configurations being connected. Then the connecting algorithm moves MO only from the other configuration until the two configurations are connected, or until another collision occurs.

Connecting two configurations by moving MO in a straight line and changing the orientation at a uniform rate is implemented as follows. When MO is moving from the configuration (x_1, θ_2) to the configuration (x_2, θ_2), the maximum distance traveled by any part of MO is less than or equal to $|x_1 - x_2| + l\,|\theta_1 - \theta_2|$ where l is half the length of the longest axis of MO (see Figure A.10). If this is less than the minimum distance from MO in either configuration to the obstacles, then no collisions occur while MO moves between the two configurations. This fact can be used to join two configurations via a straight line. In the connecting algorithm described in the previous paragraph, it is checked whether the two configurations can be connected via a straight line once the two nodes are within some preset distance. The intermediate configurations are generated in a binary tree fashion, and it is checked if they cause any collisions. The process of generating intermediate configurations and checking collisions is continued until a collision occurs, or the connection between the two configurations is established. Figure A.11 illustrates this process.

A.4.2. Serial optimization algorithm in three dimensions

A.4.2.1. Identification of collision regions

The initial orientations of MO along the candidate path are assigned such that the major axis is tangent to the path, and the second axis is in the direction of minimum potential. If this method of placing the second axis is carried out independently at each point on the path, the change in the direction of the second axis (or the variation in ω) can be large as MO travels from a point to its adjacent point. For this reason the following procedure is used. We denote the major, second and minor axes with vz,vx and vy, respectively. Let the current node be the node following the start node on the path. The angle ω of the current node is temporarily set to ω of the previous node. Then the second axis is rotated about the already determined vz of the current node until the potential on the second axis reaches the first local minimum. The direction of vx is then set parallel to the second axis, and ω of the current node

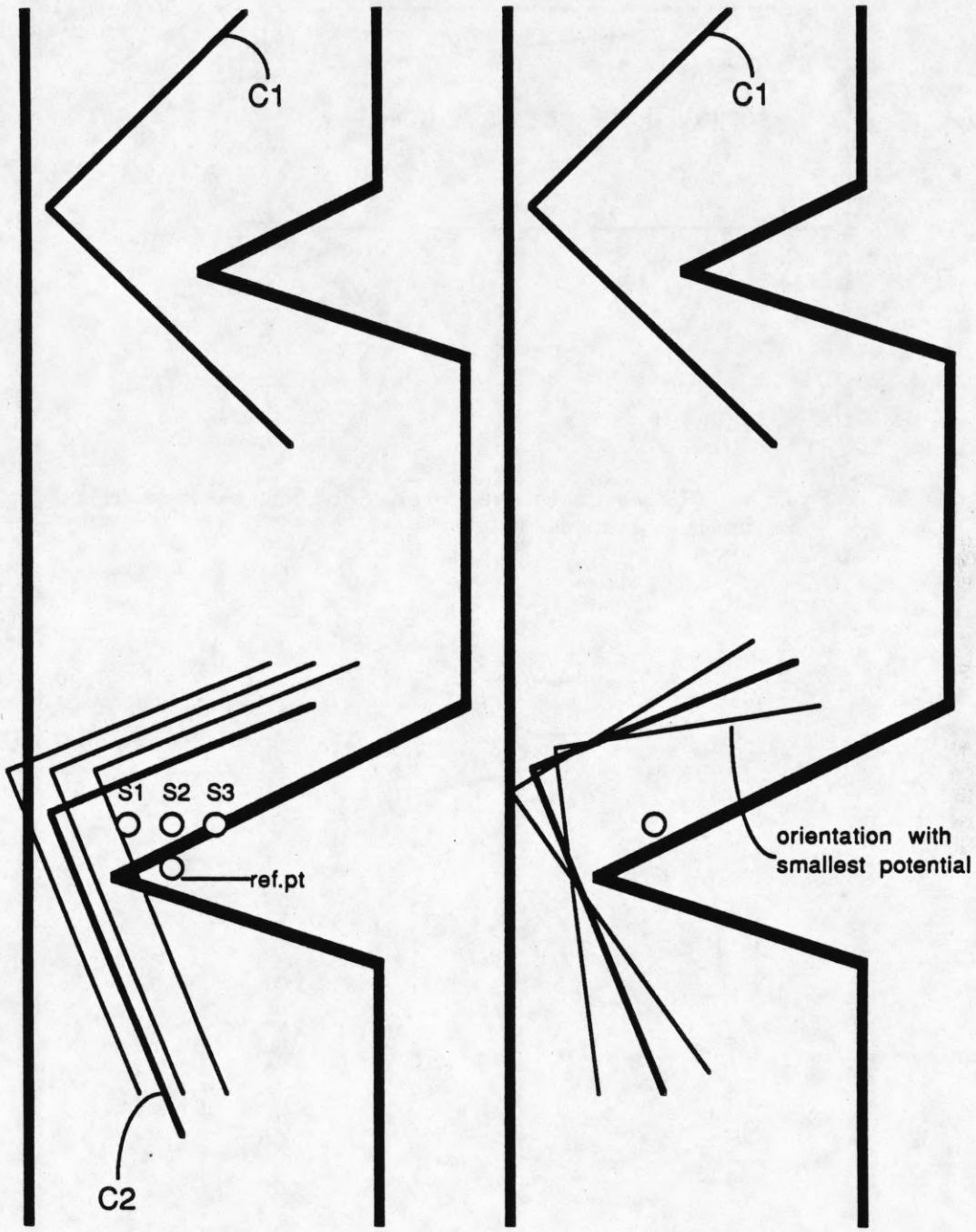


Figure A.9 The algorithm connecting two configurations C1 and C2. The new position of the reference point is determined first, and the orientation at the new position is selected to further reduce the potential.

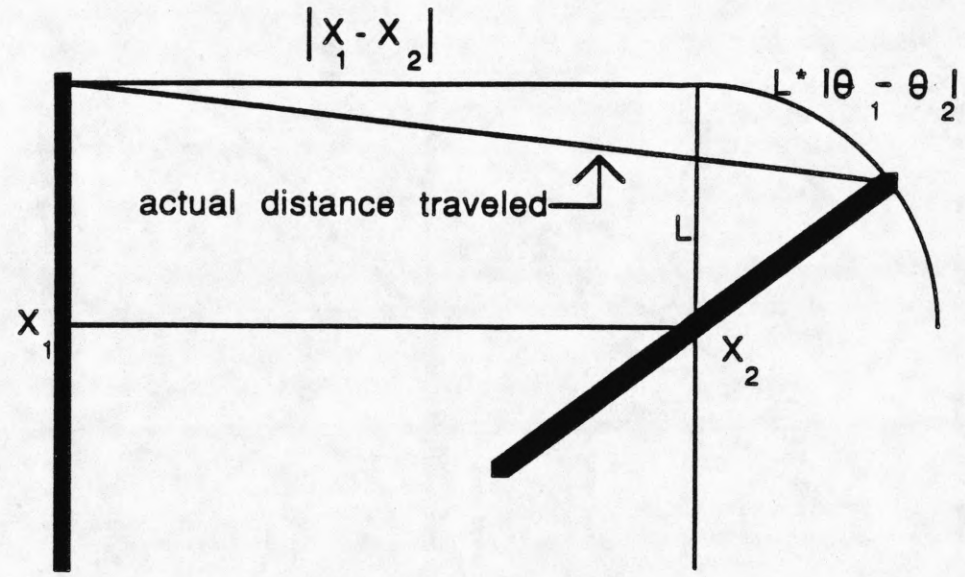


Figure A.10 Maximum distance traveled by any part of the moving object is less than $|x_1 - x_2| + l^* |\theta_1 - \theta_2|$.

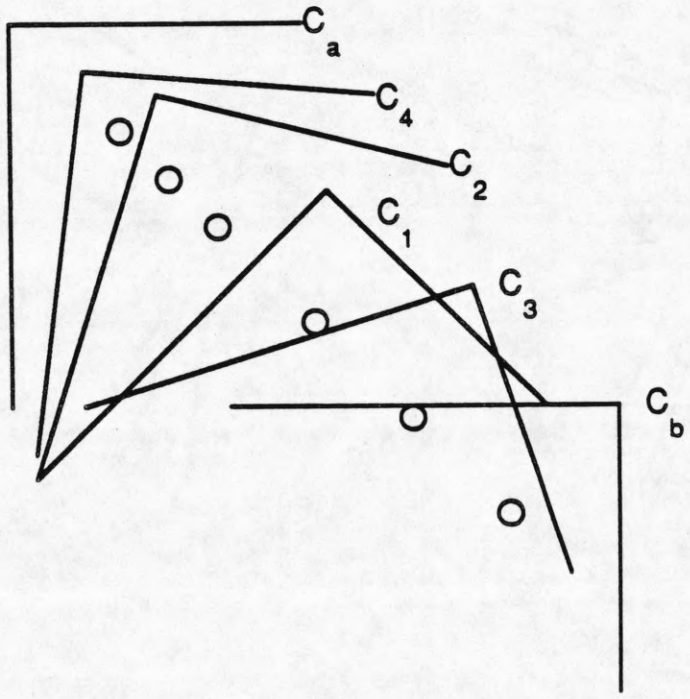


Figure A.11 Intermediate configurations generated by the algorithm for checking whether two configurations can be connected directly. Here, the intermediate configurations of C_a and C_b are generated in the order of the subscripts. Tests for collisions are performed at these intermediate configurations.

can be computed. Next, the node following the current node becomes the current node. This procedure is repeated until the current node is the node right before the goal node. If the orientation of the goal node is not specified, then the same procedure is applied to the goal node. This completes the initial assignment of the MO along the initially chosen path. Given the initial path and orientations along it, the collision regions and their collision centers are found as in 2D.

A.4.2.2. Feasible configurations in collision centers

The problem of uniformly quantizing the three-dimensional orientation space could be considered as the problem of uniformly placing a given number of points on a sphere in four dimensions, but a different approach is used in this thesis. The space of vector vz, which determines Euler angles ϕ and θ, can be considered as a sphere. Thus, uniformly quantizing the $\{\phi, \theta\}$ space into N points is equivalent to placing N points on the sphere, making them separated from one another as much as possible. Such a property is found in the vertices of regular polyhedra, and the quantization procedure is discussed in Section 2.4.2. After all the directions of vz are determined for a given number of N, the vz of MO is placed in each of these directions. Then ω is incremented successively by a small preset amount, UA, and the potential on MO is computed. The values of ω with locally minimum potential and without collisions are stored with the corresponding vz as possible feasible orientations. This is done for all the directions in the quantized $\{\phi, \theta\}$ space[15].

Once the possible feasible orientations are found, the reference point of MO is allowed to move in the direction perpendicular to the path to further lower the potential. A circle of radius UL centered at the collision center is drawn in the plane perpendicular to the path. The reference point of MO is moved on the circle at angular interval UA1, and the total potential on MO at each point is computed. If the potential on MO is the smallest at the original position, the reference point of MO stays at the collision center. Otherwise, the reference point of MO is moved in the direction toward the point on the circle yielding the smallest potential. The reference point of MO is moved in this direction at an interval of UL as long as the potential on MO becomes lower. The reference point of MO, however, should not be allowed to move outside of the vicinity of the collision region. If not, the reference point of MO may run off to infinity in some cases. In the actual implementation, the reference point of MO is allowed to take NSTEP steps of length UL. After the position of the reference point of MO is obtained for each of the possible feasible orientations, each of the configurations is tested for collisions. Those without collisions are selected as the feasible configurations of MO in the particular collision region.

A.4.2.3. Connecting feasible configurations

When selecting the feasible configurations to be connected, the best first search is used as in 2D. Angle of deviation (AD) is introduced in 2D as a measure to select the first feasible configuration to be tried. Since the orientation space is three-dimensional in 3D, a more elaborate method has to be used to measure how close two orientations are with respect to the direction of the path. Let vt be the vector tangent to the path. The cross product, vc, of vt with vz describes the relative position of vz with respect to vt. The norm of the difference of two vc's, $| vc_1 - vc_2 |$, measures how far apart vz_1 and vz_2 are with respect to vt. Note that it is very cumbersome to use Euler angles ϕ and θ to measure the difference in vz because of their singularities at $\phi = 0$ and $\phi = \pi$. The difference in the third Euler angle $|\omega_1 - \omega_2|$ is used to measure the difference in the amount of rotation of MO about vz_1 and vz_2, respectively. A weighted sum of $| vc_1 - vc_2 |$ and $|\omega_1 - \omega_2|$ can be used to select a feasible configuration of the neighboring collision that is closest to the feasible configuration of the current collision region.

Given two configurations to be connected in neighboring collision regions and the path from MPV between them, the process of taking a step and adjusting the orientation at the new position is repeated. As in 2D the reference point of MO is allowed to move within 45 degrees of the direction of the path from MPV. The reference point of MO is moved parallel to the path from MPV by the same length as the internode distance in the path from MPV. On the plane perpendicular to the direction of the path and passing through the new position of the reference point, a circle of radius equal to the internode distance centered at the new reference point is drawn. After a number of points are placed on the circle at an angular interval of UA1, the potential on MO is computed with the reference point placed at these points on the circle and also at the center of the circle. The point with the smallest potential is chosen as the new position of the reference point of MO. The potential on MO at the new position is now relaxed by changing the orientation of MO. Euler angles are used in this process. First, ϕ is changed by UA at a time while holding θ and ω constant, and is set to the value with minimum potential. Next, θ and ω are similarly changed to minimize the potential on MO.

---

[15]This procedure of finding possible feasible orientations is computationally expensive. For example, if N=12 and the increments in ω is 15 degrees, the total potential on MO has to be computed 288 times. A more efficient algorithm utilizing the shape of MO seems possible.

Connecting two different configurations by checking the existence of collisions of MO in the intermediate configurations generated by a simple interpolation can speed up the connecting algorithm as in 2D. The interpolation of the position of the reference point is straightforward, but the interpolation of the orientation in 3D must be given careful consideration. The Euler angles cannot be used in the interpolation process because of the singularities mentioned earlier. For example, consider interpolating the orientation between two orientations O1 and O2 represented on a Gaussian sphere in Figure A.12. Interpolating the Euler angles ϕ and θ yields a very awkward and inefficient transition from O1 to O2 as shown in Figure A.12a. When the interpolation is done using the vector vz, the resulting intermediate orientations lie on a great circle of the Gaussian sphere as shown in Figure A.12b. This is the shortest way to go from one orientation to another, and the interpolation of ϕ and θ is done using vector vz. The interpolation of the third angle ω is done in a straightforward manner, since it is a scalar.

The above method of interpolating the orientation between two configurations is not always successful. Consider an example shown in Figure A.13. An L-shaped moving object is in the corner of a narrow L-shaped hallway. The two nearer walls are not drawn. The two configurations of MO to be connected are shown in Figures A.13a and b. The arrow denotes the major (longest) axis of MO. The natural and only way to make the transition is to rotate MO about one of its legs, as shown in Figure A.13c. This makes the major axis of MO to trace out a circle that is not a great circle of the Gaussian sphere (see Figure A.13d). If the major axis of MO is forced to trace a great circle, MO collides with one of the walls. This is somewhat expected since the simple interpolation of the configuration does not take into account the shapes of MO and the free space. A key to solving this problem is to use the potential function information during the interpolation process. Suppose that the intermediate orientations are obtained on a great circle. Instead of the major axis tracing through these intermediate orientations, the major axis is allowed to select its next orientation from a range of orientations as illustrated in Figure A.13e. The orientations within 45 degrees of the great circle are examined at each step, and the orientation with the smallest potential is selected as the next orientation of the major axis. The third Euler angle ω is also adjusted to minimize the potential on MO. This procedure is repeated until the two configurations are connected. If a collision occurs before they are connected, then it is assumed that the two configurations are not possible to be connected. This modification to the the connecting algorithm allows a wide range of orientations to be examined, and most of the configurations that can be connected are successfully connected by the modified algorithm.

A.5. Sidetracking Algorithm

A.5.1. Places for sidetracking

As already explained, MO can sidetrack only from the junction regions, and feasible configurations have to be found in these regions in addition to the collision regions. If, however, a junction node lies in free space wide enough to allow MO to rotate by 360 degrees, then an excursion is not needed to change the orientation of MO. Thus, feasible configurations at such a junction need not be found. It may happen that a junction node lies in one of the collision regions. Then the collision region is cut into two separate collision regions by the junction node, and the feasible configurations are found at the junction node and in the two collision regions.

A.5.2. Selection of the feasible configurations to be connected by STA

When using NFNB nodes as intermediate nodes in trying to connect a backward node to a forward node, all possible permutations using none through all of NFNB nodes are generated in order not to miss any sequence of NFNB nodes that may yield a solution. Figure 5.2b shows all the permutations of NFNB nodes in one of the junction regions. If there happens to be another backward node in the junction region, it is completely ignored since the existing backward node has already failed to be connected to one of the forward nodes. If certain nodes are found to be connected, or are not connectable, this information is recorded. Storing this information in a two-dimensional array results in a huge saving in computation time since the node connectivity information is used repeatedly.

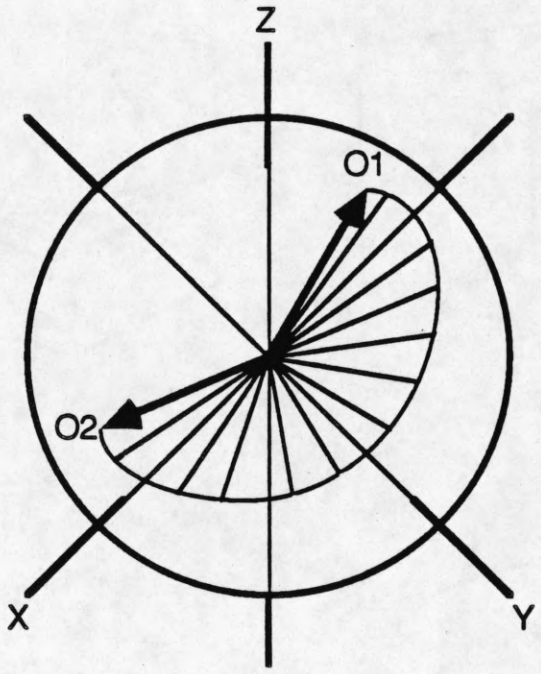


Figure A.12a Interpolation of orientation using Euler angles.

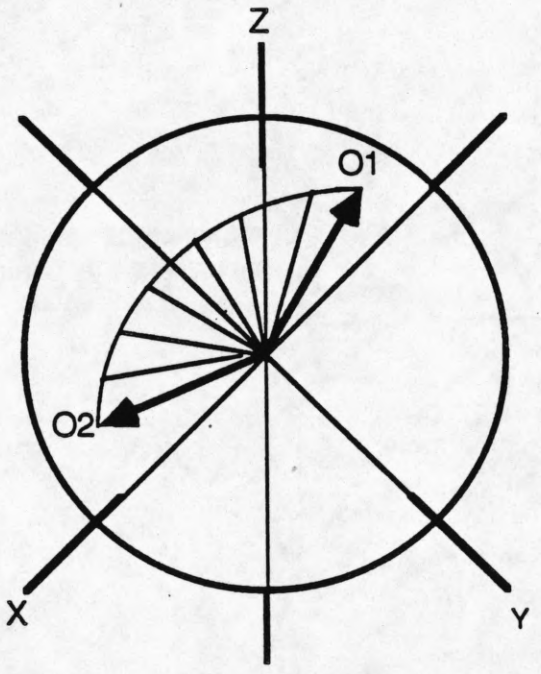


Figure A.12b Interpolation of orientation using vectors.

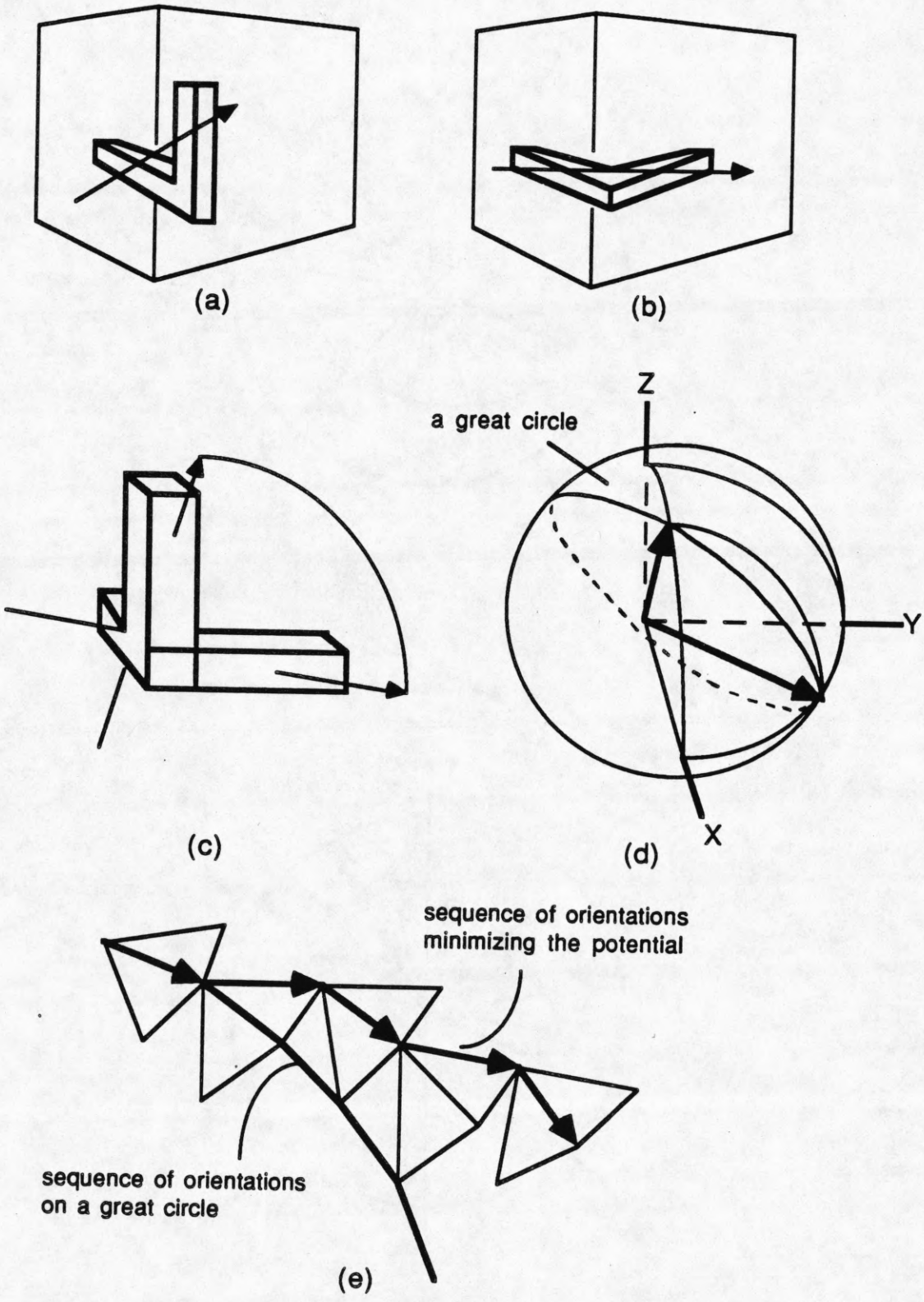


Figure A.13 A situation illustrating that a straightforward interpolation of the orientation vector does not always give good intermediate orientations.

REFERENCES

[ATFA66]
Michael Athens and Peter L. Falbs, *Optimal Control*. New York: McGraw Hill, 1966.

[BROO83]
Rodney A. Brooks, "Solving the Findpath Problem by Good Representation of Free Space," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-13, pp. 190-197, March/April 1983.

[BRLP83]
Rodney A. Brooks and Tomás Lozano-Pérez, "A Subdivision Algorithm in Configuration Space for Findpath with Rotation," *International Joint Conference on Artificial Intelligence*, Karlsruhe, Germany, 1983.

[BRHO75]
Arthur E. Bryson and Yu-Chi Ho, *Applied Optimal Control*. Washington: Hemisphere Publishing Corporation, 1975.

[CHAT82]
Raja Chatila, "Path Planning and Environment Learning in a Mobile Robot System," *Proceedings of the European Conference on Artificial Intelligence*, Orsay, France, July 1982.

[CHLA85]
Raja Chatila and Jean-Paul Laumond, "Position Referencing and Consistent World Modeling for Mobile Robots," *Proceedings of IEEE International Conference on Robotics and Automation*, St. Louis, Missouri, March 1985.

[CHAT85]
R. Chattergy, "Some Heuristics for the Navigation of a Robot," *The International Journal of Robotics Research*, vol. 4, no. 1, Spring 1985.

[CHVI87]
Y. C. Chen and M. Vidyasagar, "Optimal Trajectory Planning for Planar n-Link Revolute Manipulators in the Presence of Obstacles," Technical Report, University of Waterloo, Waterloo, Ontario, Canada, 1987.

[CHEW85]
L. Paul Chew, "Planning the Shortest Path For a Disc in $O(n^2 \log n)$ Time," *Proceedings of the ACM Symposium on Computational Geometry*, Baltimore, Maryland, 1985.

[COMB68]
Paul G. Comba, "A Procedure for Detecting Intersections of Three-Dimensional Objects," *Journal of the Association for Computing Machinery*, vol. 15, no. 3, pp. 354-366, July 1968.

[DONA84]
Bruce Donald, "Motion Planning with Six Degrees of Freedom," Massachusetts Institute of Technology Artificial Intelligence Laboratory, AI-TR-791, 1984.

[DRYS79]
R. L. Drysdale, "Generalized Voronoi Diagrams and Geometric Searching," Stanford, CS Report STAN-CS-79-705, Stanford, California, January 1979.

[DYMC70]
Peter Dyer and Stephen R. McReynolds, *The Computation and Theory of Optimal Control*. New York: Academic Press, 1970.

[ERLP86]
Michael Erdmann and Tomás Lozano-Pérez, "On Multiple Moving Objects," *Proceedings of IEEE International Conference on Robotics and Automation*, San Francisco, California, April 1986.

[GIJO85]
Elmer G. Gilbert and Daniel W. Johnson, "Distance Functions and Their Applications to Path Planning in the Presence of Obstacles," *IEEE Transactions on Robotics and Automation*, vol. RA-1, pp. 21-30, March 1985.

[HERM86]
Martin Herman, "Fast, Three-Dimensional, Collision-Free Motion Planning," *Proceedings of IEEE International Conference on Robotics and Automation*, San Francisco, California, April 1986.

[KESH85]
K. Kedem and M. Sharir, "An Efficient Algorithm for Planning Collision-Free Motion of a Convex Polygonal Object in 2-dimensional Space Amidst Polygonal Obstacles," *Proc of the ACM Symposium on Computational Geometry*, Baltimore, Maryland, 1985.

[KHAT85]
O. Khatib, "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Proceedings of IEEE International Conference on Robotics and Automation*, St. Louis, Missouri, March 1985.

[KOCH85]
E. Koch, C. Yen, G. Hillel, A. Meystel and C. Isik, "Simulation of Path Planning for a System with Vision and Map Updating," *Proceedings of IEEE International Conference on Robotics and Automation*, St. Louis, Missouri, March 1985.

[KZBR85]
D. T. Kuan, J. C. Zamiska, and R. A. Brooks, "Natural Decomposition of Free Space for Path Planning," *Proceedings of IEEE International Conference on Robotics and Automation*, St. Louis, Missouri, March 1985.

[LOPE87]
Thoms Lozano-Perez, "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 3, pp. 224-238, June 1987.

[LPWE79]
Tomás Lozano-Pérez and Michael A. Wesley, "An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles," *Communications of the ACM*, vol. 22, no. 10, pp.560-570, October 1979.

[LUME87]
V. J. Lumelsky, "Effect of Kinematics on Motion Planning for Planar Robot Arms Moving Amidst Unknown Obstacles," *IEEE Journal of Robotics and Automation*, vol. RA-3, no. 3, pp. 207-223, June 1987.

[MADD86]
Sanjeev R. Maddila, "Decomposition Algorithm for Moving a Ladder Among Rectangular Obstacles," *Proceedings of IEEE International Conference on Robotics and Automation*, San Francisco, California, April 1986.

[MINI78]
Edward Minieka, *Optimization Algorithms for Networks and Graphs Vol 1*,. Marcel Dekker, Inc., 1978.

[NGUY84]
Van-Duc Nguyen, "The Findpath Problem in the Plane," Massachusetts Institute of Technology Artificial Intelligence Laboratory, AI Memo 760, 1984.

[OORE86]
B. John Oommen and Irwin Reichstein, "On Translating Ellipses Amidst Elliptic Obstacles," *Proceedings of IEEE International Conference on Robotics and Automation*, San Francisco, California, April 1986.

[RIPA81]
Richard P. Paul, *Robot Manipulators*. Massachusetts: MIT Press, 1981.

[RLWU85]
P. J. de Rezende, D. T. Lee, and Y. F. Wu, "Rectilinear Shortest Paths with Rectangular Barriers," *Proceedings of the ACM Symposium on Computational Geometry*, Baltimore, Maryland, 1985.

[SCSH83a]
Jacob T. Schwartz and Micha Sharir, "On the Piano Movers' Problem: I. The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers," *Communications on Pure and Applied Mathematics*, vol. 34, pp. 345-398, 1983.

[SCSH81]
Jacob T. Schwartz and Micha Sharir, "On the Piano Movers' Problem: II. Techniques for Computing Topological Properties of Real Algebraic Manifolds," Courant Institute of Mathematical Sciences, Report No. 39, 1981.

[SCSH83b]
Jacob T. Schwartz and Micha Sharir, "On the Piano Movers' Problem: III. Coordinating the Motion of Several Independent Bodies Amidst Polygonal Barriers," *International Journal of Robotics Research*, vol. 2, no. 3, pp. 46-75, 1983.

[SIWA86]
Sanjiv Singh and Meghanad D. Wagh, "Robot Path Planning Using Intersecting Convex Shapes," *Proceedings of IEEE International Conference on Robotics and Automation*, San Francisco, California, April 1986.

[THOR84]
Charles E. Thrope, "Path Relaxation: Path Planning for a Mobile Robot," *Proceedings of AAAI*, Austin, Texas, 1984.