# A Search Algorithm for Motion Planning with Six Degrees of Freedom*

**Bruce R. Donald**
*Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.*

Recommended by H.H. Nagel and Daniel G. Bobrow

ABSTRACT

*The motion planning problem is of central importance to the fields of robotics, spatial planning, and automated design. In robotics we are interested in the automatic synthesis of robot motions, given high-level specifications of tasks and geometric models of the robot and obstacles. The "Movers'" problem is to find a continuous, collision-free path for a moving object through an environment containing obstacles. We present an implemented algorithm for the classical formulation of the three-dimensional Movers' problem: Given an arbitrary rigid polyhedral moving object P with three translational and three rotational degrees of freedom, find a continuous, collision-free path taking P from some initial configuration to a desired goal configuration.*

*This paper describes an implementation of a complete algorithm (at a given resolution) for the full six degree of freedom Movers' problem. The algorithm transforms the six degree of freedom planning problem into a point navigation problem in a six-dimensional configuration space (called C-space). The C-space obstacles, which characterize the physically unachievable configurations, are directly represented by six-dimensional manifolds whose boundaries are five-dimensional C-surfaces. By characterizing these surfaces and their intersections, collision-free paths may be found by the closure of three operators which (i) slide along five-dimensional level C-surfaces parallel to C-space obstacles; (ii) slide along one- to four-dimensional intersections of level C-surfaces; and (iii) jump between six-dimensional obstacles. These operators are employed by a best-first search algorithm in C-space. We will discuss theoretical properties of the algorithm, including completeness (at a resolution). This paper describes the heuristic search, with particular emphasis on the heuristic strategies that evaluate local geometric information. At the heart of this paper lie the design and implementation of these strategies for planning paths along level C-surfaces and their intersection manifolds, and for reasoning about motions with three degrees of rotational freedom. The problems*

* This report describes research done at the Artificial Intelligence laboratory at the Massachusetts Institute of Technology. Support for the Laboratory's AI research is provided in part by the System Development Foundation, in part by the Office of Naval Research under Office of Naval Research contract N00014-81-K-0494, and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80-C-0505 and N00014-82-K-0334. The author is supported in part by a NASA fellowship administered by the Jet Propulsion Laboratory.

*Artificial Intelligence* **31** (1987) 295–353
0004-3702/87/$3.50 

*of controlling the interaction of these strategies, and of integrating diverse local experts for geometric reasoning provide an interesting application of search to a difficult domain with significant practical implications. The representations and algorithms we develop impact many geometric planning problems, and extend to Cartesian manipulators with six degrees of freedom.*

## 1. Introduction

The motion planning problem is of central importance to the fields of robotics, spatial planning, and automated design. In robotics we are interested in the automatic synthesis of robot motions, given high-level specifications of tasks and geometric models of the robot and obstacles. The problem is to find a continuous, collision-free path for a moving object through an environment containing obstacles; hence it has also been called the *Find-Path* or *Piano Movers'* problem.

We will confine ourselves to the *classical*[1] formulation of the Movers' problem: Given an arbitrary rigid polyhedral moving object $P$, find a continuous, collision-free path taking $P$ from some initial configuration to a desired goal configuration. We are particularly interested in the three-dimensional Movers' problem, for an object with three translational and three rotational degrees of freedom. This paper describes an implementation of a complete algorithm (at a given resolution) for the full six degree of freedom Movers' problem.

Both the moving object and each obstacle are modeled as the finite, rigid, possibly overlapping union of convex three-dimensional polyhedra. Note that the union need not be connected nor convex. The algorithm transforms the six degree of freedom planning problem into a point navigation problem in a six-dimensional configuration space (called *C-space*). The C-space obstacles, which characterize the physically unachievable configurations, are directly represented by six-dimensional manifolds whose boundaries are five-dimensional *C-surfaces*. By characterizing these surfaces and their intersections, collision-free paths may be found by the closure of three operators, called point navigation operators. A point navigation operator

(i) slides along five-dimensional level C-surfaces parallel to C-space obstacles;

(ii) slides along one- to four-dimensional intersections of level C-surfaces; or

(iii) jumps between six-dimensional obstacles.

(See Fig. 1.) These point navigation operators form the basis of a best-first search employing a set of heuristic strategies that evaluate local geometric information. Each strategy plans motions that are realized using the point navigation operators. At the heart of this paper lie the design and implementa-

[1] A terminology due to [37].

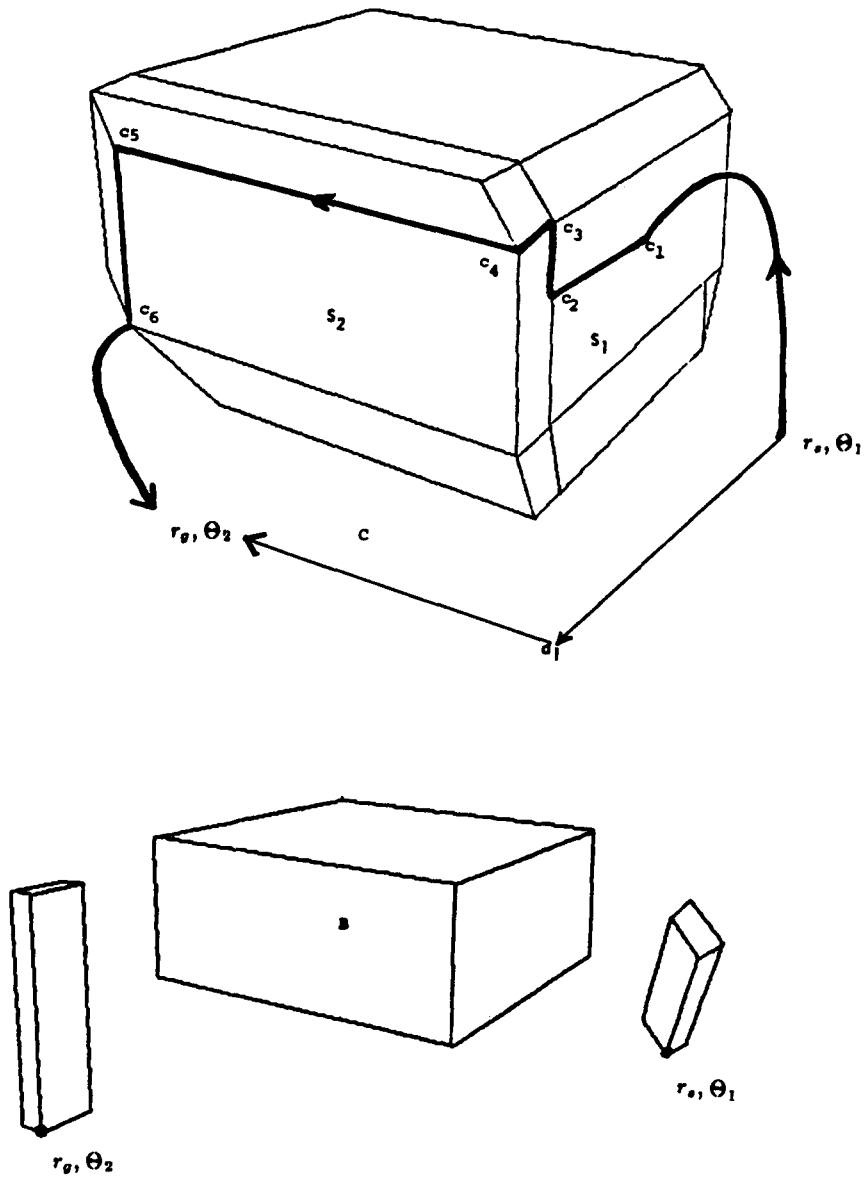


FIG. 1. We can represent the configuration of a polyhedron $A$ by a pair, $(T, \Theta)$, where $T$ is a translation of $A$ and $\Theta$ is a rotation of $A$. The problem of moving $A$ from configuration $(r_s, \Theta_1)$ to $(r_g, \Theta_2)$ is transformed to the problem of navigating a *configuration point*, $r$, past $C$, which is the C-space obstacle due to $B$. $S_1$ and $S_2$ are C-surfaces bounding $C$. The configurations $c_i$ lie on the boundary of $C$, while $d_1$ is in free space. Two trajectories around $B$ are shown. Note that the path segments $(c_6,(r_g, \Theta_2))$ and $(d_1,(r_g, \Theta_2))$ must also include a rotation. (The actual reference point is at the centroid of $A$, but for the purposes of exposition, we have placed it at a vertex as shown.)

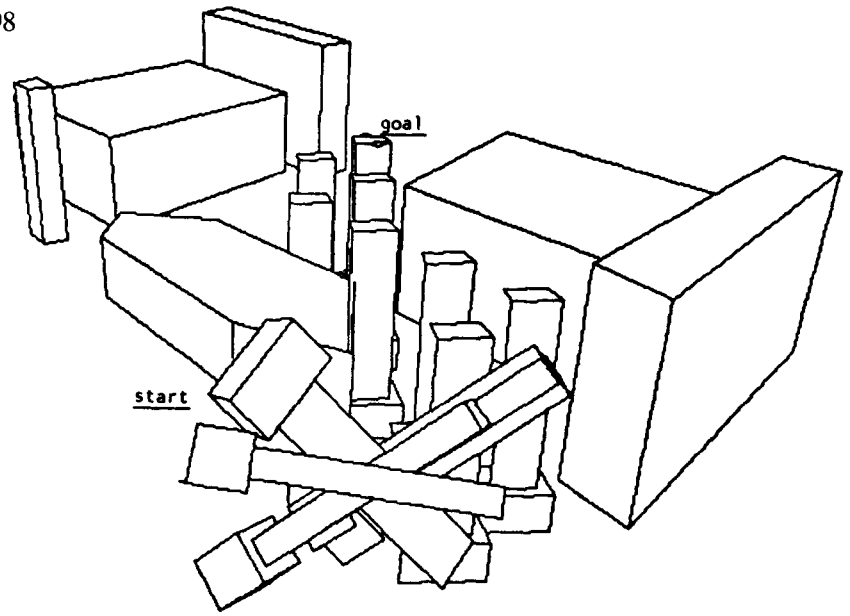


FIG. 2. An example of a solution path for the classical Movers' problem with six degrees of freedom. This illustration is a "time-lapse" picture of a path found by our planner for a hammer-shaped object. In all our examples, the workspace is bounded by a box (which is not shown). This solution path requires use of all three rotational degrees of freedom.

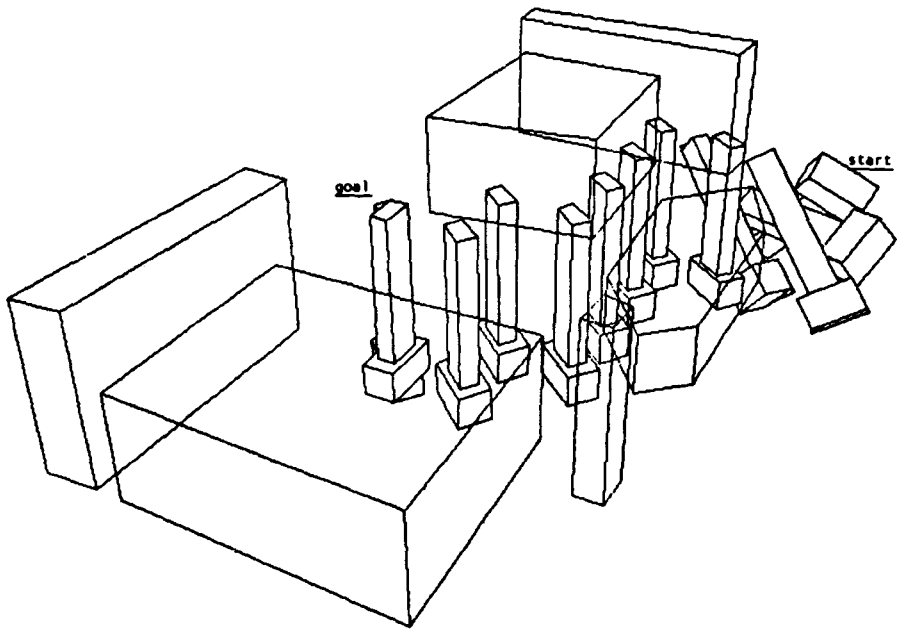


FIG. 3. A different view of the solution path for the hammer example, with the obstacles "transparent" to allow us to view the rotations better.

tion of strategies for planning paths along level C-surfaces and their intersection manifolds, and for reasoning about motions with three degrees of rotational freedom. The problems of controlling the interaction of these strategies, and of integrating diverse local experts for geometric reasoning provide an interesting application of search to a difficult domain with significant practical implications.

Examples of "classical" find-path problems solved by our planner may be found in Figs. 2–8. These problems are beyond the competence of previous implemented algorithms. In general, find-path problems with more than three degrees of freedom have proven extremely difficult to solve. We believe that in part, this difficulty has been due to the unresolved issues in the mathematics of spatial planning. This theoretical foundation of our planner was described in a

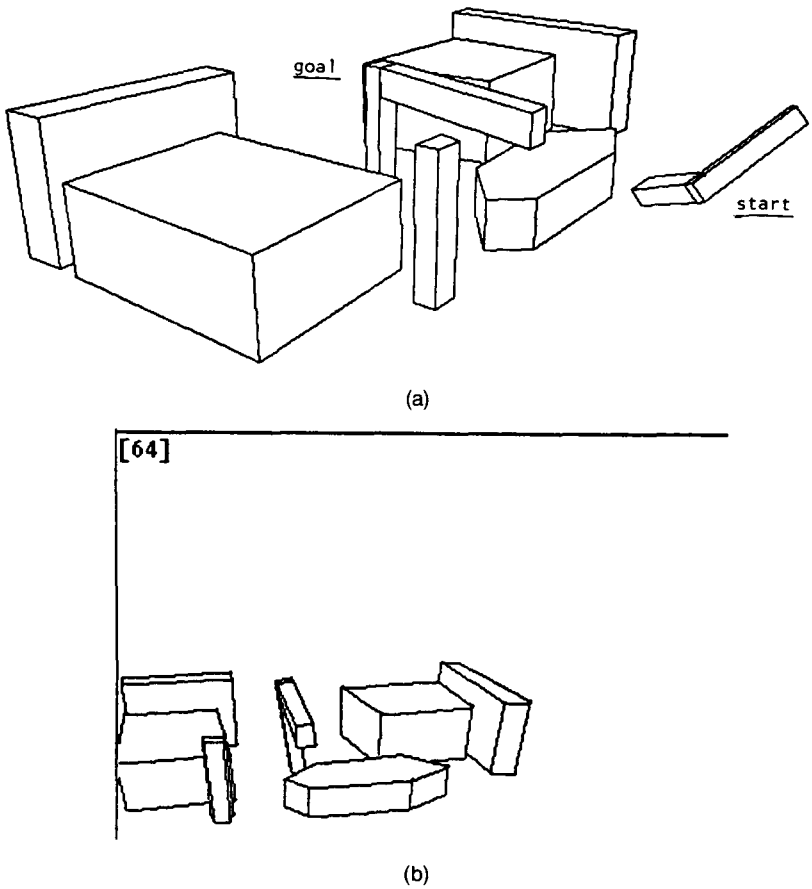


FIG. 4. (a) A find-path problem for an L-shaped object. The L-shaped object is shown amidst obstacles in the start and goal configurations. (b) Solution path I, frame 64 (final configuration).

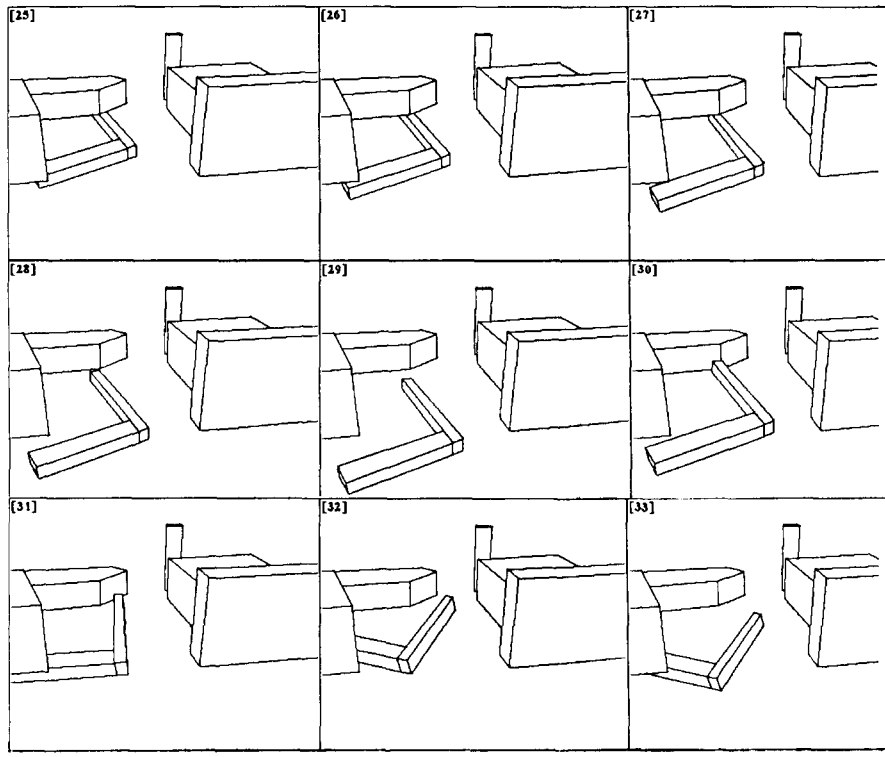


FIG. 5. Detail of solution path for the L-shaped object find-path problem; frames 25–33.

companion report [12]. In an appendix we provide the mathematical details that are relevant. In this paper, we concentrate on the search algorithm with particular emphasis on the heuristic strategies that evaluate local geometric information, and on the interaction of these strategies. Note that the solutions that our planner finds are not "optimal."

### 1.1. Configuration space

The *configuration* of an object is a vector of parameters representing its combined translation and orientation, relative to a specified coordinate system. For the classical Movers' problem in the plane, a typical configuration

$$(x, y, \theta)$$

represents a displacement (translation) of $(x, y)$, and a rotation by $\theta$. (For example, imagine a polygon displaced by $(x, y)$, and rotated by $\theta$ about one of

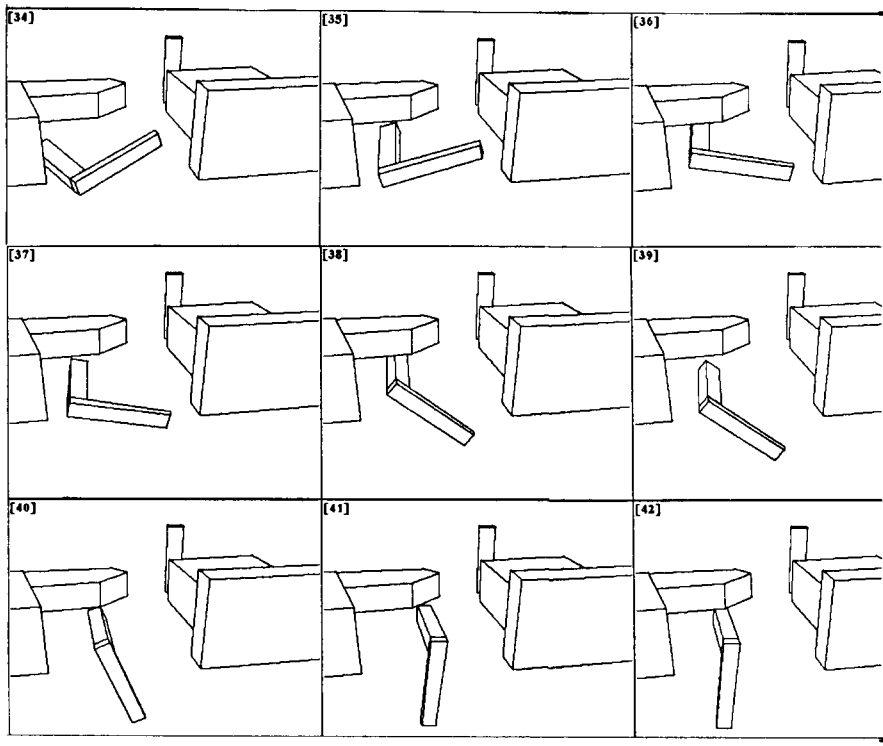


FIG. 6. Detail of solution path for the L-shaped object find-path problem; frames 34–42.

its vertices.) For the six degree of freedom classical Movers' problem, a typical configuration

$$X = (x, y, z, \mathcal{R}(\Theta))$$

represents a displacement (translation) of $(x, y, z)$, and a three-dimensional rotation $\mathcal{R}(\Theta)$. The three-dimensional rotation group is a three-parameter family; typical representations of rotations include Euler angles [42], spherical angles, and quaternions [19]. For example, if the Euler angles $\Theta = (\psi, \theta, \phi)$ are employed, then they determine a 3 by 3 rotation matrix which functions as $\mathcal{R}(\Theta)$ in the rotation group. It is convenient to identify the rotation operator with its parameterization, that is, to express $X$ as

$$X = (x, y, z, \psi, \theta, \phi) ,$$

or, more simply, as

$$X = (r, \Theta) ,$$

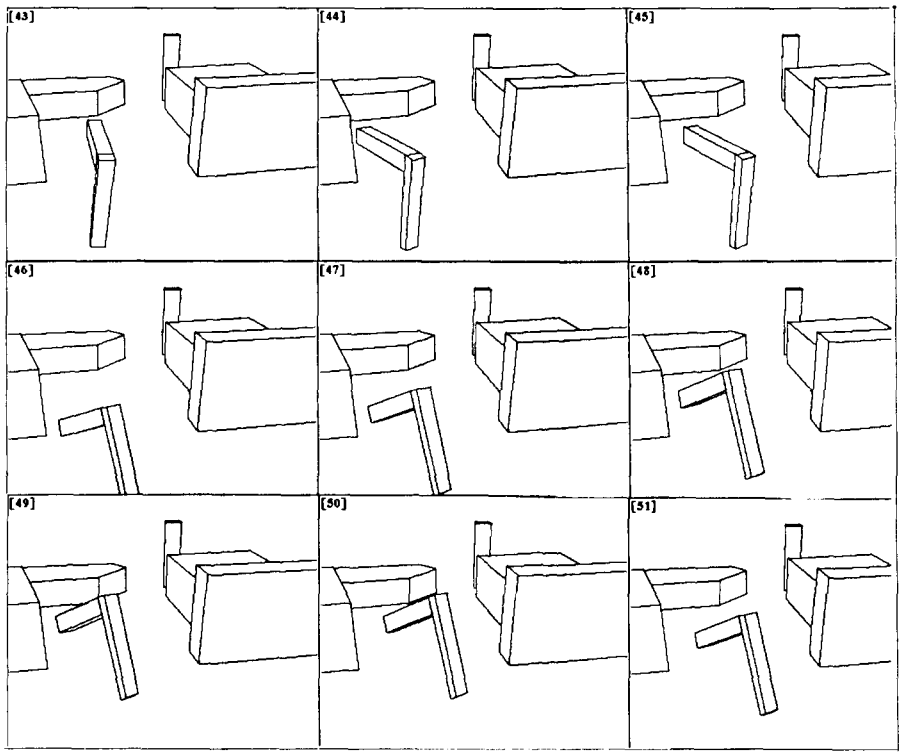


FIG. 7. Detail of solution path for the L-shaped object find-path problem; frames 43–51.

where $\boldsymbol{r}$ denotes a three-dimensional translation vector, and $\Theta$ some three-dimensional rotation. This second notation is independent of the particular representation chosen for rotations.[2]

Using configuration space, reasoning about the motion of a complicated three-dimensional body amongst obstacles may be transformed into reasoning about a point in a six-dimensional configuration space. The transformation described by Lozano-Pérez [24] entails "shrinking" the moving object to a point, and correspondingly "growing" the obstacles. In principle, the point may then be navigated around the grown obstacles by means of the point navigation operators (above).

[2] More formally, the configuration space is the product space of the space of translations and the space of rotations for an object. In three dimensions, the space of translations is Euclidean three-space $\mathbb{R}^3$ and the space of rotations is the three-dimensional rotation group or *special orthogonal group*, SO(3). For the classical Movers' problem we will employ the configuration space $\mathbb{R}^3 \times SO(3)$. In practice, we will represent rotations as members of a three-parameter family (for example, Euler angles), but they parameterize an isometry and $\mathbb{R}^3 \times SO(3)$ is not a vector space.

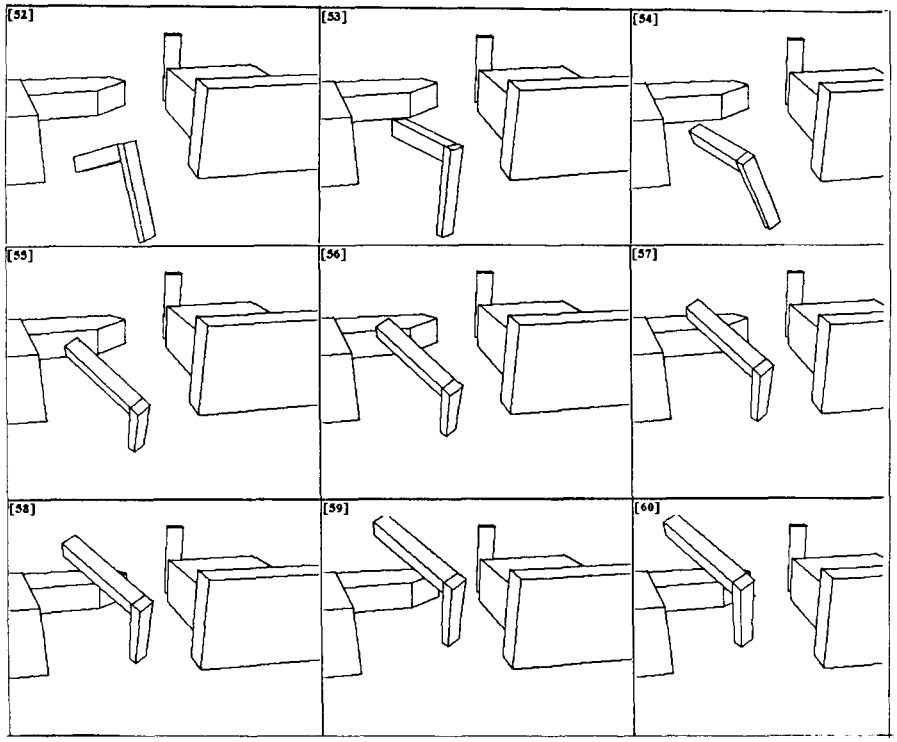


FIG. 8. Detail of solution path for the L-shaped object find-path problem; frames 52–60.

Here is an overview of our representation for C-space obstacles. C-space obstacles are modelled by unions of C-space volumes. These volumes may be represented as the intersection of several generalized half-spaces. Each half-space can be defined in terms of a real function $f$ on C-space, by saying that it's all the points whose $f$-value is nonpositive. Given this description, a surface of the volume is (part of) the surface in C-space defined as those points whose $f$-value is zero. In general, the points whose $f$-value is some constant form a surface "parallel" to the surface of the volume, which we will call a *level* surface. Furthermore, the $f$-value at any configuration represents the translational distance to the surface. One utility of this way of describing obstacles is that our algorithm needs to be able to compute normals and tangents to these surfaces, and these functions give a direct way to do that.

More precisely, a volume in the configuration space $\mathscr{C}$ may be represented by the intersection of a finite number of half-spaces (see Fig. 9). Each half-space may be defined via some smooth, real-valued function of $\mathscr{C}$,

$$f_i : \mathscr{C} \rightarrow \mathbb{R} .$$

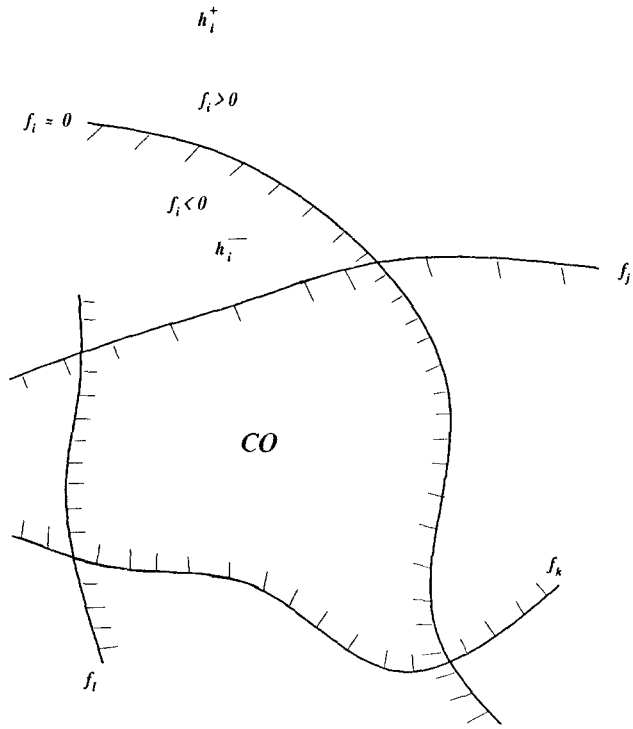


FIG. 9. The region CO is the intersection of the half-spaces $h_i^-$, $h_j^-$, $h_k^-$, and $h_l^-$.

In general, the (closed) half-space $h_i^-$ is the set of all points in $\mathscr{C}$ where $f_i$ is negative-valued or zero:

$$h_i^- = \{v \in \mathscr{C} \mid f_i(v) \leq 0\} .$$

The common intersection of a number of such half-spaces yields a volume in $\mathscr{C}$. Lozano-Pérez [24] showed how C-space obstacles can be modeled in this manner, and gave the form of the functions $f_i$. The boundaries of C-space obstacles are called *C-surfaces*. Note that each C-surface lies within the kernel of some constraint $f_i$. Our search algorithm needs to compute the normal and tangent spaces to these C-surfaces. The normal can usually be derived from the gradient $\nabla f_i$ (this requires placing an appropriate Riemannian metric on the tangent space). When a real-valued function $f_i$ on configuration space is used to describe constraints in that C-space (i.e., C-space obstacles), we call it a *C-function*. The form and interpretation of C-functions are presented in [6, 12, 24]. A surface "parallel" to a C-surface is called a level C-surface, and represents the set of configurations where $f$ has a certain fixed value. This value

is termed the level of the level C-surface. The boundary of the C-space obstacle is a special case of level C-surface, where the level is zero.

With this representation, we can find a collision-free path for a point amidst six-dimensional C-space obstacles in $\mathbb{R}^3 \times \mathrm{SO}(3)$ through the *closure* of the three point navigation operators. Implementing the point navigation operators requires surmounting certain mathematical problems. We present a brief overview of the solutions in an appendix. Two critical concepts described there are *applicability* and *redundancy*. We give informal definitions for them here.

– *Applicability*. A *feature* of a polyhedron is a face, edge, or vertex on its boundary. The C-functions have the property that their conjunction enforces a disjointness criterion for the moving object and the obstacles. Each C-function is generated by a pair $(g_A, g_B)$ where $g_A$ and $g_B$ are features of the moving object and of an obstacle, respectively. The set of all C-functions is generated by considering all pairwise interactions (possible contacts and intersections) of moving object and obstacle features. However, these interactions can only occur in certain orientations; for example, consider two cubes (one moving and one fixed). Since they have 12 edges each, they generate 144 edge-edge constraints. Roughly speaking, we say an edge-edge constraint is *applicable* at an orientation when the generating edges can touch without the cubes' interiors intersecting. However, at any fixed orientation only certain edges can touch and hence only certain edge-edge constraints are applicable.[3] The region of rotation space where a C-function $f_i$ is applicable is called its *applicability region*, $\mathscr{A}_i \subset \mathrm{SO}(3)$. For a given orientation $\Theta$, the set of all applicable C-functions is called the *applicability set*. See Appendix A.2 for a more extensive treatment.

– *Redundant and nonredundant constraints*. If a configuration $X$ is in free space, the set of constraints which is (locally) relevant to motion planning from $X$ is a subset of the applicable, positive-value C-functions at $X$. However, the value of a C-function does more than merely indicate which side of a C-surface $X$ is on. A C-function's value represents the translational distance[4] to that surface. Thus, C-functions provide a collection of pseudo-metrics on C-space. Using these metrics, it is possible to order C-surfaces by their closeness to a configuration $X$ (simply sort the C-functions on their value at $X$). We say that a C-surface is *redundant* if it is subsumed by a nearer, intervening constraint. In Fig. 10, for example, $f$ and $g$ are nonredundant constraints at $X$, but $h$ is redundant since it is subsumed by $f$. It is useful to determine the set of nonredundant constraints at $X$ since this is the smallest set of constraints that are locally relevant to motion planning.

[3] Note that our definition of applicability does not consider the other obstacles. The *feasibility* of a contact is also constrained by the other C-space obstacles.

[4] I.e., the distance in $\mathbb{R}^3$.

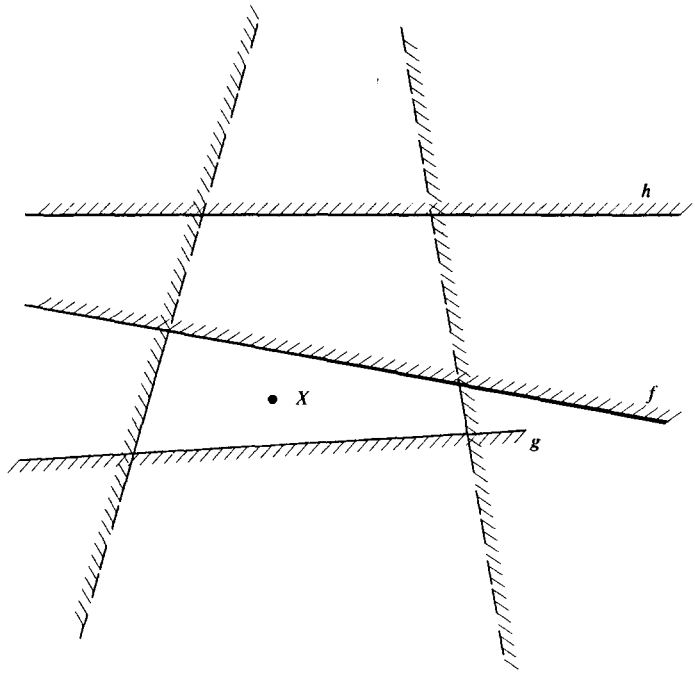


FIG. 10. $h$ is a redundant constraint.

While the point navigation operators define a complete search space, it is evidently too large, in general, to be searched in its entirety. This paper focuses on a search algorithm which uses the point navigation operators as primitives. While still complete (at a resolution), the algorithm employs heuristic strategies to evaluate local geometric information in C-space in order to guide and to limit the search.

### 1.2. Review of previous work

Planning problems have two components: characterizing the constraints and searching for a solution which satisfies the constraints. One attempts to achieve a *complete* (in some sense, "exact") characterization of the constraints and a complete search algorithm for the representation. Previous motion planning algorithms fall into three broad classes: (1) theoretical but unimplemented algorithms which are complete,[5] (2) implemented approximate algorithms which are not complete, and (3) implemented complete algorithms which address three or fewer degrees of freedom. An extensive completeness review

[5] In general, the complete algorithms are complete only to a resolution; exceptions are [33, 34, 38–40].

is beyond the scope of this paper; however, for a detailed analysis, see [12]. The algorithm presented in this paper is a representation-complete, search-complete algorithm (at a given resolution) for the six degree of freedom classical Movers' problem in $\mathbb{R} \times SO(3)$.

This paper is based on [12]. The foundations of our approach lie in [24, 25, 28, 39]. While the theoretical algorithm of Schwartz and Sharir [39] is complete, it is $O(n^{4096})$ for the six degree of freedom problem. Thus it serves chiefly as an existence proof. Reif [37] and Hopcroft et al. [20] have demonstrated complexity-theoretic results. Schwartz and Sharir, in [40], address the coordinated motion of circular bodies in the plane. For a survey of algorithmic motion planning, see [46]. A survey of issues in robotics can be found in [2]. For related work on motion planning, see [8, 17, 21, 29, 30, 38, 43]. Wingham [45] and Popplestone et al. [36] consider related issues in geometric planning problems. In particular, Canny [8] improved on Donald's representation [12] of the configuration space constraints, giving an algebraic formulation which permits a somewhat cleaner solution to certain intersection problems.

Brooks and Lozano-Pérez [6] implemented a general path-finding algorithm for a polygonal object in the plane with two translational and one rotational degrees of freedom. Their planner uses hierarchical subdivision of the three-dimensional configuration space[6] $\mathbb{R}^2 \times S^1$. Lozano-Pérez, in [25], described approximate solutions for Cartesian manipulators with six degrees of freedom (in principle) which consider three-dimensional *slice-projections* of C-space. Lozano-Pérez [26] has recently reported a motion planning algorithm for 6 DOF linked arms which is also complete to a resolution.

Global methods for path planning attempt to construct a model of the connectivity of free space which can be related to the generalized Voronoi diagram (GVD) (see [15]). In particular, Brooks [4] implemented a two-dimensional path-planner which models the free space as an overlapping union of generalized cones. This work was extended to a six-link manipulator for moving payloads with four degrees of freedom [5].

Using an approach called *retraction*, Ó'Dúnlaing et al. [33, 34] construct a Voronoi diagram for a two-dimensional workspace and consider moving simple objects (a disk, a line segment) along it. Nguyen [31] discusses the relationship of global algorithms in the plane to the GVD. Donald presents criteria for the design and integration of local and global algorithms in [11], and in [12] develops a definition and new theoretical results for a six-dimensional extension of the GVD, called the *C-Voronoi diagram*, and relates its structure to the (level) C-surface intersection manifolds. Canny [9] suggests an algorithm for computing the C-Voronoi diagram.

Recently, researchers have begun to consider *fine motion* and motion planning with *uncertainty* [3, 14, 16, 27, 29]. For the six degree of freedom

[6] $S^1$ is the unit circle, on which two-dimensional rotations may be represented.

case, both presume algorithms and representations which are derived in this paper.

## 2. A Planning System for the Classical Movers' Problem with Six Degrees of Freedom

In this section, we describe the design and implementation of a planning system for the classical Movers' problem with six degrees of freedom. The planning algorithm required the solution of several representational and algorithmic questions and the implementation of the point navigation operators. The solutions to these problems are surveyed in Appendices A and B. In this section we will simply assume that these problems are solved, and proceed to employ the solutions in constructing a planning algorithm. Of particular importance will be two effective procedures by Donald [12], which address the *intersection problem* in C-space:

(i) Given two or more level C-surfaces, construct their intersection manifold.

(ii) Given a C-surface and a trajectory,[7] find their intersection. Determine whether the intersection lies on the boundary of a C-space obstacle.

The immediate application of (i) is the *sliding* problem: How to slide along one level C-surface, and how to slide along the intersection of two or more level C-surfaces.

Using the point navigation operators (Section 1), we implemented a best-first search algorithm in C-space. The algorithm has theoretical properties which include completeness (at a resolution). This section describes the heuristic search, with particular emphasis on the heuristic strategies that evaluate local geometric information, and on the interaction of these strategies.

### 2.1. Definitions

A topological space[8] $M$ is called an $n$-dimensional *manifold* if it is locally homeomorphic to $\mathbb{R}^n$. A *chart* is a way of placing a coordinate system on $M$; if $U$ and $V$ are open subsets of $M$, two homeomorphisms $f : U \to f(U) \subseteq \mathbb{R}^n$ and $g : V \to g(V) \subseteq \mathbb{R}^n$ have *$C^\infty$-overlap* if the maps

$$f \circ g^{-1} : g(U \cap V) \to f(U \cap V),$$
$$g \circ f^{-1} : f(U \cap V) \to g(U \cap V),$$

are also $C^\infty$ (that is, possessing continuous partial derivatives of all orders). A family of pairwise $C^\infty$-overlapping homeomorphisms whose domain covers $M$ is

[7] A *trajectory* in C-space is the image of an embedding of the unit interval in $\mathbb{R}^3 \times \mathrm{SO}(3)$.

[8] To be precise, $M$ must in addition be a *second-countable Hausdorff space*.

called an *atlas* for $M$. A particular member $(f, U)$ of an atlas $\mathcal{U}$ is called a *chart* (for the atlas $\mathcal{U}$), or a coordinate system for $U$. For a good introduction to differential geometry, see, for example, [41].

In this paper we usually specify charts via the inverse form $h : R \to M$ (where $R$ is an open subset of $\mathbb{R}^n$) with the understanding that it is the inverse (or set of local inverses) $h^{-1}$ which provides the family of charts $\{(h^{-1}, W_i)\}$, for $\bigcup_i W_i = h(R)$. As an example, consider the map $h$ that specifies a chart for a five-dimensional level C-surface:

$$h : \mathbb{R}^5 \to \mathbb{R}^3 \times \mathrm{SO}(3),$$
$$(y, z, \psi, \theta, \phi) \mapsto \left( -\frac{E_2 y + E_3 z + E_4 - l}{E_1}, y, z, \psi, \theta, \phi \right).$$

Here the $E_i$ are smooth, real-valued functions[9] on SO(3), that is, $E_i : (\psi, \theta, \phi) \to \mathbb{R}$. The inverse map $h^{-1}$ is obvious, and provides a chart for the five-dimensional submanifold of $\mathbb{R}^3 \times \mathrm{SO}(3)$. Donald [12] derives such charts, in the form of $h$; in this paper, we will take them for granted.

### 2.2. Introduction

We are now ready to describe a planning system for the find-path problem in $\mathbb{R}^3 \times \mathrm{SO}(3)$. The algorithm has the structure of a search and is complete (at a given resolution). The basic idea is as follows.

Configurations and tangent vectors are represented as 6-tuples of floating point numbers. C-functions have canonical forms as trigonometric polynomials, given in Appendix B.1. They are represented by their floating point coefficients.[10]

We define a finite partition of C-space into regions called neighborhoods. The neighborhoods are defined by a quantization of the factor spaces of C-space. The size of the quantization, and hence of the neighborhoods, is determined by the resolution. This partition defines the nodes of the search space. The search operators between neighborhoods correspond to motions in configuration space. A best-first search algorithm is employed to find a path.

We are able to define and implement certain procedures called *local operators*. When applied at a (representation of a) configuration, a local operator attempts to move the robot in a specified direction until either the subgoal or an intervening C-surface is reached. The local operators have the general form

Move($X$:configuration, $\hat{v}$:direction, *limit*:configuration),

[9] By applying + to the $E_i$, we of course mean to specify the function whose value is their sum.
[10] In principle, rational arithmetic could be employed instead.

and are designed to return $X'$, the configuration reached in direction $\hat{v}$, and the reason for stopping (which will either be "reached subgoal" or the name of the C-surface which halted progress). The local operator assumes that $X$ is in free space, and ensures that there exists a collision-free path along $\hat{v}$ taking the robot from configuration $X$ to $X'$. Furthermore, we insist that $limit = X + t\hat{v}$, for some positive $t$. In general, $\hat{v}$ can be represented as a tangent vector to $\mathbb{R}^3 \times \mathrm{SO}(3)$; the space of directions is locally isomorphic to $\mathbb{R}^6$.

Many different Move operators can be defined. Let $X = (x, \Theta)$. We will restrict $\hat{v}$ to be either a pure translation

$$\hat{v} \in \mathbb{R}^3 \times \{0\}$$

or a pure rotation

$$\hat{v} \in \{+\hat{\psi}, -\hat{\psi}, +\hat{\theta}, -\hat{\theta}, +\hat{\phi}, -\hat{\phi}\} .$$

The closure of these operators is complete for the space of configurations. By this we mean that in the absence of obstacles, there is some finite sequence of operators which carries any configuration $X$ into any other configuration $Y$. It is often convenient to think of these operators as Translate$(X, \hat{u}, x')$ (where $\hat{u} \in \mathbb{R}^3$ and $x'$ is a goal translation) and Rotate$(X, \hat{\varphi}, \varphi')$ (where $\hat{\varphi}$ is an angular direction and $\varphi'$ is a goal angle). The theory and implementation of Translate and Rotate is discussed in Appendix B.

Given the local operators, we can define more sophisticated local strategies for spatial reasoning. These strategies are implemented by *local experts*[11] in C-space. For example, one local expert attempts to circumnavigate C-space obstacles by sliding along intersections of level C-surfaces. Another, "greedy" expert tries to translate or rotate straight towards the goal. A local expert typically examines the local geometric environment of C-surfaces, their normals and intersections. It also takes into account the history of planning. The local experts can be thought of as issuing "commands" in terms of the local operators. Depending on the results of these attempted motions, an expert may issue other local operator commands, and either directly invoke or leave a forwarding message for another local expert.

To summarize: a local *operator* is an algorithm for moving along a specific trajectory until a constraint is encountered (or a subgoal is reached). A local *expert* is a strategy for choosing the trajectory based on an examination of the history of planning and the local geometry. When a local expert chooses a trajectory, it calls on some sequence of local operators to realize it.

[11] The term *local expert* was brought to my attention in discussions with Van-Duc Nguyen, Tomás Lozano-Pérez, and Rodney Brooks.

### 2.2.1. *Planning and search*

The planning algorithm is implemented as a search of the configuration space. The search constructs a graph of neighborhoods which have been explored. (We will be more precise about the term *neighborhood* later.) Each mode in the search graph is associated with a configuration and contains information about the local geometry and the history of planning. The search algorithm chooses a node for exploration. Several local experts are then applied at that node. Each expert can produce a new search node. All of these are sons of the explored node, and are added to the search queue. The new sons are connected to their father by the arcs of the search graph and each son may be thought of as an *exploration* from the father.

If at any point in the search, two explorations reach the same neighborhood, the planner attempts to merge the associated nodes into one node.

The search algorithm is best-first [32] with the metric of progress established as distance from the goal. (This requires placing a metric on both translation and rotation space.) Other search measures (such as path length, or time) would also be possible, and an A* search strategy could be exploited to find optimal paths. In practice this would probably require adding new local experts in order to ensure reasonable performance.

As search nodes are explored, they are entered in a priority queue, called the *search queue*. The nodes in the search queue are ordered by a metric on configurations. Some search strategies we discuss require two search queues: when the primary queue is exhausted, then nodes from the reserve queue are explored.

We will proceed as follows. First, using the local operators alone, we can define a complete search strategy (at a given resolution). This search strategy can be considered the most primitive local expert, and is known as the "bumble strategy." By applying the bumble strategy at every search node, we are guaranteed to find a path (at a given resolution) if one exists (Fig. 11).

Next, we will define more complicated local experts which will be applied to search nodes at the same time as the bumble expert. These experts greatly improve the performance of the planner.

## 2.3. A complete search strategy

A search node is associated with a configuration. Every configuration is in turn associated with a neighborhood of C-space. The neighborhoods form a partition of C-space. Since many configurations are associated with one neighborhood, so several search nodes may have configurations lying in the same neighborhood.

Assume the neighborhoods are "small." If the configurations of two search nodes are in the same neighborhood, it indicates that they should, if possible,

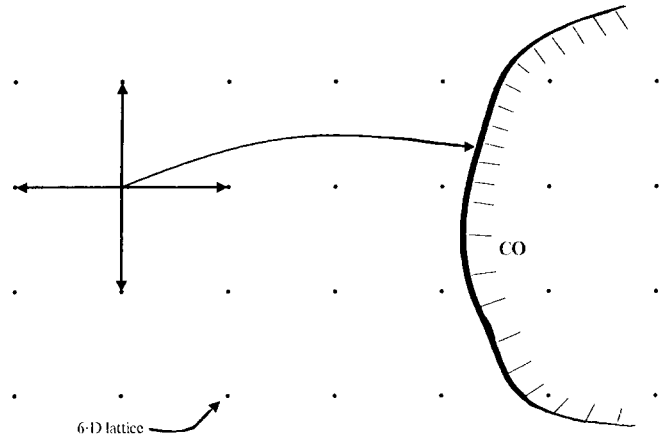


FIG. 11. Schematic illustration of the "bumble" strategy (an exhaustive search). A fine six-dimensional lattice is thrown across C-space. By exploring from one configuration to its neighbors in the lattice, a path will eventually be found, if one exists at the lattice resolution. Fortunately, it is also possible to take large steps in the lattice, and simply record the neighborhoods the path visits.

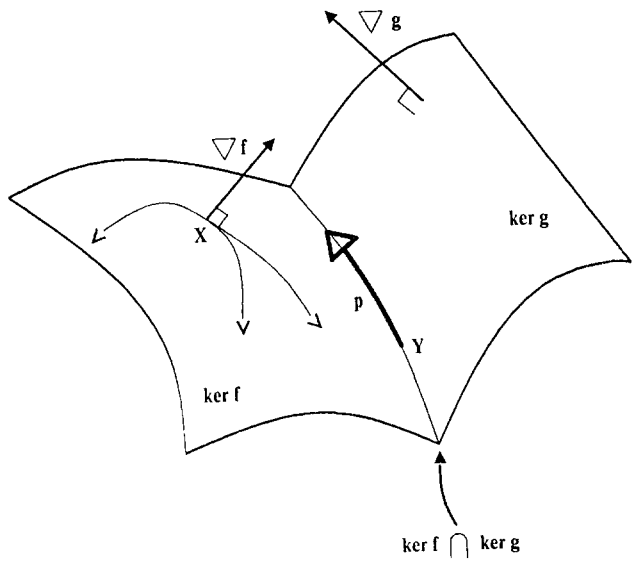


FIG. 12. $f, g : \mathbb{R}^3 \times SO(3) \to \mathbb{R}$ are C-functions which describe two level C-surfaces, ker $f$ and ker $g$. The level C-surfaces are smooth, five-dimensional manifolds parallel to C-space obstacle boundaries. From $X \in \ker f$, three paths sliding along the level C-surface ker $f$ are shown. Each path is orthogonal to $\nabla f$. The *sliding expert* plans such paths along five-dimensional level C-surfaces. $(\ker f) \cap (\ker g)$ is the intersection of the two level C-surfaces, and is a four-dimensional manifold. The *intersection expert* plans paths along intersection manifolds. Such a path $p$ is shown from configuration $Y$.

be merged into one node, since they are close together. By keeping track of the set of explored neighborhoods, we can avoid redundant explorations. If the neighborhoods are sufficiently small, then the search will be complete at a resolution closely related to the neighborhood size.

It is possible to devise a complete search strategy (at a given resolution) using just the local operators. We first throw a fine six-dimensional lattice[12] over configuration space. The lattice is used to keep track of the *state* of the planner, i.e., which neighborhoods have been explored, and for computing the *connectivity* of these neighborhoods. The lattice will "wrap around" in the rotational dimensions, but this is easily implemented using modular arithmetic. We will define an adjacency function for points in the lattice; in addition, when a neighborhood is explored, the corresponding node in the lattice is marked. When a search node is chosen for exploration:

(i) $X$, the configuration of the search node, is mapped to $L$, a point in the lattice. $L$ is the *name* of the neighborhood $\mathcal{N}(L)$ centered on $L$, which contains $X$.

(ii) The unexplored neighborhoods adjacent to $\mathcal{N}(L)$ are found. Each of these neighborhoods is also identified by a central lattice point.

(iii) The planner attempts to move to each of the unexplored, adjacent neighborhoods.

(i) has the effect of mapping a neighborhood of C-space to a canonical element (which lies on the lattice) in its interior. These neighborhoods decompose $\mathbb{R}^3 \times \mathrm{SO}(3)$ into equivalence classes with the same canonical element. When a neighborhood is reached for the first time, we mark its lattice point as explored. The search terminates when a neighborhood containing the goal is reached, and when that exploration can be connected to the goal configuration.

### 2.3.1. *Implementation of neighborhoods and lattices*

In principle, it is possible to implement the lattice as a six-dimensional array (with modular indexing for the rotational dimensions). In practice, for any fine resolution, this array will be enormous, and very sparse. Although an adversary can design a find-path problem for which our planner must explore the entire lattice, in practice this does not occur. However, we must maintain a record of what neighborhoods have been explored, in order to generate the unexplored neighbors for a search node. Since the array is sparse, we will employ a different strategy.

A partial order can be defined on lattice points by considering them as six-dimensional vectors. This order has no particular geometric significance for the rotational dimensions, but it can be used to store explored lattice points in a binary tree. Since the vast majority of neighborhoods are never explored, the

[12] I.e., the factor spaces of the parameter space are quantized, and the lattice is a partial order on the Cartesian product of the factor space quantizations.

tree is typically small, even for fine lattices. To mark a lattice point as explored, we insert into the binary tree. To find whether a lattice point has been explored, we search the tree.

It is desirable to employ a fine lattice in order to ensure completeness at a fine resolution. The use of a binary tree to record explored configurations effectively removes the problem of lattice size for storing explored configurations. For example, if we segment C-space into an $N \times N \times \cdots \times N$ lattice, then an array would have to be $N^6$ long. But the binary tree need store only the explored location, and (if height-balanced) can access any leaf in $O(\log N)$ operations.

If the lattice resolution is fine, then the planner as described so far will take very small steps for each search exploration. This has been remedied as follows: If a local operator is invoked to find whether *limit* may be attained from $X$ in direction $\hat{v}$, it must effectively intersect a path in direction $\hat{v}$ with all C-surfaces. It is not much harder to find the *first* constraint along the path $p(t) = X + t\hat{v}$ (even if it is beyond *limit*): in particular, we note that all intersections along the path $p$ may be sorted on distance from $X$. The complexity of finding this first intersection along $p$ is independent of the lattice resolution (since the intersection algorithm has nothing to do with the lattice; see [12]). We can "sample" the portion of the path which lies in free space at the lattice resolution. All of these configurations are then marked as "explored," and as reachable from their immediate neighbors along the path. Thus they form a connected chain in the lattice along the path $p$. While all these configurations are in some sense sons of $X$, in practice we will select only one or two to be entered in the primary search queue. These sons might be (1) the son which is closest to the goal, and (2) some son at a reasonably large step away from $X$. This step size, called the *bumble resolution*, might be 3 to 10 times the lattice resolution. The other sons should be kept on a reserve queue, which can be explored when the primary search queue is depleted or exhausted.

In practice, it may be preferable to enter *ranges* in the exploration tree, for example, to record that all lattice points

$$(x, y, z, \psi, \theta, \phi) \leq L \leq (x + kd_T, y, z, \psi, \theta, \phi)$$

(for some integer $k$) are explored. This requires keeping an exploration tree of *lines* instead of configurations, with the intent of minimizing the number of exploration tree entries. When lines are entered into the tree, they may be merged with previous lines to form connected components of explored regions. These operations are supported by hierarchical subdivision algorithms. At this point in the experimental use of the planner, it is still too early to tell whether this optimization is necessary.

In practice we have had no problem in selecting a very fine resolution for the

lattice (one selects a fine lattice resolution, and a considerably larger Bumble resolution or step size, as described above). This lattice-based strategy is not only theoretically complete for a given resolution, but has also been used to find very complicated paths for the six degree of freedom classical Movers' problem. However, the algorithm has an "excessively local" flavor—it is clumsy and quite slow when employed alone (hence the strategy's name). We can construct much "smarter" heuristic experts which attempt to exploit coherence in C-space. When these experts are used in conjunction with the bumble strategy, we obtain a planner which is not only complete, but which can solve complicated problems in a reasonable amount of time. We continue to find the lattice useful for recording the planner's explorations by the local experts.

### 2.3.2. *Keeping track of connectivity*

Suppose a subsequent exploration reaches the same neighborhood. There are two choices, which we call the *mark* algorithm and the *connect* algorithm:

– *The mark algorithm.* Discard the exploration, since the neighborhood is already explored. In practice, the mark algorithm often suffices for path-finding. The mark algorithm computes a directed, spanning tree $T$ of explored neighborhoods, which is rooted at the start configuration.

– *The connect algorithm.* Connect together the search nodes for all explorations to that neighborhood. The connect algorithm is more complicated, and requires the following bookkeeping (see Fig. 13). Let $\mathcal{N}$ be a neighborhood of

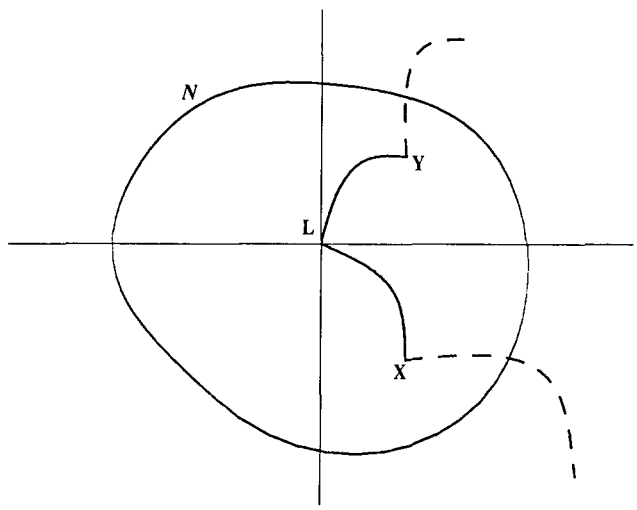


FIG. 13. The lattice point $L$ is at the center of a neighborhood $\mathcal{N}$ of C-space. Search explorations arrive at configurations $X$ and $Y$ in $\mathcal{N}$. The planner attempts to find a path connecting $X$ and $Y$, by trying to connect both configurations to $L$.

$\mathbb{R}^3 \times SO(3)$, and $L \in i(\mathcal{N})$ be a lattice point which is the canonical element for $\mathcal{N}$. Suppose $X$ is an exploration of $\mathcal{N}$, i.e., $X \in \mathcal{N}$ is the final configuration in some motion reaching $\mathcal{N}$. Let $s(X)$ denote the search node for $X$. (If $X$ is the first exploration of $\mathcal{N}$, then create a search node $s(L)$ for $L$.) Determine whether there exists a path from $X$ to $L$ (using the local operators). If so, connect $s(X)$ and $s(L)$ together.

The connect algorithm computes a more complete connectivity graph for the neighborhoods of C-space. It computes an undirected graph $H$ of explored neighborhoods, which may contain cycles. As long as $H$ is connected, then $T$ is a spanning tree for $H$, and the mark algorithm is complete for planning a connected path along $H$. However, not all planning strategies admit this kind of "connected planning." In particular, when we consider strategies which construct partial paths and planning islands (which may later connect up), the connect algorithm is necessary (see the *suggestor* strategy, Section 2.4.8, for an example).

### 2.3.3. *Discussion of the bumble strategy*

Suppose the lattice spacing is $d_T$ and $d_R$ in the translational and rotational dimensions. Then the adjacent lattice points to $L = (x, y, z, \psi, \theta, \phi)$ will be:

$$
\begin{aligned}
&(x \pm d_T, y, z, \psi, \theta, \phi)\,,\\
&(x, y \pm d_T, z, \psi, \theta, \phi)\,,\\
&(x, y, z \pm d_T, \psi, \theta, \phi)\,,\\
&(x, y, z, \psi \pm d_R (\mathrm{mod}\, 2\pi), \theta, \phi)\,,\\
&(x, y, z, \psi, \theta \pm d_R (\mathrm{mod}\, \pi), \phi)\,,\\
&(x, y, z, \psi, \theta, \phi \pm d_R (\mathrm{mod}\, 2\pi))\,.
\end{aligned}
$$

Each adjacent lattice point is the center of a neighborhood of configurations which is contiguous to the neighborhood of $L$. Each such neighborhood can be reached (if it is in free space and there is no intervening C-surface) by the local operators Translate and Rotate. Since there are 12 neighbors for each lattice point, we have found it inadvisable to explore them all for each search node expansion. Instead, the set of unexplored adjacent neighborhoods is ranked (in terms of proximity to the goal), and motions towards the top $k_T$ translational and $k_R$ rotational neighbors are attempted (typically, $k_T \approx 3$ and $k_R \approx 2$). If the node is reexplored later, motions toward $k_T + k_R$ more of the unexplored neighbors will be attempted (if there are that many left). When using the mark algorithm, we say an exploration is successful if it reaches a new (unexplored) neighborhood. If an exploration is successful, then a new search node is created and the neighborhood is marked as explored. Since the neighborhood's "name" is its lattice point, this simply corresponds to marking the lattice point. Whether successful or not, all explorations are recorded at the parent search node so that they will not be tried again.

Suppose $X$ is a configuration in neighborhood $\mathcal{N}(L)$, with associated lattice point $L$. The unexplored adjacent lattice points to $L$ indicate a set of subgoals to be attained from $X$. The bumble strategy ranks these subgoals, chooses some of them, and selects trajectories which may attain them. The local operators are then employed to (try to) realize the selected trajectories. These explorations are then recorded so that only new explorations will be pursued in the future. Note that the planner is not constrained to move along the lattice, and that although the subgoals lie on the lattice, the motion from $X$ to any subgoal does not, unless $X = L$.

The local experts are considerably more sophisticated than the bumble strategy. Their subgoals need not lie on the lattice, and the motions specified to the local operators need not lie along the lattice. The lattice is still employed to keep track of the planning history and the connectivity of explored neighborhoods.

Clearly, the arcwise-connected sets of lattice points are closed under the operators Translate and Rotate. If a path exists at the lattice resolution, then the search is guaranteed to find it. We see now exactly what the *resolution* for this find-path algorithm is: by choosing a sufficiently fine lattice, the algorithm is (trivially) complete at the lattice resolution. One final point: the start and goal configurations may not lie directly on the lattice. This is not a problem, however, since the local operators can ensure that there exists a path from the start and goal to the nearest lattice point.

## 2.4. Local experts for the find-path problem

### 2.4.1. *Path planning versus continuous intersection detection: Why we need local experts*

The Translate and Rotate operators detect collisions along continuous trajectories.[13] Given these operators, it is possible to devise a complete path-planning algorithm based on something like the bumble strategy, above. However, while complete, this is not a particularly *good* algorithm, in that it says nothing about how or when the operators should be applied. The domain of the operators is large and for realistic path planning, it is necessary to know where, and in what directions, to apply them.

> Algorithms which can detect intersections with obstacles for a robot following a continuous trajectory say nothing about how to plan these trajectories. However, they can be used to find a path by exhaustive search.

The Translate and Rotate operators use the constraints in C-space to detect

[13] This discussion also holds for the general Move operator.

collisions. However, these constraints can also be employed to plan paths. In Section 1, we proposed an idealized planner which constructed the intersection manifolds of level C-surfaces, and slid along these manifolds to navigate around C-space obstacles. Such a planner could exploit *coherence* in the configuration space: by examining C-space constraints an algorithm can be devised for intersecting and sliding on C-surfaces to circumnavigate C-space obstacles. In the following sections, we describe a planner which approaches the idealized planning algorithm of Section 1. The local experts are strategies for reasoning about the local geometry of the configuration space, and for exploiting geometric constraints to plan collision-free paths. When applied to a search node, each local expert examines the local geometry and history of planning to propose one or more path segments. Each path segment is realized by means of the local operators, which ensure that a collision-free path exists.

### 2.4.2. *Designing local experts*

In the exploration tree of C-space neighborhoods, we have seen one type of information that must be maintained for planning. In designing local experts, we must address the following questions:

(i) What constitutes a local description of a (level) C-surface?

(ii) What information should be stored at a search node?

Question (i) can be stated, "What constitutes a sufficiently rich description of the local geometry in C-space to allow robust local experts?" Question (ii) relates more to the history of planning, and the connectivity of the explored search neighborhoods. For example, we want to record the results of previous applications of experts at a search node, and the adjacent nodes in the search graph.

*The local description of a C-surface*

A C-surface has a normal at point $X$. Motions tangent to the C-surface at $X$ will have instantaneous velocities orthogonal to the normal. We must characterize the normal and tangents to a C-surface in order to plan trajectories which slide along it.

Let $f$ be an applicable, positive-valued C-function at $X$. We can check that $f$ is nonredundant at $X$; alternatively, we may heuristically assume $f$ is nonredundant if its value at $X$ is small. We wish to develop a local characterization of $f$ at $X$, that is, of the level C-surface $S = \{Y \mid f(Y) = f(X)\}$ about $X$. We should think of $S$ as the kernel of the auxiliary function

$$f_X : \mathbb{R}^3 \times \mathrm{SO}(3) \to \mathbb{R}\,, \qquad Y \mapsto f(Y) - f(X)\,.$$

The local characterization will have two parts, one of which is invariant, and one of which will change for different subgoals. The invariant part of the description is a pair

$$(f(X), \nabla f)$$

consisting of the *value* of $f$ at $X$ and the *normal* to $S$ at $X$. Now, the normal $\nabla f(X)$ to $S$ at $X$ will depend on the Riemannian metric defined on the tangent space at $X$. In principle, there is a natural choice for the Riemannian metric as follows [1, 16]. The moment of inertia tensor of the moving object defines a field of inner products on C-space. This choice is natural in the sense that it yields a quadratic form which computes the correct kinetic energy for the moving object. However, for computational reasons we will use a simpler metric. For gross motion, it is sufficient to employ a metric which admits construction of $\nabla f(X)$ using the partial derivatives of $f$ at $X$, with respect to the parameterization of C-space. Hence if rotations are parameterized by Euler angles, then

$$\nabla f = \left( \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi} \right).$$

Assume that $\nabla f$ is normalized to be a unit vector. We now wish to characterize the relationship of the C-surface to some subgoal, $G$: this requires some way of talking about directions in $\mathbb{R}^3 \times \mathrm{SO}(3)$. Specifically, we wish define a "vector" algebra on configurations, such that

$$\lim_{G \to X} \| G - X \| = 0$$

and

$$\lim_{G' \to G} (G - X) \cdot (G' - X) = 1 .$$

These equations express the vector space characteristics which are required for our computations on tangent vectors. To construct this algebra, it is possible to define a field of inner products over $\mathbb{R}^3 \times \mathrm{SO}(3)$, i.e., to define an inner product on the tangent space to each point. Thus $\mathbb{R}^3 \times \mathrm{SO}(3)$ is a Riemannian manifold [1]. If two tangent vectors—i.e., directions—are applied to the same point, this inner product allows us to talk about the angle between two such tangent vectors, or of the angle between an arbitrary tangent vector to $\mathbb{R}^3 \times \mathrm{SO}(3)$ and the normal to a C-surface. However, the inner product is somewhat arbitrary for our application. Alternatively, we could also construct geodesics on $P^3$, the 3-sphere with antipodal points identified. These approaches are probably too elaborate for a heuristic strategy.

*Heuristics for evaluating directions in the tangent space*

A basic issue is that placing a metric on a non-Abelian group, such as SO(3), is a difficult problem. We will demonstrate the metric that our planner employs, and then show that it is adequate for this application. In particular, the metric is adequate when applied to three one-dimensional slices of SO(3). (These are

the slices considered by the Rotate operator.) Note, however, that a metric may also be derived by representing rotations as unit quaternions. In this case, the metric is obtained by considering rotations as points on the 3-sphere $S^3$ embedded in $\mathbb{R}^4$ [7].

Suppose we employ rotation matrices to represent rotations. (The implemented planner uses Euler angles.) If we are willing to tolerate singularities in the representation, it is often convenient to identify a rotation matrix in SO(3) with the vector of three angles, $(\psi, \theta, \phi)$ which determine it. The angles $(\psi, \theta, \phi)$ form a three-dimensional *angle space*, $Q^3$. The rotation matrix corresponding to $(\psi, \theta, \phi)$ is of course $\mathcal{R}(\psi, \theta, \phi)$. (The singularities induce an equivalence relation on $Q^3$, where two points in angle space are equal when the rotation matrices they determine are equal.) Most of the time, the identification of SO(3) with $Q^3$ does not lead to problems. However, when we wish to compute directions, and differences of configurations, it is necessary to distinguish between SO(3) and $Q^3$.

We can state this more concisely as follows: SO(3) is a three-dimensional manifold. The mapping $\mathcal{R}$ from Euler angles to rotation matrices is a chart for SO(3):

$$\mathcal{R} : Q^3 \rightarrow \mathrm{SO}(3) .$$

We typically describe a rotation $\mathcal{R}(\Theta) \in \mathrm{SO}(3)$ by its chart coordinates $(\psi, \theta, \phi) = \Theta \in Q^3$. This makes it convenient to identify $\Theta$ with $\mathcal{R}(\Theta)$, so that in general, instead of dealing with the manifold directly, we will work with a chart for the manifold. In this section alone, however, we must distinguish between the domain and image of $\mathcal{R}$.

We can compute a direction in $\mathbb{R}^3 \times \mathrm{SO}(3)$ by simply subtracting two configurations (of course the angles must be subtracted[14] $(\mathrm{mod}\, 2\pi)$) to yield a six-dimensional direction vector. Using this arithmetic, the *goal direction* is denoted $G - X$. We will use the convention that the first three coordinates of $G - X$ arise from $\mathbb{R}^3$, and the second three coordinates arise from $Q^3$.

Let $G = (G_x, G_\Theta)$ and $X = (X_x, X_\Theta)$. Since $G - X$ is clearly well defined when $G$ and $X$ differ only by a translation, assume that $G$ and $X$ differ only by a rotation. Assume further that rotations are represented by Euler angles. Note that, in general $G - X$ is not a rotation which carries the moving object at orientation $X$ into the moving object at orientation $G$. However, $G - X$ does represent the difference in orientation, i.e., it specifies a displacement in the *angle space* which will carry $X$ into $G$. For example, if $G_\Theta = (45°, 50°, 90°)$ and $X_\Theta = (45°, 45°, 45°)$, then there are rotation matrices $\mathcal{R}(G_\Theta)$ and $\mathcal{R}(X_\Theta)$. (We use degrees, not radians in this example, since the symbol $\pi$ will soon be used for a projection map). Note that

[14] $(\mathrm{mod}\, \pi)$ for the middle Euler angle.

$$\mathcal{R}(45°, 50°, 90°) \neq \mathcal{R}(45°, 45°, 45°)\mathcal{R}(0°, 5°, 45°) ,$$

where $\mathcal{R}\mathcal{R}'$ indicates composition of rotations. However, the path in the angle space

$$\begin{aligned} p(t) &= X_\Theta + t(G_\Theta - X_\Theta) \\ &= (45°, 45°, 45°) + t(0°, 5°, 45°) \quad \text{for } t \in [0, 1] \end{aligned}$$

will work, since it corresponds to the rotational path

$$\begin{aligned} \mathcal{R}(p(t)) &= \mathcal{R}(X_\Theta + t(G_\Theta - X_\Theta)) \\ &= \mathcal{R}((45°, 45°, 45°) + t(0°, 5°, 45°)) . \end{aligned}$$

Considering configuration space as the product space of translation space and the angle space, we see that $G - X$ is well defined. $G_\Theta - X_\Theta$ specifies a direction and a distance to be traveled in angle space in order to carry $X_\Theta$ into $G_\Theta$. Furthermore, along the path from $X_\Theta$ to $G_\Theta$, the corresponding rotations specified by the angle space trajectory $p$ are well defined. For all $G \in \mathbb{R}^3 \times \mathrm{SO}(3)$, we will treat the space of directions $G - X$ as the tangent space $T_X$ to $\mathbb{R}^3 \times \mathrm{SO}(3)$ at $X$. Properly, $T_X$ is the product space of the tangent space to $\mathbb{R}^3$ at $X_x$, and the three-dimensional angle space $Q^3$.

We now define a map from $T_X \times T_X$ to the plane, which will function in place of an inner product. First, define the natural projection maps from $T_X$ onto its factor spaces:

$$\pi_{\mathbb{R}^3} : T_X \to \mathbb{R}^3 , \qquad (G - X) \mapsto (G_x - X_x) ;$$

$$\pi_\Theta : T_X \to Q^3 , \qquad (G - X) \mapsto (G_\Theta - X_\Theta) .$$

Let $u \cdot v$ denote the standard inner product on $\mathbb{R}^3$, for vectors $u$ and $v$. If $u$ and $v$ are projections (under $\pi_{\mathbb{R}^3}$) of direction vectors in $T_X$, we say that $u$ and $v$ are *translationally orthogonal* if $u \cdot v = 0$. Let $(q_1, q_2, q_3), (w_1, w_2, w_3) \in Q^3$. Assume that each pair of angles $q_i$ and $w_i$ (for $i = 1, 2, 3$) is normalized so that

$$|q_i - w_i| \leq 180° .$$

(Note that this normalization is critical.) Now, define

$$n_Q((q_1, q_2, q_3), (w_1, w_2, w_3)) = q_1 w_1 + q_2 w_2 + q_3 w_3 .$$

$n_Q$ will function in place of an inner product on $Q^3$. We say that two rotational directions $q$ and $w$ are *rotationally orthogonal* if $n_Q(q, w) = 0$.

We may now define $\Phi_X$, which will function in place of an inner product on $T_X$. First, let

$$D = G - X\,, \qquad D' = G' - X\,.$$

Assume that $D_x$, $D'_x$, $D_\Theta$, and $D'_\Theta$ are all normalized to be length 1 (where the length of $D_\Theta$ is defined as $n_Q(D_\Theta, D_\Theta)^{1/2}$). Finally,

$$\Phi_X : T_X \times T_X \to \mathbb{R}^2\,,$$
$$(D, D') \mapsto (\pi_{\mathbb{R}^3}(D) \cdot \pi_{\mathbb{R}^3}(D'), n_Q(\pi_Q(D), \pi_\Theta(D')))\,.$$

So $\Phi_X$ yields a pair consisting of the dot product of the translational components of the direction vectors, and the $n_Q$ product of the rotational direction vectors. If $\Phi_X(D, D') = (0, 0)$ we say that $D$ and $D'$ are orthogonal directions in the tangent space $T_X$. Note that two directions are orthogonal if, and only if, their translational components are orthogonal and their rotational components are orthogonal.

This discussion extends naturally to other representations for rotations. For example, if spherical angles [22] are used, then the difference in orientation *is* the rotation carrying $X$ into $G$, that is, $G_\Theta - X_\Theta$ is a rotation carrying the moving object at orientation $X_\Theta$ into the moving object at orientation $G_\Theta$. We should stress that the natural Riemannian inner product could be used instead of $\Phi_X$. This would complicate the representations employed in subsequent sections. $\Phi_X$ and $n_Q$ are heuristic measures on directions in $T_X$. We will later discuss why, for our purposes, they are good heuristic measures.

*Evaluating normals and gradients to C-surfaces*

The local description of a C-surface relative to some subgoal is designed to address the following qualitative questions:

(i) Is the C-surface locally tangent or locally orthogonal to the goal direction?

(ii) Is the C-surface locally orthogonal to any rotational motion?

Recall that a level C-surface $\ker f$ is described by a real-valued C-function $f$. Assume that normals and tangent vectors are appropriately normalized. Question (i) may be resolved by examining

$$\Phi_X((G - X), \nabla f(X))\,. \tag{2.1}$$

When (2.1) approaches $(0, 0)$, we say that $\ker f$ is locally tangent to the goal direction. Note that (2.1) makes sense: $f$ maps parameters of the form $(x, y, z, \psi, \theta, \phi)$ to real numbers, and hence the gradient of $f$,

$$\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi}\right)$$

is clearly a direction in $T_X$.

We will also employ

$$\pi_{\mathbb{R}^3}(G - X) \cdot \pi_{\mathbb{R}^3}(\nabla f(X)) . \tag{2.2a}$$

When (2.2a) approaches 0, we say that ker $f$ is (locally) translationally tangent to the goal direction. Symmetrically, when (2.1) (resp. (2.2a)) approaches (1, 1) (resp. 1), we say that ker $f$ is locally orthogonal (resp. translationally orthogonal) to $G - X$. A similar calculation yields the rotationally tangent and orthogonal C-surfaces to the goal direction:

$$n_Q(\pi_\Theta(G - X), \pi_\Theta(\nabla f(X))) . \tag{2.2b}$$

*Why $\Phi_X$ and $n_Q$ are good heuristic measures*

Suppose that the rotational direction is along one of the axes. (Let us say the direction is $\hat{\phi}$.) To tell whether a C-surface is rotationally orthogonal (or tangent) to the $\hat{\phi}$-direction, we simply examine the magnitude of $\partial f/\partial \phi$, which can be obtained directly from $\nabla f(X)$. This is because

$$n_Q(\hat{\phi}, \pi_\Theta(\nabla f(X))) = n_Q\left((0, 0, 1), \left(\frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi}\right)\right) = \frac{\partial f}{\partial \phi} .$$

In other words, the map $n_Q$ need not be employed. Since the implemented Rotate operator moves along the rotational axes in directions

$$\hat{v} \in \{+\hat{\psi}, -\hat{\psi}, +\hat{\theta}, -\hat{\theta}, +\hat{\phi}, -\hat{\phi}\} ,$$

this is the most common—but not the only—test for rotationally orthogonal (or tangent) C-surfaces. This information is used by the rotation experts to choose rotational subgoals that move away from C-surfaces.

*Description of a search node*

The following information is stored at each search node. Lazy evaluation is implemented so that some of these objects (for example, the set of all applicable C-surfaces) may not be computed until they are required.

(i) The configuration $X$ of the search node.

(ii) The lattice point for $X$, which is the unique identifier for the neighborhood about $X$.

(iii) The applicability set at $X$.

(iv) $A$, the set of nonredundant constraints at $X$, sorted on increasing

value. The nonredundant constraints may be approximated by the applicable constraints having small positive (or zero) values at $X$.

(v) The parent node.

(vi) The *From-Direction* (the direction traversed from the parent node to this node).

(vii) The sons of this node. These include "unsuccessful" explorations which did not reach a subgoal, or which reached a previously explored neighborhood.

(viii) The C-surfaces on which $X$ lies which also bound C-space obstacles, that is, all $f \in A$ such that $f(X) = 0$ and $\ker f$ bounds a C-space obstacle at $X$.

(ix) An *explanation* of how this node was reached. An explanation typically includes the *name* of the local expert that planned the move, and enough information to reconstruct the move. For example, the experts which slide along level C-surfaces leave an explanation containing the names of the constraints, their *levels* at the parent node, and the *parameterization* chosen for the intersection manifold.

Much of the information stored at a search node is used to record the history of the planning. An expert which planned the move to a search node $s$ will not be applied again with the same parameters. As an example, consider the intersection expert, which attempts to slide along intersection manifolds, and the greedy expert, which attempts to move straight towards the goal. We discuss these experts in more detail in the next section. If applied to $s$, the C-surface intersection expert will not attempt to construct and slide along the same intersection manifold which led to $s$, unless it can slide in a different direction along the intersection manifold. By recording the From-Direction for a node, the planner can avoid repeating unfruitful explorations. In particular, different experts can advise motion in the same direction; thus a particular intersection manifold may point in the same direction which was previously (or simultaneously) attempted by the greedy expert. Whether successful or not, reexploration in this direction may be avoided by examining the From-Directions of the sons of $s$. An additional constraint is provided by the From-Direction of $s$ itself: there is typically no point in exploring back in the direction we came from. The process of leaving information for some expert which may be applied in the future is known as "forwarding." As we shall see, the performance of one expert can provide strong hints as to what expert should be applied next.

The planner computes computes local descriptions for the C-surfaces in $A$. Naturally, parts of these descriptions will change for different subgoals. The local characterizations of C-surfaces allow the planner to find the set of C-surfaces to which the goal direction is tangent (or orthogonal) as described above. When a planning direction is chosen, these C-surfaces clearly provide strong constraints.

We are now ready to discuss the experts themselves. The bumble strategy is

also applied at each node, since it is a guarantee of completeness. In light of the previous discussion, we will omit any discussion of the detection and pruning out of explorations in unfruitful directions (as determined by the planning history). We will consider the application of particular experts to a search node $s$ (at configuration $X$) which has parent $s_0$.

### 2.4.3. *The greedy expert*

The greedy expert attempts to translate or rotate directly towards the goal. The expert is necessary as an "end-game" strategy, in order to close in on a particular subgoal without worrying about finding the appropriate intersection manifold. The greedy expert illustrates two important heuristics: *forwarding* and *backing off*. Suppose the greedy expert translates from a parent node $s_0$ to a son $s$. An appropriate explanation for the move will be left at $s$. If the same subgoal is intact when the planner explores $s$, the greedy expert will not attempt translation again. Instead, the rotation expert (see Section 2.4.6) might be invoked. The effect is one of translating until an obstacle is hit, and then rotating to get around it. Alternatively, the sliding expert (which slides along level C-surfaces) might be invoked. This coupling of experts is termed the "hit and slide" strategy (see Fig. 14). However, the planner does not directly recurse by calling the sliding expert immediately after the greedy expert. Instead, a suggestion is left by way of explanation at $s$, and when $s$ is explored in the search, the appropriate follow-up expert is invoked. The exact choice for

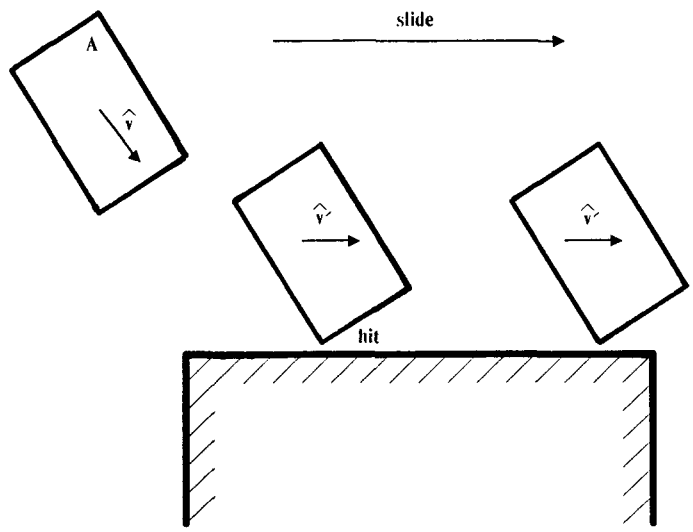


FIG. 14. An idealized illustration of the hit and slide strategy. Some expert moves the robot in direction $\hat{v}$ until a C-surface $S$ is hit at $X$. When the planner tries to move from $X$, the sliding expert is invoked to slide along $S$ in the goal direction.

which expert is invoked will depend on the history of planning (typically, what neighborhoods and directions have been explored from $s_0$ and $s$), and on the local geometry of C-surfaces about $s$.

Suppose that all experts moved the robot as far as they could, that is, moved until a constraint was hit and left the robot touching the constraint. This could result in jamming the robot up against many C-surfaces at once. It can prove very difficult to extricate the robot from this logjam situation. In fact, it is usually not preferable to move all the way up to an obstacle. Instead, we wish to detect this intersection with a planned trajectory $p$, and then *back off* from the obstacle boundary (along $p$). Thus if $p(0) = X$ and $p(1) = Y$ is the first intersection of $p$ with a C-space obstacle boundary, then it makes good sense to move to $p(0.8)$. This has the effect of leaving the robot in the channel between obstacles instead of jamming it up in corners. Of course, if it is necessary to move to $p(0.95)$, then the greedy and bumble strategies will ultimately converge.

2.4.4. *The intersection expert*

The mathematics of intersection manifolds in $\mathbb{R}^3 \times \mathrm{SO}(3)$ is sketched in the appendix. The intersection expert attempts to find two C-surfaces in $A$ whose intersection manifold contains a path which makes progress towards a subgoal. The path may be a pure translation or a pure rotation. We will begin by describing the process of finding a translational path which slides along an intersection manifold. First, all C-surfaces in $A$ which are nearly translationally tangent to the goal direction are selected. We select the first few of these which have the smallest value at $X$. Ideally, these are the closest nonredundant constraints at $X$. Call this set $A'$. The explanations for the moves from $s_0$ to $s$ and from $s$ to any sons of $s$ will yield a set of previously explored intersection manifolds. (An intersection manifold may be identified by the name of the intersected C-surfaces, their levels, and the chosen parameterization.) The C-surfaces in $A'$ are pairwise intersected [12], after appropriate pruning as indicated by previously explored intersection manifolds. Each intersection manifold $\ker f \cap \ker g$ is constructed. A translation or rotation vector $\hat{v}_{f,g}$ is chosen such that the path $p_{f,g}(t) = X + t\hat{v}_{f,g}$ slides along the intersection manifold of the two level C-surfaces $\ker f$ and $\ker g$ at $X$. The intersection expert then selects the direction $\hat{v}_{f,g}$ which is closest to the goal direction (and which is not pruned out by consideration of the planning history). Suppose $\hat{v}_{f,g}$ is a pure translation. The local operator Translate is called to move from $X$ in direction $\hat{v}_{f,g}$ until a C-surface is struck[15] or the point on the trajectory $p_{f,g}$ which maximizes proximity to the goal is reached.

Now, suppose $\hat{v}_{f,g}$ is a pure rotation. Our experimental implementations

[15] Although we also employ the backing off heuristic here.

have intersected two level C-surfaces ker $f$ and ker $g$ to yield pure rotational paths sliding along the intersection manifold of ker $f \cap$ ker $g$ (see [12] for the details). It is not hard to show that these paths may be approximated to an arbitrary resolution by successive applications of the local operators, with only a linear increase in the number of path segments as the resolution grows finer. We have also found it useful to approximate the rotational path along the intersection as follows.

Given two level C-surfaces ker $f$ and ker $g$ at configuration $X$, we wish to choose a direction from $X$ tangent to both. For example, if the configuration space were isomorphic to $\mathbb{R}^3$, then ker $f$ and ker $g$ would both be two-dimensional surfaces in 3-space, and this direction would be $\nabla f(X) \times \nabla g(X)$ (where $\times$ denotes the standard cross-product on $\mathbb{R}^3$). In the tangent space to a six-dimensional C-space, there is a four-dimensional subspace of tangent vectors to ker $f$ and ker $g$ at $X$. We will demonstrate an operator analogous to $\times$ which produces one such tangent vector in a natural way. (It is also possible to solve for all such tangent vectors.)

We begin by defining an extended product on the tangent space to $\mathbb{R}^3 \times \mathrm{SO}(3)$ at $X$. Let $V = (V_x, V_\Theta) \in T_X$ be a tangent vector at $X$. We may think of $V_x$ and $V_\Theta$ as the translational and rotational components of a six-dimensional velocity vector $V$ at $X$. If $W = (W_x, W_\Theta) \in T_X$ is another tangent vector at $X$, we define the extended product of $V$ and $W$ by

$$V \hat{\times} W = (V_x \times W_x, V_\Theta \times W_\Theta) .$$

The cross-products on the right-hand side are simply the standard three-dimensional cross-products. (See (2.3) for why this makes sense for the rotational components, $V_\Theta \times W_\Theta$.) If $V = \nabla f$ and $W = \nabla g$ then $V \hat{\times} W$ is tangent to both ker $f$ and ker $g$ at $X$. Since $\hat{\times}$ only operates on tangent vectors to $\mathbb{R}^3 \times \mathrm{SO}(3)$ which have the same point of application, we will never have reason to confuse it with $\times$, which can only be applied to three-dimensional tangent vectors.[16]

Let $f, g \in A'$ be C-functions generating the C-surfaces ker $f$ and ker $g$ at $X$. Observe that the tangent vector $\nabla f(X) \hat{\times} \nabla g(X)$ is tangent to both ker $f$ and ker $g$ at $X$. We can locally approximate a pure rotational trajectory sliding along the intersection of $f$ and $g$ by a path in direction

$$\pi_\Theta(\nabla f(X)) \times \pi_\Theta(\nabla g(X)) . \tag{2.3}$$

Note that this is well defined since

[16] The $\hat{\times}$ operator may be viewed as follows. SO(3) is a Lie group, and so there exists a canonical smooth identification of any tangent space with the tangent space at the identity, $T_e\mathrm{SO}(3)$. $T_e\mathrm{SO}(3)$ is the Lie algebra of SO(3), and is isomorphic to $\mathbb{R}^3$. The Lie algebra enjoys a commutator bracket operation, which in this case is simply the usual cross-product.

$$\pi_{\Theta}\left(\left(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}, \frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi}\right)\right) = \left(\frac{\partial f}{\partial \psi}, \frac{\partial f}{\partial \theta}, \frac{\partial f}{\partial \phi}\right).$$

The differential rotations from $X$ are isomorphic to a three-dimensional vector space, and hence the cross-product

$$\left(\frac{\partial f}{\partial \psi}(X), \frac{\partial f}{\partial \theta}(X), \frac{\partial f}{\partial \phi}(X)\right) \times \left(\frac{\partial g}{\partial \psi}(X), \frac{\partial g}{\partial \theta}(X), \frac{\partial g}{\partial \phi}(X)\right)$$

is also well defined, and guaranteed to be tangent to $\ker f$ and $\ker g$ at $X$. The Rotate operator can be called in succession on the largest components of (2.3) in order to approximate the sliding trajectory. Of course, it is also possible to reevaluate the tangents after each step.

Planning along intersection manifolds of three or more level C-surfaces is analogous but more complicated. See [12] for details.

2.4.5. *The sliding expert*

The sliding expert attempts to find a path sliding along one level C-surface at $X$, which makes progress towards the goal. The sliding expert can be thought of as a less constrained version of the intersection expert. The sliding expert tries to choose a C-surface in $A'$ to which the goal direction is (almost) tangent. As Donald [12] shows, it is possible to choose a parameterization along a C-surface which maximizes progress. This path along the C-surface can then be realized (at a desired resolution) by successive applications of the local operators. However since there are many paths from $X$ sliding along a C-surface at $X$, we need to develop a good heuristic strategy.

Our motivation is as follows. There are an uncountable number of paths from $X$ sliding along a C-surface at $X$. We could maximize a directional derivative at $X$ to choose a locally optimal search direction. This would work once; however, this would not solve the problem of state: it is necessary to partition the set of paths into "neighborhoods," and to mark a neighborhood of paths as explored when a representative from that neighborhood is selected and attempted by a local operator. In principle, this suggests a computation involving homotopic equivalence classes [11, 12]. However, this requires a global computation in C-space. In particular, the image of all paths in an equivalence class may cover $\mathbb{R}^3 \times \mathrm{SO}(3)$, even if there are several classes. We wish to find a way to partition the paths from $X$ into neighborhoods, sample a canonical element from the neighborhood, and evaluate it as a local move in the search.

Given a C-surface normal $\nabla f$ at $X$, we wish to choose a direction $\hat{v}$ sliding along the C-surface $\ker f$ which maximizes progress to a subgoal. Let $\mathscr{B} = (\hat{x}, \hat{y}, \hat{z}, \hat{\psi}, \hat{\theta}, \hat{\phi})$ be the obvious orthonormal basis for the tangent space to $\mathbb{R}^3 \times \mathrm{SO}(3)$, and $-\mathscr{B} = (-\hat{x}, -\hat{y}, -\hat{z}, -\hat{\psi}, -\hat{\theta}, -\hat{\phi})$.

Next, we form a set of vectors orthogonal to $\nabla f(X)$ as follows:

$$D = \{\nabla f(X)\} \otimes (\mathcal{B} \cup -\mathcal{B}),$$

where $\mathcal{P} \otimes \mathcal{Q} = \{p \hat{\times} q \mid p \in \mathcal{P}, q \in \mathcal{Q}\}$. All of these vectors are tangent to $\ker f$ at $X$. We then choose the direction $\hat{v} \in D$ which maximizes $\Phi_X(\hat{v}, (G - X))$, where the $G - X$ is the goal direction. If $\Phi_X$ is the heuristic product on tangent vectors instead of the single-valued Riemannian inner product, then both components of the image of $\Phi_X$ should be maximized.

To understand this strategy, consider the following example: Suppose we employ a basis $\mathcal{B}'$ which only spans $\mathbb{R}^3$. Then the expert will choose the available translation sliding along the level C-surface which maximizes progress towards the goal. Once the direction $\hat{v}$ is chosen, the Translate operator is invoked to slide along the level C-surface until a constraint is reached.

There is no need for the basis $\mathcal{B}$ to be orthogonal; this was merely adopted for the sake of intuitive development. The basis provides a sampling of the function space of paths tangent to the C-surface about $X$.

*A conjecture on completeness using extended spanning sets*

By using the basis $\mathcal{B}$, we obtain a 12-way sampling of the space of directions orthogonal to $\nabla f$ at $X$—in other words, there are 12 vectors in $D$. Imagine using another set of vectors, $\mathcal{B}^+$, which is larger than $\mathcal{B}$, to construct $D$. Then $D$ would provide a *finer* sample of the space of directions, since more directions would be sampled. In principle it should be possible for a sample to be complete at a given resolution. We formalize this idea as follows:

A *spanning set* for a space $V$ is a set of vectors which spans $V$ yet which is not necessarily a basis. A spanning set is a basis for $V$ which has been extended by adding other vectors. We conjecture that there exist certain spanning sets which might be employed to construct a complete planning algorithm *without* the bumble strategy. What constitutes such a complete spanning set? The analogue of resolution for an arbitrary spanning set $\mathcal{B}^+$ would consist in (1) the cardinality of the spanning set and (2) the uniformity of distribution of the vectors

$$\mathcal{B}^+ \cup -\mathcal{B}^+$$

about the unit five-dimensional sphere $S^5$ in the tangent space at $X$. The greater the number of vectors in the spanning set, and the more uniform their distribution about $S^5$, the finer the resolution of the planner. The development of such a planning algorithm requires surmounting additional theoretical and technical difficulties.

### 2.4.6. *The rotation expert*

The rotation expert is built on the rotational operator Rotate, and is designed to handle some of the special problems of moving through rotation space that

are discussed in the appendix. The rotation expert might be called to accomplish a simple rotational subgoal, or in conjunction with some more elaborate strategy. In particular, when a translational motion terminates by striking a C-surface, forwarding messages are left for both the sliding expert and the rotation expert. The former has been discussed as the "hit and slide" strategy (Fig. 14); the latter is known as the "hit and rotate" technique (Fig. 15).

The first problem that the rotation expert must deal with is the "wrap around" in rotation space. A subgoal $\phi_0$ can be reached in directions $+\hat{\phi}$ and $-\hat{\phi}$, although typically one is "shorter." In conjunction with the planning history, the rotation expert, on successive applications to the same node, can develop strategies for rocking back and forth on a slice of rotation space.

The Rotate operator is more constrained than the Translate operator (in that it can only be applied in $\pm\hat{\psi}$, $\pm\hat{\theta}$, and $\pm\hat{\phi}$). Hence the rotation expert must have a method for approximating rotational trajectories (specified in the angle space) which are linear combinations of the rotational basis vectors, such as

$$\hat{v} = a\hat{\psi} + b\hat{\theta} + c\hat{\phi} \tag{2.4}$$

for some scalars $a$, $b$, and $c$.

In terms of the completeness of the algorithm, there is no need for a Rotate operator in direction (2.4) (provided a path along $\hat{v}$ lies in open sets of free space). Donald [12] shows that a continuous path may be approximated as closely as desired by a sequence of moves along the rotational axes, and that the number of staggered path segments required grows only linearly as the resolution becomes finer. In practice this use of the restricted Rotate operator has proved adequate in our path-finding experiments. However, it is heuristically useful to realize such paths as accurately as desired, since this allows

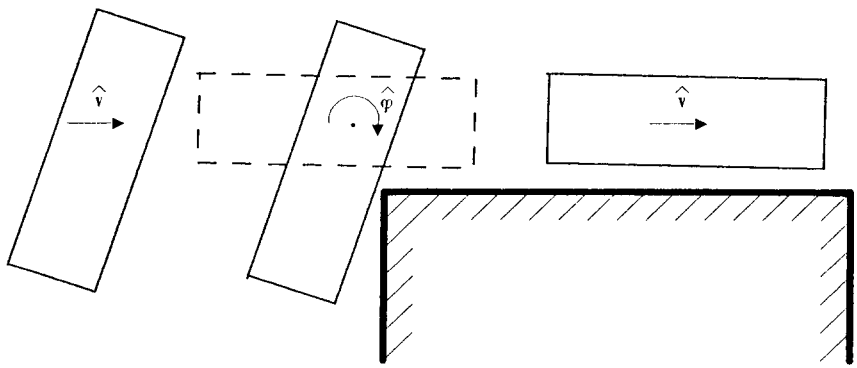


FIG. 15. An idealized illustration of the hit and rotate strategy. Some expert moves the robot in direction $\hat{v}$ until a C-surface $S$ is hit at configuration $X$. When the planner tries to plan a move from $X$, the rotation expert is called to calculate a rotation away from $S$ (in direction $\hat{\phi}$). From the new configuration, direction $\hat{v}$ can be pursued again.

higher-level experts to suggest arbitrary rotational trajectories. Given such a trajectory, the rotational directions are ranked by magnitude of change, and the unexplored direction of greatest change is first attempted. On failure, or upon successive applications of the rotation expert to the search node, the other directions in (2.4) will be attempted. This process leads to the approximation of arbitrary pure rotations by a staggered sequence of rotations along the axes. If the extent of each rotation is limited, the approximation can be made arbitrarily fine. To approximate motion in a direction such as (2.4), the planner actually attempts several of the directions simultaneously, which results in a spanning "box" of rotational moves about the idealized trajectory (in the absence of obstacles).

Suppose $a$, $b$, and $c$ in the idealized trajectory (2.4) are positive. This yields a set of positive, or "forward" rotational directions, and a set of "backwards" rotational directions which can attain the goal. Which directions are forward and which are backward depend upon the distance (in the vector parameter space $Q^3$) of the goal from $X$, that is, on $\pi_\Theta(G - X)$. For example, if $G_\phi - X_\phi$ is negative and small, then $+\hat{\phi}$ will be a backwards direction, and $-\hat{\phi}$ will be a forward direction.

The rotation expert develops and ranks these sets of forward and backward rotational directions. By examining the planning history and the local geometry of C-surfaces at $X$, these sets of directions are in turn pruned. In particular, local C-surfaces that would block a particular rotational motion are detected. For a direction $\hat{v}$, this is done by examining the magnitude of the directional derivative in $\hat{v}$. The importance of such an impediment is then heuristically ranked by the closeness of the C-surface at $X$. Special consideration is given to C-surfaces which have a history of proving troublesome. For example, when an expert runs into a C-surface, the reason for stopping is left as part of the move explanation. If the rotation expert is invoked as part of a "hit and rotate" strategy, then we must ensure that the planner tries to rotate away from the C-surface(s) which blocked progress. The rotational directions which point away from C-surfaces may be found by examining $\nabla f$. The process of determining the rotational constraints from the local geometry of C-surfaces is closely related to our earlier discussion of detecting rotationally orthogonal C-surfaces.

Thus the requested rotational trajectory and rotational goal provide a set of desired rotational motions. The planning history supplies a set of rotational constraints, and from the local C-surface geometry can be inferred a set of preferred and prohibited motions. The constraints, preferences, and prohibitions are intersected with the forward and backward desires. This yields a set of rotational directions which will be attempted using the Rotate operator. Depending on the kind of invocation, the rotation expert may apply the Rotate operator up to some fixed number of times—this is particularly useful when it must attempt to approximate an idealized rotational trajectory which is a linear combination of the basic rotational directions.

Canny [8] has recently extended the Rotate operator for directions such as (2.4), corresponding to uniform rotation.

### 2.4.7. *The around expert*

The around expert attempts to circumnavigate obstacles by sliding around their boundary. An idealized illustration of the around expert is shown in Fig. 16. The around expert is similar to the sliding expert, except that instead of attempting to find a C-surface which contains a path *towards* the goal, the around expert searches for a C-surface which is (roughly) *locally orthogonal* to the goal direction. Next a path is planned sliding along this surface in the direction $\hat{v}'$ orthogonal to the goal direction; the path is attempted using a local operator. Typically, this motion will result in a search node $s'$ which is farther from the goal than the parent node, $s$. Ordinarily, $s'$ would not be explored soon, since other search nodes would appear more promising to the planner's best-first strategy. In order to give the around strategy a chance, the around expert explicitly places $s'$ at the front of the search queue and calls the planner recursively.

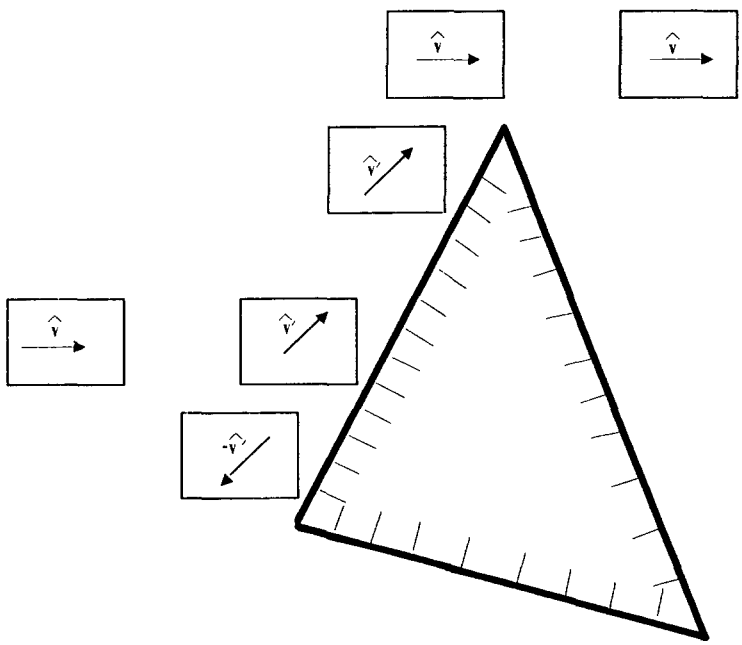

FIG. 16. An idealized illustration of the around expert. When progress for the moving object in the goal direction $\hat{v}$ is blocked, the expert attempts to find a C-surface which is roughly orthogonal to $\hat{v}$. A sliding motion (either $\hat{v}'$ or $-\hat{v}'$) is then planned along this level C-surface (around the obstacle). The resulting search node is then expanded.

The around expert can also invoke the intersection expert. Recall that the intersection expert ordinarily tries to construct tangent intersection manifolds which contain paths towards the goal. However, when called from the around strategy, it can construct intersection manifolds locally orthogonal to the goal direction. To construct the intersection set of locally orthogonal level C-surfaces, we perform a pairwise intersection of C-surfaces locally orthogonal to the goal direction at $X$.

#### 2.4.8. *The suggestor*

The suggestor is a strategy for proposing good subgoals in configuration space. As we saw in [11], one of the problems with local operators even if they are complete (that is, their closure covers configuration space), is that without good subgoals, they may take a long time to converge. The suggestor is a heuristic strategy for setting subgoals in C-space.

First, a very coarse lattice is thrown over C-space. This lattice is then searched for a *sequence J* of free configurations (*not* a path) stepping through the lattice to the goal. If no such sequence can be found, then configurations on a promising partial sequence are employed. These configurations may then be set as subgoals, and the planner can be called recursively. The configurations $J$ represent intermediate planning islands of safe configurations. If paths can be found between these configurations, then the find-path problem is solved. Otherwise, expanding from any partial paths found can also prove useful, in that the planning islands effectively distribute the application of local experts and operators over more of configuration space.

The suggestor complicates the connectivity of the explored neighborhoods graph. The ability to explore arbitrary subgoals and suggested paths requires more complicated bookkeeping for neighborhood exploration: we must employ the *connect strategy*, in order to know when partial paths link up. If partial paths not rooted at the start neighborhood are permitted, then the graph of explored neighborhoods will not necessarily be connected, and the mark strategy will fail (the mark strategy constructs a directed, spanning tree for a connected, rooted graph of explored neighborhoods). Happily the connect strategy will succeed, since it is defined on an arbitrary graph. An algorithm for the connect strategy is discussed in Section 2.3.2.

### 2.5. Examples of the local experts in use

In Fig. 17, we show a very simple example of a path found using local experts. Listing I (Fig. 18) shows a log of the expert explanations for each move.

The "Thor's Hammer" example in Section 1 was produced by disabling all experts, and employing only the bumble strategy (see Figs. 2 and 3). In the accompanying figures (Figs. 19–24), we show a path found by a strategy comprising all the experts described above. The solution path is very different,

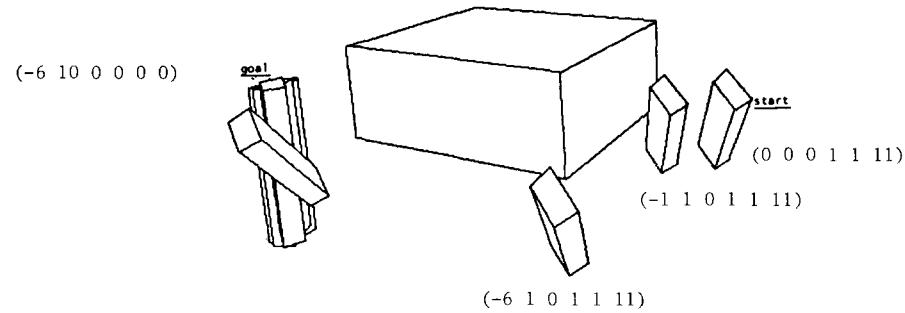


FIG. 17. A path which was found using local experts. This find-path problem is very easy (it is used as an example in Section 1). The lattice points of configurations along the solution path are as in listing I, Fig. 18.

```
(find-path *s1 *g1)
  Verifying the start and goal points...
  start: (0 0 0 1 1 11), goal: (-6 10 0 0 0 0).
Starting search, boss...
Exploring (0 0 0 1 1 11)...
Local expert: I translated straight towards goal, reaching ((-1 1 0 1 1 11))
Exploring (-1 1 0 1 1 11)...
Local expert: I slid along a level C-surface, reaching ((-6 1 0 1 1 11))
Exploring (-6 1 0 1 1 11)...
Local expert: I translated straight towards goal, reaching ((-6 10 0 1 1 11))
Exploring (-6 10 0 1 1 11)...
Rotation expert: Found 0 guiding constraints on rotational motion.
Rotation expert: Intersected rotational constraints with desired
                 rotations yielding possible motions in
                 (MINUS PHI) (MINUS PSI) THETA).
Rotation expert: I am trying to rotate in (PLUS THETA)...
Local expert: I rotated to reach ((-6 10 0 1 1 0))
Exploring (-6 10 0 1 1 0 )...
Rotation expert: Found 0 guiding constraints on rotational motion.
Rotation expert: Intersected rotational constraints with desired
                 rotations yielding possible motions in
                 ((MINUS PHI) (MINUS PSI)).
Rotation expert: I am trying to rotate in (MINUS PHI)...
Local expert: I rotated to reach ((-6 10 0 0 1 0))
Exploring (-6 10 0 0 1 0)...
Rotation expert: Found 0 guiding constraints on rotational motion.
Rotation expert: Intersected rotational constraints with desired
                 rotations yielding possible motions in
                 ((MINUS PSI)).
Rotation expert: I am trying to rotate in (MINUS PSI)...
Local expert: I rotated to reach ((-6 10 0 0 0 0))
Exploring (-6 10 0 0 0 0)...
  [success!] Saving and drawing final path...
Back to Lisp Top Level in Lisp Listener 2
```

FIG. 18. Listing I: The log of expert explanations for the path in Fig. 17.

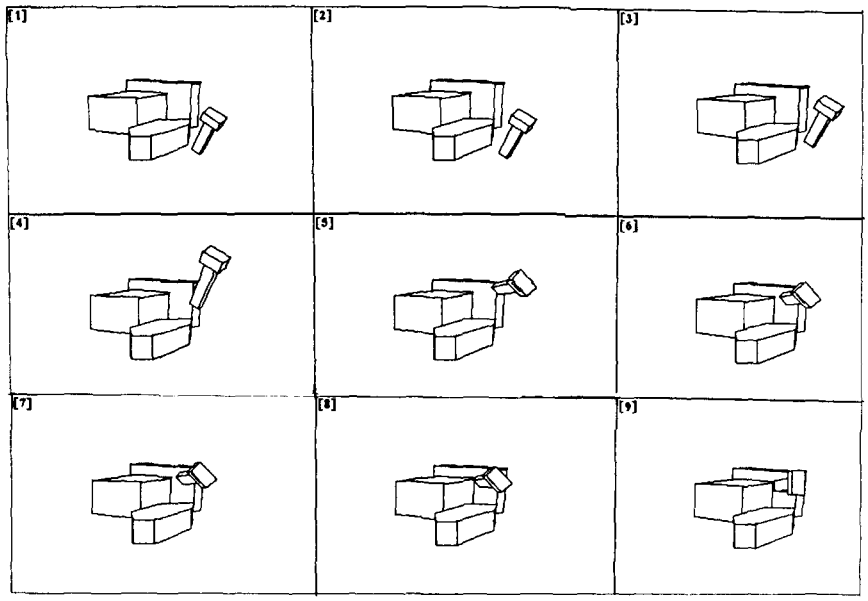


FIG. 19. View I: frames 1–9. These 18 frames show a solution path for the "Thor's Hammer" Movers' problem. Local experts (as described in Section 2) are employed to slide the moving object along level C-surfaces. Three views are shown. The final configuration is only visible in view II (Fig. 23).

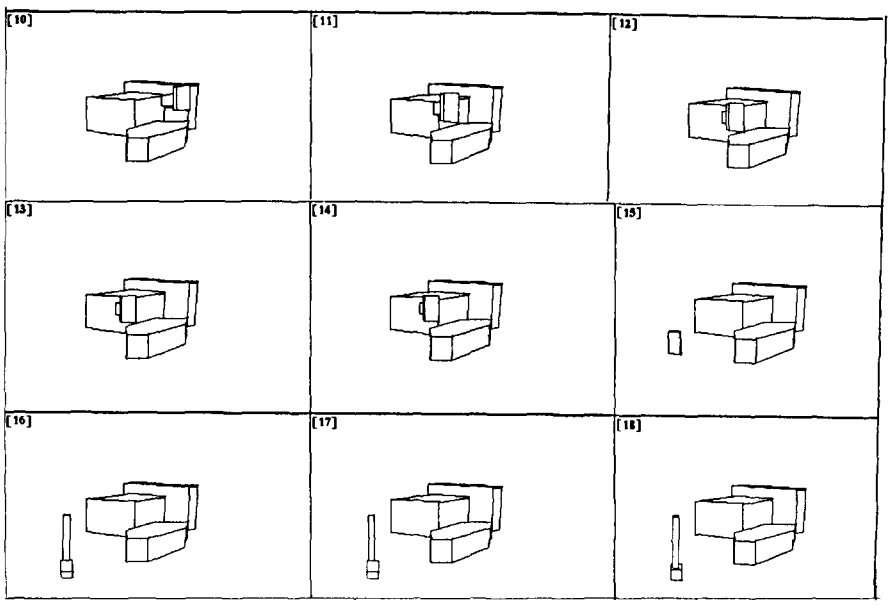


FIG. 20. View I: frames 10–18.

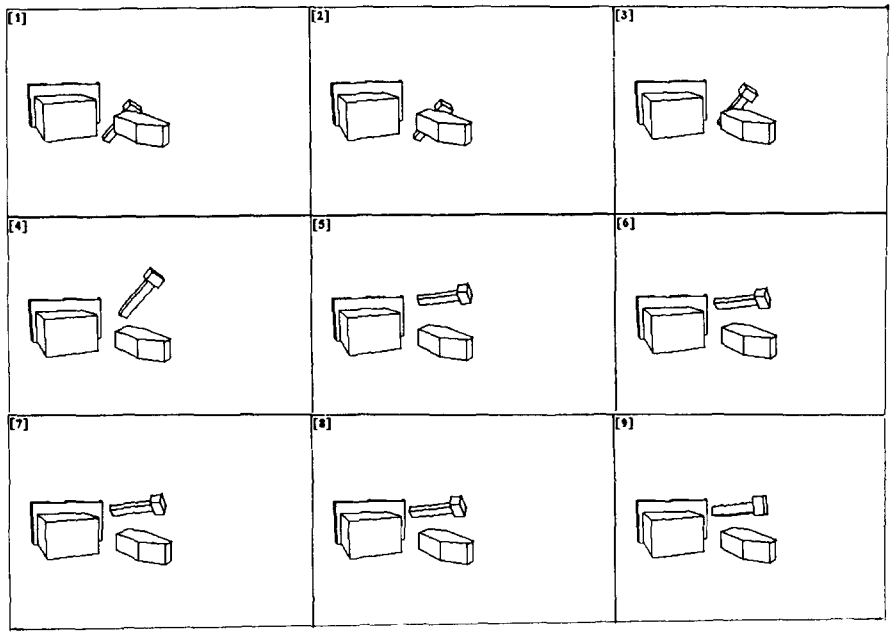


FIG. 21. View II: frames 1–9 of the Thor's Hammer example using local experts.

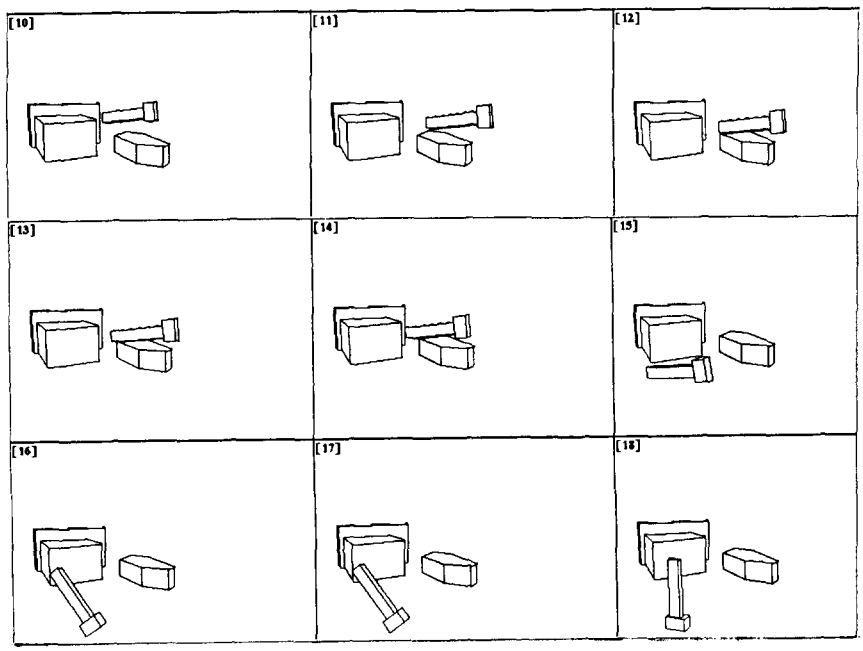


FIG. 22. View II: frames 10–18.

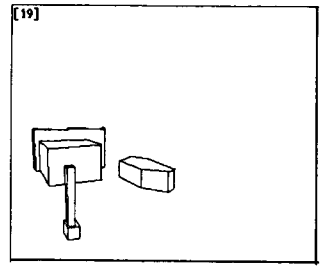


FIG. 23. View II: frame 19. The final configuration

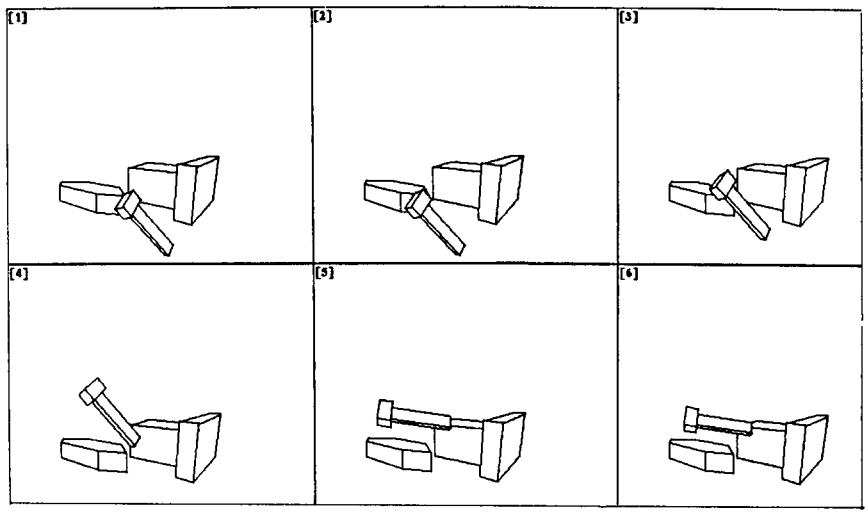


FIG. 24. View III: A detail of frames 1–6.

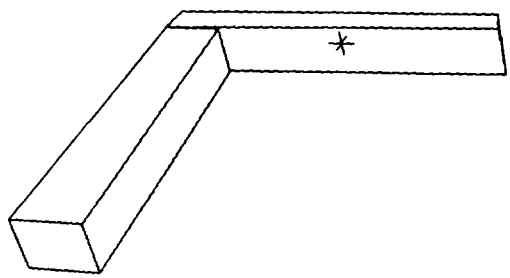

FIG. 25. The reference point on the L-shaped object.

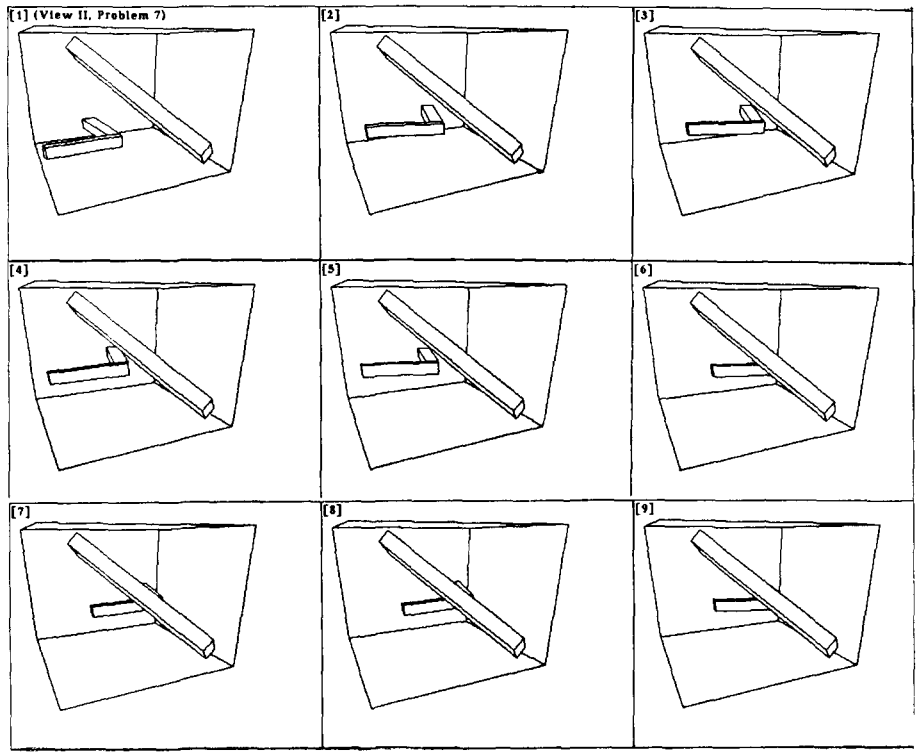


FIG. 26. Solution path. View II: frames 1–9.

and tends to slide around obstacles instead of finding convoluted paths between them.

Figures 25–32 show the solution for a find-path problem in a Cartesian workspace. A *Cartesian workspace* is a bounding box beyond which the reference point may not translate. However, the bounding box imposes no restrictions on rotations. The Movers' problem in a Cartesian workspace is similar to the motion planning problem for Cartesian manipulators, and the L-shaped object may be thought of as the (wrist and) payload. First, we show the reference point on the L-shaped object. Next two views are presented of the path found within the workspace, around a large, diagonally-placed obstacle. View II is a view from the side; view I is a view from the top. Only the back faces of the rectangloid workspace are shown. Since the rotation from frames 13 to 14 is very large ( $> \pi$ in the $-\hat{\psi}$ direction), a detail of the rotation is also shown.

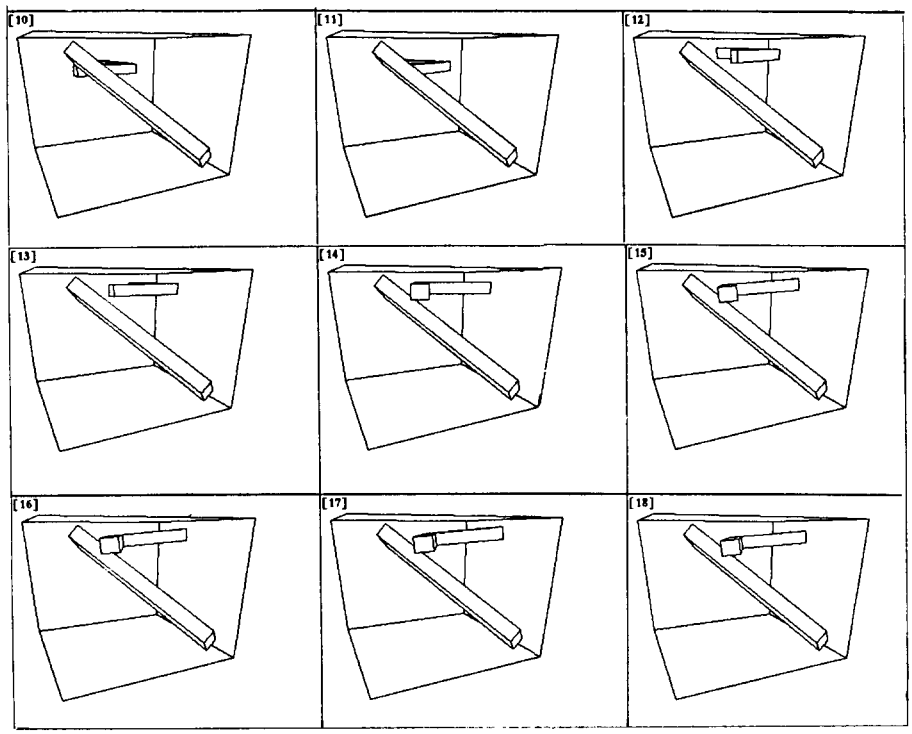


FIG. 27. Solution path. View II: frames 10–18.

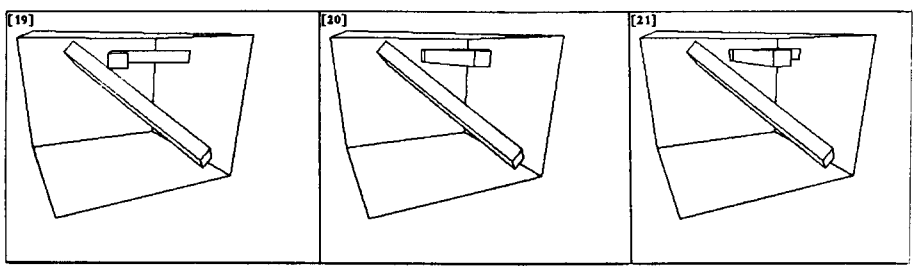


FIG. 28. Solution path. View II: frames 19–21.

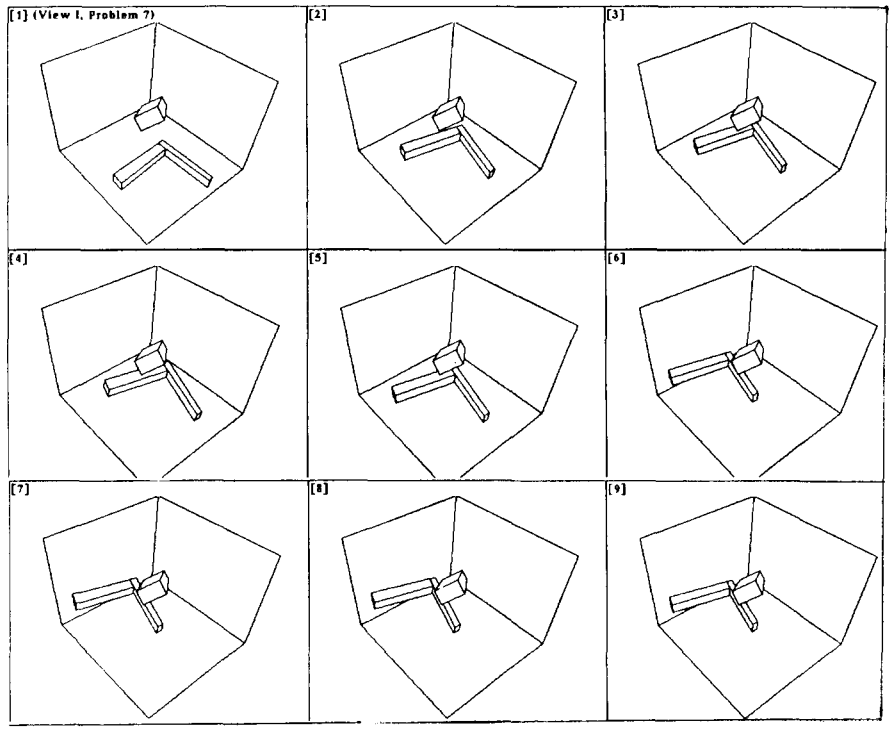


FIG. 29. Solution path. View I: frames 1–9.

### 2.6. Complexity

The search algorithm is polynomial in the size of the lattice. In general, one envisions a planner which employs a finer resolution for more complicated environments. It appears that the number of lattice points would have to grow exponentially with the environmental complexity; thus, of course, our algorithm is not polynomial in the size of the input.

### 2.7. Implementation

The planner was implemented on a Symbolics 3600. The running times for the examples in Figs. 2, 3, 5–8, and 26–31 were on the order of several hours. The examples in Figs. 19–24 took about 20 minutes. The example in Fig. 17 took under two minutes.

## 3. Conclusions

In this paper we developed a search algorithm for find-path problems with six degrees of freedom. The algorithm is based on the representations and

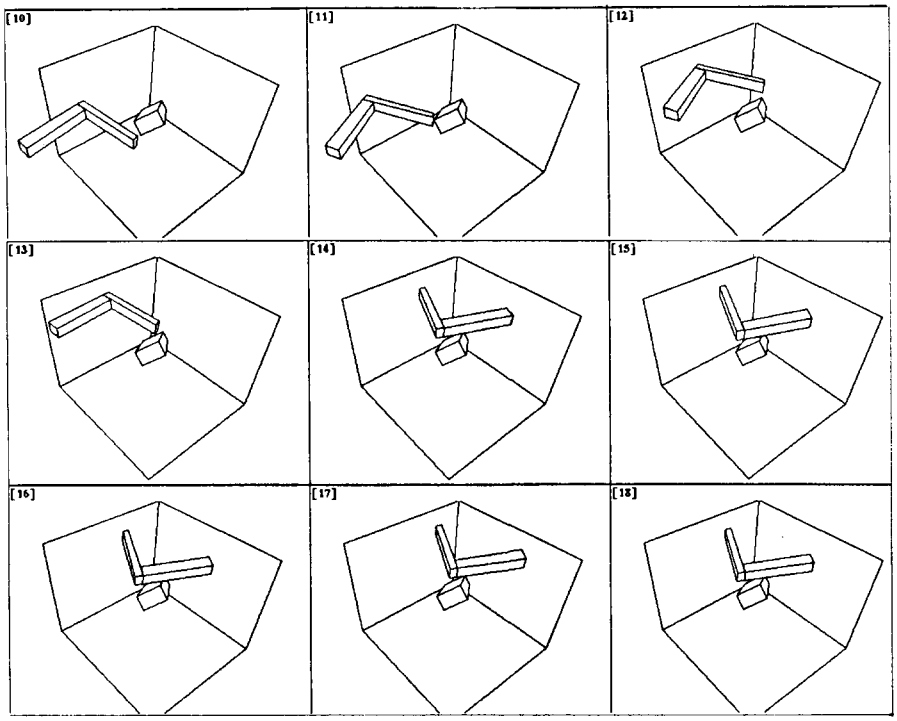


FIG. 30. Solution path. View I: frames 10–18.

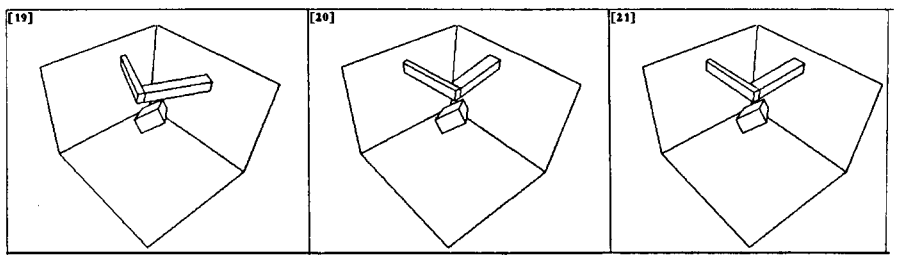


FIG. 31. Solution path. View I: frames 19–21.

mathematics developed by Donald [12]. To demonstrate the competence of the representations and the feasibility of the algorithm, a planning system for the classical find-path problem with six degrees of freedom was implemented. The planner is of considerable intrinsic interest, in that it is complete (for a given resolution). Experiments have demonstrated that this algorithm can solve find-path problems requiring six degree of freedom solutions that were beyond the competence of earlier, approximate planners. We believe this argues that

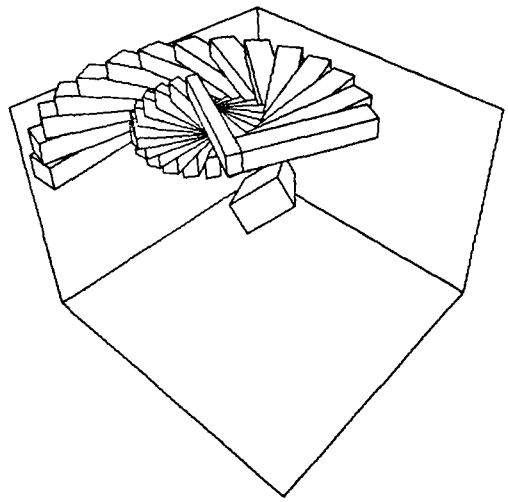

FIG. 32. Detail of the rotation from frames 13–14

the mathematical framework of Donald [12] impacts a class of geometric planning problems for three-dimensional objects.

The planning algorithm may be explained by analogy with the point navigation operators. The C-space transformation reduced the motion planning problem to the task of navigating a point in $\mathbb{R}^3 \times SO(3)$. Since the path for the point must avoid the C-space obstacles, which are curved, six-dimensional manifolds with boundary, clearly paths can be found in C-space by the closure of three operators:

(i) slides along one- to four-dimensional intersections of level C-surfaces;
(ii) slides along five-dimensional level C-surfaces;
(iii) jumps between six-dimensional obstacles.

While these operators define a complete search space, it is clearly too big, in general, to be searched in its entirety. Therefore we developed local experts to guide and to limit the search. The algorithm is a best-first search employing a set of heuristic strategies that evaluate local geometric information. Each strategy plans motions that are realized using the point navigation operators. At the heart of this research lie the design and implementation of strategies for planning paths along level C-surfaces and their intersection manifolds, and for reasoning about motions with three degrees of rotational freedom. The problems of controlling the interaction of these strategies, and of integrating diverse local experts for geometric reasoning provide an interesting application of search to a difficult domain with significant practical implications.

There is much work to be done. The find-path algorithm can be easily extended to robot manipulators with six degrees of freedom in which translations can be decoupled from rotations. This class includes Cartesian man-

ipulators (for example, the IBM RS/1). The adaptation of this work to a production environment presents interesting engineering challenges.

In principle, our algorithms and representations can be extended to revolute-joint, linked arms with six degrees of freedom. However, the C-space of the linked-arm problem is the six-dimensional torus,

$$S^1 \times \cdots \times S^1 \quad \text{(to 6)}$$

which has a very different structure from $\mathbb{R}^3 \times \mathrm{SO}(3)$. It is our hope that this research can present a methodology for formulating the geometric constraints for arbitrary configuration spaces, and that a similar structure will be found for constraints on the 6-torus. See [26] for work in this direction.

Our planning algorithm is complete (at a resolution), in that the representation employed is complete, and in that the search is guaranteed to find a path if one exists at that resolution. However, since it is a search algorithm, we cannot provide a polynomial time bound. (Although clearly the algorithm is polynomial in the resolution.) Our motivation has been to address the completeness issue first, by resolving fundamental representational questions; now, one of the most important remaining tasks is to develop complete, polynomial-time algorithms which can actually be implemented. We do not believe that the worst-case behavior of the six degree of freedom planner is inherent in the representation, and conjecture that a polynomial-time algorithm which plans paths along intersection manifolds can be devised. Indeed, the theoretical results on the C-Voronoi diagram (CVD) [12] are suggestive that the limiting complexity of the approach may be the complexity of constructing the CVD or an equivalent chain of intersection manifolds. More research is needed on the topology of the CVD. A fast planning system might determine what constraints construct the CVD, and, using these constraints, construct a chain of intersection manifolds which could attain the goal. The first step in this effort would bound the complexity of the CVD and the intersection chains. In this vein, Canny [9] has suggested an algorithm for computing the CVD.

## Appendix A. Representing Constraints in the Configuration Space

C-functions model constraints on motion generated by pairs of cells $(g_a, g_b)$ where $g_a$ and $g_b$ are boundary cells on the robot and on an obstacle, respectively. The constraints are defined such that their conjunction enforces a disjointness criterion for $A$ and $B$. Lozano-Pérez [24] identified three types of interactions: (face, vertex), (vertex, face), and (edge, edge), which to preserve tradition we shall term type (a), (b) and (c) constraints. However, these interactions can only occur in certain orientations; for example, it is easily seen that although two cuboids generate 144 type-(c) constraints, at any fixed

orientation only certain edges can interact and hence only certain type-(c) constraints are applicable. The region of rotation space where a C-function $f_i$, is applicable is its applicability region, $\mathscr{A}_i \subset \mathrm{SO}(3)$. The domain of the partial function $f_i$, then, is $\mathbb{R}^3 \times \mathscr{A}_i$.

For the two-dimensional Movers' problem, the rotation space is the 1-sphere and the applicability regions $\mathscr{A}_i$ are simply sectors on $S^1$. While Lozano-Pérez, in [24] was able to define the form of C-space constraints $f_i$, Donald, in [12], first formulated the applicability regions in SO(3).

We begin by defining $\mathrm{CO} \subseteq \mathbb{R}^3 \times \mathrm{SO}(3)$, the space of forbidden configurations:

$$\mathrm{CO} = \{X \mid \bigvee_a C_a(X)\}\,, \tag{A.1}$$

where $C_a$ is a *constraint sentence* [6]. $a$ is indexed by C-space obstacles. For each C-space obstacle $O_a$, $C_a$ maps a configuration $X$ to *true* or *false*, depending on whether $X$ is inside $O_a$. Equation (A.1) states that if $X$ is inside any C-space obstacle, then it is in CO. For $X = (x, \Theta)$,

$$C_a(x, \Theta) = \bigwedge_i (\Theta \in \mathscr{A}_i \Rightarrow f_i(x, \Theta) \leq 0)\,. \tag{A.2}$$

Let us parse (A.2). For a configuration $X$, for each C-function $f_i$ such that $X$ is in the domain of $f_i$, $f_i(X)$ must be negative-valued (or zero) for $X$ to be inside the C-space obstacle $O_a$. To determine whether $X$ is in the domain of $f_i$, test whether the rotational component of $X$ is within the applicability region $\mathscr{A}_i$. The index $i$ in (A.2) ranges over the set of all C-functions $\{f_1, \ldots, f_n\}$ which define the C-space obstacle $O_a$. We call such a set of C-functions a *family* of C-functions. This family is generated by considering pairwise interactions of features on the boundary of $A$ and features on the boundary of $B$, where $A$ is a convex polyhedron on the moving object, and $B$ is a convex obstacle polyhedron. In three dimensions, a family of C-functions corresponds to a set of constraints resulting from the possible interactions of one polyhedral component of the moving object, and one obstacle polyhedron, whence

$$\begin{aligned}\mathrm{family}(A, B) = {} & (\mathrm{faces}(A) \times \mathrm{vert}(B)) \cup (\mathrm{vert}(A) \times \mathrm{faces}(B)) \\ & \cup (\mathrm{edges}(A) \times \mathrm{edges}(B))\,.\end{aligned}$$

Of course, in both two and three dimensions, at a given orientation, only a subset of this family is applicable.

Next, we define $F = (\mathbb{R}^3 \times \mathrm{SO}(3)) - \mathrm{CO}$ to be the space of free configurations. We construct $\mathscr{A}_i$ as the intersection of a set of half-hyperspaces on SO(3):

$$\mathscr{A}_i = \{\Theta \in \mathrm{SO}(3) \mid \bigwedge_j (g_j(\Theta) \geq 0)\}\,, \tag{A.3}$$

where $g_j : \mathrm{SO}(3) \to \mathbb{R}$ is an *applicability constraint function* (ACF). A C-function $f_i$ is said to be applicable for a configuration $X = (x, \Theta)$ if $\Theta \in \mathscr{A}_i$. In this section, we will summarize the form of the ACFs. Geometrically, the applicability regions $\mathscr{A}_i$ are three-dimensional manifolds (with boundary) on the projective 3-sphere. Their boundaries comprise the two-dimensional manifolds $\mathscr{A}_i \cap \ker g_j$. ($j$ indexes over the set of functions used to construct $\mathscr{A}_i$. There are typically three or four $g_j$, as we will see later.) We call $\ker g_j$ an *ACF boundary*.

For a C-function $f_i$ we define a *level* C-surface to be the set of configurations $X$ where $f_i$ is applicable and $f_i(X) = l$, for some level $l$. Thus a level C-surface is the level set $f_i^{-1}(l)$. Of particular interest is the C-surface

$$\ker f_i = f_i^{-1}(0) = \{X \mid f_i(X) = 0\} ,$$

which contains a boundary patch of a C-space obstacle. An intersection manifold of $m$ level C-surfaces therefore has the general form

$$\bigcap_{i=1}^{m} (f_i^{-1}(l_i) \cap (\mathbb{R}^3 \times \mathscr{A}_i)) .$$

We now define paths in C-space. Given a start configuration $s$ and a desired goal configuration $g$, a successful collision-free path is an embedding $p : I^1 \to \mathbb{R}^3 \times \mathrm{SO}(3)$ such that $p(0) = s$, $p(1) = g$, and $p(I^1) \subset F$. $I^1$ denotes the closed unit interval, $[0, 1]$.

### A.1. The geometric interpretation for C-functions

Let $P$ denote any rigid, convex set, or the normal to a face of a polyhedron in $\mathbb{R}^3$. In our case, $P$ will be a polyhedron, a face, edge or vertex of a polyhedron, or the normal to a face. If $\Theta$ is an orientation, and $\mathscr{R}(\Theta)$ is the corresponding rotation operator, then $P(\Theta)$ denotes $\mathscr{R}(\Theta)$ applied to $P$. Note that the results of this section are independent of any particular representation chosen for rotations. Consider the interaction of an obstacle polyhedron $B$ and a moving polyhedron $A$, where both $A$ and $B$ are convex, with outward normals. Let $f_p$ be in the family of C-functions generated for $A$ and $B$. $f_p$ models a constraint on the motion of $A$. For example, $f_p$ might be generated by considering the interaction of a face of $A$ and a vertex of $B$. For a given orientation $\Theta$, the projection into $\mathbb{R}^3$ of any (applicable) C-surface $f_p^{-1}(0)$ is a plane corresponding to a face of the polyhedron resulting from the Minkowski sum of $B$ and $\ominus A = \{-a \mid a \in A\}$, that is,

$$B \ominus A(\Theta) = \{b + a(\Theta) \mid b \in B, a \in \ominus A\} .$$

$B \ominus A(\Theta)$ is the projection into $\mathbb{R}^3$ of the C-space obstacle at orientation $\Theta$. In

effect, we have parameterized the plane equations of faces of $B \ominus A(\Theta)$ by $\Theta$. Here is the form of the parameterized plane equations derived by Lozano-Pérez [24]: $a_i(\Theta)$ is a vertex of $\ominus A(\Theta)$ and $b_j$ is a vertex of $B$. Then C-functions take the form:

$$f_p(x, \Theta) = \langle N(\Theta), x \rangle - \langle N(\Theta), (a_i(\Theta) + b_j) \rangle \,, \tag{A.4}$$

where $x$ is a point in $\mathbb{R}^3$ and $\langle \cdot, \cdot \rangle$ denotes the inner product. $N(\Theta)$ is the real-space component of the C-surface normal at orientation $\Theta$, and is defined as follows: for a type-(a) C-function, $N(\Theta)$ is the normal of a face of $\ominus A(\Theta)$. For a type-(b) C-function, $N(\Theta)$ is the normal of a face on $B$, and hence is constant. For a type-(c) C-function, $N(\Theta)$ is the cross-product of an edge on $B$ and an edge on $\ominus A(\Theta)$.

### A.2. Applicability constraints

To define the applicability constraints, we consider a family of C-functions in isolation (that is, an environment comprising only the obstacle $B$ and the moving polyhedron $A$). We perform an analysis to see what generators can interact at what orientations. While C-functions are defined on $\ominus A(\Theta)$, applicability constraints are defined from $A(\Theta)$.

**Definition.** Consider a constraint $c$, generated by $(g_a, g_b)$ where the pair $(g_a, g_b)$ is either (a) a face of $A$ and a vertex of $B$, (b) a vertex of $A$ and a face of $B$, or (c) an edge of $A$ and an edge of $B$. We say $c$ is *applicable* at orientation $\Theta$ if some pure translation of $A(\Theta)$ can bring $g_a(\Theta)$ in contact with $g_b$, such that the interiors of $A(\Theta)$ and $B$ remain disjoint.

Suppose constraint $c$ is applicable and its generators are placed in contact. The convexity of $A(\Theta)$ and $B$ implies the existence of a separating plane $P$ between their interiors. Obtaining the equation of this plane, when it is unique, provides necessary and sufficient conditions for applicability. The applicability constraints ensure that the vertices adjacent to $g_a(\Theta)$ on the edge graph of $A(\Theta)$ lie on one side of $P$, and that the vertices adjacent to $g_b$ on the edge graph of $B$ lie on the other side of $P$. The generators themselves may lie on $P$.

Let $f(\Theta)$ be a face on a moving polyhedron $A(\Theta)$, with a normal $N(\Theta)$. Let $b_j$ be a vertex on obstacle $B$. $(f, b_j)$ generates a type-(a) constraint. The plane containing $f(\Theta)$ must be our separating plane, when the constraint is applicable. Let $R$ be the set of adjacent vertices to $b_j$ on the edge graph of $B$. Then the type-(a) constraint is applicable at orientation $\Theta$ if, and only if, for all $b_n \in R$,

$$b_n \cdot N(\Theta) - b_j \cdot N(\Theta) \geqslant 0 \,. \tag{A.5}$$

Now, let $f$ be a face of $B$ with normal $N$. Let $a_i$ be a vertex of $A$, so $(a_i, f)$

generates a type-(b) constraint. Then the plane containing $f$ will be our separating plane when the constraint is applicable. Let $R$ be the vertices adjacent to $a_i$ on the edge graph of $A$. Then the type-(b) constraint is applicable at orientation $\Theta$ if, and only if, for all $a_n \in R$,

$$a_n(\Theta) \cdot N - a_i(\Theta) \cdot N \geq 0 . \tag{A.6}$$

Consider

$$g_k(\Theta) = b_n \cdot N(\Theta) - b_j \cdot N(\Theta) \tag{A.7}$$

as a mapping $g_k : \mathrm{SO}(3) \rightarrow \mathbb{R}$. We call $g_k$ a type-(a) applicability constraint function (ACF). (There are several ACFs for one type-(a) C-function or indeed for any C-function, and they are indexed here by $k$.) For the symmetric case from (A.6), we call

$$g_k(\Theta) = a_n(\Theta) \cdot N - a_i(\Theta) \cdot N \tag{A.8}$$

a type-(b) ACF. The region on SO(3) where $g_k$ is positive-valued defines a half-hyperspace of SO(3). Equations (A.6) and (A.5) define the applicability region for a type-(a) or type-(b) constraint as the intersection of these half-hyperspaces. This yields the conjunction promised earlier in (A.3). The number of ACFs for a type-(a) or type-(b) constraint is equal to the cardinality of the coboundary of the generating vertex (i.e., $|R|$).

A type-(c) constraint is generated by $(e_a, e_b)$, where $e_a$ and $e_b$ are edges of $A$ and $B$ (resp.) The separating plane (when the constraint is applicable) has the normal $N_P(\Theta) = e_a(\Theta) \times e_b$, that is, the cross-product of the directed edge vectors. Now, let $N_1(\Theta)$, $N_2(\Theta)$ be the normals to the faces $e_a(\Theta)$ bounds. Let $T_1(\Theta)$, $T_2(\Theta)$ be the tangents to these faces, as shown in Fig. A.1. Let $N_3$, $N_4$ be the normals to the faces $e_b$ bounds. Let $T_3$, $T_4$ be the tangents to these faces, as shown in Fig. A.2. Then, the type-(c) constraint generated by $(e_a, e_b)$ is applicable at orientation $\Theta$ if, and only if,

$$\begin{aligned} &k_B = \mathrm{sign}(T_3 \cdot N_P(\Theta)) = \mathrm{sign}(T_4 \cdot N_P(\Theta)) , \\ &k_A = \mathrm{sign}(T_1(\Theta) \cdot N_P(\Theta)) = \mathrm{sign}(T_2(\Theta)) \cdot N_P(\Theta)) , \\ &k_A \neq k_B . \end{aligned} \tag{A.9}$$

Thus (A.9) defines analogous ACFs for type-(c) constraints. It is actually possible to define a less complicated, but weaker form of the type-(c) ACFs, which is necessary but not sufficient for applicability. The weak ACFs have the property that each is the scalar product of two type-(a) or two type-(b) ACFs. It is easier to find their zeros. Our implementation finds the zeros of the weak

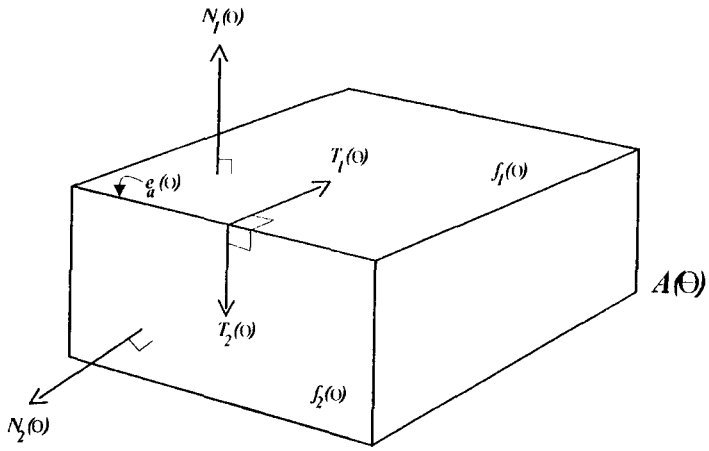


FIG. A.1. The tangent vectors $T_1(\Theta)$, $T_2(\Theta)$ and normals $N_1(\Theta)$, $N_2(\Theta)$ to the faces cobounding $e_a(\Theta)$. These vectors rotate with $e_a(\Theta)$.

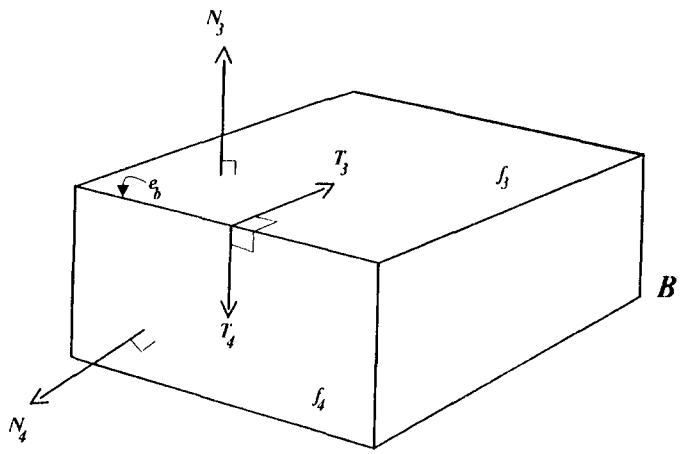


FIG. A.2. The tangent vectors $T_3$, $T_4$ and normals $N_3$, $N_4$ to the faces cobounding $e_b$. These vectors are fixed with $e_b$.

ACFs and then uses (A.9) to check for sufficiency.[17] For proofs and more detail, see [12].

## Appendix B. Mathematical Tools for Motion Planning in a Six-dimensional Configuration Space

We now discuss specific issues which are critical for the implementation of the point navigation operators. The fundamental topic is the *intersection problem*

[17] Thus (A.9) are called *disambiguating* applicability constraints in [12].

in high-dimensional configuration spaces:

(i) How do we intersect high-dimensional level C-surfaces to construct an intersection manifold?

(ii) How do we intersect a trajectory in configuration space with C-space constraints?

In $\mathbb{R}^3 \times SO(3)$, the equations for some constraints (notably, type-(c) constraints) can fill several pages. For this reason, I first computed the general form of the intersections for an arbitrary constraint, and then solved all intersections using MACSYMA [23]. The results were then optimized and compiled into LISP.

Our approach has been to (1) derive these constraints (and the ACFs) from some arbitrary representation for rotations, (2) reduce each constraint to a series of simpler, canonical forms which are affine, bilinear, or quadratic in the terms of interest, and (3) develop simple mathematical procedures for operating on the canonical forms. For example, to construct an intersection manifold for $n$ constraints, we essentially need to solve a set of $n$ simultaneous equations, each of the form $f(X) = 0$, where $X \in \mathbb{R}^3 \times SO(3)$. We proceed as follows. Let $D = \{x, y, z, \psi, \theta, \phi\}$ be the set of all the degrees of freedom. First we select $P$, a subset of $6 - n$ elements of $D$. $P$ will parameterize the intersection manifold. The variables in $P$ will be the free variables which the planner can choose; the variables $D - P$ will vary dependently with $P$ so as to stay on the intersection manifold. Mechanically, this entails (1) solving the $n$ constraints simultaneously eliminating all but one variable in $D - P$, and (2) expressing all dependent degrees of freedom $D - P$ in terms of the free variables $P$.

### B.1. Canonical forms for C-functions and AFCs

At this point we must commit ourselves to a particular representation for rotations. The implemented planner uses a rotation matrix specified by Euler angles, $\Theta = (\psi, \theta, \phi)$. The results of this section all extend trivially to other representations for rotations, although the coefficient-level detail is different.[18]

**Definition.** The *linear form* for a C-function $f : \mathbb{R}^3 \times SO(3) \to \mathbb{R}$ is an equivalent expression

$$f(x, y, z, \Theta) = E_1 x + E_2 y + E_3 z + E_4 ,$$

where $E_i : SO(3) \to \mathbb{R}$ for $i = 1, 2, 3, 4$.

**Definition.** A *trigonometric quadratic form* (TQF) (in $\phi$) for a C-function $f$ is an equivalent expression

[18] See [12] for comments on implementing a different representation for rotations (such as spherical angles, quaternions, or joint angles for a Cartesian manipulator).

$$f(x, y, z, \psi, \theta, \phi) = F_1 \sin \phi + F_2 \cos \phi + F_3 ,$$

where $F_i : \mathbb{R}^3 \times (\psi, \theta) \rightarrow \mathbb{R}$ for $i = 1, 2, 3$.

Analogously, a TQF (in $\phi$) for an ACF $g : \mathrm{SO}(3) \rightarrow \mathbb{R}$ is an equivalent expression

$$g(\psi, \theta, \phi) = G_1 \sin \phi + G_2 \cos \phi + G_3,$$

where $G_i : (\psi, \theta) \rightarrow \mathbb{R}$ for $i = 1, 2, 3$. The TQFs are defined here in $\phi$—of course we must also define the TQFs in $\psi$ and in $\theta$ in the natural way. $\phi$ will be our typical example angle in this discussion, however.

**Claim.** *Every C-function can be expressed as a linear form and as a TQF in $\psi$, $\theta$, and $\phi$; similarly, every ACF can be expressed as a TQF in $\psi$, $\theta$, and $\phi$.*

It is possible to develop simple mathematical intersection procedures operating on the canonical forms: Once the C-functions and ACFs have been expressed in the canonical forms, the intersection algorithms of [12] may be employed to calculate intersection manifolds of level C-surfaces, and to compute the intersection of a path with a C-surface. Let us consider some easy examples: Solving for the intersection to three level C-surfaces in linear form is clearly no harder than intersecting three planes. The obvious representation for the resulting intersection manifold is a map $h : (\psi, \theta, \phi) \rightarrow \mathbb{R}^3 \times \mathrm{SO}(3)$, whence the translational degrees of freedom are parameterized by the rotational degrees of freedom along the manifold. The intersection of two TQFs may be effectively calculated by a procedure for intersecting quadratics. Intersecting a pure translational path with a C-surface is equivalent to intersecting a line with a plane. A *TQF surface* is a level set $f^{-1}(l)$ whose defining function $f$ is in TQF. Intersecting a TQF surface with a pure rotational path in the $\pm\hat{\phi}$, $\pm\hat{\psi}$, $\pm\hat{\theta}$ directions reduces to finding the roots of a quadratic. Such paths move along the rotational "axes" of C-space. It is not hard to show that a continuous path through rotation space can be approximated as closely as desired by a sequence of linear motions along the rotational axes. Furthermore, the number of path segments required grows only linearly as the resolution of the approximation becomes finer.

### B.2. Moving through rotation space

Thus it is possible to intersect trajectories with C-surfaces. Once an intersection is found, we must then determine (1) whether the C-surface is applicable, and (2) whether it lies on the boundary of a C-space obstacle. The question of applicability may be resolved a priori by maintaining and updating an accurate set of applicable constraints as the planner moves through rotation space. This

set is called the *applicability set*. As the planner moves from $\Theta$ to $\Theta'$, the updating algorithm must detect which constraints have *expired* (ceased to be applicable) and which new constraints have been *activated* (become applicable). The expired constraints are deleted from the applicability set, and the new constraints are added. In this manner the trajectory will be intersected only with the applicable constraints.

A constraint expires when the trajectory passes out of its applicability region; symmetrically, a constraint is activated when the trajectory enters its applicability region. Both events may be detected by intersecting the trajectory with the ACF boundaries. Donald [12] shows that whenever a constraint expires, it is replaced by one or more constraints with *neighboring generators*. Thus it is possible for the dynamic computation of replacement applicability sets to be highly local in character. Since ACFs can be expressed in TQF, the same procedure used to intersect trajectories with C-surfaces can be employed to intersect trajectories with ACF boundaries. The algorithm decomposes the image of the trajectory into equivalence classes where the applicability set is invariant. Hence it can in principle be used to map out these equivalence classes on SO(3).

There are also two ways to determine if an intersection lies on the boundary of a C-space obstacle. Let $X$ be the intersection point of a trajectory with an applicable C-surface ker $f$. Then ker $f$ bounds a C-space obstacle at $X$ if either of the following holds:

(i) All applicable C-functions in $f$'s family are negative or zero-valued at $X$.

(ii) If the projection of $X$ into real space lies within the displaced face of the Minkowski solid corresponding to the generators for $f$.

Note that if all intersections with C-surfaces—including nonapplicable C-surfaces—have been sorted along the trajectory and if $X$ is the *first* intersection for which (ii) holds, then $f$ is applicable *and* $X$ lies on the boundary of the C-space obstacle.

ACKNOWLEDGMENT

I would like to thank Tomas Lozano-Pérez, Rod Brooks, Eric Grimson, Mike Brady, Mike Erdmann, John Canny, and Steve Buckley for comments on early drafts of this paper, and especially for many discussions about motion planning and robotics. Their help has been invaluable.

REFERENCES

1. Arnold, V.I., *Mathematical Methods of Classical Mechanics* (Springer, New York, 1978).
2. Brady, J.M., Hollerbach, J.M., Johnson, T.J., Lozano-Pérez, T. and Mason, M.T. (Eds.), *Robot Motion: Planning and Control* (MIT Press, Cambridge, MA 1983).
3. Brooks, R.A., Symbolic error analysis and robot programming, *Int. J. Rob. Res.* **1** (1) (1982) 29–68.
4. Brooks, R.A., Solving the find-path problem by good representation of free space, *IEEE Trans. Syst. Man Cybern.* **13** (1983) 190–197.

5. Brooks, R.A., Find-path for a PUMA-class robot, in: *Proceedings AAAI-83*, Washington, DC, 1983.
6. Brooks, R.A. and Lozano-Pérez, T., A subdivision algorithm in configuration space for findpath with rotations, in: *Proceedings IJCAI-83*, Karlsruhe, F.R.G., 1983.
7. Brou, P., Finding the orientation of objects in vector maps, Ph.D Thesis, Department of Electrical Engineering and Computer Science, MIT, Cambridge, MA, 1983.
8. Canny, J., On detecting collisions between moving polyhedra, *IEEE Trans. Pattern Anal. Mach. Intell.* **8** (2) (1986) 200–209.
9. Canny, J., A Voronoi method for the Piano-Movers' Problem, in: *Proceedings IEEE International Conference on Robotics and Automation*, St. Louis, MO, 1985.
10. Chatila, R., Systèm de navigation pour un robot mobile autonome: Modelisation et processus décisionnels, Ph.D. Thesis, L'Université Paul Sabatier de Toulouse, 1981.
11. Donald, B.R., Hypothesizing channels through free-space in solving the findpath problem, AI Memo 736, MIT AI Lab., Cambridge, MA, 1983.
12. Donald, B.R., Motion planning with six degrees of freedom, AI-TR-791, Artificial Intelligence Lab., MIT, Cambridge, MA, 1984.
13. Donald, B.R., On motion planning with six degrees of freedom: Solving the intersection problems in configuration space, in: *Proceedings IEEE International Conference on Robotics and Automation*, St. Louis, MO, 1985.
14. Donald, B.R., Robot motion planning with uncertainty in the geometric models of the robot and environment: A formal framework for error detection and recovery, in: *Proceedings IEEE International Conference on Robotics and Automation*, San Francisco, CA, 1986.
15. Drysdale, R.L., Generalized Voronoi diagrams and geometric searching, Department of Computer Science, Stanford University, Stanford, CA, 1979.
16. Erdmann, M.A., Using backprojections for fine-motion planning with uncertainty, *Int. J. Rob. Res.* **5** (1) (1986).
17. Faverjon, B., Obstacle avoidance using an octree in the configuration space of a manipulator, INRIA, Le Chesney, France, 1984.
18. Gouzenes, L., Strategies for solving collision-free trajectories problems for mobile and manipulator robots, Laboratoire d'Automatique et d'Analyse des Systemes du CNRS, Toulouse, France, 1983.
19. Hamilton, W.R., *Elements of Quaternions* (Chelsea, New York, 1969).
20. Hopcroft, J., Joseph, D. and Whitesides, S., On the movement of robot arms in 2-dimensional regions, TR 82-486, Cornell University, Computer Science Department, Ithaca, NY, 1982.
21. Hopcroft, J. and Wilfong, G., On the motion of objects in contact, TR-84-602, Computer Science Department, Cornell University, Ithaca, NY, 1984.
22. Kane, T.R. and Levinson, D.A., Successive finite rotations, *J. Appl. Mech.* **5** (1978) 945–946.
23. LCS Mathlab Group, MACSYMA reference manual, Lab. Computer Science, MIT, Cambridge, MA, 1983.
24. Lozano-Pérez, T., Spatial planning: A configuration space approach, *IEEE Trans. Comput.* **32** (1983) 108–120.
25. Lozano-Pérez, T., Automatic planning of manipulator transfer movements, *IEEE Trans. Syst. Man Cybern.* **11** (10) (1981) 681–698.
26. Lozano-Pérez, T., Motion planning for simple robot manipulators, in: *Proceedings Third International Symposium on Robotics Research*, Gouvieux, France, 1985.
27. Lozano-Pérez, T., Mason, M. and Taylor, R.H., Automatic synthesis of fine-motion strategies for robots, AI Memo 759, Artificial Intelligence Lab., MIT, Cambridge, MA, 1983.
28. Lozano-Pérez, T. and Wesley, M.A., An algorithm for planning collision-free paths among polyhedral obstacles, *Commun. ACM* **22**, 10 (1979).
29. Mason, M.T., Compliance and force control for computer-controlled manipulators, *IEEE Trans. Syst. Man Cybern.* **11** (6) (1981) 418–432.
30. Moravec, H.P., Visual mapping by a robot rover, in: *Proceedings IJCAI-79*, Tokyo, Japan, 1979.

31. Nguyen, Van-Duc, The find-path problem in the plane, AI Memo 760, Artificial Intelligence Lab., MIT, Cambridge, MA, 1983.
32. Nilsson, N., *Principles of Artificial Intelligence* (Tioga, Palo Alto, CA, 1980).
33. Ó'Dúnlaing, C. and Yap, C., The Voronoi diagram method of motion planning: I. The Case of a Disc, Courant Institute of Mathematical Sciences, New York, 1982.
34. Ó'Dúnlaing, C., Sharir, M., and Yap, C., Retraction: A new approach to motion planning, Courant Institute of Mathematical Sciences, New York, 1982.
35. Paul, L., *Robot Manipulation* (MIT Press, Cambridge, MA, 1981).
36. Popplestone, R.J., Ambler, A.P. and Bellos, I.M., An interpreter for a language for describing assemblies, *Artificial Intelligence* **14** (1) (1980) 79–107.
37. Reif, J.H., The complexity of the Movers Problem and generalizations, *Proc. IEEE Foundations Comput. Sci.*, San Juan, PR (1979) 421–427.
38. Schwartz, J.T. and Sharir, M., On the Piano Movers Problem, I: The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers, Rept. No. 39, Courant Institute of Mathematical Sciences, New York, 1982.
39. Schwartz, J.T. and Sharir, M., On the Piano Movers Problem, II: General techniques for computing topological properties of real algebraic manifolds, Rept. No. 41, Courant Institute of Mathematical Sciences, New York, 1982.
40. Schwartz, J.T. and Sharir, M., On the Piano Movers Problem, III: Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal barriers, Courant Institute of Mathematical Sciences, New York, 1982.
41. Spivak, M., *A Comprehensive Introduction to Differential Geometry* (Publish or Perish, Berkeley, CA, 1979).
42. Symon, K.R., *Mechanics* (Addison-Wesley, Reading, MA, 1971).
43. Udupa, S., Collision detection and avoidance in computer-controlled manipulators, Ph.D. Thesis, Department of Electrical Engineering, California Institute of Technology, Pasadena, CA, 1977.
44. Widdoes, C., A heuristic collision avoider for the Stanford robot arm, Stanford, Artificial Intelligence Laboratory, Stanford, CA, 1974.
45. Wingham, M., Planning how to grasp objects in a cluttered environment, M. Phil. Thesis, Department of Artificial Intelligence, Edinburgh, U.K., 1977.
46. Yap, C.K., Algorithmic motion planning, in: J.T. Schwartz and C.K. Yap (Eds.), *Advances in Robotics* **1** (Erlbaum, Hillsdale, NJ, 1986).

*Received February 1985; revised version received May 1986*