IEEE JOURNAL OF ROBOTICS AND AUTOMATION, VOL. RA-3, NO. 3, JUNE 1987

A Simple Motion-Planning Algorithm for General Robot Manipulators

TOMÁS LOZANO-PÉREZ, MEMBER, IEEE

Abstract—**A simple and efficient algorithm is presented, using configuration space, to plan collision-free motions for general manipulators. An implementation of the algorithm for manipulators made up of revolute joints is also presented. The configuration-space obstacles for an n degree-of-freedom manipulator are approximated by sets of $n - 1$-dimensional slices, recursively built up from one-dimensional slices. This obstacle representation leads to an efficient approximation of the free space outside of the configuration-space obstacles.**

I. Introduction

THIS PAPER presents an implementation of a new motion-planning algorithm for general robot manipulators moving among three-dimensional polyhedral obstacles. The algorithm has a number of advantages: it is simple to implement, it is fast for manipulators with few degrees of freedom, it can deal with manipulators having many degrees of freedom (including redundant manipulators), and it can deal with cluttered environments and nonconvex polyhedral obstacles. An example of a path obtained from an implementation of the algorithm is shown in Fig. 1.

The ability to plan automatically collision-free motions for a manipulator given geometric models of the manipulator and the task is one of the capabilities required to achieve *task-level* robot programming [15]. Task-level programming is one of the principal goals of research in robotics. It is the ability to specify the robot motions required to achieve a task in terms of task-level commands, such as "Insert pin-A in hole-B," rather than robot-level commands, such as "move to 0.1, 0.35, 1.6."

The motion-planning problem, in its simplest form, is to find a path from a specified starting robot configuration to a specified goal configuration that avoids collisions with a known set of stationary obstacles. Note that this problem is significantly different from, and quite a bit harder than, the collision detection problem: detecting whether a known robot configuration or robot path would cause a collision [1], [4]. Motion planning is also different from on-line obstacle avoidance: modifying a known robot path so as to avoid unforeseen obstacles [6], [9], [10], [11].

Manuscript received May 19, 1986; revised December 11, 1986. This work was supported in part by the Office of Naval Research under Contract N00014-82-K-0494, in part by a grant from the System Development Foundation, in part by the Advanced Research Projects Agency under Office of Naval Research Contracts N00014-85-K-0214 and N00014-82-K-0334, and in part by a National Science Foundation Young Investigator Grant.

The author is with the Artificial Intelligence Laboratory and the Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, Cambridge, MA 02139, USA.

IEEE Log Number 8714998.

Although general-purpose task-level programming is still many years away, some of the techniques developed for task-level programming are relevant to existing robot applications. There is, for example, increasing emphasis among major robot users on developing techniques for off-line programming, by human programmers, using computer-aided design (CAD) models of the manipulator and the task. In many of these applications motion planning plays a central role. Arc welding is a good example; specifying robot paths for welding along complex three-dimensional paths is a time-consuming and tedious process. The development of practical motion-planning algorithms could reduce significantly the programming time for these applications.

A great deal of research has been devoted to the motion-planning problem within the last five to eight years, e.g., [2], [3], [5], [7], [8], [12]–[14], [16], [17], [19], [20]. However, few of these methods are simple enough and powerful enough to be practical. Practical algorithms are particularly scarce for manipulators made up of revolute joints, the most popular type of industrial robot. The author is aware of only two previous motion-planning algorithms that are both efficient and reasonably general for revolute manipulators with three or more degrees of freedom [2], [7]. Brook's algorithm [2] has demonstrated impressive results but is fairly complex. Faverjon's algorithm [7], on the other hand, is appealingly simple. The basic approach of the algorithm described here is closely related to the method described by Faverjon. Many of the details of the present algorithm, however, especially the treatment of three-dimensional constraints and the free space representation, are new and more general.

The approach taken in this algorithm is similar to that of [7], [8], [12], [13] in that it involves quantizing joint angles. It differs in this respect from exact algorithms such as [17], [19]. On the other hand, the quantization approach lends itself readily to efficient computer implementation. The approach taken here differs from most previous configuration-space algorithms, for example, [4], [5], [13], [14], in that no attempt is made to characterize the *surfaces* of the obstacles in the configuration space. Instead, the approximate obstacles are built up from a series of one-dimensional ranges of forbidden values. As with many motion-planning algorithms, including [5], [13], [14], [19], the free-space outside of the obstacles in this algorithm is represented as a collection of cells. The connectivity graph of the cells is used to search for a path. A cellular representation of the free-space contrasts with the use of a one-dimensional subset of the free-space such as the Voronoi diagram; see, for example, [2], [17].

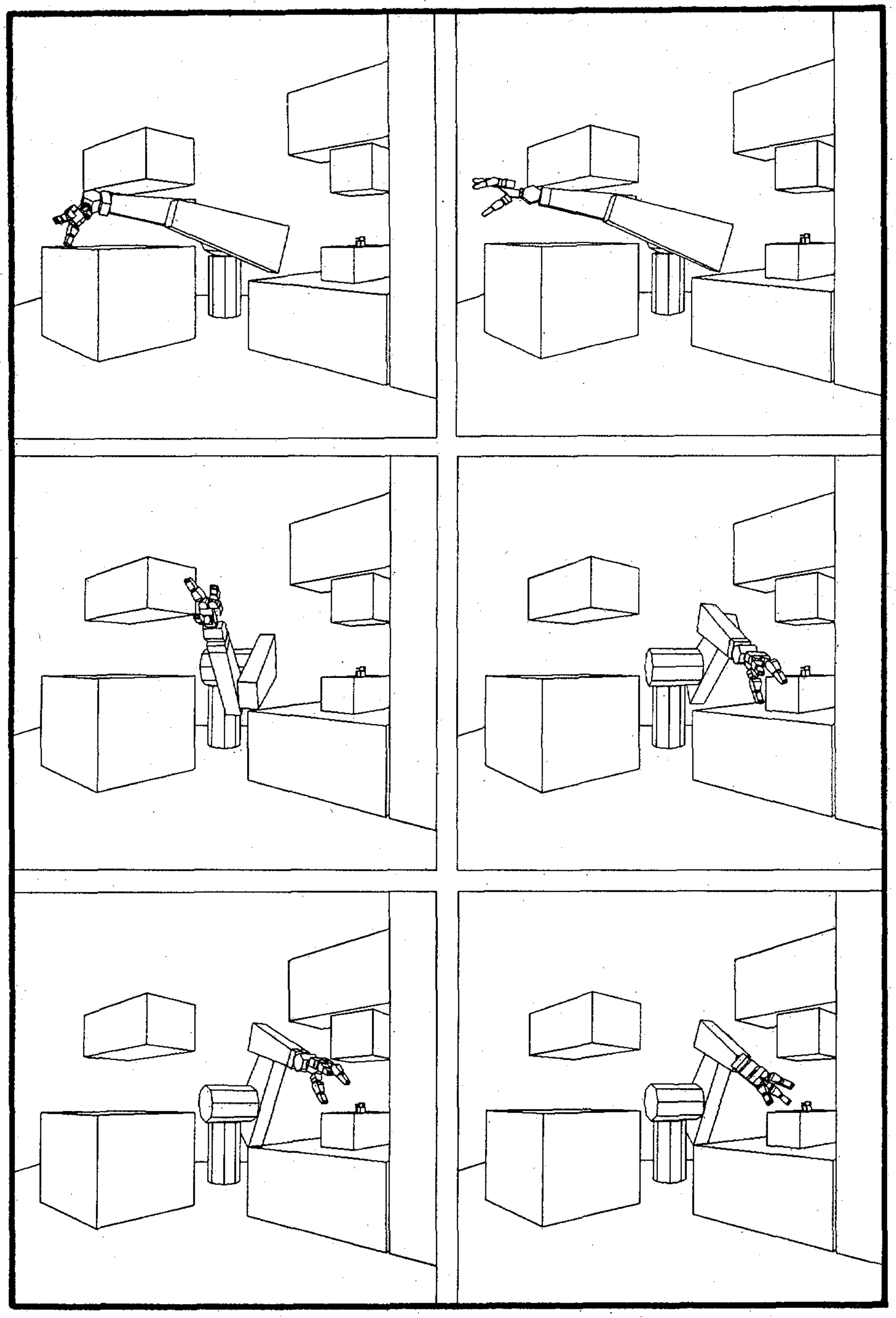


Fig. 1. Path for all six links of Puma obtained using algorithm described here. Three-fingered hand shown as end-effector does not change configuration during motion, but its shape is taken into account during motion planning. Total planning time on Symbolics 3600 was approximately 3 min. Joints were sampled at $\pm 3°$.

The purpose of this paper is to show that motion planning for general manipulators can be both simple and relatively efficient in most practical cases. There is no reason why motion planning should be any less practical than computing renderings of three-dimensional solids in computer graphics. In both cases, there are many simple numerical computations that can benefit from hardware support. In fact, it is worth noting that in the examples in Fig. 1 it took longer to compute the hidden-surface displays in the figures than to compute the paths.

II. The Basic Approach: Slice Projection

The *configuration* of a moving object is any set of parameters that completely specify the position of every point on the object. Configuration space (C space) is the space of configurations of a moving object. The set of joint angles of a robot manipulator constitute a configuration. Therefore, a robot's joint space is a configuration space. The Cartesian parameters of the robot's end effector, on the other hand, do not usually constitute a configuration because of the multiplicity of solutions to a robot's inverse kinematics. It is possible to

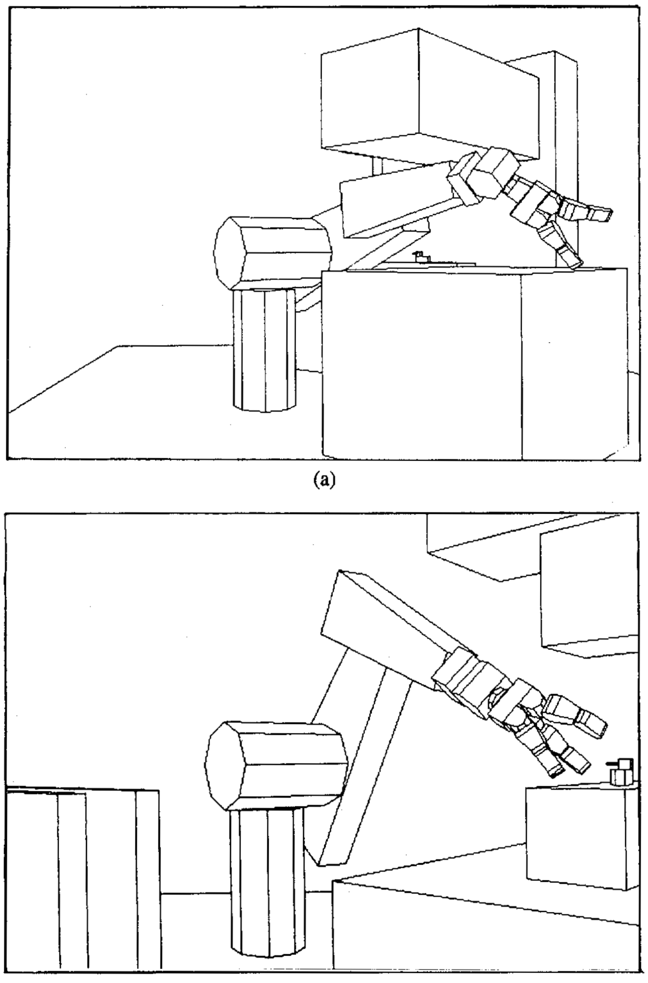


Fig. 2. (a) Closeup of initial configuration of manipulator for path in Fig. 1(b) (from different viewpoint). (b) Closeup of final configuration of manipulator for path in Fig. 1(b). Note that full polyhedral descriptions of arm and end-effector are used in algorithm.

map the obstacles in the robot's workspace into its configuration space [3]–[5], [13], [14]. These *C-space obstacles* represent those configurations of the moving object that would cause collisions. *Free space* is defined to be the complement of the C-space obstacles.

Motion planning requires an explicit characterization of the robot's free space. The characterization may not be complete, for example, it may cover only a subset of the free space. However, without a characterization of the free space, one is reduced to trial and error methods to find a path. In this paper we show how to compute approximate characterizations of the free space for simple manipulators. By simple manipulators we mean manipulators composed of a nonbranching sequence of links connected by either revolute or prismatic joints (see [18] for a treatment of the kinematics of simple manipulators). We restrict the position of link zero of a simple manipulator to be fixed. Most industrial manipulators (not including parallel-jaw grippers) are simple manipulators in this sense.

The C-space obstacles for a manipulator with n joints are, in general, n-dimensional volumes. Let C denote an n-dimensional C-space obstacle for a manipulator with n joints. We represent approximations of C by the union of $n - 1$-dimensional *slice projections* [13], [14]. Each $n - 1$-dimensional configuration in a slice projection of C represents a range of n-dimensional configurations (differing only in the value of a single joint parameter) that intersects C.

A slice projection of an n-dimensional C-space obstacle is defined by a range of values for one of the defining parameters of the C space and an $n - 1$-dimensional volume. Let $q = (q_1, \cdots, q_n)$ denote a configuration, where each q_i is a joint parameter, which measures either angular displacement (for revolute joints) or linear displacement (for prismatic joints). Let π_j be a projection operator for points, defined such that

$$\pi_j(q) = (q_1, \cdots, q_{j-1}, q_{j+1}, \cdots, q_n).$$

Let $\Pi_{[a_j, b_j]}(S)$ be a projection operator for point sets S, defined such that

$$\Pi_{[a_j, b_j]}(S) = \{\pi_j(q) \mid q \in S \text{ and } q_j \in [a_j, b_j]\}.$$

Then, the slice projection of the obstacle C for values of $q_j \in [a_j, b_j]$ is

$$\Pi_{[a_j, b_j]}(C).$$

The definition of slice projection is illustrated in Fig. 3. In the foregoing example, joint j is called the *slice joint* while the other joints are known as *free joints*.

Note that a slice projection is a *conservative* approximation of a segment of an n-dimensional C-space obstacle. An approximation of the full obstacle is built as the union of a number of $n - 1$-dimensional slice projections, each for a different range of values of the same joint parameter (Fig. 3). Each of the $n - 1$-dimensional slice projections, in turn, can be approximated by the union of $n - 2$-dimensional slice projections and so on, until we have a union of one-dimensional volumes, that is, linear ranges. This process is illustrated graphically in Fig. 3. Note that the slice projection can be continued one more step until only zero-dimensional volumes (points) remain, but this is wasteful.

Consider a simple two-link planar manipulator whose joint parameters are q_1 and q_2. C-space obstacles for such a manipulator are two-dimensional. The one-dimensional slice projection of a C-space obstacle C for $q_1 \in [a, b]$ is some set of linear ranges $\{R_i\}$ for q_2. The ranges must be such that if there exists a value of q_2, call it d, and a value $q_1 \in [a, b]$, call it c, for which $(c, d) \in C$, then d is in one of the R_i (Fig. 3).

A representation of a configuration space with obstacles is illustrated in Fig. 4(b), for the two-link manipulator and obstacles shown in Fig. 4(a). The actual configuration space is the surface of a torus since the top and bottom edge of the diagram coincide ($0 = 2\pi$), as do the left and right edge. The obstacles are approximated as a set of q_2 ranges (shown dark) for a set of values of q_1. The resolution is 2° along the q_1 axis.

If the manipulator has three links, its configuration space can be constructed as follows.

1) Ignore links beyond link 1. Find the ranges of legal values of q_1 by considering rotations of link 1 around the fixed base.

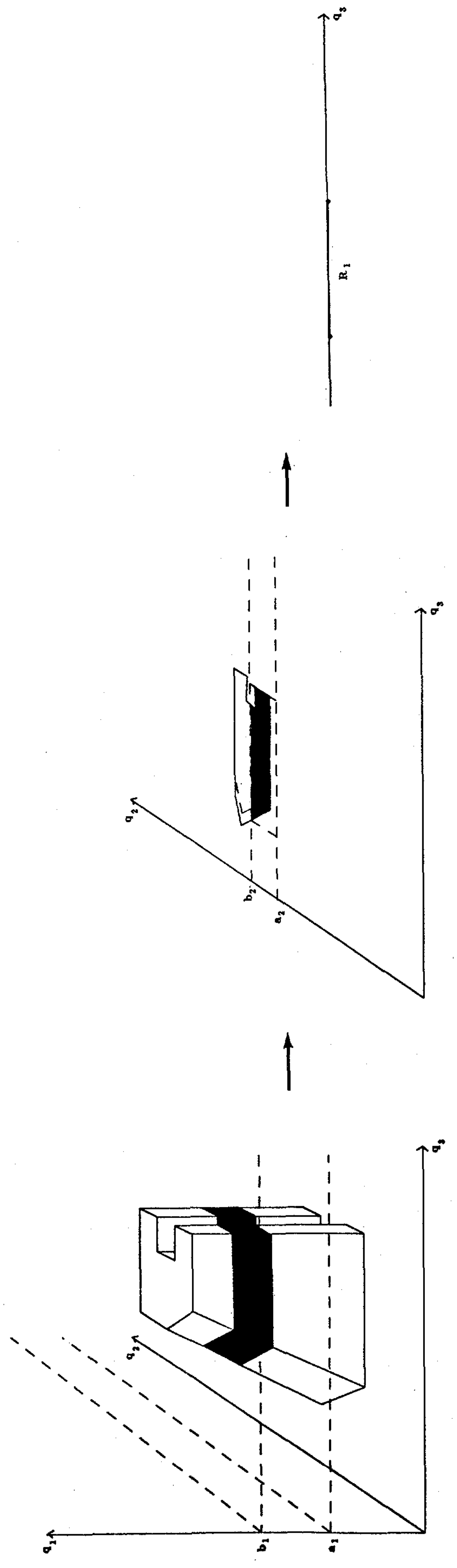


Fig. 3. Slice projection of three-dimensional obstacle into list of two-dimensional slices that are in turn represented by one-dimensional slices.

2) Sample the legal range of q_1 at the specified resolution. Do steps 3–5 for each of the value ranges of q_1.
3) Ignore links beyond link 2. Find the ranges of legal values of q_2 by considering rotating link 2 around the positions of joint 2 determined by the current value range of q_1.
4) Sample the legal range of q_2 at the specified resolution. Do step 5 for each of these value ranges of q_2.
5) Find the ranges of legal values of q_3 by considering rotating link 3 around the position of joint 3 determined by the current value ranges of q_1 and q_2.

Some sample slices from a configuration space computed in this way can be seen in Fig. 5.

Note that the process just described is an instance of the following simple recursive process. To compute C space (i),

1) ignore links beyond link i, and find the ranges of legal values of q_i by considering rotating link i around the positions of joint i determined by the current value ranges of $q_1, \cdots, q_{i-1}$;
2) if $i = n$, then stop; else sample the legal range of q_i at the specified resolution. Compute C space ($i + 1$) for each of these value ranges of q_i.

Observe that the basic computation to be done is that of determining the ranges of legal values for a joint parameter given ranges of values of the previous joints. This computation is the subject of Section III.

The recursive nature of the C-space computation calls for a recursive data structure to represent the C space. The current implementation uses a tree whose depth is $n - 1$, where n is the number of joints, and whose branching factor is the number of intervals into which the legal joint parameter range for each joint is divided (Fig. 6). The leaves of the tree are ranges of legal (or forbidden) values for the joint parameter n. Many of the internal nodes in the tree will have no descendants because they produce a collision of some link $i < n$.

The main advantage of a representation method built on recursive slice projection is its simplicity. All operations on the representation boil down to dealing with linear ranges, for which very simple and efficient implementations are possible. The disadvantages are the loss of accuracy, and the rapid increase of storage and processing time with dimensionality of the C space. Contrast this approach with one that represents the boundaries of the obstacles by their defining equations [4], [5]. Using the defining equations is cleaner and more accurate, but the algorithms for dealing with interactions between obstacle boundaries are very complex. I believe that the simplicity of slice projection outweighs its drawbacks. These drawbacks can be significantly reduced by exercising care in the implementation of the algorithms.

III. Slice Projections for Polygons

The key step in our approach is computing one-dimensional slice projections of C-space obstacles, that is, determining the range of forbidden values of one joint parameter, given ranges of values for all previous joint parameters. We will illustrate how these ranges may be computed by considering the case of

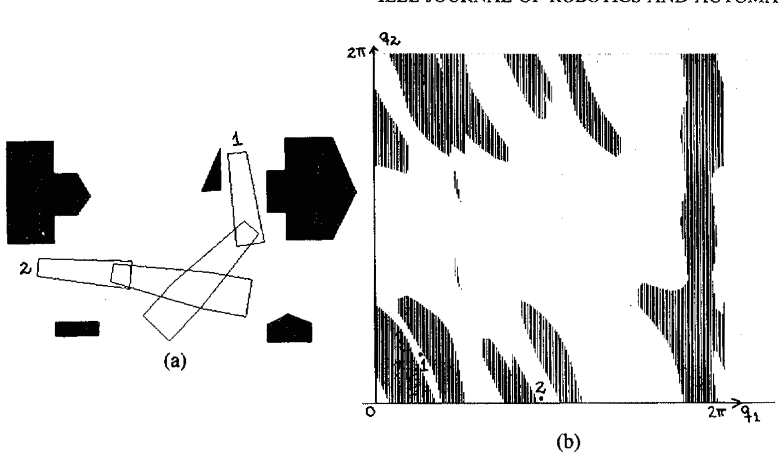


Fig. 4. (a) Two-link revolute manipulator and obstacles. (b) Two-dimensional C space with obstacles approximated by list of one-dimensional slice projections (shown dark). Initial and final position of manipulator are shown in input space and C space.

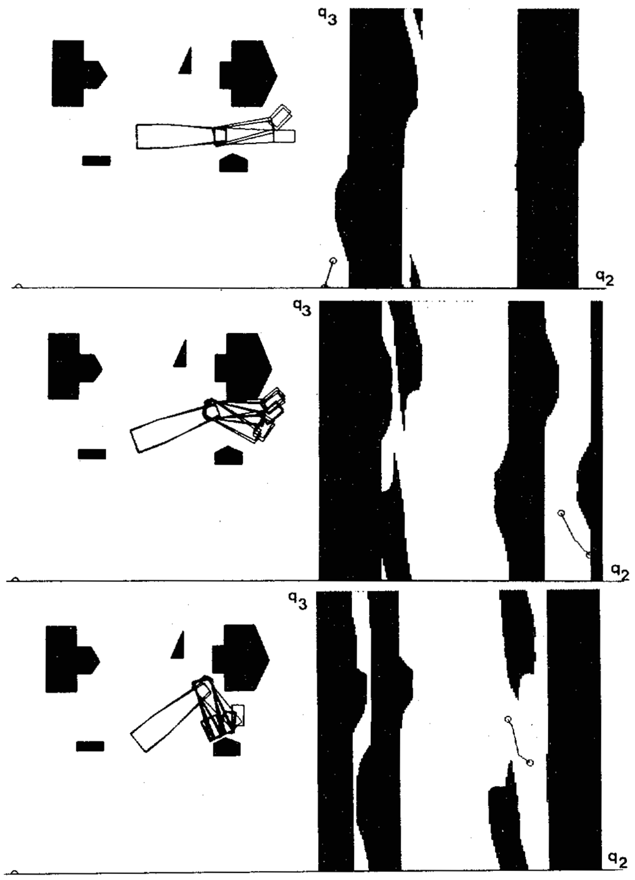


Fig. 5. Configuration space slices for three link revolute manipulator. Figures on right show samples of two-dimensional slice projections used to approximate three-dimensional configuration space. Each slice shows constraints on q_2 and q_3 for different range of values of q_1 (note different orientations of manipulator's first link in right figures). On left, manipulator is shown in number of configurations along path shown on slice diagram; initial and final configurations of paths are indicated by circles.

planar revolute manipulators moving among planar obstacles. We will first discuss this problem informally and then derive the solution from the equations of C surfaces.

A. A Geometric View

Assume that joint k, a revolute joint, is the free joint for a one-dimensional slice projection and that the previous joints are fixed at known values. Note that we assume, for now, that the previous joints are fixed at *single* values rather than *ranges* of values; we will see in Section III-C how to relax this restriction. We require that the configuration of the first $k - 1$ links be safe, that is, no link intersects an obstacle. This is guaranteed by the recursive computation we saw in Section II. Given these assumptions, we need to find the ranges of values of the single joint parameter q_k that are forbidden by the presence of objects in the workspace.

The ranges of forbidden values for q_k will be bounded by angles where link k is just touching an obstacle. For polygonal links moving among polygonal obstacles, the extremal contacts happen when a vertex of one object is in contact with an edge of another object. Therefore, the first step in computing the forbidden ranges for q_k is to identify those *critical values* of q_k for which some obstacle vertex is in contact with a link edge or some link vertex is in contact with an obstacle edge (Fig. 7).

The link is constrained to rotate about its joint; therefore, every point on the link follows a circular path when the link rotates. The link vertices, in particular, are constrained to known circular paths. The intersection of these paths with obstacle edges determine some of the critical values of q_k, for example, B in Fig. 7. As the link rotates, the obstacle vertices also follow known circular paths relative to the link. The intersection of these circles with link edges determine the remaining critical values for q_k, for example, A in Fig. 7.

Determining whether a vertex and an edge segment can intersect requires first intersecting the circle traced out by the vertex and the infinite line supporting the edge to compute the potential intersection points. The existence of such an intersection is a *necessary* condition for a contact between link and obstacle, but it is not *sufficient*. Three additional constraints must hold (Fig. 8). 1) *In-edge* constraint is that where the intersection point must be within the finite edge segment, not just the line supporting the edge. 2) For *orientation* constraint, the orientation of the edges at the potential contact must be compatible, that is, the edges that define the contact vertex must both be outside of the contact edge. For the *reachability* constraint for nonconvex objects, there must not be other contacts that prevent reaching this point.

The in-edge constraint can be tested trivially given the

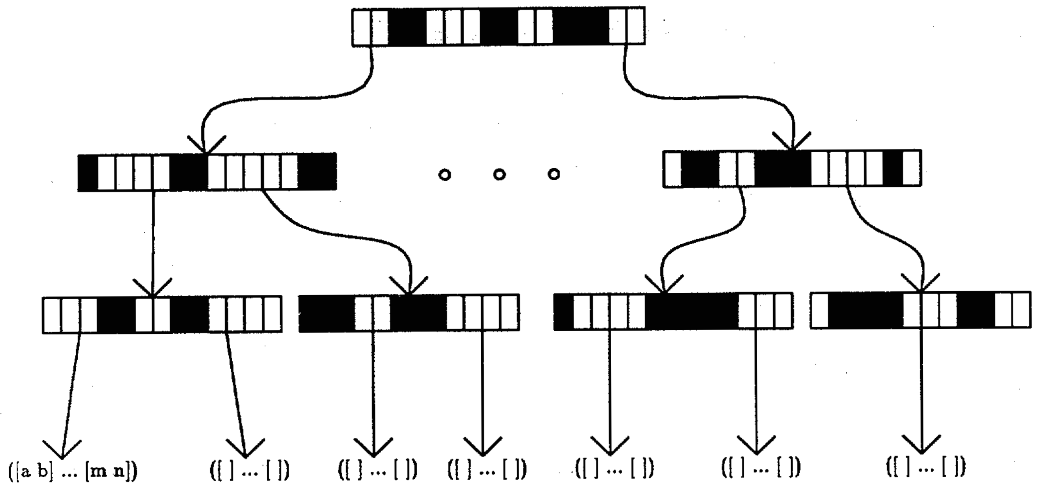


Fig. 6. Recursive nature of C space leads to recursive data structure: an n-level tree whose leaves represent legal ranges of configurations for robot manipulator.

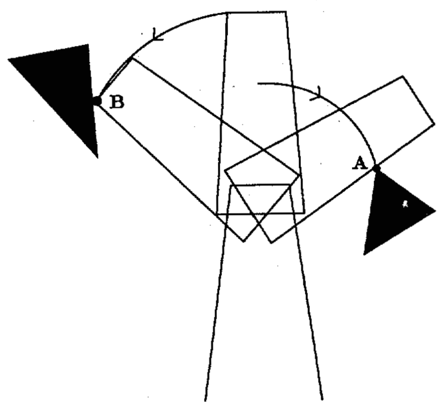


Fig. 7. Contact conditions for computing one-dimensional slice projections. (a) Vertex of obstacle and edge of link. (b) Vertex of link and edge of obstacle. Circles indicate path of vertices as link rotates around specified joint.

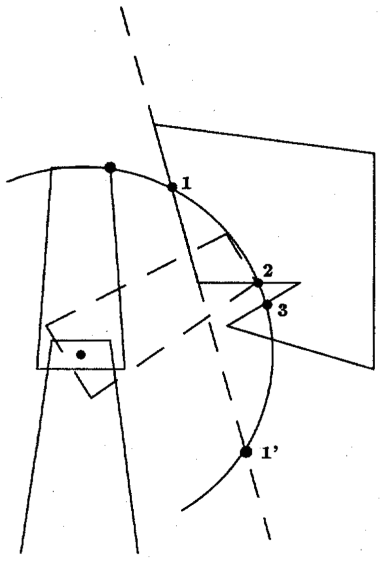


Fig. 8. Given intersection of vertex circle and edge line, following conditions must be met for feasible contact. a) Contact must be in edge segment, contact 1 satisfies this but 1′ does not; b) edges that define contact vertex must both be outside of contact edge, contact 1 satisfies this but contact 2 does not; c) contact must be reachable, contact 1 satisfies this, but contact 3 does not (this condition is only relevant for nonconvex objects).

potential contact point and the endpoints of the contact edge. Since we know that the contact point is on the line of the edge, all that remains to be determined is whether it lies between the endpoints of the edge. This can be done by ensuring that the x and y coordinates of the contact point are within the range of x and y coordinates defined by the edge endpoints. Note that for contacts involving link edges and obstacle vertices, the position of the endpoints of the link edge must be rotated around the joint position by the computed value of the joint angle at the contact.

The orientation constraint can also be tested simply. All that is required is that the two edges forming the contact vertex be on the outside of the contact edge. Polygon edges are typically oriented so that they revolve in a counterclockwise direction about the boundary. Therefore, the outside of the polygon is on the right of the edge as we traverse the boundary. Given this, the feasibility of a contact can be verified simply by comparing the absolute orientations of the edges involved in the contact.

The reachability constraint, on the other hand, requires examining all the contacts of the link with a given obstacle that satisfy the first two constraints. For each contact angle q we determine whether values of q_k greater than q cause collision or whether values less than q cause collision (Section III-B). The contact angles together with the collision directions can be merged to form the ranges of forbidden values for q_k. This process is illustrated in Fig. 9.

B. Derivation Using C Surfaces

The two types of contacts (vertex–edge and edge–vertex) give rise to the two basic types of C-space boundary (hyper-) surfaces [3]–[5], [14]. One type of C surface (type A) characterizes the configuration of the moving object for which a vertex of the stationary obstacle is in contact with the infinite line supporting an edge of the moving object. The other (type B) characterizes the configuration of the moving object for which the infinite line supporting an edge of the stationary obstacle is in contact with a vertex of the moving object. The equations of such surfaces are parameterized by the configuration parameters of the moving object. For planar polygons, x,

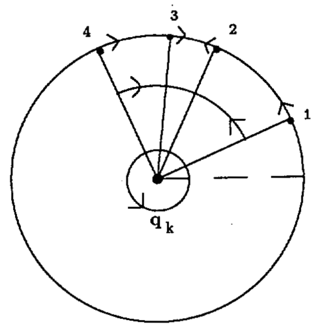


Fig. 9. Constructing ranges of forbidden values using potential contact angles and collision directions.

y, θ can be used as configuration parameters; for manipulators, the q_i are the configuration parameters.

For a revolute joint, choose the coordinate system to be located at the joint. The coordinate representation of all of the vectors will be relative to this coordinate system. We represent a line supporting an edge by an equation of the form: $\boldsymbol{n}\cdot\boldsymbol{x} + d = 0$. Where $\boldsymbol{n}$ is the (outward pointing) unit vector that is normal to the line and the absolute value of d is the perpendicular distance to the edge from the origin. The condition for a vertex $\boldsymbol{v}$ being in contact with such a line is simply $\boldsymbol{n}\cdot\boldsymbol{v} + d = 0$.

For a type B contact, we are given a link vertex whose initial position vector (for $q_k = 0$) is $\boldsymbol{v}$ and an obstacle edge whose line equation is $\boldsymbol{n}\cdot\boldsymbol{x} + d = 0$. If the link angle is q_k, the coordinates of the rotated link vertex are

$$\boldsymbol{v}' = (v_x \cos q_k - v_y \sin q_k,\ v_x \sin q_k + v_y \cos q_k).$$

Substituting into the line equation yields a simple trigonometric equation in q_k (all the other terms are constant):

$$(n_x v_x + n_y v_y) \cos q_k + (n_y v_x - n_x v_y) \sin q_k + d = 0. \quad (1)$$

From the definition of the scalar and vector product, we have that

$$n_x v_x + n_y v_y = \|\boldsymbol{v}\| \cos \phi \qquad n_y v_x - n_x v_y = \|\boldsymbol{v}\| \sin \phi$$

where ϕ is the angle between $\boldsymbol{n}$ and $\boldsymbol{v}$. From this, it is clear that the C-surface equation is merely

$$\|\boldsymbol{v}\| \cos (q_k - \phi) = -d.$$

The solution to this equation is

$$q_k = \cos^{-1}\left(\frac{-d}{\|\boldsymbol{v}\|}\right) + \phi. \quad (2)$$

Fig. 10 illustrates this situation. There is one such C surface for each combination of link vertex and obstacle edge. Of course, only convex vertices need be considered; no contact is possible at a concave vertex.

Using the same notation, except that the edge is a link edge and the vertex an obstacle vertex, the equation for a type-A C surface is

$$(n_x v_x + n_y v_y) \cos q_k - (n_y v_x - n_x v_y) \sin q_k + d = 0. \quad (3)$$

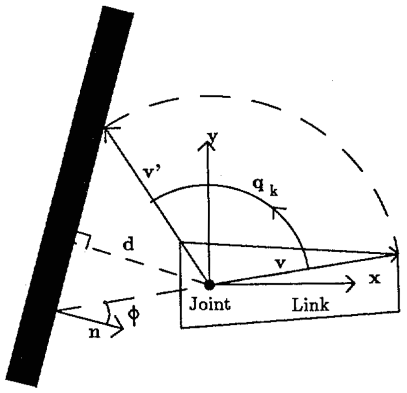


Fig. 10. Illustration of terms used in (1)–(4).

The only difference is the sign of the coefficient of $\sin q_k$. This difference arises from the fact that we are thinking of the obstacle vertex as counter-rotating while the link stands still. That is, the direction of rotation of the vertex is the opposite of q_k; this changes the sign of the sine of the angle. The solution to this equation is:

$$q_k = \cos^{-1}\left(\frac{-d}{\|\boldsymbol{v}\|}\right) - \phi. \quad (4)$$

One such C surface exists for each combination of (convex) obstacle vertex and link edge.

Note that there are generally two solutions to each of the equations (arising from the arccosine) since they correspond to intersections of a circle traced out by a vertex and an infinite line supporting an edge. These solutions, however, do not necessarily represent feasible contacts between the link and an obstacle. The remaining constraints illustrated in Fig. 8 must also be satisfied. Of course, when the magnitude of the argument to the arccosine is greater than one, this indicates an infeasible contact, that is, the line is beyond the reach of the vertex.

The in-edge constraint can be checked, as described before, by computing the coordinates of the intersection point and the positions of the edge endpoints, given the computed values of q_k.

The solutions obtained from (2) and (4) must also satisfy the orientation constraint. One way of testing this constraint is by ensuring that the polygon edges that intersect at the contact vertex both point outward from the contact edge. If $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are the edge vectors pointing away from the vertex (Fig. 11), then the orientation constraint boils down to

$$\text{sgn}\,(\boldsymbol{n} \cdot \boldsymbol{e}_1) \geq 0 \qquad \text{sgn}\,(\boldsymbol{n} \cdot \boldsymbol{e}_2) \geq 0$$

where $\text{sgn}\,(x) = x/|x|$ for $x \neq 0$ and 0 otherwise.

The reachability constraint is handled as described in Section III-B. To do that, we must be able to tell whether an increase in q_k will move the link towards or away from the obstacle. This can be done by computing the derivative of (1) and (2). The left side of these equations is a measure of the perpendicular distance of a vertex from an edge. The sign of the derivative of this distance with respect to q_k will indicate whether a change in q_k will move further into contact or away.

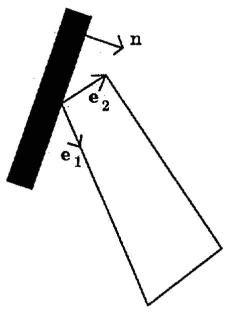


Fig. 11. Testing orientation constraint for polygonal contact.

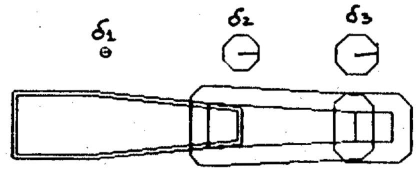


Fig. 12. The kth manipulator link can be grown by radius δ_k: maximum cartesian displacement of any point on link in response to joint displacements ϵ_i for $i \leq k$.

For example, the sign of the derivative of the distance for a type B contact (see (1)) is determined by the sign of $-\sin(q_k + \phi)$ evaluated at the value of q_k that gives rise to contact.

C. The Effect of Ranges of Joint Angles

Our discussion thus far has been limited to situations where all the joints except the last have known fixed values. The definition of one-dimensional slice projections allows all the joints, save one free joint, to be within a range, not just a single value. We can readily convert the slice projection problem (for ranges of joint values) to the simpler cross section projection problem (for single joint values) we have already discussed. The idea is to replace the shape of the link under consideration by the area it sweeps out when the joints defining the slice move within their specified value ranges [13], [14]. Any safe placement of the expanded link represents a range of legal displacements of the original link within the specified joint ranges.

In most cases, instead of computing the exact swept volumes, we can use a very simple approximation method. Assume the manipulator is positioned at the configuration defined by the midpoint of all the joint value ranges specified for the slice projection. Compute the upper bound on the largest Cartesian displacement of any point on link k in response to any displacement within the specified range of joint values. Call this bound δ_k. If we "grow" each link by its corresponding radius δ_k, the grown link includes the swept area.

A polygonal approximation to the grown link can be obtained by computing the "set sum" of "Minkowski sum" of the link and a polygon enclosing a circle of radius δ [14]. An example of such a grown manipulator can be seen in Fig. 12.

We can illustrate this approach by considering how to compute δ_k for a planar manipulator composed of n revolute joints. The motion of a joint affects the displacement of all subsequent links. Therefore, the maximum Cartesian displacement of each link depends on the maximum total distance from any point on a link to the base joint and on the maximum angular displacement of the link. The maximum distance of points on link k is the sum of the distances between all previous joints plus the distance of any point on link k from joint k. The angular displacement of link k in a planar revolute manipulator is also the sum of the angular displacements of all the previous joints. Given the distance d and the angle θ, the magnitude of the displacement (chord of a circle) is $\sqrt{2(1-\cos\theta)}$.

Let the allowed angle range for q_i be $\alpha_i + \epsilon_i$; let r_i be the maximum distance of any point on link i from joint i; and let l_i be the distance from joint i to joint $i+1$. The value of δ_k is

$$\delta_k = \left[\left(\sum_{i=1}^{k-1} l_i\right) + r_k\right] \sqrt{2\left(1-\cos\left(\sum_{j=1}^{k} \epsilon_j\right)\right)}.$$

Because the last link's motion is never quantized when computing the C space, we have that $\epsilon_k = 0$. This value of δ_k is very conservative; it is the largest displacement anywhere in the work space. In fact, it corresponds to the displacement in a link when all the previous links are fully outstretched, that is, all the $\alpha_j = 0$, $j \leq k$. Different configurations would yield smaller values of δ_k.

In Fig. 12 the relevant parameter values are $\epsilon_1 = 2^0$, $\epsilon_2 = 2^0$, $\epsilon_3 = 0$, $l_0 = 0$, $l_1 = 17.0$, $l_2 = 17.0$, $r_1 = 18.44$, $r_2 = 17.26$, $r_3 = 5.385$. Therefore, the values of the δ_k are $\delta_1 = 0.644$, $\delta_2 = 2.39$, $\delta_3 = 2.749$. Note the growth in the value of δ_k as the distance from the base increases. Because of this, one might want to choose a finer quantization for joints associated with long links near the base, for example, joint two in our example.

In some applications, if the ϵ_k are small, it may be preferable to ignore the effect of small ϵ_k during planning and simply check the resulting path for collisions. Of course, if the joint ranges ϵ_k are large, these gross approximations may be too conservative and the exact swept volume should be used.

D. Prismatic Joints

The discussion so far has concentrated on revolute joints, but the approach is not limited to them. If any of the joints are prismatic, only the computation of one-dimensional slices will be different and, in fact, it will be simpler.

As before, the key problem is computing the critical value of the joint parameters for which a link is in contact with an obstacle. These contacts involve contact of a vertex and an edge. So, as in the case for revolute joints, we need to determine the locus of motion of link vertices relative to obstacle edges and the locus of motion of obstacle vertices relative to link edges. For links actuated by revolute joints, we have seen that the vertices trace out circles. For links actuated by prismatic joints, the points on the link trace out lines. Potential points of contact occur where the lines defined by the motion of the vertices intersect the edges. This operation replaces the intersection of circles and lines in the preceding discussion.

The points of intersection must still satisfy the in-edge, orientation, and reachability constraints. Note that the in-edge constraint must now be modified to check, not only that the intersection point is within the finite edge segment of the polygon, but also that the contact is within the range of motion of the joint.

IV. Slice Projections for Polyhedra

The basic approach described in Section III carries over directly to three-dimensional manipulators and obstacles. There is, however, one significant difference: there are three types of contacts possible between three-dimensional polyhedra. The three contact types are type A, vertex of obstacle and face of link; type B, vertex of link and face of obstacle; and type C, edge of link and edge of obstacle.

Let us consider type B contacts first. Each revolute joint is characterized by an axis of rotation. As the joint rotates, link vertices trace circles in a plane whose normal is the joint axis. The intersection of this circle with the plane supporting an obstacle face defines two candidate points of contact (see the appendix). As in the two-dimensional case, possible contacts must satisfy three constraints to be feasible. For the in-face constraint, the contact must be within the obstacle face; for the orientation constraint, all of the link edges meeting at the vertex must be outside of the obstacle; and for the reachability constraint, for nonconvex polyhedra, there must not be any earlier contacts that prevent reaching this one.

The in-face constraint can be checked using any of the existing algorithms for testing whether a point is in a polygon. The orientation constraint can be enforced by checking that the dot products of the face normal with each of the vectors from the contact vertex to adjacent vertices is positive [5]. The reachability constraint is enforced exactly as in the two-dimensional case by merging the forbidden angle ranges.

Type A contacts are handled analogously to type B contacts except that now the vertex belongs to an obstacle and the face to a link. The axis of rotation is still that of the manipulator joint.

Detecting type C contacts requires detecting the intersection of a line (supporting a link edge) rotating about the joint axis and a stationary line (supporting an obstacle edge). The solution for this case can be found in the appendix. Of course, an intersection point must be inside both edge segments to be feasible. There is also an orientation constraint which is a bit more difficult to derive than those for type A and B contacts but not particularly difficult to check (for the derivation, see [5]). The appendix shows the details of these computations.

V. Free-Space Representation

Having obtained a conservative approximation of the C-space obstacles, the free space is simply the complement of all the obstacles. Since the obstacles are ultimately represented as sets of linear ranges, the complement is trivial to compute. A two-dimensional free space, for example, will be represented as a list of one-dimensional slices. Each slice represents the ranges of *legal* values of q_2 for some small range of values of q_1. This is in itself a reasonably convenient representation of the free space but not very compact. If we were to try to find paths through the individual slices a great deal of time would be wasted searching through nearly identical slices. A more compact representation is called for, one that captures some of the coherence between adjacent slices.

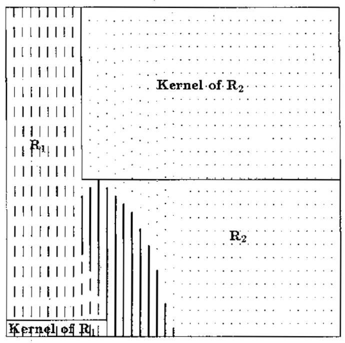


Fig. 13. Illustration of definition of free-space regions. Bold lines indicate configuration space obstacles. Two regions are indicated in dashed lines. Kernels are rectangular areas within regions corresponding to common intersection of all free ranges in region.

The free-space representation used in the current implementation is made up of *regions*. A region is made up of linear ranges from a set of adjacent slices such that the ranges all overlap. The area of common overlap of all the slices in a region is rectangular and called the region's *kernel* (Fig. 13).

The regions are built by looping over the slices from left to right in the diagrams, that is, from $q_1 = 0$ towards $q_1 = 2\pi$. Each legal range in the first slice initializes a new region. For each region, we keep track of the legal range of q_2 values common to all the slices in the region; this is the kernel. As each new slice is considered, the ranges in that slice are compared to the kernel of the regions in the preceding slice. If a range overlaps the kernel of some region, then that range is added to the region and the kernel updated by intersecting it with the new range. If a range does not overlap any previous region, it is used to start a new region. Note that, in general, more than one range in the new slice may overlap the kernel of a region in the previous slice. The implemented algorithm chooses the lower range (smaller values of q_2) to add to the region; the higher ranges are used to start new regions. By this construction, we guarantee that all the slices in a region share a common range of q_2 values; this is the kernel. In practice, we require some minimum overlap between slices in the same regions to avoid very narrow kernels.

Free-space regions are nonconvex and so points within the region may not always be connectable by a straight line. There is, however, a simple method for moving between points within the region: move from each point along its slice to the edge of the kernel and connect these kernel points with a straight line.

To search for a path between points in different regions requires representing the connectivity of the regions. We build a *region graph* where the nodes are regions and the links indicate regions with common boundary. Associated with each region is a set of links to adjacent regions, where each link records the location of the overlap in addition to the adjacent region. Regions have neighbors primarily in the q_1 direction; for these neighbors, the range of q_2 values at the common region boundary is stored with the link. By construction, regions only have q_2 neighbors at the $0 = 2\pi$ boundary. Anywhere else the region is bounded above and below by obstacles.

In general, each n-dimensional slice is represented as a list

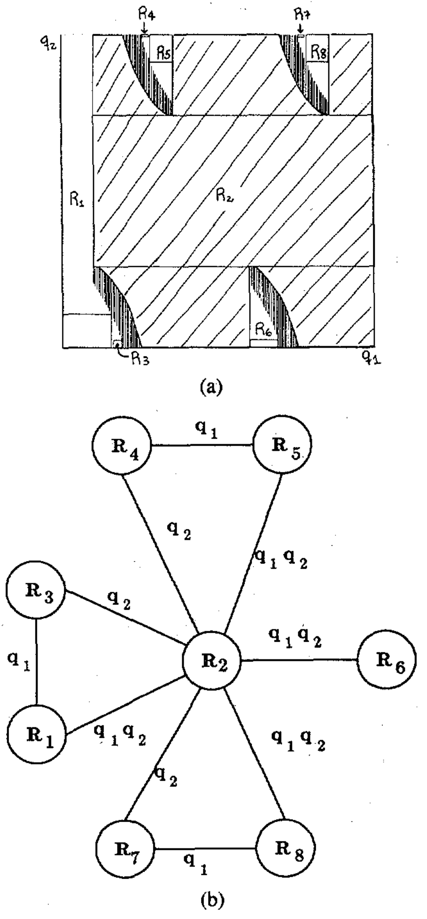


Fig. 14. (a) Regions for two-joint C space. Rectangles are region *kernels*. Hashed area shows region R_2. (b) Region graph corresponding to regions in part A. Link labels indicate existence of common boundary in q_1 and/or q_2 directions.

of $n - 1$-dimensional slices, and one-dimensional slices are a list of ranges of joint values. We have seen that two-dimensional regions are constructed by joining neighboring one-dimensional slice-projections. In principle, we could construct three-dimensional regions by joining neighboring two-dimensional regions, and so on. Instead, for three-dimensional C spaces we simply build two-dimensional regions for each range of values of the first joint parameter and represent the connectivity among these regions in the region graph (Fig. 15). The connectivity is determined by detecting overlap between region kernels in neighboring two-dimensional slices, that is, slices obtained by incrementing or decrementing the first joint parameter. When overlap exists, the area of overlap is associated with the corresponding link in the region graph. This method is readily extended to n-dimensional slices by considering as neighbors slices obtained by incrementing or decrementing one of the first $n - 2$ joint parameters used to define the two-dimensional slice.

The main feature of this region representation is that it exploits the coherence of the free space; thus, for example, it does not introduce many arbitrary divisions in the free space such as are introduced by octree-type representations [7]. Exploiting the natural coherence has a number of practical advantages. The main result is the compactness of the representation: very few regions are required to represent rather complex free spaces. Another important result is low branching in the region graph: each region has relatively few neighbors. These characteristics of the representation also make possible some of the heuristic search techniques described in Section VII.

VI. Searching for a Path in the Region Graph

In this section, a technique for searching a region graph is described. This technique applies to searching any subset of the C space; it is not necessary that the complete C space be examined before any searching is done. Section VII describes some heuristic strategies for limiting what parts of the C space are actually explored.

Path searching is done by an A^* search in the region graph from the region containing the start point to the region containing the goal point. During the search, a list of search nodes is kept. Each search node is associated with some intermediate region in the region graph and represents a set of regions connecting the start region to that intermediate region. For each node, we also keep track of an *entry point* on the region boundary that represents the location where the robot path would enter the region. When a search node is expanded by extending the region path to an adjacent region, the entry point is moved to the closest point on the common boundary between the two regions. The entry point to the next region becomes the *exit point* for the current region.

To carry out the search, we must associate with each search node an actual distance covered and an underestimate of the remaining distance to the goal. We use the distance between entry points to define the distance between two regions and the underestimate is the distance between the entry point and the goal. Of course, these distances are based on differences between the joint parameters modulo 2π. Once having found a list of regions connecting the start to the goal, the actual path is obtained by connecting the entry points and exit points of the regions. The entry point of the start region is the start point and the exit point of the goal region is the goal point.

A typical path found by the algorithm using the simple strategy described earlier is shown in Fig. 16. The paths tend to be jagged; some postprocessing to smooth the path would be desirable and is currently under investigation. On the other hand, because of the compactness and low branching of the region representation, searching for a path tends to be very fast (less than half a second for two-dimensional C spaces).

VII. Heuristic Subsets of the C Space

Having built a C space, it may be searched repeatedly for different paths. Changes to the environment, however, will cause parts of the C space to be recomputed. In rapidly changing environments, it may not be appropriate to compute the complete C space since only small sections of the C space will ever be traversed. This section describes experience with a number of simple heuristic strategies that help select the subset of the C space relevant to a particular path.

A. Decoupling the Degrees of Freedom

The path shown in Fig. 1 was computed using two simple heuristics to choose subsets of the C space. First plan a path

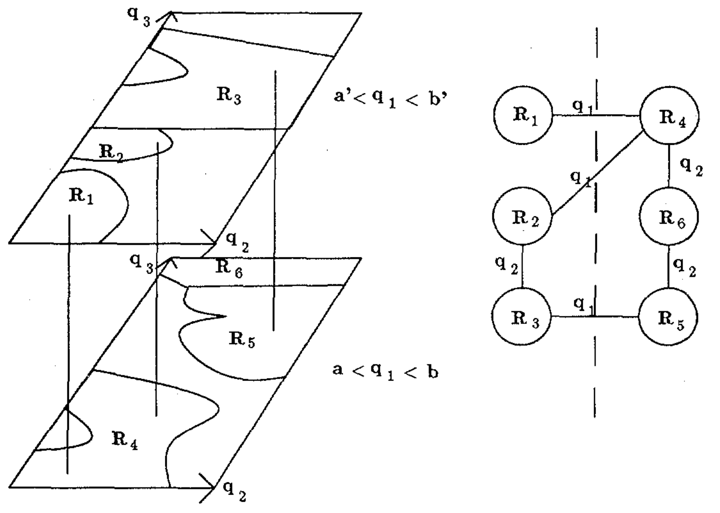


Fig. 15. Region connectivity for three dimensional slices; regions can have neighbors in q_1 direction.

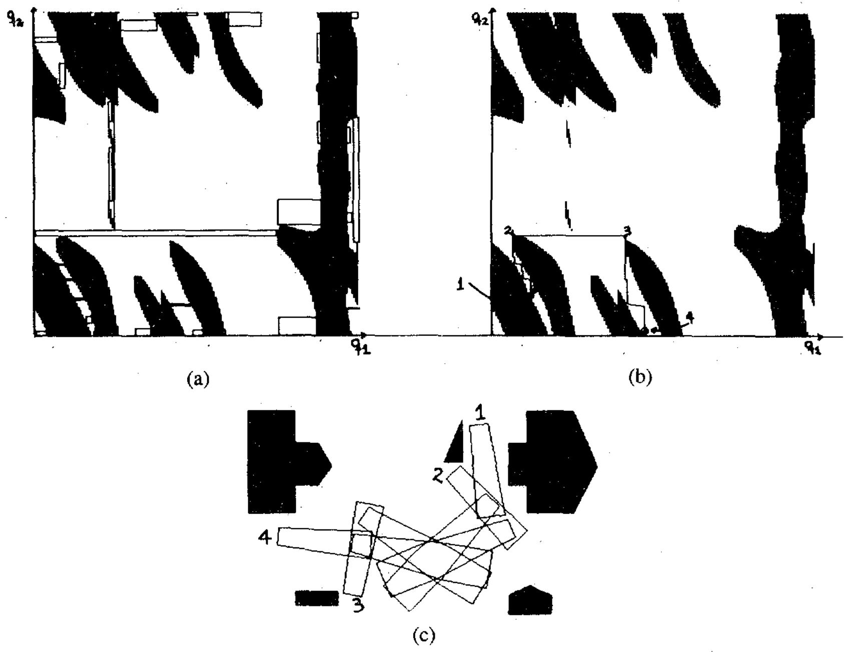


Fig. 16. (a) Regions kernels for example in Fig. 4. (b) Path found between start (1) and goal (4) configurations. (c) Some intermediate configurations.

for the first three links and a simple conservative approximation of the rest of the manipulator (the last three links, the end-effector, and the load), see Fig. 17. The origin and goal for this path are chosen to be the points in free space closest to the (projection of the) actual origin and goal. Note that these points may differ from the actual origin and goal in all of the joints. Having found such a path, there remains finding paths in the six-dimensional C space between the actual origin (resp. goal) and the origin (resp. goal) of the path. For all these paths, we compute only the portion of the C space bounded by the joint values of the origin and goal configurations.

This strategy has the effect of nearly decoupling the degrees of freedom. The six-dimensional planning is confined to the areas near the origin and goal. Of course, this strategy will fail to find a path in the worst case, but this strategy has proven to be reliable and efficient in most practical situations.

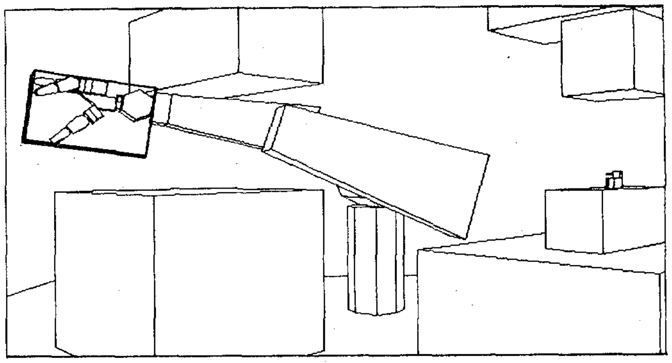


Fig. 17. First three links of manipulator of Fig. 1. Last three links and end effector have been replaced by simple bounding box.

B. Modifying Low-Dimensional Paths

One alternative approach for searching for a path from configuration $(q_1, \cdots, q_n)$ to configuration $(q_{g,1}, \cdots q_{g,n})$ is the following.

1) Ignore all but the first two joints. Build the free space for this reduced manipulator. Find a sequence of free space regions that contain a path from (q_1, q_2) to (q_1', q_2'). Let $i = 3$.
2) Expand the portion of the free space included in the regions found so far to incorporate the link i. That is, quantize the range of values in the region and use them to compute one-dimensional slice projections of the C space for the first i links.
3) Search for a sequence of free-space regions that contain a path from $(q_1, \cdots, q_i)$ to $(q_1', \cdots, q_i')$. If $i = n$ then stop; else increment i and to to step 2.

The idea behind this strategy is to focus on relevant sections of the configuration space by first finding paths for successively "longer" manipulators, starting with a two-link manipulator and going all the way through to a manipulator with n links.

It is important that at each stage we consider not just a single path of the "shorter" manipulator, but a sequence of regions that span all the free-space between a set of obstacles. The addition of link n will typically change the required path for the first $n - 1$ links. Therefore, the search space should include more than a single path so as to avoid the need to perform a backtracking search.

This method has been implemented. The path for the four-degree-of-freedom manipulator shown in Fig. 18 was found by this technique. The technique leads to significant time savings on problems involving more than two degrees of freedom. Of course, since the complete configuration space is not computed, the time to plan a subsequent motion in the same work space will be as long as that for the initial motion.

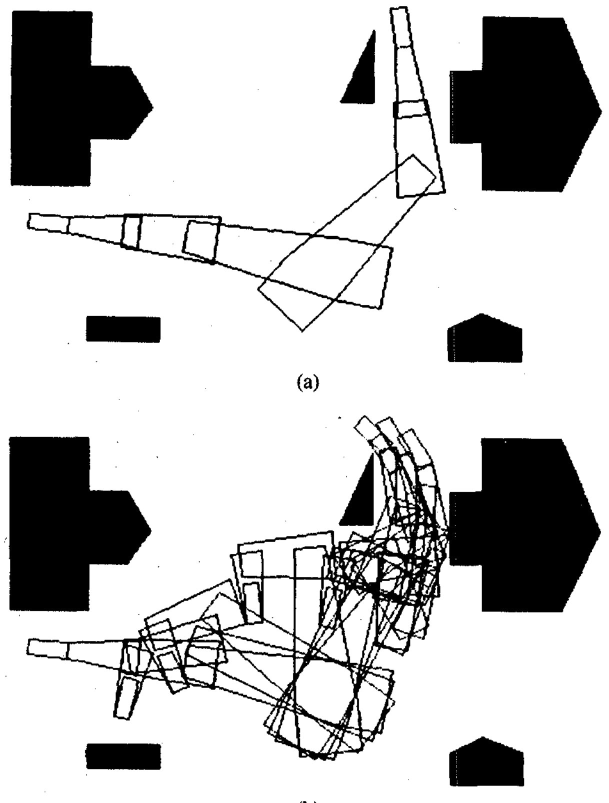


Fig. 18. (a) Initial and goal configurations for two-dimensional manipulator with four degrees of freedom. (b) Path found by algorithm in Section VII-B.

VIII. Discussion

The main advantages of the algorithm described here are 1) it is simple to implement, 2) it is fast for manipulators with few degrees of freedom, 3) it can deal with manipulators having many degrees of freedom including redundant manipulators, and 4) it can deal with cluttered environments and nonconvex polyhedral obstacles. The total wall-clock time to compute the C-space obstacles and then plan a path for the two-link example shown in Figs. 4 and 16 is 6 s on a Symbolics 3600 Lisp Machine with floating-point operations performed in software. These times could be improved by carefully recoding the algorithm and use of floating-point hardware, but they are already quite a bit faster than a human using an interactive programming system (on-line or off-line).

The main disadvantages of the algorithm are that the approximations introduced by the quantization may cause the algorithm to miss legal paths in very tight environments, and the rapid growth in execution time with the number of robot joints. This last drawback is probably inherent in any general motion planner; the worst-case time bound will be exponential in the number of degrees of freedom [19]. Certainly, the algorithm described here is exponential in the number of degrees of freedom of the robot. The execution time is dominated by the time to build the configuration space. The worst-case asymptotic complexity for building a complete (quantized) C space a robot with k degrees of freedom, in which the joint ranges are divided into r values, in which the robot description has m faces and edges, and in which the environment has n faces and edges is $O(r^{k-1}(mn)^2)$. In practice, as we pointed out earlier, except for $k = 2$ one never builds the complete C-space representation.

The performance of this algorithm shows that motion planning algorithms can be fast enough and simple enough for practical use. I believe that in many applications automatic motion planning will be more time effective than interactive off-line programming of robots. In fact, the planning times will probably be on the order of the times required to perform hidden surface elimination in graphics systems.

Appendix

Computing Contact Angles for Polyhedra

In what follows we assume that we are dealing with a convex polyhedron describing link k and an obstacle polyhedron (not necessarily convex). The coordinate system is chosen so that the origin corresponds to the position of revolute joint k and the z axis is aligned with the joint axis (Fig. 19). The coordinate representation of all vectors is relative to this coordinate system. We assume that the initial position of the link polyhedron corresponds to q_k is zero. We

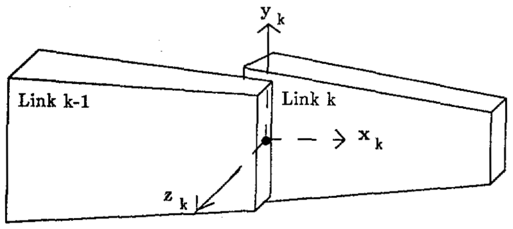


Fig. 19. Joint coordinate system.

are interested in computing values of q_k for which the link is in contact with the obstacle polyhedron.

Type B Contact: Vertex of Link and Face of Obstacle

We are given a vertex of the link whose position vector is $\boldsymbol{v}$ and an obstacle face whose plane equation is $\boldsymbol{n}\cdot\boldsymbol{x} + d = 0$ ($\boldsymbol{n}$ is the plane's outward-facing unit normal). We solve for the angle q_k that rotates the vector onto the plane. We obtain the equation for q_k by substituting the vertex's position, rotated by q_k, into the plane equation and solving for q_k.

The coordinates of the position vector for the rotated vertex are

$$v' = (v_x \cos q_k - v_y \sin q_k,\ v_x \sin q_k + v_y \cos q_k,\ v_z).$$

Substituting into the plane equation yields $\boldsymbol{n}\cdot\boldsymbol{v}' + d = 0$, this yields a simple trigonometric equation

$$(n_x v_x + n_y v_y) \cos q_k + (n_y v_x - n_x v_y) \sin q_k = -d - n_z v_z \quad (5)$$

whose solution is (6).

$$q_k = \cos^{-1}\left(\frac{-n_z v_z - d}{\sqrt{(v_x^2+v_y^2)(1-n_z^2)}}\right) + \arctan\,(n_y v_x - n_x v_y,\ n_x v_x + n_y v_y). \quad (6)$$

Equation (5) is the equation for a C surface of type B [14]. Note that if we let $n_z = 0$, then (5) and (1) are essentially identical. The arctangent simply computes the angle between the plane normal and the projection of v on the xy plane; this magnitude is analogous to ϕ in the planar case. Equations (6) and (2) are also related in the same way.

The left side of (5) represents the perpendicular distance of the rotated vertex from the obstacle plane. The sign of the derivative of this quantity with respect to q_k can be used to determine whether increasing or decreasing q_k causes a collision.

The orientation constraint simply requires testing whether the other endpoint of all the edges meeting at the contact vertex are on the outside of the plane. This is done by substituting the position vector of these endpoints into the left side of the plane equation and testing that the value is positive.

Type A Contact: Vertex of Obstacle and Face of Link

We are given an obstacle vertex whose position vector is $\boldsymbol{v}$ and a link face whose plane equation is $\boldsymbol{n}\cdot\boldsymbol{x} + d = 0$ ($\boldsymbol{n}$ is the plane's outward-facing normal). The solution for q_k is almost identical to the type A case, the only difference is the sign of the first argument to the arctangent. This reflects the fact that in type A contact we are treating the link as stationary and assuming the object is rotating in the opposite direction. This changes the sign of the sine of the angle.

Typc C Contact: Edge of Obstacle and Edge of Link

This case is substantially more difficult; we follow the derivation in [1]. We represent points on the edges parametrically in t. Therefore, points on the link's edge are represented by $t_l\boldsymbol{m} + \boldsymbol{v}$ where $\boldsymbol{v}$ is the position vector of one of the endpoints of the edge and $\boldsymbol{m}$ is a vector along the edge (actually the difference vector between the endpoints). The parameter $t_l \in [0, 1]$ parameterizes along the edge. We can represent the vector along the obstacle edge similarly as $t_0\boldsymbol{n} + \boldsymbol{w}$ for $t_0 \in [0, 1]$.

As the edge rotates around the z axis, points on the edge trace out circles. The equation for points on those circles are

$$x^2 + y^2 = (m_x t_l + v_x)^2 + (m_y t_l + v_y)^2$$
$$z = m_z t_l + v_z.$$

These can be combined by solving the second equation for $t_l = (z - v_z)/m_z$ and substituting into the first to obtain

$$x^2 + y^2 = \left(\frac{m_x}{m_z}(z - v_z) + v_x\right)^2 + \left(\frac{m_y}{m_z}(z - v_z) + v_y\right)^2. \quad (7)$$

This is an implicit equation for points on the rotation surface.

The parametric form of the obstacle edge can be used to solve for the intersection of the edge with the rotation surface:

$$x = n_x t_0 + w_x \qquad y = n_y t_0 + w_y \qquad z = n_z t_0 + w_z.$$

Substituting into (7) gives a quadratic equation in t_0.

Define the following terms:

$$p = (n_x^2 + n_y^2)m_z^2 - (m_x^2 + m_y^2)n_z^2$$

$$q = 2[(n_x w_x + n_y w_y)m_z^2 - (m_x^2 + m_y^2)(w_z + v_z)n_z - (m_x v_x + m_y v_y)m_z n_z]$$

$$r = (w_x^2 + w_y^2)m_z^2 - [(m_x(w_z - v_z) + v_x m_z)^2 + (m_y(w_z - v_z) + v_y m_z)^2].$$

The quadratic equation that must be solved for t_0 is

$$p t_0^2 + q t_0 + r = 0.$$

Having t_0 we can solve for t_l since we know that the z values at contact must be equal. Therefore,

$$t_l = \frac{n_z t_0 = w_z - v_z}{m_z}.$$

Given values for t_l and t_0, we must first check that they are in the range [0, 1] (the in-edge constraint), then we can compute points of intersection on each of the edges. Let $\boldsymbol{l}$ be the position vector of the intersection point on the link edge and $\boldsymbol{o}$ the position of the intersection point on the object edge. Then,

$$q_k = \arctan\,(l_x o_y - l_y o_x,\ l_x o_x + l_y o_y).$$

Note, however, that we have assumed, when deriving (7) that $m_z \neq 0$. In the not uncommon event that it is, then all the points on the rotation surface have $z = v_z$ and so will the intersection point with the obstacle edge. We can use this to obtain $t_0 = (v_z - w_z)/n_z$. We can then solve for t_l by using the fact that the contact point on the link edge will be on the same circle as the contact point on the obstacle edge:

$$(m_x t_l + v_x)^2 + (m_y t_l + v_y)^2 = (n_x t_0 + w_x)^2 + (n_y t_0 + w_y)^2.$$

However, we know the value of t_0 so this is a quadratic equation for t_l. Given the values of t_l and t_0 we can solve for q_k as before.

In addition to solving for the value of the contact angle we must compute the derivative of the distance to the contact as a function of q_k. As before, this will determine whether the contact angle is a potential upper or lower bound for a contact range. Unfortunately, this is not as simple as it is for type A and B contacts.

When the two edges are in contact, any motion component perpendicular to both of them will cause a collision while a component of motion along either edge will not cause a collision. The direction perpendicular to both edges is simply the cross product of the two edge vectors (given the link edge rotated to the contact angle):

$$c(q_k) = m(q_k) \times n.$$

The q_k dependence has been indicated explicitly.

One problem here is that we do not know whether c is outward pointing or not. We can decide that by dotting c with a direction known to point into the link. If e_1 and e_2 are the direction vectors of edges meeting the link edge at one of its vertices (see Fig. 20), then $e_1 + e_2$ is a direction pointing into the link volume. Let $k = \text{sgn}\,(c \cdot (e_1 + e_2))$, then kc is the outward-pointing normal we require (see [5] for a careful derivation).

Given the values of k, the type-C C-surface equation can be written as [14]

$$kc(q_k) \cdot (w - v(q_k)) = 0. \qquad (8)$$

Differentiating the left side with respect to q_k yields

$$((w_y m_x - w_x m_y) n_z + (d_z m_x - v_x m_z) n_y$$
$$- (d_z m_y - v_y m_z) n_x) k \sin q_k + ((w_y m_y + w_x m_x) n_z$$
$$+ (d_z m_y - v_y m_z) n_y - (d_z m_x - v_x m_z) n_x) k \cos q_k$$

where $d_z = v_z - w_z$. If this derivative is negative then increasing q_k will cause a collision.

We are not finished yet. We must guarantee that the contact satisfies the orientation constraint. The following are necessary and sufficient conditions for this [5]:

$$s = \text{sgn}\,(c \cdot e_1) = \text{sgn}\,(c \cdot e_2)$$

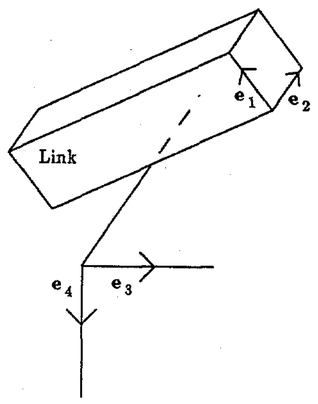


Fig. 20. Definition of e_i for $i = 1, 2, 3, 4$.

$$s' = \text{sgn}\,(c \cdot e_3) = \text{sgn}\,(c \cdot e_4)$$

$$s \neq s'.$$

These conditions are analogous to the type A and B cases.

Although the derivation of the type C case is a bit involved, the actual amount of computation involved is not large.

Acknowledgment

My thanks to Bruce Donald, Mike Erdmann, Rod Brooks, and Eric Grimson for their comments on an earlier draft of this paper. I also thank the reviewers for their helpful suggestions.

References

[1] J. W. Boyse, "Interference detection among solids and surfaces," *Comm. Assoc. Comput. Mach.*, vol. 22, Jan. 1979.

[2] R. A. Brooks, "Planning collision-free motions for pick-and-place operations," *Int. J. Robotics Res.*, vol. 2, no. 4, 1983.

[3] R. A. Brooks and T. Lozano-Pérez, "A subdivision algorithm in configuration space for findpath with rotation," in *Proc. 8th Int. Joint Conf. on AI*, Aug. 1983 (also *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-15, pp. 224-233, Mar./Apr. 1985; MIT AI Memo 684, Feb. 1983).

[4] J. F. Canny, "Collision detection for moving polyhedra," in *Proc. European Conf. AI*, 1984 (also MIT AI Memo 806, Oct. 1984).

[5] B. R. Donald, "Motion planning with six degrees of freedom," MIT AI Tech. Rep. 791, May 1984.

[6] E. Freund, "Collision avoidance in multi-robot systems," in *Proc. 2nd Int. Symp. Robotics Research*, Kyoto, Japan, Aug. 1984. Cambridge, MA: MIT Press.

[7] B. Faverjon, "Obstacle avoidance using an octree in the configuration space of a manipulator," in *Proc. IEEE Int. Conf. Robotics*, Atlanta, GA, Mar. 1984.

[8] L. Gouzenes, "Strategies for solving collision-free trajectory problems for mobile and manipulator robots," *Int. J. Robotics Res.*, vol. 3, no. 4, 1984.

[9] N. Hogan, "Impedance control: An approach to manipulation," presented at the American Control Conf., June 1984.

[10] O. Khatib and J. F. Le Maitre, "Dynamic control of manipulators operating in a complex environment," in *Proc. 3rd CISM-IFToMM*, Udine, Italy, Sept. 1978.

[11] B. H. Krogh, "Feedback obstacle avoidance control," in *Proc. 21st Allerton Conf.*, Univ. of Illinois, Oct. 1983.

[12] C. Laugier and F. Germain, "An adaptive collision-free trajectory planner," in *Proc. Int. Conf. Adv. Robotics*, Tokyo, Japan, Sept. 1985.

[13] T. Lozano-Pérez, "Automatic planning of manipulator transfer move-

ments," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-11, pp. 681–698, Oct. 1981 (also MIT AI Memo 606, Dec. 1980).

[14] ——,"Spatial planning: A configuration space approach," *IEEE Trans. Comput.*, vol. C-32, pp. 108–120, Feb. 1983 (also MIT AI Memo 605, Dec. 1980).

[15] ——,"Robot programming," *Proc. IEEE*, vol. 71, pp. 821–841, July 1983 (also MIT AI Memo 698, Dec. 1982).

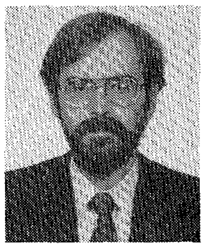
[16] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Comm. Assoc. Comput. Mach.*, vol. 22, pp. 560–570, Oct. 1979.

[17] C. O'Dúnlaing, M. Sharir, and C. K. Yap, "Retraction: A new approach to motion planning," in *Proc. 15th ACM STOC*, pp. 207–220, 1983.

[18] R. P. Paul, *Robot Manipulators*. Cambridge, MA: MIT Press, 1981.

[19] J. Schwartz and M. Sharir, "On the piano mover's problem II," in *Advances in Applied Mathematics*, 1983.

[20] S. Udupa, "Collision detection and avoidance in computer controlled manipulators," in *Proc. 5th Int. Joint Conf. AI*, Cambridge, 1977.



Tomás Lozano-Pérez (M'86) received the B.S., M.S., and Ph.D. degrees in computer science from the Massachusetts Institute of Technology, Cambridge, in 1973, 1976, and 1980, respectively.

Before joining the MIT faculty in 1981 he was on the research staff at IBM T. J. Watson Research Center during 1977, and at the MIT AI Laboratory during 1974 and again during 1980. He is presently Associate Professor of Computer Science and Engineering at MIT where he is a member of the Artificial Intelligence Laboratory. His research interests are in artificial intelligence and robotics; he teaches courses in these areas at MIT. He has written over 30 research papers mostly in robot manipulation and computer vision. He is coeditor of the *International Journal of Robotics Research*. He is coeditor of the book *Robot Motion* (MIT Press, 1982).

Dr. Lozano-Pérez was Program Chairman of the 1985 IEEE International Conference on Robotics and Automation. He is a recipient of a 1985 Presidential Young Investigator Award.