A SIMPLE MOTION PLANNING ALGORITHM FOR GENERAL ROBOT MANIPULATORS

Tomás Lozano-Pérez
MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Mass. 02139 USA

Abstract: This paper presents a simple and efficient algorithm, using configuration space, to plan collision-free motions for general manipulators. We describe an implementation of the algorithm for manipulators made up of revolute joints. 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.

1. 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 non-convex polyhedral obstacles. An example path obtained from an implementation of the algorithm is shown in Figure 1.

The ability to automatically plan 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 a 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].

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 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, 13, 14, 16, 17, 19, 20]. But, 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. I know of only three previous motion-planning algorithms that are both efficient and reasonably general for revolute manipulators with three or more degrees of freedom [2, 7, 12]. Brooks'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.

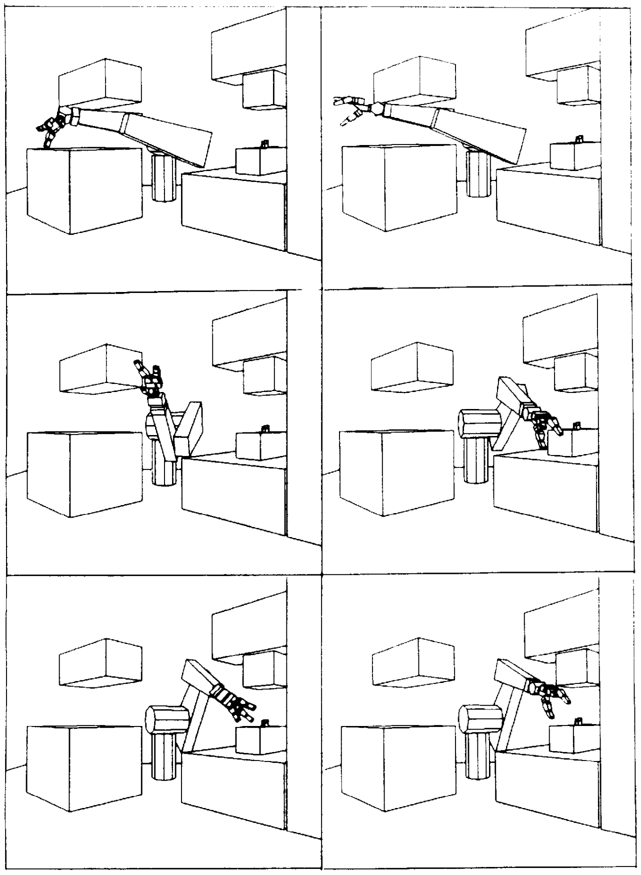
Figure 1. A path for all six links of a Puma, plus a three-fingered hand, obtained using the algorithm described here.

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 purpose of this paper is to show that motion planning for general manipulators can be both simple and relatively efficient in most practical cases. I see 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 Figure 1 it took about the same time to compute the hidden-surface displays in the figures as to compute the paths.

2. 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 map the obstacles in the robot's workspace into its configuration space [3, 4, 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. But, 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 non-branching 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 the $\mathbf{q} = (q_1, \ldots, q_n)$ denote a configuration, where each q_i is a joint parameter, each of which measures either angular displacement (for revolute joints) or linear displacement (for prismatic joints). Let $\{\mathbf{q} \mid q_j \in [\alpha, \beta]\}$ be the set of all configurations for which $q_j \in [\alpha, \beta]$ and let π_j be a projection operator such that

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

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

$$\pi_j(C \cap \{\mathbf{q} \mid q_j \in [\alpha, \beta]\})$$

The definition of slice projection is illustrated in Figure 2. In the example above, joint j above is called the *slice joint* while the other joints are known as *free joints*.

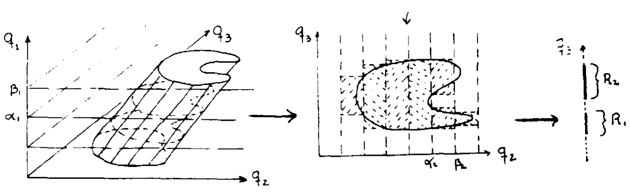

Figure 2. Slice Projection of a three-dimensional obstacle into a list of two-dimensional slices that are in turn represented by one-dimensional slices.

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 (Figure 2). 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 Figure 2. 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 $q_1 \in [\alpha, \beta]$ 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 ω, and a value $q_1 \in [\alpha, \beta]$, call it ς, for which $(\varsigma, \omega) \in C$, then ω is in one of the R_i (Figure 2).

A representation of a configuration space with obstacles is illustrated in Figure 3b, for the two link manipulator and obstacles shown in Figure 3a. 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 degrees along the q_1 axis.

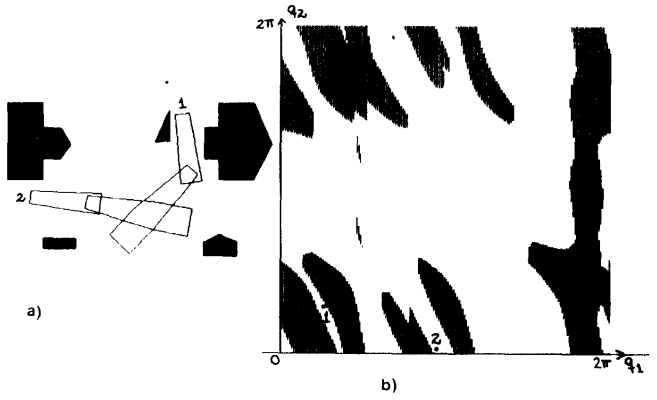

Figure 3. (a) Two link revolute manipulator and obstacles. (b) Two dimensional C-space with obstacles approximated by a list of one dimensional slice projections (shown dark). The initial and final positions of the manipulator are shown in the input space and the C-space.

For general manipulators with i links, the configuration space can be constructed as follows:

To compute C-space(i):

1. Ignore links beyond link i. 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 $q_1, \ldots, q_{i-1}$.
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 3.

The recursive nature of the C-space computation calls for a recursive data structure to represent the C-space. In my implementation I use 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 (Figure 4). 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.

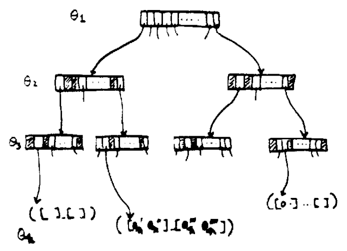

Figure 4. The recursive nature of the C-space leads to a recursive data structure: an n-level tree whose leaves represent legal ranges of configurations for the robot manipulator.

3. 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 planar revolute manipulators and obstacles.

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 3.3 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 2. 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 (Figure 5).

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 Figure 5. 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 Figure 5.

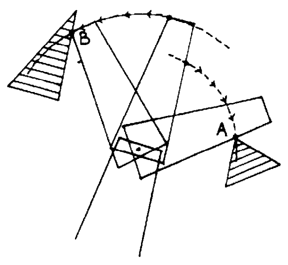

Figure 5. Contact conditions for computing one dimensional slice projections: (a) Vertex of obstacle and edge of link (b) vertex of link and edge of obstacle. The circles indicate the path of the vertices as the link rotates around the specified joint.

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 (Figure 6): *in-edge constraint* – the intersection point must be within the finite edge segment, not just the the line supporting the edge; *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; *reachability constraint* – for non-convex objects, there must not be other contacts that prevent reaching this point.

The in-edge constraint can be tested trivially given the 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.

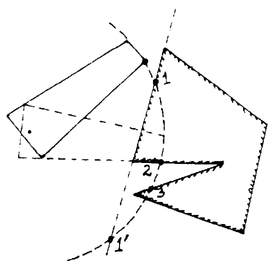

Figure 6. Given the intersection of a vertex circle and an edge line, the following conditions must be met for a feasible contact: (a) The contact must be in the edge segment, contact 1 satisfies this but 1′ does not (b) The edges that define the contact vertex must both be outside of the contact edge, contact 1 satisfies this but contact 2 does not. (c) The contact must be reachable, contact 1 satisfies this, but contact 3 does not (this condition is only relevant for non-convex objects).

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 3.2). 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 Figure 7.

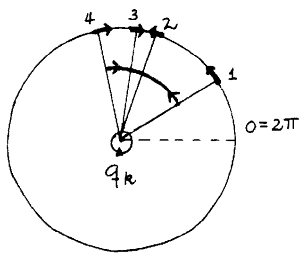

Figure 7. Constructing ranges of forbidden values using the potential contact angles and the collision directions.

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 crossection 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 magnitude, δ_k, of the largest cartesian displacement of any point on link k in response to any displacement within the specified range of joint values. If we "grow" each link by its corresponding radius δ_k, the grown link includes the swept area.

4. Slice Projections for Polyhedra

The basic approach described in Section 3 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. As in the two-dimensional case, possible contacts must satisfy three constraints to be feasible: in-face constraint - the contact must be within the obstacle face, orientation constraint - all of the link edges meeting at the vertex must be outside of the obstacle, and reachability constraint - for non-convex 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 require detecting the intersection of a line (supporting a link edge) rotating about the joint axis and a stationary line (supporting an obstacle edge). 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]).

5. 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.

The free space representation I use is made up of *regions*. A region is made up out of overlapping ranges from a set of adjacent slices (Figure 8). The area of common overlap of all the slices in a region is rectangular and called the region's *kernel*. In practice, we require some minimum overlap between slices in the same regions to avoid very narrow kernels.

Free space regions are non-convex 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 are a set of links to adjacent regions, each link records the area of overlap. 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.

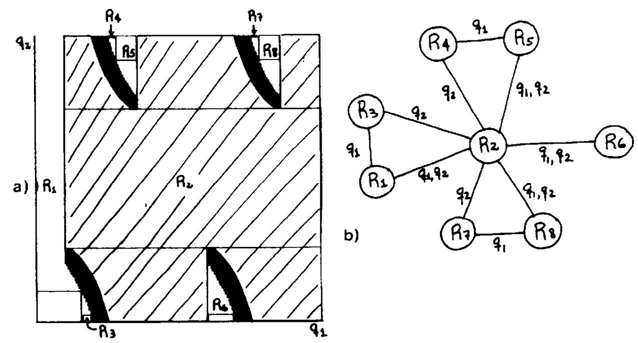

Figure 8. (a) Region definition for two link C-space. The rectangular regions are the region *kernels*. The shaded area shows region R_2. (b) Region graph corresponding to the regions in part A. The link labels indicate the existence of a common boundary in the q_1 and/or q_2 directions.

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 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 (Figure 9). 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.

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.

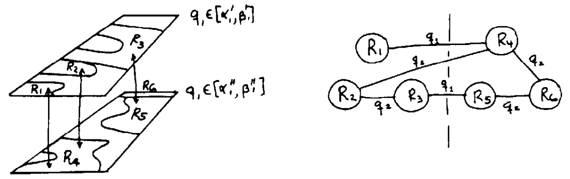
$q_1 \in [\alpha_1', \beta_1']$
$q_1 \in [\alpha_1'', \beta_1'']$
R_1
R_2
R_3
R_4
R_5
R_6
q_1
q_2

Figure 9. Region connectivity for three dimensional slices; regions can have neighbors in q_1 direction.

6. Heuristics for building 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.

The path shown in Figure 1 was computed using two simple heuristics to subset the C-space: First plan a path for the first 3 links and a simple bounding box for the rest of the manipulator (the last three links, the end-effector and the load). The origin and goal for this path are chosen to be the closest points in free space to the actual origin and goal. Having found such a path, there remains finding paths in the full-dimensional C-space between the actual origin (resp. goal) and the origin (resp. goal) of the path. This strategy has the effect of decoupling the degrees of freedom. For all these paths, we compute only the portion of the C-space bounded by the joint values of the origin and goal configurations.

7. Discussion

The main advantages of the algorithm described here are: 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 non-convex 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 Figure 3 and 10 is six seconds on a Symbolics 3600 Lisp Machine with floating-point operations performed in software. These times could be improved by carefully re-coding the algorithm, but they are already quite a bit faster than a human using an interactive programming system (on-line or off-line).

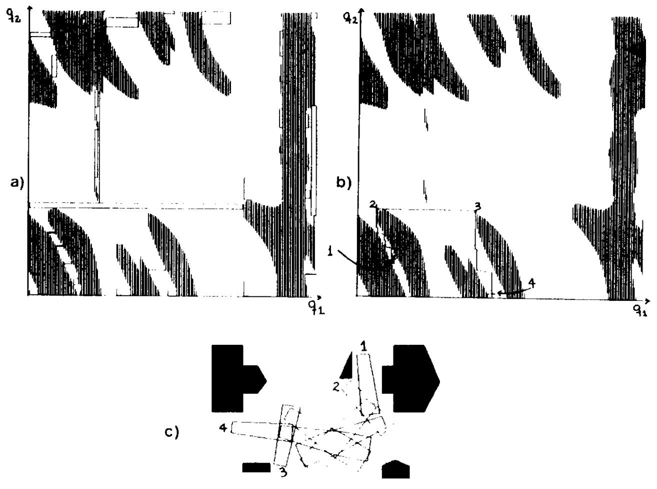

Figure 10. (a) Regions for example in Figure 3 (b) Path found between start (1) and goal (4) configurations (c) Some intermediate configurations.

The main disadvantages of the algorithm are: 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].

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.

Acknowledgments. This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence research is provided 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 the Office of Naval Research under contract N00014-82-K-0494. The author's research is also supported by an NSF Presidential Young Investigator grant.

Bibliography

1. J. W. Boyse, "Interference Detection Among Solids and Surfaces", *Comm. of ACM*, Vol. 22, No. 1, Jan. 1979.
2. R. A. Brooks, "Planning Collision-Free Motions for Pick-and-Place Operations", *Intl. J. Robotics Research*, 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. Eighth Int. Joint Conf. on A I, Aug. 1983. Also *IEEE Trans. on SMC*, Vol. SMC-15, No. 2, 224 -233, Mar/Apr 1985. Also MIT AI Memo 684, Feb. 1983.
4. J. F. Canny, "Collision Detection for Moving Polyhedra", Proc. European Conf. A I, 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", Proc. Second Intl. Symp. Robotics Research, Kyoto, August 1984. Published by MIT Press, Cambridge, Mass.
7. B. Faverjon, "Obstacle Avoidance Using an Octree in the Configuration Space of a Manipulator", Proc. IEEE Intl. Conf. Robotics, Atlanta, March 1984.
8. L. Gouzenes, "Strategies for Solving Collision-Free Trajectory Problems for Mobile and Manipulator Robots", *Intl. J. Robotics Research*, Vol. 3, No. 4, 1984.
9. N. Hogan, "Impedance Control: An Approach to Manipulation", Amer. Control Conf., June 1984.
10. O. Khatib and J. F. Le Maitre, "Dynamic Control of Manipulators Operating in a Complex Environment", Proc. Third CISM-IFToMM, Udine, Italy, Sept. 1978.
11. B. H. Krogh, "Feedback Obstacle Avoidance Control", Proc. 21st Allerton Conf., Univ. of Ill., Oct. 1983.
12. C. Laugier and F. Germain, "An Adaptive Collision-Free Trajectory Planner", Proc. Int. Conf. Adv. Robotics, Tokyo, Sept. 1985.
13. T. Lozano-Pérez, "Automatic Planning of Manipulator Transfer Movements", *IEEE Trans. on SMC*, Vol. SMC-11, No. 10, 681 - 698, Oct. 1981. Also MIT AI Memo 606, Dec. 1980.
14. T. Lozano-Pérez, "Spatial Planning: A Configuration Space Approach", *IEEE Trans. on Computers*, Vol C-32, No. 2, 108 - 120, Feb. 1983. Also MIT AI Memo 605, Dec. 1980.
15. T. Lozano-Pérez, "Robot Programming", *Proceedings of the IEEE*, Vol 71, No. 7, 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. of the ACM*, Vol. 22, No. 10, 560 - 570, October 1979.
17. C. O'Dúnlaing, M. Sharir, and C. K. Yap, "Retraction: A New Approach to Motion Planning", *15th ACM STOC*, 207 - 220, 1983.
18. R. P. Paul, *Robot Manipulators*. MIT Press, 1981.
19. J. Schwartz and M. Sharir, "On the Piano Mover's Problem II", Courant Inst. of Math. Sci. Tech. Rep. 41, Feb. 1982.
20. S. Udupa, "Collision Detection and Avoidance in Computer Controlled Manipulators", Proc. Fifth Intl. Joint Conf. AI, Cambridge, 1977.