Two-Phase Path Planning for Robots With Six or More Joints

P. E. Dupont*

S. Derby

Mechanical Engineering Department,
Rensselaer Polytechnic Institute,
Troy, NY 12181

This paper presents a new method for planning collision-free paths for robots with any number of joints. It is particularly well suited for use with kinematically redundant robots. The algorithm is general in that it does not impose restrictions on the geometry, motion, or payload of the robot. It does not try to locate an optimal path. Instead, it attempts to locate a reasonable path while mapping a minimal amount of configuration space (c-space). The method involves iteratively modifying a connected path between the initial and goal configurations to avoid all intervening obstacles. Information from the world model is used to guide path modification. This approach is of particular value in high-dimensional cases for which exhaustive searches are impractical. In the worst case, the algorithm maps a straight-line path in c-space to the goal and the surfaces of the interfering obstacles along this path. An example for a seven-degree-of-freedom robot is included.

1 Introduction

To perform a task, a robot must move its joints in a coordinated fashion so that its payload or tool moves from point to point or along a specified trajectory. During the motion, the entire robot, including its tool or payload, must not collide with any obstacles in the workspace. Potential obstacles include fixtures, workpieces, machines, and the robot itself. The goal of the path planner is to automatically develop a suitable collision-free path.

The inputs to the planner are models of the robot and its environment and a motion description. The robot model consists of a geometric description and a kinematic description (joint locations, orientations, and limits). The environment model is assumed to be a complete geometric description of the robot's static workspace. The motion description consists of a desired end-effector motion. In the simplest and most common case, this involves moving the end effector from its current location to another position and orientation. Its motion along the path is not constrained. In other cases, the end effector position and/or orientation along the path may be prescribed.

This paper is arranged as follows. The remainder of this section describes the benefits of redundancy. Section 2 discusses prior work in path planning. This includes a description of work by the authors which motivated the design of the algorithm detailed in Section 3. This is followed by a description of the implementation in Section 4 which includes an example.

1.1 Kinematic Redundancy. Six degrees of freedom are needed to position and orient a rigid body in space. These can be divided into three positional coordinates and three rotational coordinates. For a robot to arbitrarily position and orient its gripper in space, it must have at least six joints. When a robot has seven or more joints, it is kinematically redundant. This means that for most hand positions and orientations of a robot with n joints ($n>6$), there exists an $(n-6)$-parameter family of associated arm configurations.

For any particular application, it is best to use the simplest possible robot. Used unnecessarily, the additional joints of a redundant robot could mean higher costs and loss of rigidity and accuracy. There are three ways to use redundancy to advantage, however. The first of these is for joint limit avoidance. The joints of most, if not all, commercially available robots are not free to rotate continuously. In fact, most joints have a range of less than 2π radians. Redundancy can be used to avoid approaching these limits and thus to avoid the resulting motion constraints [1, 2].

Secondly, redundancy can be used for singularity avoidance. All robots, regardless of the number of joints, possess positions of singularity at which the actual number of degrees of freedom is less than six. One example is when several joints axes become aligned. In these configurations, large joint motions are required to achieve small hand motions. The extra degrees of freedom in a redundant arm can sometimes be used to avoid these configurations.

The third and most important use of redundancy is for obstacle avoidance. The increased dexterity of a redundant arm allows it to reach around, over, under and through obstacles as well or better than a human arm. This is of great value in cluttered or unplanned environments. Some exotic examples are the in-space repair of satellites and battlefield munitions resupply [3]. Other applications could involve flexible manufacturing workcells or operations on complicated workpieces such as reaching inside an automobile body.

Redundancy does make the task of path planning more difficult. The dimensionality of the problem is increased and the concept of arm separability can be hard to apply. Arm separability is the notion, widely used in path planning, that the arm can be decomposed into major and minor linkages. Most of the gross motion of the arm is attributed to the major linkage consisting of the first few joints (usually three) and

*Current Address: Robotics Laboratory, Pierce Hall, Harvard University, Cambridge, MA 02138.

Contributed by the Mechanisms Committee for publication in the JOURNAL OF MECHANICAL DESIGN. Manuscript received March 1988.

links of the robot. The remaining joints associated with the hand are ignored or frozen during most of the path planning process. In a well-designed redundant robot, it is probable that the major linkage will possess more than three joints and more than two sizable moving links [3]. Since problem difficulty seems to increase exponentially with degrees of freedom, a path planner for redundant robots should emphasize search efficiency over path optimality.

2 Historical Review

Many papers on path planning have appeared in the literature. While so much work has been done, progress toward developing efficient, general techniques has been slow. This lack of progress is not surprising since it appears that the computational complexity of the path-planning problem is exponential in the number of degrees of freedom [4, 5].

The most promising techniques developed so far fall into two categories. The first of these solves the path-planning problem by incrementally building a path from the initial robot configuration to the goal [6–16]. These typically use only local information about the environment. Consequently, they can be used on-line since the amount of computation at each step is small. Their major disadvantage is that they are heuristic methods which provide no guarantee that a solution will be found even if one exists. They can become stuck in stable, non-goal positions from which their escape requires additional algorithmic intelligence. While the methods described in [15, 16] are nonheuristic, they have yet to be fully implemented or extended beyond two dimensions.

The second category reduces the robot path-planning problem to that of finding a path for a point. The position of the entire robot can be determined from the values of the joint coordinates. This set of coordinates is called a configuration. In the space of configurations, called configuration space or *c*-space, the robot is represented by a point. If the joints have limits, these form the boundaries of the space. Otherwise, points with angular coordinates differing by multiples of 2π are identified with each other. These algorithms generally build sets of obstacle-free regions of *c*-space. A graph representing the connectivity of these sets is formed and searched for an optimal point path between the initial and goal configurations [17–29].

The difficult and time-consuming step in this approach involves mapping obstacles into *c*-space. As a result, most successful methods use arm separability and other simplifications [19, 21, 28].

Some papers have examined characterizing the surfaces of the transformed obstacles [27, 30]. Other works, while not computing the obstacle surfaces directly, do so indirectly by finding forbidden intervals for each link and building cellular representations of free space [21, 28]. For example, Faverjon [21] maps all obstacles into a discretized *c*-space for the first three joints. Starting from the base, obstacle-filled joint intervals for each link are recursively computed from a simplified robot model. From these, an octree representation of *c*-space is formed. The octree is searched for an optimal path which keeps the neglected robot's hand away from obstacles.

Lozano-Pérez presents a generalization of this algorithm in [28]. The algorithm produces some fast and impressive results. Like [21] it builds approximations of *c*-space obstacles from a series of one-dimensional forbidden ranges. To limit computation, it does not try to find a globally optimal path and so only maps that part of *c*-space bounded by the initial and goal joint values. For most of the motion, only the first three joints are used. The wrist joints are only allowed to move near the initial and goal points. This algorithm is fast for problems of low dimension.

Much of the success of these algorithms can be attributed to their pruning of the search space by reducing problem dimensionality. In cases where arm decomposition is not possible, such as with kinematic redundancy, we would like to structure our path search so as to avoid an exhaustive mapping. Ideally, we would like to map only that portion of *c*-space necessary to find a reasonable path or to conclude that no path exists at the resolution of the map.

Prior work by the authors has shown that a simple heuristic search in a discretized *c*-space can locate collision-free paths for seven-jointed robots operating in simple environments [31]. In this work a uniform discretization of seven-dimensional *c*-space is used. The equally sized cells correspond to the volume swept out by the robot as its joints vary between the limits of the cell. The algorithm builds a free-space path from initial to goal configurations composed of a set of contiguous, empty cells containing both points. The contents of a particular cell are only evaluated if it is being considered for the path. At that time, it is labeled full or empty by checking for interference between the robot's swept volume and the world model.

At each step of the algorithm, heuristics are used to select the best candidate cell for path extension. The heuristics, codified as sets of fuzzy production rules, include favoring the direction of the closest goal configuration, discouraging *c*-space direction changes and, when motion is impeded, moving perpendicular to and, if necessary, away from an obstacle. The algorithm incorporates backtracking and loop elimination. In the worst case, it will map the free-space region containing the initial configuration and those full cells which bound the region. In the average case, it was hoped that the planner would search efficiently and map just a small portion of this space.

In an environment of moderate complexity, the heuristics do not perform an efficient search. Erratic changes in search direction occur in high dimensions. The heuristics tend to emphasize aligning individual joints to their goal values as opposed to moving the entire arm to the proximity of the goal configuration. A conflict exists between moving towards the goal and avoiding obstacles which these heuristics are not powerful enough to resolve. The real weakness of this algorithm is that while knowledge of the entire world is available, only local information is being used. The algorithm differs from local-information, iterative methods only in that a map is compiled during the search. Therefore, it is not surprising that the method shares some of their drawbacks.

This experience indicates that any high-dimensional search algorithm must do two things. First, it must organize its search in a global sense. Secondly, it must develop search strategies based on its robot and world models. The method described below incorporates these ideas.

3 Two-Phase Planning: A String-Stretching Search Strategy

The basic method can be described using the analogy of a point path in *c*-space being represented by a string. The algorithm operates in two phases. In the first phase, a string is tightly connected between the initial configuration and the closest, obstacle-free goal configuration. At this point, the string is straight and may pass through obstacles. The path along the string is now checked to identify the segments which pass through empty space and those which pass through obstacles.

The second phase of the algorithm uses information from the world model to incrementally modify the path. The obstacle segments are stretched along the sides of the obstacles until they lie entirely outside of them. This process is depicted for a two-dimensional *c*-space in Fig. 1. The final modified segments lie on the $(n-1)$-dimensional surfaces of the obstacles.

In the second phase, the string could be moved around

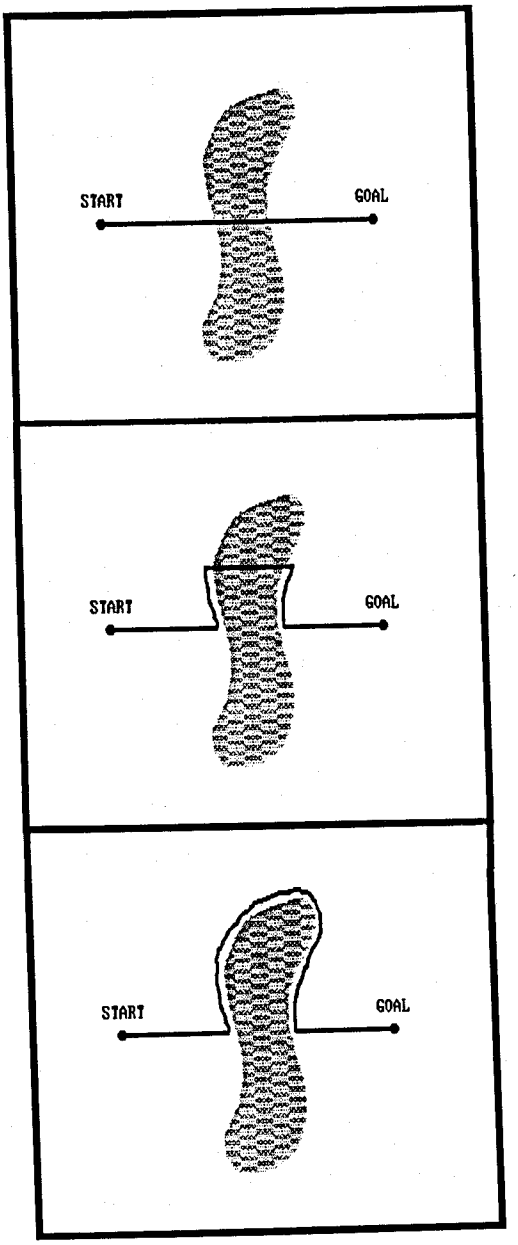

Fig. 1 Path modification by string stretching: (a) Straight-line path from phase 1 (b) Partially stretched obstacle segment during phase 2 (c) Final path following obstacle boundary

obstacles individually or as a group. It is generally easier to use a divide-and-conquer approach and avoid obstacles individually. In most applications of interest, path modification (string-stretching) can be first performed in the most promising direction. This direction can be chosen using robot and world model information and is called the strategy direction.

It is possible that no path exists for certain segments even when a path for the entire string exists. In these cases, it is necessary to modify the string for several obstacle segments at once. An example of this is shown in Fig. 2.

By using strategies developed from global information, the most promising regions of *c*-space are explored first. It is suggested that most problems can be solved using these strategies. In more difficult situations, a systematic exploration of the obstacle surfaces defining the segment must be performed since any path around an obstacle will lie on its surface or be in the same class of paths.

Failure occurs if the entire obstacle surface is explored and no passageway through it or around it is found for the string.

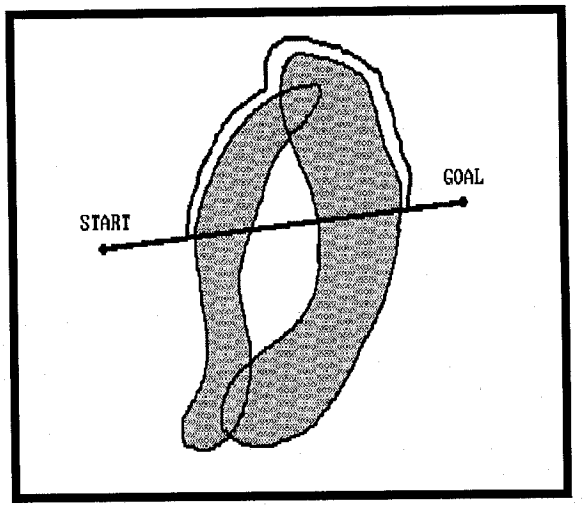

Fig. 2 The two obstacle segments must be combined to find the path shown

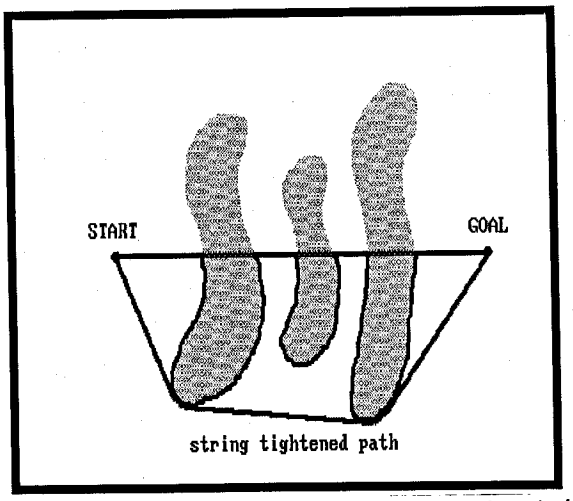

Fig. 3 Local path optimization by string tightening. The phase 1, phase 2, and string-tightened paths are shown.

Note that many potential goal configurations may exist due to redundancy and failure for one does not imply failure for all. When failure for a particular goal configuration occurs, the initial point is isolated from the region containing the goal by some combination of obstacles and joint limits. When selecting an alternate goal, care should be taken to choose one within the region of the initial configuration as determined by the current *c*-space map.

The string-stretching search strategy removes the conflict experienced by the authors' simple heuristic planner by separating the problem into two phases. The first phase forms a path connecting the initial configuration to the goal. The second phase modifies this path in a systematic and intelligent fashion until it is collision-free or it is determined that no solution exists. If desired, the collision-free path can be locally optimized by string-tightening as shown in Fig. 3. For this process, the obstacles are considered impervious to the string which is tightened between START and GOAL.

3.1 String Modification Strategies. In order to devise strategies for path modification, we wish to make full use of the information contained in the models of the robot and world. Ideally, this information is obtained through simple calculations by considering such things as:

- the approach and departure directions of the robot with respect to the obstacle,

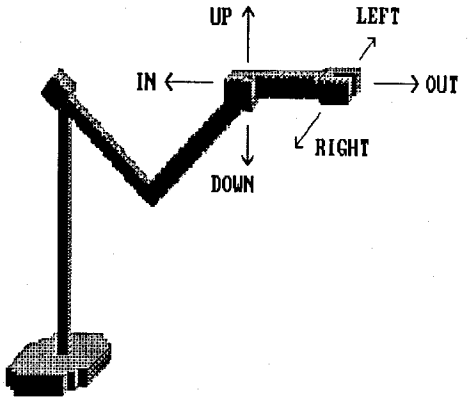

Fig. 4 Path modification strategies

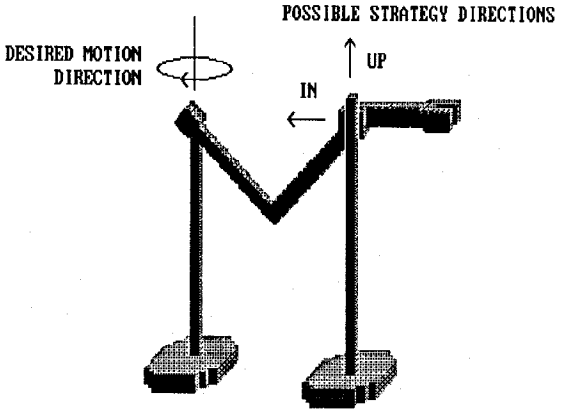

Fig. 5 Strategy selection example

- the relative location of the obstacle with respect to all the robot's links,
- the geometry of the obstacle,
- the proximity of other obstacles to the one in question and
- the location of the detected interference on the robot and on the obstacle.

A list of possible strategies can be developed for a robot based on its geometry. For an articulated arm, as shown in Fig. 4, these might be UP, DOWN, LEFT, RIGHT, IN, and OUT. These strategies are applied to those parts of the robot which cause interference. In *c*-space, they correspond to directions which depend on the current manipulator configuration and can be found using the Jacobian. The list of strategies can be reduced as follows:

- Eliminate those strategies disallowed by the obstacle. For example, a robot cannot move under an obstacle attached to the floor or inside an obstacle located near the robot's base.
- Eliminate or discourage those strategies which will lead other links into the obstacle.
- Eliminate or discourage those strategies which may lead any link into a nearby obstacle.

The best strategy of those remaining can be chosen by considering the interference locations on the robot and obstacle. Consider the example in Fig. 5. The arm configuration at which interference begins is shown. If the possible strategies are UP and IN, clearly UP is preferred. This can be deduced from the fact that the arm is extended radially well beyond the obstacle, but is near its top. Another consideration is any strategies previously selected for adjacent segments. If string-tightening optimization is to be used later, the final path may be shorter if the same strategy is used for adjacent segments.

If a single obstacle segment passes through several obstacles

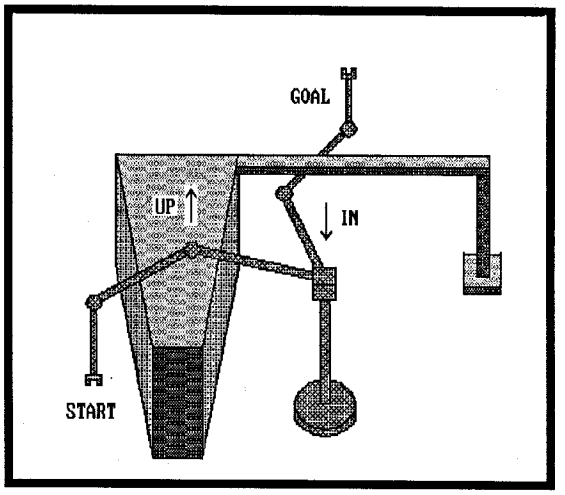

Fig. 6 Selecting a strategy for a complex obstacle shape

or segment modification introduces new obstacles, a strategy should be selected based on criteria for all of them. Strategy selection can seem difficult in some cases. Consider the example in Fig. 6. The START configuration in this figure suggests the use of the strategy UP. However, the arch over the GOAL seems to preclude the use of UP and suggests instead the use of IN. The wedge under the START configuration, however, seems to prevent the use of IN. This dilemma is solved by introducing a back-off direction as described in Section 3.3. The obstacle can be avoided by choosing either UP or IN as the strategy direction and the other as the back-off direction.

3.2 Two-Phase Planning in a Discretized Joint Space. Configuration space can be discretized into a hierarchical tree similar to an octree [32, 33]. For a robot with n joints, the tree would have a branching factor of 2^n. However, for ease of description and implementation, we use a uniform discretization in this paper. Each joint range is divided into equally sized increments. The resulting cells are called voxels, using octree terminology. Octrees and their properties are discussed in Section 4.1.1. Voxels correspond to the volume swept out by the robot as its joints vary between the voxel limits.

The swept volume of a link depends on the range of joint values for that link and for all links preceding it. This volume can be computed by a series of sweeps beginning with the link of interest and proceeding backwards to the base. The status of a voxel as empty or obstacle-filled space is determined by an interference check between the robot's swept volume and the world model. Smaller voxels yield a better representation of *c*-space, but, since a larger number must be used, require more computation. If an algorithm fails to find a path at a particular *c*-space resolution, it does not mean that one could not be found at a higher resolution.

In a discretized joint space, the tight string of phase one corresponds to a set of contiguous voxels enclosing the initial configuration, goal configuration and the straight line between them. Each of these voxels is checked for interference. During modification, the full voxels forming obstacle segments are replaced by sets of voxels lying on the $(n-1)$-dimensional, *c*-space obstacle surfaces.

3.3 Discretized-Space String Stretching. A brute force approach to string stretching would be to map with voxels the entire surfaces of interfering obstacles and to perform a graph search for the minimal length path. Instead, world model information is used to stretch the voxel "string" around obstacles in the most promising *c*-space directions.

In the discretized space, an obstacle segment includes the set

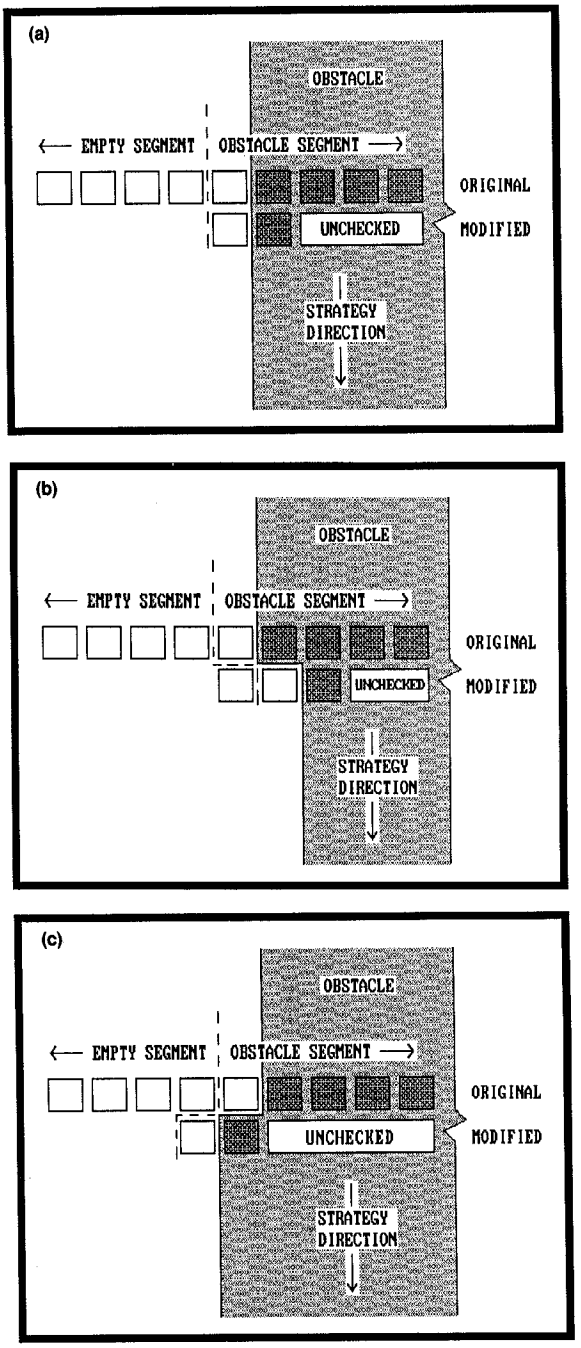

Fig. 7 Segment modification. While all voxels were checked for interference in phase 1, only the surface voxels are checked here. (*a*) Obstacle segment length is unchanged. (*b*) Length of obstacle segment is decreased by one voxel. (*c*) Motion in strategy direction requires backing off from obstacle. Segment length increases by one voxel.

of voxels between the first and last contact with an obstacle along the straight-line path. It begins and ends with the single empty voxels which precede and follow the set. In path modification, we wish to move an obstacle segment toward the edge of an obstacle in the strategy direction. In our discretized space, each voxel has $2n$ side neighbors. Since the direction into the obstacle is blocked and we would prefer not to backtrack, there are at most $(2n-2)$ possible directions in which to move the segment.

The direction is chosen to most closely correspond to the 3-D world strategy direction. Note that while the world strategy direction usually stays the same during segment modification, this is not necessarily true of the corresponding *c*-space direction since it depends on the current manipulator configuration.

When modifying a segment by string stretching, the idea is to maintain a contiguous set of voxels leading from one empty segment to the next. At each step of modification, the empty ends of the segment are stretched one voxel in the strategy direction and the new segment is shortened to maintain a single empty voxel on each end.

Figure 7 contains a two-dimensional example. There are three cases to consider. The neighbor in the strategy direction of the empty end voxel is checked first. If, as in Fig. 7(*a*), it is empty, it becomes the new end voxel. The original empty voxel is added to the end of the adjacent empty segment. The strategy direction neighbor of the first full voxel is now checked.

If it is full, then all checking for this end of the segment is complete. If it is empty, as in Fig. 7(*b*), then shortening is possible. This voxel becomes the new end voxel with its predecessor transferred to the adjacent empty segment. The next full voxel's neighbor must now be checked with this process continuing until the neighbor is full. This procedure trims the obstacle segment length.

If the neighbor of the original empty end voxel is full, as in Fig. 7(*c*), it is necessary to back off from the obstacle. This is done by adding empty voxels one at a time to the end of the original segment in a direction opposite blockage. In the figure, this means backing into the empty segment. Backing off is stopped as soon as the strategy direction neighbor is empty. If a back-off voxel is full, alternate back-off directions which move the interfering link away from the obstacle can be tried. Otherwise, the algorithm can backtrack to the previous step in segment stretching. If these fail, alternate strategies can be employed followed by a complete surface mapping.

Interference checking occurs only at the ends of the obstacle segments. The interior voxels remain in a straight line and are carried along only as a sequence of directions with respect to the segment end voxels. Since the interior voxels are never checked, string stretching has the effect of mapping a curve on the obstacle surface. The minimum number of interference checks during a modification step is four, covering the two empty ends and their full neighbors. These checks guarantee that the two ends of the path are empty and that the segment cannot be shortened. If any interference checks identify additional obstacles, the current strategy may need to be updated. Segment modification is complete when the obstacle-segment length goes to zero or the entire surface has been mapped.

3.4 Algorithm Performance. In the worst case, the string-stretching algorithm maps only the straight-line path to the goal and the *c*-space surface voxels of interfering obstacles along this path. This is an improvement over algorithms which compute the *c*-space bounding joint values for all obstacles in all cases. Given the same problem dimension, string stretching is comparable in the worst case to those methods which completely map the obstacle boundaries of the free-space region bounded by the initial and final joint values.

By simple analysis of the robot and world models, it is often possible to select the most promising search directions which we have called strategy directions. In many useful cases, a strategy directed search can drastically reduce the amount of obstacle-surface mapping needed to find a path. This is important since the most time-consuming part of path planning is often mapping. In addition, since strategies are devised from world model information as the easiest ways around obstacles, the paths generated from them are likely to be efficient.

4 Implementation

4.1 Solid Modeling. The planning algorithm was tested with two different methods for calculating interference be-

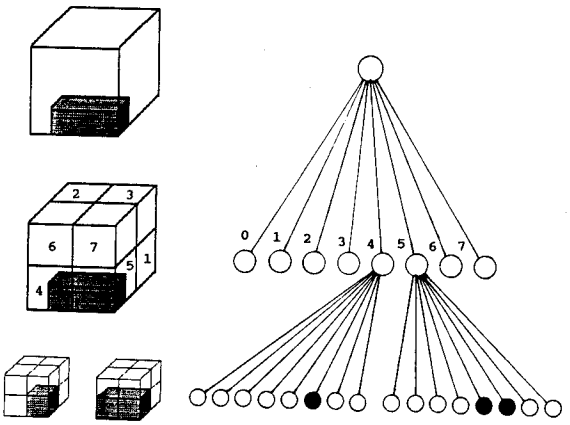

Fig. 8 Octree example

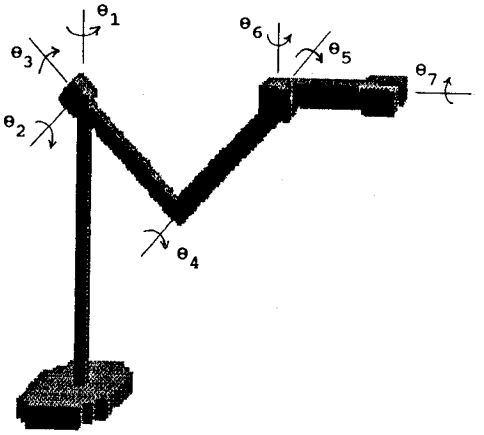

Fig. 9 Octree model of seven-jointed robot. Joint 3 increases arm dexterity by allowing it to move out of a vertical plane.

tween the robot and its environment. In the first, an exact swept-volume calculation was used with octrees to represent the robot and obstacles. The second method used a swept volume approximation with polytope models of the robot and workspace. Each method is discussed below.

4.1.1 Octree Method. An octree is a tree structure of degree eight. The octree universe is a cube which can be divided into eight octants. Each of these octants can be recursively subdivided to obtain any desired resolution. Within the tree structure, a node corresponds to a cube. If the cube is subdivided, the node has eight children corresponding to the eight octants. (See Fig. 8 for an example of a simple object and its octree). Nodes are labeled full, partial or empty, according to the cube's contents. Full or empty nodes are terminal while partial nodes have children. Cubes are called object elements or obels. Obels at the lowest level are called voxels.

Octree solid modeling has several advantages. Algorithms exist to perform all of the usual solid modeling operations on octrees using mostly integer arithmetic [32]. Real-time performance has been demonstrated on parallel processors [33]. Variable resolution operations can be performed with octrees simply by limiting tree descent to the appropriate depth.

Along with these advantages come some disadvantages. The number of nodes in the tree increases exponentially with the number of levels in the tree. This means that the time required for translations, rotations, and swept volumes is also exponential in the number of tree levels. Our implementation in *C* on a μVAX II was very slow for these operations. The interference, difference, and union operations were fast, however.

Another problem with the octree structure is that tree operations require rounding to full or empty at the voxel level. To be conservative, all partial voxels must be rounded to full. As a result, the object grows with each operation. This can be controlled by operating on the original octree whenever possible and by using a high-resolution tree. Since our implementation was so slow, use of a high-resolution tree was not possible.

Determining the contents of a *c*-space voxel involves computing the swept-volume octree of the robot corresponding to the voxel and checking for interference. To form the exact swept volume of link j requires j sweeps and $j-1$ concatenated translations and rotations. As discussed above, round-up error is introduced with each operation. If tree depth is not sufficient, the transformed object appears to have grown. This accumulated growth is greatest in the swept volumes of the distal links which require the most operations to generate. Because only eight levels were used, this growth was significant.

4.1.2 Polytope Method. While inherently approximate, this method was adopted for its high speed. The robot and obstacles are represented as unions of polytopes. A simple approximation to swept volume is used with a distance function to check for interference.

The swept volume approximation is similar to one described in [28]. Each link is placed at the mid-range joint value of the sweep and grown by the largest linear displacement associated with the sweep ranges. This is a conservative upper bound on the swept volume which is very easy to compute. Its accuracy is related to the size of the joint intervals.

For a planar robot with revolute joints, the growth radius, d_k, for link k is given by

$$d_k = \frac{1}{2}\sum_{i=1}^{k}\left[\left(\sum_{j=i}^{k-1} l_j + r_k\right)\sqrt{2(1-\cos\epsilon_i)}\right]$$

where ϵ_i is the sweep range of joint i, l_j is the distance from joint j to joint $j+1$ and r_k is the distance from joint k to the most distant point on link k. The term in square brackets corresponds to the length of a chord subtending an arc with included angle ϵ_i and a radius extending from joint i to the farthest point on link k. Since the *c*-space voxels are of constant size, d_k, $k=1 \ldots, n$ are also constant.

Since a distance function is used [34], the links are never actually grown by d_k. As long as the distance between link k and the obstacles is greater than d_k, there is no interference. Checking a *c*-space voxel's status consists of two steps. The vertices of each link's polytopes are transformed to place the robot at the mid-range joint configuration. Then the distances to the obstacles are computed and compared with the appropriate d_k.

The accuracy of the robot and obstacle models can be increased by using more polytopes or polytopes with more vertices. Both will increase solution time, but especially the former since pair-wise distance checking is performed.

4.2 Two-Phase Planning. Currently, the algorithm searches using only the best strategy. The Jacobian is used to find the *c*-space neighbor direction from the world strategy direction based on the current robot configuration. The strategy is applied to the link(s) causing interference.

For the case of a robot with n joints, our uniform discretization of *c*-space is equivalent to using a tree structure of degree 2^n and ignoring all nodes but the voxels. The number of voxels in a *c*-space with k joint subdivisions is k^n. For reasonable values of k and n, this number of voxels would consume a large amount of memory. However, we really only need to store voxels which are checked for interference. This number is proportional to the *c*-space surface area of the obstacles in the worst case. In most cases, it is much less.

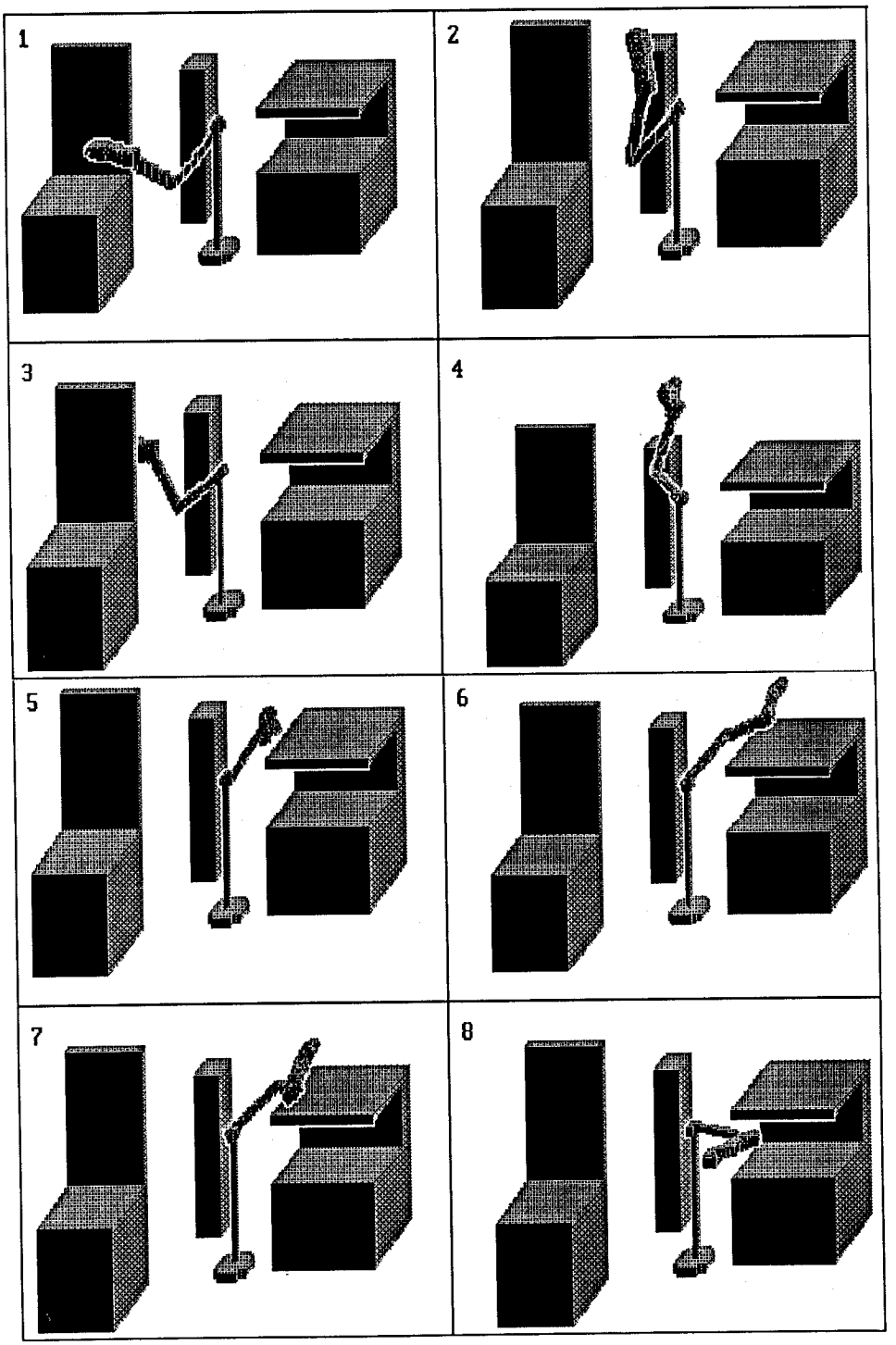

Fig. 10 Path example. All seven joints were free to move along the entire path. The coarse image of the robot, especially noticeable near the gripper, is due to the octree rounding error discussed in section 4.1.1.

4.3 Example. Paths were planned for redundant robots such as the one depicted in Fig. 9. The possible strategies were UP, DOWN, IN, OUT, LEFT, and RIGHT as shown in Fig. 4. An example path appears in Fig. 10. The limits on joint 1 forced the arm to rotate clockwise about the base to reach the goal. The strategies chosen for the obstacles from left to right were IN, UP, and UP. Notice that the path requires the configuration point to go outside the *c*-space region defined by the initial and goal joint values for joints 1, 2, and 4.

String-tightening optimization would improve this path. This can be seen by considering frames 2, 3, and 4. The configuration of frame 3 lies on the straight-line path of phase 1. String tightening would produce a more direct motion between the configurations of frames 2 and 4.

Table 1 lists performance data for this example. The discretization interval for all joints was 5 degrees. The number of voxels checked for interference was 119 in phase 1 and 235 in phase 2. These numbers should be compared with the total number of voxels in *c*-space which is greater than $2 \cdot 10^{12}$. As expected, a path was found while mapping a very small portion of *c*-space.

The execution times corresponding to the planning and in-

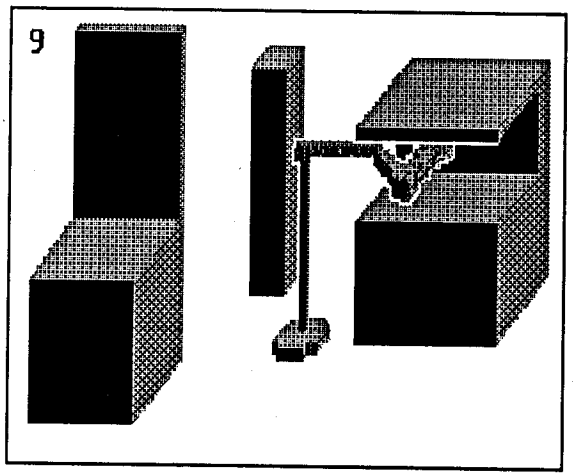

Fig. 10 (Continued)

Table 1 Example performance data

Voxel Edge Length = 5° Total Number of C-space Voxels > $2 \cdot 10^{12}$		
	Phase 1	Phase 2
Number of Voxels in Path	114	200
Number of Voxels Checked for Interference	119	235

CPU Time (Phases 1 & 2)		
	Polytope Method (Sun 4)	Octree Method (μVAX II)
Interference Checking	23.27 seconds	25 hours, 19 minutes
Planning Algorithm (μVAX II)	1 minute, 22 seconds	
Total Time	1 minute, 45.27 seconds	25 hours, 20.3 minutes

interference-checking portions of the code are listed separately. The planning time was the same for both modeling methods. Using the polytope approximation method, total CPU time was less than two minutes. Total CPU time using octrees was over 25 hours.

The solution time of the polytope approximation technique is excellent despite the fact that the code was not optimized. The expected high-speed performance of octree solid modeling was not achieved on our serial computer. The maximum tree level was limited to eight to keep the run times in hours. With only eight levels, the growing error resulting from multiple operations was significant. In fact, the accuracy was comparable to that achieved by the polytope approximation method.

5 Conclusions

A new algorithm for planning collision-free paths has been described. The basic algorithm can be used with any solid modeling representation of the robot and obstacles. Used with the polytope modeling scheme, it is fast.

For redundant robots, or any applications for which arm decomposition is not feasible, the algorithm provides the means to find reasonable paths while often performing fewer computations than other methods. This is possible because full use is made of the robot and world models to develop path modification strategies. The example in the previous section demonstrates that in a reasonably complicated case, a solution can be found using simple strategies.

While the algorithm is heuristic in that the most promising search directions are explored first, the search can continue until a path is found or it is determined that no path exists. The algorithm does not exhibit the erratic search behavior which characterized the authors' local heuristic planner. It also avoids the deadlocks experienced by iterative planners. This is accomplished by separating the planning problem into two phases. The first phase forms a path to the goal. The second phase modifies this path in a systematic and intelligent fashion to avoid all interfering obstacles.

In many instances, the algorithm avoids the complete *c*-space mapping inherent in many global, path-planning algorithms. In the worst case, it maps a straight-line, voxel path from the initial point to the goal and the *c*-space surface voxels of the interfering obstacles along this path. As our example demonstrates, the use of search strategies often eliminates the need to map the majority of these surface voxels.

Acknowledgment

We wish to thank Dr. Daniel W. Johnson for his suggestions and for providing the distance-function code used with the polytope method.

6 References

1 Liegeois, A., "Automatic Supervisory Control of the Configuration of Multibody Mechanisms," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. SMC-7, No. 12, December 1977.

2 Dupont, P. E., "Optimal Kinematic Control of Redundant Robots," Master's Thesis, M.E. R.P.I., 1984.

3 Martin Marietta Denver Aerospace, "Phase I-Intelligent Task Automation," Air Force Wright Aeronautical Laboratories, Technical Report AFWAL-TR-85-4062, Vol. 3, pp. 194-208, 214-215, April 1986.

4 Reif, J. H., "Complexity of the Mover's Problem and Generalizations Extended Abstract," *Proceedings of the 20th Annual IEEE Conference on Foundations of Computer Science*, pp. 421-427, 1979.

5 Schwartz, J. T., and Sharir, M., "On the 'Piano Movers' Problem II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds," Computer Science Technical Report No. 41, Courant Institute, New York University, February 1982.

6 Grechanovsky, E., and Pinsker, I., "An Algorithm for Moving a Computer-Controlled Manipulator While Avoiding Obstacles," *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Karlsruhe, West Germany, pp. 807-813, August 1983.

7 Harmon, S. Y., "The Ground Surveillance Robot (GSR): An Autonomous Vehicle Designated to Transit Unknown Terrain," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 3, pp. 266-279, June 1987.

8 Hirukawa, H., and Kitamura, S., "A Collision Avoidance Algorithm for Robot Manipulators Using the Potential Method and Safety First Graph," *Japan-U.S.A. Symposium on Flexible Automation*, pp. 99-102.

9 Hogan, N., "Some Computational Problems Simplified by Impedance Control," *Computers in Engineering 1984*, pp. 203-209, 1984.

10 Khatib, O., "Real-Time Obstacle Avoidance for Manipulators and Mobile Robots," *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO pp. 500-505, March 1985.

11 Krogh, B. H., "A Generalized Potential Field Approach to Obstacle Avoidance Control," *SME Conference-Robotics Research: The Next Five Years and Beyond*, Bethlehem, PA, SME Paper MS84- 484, August 1984.

12 Loeff, L. A., and Soni, A. H., "An Algorithm for Computer Guidance of a Manipulator in Between Obstacles," ASME *Journal of Engineering for Industry*, pp. 836-842, August 1975.

13 Maciejewski, A. A., and Klein, C. A., "Obstacle Avoidance for Kinematically Redundant Manipulators in Dynamically Varying Environments," *International Journal of Robotics Research*, Vol. 4, No. 3, pp. 109-117, Fall 1985.

14 Petrov, A. A., and Sirota, I. M., "Obstacle Avoidance by a Robot Manipulator Under Limited Information About the Environment," *Automatic Remote Control*, Vol. 44, No. 4, Pt. 1, pp. 431-440, April 1983.

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

16 Lumelsky, V., and Sun, K., "Gross Motion Planning for a Simple 3D Articulated Robot Arm Moving Amidst Unknown Arbitrarily Shaped Obstacles," *Proceedings of the IEEE International Conference on Robotics and Automation*, Vol. 3, pp. 1929-1934, 1987.

17 Andresen, F. P., Davis, L. S., Eastman, R. D., and Kambhampati, S., "Visual Algorithms for Autonomous Navigation," *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, pp. 856-861, March 1985.

18 Brooks, R. A., "Solving the Find-Path Problem by Good Representation of Free Space," *IEEE Transactions on Systems, Man and Cybernetics*, Vol. SMC-13, No. 3, pp. 190-197, March/April 1983.

19 Brooks, R. A., "Planning Collision-Free Motions for Pick-and-Place Operations," *International Journal of Robotics Research*, Vol. 2, No. 4, pp. 19-44, Winter 1983.

20 Chien, R. T., Ling Zhang, and Bo Zhang, "Planning Collision-Free Paths for Robotic Arm Among Obstacles," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. PAMI-6, No. 1, January 1984.

21 Faverjon, B., "Obstacle Avoidance Using an Octree in the Configuration

Space of a Manipulator," *Proceedings of the International Conference on Robotics*, Atlanta, GA, pp. 504–512, March 1984.

22 Gouzènes, L., "Strategies for Solving Collision-free Trajectories Problems for Mobile and Manipulator Robots," *International Journal of Robotics Research*, Vol. 3, No. 4, pp. 51–65, Winter 1984.

23 Hasegawa, T., "Collision Avoidance Using Characterized Description of Free Space," *'85 ICAR*, pp. 69–76, 1985.

24 Kambhampati, S., and Davis, L. S., "Multiresolution Path Planning for Mobile Robots," *IEEE Journal of Robotics and Automation*, Vol. RA-2, No. 3, pp. 135–145, September 1986.

25 Kuan, D. T., Zamiska, J. C., and Brooks, R. A., "Natural Decomposition of Free Space for Path Planning," *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO pp. 168–173, March 1985.

26 Laugier, C., and Germain, F., "An Adaptive Collision-Free Trajectory Planner," *'85 ICAR*, pp. 33–41, 1985.

27 Lozano-Pérez, T., "Spatial Planning: A Configuration Space Approach," *IEEE Transactions on Computers*, Vol. C-32, No. 2, pp. 108–120, February 1983.

28 Lozano-Pérez, T., "A Simple Motion-Planning Algorithm for General Robot Manipulators," *IEEE Journal of Robotics and Automation*, Vol. RA-3, No. 3, pp. 224–238, June 1987.

29 Udupa, S., "Collision Detection and Avoidance in Computer Controlled Manipulators," Ph.D. dissertation, Department of Electrical Engineering, California Institute of Technology, 1977.

30 Donald, B. R., "On Motion Planning with Six Degrees of Freedom: Solving the Intersection Problems in Configuration Space," *Proceedings of the IEEE International Conference on Robotics and Automation*, St. Louis, MO, pp. 536–541, March 1985.

31 Dupont, P. E., "A Simple Heuristic Path Planner for Redundant Robots," *Proceedings of the ASME 20th Biennial Mechanisms Conference*, Kissimmee, Florida, September 25–28, 1988.

32 Meagher, D., "Geometric Modeling Using Octree Encoding," *Computer Graphics and Image Processing*, Vol. 19, pp. 129–147, 1982.

33 Meagher, D., "A New Mathematics for Solids Processing," *Proceedings of the NCGA '84*, 1984.

34 Gilbert, E. G., Johnson, D. W., and Keerthi, S. S., "A Fast Procedure for Computing the Distance Between Complex Objects in Three-Dimensional Space," *IEEE Journal of Robotics and Automation*, Vol. RA-4, pp. 193–203, 1988.