Computing the Topology of Configuration Space

John J. Fox Anthony A. Maciejewski

School of Electrical Engineering
Purdue University
West Lafayette, Indiana 47907

Abstract—**In this work, an algorithm is developed for generating the connectivity graph for a class of articulated manipulators. The algorithm is based upon the ability to determine whether two distinct obstacles in configuration space intersect. The efficiency of the test which is developed lies in the ability to determine the intersection relation by evaluating the curves which describe the configuration space obstacles at only a small number of points.**

I. INTRODUCTION

A. Motivation

The problem of planning collision-free paths for manipulators has received an enormous amount of attention over the past decade. A large segment of this research derives from the seminal work of Lozano-Pérez on configuration space representations [4]. This approach can be grossly described as being comprised of two phases: the representation of those robot configurations which do not result in a collision, i.e. FindSpace, and the search for a path within this representation which connects the initial and final configurations, i.e. FindPath.

A standard approach to FindSpace has been to employ a structure called a "connectivity graph." In this representation, free space is partitioned into a set of path connected regions. A graph structure is then generated which represents these regions and their adjacency.

In this paper, a well known sweep line algorithm for generating the set of intersections for n line segments [7] is modified to generate the connectivity graph for a class of planar manipulators. The algorithm is based upon the development of an efficient method for determining whether two cspace obstacles intersect. Although a seemingly restrictive class, by adding an orthogonal axis to the manipulators being studied, the fundamental results of the work remain the same but now a significant number of the manipulators found in industry can be modeled.

B. Relationship to Previous Work

The notion of representing free space via a structure similar to a connectivity graph is, perhaps, the single most widely investigated approach to the solution of the FindSpace problem [3]. This approach is originally found in [8] in which free space is partitioned based upon the types of contacts which the robot can make with its environment.

The approach described by this paper is similar to those summarized in [3] however, it is novel in that there is no explicit calculation of free space, a process which is extremely time consuming. Instead, the manipulator's free space is described implicitly by characterizing a topological property of obstacles, namely, their connectivity.

The algorithm presented here is also strongly related to work in which the boundaries of configuration space obstacles are analytically described [1], [6]. The fundamental difference in the mathematical representation used here is that the curves describing configuration space obstacles are not only described by their positions but also by the tangents to these curves.

II. PRELIMINARIES

In this section, an algorithm for testing for intersections between cspace obstacles is derived. Details on the algorithm, its limitations and an application to planning the motions of articulated manipulators are found in [5].

A. An Intersection Test for Cspace Obstacles

Consider the obstacles labeled F and G in Fig. 1. For the reasons discussed in [5], an intersection test need only concern itself with contact with both obstacles along the second link. For this to occur, the orientation of the second link must be parallel to the line connecting the two obstacles. Since the orientation of the second link is determined by $\theta_1 + \theta_2$, this condition leads to the constraint

$$\theta_1 + \theta_2 = \tan^{-1} \frac{y_F - y_G}{x_F - x_G}. \quad (1)$$

Thus if the configuration space obstacles F and G intersect, they must do so somewhere along these lines. Therefore, to check for a possible intersection, one can simply check to see if both curves representing the obstacles F and G intersect with the lines defined by (1).

Rather than attempting to explicitly calculate the intersection points of the cspace obstacles with the lines defined in (1), a task which is nontrivial, a test will be developed for determining whether or not such an intersection exists. Recall that the minimum and maximum

Manuscript received Aug. 1, 1992. This work was supported by the National Science Foundation under grant CDR 8803017 to the Engineering Research Center for Intelligent Manufacturing Systems.

0-7803-0720-8/92 $3.00

distances from a curve to a line will occur at the points along the curve at which the tangent matches the slope of the line. It has been shown in [5] that the tangent to a configuration space obstacle is given by

$$m = \frac{-l_2}{l_2 + l_1 \cos\theta_2}. \quad (2)$$

Thus, setting the tangent equation of the configuration space curve equal to the slope of the constraint equation (1), results in

$$\frac{-l_2}{l_2 + l_1 \cos\theta_2} = -1 \quad (3)$$

the solution of which is given by $\theta_2 = \pm\pi/2$. Therefore, to determine if the lines defined by (1) intersect a configuration space curve one must evaluate the curve at only two points, i.e. those at $\theta_2 = \pm\pi/2$, and check to see if they bracket any of the lines defined by (1). If the value of l_2 is greater than the actual link length at $\theta_2 = \pm\pi/2$ then the end points of the curve should be used in their intersection test. Proof of the necessity and sufficiency of this condition can be found in [2].

B. Assumptions and Terminology

Throughout the description of the algorithm, the following terminology and conventions will be used. A particular point (Θ_1, Θ_2) in free space has an ordered pair of ***bounding obstacles***. These obstacles are defined by intersecting all of the obstacles in the environment with the line $\theta_2 = \Theta_2$. The obstacle whose intersection with this line takes place at the maximal θ_1 such that $\theta_1 < \Theta_1$ is termed the *left bounding obstacle*. If no such obstacle exists, then the boundary of cspace corresponding to the minimum along the θ_2 axis will be considered the left bounding obstacle. The *right bounding obstacle* is defined in an analogous manner. Calculating the bounding obstacles can, in general, be performed once the extrema of the cspace obstacles are known and the pairs of intersecting obstacles have been determined. It does not require the evaluation of all of the obstacles at a particular Θ_2.

A *region* will be defined as a subspace of free space for which every point has the same pair of bounding obstacles and, given any two points in the region, a collision-free path exists which does not consist of points from another region. An implication of this definition is that multiple regions which have the same pair of bounding obstacles may exist. Two regions, R_1 and R_2, will be considered to be *adjacent* if for any points $p_1 \in R_1$ and $p_2 \in R_2$, a collision-free path exists between p_1 and p_2 which consists entirely of points in $R_1 \cup R_2$.

Finally, for the purposes of illustration, the manipulator has been assumed to have the arbitrary joint limits of $\pm\pi$, however, this work is independent of this assumption.

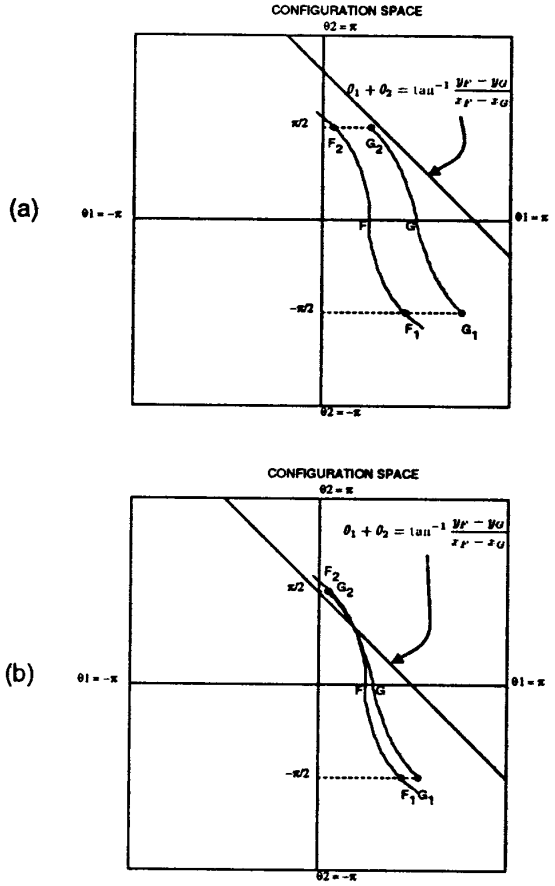

Fig. 1 Examples of the application of the intersection test. Part (a) shows an example of the test failing, and part (b) shows it succeeding.

III. DESCRIPTION OF THE ALGORITHM

A. Sweep Line Algorithms

A particularly efficient approach to certain problems in computational geometry is the Sweep Line Algorithm [7]. Although the details of applying this technique vary from application to application, the basic principals remain the same. Namely, a hypothetical line is swept through the data thereby dividing the problem domain into two subspaces. These subspaces are distinguished by the fact that the solution which has already been established in one subspace is unaffected by the generation of the solution to the other halfspace. The principal advantages of the sweep line approach is that it provides a fairly straightforward mechanism for organizing geometric data and, typically, the resulting algorithms are fairly efficient.

The principal mechanism for establishing the flow of control within such an algorithm is an event queue consisting of those points at which the sweep line will halt.

In the context of this paper, these events are those configurations at which new regions must be introduced and are, therefore, the upper and lower extrema of the configuration space obstacles and the points at which obstacles intersect.

The approach taken in this paper will be a variation on the sweep line approach and is based upon an algorithm for determining the set of intersections of a group of line segments [7]. Although the general approach taken will be that of sweeping a line through the data and generating the connectivity graph, the algorithm will not have the property that the line partitions the data space into two subspaces which have the properties described above. This is a direct result of not having explicitly calculated the intersection points of obstacles. Rather, as will be shown, the information used to determine the intersection predicate as defined in Section II will be used to establish relative information regarding the θ_2 abscissa of the intersection point.

Throughout the remainder of the paper, events corresponding to the upper extrema of obstacles will be referred to as split events. Reaching a lower extrema of an obstacle will be termed a merge event, and, finally, the intersection of two obstacles will be referred to as an intersect event. Pseudo code for split events and intersect events is found in Figs. 2 and 3, respectively. Although a merge event is sufficiently similar to a split event that it does not require an independent explanation, it should be noted that in some ways it is considerably simpler since it doesn't require the introduction of intersection events. Although the pseudocode which has been provided has assumed that the θ_2 abscissa of all of the events are distinct, it is a straightforward modification to handle the more general case.

There are four data structures which have been employed in the pseudocode provided: locations, nodes, obstacles and the event queue. Locations are points in cspace represented as two dimensional vectors. Nodes are structures which correspond to nodes in the connectivity graph. They consist of three data fields: the left and right bounding obstacles, LBO and RBO; and a list of the adjacent nodes. Obstacles can, for all practical purposes, be considered to be integers corresponding to obstacles in the workspace. Finally, the event queue is implemented as a LIFO stack.

Within the pseudocode, the existence of a number of subroutines has been assumed. The simplest of these, **BuildEvent()**, combines the pointers to the two intersecting obstacles into a structure suitable for being placed on the event queue. **FindBoundingObs()** returns the two bounding obstacles of the region corresponding to the input node. This operation is performed primarily by examining the extrema of the cspace obstacles and through knowledge of the existence of intersections between obstacles. **LeftNode()**, **RightNode()** and **CorrectNode()** return the nodes to which the algorithm is about to add children. The principal information used to determine this node is a table consisting of a list of each region which appears along the right or left side of a particular obstacle. Choosing the correct node is performed by incorporating knowledge of those intersection events in which the obstacle played a part.

Prior to beginning the line sweep, a preprocessing stage calculates the extrema of each obstacle and the information required for performing the intersection test of Section II. Sorting the extrema in order of decreasing θ_2 yields the initial event queue. Physically, this corresponds to having swept through the data without having considered the possibility of intersecting obstacles. Finally, the connectivity graph is initialized with a single node corresponding to the northern highway.

```
SplitEvent(Obs, Loc)
  begin
    [LeftObs, RightObs] = FindBoundingObs(Loc);
    Node = CorrectNode(Obs, LeftObs, RightObs);
    Left.LBO = LeftObs;   Left.RBO = Obs;
    Right.LBO = Obs;   Right.RBO = RightObs;
    Left.Adj = {Node};   Right.Adj = {Node};
    Node.Adj = Node.Adj ∪ {Left, Right};
    IntersectSet = {Obstacle such that
      Obstacle ∈ PrevObstacles and
      Obstacle intersects Obs};
    Sort IntersectSet
    for each Intersect ∈ IntersectSet
      Push( BuildEvent(Intersect), EventQueue)
    PrevObstacles = PrevObstacles ∪ {Obs};
  end
```

Fig. 2 Pseudocode for the splint event procedure.

```
IntersectEvent(LeftObs, RightObs)
  begin
    OldLeftNode = LeftNode(LeftObs);
    OldRightNode = RightNode(RightObs);
    Left.LBO = OldLeftNode.LBO;
    Left.RBO = RightObs;
    Middle.LBO = RightObs;
    Middle.RBO = LeftObs;
    Right.LBO = LeftObs;
    Right.RBO = OldRightNode.RBO ;
    Left.Adj = {OldLeftNode};
    Right.Adj = {OldRightNode};
    OldLeftNode.Adj = OldLeft.Adj ∪ {Left};
    OldRightNode.Adj = OldRightNode.Adj
      ∪{Right};
  end
```

Fig. 3 Pseudocode for the intersect event procedure.

Before proceeding, consider for a moment the implications of placing intersection events on the event queue immediately after processing the split event. The advantage of placing the events on the queue in this manner is that it avoids the issue of explicitly calculating the intersection points. Sorting the events with the approach described in Appendix A ensures that the intersection events for a particular obstacle are sorted by the θ_2 abscissa of the intersection point, however, it provides minimal information regarding the relative location of the intersection point with respect to either the extrema of other obstacles or intersections which do not directly involve the newly introduced obstacle. This has two principal effects.

First, under certain conditions bounding obstacles of a particular region may be impossible to determine without ambiguity. Fortunately, the nature of the configuration space obstacles is such that there can never be more than two different possible sets of bounding obstacles for a particular region which is generated by this algorithm. Rather than attempting to disambiguate between the two alternatives by iterating along one of the bounding obstacles, the region is labeled as ambiguous and is resolved only if needed for the FindPath procedure. The key piece of information which is obtained is that the region exists and can serve a part in generating a path, not necessarily the fact that its has some particular pair of bounding obstacles.

The second problem is a bit more subtle and does not affect the final outcome of the algorithm. In particular, it is possible that an intersection event will be processed prior to the introduction of an obstacle which should serve as a bounding obstacle for the nodes which result. The only effect of this is that the bounding obstacles of the incorrect node and its children must be modified and an edge must be redirected within the graph.

IV. IMPLEMENTATION RESULTS

The purpose of this section is twofold. First, a simple example is provided which illustrates the nature of the algorithm and demonstrates the effects of processing the different events. Second, complexity results and some timing results are provided so that the reader can better evaluate the algorithm.

A. A Simple Example

Figs. 4 (a)-(f) illustrate the execution of this algorithm on a simple example. In Figs. 4 (a) and (b), the obstacles have been drawn with a dashed line to reflect the fact at this stage they have not yet been reached by the sweeping line. In Figs. 4 (b)-(e), the horizontal dashed line represents the location of the sweep line corresponding to the event which has just been processed. Fig. 4 (f) shows the resulting connectivity graph. In this figure the dashed lines illustrate the partitioning of free space into regions.

To illustrate the processing that is required, we shall consider in detail the processing which occurs between the situation depicted in Fig. 4(b) and the situation in Fig. 4(c). As shown in Fig. 4(b), the algorithm has just finished processing the split event corresponding to the introduction of Obstacle 1. The result of this operation was the addition of nodes 2 and 3 to the connectivity graph along with the appropriate edges.

At this point, the next event in the queue is a split event corresponding to the upper extrema of Obstacle 2. The left and right bounding obstacles, *LeftObs* and *RightObs*, are determined to be T1_MIN and Obstacle 1, respectively. As a result, the split event will introduce two nodes which are children of the node currently corresponding to this pair of bounding obstacles, i.e. Node 2. Fig. 4(c) shows the definition of these two new nodes.

Next, the subset of obstacles which are already under consideration and which intersect the new obstacle are determined. In the example provided, the only intersection which takes place is with Obstacle 1. In general, this set would then be sorted as discussed in Appendix A and intersection events pushed onto the event queue. Finally, some bookkeeping takes place with regards to updating the **CurrentNode** and **PrevObstacles** information. When the processing is completed, there is an intersection event corresponding to the intersection of Obstacles 1 and 2 on the top of the event queue. The algorithm would then proceed in a similar manner until the event queue is empty, resulting in the connectivity graph shown in Fig. 4(f).

B. Worst-Case Time Complexity

The worst-case input for this algorithm can be seen to be one in which each obstacle intersects every other obstacle. Assuming there are n obstacles, this scenario results in n **SplitEvent**()'s, $\frac{n(n-1)}{2}$ **IntersectEvent**()'s, and n **MergeEvent**()'s. Executing **IntersectEvent**() can be performed in $O(n)$ time. Executing **MergeEvent**() can be performed in $O(n)$ time, since this routine is dominated by the amount of time required to determine the bounding obstacles of a location in cspace. In the worst case, this requires examining information regarding each of the obstacles. Executing the j'th **SplitEvent**() can be performed in $O(j \log j)$ time, since it is dominated by sorting the set of intersection events. Also, the algorithm requires an $O(n \log n)$ preprocessing stage for generating the information required to perform the intersection test and to generate the initial event queue. Hence, the overall worst case time complexity of the algorithm is $O(n^3)$ and is dominated by the processing of the intersection events.

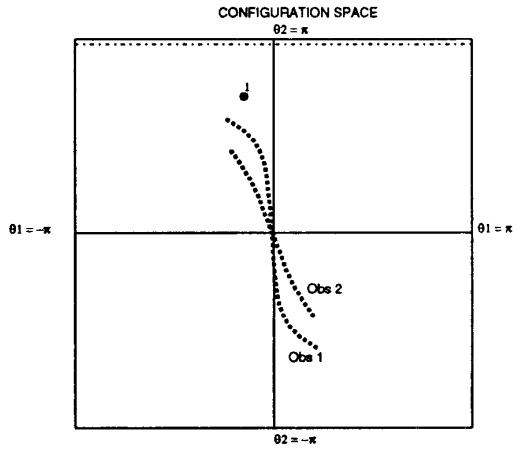

Fig. 4(a) Initialization.

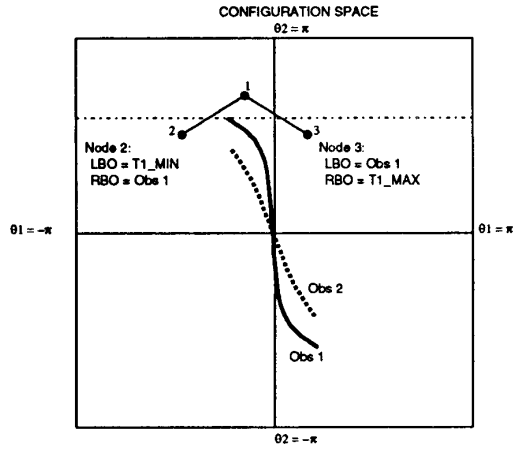

Fig. 4(b) After the first Split event.

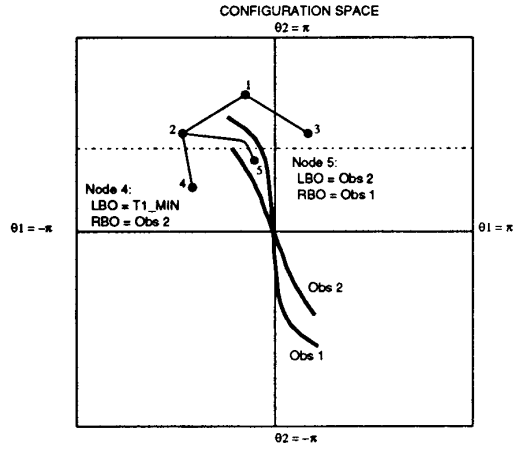

Fig. 4(c) After the second Split event.

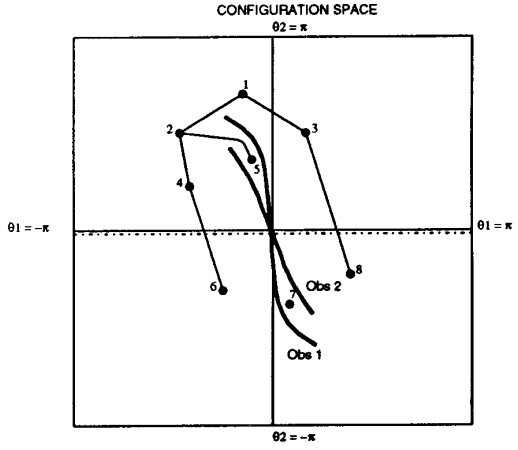

Fig. 4(d) After the Intersect event.

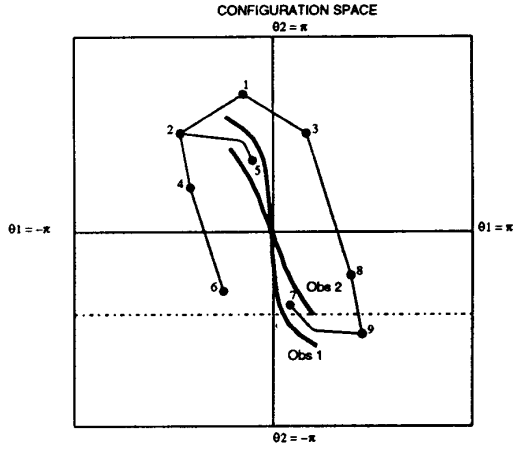

Fig. 4(e) After the first Merge event.

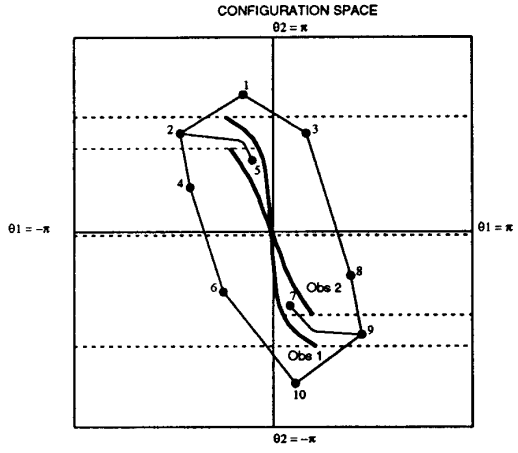

Fig. 4(f) Solution.

C. Timing Results

The algorithm described above has been run on numerous examples. The time required to process an environment with 20 point obstacles took 21.5 msec. on a SPARC-IPC workstation containing a 15.7 MIPS RISC architecture. Of this time, 10.89 msec was required for preprocessing and 9.6 msec was required to actually calculate the connectivity graph.

V. CONCLUSIONS

In this paper, an efficient algorithm for generating the connectivity graph for a class of articulated manipulators has been introduced. The structure of the code is that of a sweep line algorithm and pseudo code has been provided to illustrate the mechanisms required for handling the key events. The efficiency of the algorithm rests in a test for determining the existence of an intersection between distinct configuration space obstacles without requiring the exhaustive calculation of the curve describing this obstacle, as has previously been the case. A specific example has been discussed to illustrate the operation of the algorithm, and complexity results have been provided.

APPENDIX

This appendix gives a procedure for sorting the points of intersection of two obstacles based upon the intercept value of the line describing where the intersection must take place. Let $\theta_1 + \theta_2 = K_A$ and $\theta_1 + \theta_2 = K_B$ be lines which intersect a configuration space obstacle representing a point at a radius $R > l_1$. Let $(\theta_{1,A}, \theta_{2,A})$ and $(\theta_{1,B}, \theta_{2,B})$ denote the intersection points of these lines with the obstacle. We have the following lemma.

Lemma 1: If $\theta_{2,A}$ and $\theta_{2,B}$ are in the closed interval $[-\frac{\pi}{2}, \frac{\pi}{2}]$ and $K_A > K_B$ then $\theta_{2,A} > \theta_{2,B}$

Proof: By contradiction.

Assume $K_A > K_B$ but $\theta_{2,A} < \theta_{2,B}$. In [5], it was shown that the slope of a configuration space obstacle is given by

$$\frac{d\theta_2}{d\theta_1} = \frac{l_2 + l_1 \cos\theta_2}{-l_2}. \qquad (A.1)$$

It is apparent that for $\theta_2 \in [-\frac{\pi}{2}, \frac{\pi}{2}]$, $\frac{d\theta_2}{d\theta_1} \leq -1$. Since the function describing the obstacle is continuous and, as a consequence of the Mean Value Theorem of calculus, the chord between $(\theta_{1,A}, \theta_{2,A})$ and $(\theta_{1,B}, \theta_{2,B})$ must have a slope $m < -1$, that is,

$$\frac{\theta_{2,A} - \theta_{2,B}}{\theta_{1,A} - \theta_{1,B}} < -1 \qquad (A.2)$$

Since $\theta_{2,A} - \theta_{2,B} < 0$, $\theta_{1,A} - \theta_{1,B} > 0$ Therefore,

$$\theta_{2,A} - \theta_{2,B} < \theta_{1,B} - \theta_{1,A} \qquad (A.3)$$

or

$$\theta_{1,A} + \theta_{2,A} < \theta_{1,B} + \theta_{2,B} \qquad (A.4)$$

or

$$K_A < K_B, \qquad (A.5)$$

a contradiction. Hence $\theta_{2,A} > \theta_{2,B}$. ∎

Also,

Lemma 2: If $\theta_{2,A}$ and $\theta_{2,B}$ are in the open intervals $[-\pi, -\frac{\pi}{2}]$ or $[\frac{\pi}{2}, \pi]$ and $K_A > K_B$ then $\theta_{2,A} < \theta_{2,B}$

Proof:

Similar to the proof of Lemma 1.

Analogous lemmas exist for obstacles at a radius $R < l_1$, however the proofs are slightly different.

Details regarding the determination of the θ_2 interval in which a particular intersection takes place and the algorithm which results as an application of Lemmas 1 and 2 may be found in [2].

REFERENCES

[1] M. S. Branicky and W. S. Newman, "Rapid computation of configuration space obstacles," in *Proc. IEEE Int. Conf. Robotics Automat.*, Cincinnati, OH, May 13–18, 1990, pp. 304–310.

[2] J. J. Fox, "Path planning for articulated manipulators," Ph.D. Dissertation, Dept. of Elect. Eng., Purdue Univ., forthcoming.

[3] Y. Hwang and N. Ahuja, "Gross motion planning—A survey," to appear in *ACM Computing Surveys.*

[4] T. Lozano-Pérez, "Automatic planning of manipulator transfer movements," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC–11, no. 10, pp. 681–698, Oct. 1981.

[5] A. A. Maciejewski and J. J. Fox, "Path planning and the topology of configuration space," to appear in *IEEE Trans. Robotics Automat.*

[6] W. Meyer and P. Benedict, "Path planning and the geometry of joint space obstacles," in *Proc. 1988 IEEE Int. Conf. Robotics Automat.*, Philadelphia, PA, April 24-29, 1988, pp. 215-219.

[7] F. P. Preparata and M. I. Shamos, *Computational Geometry: An Introduction*, Springer-Verlag, New York, N.Y., 1985.

[8] J.T. Schwartz and M. Sharir, "On the piano movers' problem:I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers," *Comm. on Pure and Applied Math.*, vol. 34, pp. 345–398, 1983.