Deadlock-Free and Collision-Free Coordination of Two Robot Manipulators

Patrick A. O'Donnell and Tomás Lozano-Pérez

MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, MA., 02139

Abstract

This paper describes a method for coordinating the trajectories of two robot manipulators so as to avoid collisions between them. We assume that the robots' environment is known and so the robots' paths can be planned in advance, but that there may be significant variations in the execution time of some of the path segments. These assumptions are good models of many tasks. Our goal is to allow the motions of each manipulator to be planned nearly independently and to allow the execution of the path segments to be asynchronous. The coordination is achieved by introducing explicit coordination commands into the paths. The key problems in coordinating trajectories are to avoid *collisions* between the two robots and to avoid *deadlock*, that is, situations where each manipulator is waiting for the other to proceed.

The problem

Whenever multiple robots must operate in close proximity to each other, the potential for collision must be taken into account in specifying the robot trajectories. In this paper, we address the problem of coordinating two robot manipulators so as to avoid collisions between them, while guaranteeing that the trajectories will reach their goals. We will call this the *trajectory coordination* problem.

First a word on terminology. We will use the term *path* to refer to the shape of a motion, that is, the shape of the curve in the robot's *configuration space*. The term *trajectory* will denote the time history of positions along a path, that is, a curve through the robot's *state space*. There are infinitely many trajectories possible for a given path, each differing in the time history of velocities along the path.

In this paper, we assume that the manipulators are *loosely coupled*, that is, they share a common workspace and may actually transfer parts between them. This is in contrast to *tightly coupled* manipulators that are coupled dynamically, typically through a common load that is too heavy for a single robot.

We will be particularly interested in the problem of coordinating the trajectories of robot manipulators working in known, predictable environments. We assume that the paths of the manipulators can be planned off-line to avoid collisions with all the objects in the environment, except the other robot. There are no unforseen obstacles. This is true of most industrial tasks. We also assume that, although the robots' paths are predictable, their trajectories are less predictable. One example of this is arc welding where the speed may be adjusted in response to observed weld parameters. Or, when one of the steps in the task may involve a sensor-based operation of varying duration. There are other, simpler reasons for unpredictable trajectories, such as unavoidable error in the controller.

We have the following goals for our trajectory coordinator:

- It should be possible to plan the path for each manipulator essentially independently.
- The resulting trajectories should guarantee that the manipulators will reach their goals.
- It should be possible to execute the trajectories without precise time coordination between the manipulators.
- The safety of the manipulators should not depend on accurate trajectory control of individual manipulators.

Previous approaches

There are several existing approaches to trajectory coordination of multiple manipulators. They can be classified into *global* and *local* approaches. The global methods construct complete trajectories for all the robots that guarantee, or attempt to, that all the robots reach their goals safely. [1] describes how to construct the configuration space/time for several planar manipulators, each with two revolute joints. The trajectories of the manipulators are planned one at a time, using the swept volume, in space/time, of the previous trajectories as obstacles. [2] describe an algorithm for finding collision-free trajectories for two planar manipulators, with one prismatic and one revolute joint, by characterizing the combinatorial structure of the configuration space of the two arms.

One problem with these global methods is that they depend on carefully controlled trajectories. As we mentioned earlier, there are many applications where this lock-step coordination is not practical and/or efficient. Also, the methods are computationally intensive since they have to consider, either explicitely or implicitely, both space and time.

There has also been work [4, 8, 9] on the closely related problem of planning trajectories for multiple moving objects, not manipulators. These studies have shown that this problem is PSPACE-hard, that is, one expects the time to achieve coordination to grow exponentially with the number of moving objects. This is not surprising; it has been shown repeatedly that the complexity of motion planning grows exponentially with the number of degrees of freedom of the task. There has

also been work in the less directly relevant area of planning the motions of a single object in the presence of other moving objects whose trajactories are known [5].

Local methods for collision-avoidance and coordination make decisions at each instant of time as to what the trajectory for each robot should be. [3] develops a controller that coordinates multiple planar manipulators, with one prismatic and one revolute joint, by incorporating collision constraints into the control system of the robots. [10] describes a technique for coordinating multiple moving objects, including manipulators, by defining separating planes at each moment and ensuring that the objects stay on opposite sides of them.

These local algorithms are based on actual measurements of the robots' positions and thus can accomodate unexpected variations in trajectories or unexpected obstacle. But, in the general case, local methods cannot guarantee that each robot will reach its goal. They may reach a *deadlock*, where one robot is blocking the other. Furthermore, because these local methods rely on changing paths to avoid collisions they are not suited to situations where the paths are tightly constrained.

Local methods for coordinating multiple robots are related to local methods for avoiding collisions, notably the potential field methods originally developed in [6].

One key difference between our method and previous coordination methods is that our method attempts to decouple the path specification step from the trajectory specification step. Therefore, all collisions are avoided by using time, that is, by waiting for the other arm to get out of the way, without changing the path. We will see, however, that our method also leads to a prescription of how to change paths so as to minimize the interference between the trajectories.

Our approach

In this section we present the simplest form of our approach; in later sections we consider a number of extensions. We assume that the path of each manipulator has been planned off-line and is composed of a sequence of *path segments*. Each path segment is constrained to lie within a box in the robot's joint space, that is, the path is constrained to lie within the bounding box of the initial and final joint values of the segment. We make no further assumption about the shape of the path segments. Of course, paths produced by typical linear joint interpolation between two joint vectors are acceptable. Furthermore, we assume that we can estimate roughly the time required to execute each path segment. Under these assumptions, we observe that the trajectory coordination problem is a *scheduling* problem, like job-shop scheduling, in which space is the shared resource. Our approach to scheduling is based on an approach (see [7]) that was developed for concurrency control in databases [11].

Task-Completion Diagram

Consider the *task-completion (TC) diagram* shown in Figure 1. On the horizontal axis, we represent the segments of the path for robot A and on the vertical axis, we show the segments of the path for robot B. The sizes of the segments in the diagram are shown as constant, independent of the execution time of the segments. Below, we will make the segment length be proportional to estimated time, but we want to keep things as simple as possible at first. Let A_i denote the i^{th} path segment, $0 \leq i < m$, for robot A and B_j denote the j^{th} path segment, $0 \leq j < n$, for robot B. For our purposes, a path segment includes both of its endpoints. Let $R_{i,j}$ be the rectangle spanned

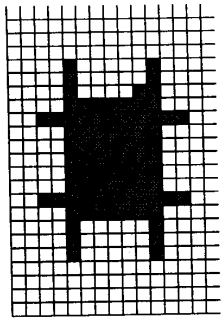

Figure 1: Task-completion diagram with constraints arising from potential collisions.

by segments A_i and B_j. If the volumes swept out by the robots while following path A_i and B_j collide, then $R_{i,j}$ is shaded in the TC diagram. The union of the shaded $R_{i,j}$ are the *collision regions*, indicating a possible collision between two path segments. Path segments that define collision regions should never be in execution at the same time.

A *schedule* for the task is any non-decreasing curve that connects the lower left corner of the diagram to the top right corner. We will not be considering the details of execution of each path segment and, therefore, we will only consider schedules composed of horizontal, vertical, and diagonal lines. That is, we will only care whether manipulator A, B, or both A and B are executing a path segment. A *safe schedule* is one that never penetrates into the interior of the union of the collision rectangles, the shaded $R_{i,j}$. It is important to note that the boundaries of the collision regions are safe. This follows from our treatment of the endpoints of a path segment as belonging to the segment. Therefore, a possible collision at the endpoint of a segment would cause both segments that share that endpoint to generate collision regions. This, in turn, entails that such an endpoint could not be on the boundary of a collision region, it must be in the interior.

Consider the following simple scheduling algorithm:

```
procedure Greedy Scheduler;
  begin
    i := 0; j := 0;
    while i < m or j < n do
      begin
        if R_{i,j} is collision free
          then begin
            if i < m then
              begin Execute A_i; i := i + 1; end
            if j < n then
              begin Execute B_j; j := j + 1; end
          end
        else
          if i < m and R_{i,j-1} is collision free
            then begin Execute A_i; i := i + 1; end
          else
            if j < n and R_{i-1,j} is collision free
              then begin Execute B_j; j := j + 1; end
        Wait for any completion signals;
      end
  end
```

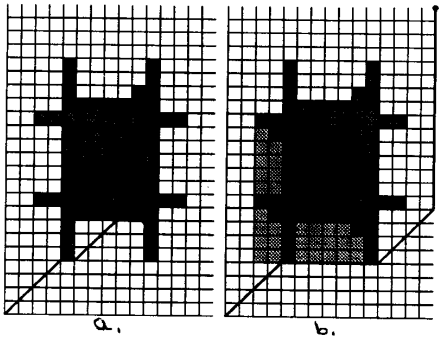

Figure 2: (a) A partial schedule that leads to a deadlock. (b) SW Closure of collision regions is shown lightly shaded. Potential deadlocks are shown black.

Basically, this algorithm keeps track of the next segment to be executed by the values of i and j. It then checks whether the simultaneous execution of both of the next segments could cause a collision. This is done by checking whether $R_{i,j}$ is shaded in the diagram. If no collision is possible, then they are both executed. If a collision is possible, then it checks whether executing the next A segment is feasible. This is done by testing $R_{i,j-1}$. If $R_{i,j-1}$ is not shaded, then we can execute A_i, that is, move along the bottom boundary of $R_{i,j}$. If $R_{i,j-1}$ is shaded and since we have already found $R_{i,j}$ shaded, we know it is impossible to execute A_i at this point. If A_i is illegal, we check whether executing B_j is legal by checking $R_{i-1,j}$.

In this algorithm, A_{-1} and B_{-1} refer to the path segments before the beginning of each path; these are either zero-length segments corresponding to the initial positions or, in the case of a cyclical path, the final segments in the path A_{k-1} and B_{n-1} respectively. Similarly, A_k and B_n are either null segments corresponding to the final point or, in the case of a cyclical path, they denote the initial segments A_0 and B_0 respectively.

It is fairly clear that this algorithm ensures that there are no collisions. It will never initiate a path segment if another segment that could lead to a collision is currently executing. But, on the other hand, it does not guarantee that a complete schedule will be produced. Figure 2 a shows a partial schedule, produced by the Greedy scheduler, where no further segments can be executed. This situation is called a *deadlock* or impasse.

We can construct a simple variation of the TC diagram we have been using so far, such that the Greedy Scheduler always finds a deadlock-free, as well as collision-free, schedule. This is done by computing the *SW-closure*[7] of the collision regions (Figure 2 b). Such a closure fills in the non-convexities in the connected components of the union of the collision regions. The SW-closure of N rectangles can be computed by a line-sweep algorithm in time $O(N \log N)$[7]. For trajectories of composed of n or fewer segments, there are at most n^2 collision regions. Therefore, the SW-closure of the TC diagram can be constructed in time $O(n^2 \log n)$.

There is one sublety when computing SW-closures. If some segment A_i has a possible collision with the final position of robot B, that is, with the final endpoint of segment B_{n-1}, then there will be a deadlock if robot A does not execute segment A_i before robot B finishes its task. This deadlock is easily

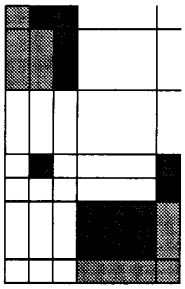

Figure 3: In this figure, segment A_2 collides with the last position in B's path and segment B_3 collides with the last position in A's path. The SW-closure places the necessary restrictions on possible schedules to avoid these special deadlocks (shown in black).

avoided by the following construction: add a null segment to the end of each manipulator's path, representing the final position. Then, when generating the SW-closure, consider the entire part of the TC diagram beyond the end of both robots' paths to be a collision region, but only in so far as it will fill in SW-closure with explicit collision regions. (Otherwise, the entire TC diagram would be filled in due to the SW-closure of this virtual collision region.) Figure 3 illustrates this procedure. In this figure, segment A_2 collides with the last position in B's path and segment B_3 collides with the last position in A's path. The SW-closure places the necessary restrictions on possible schedules to avoid these special deadlocks.

Once the SW-closure of the TC diagram is taken, a schedule exists if and only if both the origin and the goal are each not part of any collision region (or SW-closure). Clearly, if there is a collision at the goal, the task cannot be completed; similarly for a collision at the origin. Also, if the origin is included in the SW-closure of some collision region, then there is an unavoidable deadlock, and again, a safe schedule is impossible. To show the converse, assume that the origin and goal are both clear. For there to be no schedule, there would have to be a connected collision region (including SW-closure) cutting across the entire TC diagram. But, with the "virtual collision region" described above, such a connected region would have a SW-closure which included the origin. Thus, there can be no connected region cutting across the TC diagram. Put another way, the safe areas including the goal and the origin must be connected.

Further, let us assume that the origin is contained in a SW-closure, and thus there is unavoidable deadlock. By replanning part of the path of one robot using the swept volume of the other robot as an obstacle, assuming that we can find a new path to avoid this obstacle, we can guarantee that we can find a schedule to complete the task. This technique is described in greater detail in Section .

Constructing a Schedule

There are two general approaches to constructing a schedule, given a TC diagram. One is a local method, such as the Greedy Scheduler shown earlier. That particular scheduler assumes that there is a central controller that initiates the motions for both manipulators. One can also build a decentralized version of the greedy scheduler for the common case of independently

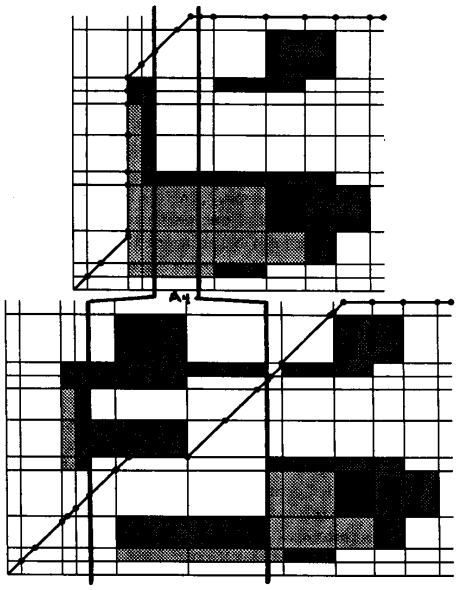

Figure 4: An example illustrating the increase of parallelism in a schedule, obtained by clearing a collision regions. Segment A_4 in the top diagram has been replaced by a new three-segment path in the bottom diagram. Note that the new path for A has longer expected time, but the increased parallelism results in a schedule that is somewhat faster overall.

controlled robots. In that case each shaded $R_{i,j}$ becomes a "lock," that is, a variable that can be indivisibly tested and set so that we can guarantee that only one process "owns" the variable. Before executing path segment A_i, A's controller must grab the locks of the shaded $R_{i,j}$, for all j. Similarly, before executing path segment B_j, B's controller must grab the locks of the shaded $R_{i,j}$, for all i. In this scenario, the locks corresponding to collisions and to the SW-closure must all the obtained. Of course, one may actually aggregate adjacent locks in a column or row into a single lock if desired.

An alternative approach to scheduling, which we can call global, involves searching the TC diagram for a schedule that is "optimal" by some measure, for example, the total execution time. This global search can also guarantee finding a legal schedule if one exists without the need to assume that the purely sequential schedules are safe. A schedule, such as may be found by this search, corresponds to a fixed sequence of activations for each of the path segments. A schedule can be characterized by the sequence of its crossings of the horizontal and vertical lines that demarcate the path segments in the diagram. Crossing each line adds to the schedule a command to wait for the completion of one segment and then to initiate the next segment. Such a schedule can be implemented in a centralized controller simply by marching down a list and issuing the appropriate START and WAIT commands. A decentralized implementation is also straightforward.

Increasing Parallelism

In the preceding discussion we have largely ignored the issue of the time to execute a schedule. In practice, time is crucial. In what follows, we will assume a slightly modified form of the TC diagram in which the axes correspond to expected execution time. Each path segment will have an expected time and this will determine its dimension in the diagram. Given this modified diagram, we can search for a schedule with the best expected execution time. The best possible schedules tend to have a great deal of parallelism, that is, they are nearly diagonal lines in the TC diagram. But, a particular TC diagram may have many collision regions near its diagonal, forcing the "best" schedule into the sequential execution of large segments of the path. The fault is not in choosing the schedule but in the original choice of paths. If the paths were chosen completely independently, there is no guarantee that much parallelism is possible. It is possible, however, to take two paths and to increase their parallelism by modifying some segments of the paths. The resulting TC diagram will allow more parallelism but the paths will generally be longer and may, therefore, increase the total execution time.

To increase the potential parallelism in a TC diagram, we pick an $R_{i,j}$, or a larger collision region formed from the union of several $R_{i,j}$, such that:

1. The region is shaded because of a collision and not because of the SW-closure operation.
2. The initial and final positions of the path segments giving rise to the collision region are free of collision.
3. The region is large enough that it causes a significant increase in the total time of the best schedule to go around it.

Having chosen one or more of these regions, new paths can be planned to connect the initial and final points of the A segments, but using as obstacles the volume swept out by B as it moves through its segments.

A safe path may not exist but, if it does, this means that the region in the new diagram will no longer need to be shaded. Of course, the new path may introduce collisions with some segments that may not have previously collided. Fortunately, such collisions will be off the diagonal and therefore will not affect the desired schedule. Also, the new path will generally be longer than the original path since there are new obstacles to be avoided. On the other hand, the impact of clearing one collision may be greater than just clearing one small region due to the impact of the SW-closure.

The process of increasing parallelism is illustrated by the simple example in Figure 4. Note that by focusing on the collisions near the diagonal, we are engaging in a crude form of space/time planning. Only the combination of segments that are going to be executed near the same time need to be considered. This is very different from, and substantially better than, the trivial strategy of using the swept volume swept out by one robot over its complete path as an obstacle while planning the path for the other robot.

Dealing with Variable Segment Times

Earlier, we indicated that in many applications, the execution times for path segments cannot be predicted reliably, especially in situations involving sensing or variable-time processes. What

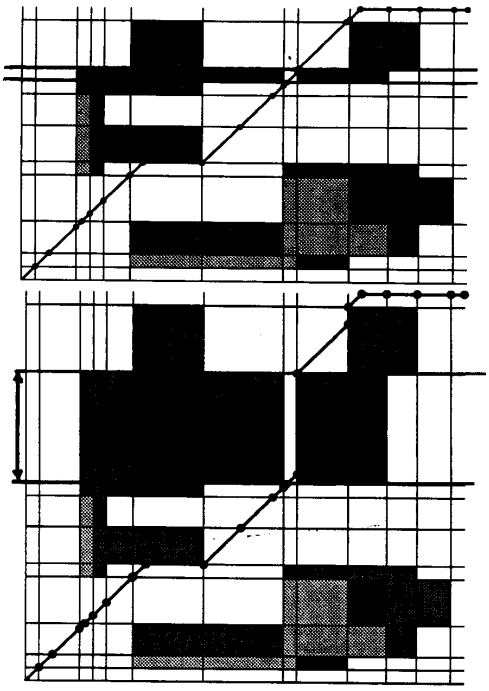

Figure 5: The effect of delay in the TC diagram is to move collision regions on or off the diagonal. This figure shows the effect on the schedule of increasing the time for segment B_8 between the top and the bottom figures.

is the impact of the change in length of one of the path segments? The crucial impact is that it may change the choice of the best schedule. Geometrically, the stretching of a segment may move some new collision region into, or out of, the path of the best schedule (Figure 5).

One simple strategy when faced with a significant delay is simply to redo the coordination of the remainder of the schedule in the modified diagram. This is the approach we have adopted. It may be possible, however, to characterize the possible changes to the schedule brought about by different changes in the execution times of the various segments and to construct a decision tree that can be used on-line. We will be investigating this option in the future.

Changing the Task

When there are substantial delays in the execution of one or more segments, it may be desirable to change the allocation of tasks from one manipulator to the other. In the preceding discussion, we have assumed that the task assignments are fixed, but this need not be the case.

Consider a task where four parts are to be taken from an input pallet, processed and taken to an output pallet. There are two processing stations, each with its own robot. The initial assignment has each robot doing two of the objects. What happens if the A robot becomes delayed waiting at the first processing step, perhaps waiting for human intervention? Usually, we would like the B robot to take over the processing of the other three parts.

Assume that the robots start each cycle in a standard position, so the last motion of a cycle is to return to that position; this assumption can be relaxed later. Then, we construct the TC diagram assuming that each robot will carry out the complete task, that is, process all parts. This expanded diagram contains all the combinations of assignments of task steps to the different robots. Furthermore, in this TC diagram, schedules can jump from the end of one part cycle to the beginning of another one that may not be adjacent to it in the diagram. This jumping around is made possible by our assumption that the endpoints of each cycle are the same. If they were not, we would have to plan the transfer motions between each pair of cycles separately, but this does not present a fundamental problem.

The method outlined above has the drawback of requiring separate planning of each part cycle, including its interactions with all other possible cycles. In practice, cyclical tasks tend to be mostly the same motions except for a few path segments, such as when picking up a new part from its own pallet location. We can construct a "generic" cycle path that represents the union of the path segments for all the cycles. This path union can be used to compute the swept volume of the robots over all instances of the cycle for different parts. We can then do the planning for the possible interactions of the two arms as if they were each executing only the generic path. For tasks in which the actions performed in each "cycle" are very different, one must consider each cycle separately using the expanded diagram suggested above.

Testing for Collisions

Our approach requires that we be able to detect potential collisions between path segments. That is, we need to identify which $R_{i,j}$ in the TC diagram need to be shaded. This is readily accomplished by computing the volume swept out by each manipulator while executing segment A_i and segment B_j and testing for these volumes for intersection. In general, it is hard to compute the exact volume swept out by the robot during a move, since it is non-convex and contains curved surfaces. In our implementation, we only compute an approximation of the volume swept out by an approximation to the links of the manipulator. In practice, this approximation works out very well.

Let us consider robot A and assume for the moment that only one joint is moving during segment A_i. For each link of the manipulator distal to the moving joint, we compute a rectangular bounding box for the link in the coordinate system of the moving joint. The problem now reduces to finding the swept area of a two-dimensional rectangle in the x-y plane of the moving joint, and sweeping the resulting polygon between the z-coordinates of the bounding box.

Figure 6 illustrates this procedure. Link 3 of a Unimation Puma robot is shown being swept through 74 deg of rotation of joint 2. The vertices of the polyhedral model of the link are projected onto the x-y plane of the coordinate system for joint 2. The bounding box for these projected points is computed, then a polygonal approximation to the swept area of that rectangle is found. The final diagram shows the entire swept volume for links 2–6.

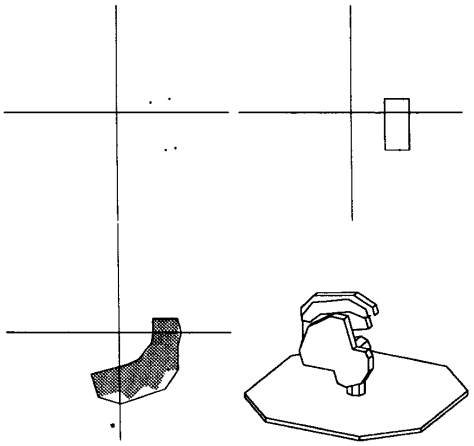

Figure 6: Steps in computing the swept volume of one link of a manipulator: (a) Link vertices, (b) bounding box, (c) sweep, (d) final result.

It is clearly possible to approximate the true swept volume of the bounding box of the link arbitrarily closeley. It is also possible to improve the algorithm so as to approximate the actual link arbitrarily closely, and thus compute a polyhedral approximation to the swept volume with better and better accuracy. However, detecting collisions between the swept volumes takes time proportional to the product of the number of edges and faces of the polyhedra, so unless there is an overriding reason to use very accurate swept volumes, it is beneficial to use simpler polyhedra (rougher approximations) to save time in collision detection.

The procedure above demonstrates the construction of the swept volume for one moving joint, and only for a revolute joint. The swept volume of a prismatic joint is trivial to compute exactly. It is not so clear how to extend the procedure to multiple moving joints. The approach we have taken is to use the volumes swept out by distal links due to the motion of the distal joints as "virtual links" when computing the swept volumes of the proximal joints. For example, if both joints 2 and 3 are moving, we first compute the swept volumes of links 3-6 due to the motion of joint 3. We then substitute these polyhedra for the actual links 3-6 and compute the swept volume due to joint 2.

This procedure for handling multiple moving joints is extremely conservative, as it computes a much larger volume that the robot is typically going to sweep out. But, this approximation is consistent with the mimimal assumptions we have made on the shape of the paths, namely, that all joints stay within the limits specified by the endpoints of the path segment. Strict coordination between the joints is not necessary.

Detection of a potential collision is achieved by simply testing whether the polyhedral approximation to the swept volumes of A_i and B_j intersect. The actual volume of the intersection does not need to be computed. Since an intersection test between to arbitrary polyhedra can be expensive, we have implemented several quicker tests to determine if the full swept volume approximation needs to be used.

Before the actual swept volume is computed for A_i, we compute a bounding box approximation to the swept volume. This can be done much more quickly than computing the full approximation. What is actually calculated is the bounding box in world coordinates of the bounding box in joint n coordinates for the swept volume due to motion of joint n. Only if there is some B_j for which the bounding boxes intersect is the full swept volume for A_i computed, and the intersection test of the full swept volume is only performed between those A_i and B_j for which the bounding boxes intersect.

Acknowledgments

This work was funded by the Office of Naval Research under the University Research Initative Program through contract N00014-86-K-0685. Additional support was provided by an NSF Presidential Young Investigator Award (Lozano-Pérez).

References

[1] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[2] S. Fortune, G. Wilfong, and C. Yap. Coordinated motion of two robot arms. In *IEEE International Conference on Robotics and Automation*, pages 1216–1223, San Francisco, 1986.

[3] E. Freund and H. Hoyer. Real-time pathfinding in multirobot systems including obstacle avoidance. *The International Journal of Robotics Research*, 7(1):42–70, February 1988.

[4] J. E. Hopcroft, J. T. Schwartz, and M. Sharir. On the complexity of motion planning for multiple independent objects. *The International Journal of Robotics Research*, 3(4):76–88, 1984.

[5] K. Kant and S. W. Zucker. Toward efficient trajectory planning: the path-velocity decomposition. *The International Journal of Robotics Research*, 5:72–89, 1986.

[6] O. Kathib. Real-time obstacle avoidance for robot manipulator and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, Spring 1986.

[7] F. P. Preparata and M. I. Shamos. *Computational Geometry*. Springer Verlag, New York, 1985.

[8] J. Reif and M. Sharir. Motion planning in the presence of moving obstacles. In *IEEE Symposium on the Foundation of Computer Science*, pages 144–154, Portland, OR, 1985.

[9] J. T. Schwartz and M. Sharir. On the piano mover's problem iii. coordinating the motion of several independent bodies: the special case of circular bodies amidst polygonal barriers. *The International Journal of Robotics Research*, 2:46–75, 1983.

[10] P. Tournassoud. A strategy for obstacle avoidance and its application to multi-robot systems. In *IEEE International Conference on Robotics and Automation*, pages 1224–1229, San Francisco, 1986.

[11] M. Z. Yannanakis, C. H. Papadimitiou, and H. T. Kung. Locking policies: safety and freedom from deadlock. In *IEEE Symposium on the Foundation of Computer Science*, pages 286–297, 1979.