Fast Motion Planning for Multiple Moving Robots

Stephen J. Buckley

IBM T.J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Abstract

This paper presents an efficient solution to the motion planning problem for multiple translating robots in the plane. In previous work, the robots were assigned priorities, and motion plans were generated for each robot in priority order through a combined space-time. The papers did not specify how to assign priorities. In this paper, it is shown that careful priority assignment can greatly reduce the average running time of the planner. A new priority assignment method is introduced which attempts to maximize the number of robots which can move in a straight line from their start point to their goal point, thereby minimizing the number of robots for which expensive collision-avoiding search is necessary. This prioritization method is extremely effective in sparse workspaces where the moving robots are the primary obstacles, as in Scheinman's *Robotworld*.

Introduction

This paper addresses the motion planning problem for multiple translating robots in the plane. A complete planner[1] has been proposed for this problem {8}, but its running time is exponential in the number of moving robots. An incomplete but more efficient approach is to assign priorities to the robots, and generate a motion plan for each robot in priority order through a three-dimensional space-time {2,1}. In the space-time, the first two dimensions represent the configuration of the moving robot, and the third dimension represents time. Obstacles in the space-time represent higher priority robots in motion. Giving robot 1 priority over robot 2 does not prohibit parallel motion of 1 and 2. Rather, it means that the motion of robot 1 is planned through the space-time first, and if their paths cross, robot 2 must take responsibility for avoiding collision. Unfortunately, planning collision-free motions for just one moving robot can take very high polynomial time relative to the size of the environment {7}, so even a priority-based planner can take a large amount of time.

Previous papers on priority-based planning did not specify how to assign priorities to the robots. This paper presents a method of assigning priorities which attempts to maximize the number of robots which can travel in a straight line from their start point to their goal point, thereby minimizing the number of robots for which expensive collision-avoiding search is necessary. This prioritization method reduces the average running time of a priority-based planner, and in many cases produces efficient motion plans as well. The method is extremely effective in sparse workspaces where the moving robots are the primary obstacles, as in Scheinman's *Robotworld* {6}. Robotworld consists of a number of two-dimensional Sawyer motors, which are magnetically suspended from a horizontal platten, and can move with two degrees of freedom about the platten.

[1] A *complete* planner must find a motion plan if one exists for a given problem instance.

The prioritization method has been incorporated into a working planner, implemented in C on an IBM RT PC. The planner has been tested on many examples, and is being used to generate programs to assemble electronic cards on a simulated electronics assembly station {5}.

Algorithm

Prioritization Rules

Consider two moving robots 1 and 2, which are candidates for straight-line motions. There may be many possible collisions between the robots if they move in straight lines, but the most critical ones occur at the start and goal regions of each robot. En-route collisions can always be avoided by proper adjustment of robot speeds, as long as start and goal conflicts have been resolved. (A method for setting the speeds is described below in the section entitled "The Linear Planner".) Let S_1 and G_1 denote the start and goal points of robot 1, respectively. Define S_2 and G_2 similarly. In an initial attempt to resolve start and goal conflicts between the robots, we apply the following prioritization rules:

1. *Start Conflicts*

   If robot 1 on a straight-line motion from S_1 to G_1 intersects robot 2 in configuration S_2, then robot 2 should be given priority over robot 1. This forces robot 1 to wait for robot 2 to pass by, allowing robot 1 to continue on a straight-line motion toward G_1. Note that this rule does not prohibit robot 2 from making a straight-line motion from S_2 to G_2.

2. *Goal Conflicts*

   If robot 1 on a straight-line motion from S_1 to G_1 intersects robot 2 in configuration G_2, then robot 1 should be given priority over robot 2. This forces robot 2 to wait for robot 1 to pass by before moving to G_2. Note that this rule does not prohibit robot 2 from making a straight-line motion from S_2 to G_2, it merely means that robot 2 may have to move at less than full speed until robot 1 has passed by.

These rules cannot always be applied on a consistent basis. Techniques for dealing with inconsistencies will be developed below.

Priority Graphs

The prioritization rules define a binary relation on the set of robots, which can be represented by a graph called a *priority graph*. In a priority graph, if robot A has priority over robot B, then a directed arc is constructed from A to B. The highest priority robots correspond to root nodes in the priority graph (nodes which have no arcs leading into them). A root node is guaranteed a collision-free straight-line motion to its goal point. Once the motion plan for a root node is complete, it can be removed from the priority graph, exposing new root nodes.

The newly exposed root nodes are also guaranteed collision-free straight-line motions to their goal points, subject to proper speed adjustment along the way (see the subsequent section entitled "The Linear Planner").

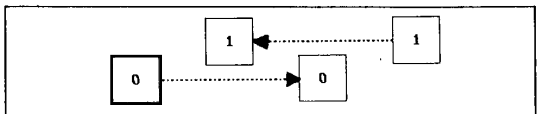

Figure 1. A motion planning problem involving two robots.

Cycles in the priority graph correspond to conflicting situations in which two or more robots have priority over each other. This indicates that all of the robots in the cycle cannot make straight-line motions to their goal points; one or more of them will have to move out of the way of the others. Figure 1 illustrates this type of situation.

Cycles can be broken by deleting one or more robots from the priority graph. Robots that have been removed will be refered to as *complex robots*. These are precisely the robots which must move out of the way of the others. Moving out of the way implies a search through space and time. Since we want to minimize the number of complex robots, it pays to delete the robot from the priority graph which has the most cycles passing through it, thereby removing the most cycles. This can be repeated until there are no more cycles. The remaining robots are guaranteed collision-free straight-line motions, and will be refered to as *linear robots*.

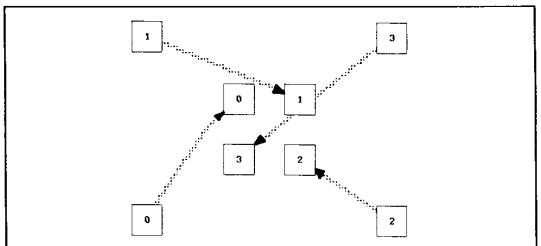

Figure 2. A motion planning problem involving four robots.

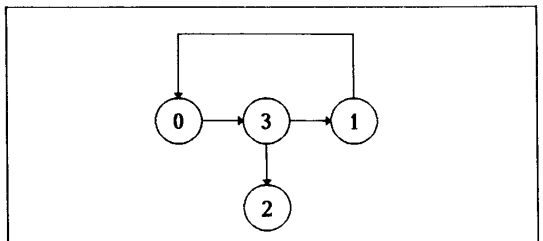

Figure 3. The priority graph for Figure 2.

Figure 2 shows an example problem involving four robots. There are no start conflicts in this example, but there are a number of goal conflicts. Figure 3 shows the priority graph. There is one cycle, through robots 0, 3 and 1. To break the cycle, we delete robot 0 from the priority graph and declare it a complex robot. Since there are no more cycles in the priority graph, the rest of the robots are linear, and can be planned in the priority order 3-1-2. Robot 0 is given the lowest priority, and must be planned by searching through space and time for a collision-free sequence of motions.

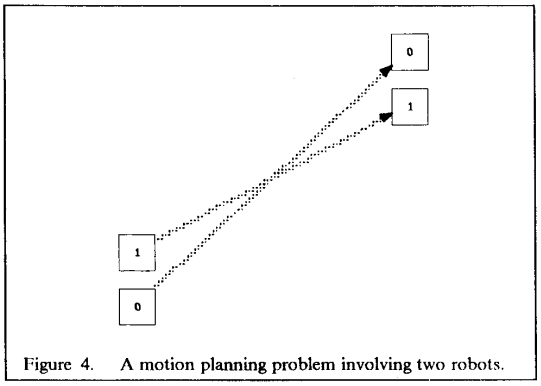

Figure 4. A motion planning problem involving two robots.

Although it worked in the previous example, complex robots cannot always be assigned lower priorities than linear robots. To understand the reason, consider Figure 4. In this example, robot 1 has start conflict priority over robot 0, but robot 0 has goal conflict priority over robot 1, creating a cycle. To break the cycle, robot 1 can be declared a complex robot, but the problem remains that robot 1 must move away from its start point before robot 0 collides with it. However, robot 1 cannot reach its goal point before robot 0 has reached its goal point. The solution to this dilemma is to search for a via point in the workspace which is a safe resting place for robot 1 with respect to the straight-line path of robot 0. Robot 1 is assigned a higher priority than robot 0 for the journey to its safe via point. Upon reaching the via point, its priority is then reduced below that of robot 0, allowing robot 0 to achieve its goal region first. Note that priority switching is necessary only for complex robots which have start conflict priority over linear robots.

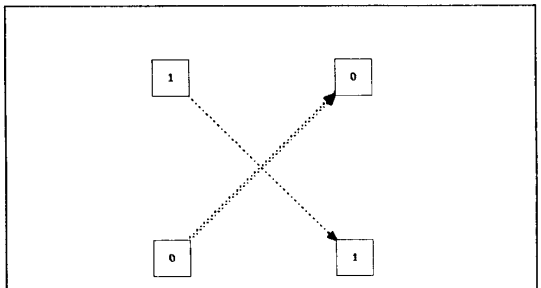

Figure 5. A motion planning problem involving two robots.

Figure 6. The swept areas of the two robots.

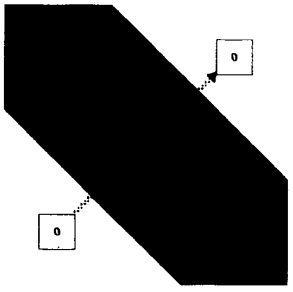

Figure 7. The configuration space of robot 0 is shown in white.

The Linear Planner

We will assume that robots can maintain a constant velocity along a specified segment of travel. A linear robot is guaranteed a collision-free motion from its start point to its goal point, as long as it sets its speed properly. To choose the robot speed, consider Figure 5. Assume that robot 0 is to avoid collision with robot 1, whose motion has already been planned. The points where the two robots may intersect can be seen by plotting their swept areas, as shown in Figure 6. The configurations of robot 0 which may lead to a collision state can be seen by plotting its *configuration space* {4}, as shown in Figure 7. In this figure, the swept area of robot 1 forms an obstacle to the motion of robot 0. The first and last possible collisions of robot 0 will be refered to as its *collision point* and its *free point*, respectively. These points are easily computed by intersecting a line segment and a polygon. The collision point and the free point of robot 1 can be computed similarly. Robot 0 can then avoid colliding with robot 1 by reaching its collision point at the time that robot 1 reaches its free point. If there are no other robots, then robot 0 can continue at maximum velocity to its goal point.

In general, there are more than just two robots. All possible collisions of a linear robot are computed by pairwise intersection with the other robots. Possible collisions are then ordered by increasing distance from the robot's start point. Finally, safe speeds are computed for each inter-collision segment.

It is important that prioritization and linear planning be computed efficiently relative to complex planning, otherwise prioritization would not be worthwhile. Both prioritization and linear planning can be computed by performing $O(n^2)$ line segment / polygon intersections, where n denotes the number of linear robots. In practice, over the course of many experiments involving up to 16 robots, the linear planner has never taken more than a couple of seconds thus far.

The Complex Planner

The prioritization method described in this paper does not call for any particular motion planning algorithm for complex robots. If desired, the Kant/Zucker {2} or Erdmann/Lozano-Pérez {1} algorithm can be used.

I designed a new algorithm, which works as follows. Assume that we are planning the motion of complex robot C. A number of candidate via points are placed in the workspace. I chose to place candidate via points at the configuration space obstacle vertices with respect to C. The graph formed by this set of candidate via points is similar to the *visibility graph* of Lozano-Pérez and Wesley {3}. Depth first search is then used to find a connected path from C's start point to its goal point, using the linear planner to plan motions between via points. Candidate via points are searched in order of increasing distance from C's goal. The first collision-free path is returned, and no candidate via point is expanded more than once. Therefore, the number of calls to the linear planner is $O(n)$, where n is the total number of robots. In this case, however, the linear planner must compute intersections over multi-segment robot paths, so its complexity increases to $O(n^3)$ intersections in the worst case. The complex planner must therefore compute $O(n^4)$ intersections in the worst case. Note that this complex planning algorithm does not attempt to find the shortest path, which would take significantly longer.

Results

The planner has been tested on many examples, two of which are documented in this section.

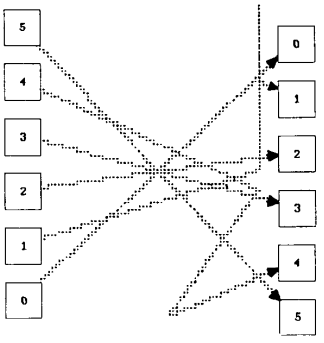

Figure 8. A motion planning problem involving six robots.

```
setpos 0 10.00 10.00
setpos 1 10.00 25.00
setpos 2 10.00 40.00
setpos 3 10.00 55.00
setpos 4 10.00 70.00
setpos 5 10.00 85.00
move 0 4 15.36 15.00 55.05 42.47 40.30 89.88 64.90 61.24 112.31
         85.00 80.00 132.41
move 1 5 74.90 45.10 64.90 74.90 90.10 109.90 74.90 89.91 132.31
         74.90 69.90 152.31 85.00 65.00 162.41
move 2 3 20.79 41.44 79.79 23.33 41.78 109.07 85.00 50.00 170.74
move 3 3 11.84 54.51 110.49 19.17 52.56 164.91 85.00 35.00 230.74
move 4 5 51.25 50.87 74.34 74.90 39.90 97.99 50.00 5.00 132.89
         61.15 9.78 275.45 85.00 20.00 299.31
move 5 3 14.69 80.00 92.25 31.67 61.89 219.07 85.00 5.00 275.96
```

Figure 9. The motion plan for the six robot problem (see Appendix I).

Figure 8 shows a motion planning problem involving six robots. Also shown in the figure are the paths computed by the planner. There were two complex robots, robots 1 and 4. (Observe their respective journeys to safe via points.) The robots were prioritized in the order 1*-4*-0-2-3-5-1-4, where 1* refers to the journey of robot 1 to a safe via point. At each path crossing in the figure, the robot with higher priority crossed first. Figure 9 shows the motion plan that was generated by the planner for this problem. The planner took 1.07 seconds for the six linear robots, and 1.73 seconds for the two complex robots.

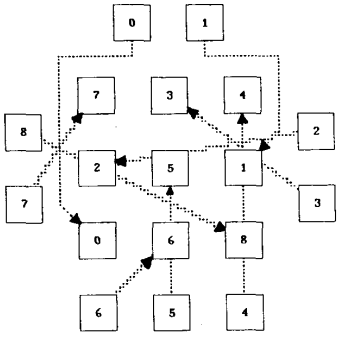

Figure 10. A motion planning problem involving nine robots.

Figure 10 shows a motion planning problem involving nine robots. Also shown in the figure are the paths computed by the planner. There were two complex robots, robots 0 and 1. The robots were prioritized in the order 0*-1*-3-4-8-7-2-5-6-0-1. Figure 11 shows the motion plan that was generated by the planner for this problem. The planner took .67 seconds for the six linear robots, and 9.7 seconds for the two complex robots.

```
setpos 0 40.00 90.00
setpos 1 60.00 90.00
setpos 2 90.00 60.00
setpos 3 90.00 40.00
setpos 4 70.00 10.00
setpos 5 50.00 10.00
setpos 6 30.00 10.00
setpos 7 10.00 40.00
setpos 8 10.00 60.00
move 0 6 39.90 80.10 9.90 40.10 80.10 10.10 19.90 80.10 30.30
         19.90 79.85 48.60 19.90 39.90 88.55 30.00 30.00 98.65
move 1 5 60.10 80.10 9.90 80.10 80.10 29.90 80.10 70.00 70.73
         80.10 59.90 80.83 70.00 50.00 90.93
move 2 3 87.27 59.55 34.55 80.00 58.33 60.83 30.00 50.00 110.83
move 3 1 50.00 70.00 40.00
move 4 2 70.00 37.50 30.00 70.00 70.00 62.50
move 5 3 50.00 25.00 50.00 50.00 41.67 100.83 50.00 50.00 109.17
move 6 2 40.00 20.00 95.75 50.00 30.00 105.75
move 7 2 12.50 43.75 22.50 30.00 70.00 48.75
move 8 2 60.00 35.00 50.00 70.00 30.00 60.00
```

Figure 11. The motion plan for the nine robot problem (see Appendix I).

Remarks

Although the planner works best in sparse environments, it can plan amidst static obstacles by modeling them as robots whose start points and goal points are identical.

Currently, the planner assumes that robots are rectangles aligned with the xy axes. This assumption is valid for Robotworld. Under different circumstances, the planner can be easily modified to handle arbitrary polygonal robots. Extension to three dimensional translating robots would also be straightforward. Extension to rotating robots would be more difficult due to the curved nature of rotation space.

Another current assumption is that robots can instantaneously change velocity between segments. Although this is a common assumption in the motion planning community, it is not feasible in practice. Additional work is required to generate smooth segment transitions while avoiding collision.

This effort has shown clearly that careful prioritization pays off. Although the motion planning problem is in theory a high polynomial problem, it has been shown that in practice one can often plan very efficiently when good heuristics are employed.

As the popularity of Robotworld-type systems increases, this algorithm should be of great utility. It may also be applicable in mobile robot applications.

Acknowledgments

I would like to acknowledge the help of Frank Park, who implemented the graphics and simulation system upon which the planner was constructed, and Jan Baartman, who suggested the "safe via point" solution for complex robots with start conflicts. I would also like to thank Chae An, Jim Korein, and the IEEE referees for valuable comments on the presentation of this paper.

Appendix I: Motion Commands

Motion plans generated by the planner were graphically simulated on a system designed by Park [5]. In his system, motion commands have the following syntax:

- **setpos** *robot x y*

  Declares the initial position of a robot.

- **move** *robot n* x_1 y_1 t_1 x_2 y_2 t_2 ... x_n y_n t_n

  Moves a robot from its current position through the indicated via points and times, which are specified as offsets from the current time.

References

1. Erdmann, M., and Lozano-Pérez, T. 1987. "On Multiple Moving Robots", **Algorithmica** 2(4), pp. 477-521.
2. Kant, K., and Zucker, S. 1984. "Toward Efficient Trajectory Planning: The Path-Velocity Decomposition", **International Journal of Robotics Research** 5(3), pp. 72-89.
3. Lozano-Pérez, T., and Wesley, M. 1979. "An Algorithm for Planning Collision-Free Paths Among Polyhedral Objects", **Communications of the ACM** 22(10), pp. 560-570.
4. Lozano-Pérez, T. 1983. "Spatial Planning: A Configuration Space Approach", **IEEE Transactions on Computers** 32(2), pp. 108-120.
5. Park, F., Korein, J., and Buckley, S. 1988. "A Simulator for a Multiple Agent Electronic Assembly System", RC 14098, IBM T.J. Watson Research Center, Yorktown Heights, NY.
6. Scheinman, V. 1987. "Robotworld: A Multiple Robot Vision Guided Assembly System", **Proc. ISRR**.
7. Schwartz, J., and Sharir, M. 1982. "On the Piano Movers Problem: II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds", Courant Institute Technical Report 41, New York University.
8. Schwartz, J., and Sharir, M. 1983. "On the Piano Movers Problem: III. Coordinating the Motion of Several Independent Bodies: The Special Case of Circular Bodies Amidst Polygonal Barriers", **International Journal of Robotics Research** 2(3), pp. 46-75.