A Motion Planner for Multiple Mobile Robots

David Parsons and John Canny *

Department of Electrical Engineering and Computer Science
Electronics Research Laboratory
University of California
Berkeley, CA 94720

Abstract

We describe an algorithm for planning the motions of several mobile robots which share the same workspace. Each robot is capable of independent translational motion in two dimensions, and the workspace contains polygonal obstacles. The algorithm computes a path for each robot which avoids all obstacles in the workspace as well as the other robots. It is guaranteed to find a solution if one exists. The algorithm takes a cell decomposition approach, where the decomposition used is based on the idea of a product operation defined on the cells in a decomposition of a two-dimensional free space. We are implementing this algorithm for the case of two robots as part of ongoing research into useful algorithms for task-level programming of the RoboWorld system.

1 Introduction

This paper describes a solution to a special case of the generalized mover's problem. The case we consider is the problem of coordinating the motion of several independent mobile robots moving in a bounded planar workspace which contains obstacles. The robots are convex polygons, and the obstacles may have general polygonal shape. The goal is to move the robots from given initial positions to given final positions in such a way that they avoid collisions either with each other or with the obstacles.

Each robot is capable of independent translational motion in 2 dimensions, and so the problem of moving k robots has a 2k-dimensional configuration space. A valid path through this space must satisfy a set of constraints of 2 distinct types: those imposed in order to avoid the collision of any robot with an obstacle, and those which avoid the collision of any pair of robots with each other. Together these constraints determine the boundary of the free space, which is the set of legal configurations of the system of robots. A valid path between two configurations exists if and only if they lie in the same connected component of free space. (For an explanation of the notion of configuration space see [LP93].)

The algorithm we describe here is a complete solution in the sense that it is guaranteed to find a valid path if one exists. The worst-case running time of the algorithm is exponential in the number of robots, which is not surprising since the problem is known to be PSPACE-hard ([HSS84]). However, we believe that the design and implementation of a complete solution is justified for two reasons: 1) in practical applications, the number of robots is a small fixed quantity, and so the complexity of the algorithm reduces to polynomial in the size of the workspace description; and 2) such a solution will provide an important benchmark for the analysis of faster heuristic algorithms. In particular, we are motivated by the need to develop practical algorithms for motion coordination in the RoboWorld environment. (RoboWorld is a general-purpose robotic assembly system consisting of a number of two-dimensional Sawyer motors magnetically suspended from a horizontal plat form; see [Sch87] for details.)

The geometric problem of computing feasible motions for a robot system in the presence of physical obstacles has received much attention during the last decade. Both theoretical lower bounds and useful algorithms for particular cases of the problem have been obtained. Excellent overviews of this work can be found in [Sha89], [Yap87], and [Whi85]. Many particular cases have been shown to be PSPACE-hard, suggesting an exponential lower bound on the worst-case running time of any algorithm which solves one of these cases (unless PSPACE = P). Indeed, all known algorithms for these cases take exponential time in the worst case, where the exponent is at least linear in the number of degrees of freedom of the robot system. For problems involving configuration spaces of very high dimension, therefore, only heuristic solutions will be acceptable in practice. See [BLL85] for a recent fast heuristic approach to a wide class of motion planning problems of high dimension.

Practical complete algorithms have been developed for some motion planning problems of dimension ranging from 2 to about 6. Most of these algorithms are based on one of two basic approaches: the "retraction" approach ([OSY83]) and the "cell decomposition" approach ([SS83]). Both approaches reduce the problem of searching for a path in configuration space to the problem of searching for a path in a graph; the approaches differ mainly in how the graph is constructed. It is also possible to categorize most algorithms as either global or local. A global algorithm calculates the entire search graph before the search begins, whereas a local algorithm uses the search to guide the graph calculation as it proceeds.

The algorithm described here takes a global cell decomposition approach. For a given instance of the problem, the algorithm proceeds in two distinct phases: first it computes a decomposition of the free space for that instance into cells, and then it searches the resulting adjacency graph for a path. The computation in the first phase depends only on the given obstacles and the shapes of the robots; that is, the cell decomposition is independent of the start and goal configurations of the problem instance. Therefore this step may be thought of as a preprocessing step, and once the graph has been computed it may be used to answer many path queries for the same obstacle set. In this sense the algorithm may properly be considered as a solution to
a motion planning *query* problem, in which extra preprocessing time is deemed acceptable in the interest of fast complete path finding for particular start/goal configuration pairs. (This distinction between preprocessing time and query time is common in many problems in computational geometry; many examples may be found in [PS85].) Since answering a particular query reduces to finding a path through a graph, a variety of fast search algorithms are applicable to the second phase.

Many previous approaches to solving the multiple mobile robot problem have been heuristic algorithms, which tend to discard paths quickly in some environments but in others may fail to find a path and wrongly conclude that none exists. A common form of such a heuristic is to plan a path for each robot in sequence, taking care at each planning step to avoid collisions with the paths generated in previous steps ([ELP87]). Thus each successive path is planned in a 3-dimensional "space-time" configuration space. This is tantamount to a greedy algorithm, since a path generated for one robot may make it impossible to find path(s) for succeeding robot(s) which don't interfere. A refinement of this technique ([Duc89]) attempts to minimize such conflicts by choosing carefully the order in which plans are generated. It does this by maximizing the number of robots which can travel in a straight line from their start point to their goal point. The algorithm therefore works well in sparse environments, but fails when many obstacles are introduced.

Our algorithm takes the general approach described for the case of 2 robots in [SS88]. They introduce the paradigm of taking the product of 2-dimensional cells in a cell decomposition of the free space for each robot alone, and then further decomposing the product cells in order to avoid collisions between the two robots. In this paper, our contribution is to apply this paradigm to the particular problem of planning coordinated motions for convex polygonal robots moving among polygonal obstacles, and to show how the linearity of the constraints in this case can be exploited to implement the algorithm in a straightforward way.

We now turn to describing the algorithm in detail. In the next section, we discuss the method in the abstract for \( k \) robots, and in section 3 we give some details of an implementation which we are developing for the case \( k = 2 \).

2 The Algorithm

2.1 Overview

A cell decomposition of a topological space \( S \) is a finite partition \( K \) of \( S \) such that each set \( c \in K \) is a connected set. Two cells \( b, c \) are said to be adjacent if \( \partial(b \cap c) \neq \emptyset \). The adjacency between them correspond to the vertices and edges of a graph \( G(K) \). The applicability of cell decompositions to motion planning stems from the fact that there exists a continuous path in \( S \) between two points \( p_1, p_2 \in S \) if and only if there is a path between \( c_1 \) and \( c_2 \) in the graph \( G(K) \), where \( K \) is any cell decomposition of \( S \) and \( c_i \) is the cell in \( K \) containing \( p_i \). The idea, then, is to construct a cell decomposition of the free space \( F \) determined by the robot system geometry and the obstacles of a particular problem instance, and then do the path searching in the resulting graph.

In order for a particular cell decomposition \( K \) of a free space to be useful for path planning, it must have the following characteristics:

1. it must be simple to find a path from any point in one cell to any point in an adjacent cell;
2. it must be simple to determine which cell \( c \in K \) contains any point \( p \in F \); and
3. the decomposition itself must be straightforwardly computable from the obstacles and the robot geometry.

The cells in the decomposition we use are all convex polytopes, and adjacencies exist wherever two polytopes share a facet. (A facet of a \( d \)-dimensional polytope is a surface of dimension \( d - 1 \), i.e. of codimension 1.) Therefore the first requirement above is particularly easy to meet: to move from a point in one cell to a point in an adjacent cell one may move along the piecewise linear path which goes first to the centroid of the facet shared by both cells, and then to the destination point. This path is guaranteed to be valid by the convexity of the cells.

The second requirement could be met by performing a series of half-space inclusion tests. That is, in order to determine if a point \( p \) is contained in a particular convex polytope \( P \), one could simply compare the point against each hyperplane containing a facet of \( P \). In fact, requirement 2 is met even more efficiently by means of an indexing data structure imposed on the cells during their construction. The third requirement is inherent in the construction algorithm, which we describe next.

2.2 The Cell Decomposition

Recall that for a system with multiple mobile robots, geometric constraints on legal motions of the system arise from two distinct sources: obstacle avoidance and inter-robot collision avoidance. The principal idea behind our cell decomposition of free space is to separate the processing of the two types of constraints. First we will discuss each type of constraint in isolation, and then show how the cell decomposition takes care of them both.

2.2.1 Obstacle constraints

If the inter-robot constraints could be ignored, the problem would reduce to solving the 2-dimensional planning problem of finding a path for a single robot in the plane, and repeating this for each robot. The 2D free space for a single robot may be computed by the "obstacle-growing" method ([LP83]), and a useful decomposition into cells of this free space is easy to obtain: since the grown obstacles still have polygonal boundaries, the region may be tessellated into convex polygons. Such a tessellation is easy to produce, and it constitutes a cell decomposition of free space for the case of a single robot.

This observation motivates the following construction for the \( k \)-robot case: first, we compute a convex tessellation \( T = \{ P_i : i = 1, \ldots, n \} \) of the 2D free space \( F_1 \) for a single robot. Here each \( P_i \) is a convex polygon, and \( T \) partitions \( F_1 \), so we have that \( i \neq j \Rightarrow P_i \cap P_j = \emptyset \) and that \( \bigcup_{i=1}^{n} P_i = F_1 \). (If the robots have different shapes, their free spaces will also be different, and so a different tessellation would need to be computed for each. In this paper we assume for simplicity of presentation that each robot has the same shape; the generalization to robots of different shapes would present no difficulty for the rest of the algorithm.) From the tessellation we next compute a collection of *product* cells. The \( k \)-fold product of the \( k \) (not necessarily distinct) polygons \( P_1, P_2, \ldots, P_k \) is defined to be the locus of points in configuration space for which the reference point of robot 1 lies
If there are \( k > 2 \) robots, we get a set of these inequalities for each of the \( \binom{k}{2} \) possible pairs of robots. Let \( L_{ij} \) denote the set of inequalities arising from the overlap condition for robots \( i \) and \( j \). Then some overlap occurs at \( p \in B \) iff
\[
\bigvee_{1 \leq i < j \leq k} \left\{ \bigwedge_{L_{ij}} f(p) \right\} \quad (2)
\]
is true, where \( \bigvee \) and \( \bigwedge \) denote disjunction and conjunction, respectively.

It is helpful to think of the inequalities geometrically as halfspaces in the 2k-dimensional configuration space. A halfspace may be represented by a hyperplane and a sign convention; e.g. the first inequality in the example (1) above becomes the hyperplane given by the equation \( H(p) = r(p) - x(p) - w = 0 \), with the convention that \( H(p) > 0 \) implies that \( p \) is a configuration free of overlaps between robot 1 and robot 2.

2.2.3 Putting the constraints together

Once we have generated the set of basic cells and the appropriate set of inter-robot constraint hyperplanes, it remains to combine them into a cell decomposition of the \( k \)-robot free space \( F_k \). The idea is to store the collection of basic cells with each hyperplane in turn. At each slicing step, any cell which intersects the current hyperplane is removed from the collection and replaced with two subcells, one lying on each side of the hyperplane. When the next hyperplane slices the collection, these new cells may be subdivided again. This process continues until each hyperplane has been considered for each basic cell.

The result of this slicing process is a set of convex cells, each of which is sign-invariant with respect to the constraint hyperplanes. That is, any two points in a single cell both lie on the same side of each hyperplane. Therefore, either a cell is a subset of free space \( F_k \) or it contains only illegal points, depending on the value of expression (2) above for points in the cell. The set of free cells constitutes the partition of free space in our cell decomposition. It is straightforward to determine the adjacencies among these cells, as described in the next section.

It can be shown that the complexity of the algorithm in most cases is not increased greatly by the cell-slicing step. A conservative upper bound on the number of subcells of each basic cell which may be divided by the constraint hyperplanes is given by the number of cells in the arrangement of those hyperplanes in the 2k-dimensional configuration space; an upper bound on this latter quantity is \( O(h^4) \) for \( h \)-hyperplanes in \( d \) dimensions ([Edel72]). A robot polygon with \( r \) vertices gives rise to \( O(r) \) constraint hyperplanes for each pair of robots. Therefore we can get at most \( O\left( \binom{k}{2} r^2 \right) \) hyperplanes per basic cell in the worst case. This reduces to \( O(k^4 r^2) \) when \( r \) is considered a constant.

In practice, this cell subdivision factor is usually much smaller. If the set of hyperplanes contains many pairs of parallel hyperplanes (as in the case of square robots), the number of possible sign sequences is greatly reduced. For two square robots, at most 9 subcells are possible. Also, a significant optimization is to avoid slicing any cell which already contains either only free configurations or only illegal ones. With this refinement, for example, at most 5 subcells will be obtained for any basic cell in the case of square robots. This optimization also ensures that all basic cells which are the product of polygons whose minimum
3 Implementation

We are implementing the planning algorithm described above for the case of two robots for the purpose of testing various heuristic algorithms for motion planning in RobotWorld. In this section we briefly describe the data and control structures used, and discuss the practical considerations of applying the algorithm to the RobotWorld environment.

The primary data structure for this algorithm is the representation of convex polytopes in the four-dimensional configuration space. The slicing of a cell is in fact completely determined by its 1-skeleton; however, since we need the 3-D facets for path planning it is necessary to explicitly represent all the faces of all dimensions, 0–4. For this we use a general incidence graph structure as described in [Edel87]. This structure represents a polytope in n dimensions as a directed acyclic graph, in which each node represents a subspace of some dimension 0 through n of the polytope and there is an edge from a d-face f to a (d−1)−face g if and only if g is part of the boundary of f. (This is similar to a winged-edge structure, but with no ordering information among the 0- and 1-dimensional subfaces is maintained.) Figure 2 shows part of the incidence graph which represents the basic cell depicted above in figure 1. (The complete incidence graph for this polytope has 49 nodes and 126 edges.)

With this representation, the slicing of a cell with a hyperplane is performed by a simple recursive algorithm which uses the 1-skeleton of the polytope as the base case. For a (3-dimensional) hyperplane H which passes through a 4-dimensional polytope P, we need to be able to compute the two polytopes P1 and P2 which satisfy

- P1 and P2 lie on opposite sides of H;
- P1 ∪ P2 = P; and
- P1 ∩ P2 = F, where F is a 3-dimensional convex polytope lying in H.

F is the common facet shared by P1 and P2. The main observation needed to construct the incidence graphs for P1 and P2

is that for every d-face f of P which has a non-null intersection with H, f ∩ H is a (d−1)-face of the shared facet F. The cell-slicing function is invoked recursively to compute these intersections, and the appropriate subfaces of P1 and P2 are constructed as the function returns up the recursive cell tree. This recursive process is illustrated in figure 3, which depicts the slicing of a 3-D polytope with a 2-D hyperplane.

The adjacency relation between cells is determined during cell construction. When the basic cells are set up, their adjacencies are known from the adjacencies of the polygons in the underlying tesselation. The basic cells adjacent to the cell \( \mathcal{H} = \{P_1, P_2, \ldots, P_k\} \) are all possible cells of the form \( \mathcal{H} = \{P_{i_1}, P_{i_2}, \ldots, P_{i_k}\} \), where A_{i_j} is a polygon adjacent to P_{i_j} in the tesselation and i \in \{1, \ldots, k\}. Polytopes are considered adjacent in the tesselation if and only if they share a common edge; thus we avoid computing paths which involve moving a robot across a polygon vertex. Clearly this gives all adjacencies across basic cell subfaces of codimension 1.

The control structure of the algorithm propagates basic cell construction from a cell to its neighbors in a depth-first style, and in so doing it automatically fills in these preliminary adjacencies among the basic cells. When a cell gets split by a hyperplane, the adjacency links which need to be updated in the cell network are also easy to determine: wherever a cell is split, we get a new adjacency between the two new subcells, and the old cell's adjacencies are inherited by each new cell whenever the new cell contains at least a section (of full rank) of the shared facet to which the adjacency corresponds. Again, the implementation performs the slicing of cells in a depth-first order for each hyperplane, and it updates the adjacencies as it proceeds. After the entire cell network is created, the illegal cells and all adjacency links connected to them are removed. The resulting network is the cell decomposition we are after.

The network itself is simply an undirected graph. Each node
in the graph is uniquely identified by an ordered pair of polygons \((P_i, P_j)\) together with a sign sequence. The node identified by such an index pair is the one corresponding to the unique subcell of the basic cell \(\mathcal{B}(P_i, P_j)\) satisfying the given signs with respect to the constraint hyperplanes. During the construction of the network, it is convenient to build an indexing structure which efficiently performs this mapping from index pairs to cell nodes. This lookup table is used in answering particular path queries in the completed cell network. First the index pairs for the start and goal points are determined, by using a planar point location algorithm in the tesselation and by comparing the points to the hyperplanes. Then the cell nodes for those index pairs are retrieved from the table. A depth-first search is then initiated from the start node to the goal node. The search is guided by a heuristic which searches from one cell to adjacent cells in order of increasing difference between the angle to the neighbor cell and the angle to the goal cell. This heuristic tends to encourage the discovery of the most direct path through the network from start to finish (although it doesn’t guarantee that the globally shortest path will be found). If a path in the cell network is found, it is converted to paths for the robots using the centroids of shared faces along the path as via points, as discussed in section 2.1 above. A path, if found, will thus be piecewise linear.

One complication arises in applying this algorithm to the RobotWorld environment. This is that each robot has attached to it a flexible cable which may interfere with the motion of other robots. The robots themselves are otherwise well approximated by squares. One heuristic means of dealing with this problem is to simply omit one of the overlap conditions for the two robots, say for example the condition \((y_2 - y_1 \leq w)\). Then any configuration in which the robots overlap in \(x\) and for which \(y_1 - y_2 \leq w\) will be deemed illegal (see figure 4). The effect of this is to leave a “column” of width \(w\) above robot 1 and below robot 2 in which the cables may safely trail along. The cables then just need to be placed so as to extend vertically to the robots from opposite sides of the platen. (This of course means that the search space is reduced, and there will exist solvable instances for which the algorithm fails to find a path. However, the exact kinematics of the cables are far from simple, and a truly complete algorithm which takes them into account is hard to imagine.)

4 Conclusion

The algorithm presented in this paper gives a complete solution to the motion coordination problem of finding collision-free paths for multiple mobile robots moving in a polygonal environment. The algorithm works for any set of \(k\) robots which have convex polygonal boundaries. The approach is to pre-compute a cell decomposition of free space, and search for a path in the resulting adjacency graph. The preprocessing phase has 4 major steps:

1. Compute the free space \(F_1\) of a single robot.
2. Partition \(F_1\) into a set \(T\) of polygons.
3. Build a network of \(k\)-fold product cells.
4. Refine the cells of the network into subcells which are sign-invariant with respect to the inter-robot constraint hyperplanes.

If \(n\) denotes the size of the environment description (e.g. \(n\) is the number of vertices in the obstacle polygons), it can be shown that the free space \(F_1\) computed in step 1 has \(O(n)\) size (see [LS87]). This implies in turn that the tesselation \(T\) computed in step 2 has \(O(n)\) polygons (an upper bound is the number of triangles in a triangulation, which is \(O(n)\)). Step 3 therefore yields \(O(n^2)\) product cells. If the complexity \(r\) of the robot polygons is considered a constant, then step 4 increases the number of cells by a factor \(C(k)\) which depends only on \(k\), and so the algorithm overall takes \(O(C(k)n^2)\) time and space. An upper bound on \(C(k)\) is \(O(k^{4k})\); however in practice it is generally much smaller. Since the number of degrees of freedom of the problem is actually \(2k\), this complexity is quite reasonable for a complete algorithm.

Once the preprocessing of a given set of obstacles is completed, the resulting cell network may be used to answer any number of path-finding queries for robots moving amidst those obstacles. Thus the algorithm is particularly useful for applications in which the obstacle set changes only infrequently. This accords well with the offline/online distinction made in many problems in robotics. Also, the average performance of the motion planner may be improved by first applying a heuristic algorithm for finding paths quickly to a given problem instance. In this setting, the complete algorithm is invoked only when the heuristic fails to find a path.

The paths generated for the robots by this algorithm are piecewise linear. If smoother paths are desired, various path-refinement techniques may be used to adjust the trajectories as a post-processing step.

References


