A SUBDIVISION ALGOUITHM IN CONFIGUKATION SPACE FOR FINDPATII WITH ROTATION

Rodney A. Brooks and Tomas Lozano-Peres

MIT Artificial Intelligence Laboratory
545 Technology Square
Cambridge, Massachusetss, 02139, U.S.A.

ABSTRACT

A recursive cellular representation for configuration space is presented, along with an algorithm for searching that space for collision free paths. The details of the algorithm are presented for polygonal obstacles and a moving object with two translational and one rotational degrees of freedom.

I. Introduction

In this paper we present an algorithm for finding collision free paths for a rigid polygonal object moving through space that is cluttered with obstacle polygons. The paths can include rotations of the object. The algorithm will find a path from a given initial position and orientation to a goal position and orientation if such a path exists, subject only to a user-specified resolution limit on displacements.

The problem addressed here is an instance of the problem known as the *find path* or *mover's problem* in robotics. The problem arises when planning the motion of a robot manipulator or mobile robot in an environment with known obstacles. For related approaches to the find path problem, see Brooks [1982], Lozano-Perez [1981, 1983a], Lozano-Perez and Wesley [1979], Moravec [1080], Schwartz and Sharir [1981, 1982], and Udupa [1977]. A survey of the. different approaches to findpath and a discussion of its role in robot task planning can be found in Lozano-Perez [1983b].

The algorithm described here is based on the configuration space approach described by Lozano-Pcrcz [1981, 1983a]. The *configuration* of a rigid object is a set of independent parameters that characterize the. position of every point in the object. We associate a local coordinate frame with a rigid object, such as a planar polygon. The configuration of the polygon can be specified by the *x,y* position of the origin of the local coordinate frame, known as the *reference point* and a *0* value indicating the rotation of the local frame relative to the global frame. The space of all possible configurations of an object is its *configuration* space. A point in the configuration space, a *configuration point,* represents a particular position of the object's reference point and an orientation of the object's axes.

The configuration space for planar polygons is three-dimensional while that of solid polyhedra is six-dimensional: three translational and three rotational dimensions. Due to the presence of the immovable obstacles some regions of the configuration space are not reachable; these regions are the *configuration obstacles.* Hence, in the configuration space, the moving object is shrunk to a configuration point while the immovable obstacles are expanded to fill all space where the presence of the configuration point would imply a collision of the object with obstacles. The findpath problem of finding a path for the object through the original space while avoiding obstacles is thus transformed to finding a path for the configuration point through the configuration space while avoiding the configuration obstacles.

The. fundamental structure of the algorithm is extremely simple. Configuration space is first divided into rectangloids with edges parallel to the axes of the space. Each rectangloid is labeled as (1) *empty* if the interior of the rectangloid nowhere intersects a configuration obstacle, (2) *Full* if the interior of the rectangloid everywhere intersects the configuration obstacles or (3) *mixed* if there are interior points both inside and outside of configuration obstacles. A free path is found by first finding a connected set of *empty* rectangloid cells that, include the initial and goal configurations and constructing a piecewise linear path through those empty cells. If such an *empty* cell path cannot be found in the initial subdivision of configuration space then a path that includes *mixed* cells is found. *Mixed* cells on the path are subdivided, by cutting them with a single plane normal to a coordinate axis, and each resulting cell is appropriately labeled as *empty, full,* or *mixed.* Another search for an empty-cell path is initiated, and so on iteratively until success is achieved. If at any time no path can be found through *won-full* cells of greater than some preset minimum size, then the problem as posed is insoluble (i.e., no collision free path exists given the resolution limit).

The conceptual and practical problems that must be solved to implement this algorithm are as follows:

(a) What is an efficient algorithm for labeling the newly cut cells as empty, *full* or *mixed?*

(b) How should configuration space be initially divided into rectangloids?

(c) Where should a *mixed* cell be cut when it is subdivided?

(d) What additional information should be kept with *mixed* cells describing the configuration space obstacles they intersect?

(e) How should the iterative search be controlled?

This report describes research done at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's Artificial Intelligence research is provided in part by the Office of Naval Research under Office of Naval Research contract N00014-81-K-0494 and in part by the Advanced Research Projects Agency under Office of Naval Research contracts N00014-80 C 0505 and N00014-82-K-0334.

H. Representation of Configuration Space Obstacles

For a convex polygonal moving object *A* and a convex polygonal obstacle *B* we write the configurations forbidden for *A* as *COA(B).* Figure 1 shows such a configuration obstacle for polygons *A* and *B* where their relative orientation is fixed. Lozano Perez [1983a] showed that $CO_A(B)$, for fixed relative orientations of convex *A* and *B,* is also a convex polygon. A

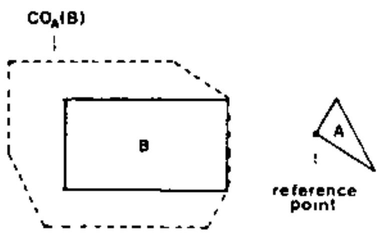


Figure 1. A configuration obstacle, $CO_A(B)$, for convex polygons A and B

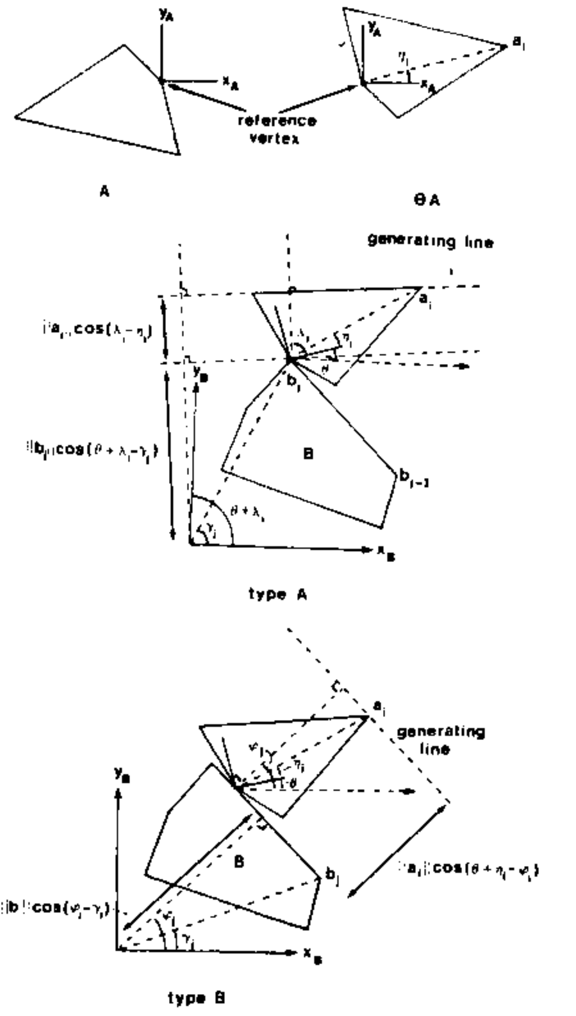


Figure 2. Derivation of types A and B surfaces.

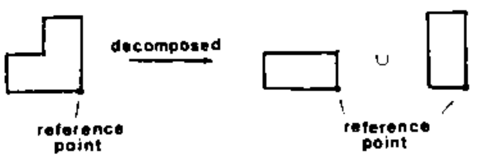


Figure 3. Non-convex moving objects can be decomposed into a union of convex objects sharing a common reference point and reference orientation.

convex polygon can be expressed as a conjunction of linear inequalities.

When changes in the orientation of the moving object are allowed, the resultant $CO^B{}_A[B)$ are neither convex nor bounded by half-spaces. Their surfaces are curved, although they do have the property that when cut by a plane normal to the rotation dimension they have polygonal cross sections (since such a cross section corresponds to a single fixed orientation).

Lozano Perez [1983a] showed that for two dimensional A and B the surfaces of the configuration obstacle could be expressed as inequalities each valid over an infinite rectangloid R in $(x, y, 0)$ space, where

$$R = [-\infty,\infty] \times [-\infty,\infty] \times [\theta_1,\theta_2]$$

for some fixed θ_1 and θ_2. These inequalities are of the form $f(x,y,\theta) \leq 0$ where f can be defined by either

$$\begin{aligned} f(x,y,\theta) &= x\cos(\theta+\lambda_i) + y\sin(\theta+\lambda_i) \\ &\quad - \|b_j\|\cos(\theta+\lambda_i-\gamma_j) \qquad \text{(type } A) \\ &\quad - \|a_i\|\cos(\lambda_i-\eta_i) \end{aligned}$$

or

$$\begin{aligned} f(x,y,\theta) &= x\cos\phi_j + y\sin\phi_j \\ &\quad - \|b_j\|\cos(\phi_j-\gamma_j) \qquad \text{(type } B) \\ &\quad - \|a_i\|\cos(\theta+\eta_i-\phi_j) \end{aligned}$$

where the ranges of validity are given by

$$[\theta_1,\theta_2] = [\phi_{j-1}-\lambda_i, \phi_j-\lambda_i] \qquad \text{(type } A)$$

and

$$[\theta_1,\theta_2] = [\phi_j-\lambda_i, \phi_j-\lambda_{i-1}]. \qquad \text{(type } B)$$

The a_i's are vertices of the "negated" moving polygon *[O)A* in Lozano Perez [11)81, 1983a]), in it,s local coordinate system. n_1 is the angle the line from the origin of that coordinate system to the point a_2 makes with the coordinate system's I axis, and >., is the angle marie by the normal to the segment from *a1* to a_{t+1}. Similarly the b_j's are the vertices of a convex obstacle polygon, the orientation of the line from the origin to *b,* and *(nellset* the orientation of the normal to the segment from b_i; to b_{j+1} The parameter *0,* a parameter of the configuration space, measures the angle between the x-axcs of the object and obstacle coordinate systems. Figure *2* shows the geometric constructions used to derive the forms of type A and D constraints.

Type *A* constraints can be thought of as being generated by a face (edge) of the moving object *A* coming into contact with a vertex of an obstacle B, and a type *D* constraint as a vertex of *A* coming into contact with a face (edge) of *B.*

We can cut up configuration space into rectangloids *R,.* For a single convex obstacle and a convex moving object (either with or without rotation) we can define the subset of the configuration space obstacle that intersects R_1 by those points in *R1* such that a conjunction Λ_1 e_7 is satisfied, where each e_i is an inequality.

In the more general case the moving object *A* is represented as a union of (possibly overlapping) convex polygons. Similarly, multiple obstacles are each represented as unions of convex polygons. Let the convex moving polygons be A_1, A_2,... A_n. Define a single reference point *P* and reference orientation for all the A_i. See Figure 3 for an example. Let the convex obstacle polygons be $_{B1B2}$, •.., B_m.

The path for object *A* can be found through the obstacle littered space by finding a path for the configuration point through configuration space filled with the union of the C O $_A$ B $_J$ ' S or

$CO^{\theta}_{A_i}(B_j)$'s. Hence, for arbitrary obstacle and moving object polygons, we first decompose them into the unions of convex polygons; then, for all pairs of obstacle and object polygons we compute the configuration space obstacles. These new obstacles are all embedded in the same configuration space, and a collision- free path is found through it for the reference point of the object to be moved.

In what follows we will represenet an obstacle embedded in configuration space as a union of finite rectangloid boxes each with edges parallel to the: coordinate axes. The boxes will each have associated with them a conjunction of inequalities which will allow us to determine the subset of the configuration space obstacle that intersects that box. We then embed these boxes into configuration space by cutting it into rectangloids.

The way in which the set of boxes bounding a particular configuration space, obstacle is chosen is discussed in section HB.

A. Representation of Configuration Space

The inequalities derived above are referred to as constraints. Each one has the form *j (x)* < 0, where *x* is the configuration point for the moving object. A point x is *inside* such a constraint if *f(x)* < 0, *outside* if *}(x)* > 0 and *on* the constraint if *f(x)* = 0. A point is contained in a configuration obstacle if it is not *outside* any of its defining constraints. A point *x* is on the surface of an obstacle if it is *on* some constraint.

If two obstacles are embedded in the same space and two of their bounding boxes intersect then the two can be decomposed into a union of boxes. Each resulting box has associated with it a disjunction of the conjunctions representing the original two obstacles. Of course, the sub boxes that only overlapped a single configuration space obstacle have just the original conjunction.

More formally, we represent configuration obstacles as follows. Configuration space is subdivided into rectangloid *cells* in order to represent the embedding of many configuration obstacles. Each cell is a closed rectangloid with edges parallel to the coordinate axes. The union of all cells is the whole space and no point is in the interior of two cells. Associated with each cell is a *sentence* of constraints of the form

$$\bigvee_{i=1}^{k} \bigwedge_{j=1}^{l_i} e_{ij}.$$

Each conjunction of constraints in a sentence will be referred to as a *clause* and each clause is made up of terms, i.e., each constraint will be referred to as a term.

A point is said to be *inside* sentence .9 if for some clause of the disjunction it is not *outside* any of the constraints. A point is said to be *outside* sentence *S* if it is *outside* some constraint in every clause. A cell *C* with associated sentence *S* is labeled as

(1) *empty* if no point of *C* is *inside S,*
(2) *full* if no point of *C* is *outside* 5,
(3) *mixed* otherwise.

The labeling is done as a by-product of trying to simplify sentence *S;* see below.

B. Simplification of Constraint Sentences

Both while constructing the original representation of configuration space and when subdividing cells during search it is necessary to associate a constraint sentence *S* with a cell *C.* It may be the case that the cell is completely *outside* or completely inside some constraints in 5. The procedure below both

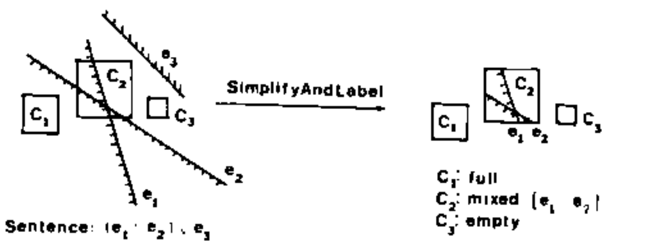


Figure 4. The SimplifyAndLabel procedure labels cells and simplifie constraint sentences of cells.

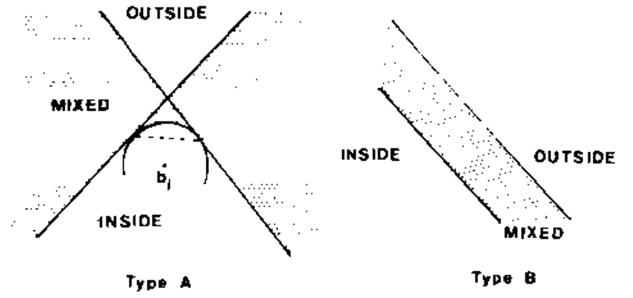


Figure 5. Slice projections into the *x-y* plane of Type *A* and Type B surfaces from configuration obstacles. These projections divide the plane into three regions: *inside, mixed,* and *outside.* The Cell Compare procedure must determine where the projection of a cell lies relative to these regions.

simplifies *S* by removing such)) constraints and, as a by product, labels the cell empty, *full,* or *mixed.*

```
procedure SimplifyAndLabel (C, S);
begin
  foreach CLAUSE in S do
    begin
      foreach TERM in CLAUSE do
        begin
          case CellCompare (C, TERM) of
               Insido: remove TERM from CLAUSE;
               Outside: begin
                            remove CLAUSE from S;
                            goto next CLAUSE;
                            end;
               Cut: ;
             endcase;
          end;
      if CLAUSE is nil
         then begin
                 label C with full
                 set S to nil;
                 exit from SimplifyAndLabel;
                 end;
    end;
  if S is nil
     then label C with empty
     else label C with mixed;
  end;
```

Figure 4 illustrates the behavior of the algorithm.

A sub-procedure CellCompare is UBed to compare a cell *C* with a single constraint, *e* say. It returns one of:

(1) Outside if no point of C is inside e,

(2) Inside if no point of C is outside e,

(3) Cut otherwise.

The formulation of CellComparc depends on the form of constraints to be considered. Note also that in general the surface of a constraint (i.e., those points on the constraint) is of lower dimension than the configuration space in which it is embedded. Thus a constraint fills only a subset of measure zero of a cell.

C. Comparing a Cell to a Single Constraint

The idea of treating the spatial and rotational components of configuration space uniformly breaks down when comparing a cell to a constraint. This is because the constraint surfaces are, in some sense, "wel! behaved" with respect to x and y, but "poorly behaved" with respect to 0 - the surfaces can creep into, and out of a cell along an edge in the 0 direction, while the vertices at both ends are on the same side of the constraint.

For a fixed 0_o both type A and type B constraint surfaces become a single, infinite, straight line. Type A and B constraints are, therefore, ruled surfaces, which are valid over some range $[0_1,0_2]$- Consider the projections into the x-y plane of the portions of these surfaces in the range $[0_1,0_2]$ (slice projections in the terminology of Lozano-Pcrez [1981, 1983a]). Points outside the projection must support columns in the 0 direction that are either completely inside or completely outside the constraint. Furthermore, there will be two disjoint regions outside the projection that correspond to these types of columns. Figure 5 illustrates the projection of both a type A and a type B constraint.

Thus the procedure CellCompare has been reduced to two more primitive operations; projection of the constraint surface into the x-y plane, and comparison of a rectangle to the two regions of the plane outside of that projection. If those regions are convex then the comparison is simple, as a convex polygon is contained in a convex set if and only if all its vertices are contained in the region.

A type B constraint projects into a strip with parallel sides. On one side is a half plane corresponding to points inside the constraint, and on the other a half plane corresponding to points on the outside. The comparison computation it trivial.

Type A constraints are considerably more difficult to deal with, as the orientation of the ruled line on the constraint surface rotates with changing 0. Consider Figure 5. Note that again both the inside and outside regions are convex, so a projected cell vertex test suffices. A point is in the outside region if it is outside both of the straight lines. It is inside if it is inside both of the straight lines and not in the small area between the circle and the point of intersection of the two lines. Such points have distance from b_j greater than the radius of the circle, and are on the opposite side of the chord (joining the two intersections of the circle with the straight lines) as point b_j

Full details can be found in a longer version of this paper: [1982].

D. Bounding Configuration Obstacles

In this section we describe how the set of bounding cells for each configuration obstacle is chosen. As 0 varies, the x and y bounds on the crosB section of $CO^e_A(B)$ normal to the 9 axis varies considerably. Hence it is best to bound $CO^0_A(B)$ by a "stack" of cells in the 0 direction; see Figure G. The cells are obtained by, first, determining a number of sub-intervals of the complete θ range and, second, determining the x and y bounds of the configuration obstacles over each sub-interval.

The sub intervals are chosen as follows. The interval $[-\pi,\pi]$ is cut at all values of the form $2n\pi + \lambda_i - \phi_j$ that, for some integer n, lie within the interval. Any constraint that is valid for some interior point of one of the resulting sub-intervals is taken as valid over the whole closed sub-interval.

A conjunction of all valid constraints over a sub-interval forms the initial constraint sentence of a cell whose θ range corresponds to the sub-interval. Later the cells will be embedded in the configuration space, and will perhaps be intersected with other cells, whence their constraint sentences will become disjunctive terms of new constraint sentences.

For a cell whose θ interval is $[\theta_1, \theta_2]$, the maximal x value, achieved over the points that satisfy its constraint sentence, must occur at a point that has a θ component within that interval. Let the θ component of this maximal point be θ_0. By convexity, the maximal point must be a vertex of the convex polygon that is the cross section of $CO_A^0(B)$ at θ_0. For a fixed θ_0, Lozano-Pérez [1983a] showed that this cross section is the convex hull of all points of the form

$$(b_{jx} + a_{ix}\cos\theta_0 - a_{iy}\sin\theta_0, \quad b_{jy} + a_{ix}\sin\theta_0 + a_{iy}\cos\theta_0, \theta_0)$$

where $a_i = (a_{ix}, a_{iy})$ and $b_j = (b_{jx}, b_{jy})$ are vertices of A and B, respectively. The maximal x value must be achieved at one of these points, which can be identified by maximizing the expressions above over values of θ_0 in the legal range. Similar computations determine the minimal value of x and the extreme values of y.

E. Constructing an Initial Representation

We are now in a position to construct an initial representation of configuration space. There are two operations. (1) Cut space into rectangloid cells. (2) Build a connectivity graph between the cells which are empty or mixed. By combining the second operation with the first it is possible to reduce its complexity to linear in the final number of cells. Full details can be found in a longer version of this paper: [1982]. The computation (as distinct from the actual algortihm) uses all the extreme x, say, values of bounding boxes to cut configuration space into strips and intersects all the boxes with those strips, cutting them when necessary. Then each strip is cut with all the extreme y values of boxes which lie within that strip. The process is repeated for each axis of configuration space.

Figure 7 shows a configuration space for polygons without rotation cut into rectangular cells. A cell is represented by a rectangloid, a constraint sentence in constraints that cut it, a label from among full, empty, and mixed, and pointers to neighboring cells. The neighbors are grouped according to their direction, e.g. +x, —x, +y and —y for two dimensional configuration space. Figure 8 shows its corresponding connectivity graph.

An important refinement to the simple connectivity graph is possible. Two neighboring cell C_1 and C_2 intersect in a rectangloid cell C one dimension lower than the original cells. Both constraint sentences S_1, from C\, and S_2, from C_2, apply to the intersection cell. If either SimplifyAndLabel(C, S_1) or SimplifyAndLabel(C, S_2) label C as full then the configuration point cannot move directly from cell C\ to C_2 Hence, although spatially neighbors, none of their empty interior points are connected by a path for the moving object's configuration point. Thus their neighbor relation can be omitted from the connectivity graph. For this reason, link A of Figure 8 can be deleted.

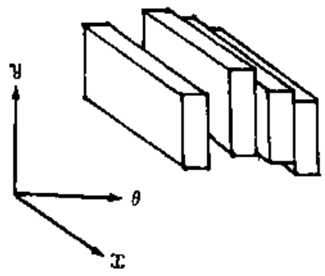


Figure 6. A "stack" of cells used to bound a three-dimensional configuration obstacle.

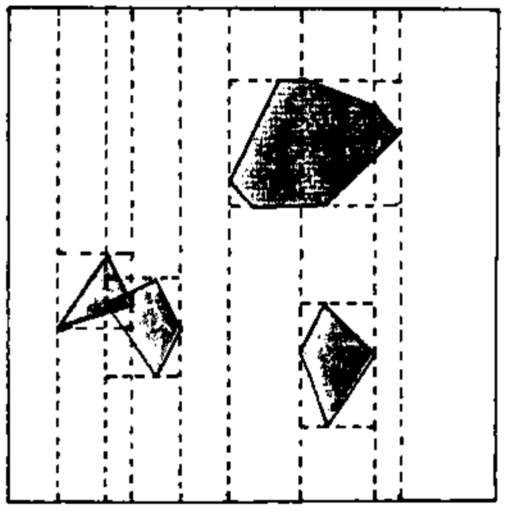


Figure 7. Configuration space without rotation with obstacles cut into rectangular cells.

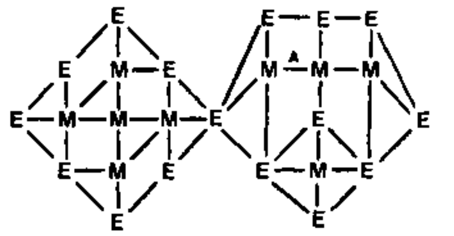


Figure 8. Connectivity graph for cells in Figure 7. The link labeled A is not actually included in the graph as it is impossible to traverse it.

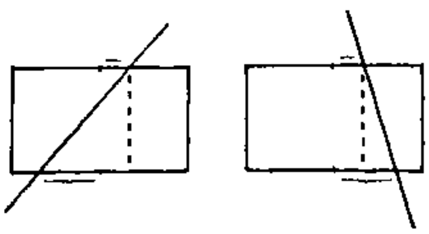


Figure 9. Given two intersection points of a constraint and boundary edges, the cell is cut at the intersection point closest to the center of the boundary edges. The two cases that are illustrated here show that this produces the simpler cell with maximal volume.

This refinement significantly cuts down the number of links in realistic connectivity graphs, greatly increasing efficiency.

A major drawback to cellular representations as above is that a number of small obstacles localized in one part of space can have significant global effects on the representation of space elsewhere, e.g., the effect of *x* bounds in Figure 7. This can become a significant problem in three -dimensional configuration space.

This problem is solved by first boxing the stack of *6* slices for a single configuration obstacle into a single cell that extends over the range [—pie,pie] in the *0* direction. Such cells are embedded into the configuration space that is sliced in the *x* and *y* directions. Then, within each cell, the cutting procedure is used all over again, first cutting normal to the 0-axis then normal to each of *x* and *y.*

The efficiency of the algorithm can be further enhanced by arranging for the second stage of cutting to be carried out only when it first becomes a candidate path cell as found by the *A* search algorithm.

ITT. Search

The representation described above for configuration space is useful only if some efficient way can be found to search it for collision free paths. Our algorithm uses the *A** (Nilsson [1971]) algorithm as its primary search engine to search the cell connectivity graph, both for paths through purely empty cells and for paths that include mixed cells.

Once a set of *empty* cells linking the initial point to the goal has been found, an actual point path through those cells must be chosen. Again the A* search procedure is used, but this time using a selected set of points in the cells along the solution path.

If no path through empty cells is available, then the representation of configuration space must be refined through cell division. It is necessary, however, to decide where the space should be refined, and how much effort should be expended in subdivision, before a new search is initiated. The point path search mentioned above is used to direct the refinement of configuration space as well as to produce a final solution path for the problem.

The efficiency of the overall search can be greatly improved by using the *divide and conquer* paradigm. There are some complications in this application, however, as each subproblem is capable of changing the global data base by refining the representation of space within its area of search. This turns out to be a significant problem and some care must be taken to minimize its adverse effects. There is an additional problem with divide and conquer in this application. The division of a problem into subproblems cannot guarantee that if the original problem is solvable then so are all the subproblems. A form of resource limited computation is used to back out of problem subdivisions that do not look promising.

A. Deciding Where to Cut a Cell

During search, the representations of *mixed* cells that lie on the candidate path are refined. A cell *C* with sentence *S* is cut into two cells, *C\ and* C_2, by splitting along one of its coordinates. Then, each sub-cell is labeled by calling SimplifyAndLabel(C_1,5) for *i* = 1,2.

The cutting operation can help the search converge on a, path through *empty* cells but only if it makes it easier to deduce if at least one of the C_1's is *full* or *empty.* This will be the case if the number of constraints appearing in at least one sentence S_1 is reduced from the number that appeared in S. Such simplification can be achieved if one of the new cells lies wholly *inside* or outside a constraint. We refer to it as the simpler sub-cell.

Two heuristic principles have also been used in the implemented algorithms described here. First, it is desirable that the new cell with simpler constraint sentence should have the maximal volume possible. This offers the possibility of finding that a large volume is either *empty* or *Full.* In one case it makes the search for a free path easier and in the other eliminates a large volume from further consideration. The second heuristic principle is that large amounts of computation in choosing the place to cut a cell, would be better spent in cutting the cell into more, less well chosen, pieces, as more cuts result in greater likelihood of actually finding *empty* or *full* sub-cells.

Thus there are three problems; (1) find cuts that lead to simpler S_1 and S_2; (2) choose the one that gives the simpler cell a large volume; (3) compute (1) and (2) quickly. Here we show how to choose a good cut according to these criteria. The method is stated in more generality than necessary so that it can easily be extended to the higher dimensional cases later in the paper.

The basic approach is as follows. Consideration is given to cutting the cell in each direction *(x, y* and *0* in this case) at some number of points for each constraint in the cell's constraint sentence. Candidate cuts are chosen wherever a constraint surface will go through a vertex of one of the new cells. A score is assigned to each such plausible cut, and the cut with minimal score is chosen as best.

It remains now to define the scoring function. If the length of the cell in the x direction is scaled to 1.0 then the score of a cut is its distance in those units from the center in the x direction. Note that this score is a proportion of the volume of the original cell that is added or subtracted from half its volume to get the volumes of the two new cells. Thus the scale of this measure is independent of the constraint or the direction of cut. The same scoring function is used in the y and *6* directions.

If a given constraint and direction of cut produces two candidate cuts then in configuration space without rotations the one with lowest score is the one that produces a simpler cell with maximal volume (helping in the achievement of (2)). This can be shown by considering two cases, illustrated in Figure 9. It does this without analysis of the constraint other than where it cuts the edges of the cell.

In our implementation we choose the cut by picking the candidate with the lowest score from among all cutting directions and single constraints. This has the effect of choosing a cut that produces a simpler cell with volume closest to half the original cell volume. The chosen cell may not be the best possible choice in terms of condition (2), but it is certainly not a poor choice. If all possible simpler cells have volume less than half the original, then this algorithm will in fact choose the cut that results in the largest such cell. If there are cells bigger than half, then the biggest one might not be chosen; condition (3) is well satisfied, however.

B. A* Cost Functions

While searching for a path of connected cells, a candidate point path through each partial cell path is "constructed". Each adjacent pair of cells intersect in a cell of one lower dimension. In the case of two dimensions without rotations, the interaction cell is a line segment and the centroid its midpoint, for example. The centroids of such interaction sub cells are used as the entry and exit points of the path through the cell, and a the path segment within a cell is the straight line joining them.

The cost of an individual path segment is just its Euclidean length within the configuration space, and the cost of a path is the sum of the segment costs. The lower bound heuristic estimate for reaching the goal point, is just the Euclidean length from the last point on the path to the goal point.

The running time of the search algorithm can be improved by making the cost of traversing a *mixed* cell greater than the cost of traversing an *empty* cell. The relative cost can be used to trade off running time of the search algorithm for length of the final path. With a large relative cost, the *A** algorithm will strongly tend to concentrate on *empty* cells. Hence, paths that require only a small amount of additional trail blazing through *mixed* cells will be chosen, even if they are very long.

C. Choosing a Point Path

When a final cell path consisting of *empty* cells has been found it is still necessary to find a path for the configuration point of the moving object through configuration space. In general, the choice of final path should be based on domain specific considerations such as kinematic or dynamic characteristics, not on purely geometric criteria. In the absence of domain-specific considerations we suggest that a desirable property of the final path is that it be smooth, essentially nulling out the artificial effects of the tesselation of configuration space by the cell structure. Our approach is to consider alternative paths going through a small set of candidate points in the interaction cell between adjacent cells in the chosen path, and to choose the one that minimizes the cost function by another *A"* search with the same cost function as detailed above.

D. Refining the Search Space

After a search has found a sequence of empty and mixed cells, the *mixed* cells along the path should be cut into at least two sub cells and the search re-executed. This simple strategy leads to a very slowly converging search algorithm, however. The ratio of cutting to searching is too low. Arbitrarily cutting cells can also be wasted effort, however, so it is better to find some selective method of determining places to make more cuts. We use the point path search to identify such places.

The point path search of the previous section is used to find a point path through the *empty* and *mixed* cells. All points at the interface between cells on that path are then examined. It is easy to determine if a point is actually in free space by evaluating the constraint sentence associated with one of the containing cells. If the point is actually in free space, but its cell is mixed, then the cell is repeatedly divided until the point lies in a newly created *empty* cell. (This same procedure is used to cut the space representation about the initial and goal points to get *empty* initial and goal cells for the original search.) The effect of this is to create small islands of empty cells along the trajectory, which will be useful for defining recursive subproblems (see next section).

Initially there may be many large cells in the configuration space representation. If an arbitrary point on the interface between two adjacent cells were chosen as a center of refinement of the cell representation, it could well be that it is far from the optimal path. By only expanding points on a candidate optimal path, the overall search operation is not misled into grossly suboptimal areas of "easy pickings".

E. Divide and Conquer

The efficiency of the search for a *free* cell path can be greatly improved by hierarchically decomposing it into more localized and smaller problems. The search space is smaller for a localized problem, resulting in reduced time for the overall search.

The problem is how to break up a global search into smaller ones when there is no a *priori* knowledge of what the smaller subproblems are. The approach taken here is to use the global search through *mixed* cells, and subsequent selection of a point path as a plan for the lower level subproblems. If there are points on the path that, after cell refinement as above, are in a known *empty* cell, then they are used as points at which the problem is broken up. The global problem then becomes a series of local ones, each from an initial point to a goal point where both are in empty cells.

Surprisingly, the order in which the subproblems are attempted is important. This is because each subproblem may require that the global data base, the cell connectivity graph, be altered, as *mixed* cells are cut. If the subproblems arc evaluated with the one closest to the global initial point first, and followed sequentially by those closer and closer to the goal point, then this cutting has adverse effects on search efficiency.

Consider the first subproblem in a sequence. In general, many cells near the goal point will have been refined by the time a sub path through empty cells has been found. Now the second subproblem is attacked. Its initial point is the goal point of the previous subproblem. When the *A* search starts at that point it finds many small *empty* cells from the recent search. These cells are in the wrong direction relative to the new goal, however; they are, nevertheless, "attractive" to the search algorithm, because the evaluation function favors *empty* cells. In addition the *empty* cells are small and, therefore, many can be appended to partial paths near the new initial point without sharp increases in the heuristic estimate of remaining cost. The result is that the *A** algorithm can spend inordinate amounts of time breadth first searching backwards over areas covered by the previous subproblem.

Fortunately, this wasted work can be avoided by solving the subproblems in the reverse order taking the one nearest to the global goal point first.

Another consideration in the use of divide and conquer is the possibility that a subproblem may be poorly chosen so that there is no very direct path between its initial and goal points. This can be the case even when the global problem is soluble. If unlimited effort is expended on solving the subproblem then it can happen that the final path "backs" out from the initial point, finds a path around the obstacles that block a more direct route from the initial to final position, then "backs" into the goal position. The two subproblems on cither side backtrack over those initial and final parts of the path. See Figure 10 for an illustration of this effect.

In such cases, after some effort has been expended to determine that the direct path is blocked, it is better to back up to the more global problem and look for a new path through mixed cells. This can be simply achieved by placing a path length limitation on subproblems. Once it is exceeded the subproblem exits with failure, forcing a new search at a higher level. In the examples given in this paper the cost limitation for subproblems is 1.5 times the cost of the global path, or 1.5 times the best cost ever computed for the sub-path, whichever is smaller.

IV. Conclusion

The algorithm described here has been implemented and tested on many randomly generated examples. Figure 11 shows two difficult cases that the algorithm solves. We believe these to be among the most difficult findpath problems ever solved by a program. The running time of the algorithm on these examples is very high, however. The actual running times for these difficult problems are on the order of tens of minutes of "wall clock" time on a single- user MIT Lisp Machine without floating point hardware; these times also include a very significant paging overhead. Simple problems, of course, run much faster: on the order of tens of seconds to a few minutes. This should be. contrasted with running times on the order of less than a minute for problems of moderate complexity using the algorithm described in Brooks [1982]. This latter algorithm cannot solve problems such as illustrated in Figure 11, however.

While the running times of the algorithm described here could be made significantly shorter by implementation changes, the fact remains that the complexity of the algorithm is high. Informal timing experiments seem to indicate that the algorithm spends much of the running time in the *A** search procedure, which is used at several points in the algorithm. If the number of cells to be searched could be reduced, the running time could be significantly reduced. One approach to doing this is to use an algorithm such as described by Brooks [1982] to identify a likely path, using a smaller and simpler moving object, and then apply the algorithm described here to verify and refine the path for the actual moving object. This would reduce the size of the search space for the algorithm. There are a number of conceptual and technical problems to be solved before this hybrid approach is practical. As of now, the algorithm described here can solve very complex problems, albeit slowly.

The questions (a) through (e) posed in section 1 have been answered for the case of polygons with rotations. Section HA addressed question (d), 1IB and Hc addressed (a), HD and HE addressed (b), HA addressed (c), and HB, HC, HD and HE addressed (e). In generalizations of the solved problem case it is necessary to further refine only a subset of these andsers.

The approach followed in this algorithm can be directly applied to configuration spaces for three-dimensional polyhedra whose orientation is fixed. This case generates a three-dimensional configuration space with linear constraints - all of section 2 applies in simplified form. We believe that the generalization to a four dimensional configuration space, such as for a polyhedra with a single rotational degree of freedom, will be straightforward. A new type of constraint surface must be dealt with, however, arising from the interaction of pairs of edges (Lozano-Perez [1983a]).

The approach could, in principle, also be generalized to configuration spaces of higher dimension, such as those for polyhedra that are allowed to rotate. The actual generalization presents a large number of problems, e.g., the CellCompare operation is substantially more difficult, and the number of cells grows extremely fast. Other algorithms suggested for this general case have the same drawback. The algorithm for the general findpath problem given by Schwartz and Sharir [1982], for example, has a very high polynomial time complexity for a fixed number of degrees of freedom and is exponential in the degrees of freedom. Our belief is that, in practice, the general six degree of freedom problem should be heuristically reduced to cases involving four or fewer degrees of freedom.

ACKNOWLEDGEMENTS

The presentation of this paper has benefitted greatly from] the detailed comments of John Hollerbach and the IJCAI referees.

REFERENCES

Brooks, Rodney A. (1982). *Solving the find-path problem by representing free space* as *generalized cones,* Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Al Memo 674, May.

Brooks, Rodney A. and Tomas Lozano-Perez (1982). A *Subdivision Algorithm in Configuration Space for Findpath with Rotation,* Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Al Memo 684, December.

Lozano-Perez, Tomas (1981). Automatic *Planning of Manipulator Transfer Movements,* IEEE Trans, on Systems. Man and Cybernetics (SMC II):681 -698.

———(1983a). *Spatial Planning: A Configuration Space Approach,* IEEE Trans, on Computers (C 32): 108-120.

———(1983b). Task *Planning,* in *Robot Motion: Planning and Control,* M. Brady e*t al.* eds., MIT Press.

Lozano-Perez, Tomas and Michael A. Wesley (1979). *An algorithm for planning collision-free paths among polyhedral obstacles,* Communications of the ACM (22):560-570.

Moravec, Hans P. (1980). *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover,* Ph.D. Dissertation Stanford AIM-340, Sept.

Nilsson, Nils J. (1971). *Problem Solving Methods in Artificial Intelligence,* McGraw-Hill, 1971.

Schwartz, Jacob T. and Micha Sharir (1981). *On the Piano Movers Problem I: The Case of a Two-Dimensional Rigid Polygonal Body Moving Amidst Polygonal Barriers,* Department of Computer Science, Courant Institute of Mathematical Sciences, NYU, Report 39, October.

Schwartz, Jacob T. and Micha Sharir (1982). *On the Piano Movers Problem IT. General Properties for Computing Topological Properties of Real Algebraic Manifolds,* Department of Computer Science, Courant Institute of Mathematical Sciences, NYU, Report 41, February.

Udupa, Shriram M. (1977). *Collision Detection and Avoidance in Computer Controlled Manipulators,* Proceedings of IJCAI 5, MIT, Cambridge, Ma., Aug. 1977, 737-748.

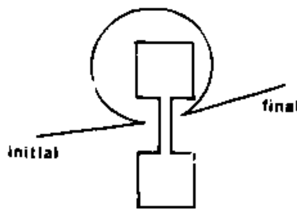
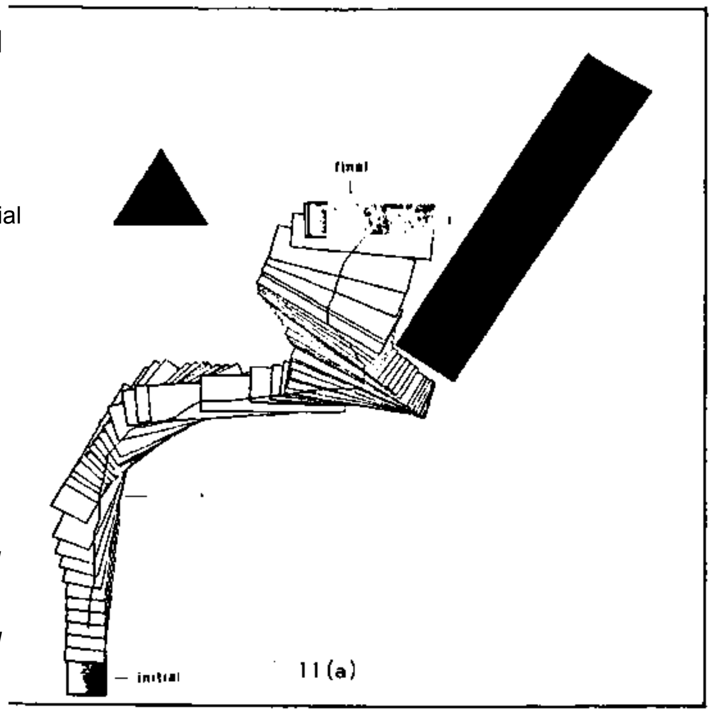


Figure 10. If no restriction is placed on the path length used to solve a subproblem, very inefficient paths may be found as illustrated here.



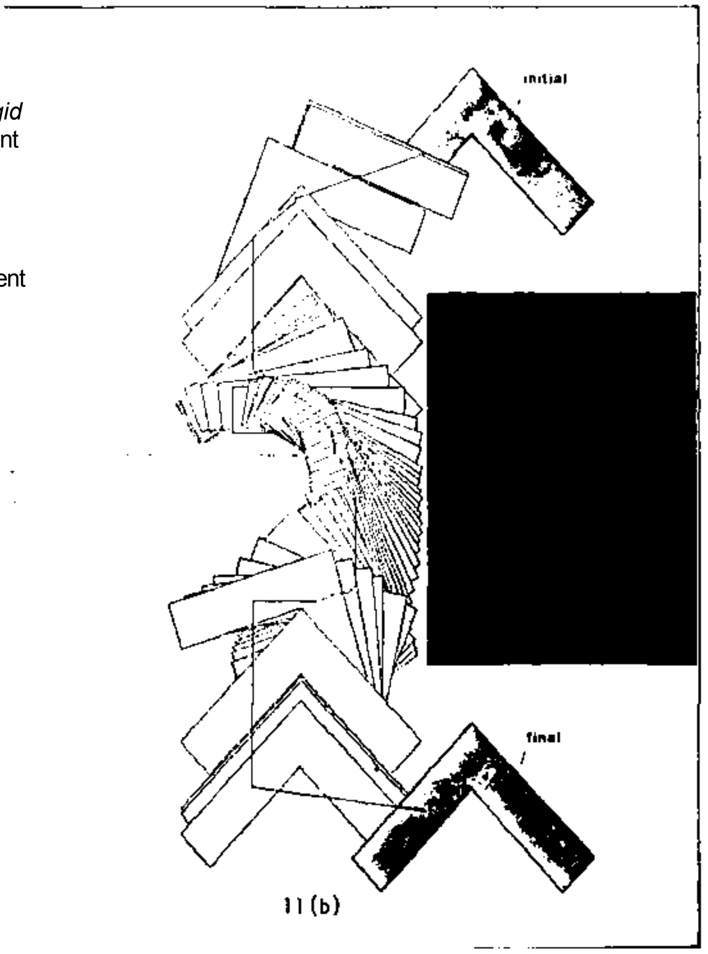


Figure 11. Paths found by the algorithm for (a) a convex moving object and (b) a non-convex moving object. The final representation of configuration space for problem (a) has 766 cells of which 122 are full, 524 are mixed and 120 are empty. The cells are linked by 2157 connecting arcs and the final path goes through 87 of the empty cells (i.e. 71.3%). For (b) there are 2138 arcs linking 1063 cells; 336 full, 570 mixed and 127 empty, while the final path goes through 79 of these (i.e. 62.2%).