Scientific Applications | F.N. Fritsch Editor

An Algorithm for Planning Collision-Free Paths Among Polyhedral Obstacles

Tomás Lozano-Pérez and Michael A. Wesley
IBM Thomas J. Watson Research Center

This paper describes a collision avoidance algorithm for planning a safe path for a polyhedral object moving among known polyhedral objects. The algorithm transforms the obstacles so that they represent the locus of forbidden positions for an arbitrary reference point on the moving object. A trajectory of this reference point which avoids all forbidden regions is free of collisions. Trajectories are found by searching a network which indicates, for each vertex in the transformed obstacles, which other vertices can be reached safely.

Key Words and Phrases: path finding, collision-free paths, polyhedral objects, polyhedral obstacles, graph searching, growing objects

CR Categories: 3.15, 3.64, 3.66, 8.1

Authors' present addresses: T. Lozano-Pérez, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA 02139; M.A. Wesley, IBM Thomas J. Watson Research Center, P.O. Box 218, Yorktown Heights, NY 10598.

1. Introduction

The problem of avoiding collisions when operating on computer models of physical objects is central to model-based manipulation systems. This paper describes an algorithm for planning safe, that is collision-free, paths for a polyhedral object among similarly described obstacles.[1] The algorithm is required to:

(1) find safe paths that might involve going near obstacles, and
(2) guarantee that these paths are short relative to a prespecified distance metric.

The simplest collision avoidance algorithms fall into the generate and test paradigm. A simple path from start to goal, usually a straight line, is hypothesized and then the path is tested for potential collisions. If collisions are detected, a new path is proposed, possibly using information about the detected collision to help hypothesize the new path. This is repeated until no collisions are detected along the path. Roughly, the three steps in this type of algorithm are:

(1) calculate the volume swept out by the moving object along the proposed path,
(2) determine the overlap between the swept volume and the obstacles, and
(3) propose a new path.

The second step, determining the overlap between the swept volume and the obstacles, is also known as an intersection or interference calculation [2, 3]. Current computer modeling techniques employ large numbers of simple surfaces to model accurately even the most common objects. It can be quite difficult to determine whether two such models overlap. This general method, which we will call the *swept volume* method, has a more fundamental drawback. The problem is in the relationship between the second and third steps. Each proposed path provides only local information about potential collisions, for example, the shape of the intersections of the volumes involved, or the identity of the obstacle giving rise to the collision. This information suggests local path changes but is not sufficient to determine when a radically different path would be better. This lack of a global view can result in an expensive search of the space of possible paths with a very large upper bound on the worst case length of the path.

A radical alternative to the swept volume method is to compute explicitly the constraints on the position of the moving object relative to the obstacles. The desired trajectory is the shortest path which satisfies all the position constraints. If the objects are modeled as collections of convex polyhedra, the position constraints can be stated in terms of the position of the vertices of the mov-

[1] We will henceforth use the term "polyhedron" for closed figures bounded by "flats" in two or three dimensions.

ing object relative to the planes of the obstacle surfaces. The trajectory problem can then be posed as an optimization problem as in Ignat'yev [5]. The difficulty with this formulation is that these position constraints, although linear, do not all apply simultaneously. It is not necessary for each point on the moving object to be outside *all* the planes of the obstacles; it is sufficient for each point to be outside *at least one* of the planes of each obstacle. This property makes traditional linear optimization methods inapplicable.

The algorithm presented in this paper is closely related to the optimization approach. The constraints on the position of an arbitrary reference point on the moving object are computed. Polyhedral obstacles in two or three dimensions give rise to sets of polyhedral *forbidden regions*; that is, regions corresponding to positions of the reference point where collisions would occur. This transformation reduces the problem of finding a safe path for the polyhedron to the simpler problem of finding a safe path for a point. This last task is accomplished by finding a path through a graph connecting vertices of the forbidden regions.

The technique of computing the position constraints on an object as constraints on a reference point is extremely powerful and has been applied independently to different problems. It has been used by Udupa [9] for planning safe paths for computer-controlled manipulators, by Lozano-Perez [6] for identifying feasible grasp points on an object, and by Adamowicz and Albano [1] for two-dimensional template layout.

Udupa uses a simple "growing" transformation on obstacles to compute approximations to the forbidden regions for the three-dimensional reference point of a three degree of freedom subset of a manipulator. The system maintains a variable resolution description of the legal positions of the reference point (the *free space*). Safe paths for the subset manipulator are found by recursively introducing intermediate goals into a straight line path until the complete path is in free space. This method has two drawbacks:

(1) Because the complete manipulator has more than three degrees of freedom, the three-dimensional forbidden regions cannot model all the constraints on the manipulator. When a trajectory fails, Udupa's system makes a correction using manipulator-dependent heuristics. The use of heuristics tends to limit the performance of the algorithm in cluttered spaces.

(2) The recursive path finder uses only local information to determine a safe path and therefore suffers from some of the same drawbacks as the swept volume method.

The algorithm presented in this paper uses a more accurate growing operation to compute the forbidden regions in both two and three dimensions. It introduces a graph searching technique for path finding which produces optimum two-dimensional paths when only translations are involved. This technique is then generalized to deal with three-dimensional obstacles and extended to deal uniformly with more than three degrees of freedom. The resulting algorithm no longer guarantees optimum paths. This algorithm has been used to plan safe trajectories for a seven degree of freedom manipulator. These trajectories have been successfully executed.

A detailed survey of previous work in collision avoidance, specifically in connection with computer-controlled manipulators, can be found in Udupa [9].

The nature of the models used for the obstacles affects the details of any collision avoidance algorithm. For concreteness, the detailed discussions and examples in this paper assume that all objects are modeled as sets of, possibly overlapping, convex polyhedra. Any object can be modeled to any desired degree of accuracy in this fashion. A method for finding collision-free paths for a single convex polyhedron among sets of convex polyhedra can be simply extended to plan safe paths for a complex moving object among complex obstacles. The extension involves finding the constraints due to each of the convex components of the moving object relative to each of the components of all obstacles. The constraints for the composite moving object are the union of the constraints on its components.

The collision avoidance algorithm is defined for three dimensions. However, the presentation is easier to follow in two dimensions; for clarity the next sections first develop the complete algorithm for the two-dimensional case and then consider the extension to three dimensions. Section 2 presents a simple form of the algorithm for the case of a polygonal object translating in the plane among polygonal obstacles. Section 3 considers the effect of allowing the moving object to rotate as well as translate. Section 4 deals with more complex moving objects with more degrees of freedom. Section 5 discusses generalization to three dimensions. Discussion of the two steps of the algorithm that are directly affected by the choice of modeling methodology is relegated to the appendices. These steps will be functionally described in the body of the paper.

2. Collision Avoidance on the Plane

Consider the problem, shown in Figure 1, of moving a point object A from position S to position G while avoiding the obstacles (shown shaded); the shortest collision-free path from S to G is also shown. The important property of this path is that it is composed of straight lines joining the origin to the destination via a possibly empty sequence of vertices of obstacles. In the case of motion in the plane with arbitrary polygonal objects, the shortest collision-free path connecting any two accessible points always has this property.

The undirected graph VG(N, L) is defined: The node

Fig. 1.

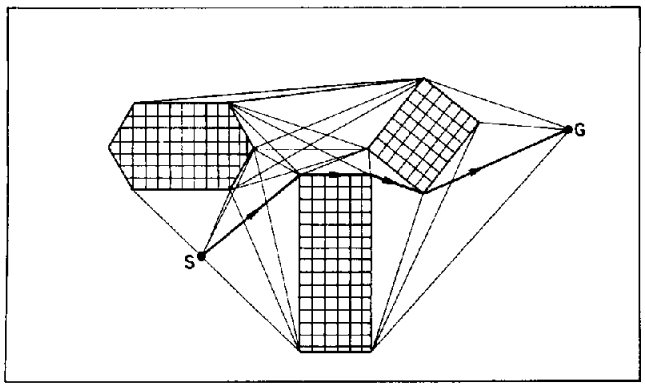


Fig. 2.

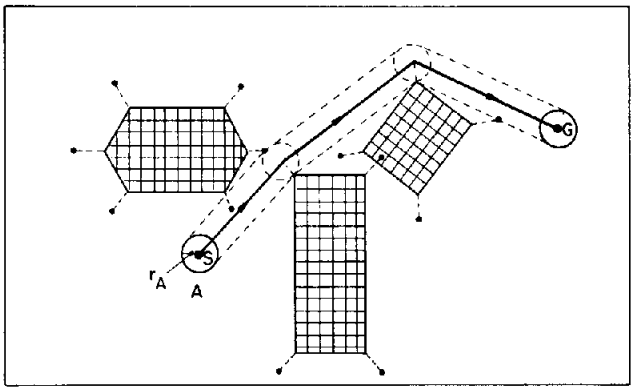


Fig. 3(a).

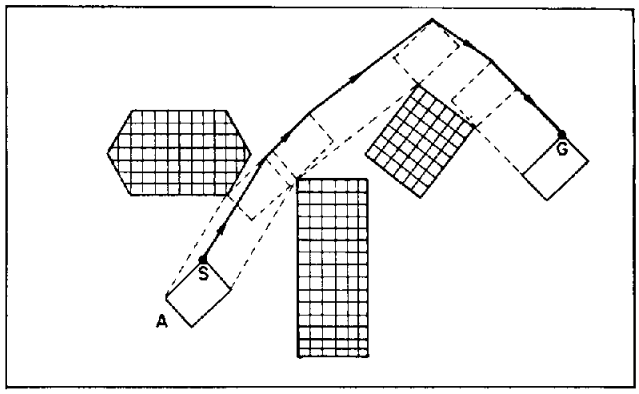
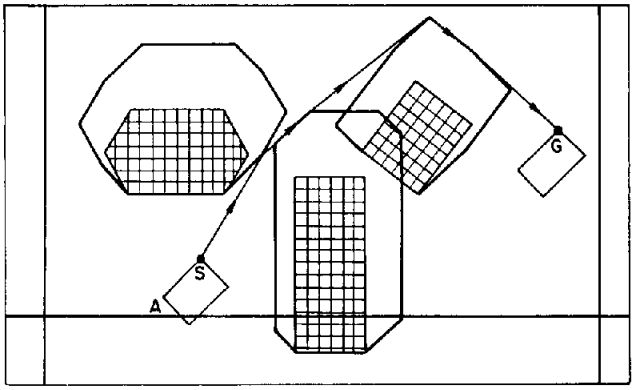


Fig. 3(b).



set N is $V \cup \{S, G\}$ where V is the set of all vertices of obstacles and the Link set L is the set of all links (n_i, n_j) such that a straight line connecting the ith element of N to the jth does not overlap any obstacle. The graph VG(N, L) is called the *visibility graph* (VGRAPH) of N since connected vertices in the graph can see each other. The VGRAPH is shown in Figure 1. The shortest collision-free path from S to G on the plane is the shortest path in the VGRAPH from the node corresponding to S to that corresponding to G when the euclidean metric is used on the links. We will call this method for finding collision-free paths for a point by finding the shortest path in a visibility graph the VGRAPH algorithm. This method was used for navigating SHAKEY [8], an early robot vehicle, and is also described in some detail in Ignat'yev [5].

The simplicity of the VGRAPH algorithm stems from the fact that the moving object A is a point. This is a good approximation for moving objects which are small in relation to the obstacles, but causes problems otherwise. Ignat'yev [5, p. 241] puts it as follows:

> The robot begins to move from the point y_0 (S in our example) along the direction to the x_{19} (a vertex). Here he must consider his dimensions in order not to run into the obstacles and walls.

This paper shows how a more general form of the collision avoidance problem can be reduced to the VGRAPH problem. In other words, it concerns how the robot "must consider his dimensions."

A simple generalization of the problem in Figure 1 is to make the moving object A a circle with nonnegligible radius r_A. The VGRAPH algorithm can be adapted to this situation by moving the vertices away from the obstacles so that they are at least r_A away from all the sides (Figure 2). Moving A so that its center point moves through the new displaced vertices will still produce a minimum distance, collision-free path. Notice, however, that the path found is different from that in Figure 1. This technique of displacing the vertices was also used in SHAKEY [8].

The VGRAPH algorithm requires that the moving object be a point; the obstacles then represent the forbidden regions for the position of that point. If the moving object is not a point, a new set of obstacles must be computed which are the forbidden regions of some reference point on the moving object. These new obstacles must describe the locus of positions of this reference point which would cause a collision with any of the original obstacles. The displaced vertices of Figure 2 are, in fact, approximations to the vertices of these new obstacles when the reference point is the center of A.

The operation of computing a new obstacle O' from an original obstacle O and a moving object A will be called *growing O by A*. This name reflects the fact that

the obstacles are being grown so that the moving object can be shrunk to the reference point. The result of growing a set of obstacles by A will be indicated by GOS(A), i.e., the *Grown Obstacle Set* of A. Note that the growing operation is closely related to that of deriving the path of a machine tool to cut out a part.

Consider the situation in Figure 3(a). The same obstacles in Figures 1 and 2 are shown but the moving object A is now a rectangular solid. Figure 3(b) shows the obstacles after they have been grown by A. It also shows the shortest collision-free path for A's reference point from S to G. This figure demonstrates how the process of growing obstacles allows representing A as a point. Notice that the boundary of the obstacle space is treated as an obstacle and is also grown, thus avoiding paths which involve moving outside the space.

The growing operation was defined as computing the locus of positions of the moving object's reference point that would cause a collision with a given obstacle. The position of the moving object has been interpreted as its (x, y) position, i.e., the grown obstacles are polygons in (x, y) space. This is an arbitrary but natural choice. Different types of moving objects would call for different choices. Figure 4(a) shows one such case in an (x, y) coordinate system. The moving object A can rotate about a fixed point and can change length. This defines a polar coordinate system (r, α). Figure 4(b) shows the region of the (r, α) space which is forbidden to the tip of A by the presence of the obstacle in Figure 4(a); an alternative way of representing this region is shown in (x, y) coordinates in Figure 4(c). The choice of representation depends on:

(1) the ease of computing the forbidden regions, i.e., growing the obstacles, versus
(2) the ease of building the VGRAPH from the grown obstacles.

The use of polyhedra as the basic unit of shape description influences our choice of obstacle representation. Polyhedra (polygons when on the plane) have boundaries which are linear equations in the coordinate variables. This property makes them computationally attractive. In this section we have represented objects as polygons in a planar cartesian coordinate system. The natural choice is to express the grown obstacles in the same space, thus making the growing operation a mapping from polyhedra to polyhedra. Notice that in Figure 4(b) the object O was interpreted as a polygon in (x, y) space and the resulting grown obstacle O' in (r, α) is not a polygon in that space.

Another factor in the choice of obstacle representation is the shape of the path between two nodes in a VGRAPH. A link connecting two nodes in the VGRAPH implies that the path between the corresponding locations does not overlap any of the obstacles. Paths have so far been shown as straight lines in cartesian space; since the grown obstacles were in this coordinate space, the use of straight lines simplifies the detection of overlap. Of course, paths could be more complicated curves which are best expressed in different coordinate systems. For example, the object in Figure 4 might move in straight lines in the (r, α) system. In that case it might be more efficient to use the polar form of the grown obstacles in detecting overlap.

The choice of representation for the grown obstacles depends on the geometric details of the application domain. The choice should be made so as to simplify the overall computation. For the sake of simplicity the next section will continue to assume that the grown obstacles are polygons in (x, y) space.

Fig. 4(a).

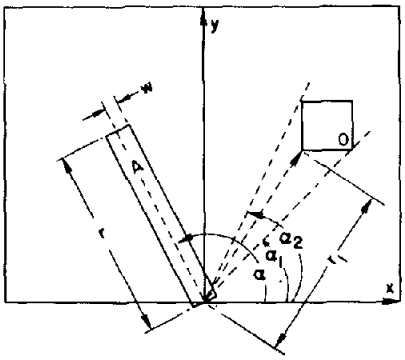


Fig. 4(b).

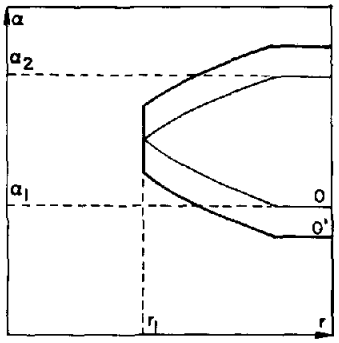


Fig. 4(c).

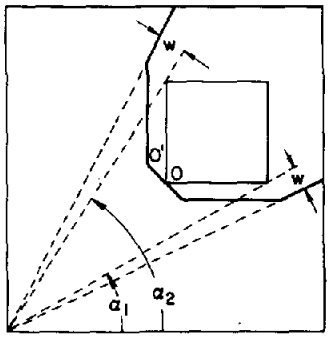


Fig. 5.

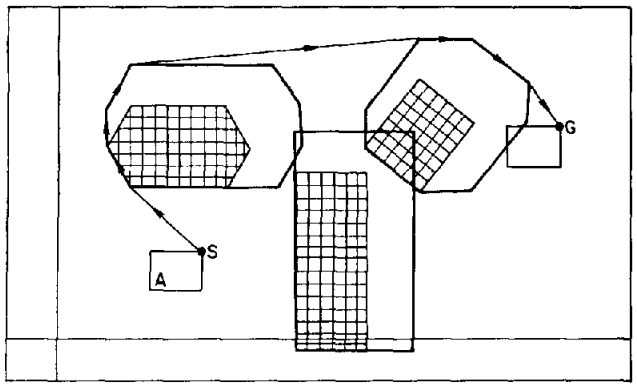
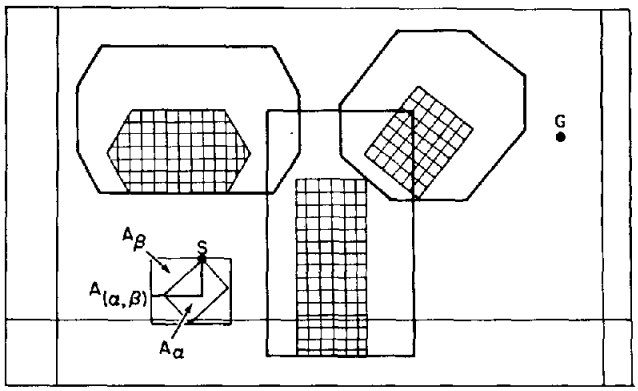


Fig. 6.



3. The Effect of Rotation

It is important to notice that the growing operation as shown in Figures 2 and 3 is sensitive to the orientation of A. This was not apparent in Figure 2 because the moving object was a circle. The orientation dependence follows from the fact that a grown obstacle is defined as the forbidden region for a reference point. The position of a point on the plane can encode only two degrees of freedom, whereas differentiating the legality of two positions of A with different orientations requires at least three degrees of freedom. Figure 5 shows that a different orientation of A from that in Figure 3 will produce different grown obstacles and a different path. To make the orientation explicit, we will denote the result of growing all the obstacles with a moving object A, whose orientation parameter is the angle α, GOS(A_α). The set of vertices of these grown obstacles will be called V_α.

To summarize, any position of A at orientation α for which A's reference point is outside all the elements of the grown obstacle set is free of collisions. The sides of each obstacle in GOS(A_α) are computed by tracing the path of A's reference point around each of the original objects while keeping A in contact with the obstacle. Before two objects collide they must first touch; therefore any position of the reference point that would cause a collision must be inside the obstacle, and any position outside must be safe. Clearly this condition presupposes that the orientation of A does not change.

Consider the problem of moving object A from position S with orientation α to G with a different orientation β. A safe trajectory cannot be found by simply computing a path that is free of collisions in GOS(A_α) and GOS(A_β) since, in changing the orientation from α to β, A must pass through the whole range of intermediate orientations. One way to find a path requires knowing what positions on the plane will allow the desired rotation to take place. The algorithm can then plan a path from the start to one of these positions, rotate to the desired orientation, and move in that orientation to the goal.

For a position to allow a change in orientation there must be no overlap between the rotating object in any of its intermediate orientations and any of the obstacles. Figure 6 shows the area that A traverses in going from orientation α to β; this area may be approximated by another polygon $A_{[\alpha,\beta]}$ shown rectangular for simplicity. This new object, called an *envelope,* can be used to grow a new obstacle set GOS($A_{[\alpha,\beta]}$), also shown in Figure 6, which represents the forbidden regions for the reference point of A in any of the orientations within the interval $[\alpha, \beta]$. We will refer to this as a *transition obstacle set.* By analogy to the vertex set V_α, the set $V_{[\alpha,\beta]}$ represents the set of vertices of obstacles in the transition obstacle set. In general we can associate with all the elements of a vertex set an orientation interval (possibly singular) as well as a position.

The problem in Figure 6 can now be solved by:

(1) finding a path starting with orientation α at S which avoids the obstacles in GOS(A_α) and which ends at a point clear of the obstacles in GOS($A_{[\alpha,\beta]}$),
(2) rotating to orientation β, and
(3) finding a path to G avoiding the obstacles in GOS(A_β).

This can be stated as a VGRAPH problem of finding the shortest path from S to G in a visibility graph defined as follows:

$$VG_{\alpha,\beta}(N_{\alpha,\beta}, L_{\alpha,\beta})$$

where

$$N_{\alpha,\beta} = V_{[\alpha,\beta]} \cup V_{[\alpha,\alpha]} \cup V_{[\beta,\beta]}$$
$$V_{[\alpha,\alpha]} = V_\alpha \cup \{S\}$$
$$V_{[\beta,\beta]} = V_\beta \cup \{G\}$$
$V_{[\alpha,\beta]}$ defined as above

and

$$L_{\alpha,\beta} = \{(n_i, n_j)\}$$

$n_i \in V_{[a,b]}$ and $n_j \in V_{[c,d]}$ where a, b, c, d are either α or β

such that the following *visibility conditions* hold on the link:

(1) the orientation intervals $[a, b]$ and $[c, d]$ must not be disjoint,
(2) n_i is outside all the obstacles in $\mathrm{GOS}(A_{[a,b]})$,
(3) n_j is outside all the obstacles in $\mathrm{GOS}(A_{[c,d]})$,
(4) the path from n_i to n_j either:
  (a) does not overlap any obstacle in $\mathrm{GOS}(A_{[a,b]})$, or
  (b) does not overlap any obstacle in $\mathrm{GOS}(A_{[c,d]})$.

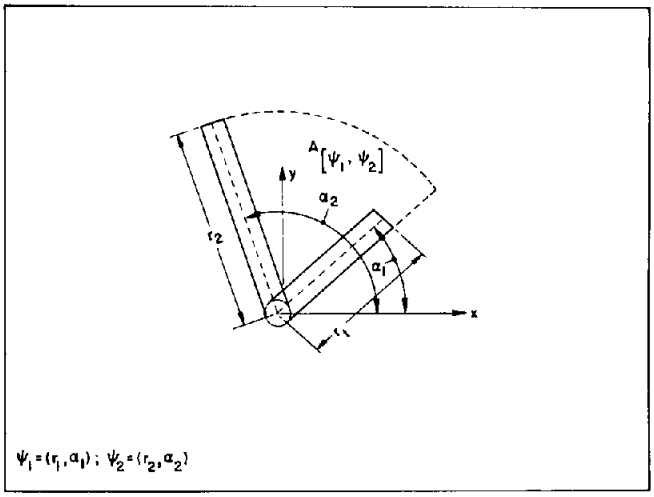
A solution path in $\mathrm{VG}_{\alpha,\beta}$ is a sequence of nodes starting at S and ending at G:

$S, n_1, n_2, \ldots, n_k, G$

in which adjacent nodes are connected by a link in $L_{\alpha,\beta}$. *Each* $n_j \in V_{[a,b]}$ is defined such that if n_j is outside all obstacles in $\mathrm{GOS}(A_{[a,b]})$, then the reference point of the moving object A can be at position n_j in any orientation within the interval $[a, b]$ without danger of collisions. Following the link from n_j to n_{j+1} means that the reference point of A must make the corresponding translation. Also, if n_j and n_{j+1} belong to different vertex sets, $V_{[a,b]}$ and $V_{[c,d]}$ respectively, then a change of orientation may also be required. The conditions on $L_{\alpha,\beta}$ require that the orientation intervals corresponding to the endpoints of a link must not be disjoint. This means that there is some orientation x, such that if $a \le b$ and $c \le d$ then $\max(a, c) \le x \le \min(b, d)$, for which A can safely be at either node of the link. Moving along the link requires first rotating to the orientation x and then translating from the first node to the second. Since the translation happens in an orientation compatible with both nodes of the link, the visibility conditions on the link require only checking for overlap with the obstacles in the obstacle set of *either* one of the nodes. Alternatively, if the path from n_j to n_{j+1} is outside all obstacles in both $\mathrm{GOS}(A_{[a,b]})$ and $\mathrm{GOS}(A_{[c,d]})$, then the rotation may take place in conjunction with the translation along the link.

The use of transition sets, e.g., $\mathrm{GOS}(A_{[\alpha,\beta]})$, has two important drawbacks. The shortest solution path in $\mathrm{VG}_{\alpha,\beta}$ is no longer guaranteed to be an optimum solution to the original problem, and failure to find a solution path in the VGRAPH does not necessarily mean that no safe trajectory exists. The reasons are twofold. The first and most basic is that paths found in this VGRAPH will change the orientation of the moving object only at locations where the full rotation can be performed. If the optimum path involves traversing a narrow passage where the orientation of A must be within a small subrange of the orientations between α and β, then this path could not be a solution path in this version of the VGRAPH algorithm. Secondly, even if the first problem were avoided, the current formulation considers orientations only in the range $[\alpha, \beta]$; it could not negotiate a passage where the moving object could only fit at an orientation outside the specified range. The latter problem can be solved simply by expanding the orientation interval, but only at the expense of making the former problem worse.

Fig. 7.



The two problems mentioned above can be alleviated by replacing the single transition obstacle set, GOS $(A[\alpha, \beta])$, by the union of several other obstacle sets, each generated with a smaller orientation interval for the moving object. In this fashion the range of legal orientations can also be extended beyond the interval $[\alpha, \beta]$. As the number of transition obstacle sets increases, the VGRAPH becomes a better match to the original problem. Unfortunately, the computational burden also increases rapidly. Each new obstacle set requires growing all the obstacles with the moving object in a new configuration, though the growing operation can be speeded up by using approximations, as will be shown later. Also, the added vertices from the extra obstacle sets make searching the visibility graph much more time consuming. Alternatively, it may be possible to derive automatically transition sets to handle narrow passages specifically, and combine these with wider-range transition sets.

4. More Degrees of Freedom

Transition obstacle sets can be used whenever the moving object has more degrees of freedom than can be represented by a point in the obstacle coordinate space. The only requirement is that it be possible to compute an envelope $A_{[x,y]}$ which is an object of the same type as A, e.g., a polygon, such that any point inside an A_z, $x \le z \le y$, is also inside $A_{[x,y]}$. This object then can be used in the growing operation to generate a transition obstacle set. There are no other restrictions on the nature of the parameter range $[x, y]$; in particular, it need not be an orientation range and both x and y may also be vectors. A point outside all of the obstacles in $\mathrm{GOS}(A_{[x,y]})$ indicates a position where each of A_z's *configuration parameters,* z_i, can safely take on values such that $x_i \le z_i \le y_i$.

Figure 7 repeats the example of Figure 4 except that now the moving object can translate in x and y as well as rotate and change its length. The choice of a coordinate system for the grown obstacles will also determine which of the coordinate variables is to be used for the configuration parameters. For example, if the grown obstacles are represented as polygons in (x, y), then (r, α) are configuration parameters and vice-versa.

Configuration parameters can also be used to deal with a moving object whose shape can change due to changes in the relative positions of its components. The object shown in Figure 8 is composed of two rectangles that are free to rotate about a common point. The shape of this object relative to a stationary obstacle can be described by:

(1) the shape of its components,
(2) their relative displacements,
(3) the two angles θ and ρ indicated in Figure 8.

In this example only the angles can change during a motion; therefore the obstacle set for this moving object must be parameterized by the value of both θ and ρ. Generally the configuration parameters describe not only the global orientation or position of the object but also the relative positions of its components.

In general, objects need not be grown in the full dimensional configuration space; instead, repeated use is made of operations on lower dimensional, partitioned, configuration spaces which allows the growing operation to work in a convenient subspace of the full configuration space. The VGRAPH algorithm described in Sections 2 and 3 remains unchanged except that the scalar parameters and intervals are replaced by vector parameters and intervals.

5. Collision Avoidance in Three Dimensions

The VGRAPH algorithm has so far been presented as an algorithm for collision avoidance on the plane. This section examines how three-dimensional obstacles affect the algorithm. This generalization does not affect the statement of the algorithm but does affect the details of the obstacle growing and graph searching. These subjects are discussed in the appendices.

The generalization to three dimensions has an unfortunate side effect. The shortest path around a polyhedral obstacle does not in general traverse only vertices of the polyhedron (Figure 9). That is, the shortest path in a VGRAPH whose node set contains only vertices of the grown obstacles is not guaranteed to be the shortest collision-free path. In general, the shortest path will involve going via points on edges of the obstacles. Our approach is to introduce additional vertices along edges of the grown obstacles so that no edge is longer than a prespecified maximum length. This method generally results in a good approximation to the optimum path.

The use of three-dimensional obstacles also has a significant effect on the execution time of the algorithm. The three-dimensional growing operation is much more time consuming than the corresponding operation in two dimensions. Grown obstacles in three dimensions are generally much more complex than the underlying objects (Appendix 1). The larger vertex sets also increase the time necessary to search the visibility graph. These effects make the use of approximations necessary for practical applications.

A great saving can be realized by using the detailed growing operation sparingly. Many application domains have the property that the moving object need only be close to obstacles at a small number of points along the path. These *care points* usually include the start and goal of the path. Elsewhere the requirements on the path are less strict; in fact, it is often undesirable to move close to the obstacles when away from the care points. This property can easily be exploited in the VGRAPH algorithm; instead of executing the detailed growing operation on each of the known obstacles, it need only be executed on those obstacles close to the care points. Away from the care points drastic approximations can safely be used. Complex objects, built up from many polyhedra, can be approximated by a single enclosing polyhedron. The moving object can be similarly approximated so as to further simplify the process. In addition, a very simple form of the growing operation (Appendix 1) can be used

Fig. 8.

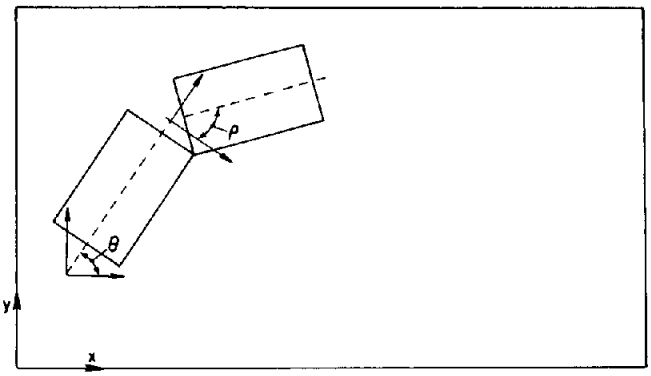


Fig. 9.

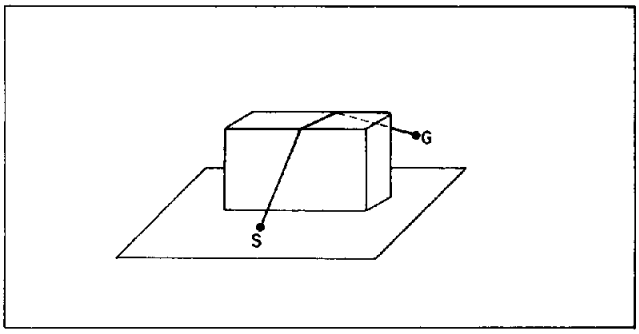


Fig. 10.

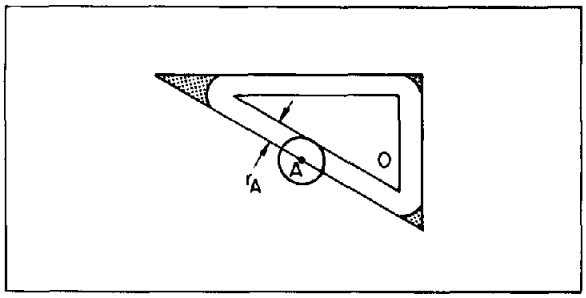
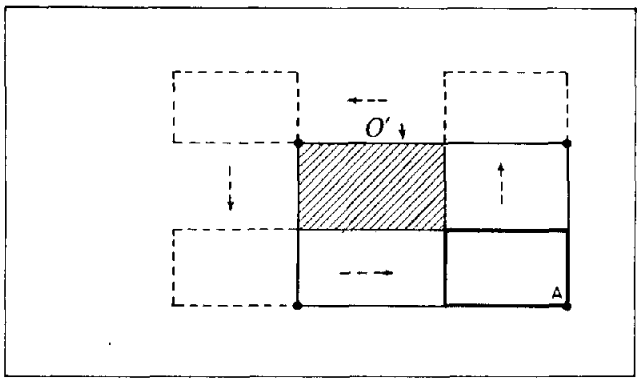


Fig. 11.



which, at the expense of accuracy, is faster and results in simpler objects.

The key to using this approximation technique is an effective way of determining which objects are close to the care points. Clearly a care point is close to an object if it is inside or close to one of the sides of the grown obstacle resulting from it. This means that the moving object when located at the care point is either inside or close to a side of the object. Approximating both the moving and the stationary objects will cause the care point to be inside the grown obstacle. This condition can be used as a criterion for careful growing. When the moving object is large relative to the obstacles, approximating it as a single object results in detailed growing of too many obstacles. The larger the moving object, the worse a simple approximation is likely to be. In particular, some part of the moving object, relatively far from the care point, will cause the grown obstacle to include the care point. The solution is to have a hierarchic decomposition of the moving object; that is, if the test fails for the roughest description, then use a slightly better approximation. In this way the source of potential collision can be better isolated. The other components of the moving object which are not involved need not be considered carefully. Udupa [9] proposed a similar variable level of detail approximation scheme.

Another way to increase the efficiency of the algorithm is to use heuristics in the graph search operation. This is discussed briefly in Appendix 2 which deals with searching the VGRAPH.

6. Summary and Discussion

This paper has shown how the simple visibility graph algorithm used for navigation of SHAKEY [8] can be extended to more general collision avoidance problems. The mechanism necessary to achieve this involves growing the obstacles and shrinking the moving object to a point. This approach has the desirable property of providing two subproblems, growing the obstacles and searching a visibility graph, which can be pursued independently. A description of our current approach to these problems is included in the appendices.

The most important remaining problem with the VGRAPH algorithm is the quantization of configuration parameters into intervals. Paths that require almost continuous changes of orientation as well as position require small quantization intervals, resulting in many transition obstacle sets, and are therefore expensive to compute.

The VGRAPH algorithm as described in this paper has been implemented in PL/1 on an IBM 370/168. It has been used to plan collision-free trajectories for a seven degree of freedom computer-controlled manipulator [10]; these trajectories have been successfully executed in the laboratory.

Appendix 1: Growing the Obstacles

This appendix describes the component of the VGRAPH algorithm that computes from an obstacle description the shape of the forbidden regions for the position of the moving object's reference point. This is called growing the obstacle. The ideas will be developed in two dimensions and then extended to their three-dimensional counterparts. In the initial two-dimensional case the degrees of freedom used for *growing* will be the x and y position of the moving object.

Consider growing a polygonal obstacle by a circular solid as in Figure 10. The simplest growing algorithm moves each of the sides of the original obstacle by a constant amount r_A and then intersects the lines to obtain the vertices of the grown polygon. The drawbacks of this algorithm are twofold:

(1) It works well only for moving objects that are nearly circular.

(2) It generates wasted space near pointed corners, as seen by the dark shaded regions in Figure 10. This problem can be alleviated by clipping the corners of the grown polygon.

This was the form of the growing algorithm used by Udupa [9].

Fig. 12(a).

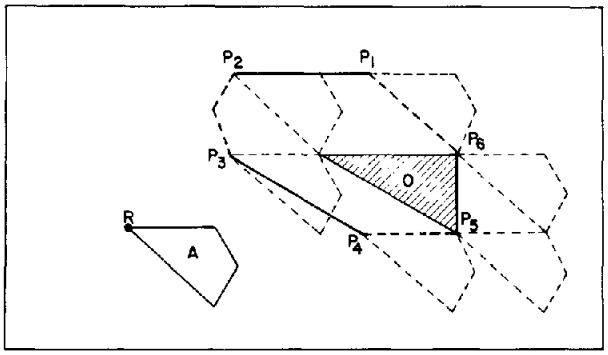
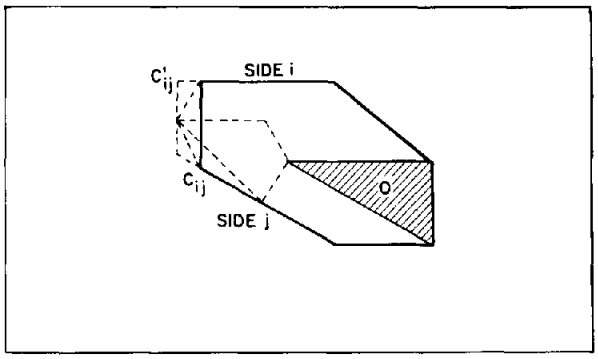


Fig. 12(b).



A simple variation of this procedure will solve problem 1 above. Figure 11 shows a convex polygon O and a moving object A; both are rectangular and both are aligned with the global coordinate axes. R, the reference point of A, coincides with one of the vertices of A. The boundary of the forbidden region for the position of A is the locus of positions of R for which A is in contact with O. This locus defines another convex polygon O' shown in Figure 11. Clearly any point inside this polygon implies a collision between A and O. This grown polygon has side $nside_i$ corresponding to each side $side_i$ of the original obstacle. The distance from $nside_i$ to $side_i$ is the perpendicular distance of R from $side_i$ minus the perpendicular distance to the point where A would first contact $side_i$. The distance from $side_i$ to this contact point on A is the minimum perpendicular distance of all of the vertices of A from $side_i$. Once the sides are displaced by this amount, the lines can be intersected to generate the grown polygon.

This method only makes use of the distance from the reference point to the contact point for a side. In polygons with interior angles of less than a right angle the method described above produces wasted space at the vertices. This waste can be reduced by simply cutting the corner at a conservative distance. A more accurate growing procedure can be obtained by determining the actual locus of motion of R along each $side_i$ as the contact point slides along the side. Figure 12(a) shows the line segments traced out by this procedure (bold lines). Notice that the line segments do not intersect and that the endpoints of these line segments correspond to the position of R when the contact point of A with O is at a vertex of O. These positions will be referred to as *maximal locus points*.

To complete the figure, notice that the locus of R as A moves from its contact point with $side_i$ to its contact point with the adjacent $side_j$ traces successive edges of A between the two contact points on A. In the course of connecting all the maximal locus points, all the edges of A are traced out in reverse order (heavy dashed lines). A simple algorithm for growing convex polygons exists which is based on merging a list of displaced edges of O with a reverse order list of displaced edges of A. To simplify the geometry of the grown object, the locus of R between successive contact points can be conservatively estimated by a straight line c'_{ij} which is parallel to the line c_{ij} connecting the maximal locus points but displaced to the position of the point on the actual locus furthest from the line c_{ij} as shown in Figure 12(b).

The *approximate* method for growing a convex polyapproach is to grow each face of the polyhedron independently and then introduce new faces to complete the grown polyhedron. The steps in the process of growing a rectangular solid O with a rotated rectangular solid A are shown in Figure 13. The locus of R as the contact point of A moves along each edge of $face_i$ is called the *maximal locus edge,* Figure 13(a). Such edges define potential new edges for the faces of the grown polyhedron. Each edge of O generates two adjacent maximal loci. These edges have to be connected in a manner analogous to the way in which maximal locus points are connected in a grown polygon, Figure 12(b). The edge has to be displaced to compensate for points on A which are closer to O than the plane defined by the two adjacent maximal locus edges and passing through the corresponding contact points. Faces are introduced to connect each pair of edges of the grown faces arising from a common edge, Figure 13(b). These new faces introduce new edges, each of which connects two points on the grown faces arising from a common vertex of A. All the edges corresponding to a single vertex also define a new set of faces, Figure 13(c). The total number of faces in a polyhedron grown from an object O in this fashion is equal to the sum of the numbers of faces, edges, and vertices of O.

The operation of growing a polyhedron is related to an operation known as *mixing* polyhedra [7]. A mixed polyhedron is the set of points which can be expressed as a linear combination of points from the two starting polyhedra. A polyhedron isomorphic to a grown obstacle can be obtained by mixing the underlying obstacle with a negative image of the moving object, as in convolution.

Fig. 13(a).

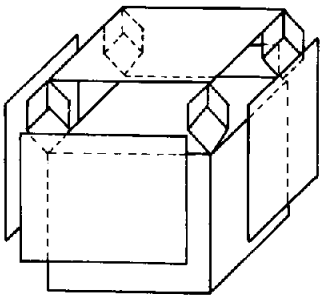


Fig. 13(b).

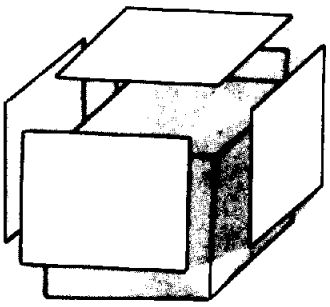
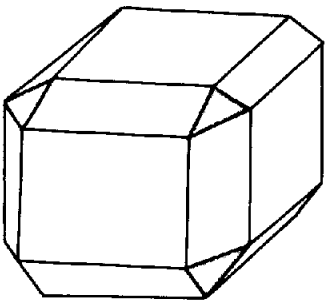


Fig. 13(c).



Appendix 2: Finding a Path

A generalized *visibility graph* VG(N, L) contains a node set N and a link set L_{ij} of links between node pairs (n_i, n_j) for which a visibility function $l(n_i, n_j)$ is true. A node is a representation of a region in an n-dimensional parameter space; for each node the associated region is represented by two n-dimensional parameter vectors ϕ_1 and ϕ_2. Individual elements of the difference vector $\delta\phi = \phi_1 - \phi_2$ will be zero when the corresponding parameter has a fixed value, or nonzero when the parameter has a range of values.

The path finding problem is defined as: Given a node set N with an associated parameter vector set, a start node n_s, and a goal node n_g, find a sequence of nodes from n_s to n_g, by way of an ordered set of intermediate nodes $n_1, \ldots, n_k$, which may be null, such that the visibility function for each node pair in sequence:

$$L_1 = l(n_s, n_1)$$
$$L_2 = l(n_1, n_2)$$
$$\ldots$$
$$L_{k+1} = l(n_k, n_g)$$

is true, and that a cost function

$$C = \Sigma c(n_i, n_j)$$

where $c(n_i, n_j) \geq 0$ is minimized.

A direct approach to finding an optimum path is to enumerate all possible paths and choose one for which C is minimum. For node sets whose cardinality is of practical interest (e.g., > 50) the computational load of the direct approach is prohibitive, and more efficient heuristic based search methods may be used.

The A^* algorithm of Hart et al. [4] allows use of efficient heuristic information. For each node, an estimate *hhat* is made of the cost h to travel from the node to the goal. Initially, n_s is placed on a list of candidate nodes for examination (the OPEN list). At each step of the algorithm, the node with minimum total path cost estimate (i.e., actual cost of reaching the node along the trial path plus *hhat*) is moved onto a CLOSE list and its minimum cost estimate visible successor nodes are placed on the OPEN list.

Hart et al. have shown that the A^* algorithm finds an optimum path when *hhat* is a lower bound estimate of the true cost h. When the estimator for *hhat* is zero and some $c(n_i, n_j) \neq 0$ $n_i, n_j \in N$, the estimate gives no heuristic information to assist in the choice of a path; as $hhat \rightarrow h$, the heuristic information increases and the average number of unsuccessful trial paths is reduced. In the context of this paper, the lower bound requirement for *hhat*, i.e., $hhat \leq h$, may be met by assuming $l(n_i, n_g)$ to be true and computing $hhat_i = c(n_i, n_g)$.

The cost function $c(n_i, n_j)$ may be tailored to suit the requirements of a particular problem environment, for example:

- –distance to be traveled in a subspace of the parameter space;
- –functions of distances in parameter space, for example, time to complete a change based on allowable rate of change of parameters, with the option of selecting the limiting (i.e., slowest) dimension;
- –special costs may be assigned to particular node sequence pairs to allow, for example, costs to be assigned depending on whether the pair allows the motion to proceed without a speed change, as opposed to pairs requiring a change of speed or an intermediate halt.

The form of the visibility function $l(n_i, n_j)$ depends on the semantics of the parameters. Two mutually exclusive classes of parameters are considered as described in Section 3:

—those that may vary continuously, that is, those embodied in the growing operation;
—those that may occupy only discrete ranges, that is, those that are represented by transition obstacle sets.

Note that this distinction between continuous and discrete parameters is an artifact introduced to simplify the handling of spaces of high dimensionality (i.e., $n > 3$) and that the partitioning of the parameter set is not unique: In general, either linear or rotary motions may be represented by continuous or discrete ranges. In the case of discrete range parameters, specific values must be chosen to enable *l* and *hhat* to be evaluated. In all cases of parameter change, a path function defines the motion effect of the change, as either linear or nonlinear motions in parameter space.

In the formalization for path planning described in the body of this paper, an obstacle A is grown under some parameter dependent transformation to produce GOS($A_{\phi 1, \phi 2}$) where ($\phi 1$, $\phi 2$) represents a range of parameters. Three parameters represent continuous motion, and the rest represent discrete motions. The visibility function is line of sight in three-dimensional orthogonal cartesian space, with the provision that visibility is possible only when the discrete parameter ranges at the start and end of the path segment overlap, and values assigned to these parameters are in the overlap region.

In many practical situations, the computational cost of evaluating *l* is very much greater than that of evaluating *hhat* for candidate successor nodes. In such cases it is computationally efficient to select a candidate successor node in terms of minimum *hhat* before computing *l*.

Acknowledgments. We would like to thank P. Will for providing the support, guidance, and encouragement for this work. D. Grossman and L. Lieberman contributed useful discussions and criticism as well as programming advice and assistance. Conversations with R. Taylor triggered the development of the multiple obstacle set approach.

Received June 1978; revised August 1979

References

1. Adamowicz, M., and Albano, A. Nesting two-dimensional shapes in rectangular modules. *Comptr. Aided Design 8,* 1 (Jan. 1976), 27-33.
2. Boyse, J.W. Interference detection among solids and surfaces. *Comm. ACM. 22,* 1 (Jan. 1979), 3-9.
3. Braid, I.C. *Designing with Volumes.* Cantab Press, Cambridge, England, 1973.
4. Hart, P., Nilsson, N.J., and Raphael, B. A formal basis for the heuristic determination of minimum cost paths. *IEEE Trans. Syst. Sci. Cybernetics SSC-4,* 2 (July 1968), 100-107.
5. Ignat'yev, M.B., Kulakov, F.M., and Pokrovskiy, A.M. Robot manipulator control algorithms. Rep. No. JPRS 59717, NTIS, Springfield, Va., Aug. 1973.
6. Lozano-Pérez, T. The design of a mechanical assembly system. Rep. No. AI-TR-397, Artif. Intell. Lab., MIT, Cambridge, Mass., Dec. 1976
7. Lyusternik, L.A. *Convex Figures and Polyhedra.* Dover Publications, N.Y., 1963. (Translated from the Russian by T.J. Smith; original copyright Moscow, 1956.)
8. Nilsson, N.J. A mobile automaton: An application of artificial intelligence techniques. Proc. Int. Joint Conf. Artif. Intell., 1969, pp. 509-520.
9. Udupa, S. Collision detection and avoidance in computer controlled manipulators. Ph.D. Th., Calif. Inst. of Technology, Pasadena, Calif., 1977.
10. Will, P.M., and Grossman, D.D. An experimental system for computer controlled mechanical assembly. *IEEE Trans. Comptrs.* (1975), 879-888.