MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A. I. Memo No. 806 October, 1984

Collision Detection for Moving Polyhedra

John Canny

Abstract. We consider the problem of moving a three dimensional solid object among polyhedral obstacles. The traditional formulation of configuration space for this problem uses three translational parameters and three *angles* (typically Euler angles), and the constraints between the object and obstacles involve transcendental functions. We show that a quaternion representation of rotation yields constraints which are purely algebraic in a higher-dimensional space. By simple manipulation, the constraints may be projected down into a six dimensional space with no increase in complexity. Using this formulation, we derive an efficient *exact* intersection test for an object which is translating and rotating among obstacles.

Acknowledgements. 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 System Development Foundation, 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.

© Massachusetts Institute of Technology, 1984.

1. Introduction

Much recent work in robotics has been directed toward the development of task-level languages. An essential component of an interpreter for a task-level language is a geometric reasoning system which can detect collisions between the manipulator and objects in the workspace, and plan collision-free paths through the workspace. The constraints on the motion of the object caused by the obstacles correspond to surfaces in the configuration space for the object. In this paper we will describe a representation of rotation which simplifies the constraints to purely algebraic form. We will describe an efficient intersection test for an object which is translating uniformly (in a straight line) among obstacles and whose rotation is almost uniform, which corresponds to uniform motion in its configuration space.

A three-dimensional solid object has three translational and three rotational degrees of freedom. Its orientation is typically specified using Euler angles, but this gives configuration space constraints which contain transcendental functions [6]. More recently, Donald [4] described a 3-d path planning algorithm which used angular representation for rotation. Schwartz and Sharir [7] described a representation for rotation using the coefficients of a 3x3 rotation matrix, giving algebraic constraints. However, this approach leads to a twelve dimensional configuration space, and 6 additional constraints must be added to ensure that all configurations lie on a 6-manifold corresponding to valid rotation matrix coefficients.

The orientation of the object may be represented using a four component quaternion [5]. This gives rise to a 7 dimensional configuration space, with one additional constraint to ensure that the quaternion has unit magnitude. Since rotation of a vector by a quaternion involves only multiplication and addition, the constraints will again be algebraic. In fact it is possible to eliminate the extra variable by using a scaling property of quaternions. This step does not increase the degree of the constraints as would, for example, elimination of the extra variable using resultants.

We then describe a procedure for detecting whether a moving polyhedral object collides with polyhedral obstacles, and for finding the collision points. Boyse [1] treated separately the cases of uniform translation and uniform rotation. In the implementation we describe, the object moves with both uniform translation and uniform rotation *through the configuration space*, which corresponds approximately to uniform rotation in real space. By expressing each coordinate of the object as a polynomial function of some parameter that varies along the path, it is possible to detect collisions along more general paths.

The paper is organised as follows: In section 2 we derive the three types of constraint between a polyhedral object and polyhedral obstacles, and show that they are algebraic if a quaternion representation of rotation is used. In section 3 we derive a slightly different representation of rotation which uses only 3 rotational parameters (as opposed to 4 for quaternions). This representation is derived by projecting the 3-sphere of unit quaternions onto a hyperplane. We show that with this representation the constraints have degree 2 as multinomials in the rotation

parameters. In section 4 we derive a predicate which defines the configuration space obstacles, that is, it delimits those regions of configuration space which correspond to interference between object and obstacle. The predicate is constructed from unions and intersections of inequalities in the constraint polynomials. In section 5 we present an intersection test for an object that is moving in a straight line in configuration space, which corresponds roughly to the object translating in a straight line in real space and rotating uniformly as it translates. Finally in section 6 we show how the constraints between a moving object and a stationary obstacle can be generalized to deal with two moving objects. We show that the complexity of collision detection is no higher if both objects are moving.

2. Object-Obstacle Constraints

The interactions between a polyhedral object and obstacles can be divided into three classes. We will use the convention of Lozano-Pérez (1983) and denote the moving object A and the obstacle(s) B. Type-A constraints arise when a vertex of B touches a face of A, Type-B when a vertex of A touches a face of B, and type-C when an edge of A touches an edge of B. The configuration of the object will be given by a position vector $\mathbf{x}$ and a four-component quaternion $\mathbf{Q}$. The interpretation of a configuration $(\mathbf{x}, \mathbf{Q})$ is that the object has been first rotated about the origin by $\mathbf{Q}$ and then displaced by $\mathbf{x}$. For each type of contact there is a corresponding constraint on the configuration of the object, and from the surfaces *in configuration space* defined by the constraints, we derive configuration space obstacles. These are regions in the configuration space which correspond to interference (overlap) between the object and obstacles. We now derive expressions for each type of constraint.

If the object A is at the origin, then a point $\mathbf{y}$ on a face of A must satisfy $\mathbf{y} \cdot \mathbf{n}_A - d_A = 0$ where $\mathbf{n}_A$ is the unit outward normal of the face and d_A is the distance from the origin to the face in the normal direction. Rotation of the face about the origin by $\mathbf{Q}$ gives $\mathbf{y} \cdot \mathrm{Rot}(\mathbf{n}_A, \mathbf{Q}) - d_A = 0$, where $\mathrm{Rot}(\mathbf{n}_A, \mathbf{Q})$ denotes rotation of the vector $\mathbf{n}_A$ by the quaternion $\mathbf{Q}$. Translation of A by $\mathbf{x}$ has the same effect as a translation of $\mathbf{y}$ by $-\mathbf{x}$, so a point lying on the rotated and displaced face of A must satisfy $(\mathbf{y} - \mathbf{x}) \cdot \mathrm{Rot}(\mathbf{n}_A, \mathbf{Q}) - d_A = 0$. For type-A contact, a vertex of the obstacle, $\mathbf{p}_B$ touches the face of A, so substituting $\mathbf{p}_B$ for $\mathbf{y}$, we obtain:

$$\text{Type A} \qquad (\mathbf{p}_B - \mathbf{x}) \cdot \mathrm{Rot}(\mathbf{n}_A, \mathbf{Q}) - d_A = 0 \tag{1}$$

For type B constraints, we begin with the equation of a face of B which is $\mathbf{y} \cdot \mathbf{n}_B - d_B$, where $\mathbf{n}_B$ and d_B are respectively the outward normal and the distance from the origin to the face. We then substitute for $\mathbf{y}$ an expression for a shifted and rotated vertex $\mathbf{p}_A$ of A, which is just $\mathrm{Rot}(\mathbf{p}_A, \mathbf{Q}) + \mathbf{x}$, giving:

$$\text{Type B} \qquad (\mathrm{Rot}(\mathbf{p}_A, \mathbf{Q}) + \mathbf{x}) \cdot \mathbf{n}_B - d_B = 0 \tag{2}$$

For type C constraints, we recall that two non-colinear edges that intersect in 3-space define a plane. The normal to this plane can be found from the cross product of the two edge directions, and the distance of the plane from the origin is the dot product of the normal and any point on one of the edges. More generally, we can test whether two edges intersect by finding the equation of the plane passing through one of the edges and parallel to the second edge, and testing whether a point on the second edge lies in this plane. Suppose the direction of the edge of object A (the vector from its tail vertex to its head vertex) is $\mathbf{e}_A$, and the direction of the edge of the obstacle B is $\mathbf{e}_B$, and there is some point $\mathbf{p}_B$ on the obstacle. Then the plane passing through the edge of B and parallel to the edge of A has equation $(\mathbf{y} - \mathbf{p}_B) \cdot (\mathbf{e}_A \times \mathbf{e}_B) = 0$. To test for intersection, we substitute for $\mathbf{y}$ the coordinates of a point on the edge A, $\mathbf{p}_A$. Since A has been rotated by $\mathbf{Q}$ and translated by $\mathbf{x}$, we have in fact $\mathbf{y} = \mathrm{Rot}(\mathbf{p}_A, \mathbf{Q}) + \mathbf{x}$, and the direction of the edge of A will be $\mathrm{Rot}(\mathbf{e}_A, \mathbf{Q})$. Making these substitutions gives equation (3):

$$\text{Type C} \qquad \mathrm{Rot}(\mathbf{e}_A, \mathbf{Q}) \times \mathbf{e}_B \cdot (\mathrm{Rot}(\mathbf{p}_A, \mathbf{Q}) + \mathbf{x} - \mathbf{p}_B) = 0 \qquad (3)$$

We can expand and rearrange the scalar triple products in (3) to give equation (4) which will prove to be a more convenient form for type C constraints:

$$\text{Type C} \qquad \mathrm{Rot}(\mathbf{p}_A \times \mathbf{e}_A, \mathbf{Q}) \cdot \mathbf{e}_B + \mathrm{Rot}(\mathbf{e}_A, \mathbf{Q}) \times \mathbf{e}_B \cdot (\mathbf{x} - \mathbf{p}_B) = 0 \qquad (4)$$

All these constraints are algebraic, that is, they are polynomial in the components of the position vectors and the quaternion $\mathbf{Q}$. We can see this from the expansion of the quaternion rotation $\mathrm{Rot}(\mathbf{y}, \mathbf{Q})$

$$\mathrm{Rot}(\mathbf{y}, \mathbf{Q}) = \mathbf{y}(q_0^2 - \mathbf{q} \cdot \mathbf{q}) + 2(\mathbf{y} \cdot \mathbf{q})\mathbf{q} + 2q_0\mathbf{q} \times \mathbf{y} \qquad (5)$$

where q_0 and $\mathbf{q}$ are the scalar and vector parts of the quaternion $\mathbf{Q}$ respectively. We can now see why equation (4) is prefered over equation (3) as a representation for type C constraints. Equation (4) gives polynomials of degree 2 in the quaternion components while equation (3) gives polynomials of degree 4.

We note here that although equation (5) is a natural expression for the rotation of a vector using a quaternion, there are alternative forms of (5) which require fewer arithmetic operations. Some of these are given in appendix I. These forms may be used for more rapid computation of rotations for particular configurations, but they do not lead to any simplification of the constraint polynomials.

3. A More Efficient Representation of Rotation

One problem with the constraints presented in the last section is that they are polynomial in 4 quaternion components, while an object has only 3 rotational degrees of freedom. We now show how one of the components can be eliminated without increasing the constraint complexity, and that this new representation is related to the old by a certain projection. We will do this first for type C constraints, and then for types A and B which will have to be modified slightly.

An important property of equation (4) for type C constraints is that all terms on the right hand side are quadratic in the quaternion components. If the quaternion is scaled by some factor α the rotated vector will be scaled by α^2. i.e.

$$\mathrm{Rot}(\mathbf{y}, \alpha\mathbf{Q}) = \alpha^2 \mathrm{Rot}(\mathbf{y}, \mathbf{Q})$$

While we normally think about quaternion rotation in terms of unit quaternions, composition with an arbitrary quaternion $\mathbf{Q}$ corresponds to rotation by a unit quaternion which is $\mathbf{Q}$ normalized, followed by scaling by the square of the magnitude of $\mathbf{Q}$. In the 7 dimensional space of position + quaternion components, the configuration of an object normally lies on a *cylinder* which is the direct product of the 3-space of positions and the 3-sphere of rotations. If we refer back to the form of the type-C constraint, equation (4), we find that scaling the quaternion leaves the sign of the left-hand side unchanged. In other words, for fixed position the manifold defined by this equation extends throughout the 4-dimensional quaternion component space and is radially symmetric about the quaternion origin. If we move along some arbitrary path in the 7 dimensional configuration space without violating any constraints, the radial projection of this path onto the cylinder of unit quaternions will not violate any constraints, and will define a collision-free path.

Given this interpretation of points in the 7-dimensional space, we could try planning paths directly in 7-dimensions, but this is very inefficient since known algebraic algorithms for this problem are worse than exponential in the number of degrees of freedom [7]. A better representation of rotation is a 3-dimensional manifold which is *not* the unit sphere, but a hyperplane. The intersection manifold of the hyperplane with the type-C constraint manifold is very simple, and can be specified by a multinomial of degree 3, whereas the intersection manifold for the sphere requires a multinomial of degree 6. In fact a single hyperplane will only project onto one half of the 3-sphere, but this is enough to represent all possible rotations since $\mathbf{Q}$ and $-\mathbf{Q}$ define the same rotation. The only caveat is that rotations of exactly 180° correspond to points at infinity on the hyperplane.

The projection of constraints from the cylinder of normalized quaternions to a hyperplane is illustrated schematically in figure 1. This figure shows a 3-dimensional configuration space consisting of 2 quaternion components and one position coordinate, and the projection of a single obstacle onto the hyperplane. The hyperplane that we choose is rather arbitrary, and in the collision detector (as

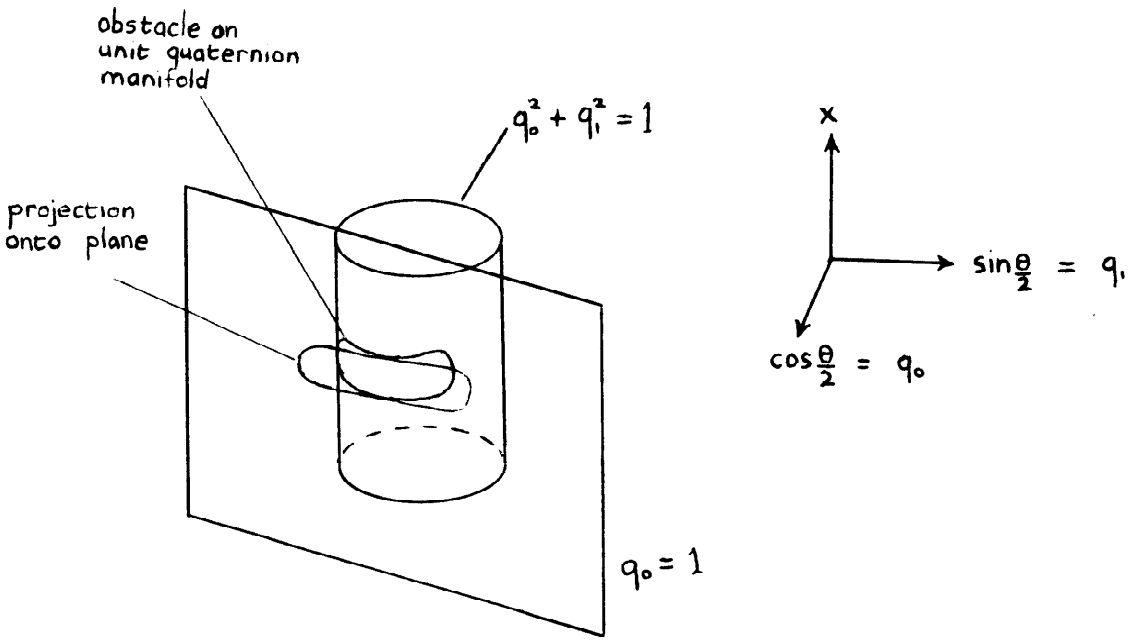

Figure 1. Projection of a configuration space constraint from the cylinder of normalized quaternions onto a hyperplane.

---

in the figure) we have chosen $q_0 = 1$. For this hyperplane the vector part of the quaternion is given by $\mathbf{q} = \mathbf{n} \tan \frac{\theta}{2}$ for a rotation about axis $\mathbf{n}$ through an angle θ.

So far we have shown how to eliminate one quaternion component in the equation for type-C constraints with no increase in complexity. The equations for type-A and type-B constraints do not exhibit invariance to quaternion magnitude, but can be made to by a simple modification. We could rewrite (1) and (2) as a sum of terms containing a rotated vector and those that do not. The former terms will scale by the square of the quaternion magnitude, but the latter will not. However, if we multiply the constant terms by $q_0^2 + \mathbf{q} \cdot \mathbf{q} = \mathbf{Q} \cdot \mathbf{Q}$ which is just the squared magnitude, the expression as a whole will again be scale-invariant.

$$\text{Type A} \qquad (\mathbf{p}_B - \mathbf{x}) \cdot \text{Rot}(\mathbf{n}_A, \mathbf{Q}) - (\mathbf{Q} \cdot \mathbf{Q}) d_A = 0 \tag{6}$$

$$\text{Type B} \qquad \text{Rot}(\mathbf{p}_A, \mathbf{Q}) \cdot \mathbf{n}_B + (\mathbf{Q} \cdot \mathbf{Q})(\mathbf{x} \cdot \mathbf{n}_B - d_B) = 0 \tag{7}$$

Once again these constraints are multinomials of degree 3 in the configuration parameters. So the representation of rotation that we will use henceforth is a 3-dimensional hyper-plane in the 4-dimensional space of quaternions. A configuration

will be represented by position coordinates and the vector quaternion components, i.e. as $(\mathbf{x}, \mathbf{q})$ assuming $q_0 = 1$. By projecting the rotation components of a configuration from the hyperplane onto the unit sphere, we can obtain a physically realizable configuration which has the same relationship to all constraints.

This formulation of configuration space constraints for polyhedra would seem to be the simplest possible. It requires a 6-dimensional space, and all constraints are multinomials of degree 3. In fact all constraints are linear in the translational parameters and quadratic in the rotational parameters. It is felt that these constraints are simple enough that it will be possible to design algorithms to triangulate the algebraic varieties defined by them in reasonable time, and hence to provide an effective procedure for the find-path problem. For the present we describe a simple collision test for a class of paths through the configuration space.

4. Testing Overlap Between Object and Obstacle

In section (2) we presented the configuration space constraints for the three types of interaction between polyhedra. It was assumed in that section that constraint surfaces correspond to the algebraic varieties defined by the constraints, or in other words, to the configurations where the value of a constraint is zero. However if we have an arbitrary point in configuration space which does not lie on one of the constraint manifolds, we need to know if it corresponds to the object being clear of all obstacles (in free space), or to an overlap between the object and some obstacle. We need a predicate which indicates whether a configuration is "safe" relative to a constraint. This is relatively easy to do for types A and B constraints, since both arise between a vertex and a plane, and they depend on a normal which points out (or in depending on convention) of a solid object. If we assume the normal in (6) and (7) points outward, the left-hand sides of (6) and (7) will be positive if the point lies above or outside the surface, and negative if it lies below.

We can now define predicates which test whether a particular configuration is safe for A and B constraints. We define a type-A predicate $A_{ij}(\mathbf{x}, \mathbf{q})$ to be true if the type-A constraint between a face i of A and a vertex j of B is *violated*, i.e. if the left-hand side of (6) is negative. Similarly we define $B_{ij}(\mathbf{x}, \mathbf{q})$ to be true if the constraint between a vertex i of A and a face j of B is violated. Each predicate defines a subset of configuration space in which its constraint is violated, and we will use predicates without arguments to denote these sets $P = \{(\mathbf{x}, \mathbf{q}) \in \Re^6 | P(\mathbf{x}, \mathbf{q})\}$ and with c superscripts to denote their complements (the set of points which do not violate the constraint) $P^c = \{(\mathbf{x}, \mathbf{q}) \in \Re^6 | \neg P(\mathbf{x}, \mathbf{q})\}$.

In the following derivation, the configuration obstacles will be defined using sets but each can be associated with a particular predicate by making the natural link between disjunction of predicates and union of sets etc. We will assume that the object and the obstacles are convex. Complicated obstacles (or objects) can be broken down into unions of possibly overlapping convex objects relatively cheaply, and for collision detection it is unimportant whether obstacles overlap.

Because type C constraints do not seem to give a simple inside/outside predicate, we need to give a slightly more complicated condition (actually two conditions) for type C overlap. We also need the conditions under which type A or B violations occur between *any* features of the object and obstacles. An object A and an obstacle B overlap if either (i) a vertex of B lies inside A, (ii) a vertex of A lies inside B or (iii) an edge of A pierces a face of B, or (iv) an edge of B pierces a face of A. Actually, there are several alternative tests for interference, especially for parts (iii) and (iv), see for example Boyse [1]. The j^{th} vertex of B lies inside A only if all the type-A constraints between the vertex and faces of A are violated, i.e. if $A_{ij}(\mathbf{x}, \mathbf{q})$ is true for all i. An overlap of type (i) occurs if any point of B lies inside A, and the region of type A violation in configuration space is:

$$\bigcup_{j \in vert(B)} \bigcap_{i \in faces(A)} A_{ij} \tag{8}$$

Similarly an overlap of type (ii) occurs if any vertex of A lies inside B, which corresponds to

$$\bigcup_{i \in vert(A)} \bigcap_{j \in faces(B)} B_{ij} \tag{9}$$

The test for a violation of type (iii) is not as straightforward because there is no simple notion of inside and outside for two edges. We avoid the ambiguity by considering an edge of the object and a face of the obstacle. In order for the edge to pierce the face, one vertex on the edge must lie above the plane of the face and one must lie below. These conditions can be easily tested since they derive from type A and B constraints, which have already been computed. The type A and B constraints also tell us the direction of the edge relative to the outward normal of the face, because if the head of the edge lies above the plane of the face and the tail lies below, the dot product of the edge and the outward normal to the face must be positive, and vice versa.

If we compute the type-C constraint between the edge of A and an edge of B *which bounds the face*, we can tell whether the cross product in (3) gives a vector that points towards the interior of the face or away from it. From this we can determine the sign of (3) that corresponds to interference between the edge and the face. If the object edge actually pierces the obstacle face, it must violate all the type-C constraints with edges that bound the face. It must also have one of its end vertices above and one vertex below the face. This later condition is defined by a type B predicate, so the test for interference of an object edge and a face is a conjunction of C and type B predicates. The region in configuration space for which an edge i of the object A pierces a face j of the obstacle, with the edge pointing out of the face, is given by

$$B_{t(i),j} \cap B^c_{h(i),j} \cap \bigcap_{k \in edges(j)} C_{ik} \tag{10}$$

Where $t(i)$ and $h(i)$ are the head and tail respectively of an edge i of A and k ranges over the edges of the j^{th} face of the obstacle. Note that the edges k of the face j are assumed to be directed in a counterclockwise chain around the face, so that the head vertex of one edge is the tail vertex of the next edge. $C_{ik}(\mathbf{x}, \mathbf{q})$ is true if the left hand side of equation (4) is negative for an edge i of A and edge k of B. The object edge will be pointing out of the face in this case because the type B predicate for the tail of the edge must be true (indicating that the tail lies on the obstacle side the face) while the predicate for the head must be false (indicating that the head lies outside the obstacle).

We need to test separately whether the edge pierces the face while pointing *into* it. For the latter test we complement the type B predicates, since the positions of the head and tail of the edge are reversed, and complement the type C predicate, because the sign of the left-hand side of equation (4) changes when the edge direction is reversed. This gives

$$B^c_{t(i),j} \cap B_{h(i),j} \cap \bigcap_{k \in edges(j)} C^c_{ik} \tag{11}$$

and to test whether the i^{th} edge of A pierces the j^{th} face of B in either direction, we simply take the union of (10) and (11).

A violation of type (iii) will occur if any edge of A pierces any face of B, which we can write as

$$\bigcup_{\substack{i \in edges(A) \\ j \in faces(B)}} \left(B_{t(i),j} \cap B^c_{h(i),j} \cap \bigcap_{k \in edges(j)} C_{ik} \right) \cup \left(B^c_{t(i),j} \cap B_{h(i),j} \cap \bigcap_{k \in edges(j)} C^c_{ik} \right) \tag{12}$$

Type (iv) constraints closely resemble type (iii), except that type-A constraints determine the position of the edge end-points. The expression for a type (iv) violation is:

$$\bigcup_{\substack{i \in edges(B) \\ j \in faces(A)}} \left(A_{j,t(i)} \cap A^c_{j,h(i)} \cap \bigcap_{k \in edges(j)} C_{ki} \right) \cup \left(A^c_{j,t(i)} \cap A_{j,h(i)} \cap \bigcap_{k \in edges(j)} C^c_{ki} \right) \tag{13}$$

The object overlaps an obstacle if its configuration is in any one of the regions (8), (9), (12) or (13), and so a single predicate can be defined to test for overlap which is the disjunction of predicates for (8), (9), (12) and (13). If there is more than one object or obstacle, we take an additional disjunction over all object / obstacle pairs. Notice that the final predicate has the form of a disjunction of conjunctions, and that configuration space obstacles can be thought of as unions of "convex" parts (regions defined by a conjunction of constraints). Each of these parts has a straightforward geometric interpretation, given below.

Each conjunction of type A constraints in equation (8) can be thought of as a copy of the object A displaced by the negation of a vector from the origin to some vertex of B. This is exactly the volume in 3-space defined by (8) if the orientation **Q** is fixed. As the orientation **Q** changes, the orientation of the 3-space volume changes. Similarly each conjunction in equation (9) defines a copy of the obstacle B displaced by the negative of some vertex of A. In this case, the orientation of the 3-space volume does not depend on **Q** but its displacement from the origin will change.

The conjunction of predicates in equation (12) defines a 3-space volume which is formed by sweeping a face j of B along an edge i of A. Two of the faces of this volume are shifted copies of the face j, and the others are parallelograms whose edges are copies of edges of j or the edge i. As orientation is varied the two end faces (copies of j) do not change orientation but move relative to each other and the volume skews. Finally the conjunction of predicates in (13) defines a 3-space volume swept out by a face j of A along an edge i of B. When orientation is changed, the orientation of the end faces (copies of j) changes accordingly, but the edges joining the two end faces do not change their orientation.

We note here that although it is necessary to include equations (8), (9), (12) and (13) in a test for overlap between object and obstacles, some simplification is possible for an algorithm which plans paths between safe configurations. This is because if the object starts in a safe configuration, and moves to a configuration of overlap, there must be a violation of type (iii) or (iv) somewhere along the path. That is, somewhere along such a path an edge of the object must pierce a face of the obstacle or vice-versa. In fact any configuration of overlap will include a violation of type (iii) or (iv) unless the object lies entirely inside an obstacle, or an obstacle is entirely contained in the object. The type (iii) and (iv) constraints define in configuration space some connected "free" components, some of which may not really be free, but these will not be reachable from a genuine free configuration. There are two other pairings of constaint types which happen to be complete from the point of view of collision avoidance, namely types (i) and (iii) (any path from a free configuration to an unsafe configuration must violate a type (i) or (iii)), and also types (ii) and (iv).

In defining the overlap predicate we have not made use of explicit "applicability constraints" for the constraint surfaces [2], and in detail in [4]. The applicability constraints delimit ranges of rotations for which a particular type of contact can occur. The configuration space obstacle between a convex object and obstacles can be expressed as a conjunction of applicable constraints (each of which is the disjunction of a constraint and its applicability constraints), instead of a disjunction of conjunctions as we have described above. Effectively we have broken down object-obstacle interactions into interactions between features (edges, faces, vertices) which are simple enough that they are always applicable. Instead we are relying on the "redundancy" [4] which is captured naturally in the disjunctive form of the constraints.

5. An Algorithm for Collision Detection

If instead of fixed values for $\mathbf{x}$ and $\mathbf{q}$ we have functions $\mathbf{x}(s)$ and $\mathbf{q}(s)$ of some parameter s, then the predicates $A_{ij}(\mathbf{x}, \mathbf{q})$ etc. become binary-valued functions of s, and the disjunction of predicates corresponding to expressions (8),(9),(12),(13) defines a binary-valued function which is true for sections of the path where there is interference and false otherwise. Dually, we can take the path defined by $(\mathbf{x}(s), \mathbf{q}(s))$ and intersect it with the union of (8),(9),(12),(13) and we will obtain the subset of the path in which there is interference. The true-false transitions of this function and of any of the predicates $A_{ij}(s)\ldots$ correspond to zero-crossings of one of the expressions (4), (6) or (7), as functions of the parameter s. Finding the zero-crossings of one of these functions gives us the free path segments for the corresponding predicate, and we can readily evaluate (8),(9),(12),(13) by computing unions and intersections of path segments.

If the functions $\mathbf{x}(s)$ and $\mathbf{q}(s)$ are polynomial in the variable s, finding the constraint zero-crossings is particularly simple. When we substitute for $\mathbf{x}$ and $\mathbf{q}$ in equations (4), (6) and (7), we obtain *univariate* polynomials in s, and we can apply standard techniques to find their zeros (Collins and Loos [3]). By considering a particular path through configuration space we have reduced a problem involving polynomials in 6 variables to one involving polynomials in one variable. The simplest possible polynomial path is a straight line in configuration space, which corresponds to linear motion and approximately uniform rotation in real space. An implementation of a collision detector for linear paths is described next.

Let us assume that the object is at some initial configuration $(\mathbf{x}_i, \mathbf{q}_i)$ and moves to some final configuration $(\mathbf{x}_f, \mathbf{q}_f)$. If we assume also that the motion between these configurations is uniform in the configuration space, at any intermediate point the configuration will be given by $(\mathbf{x}_i + s\mathbf{x}_d, \mathbf{q}_i + s\mathbf{q}_d)$ where $\mathbf{x}_d = \mathbf{x}_f - \mathbf{x}_i$ and $\mathbf{q}_d = \mathbf{q}_f - \mathbf{q}_i$. When we substitute this configuration into (4), (6) and (7) we find that all three constraints reduce to univariate cubics in the parameter s. Hence we can find closed form solutions for the zeros of each constraint in terms of the cubic coefficients.

To find the complexity of this method, we break the computation into two steps. The first step is to find all the zeros of the constraints. Since each constraint can have at most 3 zeros, and since there is a simple closed form for the roots of a cubic, we assign a fixed cost for the computation of the zeros of one constraint. The total number of constraints is $N_c = E_A E_B + F_A V_B + F_B V_A$ where F_A, V_A and E_A are respectively the number of faces, vertices and edges of the object A, and F_B, V_B and E_B are the corresponding numbers for the obstacles. The time complexity of the zero-finding phase is then $O(N_c)$.

The second step is the computation of the overlap predicate from the constraint predicates (8),(9),(12),(13). It is possible to compute the intersection or union of two binary-valued functions in time which is linear in the number of true-false transitions if the transitions are ordered in s (by a merge-like operation). We assume that a predicate $P(s)$ is represented as a sorted list of values of s which correspond

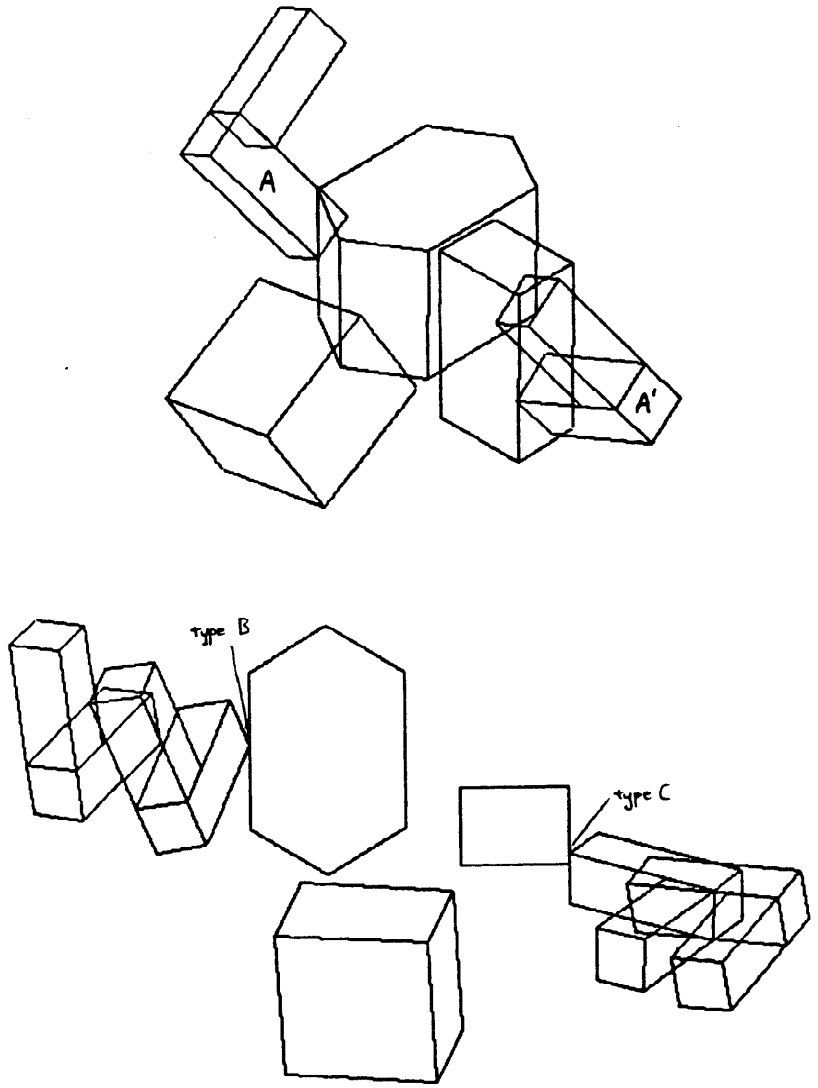

Figure 2. Collision Detection Example, (i) Object starts at A, moves uniformly to A', (ii) collisions points along the path

---

to true-false transitions of $P(s)$. To compute the conjunction or disjunction of $P(s)$ with another predicate $Q(s)$, we merge their lists of transitions, marking the true-false transitions of the result as we go.

The fastest way to merge all the constraints is to proceed in binary tree fashion by first merging pairs of constraints, then taking the outputs of this "phase" and merging pairs of them etc. The time required for each phase is linear in the *total* number of transitions of all the predicates being merged in that phase. Since merging never introduces new transitions, the total number of transitions at each phase will be less than or equal to the number of transitions before the first phase. This last number is linear in N_c, since each (cubic) constraint has at most 3 transitions. Each merge phase reduces the number of predicates by a factor of two,

so there will be about $\log_2 N_c$ merge phases, each of which takes time linear in N_c. Thus the complexity of merging all constraints is $O(N_c \log N_c)$.

An example of the operation of the collision detector is given in figure (2). In this figure the L-shaped object moves and rotates uniformly between the configurations labelled A and A'. The collision detector returns a series of transition points which correspond to the object and obstacle just touching. These are illustrated from a different angle in the next frame. The algorithm was implemented on a Symbolics Lisp Machine and takes about 10 seconds for the problem shown.

6. Collision Detection for Several Moving Objects

So far we have assumed that we are given a path for a single object moving among fixed obstacles. We can readily extend the interference test to deal with several moving objects by testing pairwise for interference between objects. The configuration space for 2 objects in motion is 12-dimensional, but note that the expressions for overlap in equations (8),(9),(12),(13) are valid independent of the configuration space. When the trajectory of both objects is known, the configuration parameters are still all functions of a single argument, so once again collision detection reduces to solving univariate polynomials. The only significant difference is in the constraint equations themselves, and we now give the general form for A, B and C constraints between moving objects.

We denote one of the moving objects A and the other B, and let the configuration of A be $(\mathbf{x}_A, \mathbf{Q}_A)$, and that of B be $(\mathbf{x}_B, \mathbf{Q}_B)$ (we will assume normalized quaternions for the moment). Recall that the equation for type-A interaction with B fixed is

$$\text{Type A} \qquad (\mathbf{p}_B - \mathbf{x}_A) \cdot \text{Rot}(\mathbf{n}_A, \mathbf{Q}_A) - d_A = 0$$

But if B has configuration $(\mathbf{x}_B, \mathbf{Q}_B)$, then the actual position of $\mathbf{p}_B$ will be $\text{Rot}(\mathbf{p}_B, \mathbf{Q}_B) + \mathbf{x}_B$, and the generalized type-A constraint is

$$\text{Generalized Type A} \qquad (\text{Rot}(\mathbf{p}_B, \mathbf{Q}_B) + \mathbf{x}_B - \mathbf{x}_A) \cdot \text{Rot}(\mathbf{n}_A, \mathbf{Q}_A) - d_A = 0 \quad (14)$$

We can again use projection of quaternions to eliminate one dimension of the configuration space, and we do this independently for $\mathbf{Q}_A$ and $\mathbf{Q}_B$. We first make equation (14) independent of quaternion magnitude. There is a term in (14) which is the dot product of two rotated vectors, and the magnitude of this term will vary with the product of the squared magnitudes of the two quaternions. To make the equation independent of magnitude, we must multiply the other terms by the squared magnitudes, which are given by dot products of the quaternions with themselves. The scale independent form is

Generalized Type A

$$(\text{Rot}(\mathbf{p}_B, \mathbf{Q}_B) + (\mathbf{Q}_B \cdot \mathbf{Q}_B)(\mathbf{x}_B - \mathbf{x}_A)) \cdot \text{Rot}(\mathbf{n}_A, \mathbf{Q}_A) - (\mathbf{Q}_A \cdot \mathbf{Q}_A)(\mathbf{Q}_B \cdot \mathbf{Q}_B) d_A = 0 \quad (15)$$

If both A and B are moving, type A and B constraints are of the same form, since both arise when a vertex of one object touches a face of the other. By symmetry, the equation for generalized type B interaction can be obtained from (15) by simply interchanging A and B subscripts.

Generalized Type B

$$(\mathrm{Rot}(\mathbf{p}_A, \mathbf{Q}_A)+(\mathbf{Q}_A\cdot\mathbf{Q}_A)(\mathbf{x}_A-\mathbf{x}_B))\cdot\mathrm{Rot}(\mathbf{n}_B, \mathbf{Q}_B)-(\mathbf{Q}_A\cdot\mathbf{Q}_A)(\mathbf{Q}_B\cdot\mathbf{Q}_B)d_B = 0 \quad (16)$$

For type C constraints we take equation (3) for edge-edge interaction for a stationary object B

$$\text{Type C} \qquad \mathrm{Rot}(\mathbf{e}_A, \mathbf{Q}_A) \times \mathbf{e}_B \cdot (\mathrm{Rot}(\mathbf{p}_A, \mathbf{Q}_A) + \mathbf{x}_A - \mathbf{p}_B) = 0$$

and rotate the edge $\mathbf{e}_B$ by $\mathbf{Q}_B$, and rotate and translate the vertex $\mathbf{p}_B$. We then obtain

Generalized Type C

$$\mathrm{Rot}(\mathbf{e}_A, \mathbf{Q}_A) \times \mathrm{Rot}(\mathbf{e}_B, \mathbf{Q}_B) \cdot (\mathrm{Rot}(\mathbf{p}_A, \mathbf{Q}_A) + \mathbf{x}_A - \mathrm{Rot}(\mathbf{p}_B, \mathbf{Q}_B) - \mathbf{x}_B) = 0$$

This form is not scale independent, but by rearranging scalar triple products and using the distributivity of rotation over cross product, we obtain

Generalized Type C

$$\begin{aligned} &\mathrm{Rot}(\mathbf{p}_A \times \mathbf{e}_A, \mathbf{Q}_A) \cdot \mathrm{Rot}(\mathbf{e}_B, \mathbf{Q}_B) - \mathrm{Rot}(\mathbf{p}_B \times \mathbf{e}_B, \mathbf{Q}_B) \cdot \mathrm{Rot}(\mathbf{e}_A, \mathbf{Q}_A) \\ &+\mathrm{Rot}(\mathbf{e}_A, \mathbf{Q}_A) \times \mathrm{Rot}(\mathbf{e}_B, \mathbf{Q}_B) \cdot (\mathbf{x}_A - \mathbf{x}_B) = 0 \end{aligned} \quad (17)$$

and this equation is independent of the magnitude of either quaternion. Thus we can represent rotations using the vector parts of the quaternions only i.e. $\mathbf{q}_A$ and $\mathbf{q}_B$, and assume the scalar parts have value 1.

For motion planning, the configuration space is now 12-dimensional and configurations are of the form $(\mathbf{x}_A, \mathbf{q}_A, \mathbf{x}_B, \mathbf{q}_B)$. The configuration obstacles in this space are again defined by the union of (8),(9),(12),(13), but where the A, B and C constraints are the generalized scale-independent constraints (15),(16),(17) instead of the original constraints (4),(6),(7). The new constraints are algebraic but their total degree is now 5 instead of 3. Each constraint is linear in the position components of the configuration and quartic in the rotation components.

The collision detection scheme described in section 5 can be readily extended to deal with the generalized constraints. Suppose the two objects move uniformly from configuration $(\mathbf{x}_{Ai}, \mathbf{q}_{Ai}, \mathbf{x}_{Bi}, \mathbf{q}_{Bi})$ to configuration $(\mathbf{x}_{Af}, \mathbf{q}_{Af}, \mathbf{x}_{Bf}, \mathbf{q}_{Bf})$. Each component of the objects' configuration will be a linear function of some parameter

s, and since the total degree of the constraints is 5, each constraint will be a quintic polynomial in the parameter s. Quintics can not be solved explicitly, but numerical methods such as the modified Uspensky algorithm [3] can be used to find all the zeros of a quintic very rapidly. The zeros of the quintics correspond to true-false transitions of predicates, and these can be merged exactly as described in section 5, and the time bounds derived there are still valid.

Thus collision detection for pairs of moving objects has the same asymptotic complexity as collision detection for a moving object and a fixed obstacle. The ratio of running times for the two methods is expected to be small (less than an order of magnitude) although this has not been verified in an implementation. For several moving objects, we find ranges of values of s which lead to collisions between pairs of objects, and then find the union of these ranges for all pairs. There are $O(n^2)$ such pairs to be tested for n moving objects, giving similar complexity to the fixed obstacle scheme when the object and obstacle are built from n convex parts.

7. Conclusions

In this paper we described the three basic types of interaction between polyhedral objects and obstacles, and derived equations for each type of interaction. We showed that the use of quaternions to represent three-dimensional rotations gives rise to algebraic constraints between object and obstacles. We gave a projection of the 3-sphere of unit quaternions onto a 3-dimensional hyperplane in the 4 dimensional quaternion space as an alternative representation of rotations that reduces the complexity of the configuration space constraints, after slight modification of some of the constraints. The new form of the constraints was given for types A, B and C interactions. They were found to be linear in the position coordinates and quadratic in the rotation components.

We then derived logical expressions for the configuration space obstacles in terms of half-spaces defined by constraint inequalities. The configuration obstacle for a convex object and obstacles was shown to be a union of intersections of half-spaces, or equivalently, as a disjunction of conjunctions of predicates. The overlap predicates were divided into 4 types (there was a subdivision of type C interaction into 2 types) and although all 4 are necessary to test for overlap, certain sets of 2 can be used for path planning between free configurations.

We showed that if the path of an object can be described by polynomials in some parameter, detecting collisions of the object with obstacles requires only the solution of a univariate polynomial. We gave an example of collision detection for paths which are linear in the configuration space, and found that in this case finding contact points required the solution of univariate cubics. The intersection/union of path segments was accomplished by merging sorted transition lists, and the total time for the algorithm was shown to be almost linear in the number of constraints.

The moving object / fixed obstacle constraints were then generalized for two moving objects. This lead to constraints that were polynomials of total degree 5 in

the configuration parameters. We outlined a collision detection scheme for a pair of uniformly moving objects and showed that it had the same asymptotic complexity as the fixed obstacle scheme, the main difference being that it required the solution of quintic rather than cubic polynomials.

Natural extensions of the work would deal with more general polynomial paths, or with different sets of algebraic constraints, such as those for a 6 degree of freedom manipulator. The ultimate objective of this work is to apply techniques from computer algebra to design a triangulation procedure for algebraic constraints, which would enable us to find collision-free paths through the configuration space.

References

[1] Boyse J.W., "Interference Detection Among Solids and Surfaces", Comm ACM, vol 22, No 1 (1979) pp 3-9.

[2] Brooks R.A. and Lozano-Pérez T., "A Subdivision Algorithm in Configuration Space for Findpath with Rotation", IJCAI 83, pp 799-806.

[3] Collins G.E. and Loos R., "Real Zeros of Polynomials", Computing Supplementum 4, (1982) pp 83-94.

[4] Donald B.R., "Motion Planning with Six Degrees of Freedom", MIT AI TR-791 (1984).

[5] Hamilton W.R., "Elements of Quaternions", 3rd edition, Chelsea Publishing Co., New York, 1969.

[6] Lozano-Pérez T., "Spatial Planning: A Configuration Space Approach", IEEE Trans. Computers, C-32, No 2 (Feb 1983) pp 108-120.

[7] Schwartz J. and Sharir M., "On the 'Piano Movers' Problem, II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds", Computer Science Department, New York University report 41 (1982).

Appendix I

Equation (5) is the form most commonly used for computing the composition of a vector and a quaternion. As given it requires a total of 22 multiplications and 14 additions, plus 2 shift operations for scalar multiplication by 2. If the quaternion is assumed to have unit magnitude, i.e. $q_0^2 + \mathbf{q} \cdot \mathbf{q} = 1$, (5) becomes

$$\mathrm{Rot}(\mathbf{y}, \mathbf{Q}) = \mathbf{y}(2q_0^2 - 1) + 2(\mathbf{y} \cdot \mathbf{q})\mathbf{q} + 2q_0\mathbf{q} \times \mathbf{y} \qquad (i)$$

which requires 19 multiplications and 12 additions by saving on a scalar product. We can obtain a greater saving by noting that the two scalar products in (5) form the expansion of a vector triple product. When these are combined we obtain

$$\mathrm{Rot}(\mathbf{y}, \mathbf{Q}) = \mathbf{y}(q_0^2 + \mathbf{q} \cdot \mathbf{q}) + 2\mathbf{q} \times (\mathbf{q} \times \mathbf{y}) + 2q_0\mathbf{q} \times \mathbf{y} \qquad (ii)$$

which at first sight does not seem to be much of a simplification. However, the vector product $\mathbf{q} \times \mathbf{y}$ occurs twice in (ii) and need only be computed once. This form requires 19 multiplications and 15 additions, and works for arbitrary quaternions. If we assume that the quaternion $\mathbf{Q}$ has unit magnitude we obtain finally

$$\mathrm{Rot}(\mathbf{y}, \mathbf{Q}) = \mathbf{y} + 2\mathbf{q} \times (\mathbf{q} \times \mathbf{y}) + 2q_0\mathbf{q} \times \mathbf{y} \qquad (iii)$$

and this form requires only 15 multiplications and 12 additions, a saving of 4 multiplications over (i).

In fact if multiplications are a serious expense, we can reduce their number still further. We have assumed in the above analysis that vector cross product requires 6 multiplications and 3 additions. It can actually be done with 5 multiplications and some extra additions. Suppose $\mathbf{u} = \mathbf{v} \times \mathbf{w}$, and denote the x-component of $\mathbf{u}$ as u_x etc. We compute the following quantities

$$(v_y + v_z)(w_z - w_y) = \alpha \qquad (v_y - v_z)(w_z + w_y) = \beta$$

$$(v_z + v_x)(w_x - w_z) = \gamma \qquad (v_z - v_x)(w_x + w_z) = \delta$$

$$(v_x - v_y)(w_y + w_x) = \epsilon$$

and the vector $\mathbf{u}$ is given by

$$u_x = \frac{\alpha + \beta}{2} \qquad u_y = \frac{\gamma + \delta}{2} \qquad u_z = \epsilon + \frac{-\alpha + \beta - \gamma + \delta}{2}$$

This form of cross product requires 5 multiplications but 16 additions, compared with 6 multiplies and 3 additions for the conventional method. It is only likely to be useful when multiplication is very expensive compared to addition.