Heuristic Motion Planning with Many Degrees of Freedom

Thomas Chadzelek Jens Eckstein Elmar Schömer *

22nd April 1996

Abstract

We present a heuristic approach to the geometric motion planning problem with the aim to quickly solve intuitively simple problems. It is based on a divide-and-conquer path search strategy which makes inquiries about feasible paths; to answer these, we develop an efficient collision detection scheme that handles translations and rotations of polyhedra to compute all times of collision. The whole algorithm can be easily implemented and has been successfully tested in a program for assembly planning.

1 Introduction

This work is based on a simple but general **path search** strategy for spaces of arbitrary dimension. It is a heuristic algorithm for the generalized movers' problem that does *not explicitly* compute or represent configuration space but rather utilizes a **collision detection** subroutine for inquiries about possible paths. The approach uses divide-and-conquer and can be applied to many concrete situations, e.g. to motion planning for a single rigid body moving freely, a jointed robot arm, or even several objects moving concurrently. The fundamental idea is described in [Sch92] and will be investigated and refined here; we show how to handle translational and rotational degrees of freedom separately to simplify implementation and speed up execution. Quaternions make it possible to describe purely rotational motion planning as a path search problem in a spherical geometry.

Special care has been taken to make the collision detection scheme as efficient as possible, especially with respect to practical applications, since it determines the total running time of our algorithm. It is shown how to compute all intervals of intersections of a polyhedron P moving by a given rotation or translation amidst polyhedral obstacles Q in time $O(n \log n)$, where $n := |P||Q|$. The notation $|P|$ means the size of a description of P's boundary. We deal directly with non-convex polyhedra and use enveloping techniques to reduce the average running time dramatically; these approaches go back to [Ca87] and [Sch94].

Former approaches to motion planning have either led to general procedures which could not be handled in practice like the famous one described by Schwartz and Sharir. Or they confined themselves to solving arbitrary instances of simple motion planning problems efficiently, e.g. moving a disc between polygons in the plane. In contrast to that we shall try here to solve simple instances of arbitrary motion planning problems efficiently, i.e. to find a practical algorithm for use with practical problems. This was especially motivated by an ongoing research project on computer aided manufacturing where we investigate the interactive simulation and planning of assembly processes including robots. Fast on-line collision detection schemes were developed in that context and influenced this work. Furthermore, motion planning was needed in order to simplify the specification of assembly plans, i.e. the engineer should be allowed to describe *what* she wants done rather than *how* to do it.

2 A Path Search Strategy

The *generalized movers' problem* consists of the description of an object to be moved together with its initial and final position and an obstacle. It is the task of *motion planning* then to find a collision-free path for the object or to decide that such a path does not exist. In the case we study here there are no restrictions with respect to the dynamics of the motion, solely geometric constraints are to be observed.

One cause for the long running time of "conventional" methods lies in the fact that precise and complete information about the clearance of the

*Lehrstuhl Prof. G. Hotz, FB 14 Informatik, Universität des Saarlandes, Saarbrücken, Germany. Please contact our WWW-server `http://hamster.cs.uni-sb.de` for more information, including technical reports.

moving object within its environment is computed. Typically, a point in a high-dimensional space is used to represent a current configuration of the scene; its coordinates reflect the degrees of freedom of all moving parts, e.g. as distances or angles. This is known as the *configuration space* approach and was first presented by Lozano-Peréz. The set of all points representing configurations where collision occurs is called *configuration obstacle*, its complement is named *free-space*. Explicit computation and representation of that information for complex scenes requires enormous time and large storage capacity.

We shall not do this but nonetheless use configuration space as an abstract concept allowing a unified description of motion planning problems. Only the intersection of a line with the configuration obstacle is computed; this can be done efficiently even in high dimensions based only on the description of the original object and obstacle. To demonstrate the use of such minimal information, we shall first describe the path search strategy in the plane, which develops from a simple nondeterministic version. We then extend it to spaces of higher dimension and show how to handle degrees of freedom separately—this will be very important for the collision detection presented in the next section.

2.1 Nondeterministic Description

Consider the following problem in two dimensions: Let $O \subset \mathbb{R}^2$ be a subset of the plane, called obstacle, together with a start and goal $\mathbf{a}, \mathbf{z} \in \mathbb{R}^2$. Find a way for a point moving from $\mathbf{a}$ to $\mathbf{z}$ avoiding the obstacle O.

Here the plane represents the configuration space of some motion planning problem; thus we assume that O is not directly accessible to the planning strategy but only by inquiries about the intersection of a line with O. Therefore a path considered by our strategy is piecewise linear and may be described by the tuple $(\mathbf{v}_0, \ldots, \mathbf{v}_n)$ of its vertices $\mathbf{v}_i \in \mathbb{R}^2$. Because of the known complexity of many motion planning problems we do not attempt to decide whether such a path exists but merely try to find one; it is in this sense that our heuristic algorithm remains incomplete.

To construct a path, we first check the straight connection $(\mathbf{a}, \mathbf{z})$ of start and goal for intersections with the obstacle; if none are found, we are done. Else a borderline b separating $\mathbf{a}$ from $\mathbf{z}$ is considered, e.g. the perpendicular bisector of the two points. This is illustrated in figure 1; $\mathbf{p}$ denotes the borderline's base point.

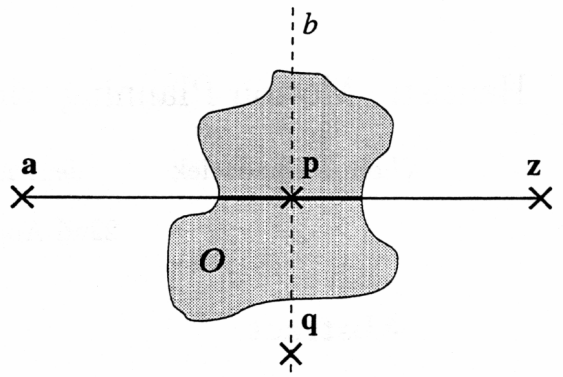

Figure 1: A simple motion planning problem

Idea: *Every path from* $\mathbf{a}$ *to* $\mathbf{z}$*, especially every collision-free one, must cross this borderline separating* $\mathbf{a}$ *from* $\mathbf{z}$*. In order to avoid the obstacle* O *such a crossing can only take place on the free sections of* b*.*

This means that we guess a *via point* $\mathbf{q}$ on the line b outside of O and try to recursively connect $\mathbf{a}$ to $\mathbf{q}$ and $\mathbf{q}$ to $\mathbf{z}$, combining the two paths to one solution $(\mathbf{a}, \ldots, \mathbf{q}, \ldots, \mathbf{z})$—a *divide-and-conquer* approach. It is clear that this method generally does not find an optimal path with regard to any criterion, e.g. path length or safety distance.

The nondeterministic procedure shown in figure 2 constructs a piecewise linear path for a point moving from $\mathbf{a}$ to $\mathbf{z}$ avoiding the obstacle $O \subset \mathbb{R}^2$; it succeeds if such a solution exists. Here the path is represented as a list of points, which initially contains only the starting point $\mathbf{a}$, this is denoted by $[\mathbf{a}]$.

given: $O \subset \mathbb{R}^2,\ \mathbf{a}, \mathbf{z} \in \mathbb{R}^2$
wanted: $path : \mathrm{List}(\mathbb{R}^2)$
invocation: $path \leftarrow \mathrm{connect}(\mathbf{a}, \mathbf{z}, [\mathbf{a}])$

```
connect(a, z : ℝ²; path : List(ℝ²)) : List(ℝ²)
    p, q, r : ℝ²;   t : ℝ;   L : IntervalList
L ← intersectSegment(a, z)              collision intervals
if empty(L) then
    path ← path ∘ [z]                          append point
else p ← (a + z)/2                               base point
    r ← (z − a)^⊥                                orthogonal
    L ← intersectLine(p, r)                check borderline
    randomly choose t ∉ L
    q ← p + t r                                    via point
    path ← connect(a, q, path)  recursive solution ...
    path ← connect(q, z, path)  ... of subproblems
return path
```

Figure 2: Nondeterministic Path Search Strategy

Two functions are used to compute the intersec-

tion of a segment and a line resp. with the obstacle O. As a result of such an inquiry, a list of intervals is expected with intersectLine$(\mathbf{p}, \mathbf{r}) = \{t \in \mathbb{R} \mid \mathbf{p} + t\,\mathbf{r} \in O\}$ etc. These intervals are called *collision intervals* in contrast to their complement, the *free intervals*; the corresponding sections on the line are named accordingly.

2.2 Deterministic Description

Making the algorithm deterministic implies searching through the possible computations of the nondeterministic machine for an accepting one, but we cannot search the continuum. So we shall restrict ourselves to a small number of promising via points on each borderline, which are then tried systematically by *backtracking*.

Recursion Depth We must diagnose failure, i.e. decide that a certain computation should no longer be tracked because it will not succeed. This is generally impossible and can only be approximated, for instance by a limit k on the depth of recursion involved in the path search scheme. If recursion exceeds that limit, we consider our last choice unsuitable and undo it. Thus we should choose k small enough to quickly skip futile computations but big enough to allow scope for a solution to exist. Three approaches have been implemented to find a useful value for k, among them:

- Try $k = 0, 1, 2, \ldots$ This is simple to implement and never overestimates k; it works fast enough because running time grows approximately exponential in k and is therefore dominated by the largest value used.
- Try $k = 0, 1, 2, \ldots$ in a breadth-first manner, keeping a tree of already known information about subproblems; this needs a considerable amount of memory. Different subdivisions of a path planning problem can be tracked simultaneously, thus we can rate the partial solutions according to some criterion, e.g. path length or recursion depth, and try to improve the quality of the constructed path.

Via Points A smart choice of them is quite critical to any practical implementation. Statistical results of pseudo-random experiments with a point moving between discs in the plane as well as theoretical considerations lead to the following concept. Choose the borderline b as the perpendicular bisector of the collision section whose centre is closest to that of $(\mathbf{a}, \mathbf{z})$. This is locally symmetrical to a known part of the obstacle and divides the path planning problem quite fairly into two subproblems.

Via points are chosen as centres of free sections of the borderline, this is again locally symmetrical and maximizes safety distance as well; experiments support this decision. Here we assume that the whole scene is bounded in some way, either naturally by surrounding obstacles or artificially by a restriction of configuration space.

Via points are then rated according to a synthesis of two criteria: local safety distance and closeness to the intended path. Therefore let d denote the distance of the via point $\mathbf{v}$ to the line segment $(\mathbf{a}, \mathbf{z})$ and l the length of the free section containing $\mathbf{v}$. We use the valuation function $V_\beta(d, l) := d/l^\beta$ and can vary $\beta \geq 0$ for diverse results, e.g. with $\beta = 0$ the rating depends only on the deviation of $\mathbf{v}$ from the original path. To emphasize safety distance, $\beta > 1$ can be used. In order of increasing V_β the various via points on a borderline are arranged; that list is restricted to a small number of points.

2.3 Generalization to $\mathbb{R}^n$

The concept of the path search strategy described above can be easily generalized to dimensions higher than two with one slight difference. The notion of a borderline must be extended to a hyper-plane separating $\mathbf{a}$ from $\mathbf{z}$, and it is no longer possible to capture all relevant information about it with a single query to "intersectLine". Thus we choose a certain number of lines within that hyper-plane hoping to find useful via points. It is our experience that the directions of these lines should form an orthogonal vector base of the configuration space, with $\mathbf{z} - \mathbf{a}$ as one axis. Note that we can no longer guarantee that a suitable crossing point exists on any fixed line.

2.4 Separate Degrees of Freedom

Consider the case where a moving object has distinct kinds of degrees of freedom, e.g. translational and rotational ones, or think of a jointed robot arm with many links. To simplify the computation of the intersection of a line with the configuration obstacle remarkably, we handle these degrees of freedom separately. This means that the configuration space C is divided accordingly, i.e. d degrees of freedom fall into n groups and we have $C = C_0 \times \cdots \times C_{n-1} \subseteq \mathbb{R}^d$. We write $(a_0, \ldots, a_{n-1}) \in C$ where the a_i may be vectors themselves and speak of "coordinates" in this generalized sense. A feasible path for a moving point must then consist of pieces changing only one coordinate each.

What does "straight connection" mean then? The algorithm given here changes coordinates in a simple cyclic order for reasons of efficiency; this allows to change the "dodging" coordinate before the colliding one when trying to reach a via point—which is necessary. The notion of dodging in a direction perpendicular to the original path is easily conserved; the motion must use another coordinate or be perpendicular within the same one in the usual sense.

Given a separation of configuration space as shown above, and start and goal points $\mathbf{a} = (a_0, \ldots, a_{n-1}) \in C$, $\mathbf{z} = (z_0, \ldots, z_{n-1}) \in C$, the planning strategy is called with $path \leftarrow \text{connect}(\mathbf{a}, \mathbf{z}, 0, [\mathbf{a}])$ and tries to compute a solution. We only show the non-deterministic version for reasons of clarity, determinism is achieved as said before and merely adds a little book-keeping. The parameter j controls the cyclic order of changing coordinates, the degrees of freedom C_j are used first; "viaPoint" yields a random via point and the index of its dodging coordinate.

```
connect(a, z : C; j : ℕ; path : List(C) ) : List(C)
    k : ℕ;   pos, q : C;   L : IntervalList
pos ← a                              current position (steps to z)
repeat  pos_j ← z_j
        L ← intersectSegment(a, pos)
        if empty(L) then  path ← path ∘ [pos]
                          a ← pos
                          j ← j + 1  (mod n)
until ¬ empty(L) ∨ a = z
if ¬ empty(L) then                   collision in coord. j
    (q, k) ← viaPoint(a, pos, L)     q dodges in coord. k
    path ← connect(a, q, k, path)
    path ← connect(q, z, j, path)
return path
```

Figure 3: Planning Strategy for Separate Degrees of Freedom

3 Efficient Collision Detection

The path search strategy issues inquiries about the intersection of a line with the configuration obstacle. These are answered by an efficient collision detection algorithm based on a description of all objects in *work space*; we restrict ourselves to polyhedra. A line in configuration space describes a coordinated motion of all moving objects with no beginning or end—a segment means a bounded motion. Intersection with the configuration obstacle corresponds to collision among the polyhedra in work space.

We separate the translational and rotational degrees of freedom by imposing the restriction, that at a given time either one object may rotate about a fixed axis with constant angular velocity or all may translate simultaneously, each with its own fixed speed and direction. This greatly reduces running time in practical applications, because enveloping techniques can be used. The section on rotations shows a suitable modification of the path search strategy for such motions.

Consider all pairs of one moving object and one fixed obstacle, exploiting the relative nature of translational motions. A predicate will be derived to test the static overlap of these polyhedra at their current position, it is described by a boolean expression. By evaluating that expression skillfully, all intervals of a given motion, where overlap occurs, can be computed quickly; this concept is due to [Ca87]. A *boundary representation* is used which lists all vertices, edges, and faces of a polyhedron as well as their adjacency relations ($|P| = v + e + f$); convexity is required for all faces.

3.1 Static Detection of Overlap

The Predicate Two polyhedra overlap iff their surfaces intersect or one is entirely contained in the other. We take pattern from Canny's boolean predicate for the first part, the second will be handled differently in our algorithm. Intersection of surfaces means that an edge of one body pierces a face of the other including degenerate cases, where the boundaries of these features interact.

Let $l := l_{\mathbf{p},\mathbf{q}}$ be the line segment or edge with vertices $\mathbf{p}$ and $\mathbf{q}$; f the face with vertices $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}$ contained in the plane $P_{\mathbf{n},n_0} : \mathbf{n}^{\mathrm{T}}\mathbf{x} = n_0$. The vertices $\mathbf{v}_i$ are ordered counter-clockwise if f is viewed contrary to $\mathbf{n}$. We now assume that $\mathbf{n}^{\mathrm{T}}(\mathbf{q}-\mathbf{p}) > 0$; if it is negative, we just exchange $\mathbf{p}$ and $\mathbf{q}$; if it is zero, the edge and face are parallel which can be safely ignored (cf. [Ch95]). Thus the result for this case is:

$$l \cap f \neq \emptyset \iff \mathbf{n}^{\mathrm{T}}\mathbf{q} \geq n_0 \ \wedge \ \mathbf{n}^{\mathrm{T}}\mathbf{p} \leq n_0$$
$$\wedge \, \forall\, 0 \leq i < k : \ \det[\mathbf{q}-\mathbf{p}, \mathbf{v}_{i+1}-\mathbf{v}_i, \mathbf{p}-\mathbf{v}_i] \geq 0$$

This yields a predicate $S(l, f)$ for the intersection of an edge and a face; its size is $\Theta(k)$. If two polyhedra P and Q are given, we construct the disjunction of $S(l, f)$ over all pairs of edges and faces; this yields the complete predicate $S(P, Q)$ for the intersection of surfaces, its size is $\Theta(|P||Q|)$.

Containment The test we use yields *true* for a superset of all configurations where $P \subset Q$, but

only for such with $P \cap Q \neq \emptyset$. Our randomized algorithm is much easier to implement than a static boolean predicate for this task. We check whether some random point $\mathbf{v}$ on the surface of P lies within Q or not; this is easily decided by looking at a ray from $\mathbf{v}$ into a fixed direction. If the ray intersects any edges or vertices of Q, we choose another point, else we count the number of intersections with faces. An odd number means that $\mathbf{v} \in Q$; of course this test must also be applied symmetrically for $Q \subset P$.

3.2 Translational Case

The moving object carries out a translation from its current position into a direction $\mathbf{r} \in \mathbb{R}^3$. We write such a general translation as a mapping $T^\lambda_\mathbf{r}(\mathbf{x}) := \mathbf{x}^\lambda := \mathbf{x} + \lambda\mathbf{r}$, where $\lambda \in \mathbb{R}$.

The Predicate We assume that the edge $l_{\mathbf{p},\mathbf{q}}$ is moving while the face remains stationary. For fixed $\mathbf{r}$ we now have to determine all relevant times of collision, i.e. all $\lambda \in \mathbb{R}$ ($[0, \lambda_{Max}]$ for "intersectSegment") with $S(T^\lambda_\mathbf{r}(l), f) = \mathit{true}$. With the above notation, $T^\lambda_\mathbf{r}(l_{\mathbf{p},\mathbf{q}}) = l_{\mathbf{p}^\lambda,\mathbf{q}^\lambda}$. The terms $\mathbf{n}^\mathrm{T}\mathbf{q} \geq n_0$ and $\det[\mathbf{q}-\mathbf{p}, \mathbf{v}_{i+1}-\mathbf{v}_i, \mathbf{p}-\mathbf{v}_i] \geq 0$ lead to simple linear inequalities in λ, whose solutions can be described by at most one interval, including $\mathbb{R}$ or $\emptyset$.

The main idea now is to evaluate S using lists of intervals as intermediate results. The boolean operations $\wedge$ and $\vee$ are substituted by intersection and union on interval lists which can be easily implemented to run in linear time.

A tree for $S(P,Q)$ has $\Theta(n)$ leafs because of the predicate's size, $n := |P||Q|$; its depth can be restricted to $O(\log n)$. That tree is pruned during evaluation if intermediate results are $\mathbb{R}$ and $\emptyset$ for union and intersection resp. The atomic inequalities can be evaluated in constant time, there are at most n intervals to handle at each level of the tree—note that neither intersection nor union can increase the number of intervals. All in all time $O(n \log n)$ suffices to determine all sections of the motion that correspond to collisions.

Containment We assume P is moving and Q remains fixed and take $\mathbf{r}$ as direction of the ray. We choose $\mathbf{v}$ within a triangle on the surface of P whose projection onto a plane orthogonal to $\mathbf{r}$ does not degenerate to a line; not every point of such a triangle can collide with an edge or vertex of Q. This part is irrelevant for the total running time of the collision detection scheme.

Enveloping Techniques A hierarchic approach is used which features ever more detailed bounding bodies around the actual polyhedra and their faces. At the highest level, approximations of smallest enclosing spheres are computed once for every object, e.g. at startup time, and associated with it; these can easily be handled even during rotations. A simple quadratic inequality can tell whether the moving sphere intersects the other at some time during the motion, in which case we proceed to the next level.

Bounding boxes are then computed in a special orthogonal coordinate system with $\mathbf{r}$ as z-axis, thus x- and y-coordinates now remain fixed under translation. We use axially parallel rectangloids; these can be computed easily and may be looked upon as the Cartesian product of x-, y-, and z-intervals; they are stored with the polyhedron for possible re-use. Collision-freeness is ascertained if the x-y-rectangles do not overlap or if the z-intervals remain disjunct even if the translation is considered. If these boxes interfere with each other, we eventually have to look at all edge-face-pairs but use bounding rectangloids for individual faces first.

The test for containment can also be improved in such a way; because of the rectangloids' alignment one object may only lie totally within another if the same holds for their bounding boxes.

3.3 Rotational Case

The moving object now rotates about a fixed axis, either through a given angle or through one complete turn. It is well known that a rotation $R^\varphi_\mathbf{r}$ can be described by an axis through the origin with direction $\mathbf{r} \in \mathbb{R}^3$, $|\mathbf{r}| = 1$ and an angle $\varphi \in \mathbb{R}$. It is applied to a vector $\mathbf{x} \in \mathbb{R}^3$ by the function

$$\mathbf{x}^\varphi := (1-\cos\varphi)\mathbf{r}^\mathrm{T}\mathbf{x}\mathbf{r} + \cos\varphi\,\mathbf{x} + \sin\varphi\,\mathbf{r}\times\mathbf{x} \quad (1)$$

Another way to deal with rotations is to use *quaternions*, and as this is quite elegant we give a short introduction here and show the applications to our problem.

3.3.1 Motion Planning for Rotations

Quaternions Quaternions $\mathbf{Q} = (q_0, \mathbf{q}) \in \mathbb{R}^4$ will now be written with a scalar part $q_0 \in \mathbb{R}$ and a vector part $\mathbf{q} \in \mathbb{R}^3$, naturally including $\mathbb{R}$ and $\mathbb{R}^3$ into the set of quaternions.

$R^\varphi_\mathbf{r}$ is given by the quaternion $\mathbf{Q}^\varphi_\mathbf{r} := \left(\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}\,\mathbf{r}\right)$ via $(0, x^\varphi) = \mathbf{Q}^\varphi_\mathbf{r} \cdot (0,\mathbf{x}) \cdot (\mathbf{Q}^\varphi_\mathbf{r})^*$. All $\mathbf{Q}^\varphi_\mathbf{r}$ are unit quaternions and form a group under quaternion product which corresponds to composition of rotations.

Spherical Geometry The set of unit quaternions can be looked upon as the *unit sphere* $\mathcal{S}_4$ in four-dimensional Euclidean space, where $\mathcal{S}_n := \{\mathbf{x} \in \mathbb{R}^n \mid |\mathbf{x}| = 1\}$. We again read $1 = (1, \mathbf{0}) \in \mathcal{S}_4$ and $\mathbf{r} = (0, \mathbf{r}) \in \mathcal{S}_4$ for convenience and identify points with their position vectors. We describe the orientation of a polyhedron by a rotation with reference to some fixed initial position which corresponds to the unity 1 of the quaternion algebra.

The orientation of a rotating object varies continuously and the corresponding quaternion traces an arc of a great circle on the unit sphere. Such a *great circle* of $\mathcal{S}_n$, i.e. a circle around the origin with radius 1, is uniquely determined by the plane in which it is contained; the latter can be given by two linearly independent vectors or two distinct points on the sphere. In the special case of two orthogonal axes $\mathbf{P}, \mathbf{Q} \in \mathcal{S}_n$ ($\mathbf{P} \perp \mathbf{Q}$ with respect to the standard scalar product of $\mathbb{R}^n$) we have $C_{\mathbf{P},\mathbf{Q}} := \{\cos\varphi\, \mathbf{P} + \sin\varphi\, \mathbf{Q} \mid \varphi \in \mathbb{R}\}$.

During a full rotation the orientation of the object describes half a great circle on $\mathcal{S}_4$, namely $C_{1,\mathbf{r}}$ for a fixed axis $\mathbf{r} \in \mathcal{S}_3$. To achieve other axes, let an initial orientation $\mathbf{P} \in \mathcal{S}_4$ be given; this yields the great circle $C_{\mathbf{P},\mathbf{r}\cdot\mathbf{P}}$ (note that $\mathbf{P}^{\mathrm{T}}(\mathbf{r} \cdot \mathbf{P}) = 0$). For two given orthogonal axes $\mathbf{P}, \mathbf{Q} \in \mathcal{S}_4$ of a great circle we can find an axis of rotation $\mathbf{r} \in \mathcal{S}_3$ with $\mathbf{Q} = \mathbf{r} \cdot \mathbf{P}$. Thus complete rotations map to great circles, and vice versa.

In this way quaternions give a neat description of rotations which is also graphical. They can be used to describe the path search strategy for purely rotational motions in a spherical rather than Euclidean geometry. In analogy to the approach described for the plane, we need only two major concepts. The *direct connection* of two positions is given by the unique great circle containing them. The notion of *perpendicular dodging* also has a simple meaning based on the following lemma: *Great circles on $\mathcal{S}_4$ are perpendicular iff the corresponding axes of rotation are.*

Path Search Strategy Quaternions and the spherical geometry are applied to motion planning by a modification of our path search strategy, which is thought of as taking place on the unit sphere $\mathcal{S}_4$. Initial and final configurations $\mathbf{a}$ and $\mathbf{z}$ are mapped to points via the correspondence with quaternions mentioned above. Then the straight motion from $\mathbf{a}$ to $\mathbf{z}$, i.e. a rotation about the axis associated with the great circle $C_{\mathbf{a},\mathbf{z}}$ through these two points, is checked for its free and unfree segments. To dodge collision situations, we look at great circles perpendicular to $C_{\mathbf{a},\mathbf{z}}$, in four dimensions there is one degree of freedom for this choice. We apply techniques similar to those mentioned for $\mathbb{R}^n$ to select a number of circles and to find suitable via points on them; this involves checking full rotations about axes perpendicular to that of the original "straight" motion. These via points are then aimed at directly and lead to subdivisions of the problem etc.

3.3.2 Dynamic Collision Detection

We want to find out all points of time of a given rotation of one object at which collision with a certain obstacle takes place and describe these as lists of intervals. This is done just like in the translational case, with the exception that rotations lead to quadratic inequalities deriving from $\alpha\cos\varphi + \beta\sin\varphi + \gamma \geq 0$ (cf. equation 1) instead of linear ones. The test for containment and our enveloping strategy are also influenced slightly.

Containment We randomly pick $\mathbf{v}$ from a triangle on P's surface that is not perpendicular to the axis of rotation $\mathbf{r}$. The ray is replaced by the circle $\mathbf{v}$ describes during rotation; if it does not intersect Q, we send a ray from the point's original position to find out whether $\mathbf{v} \in Q$ initially holds or not.

Enveloping Techniques Bounding boxes are computed in a cylindrical coordinate system whose axes represent angle, height, and distance with respect to the fixed rotational axis. Here the angle corresponds in its meaning to the z-coordinate in the translational case, which was aligned to the direction of motion.

References

[Ca87] John F. Canny, *The Complexity Of Robot Motion Planning* , MIT-Press, (1987)

[Ch95] Thomas Chadzelek, *Heuristic Motion Planning with Many Degrees of Freedom*, Technical Report A 08/95, FB 14 Informatik, Univ. des Saarlandes, Saarbrücken, (1995)

[Sch92] Achim Schweikard, *A Simple Path Search Strategy Based on Calculation of Free Sections of Motions*, Engng. Applic. Artif. Intell. Vol. 5, No. 1, (1992), pp. 1–10

[Sch94] Elmar Schömer, *Interaktive Montageplanung mit Kollisionserkennung*, Dissertation im FB 14, Univ. des Saarlandes, (1994)