Heuristic Motion Planning with Many Degrees of Freedom

Thomas Chadzelek* Günter Hotz Elmar Schömer

Technical Report A 08/95 August 1995

*Master's Thesis

e-mail: {chadzele,hotz,schoemer}@cs.uni-sb.de
WWW: http://hamster.cs.uni-sb.de

Fachbereich Informatik
Universität des Saarlandes
66123 Saarbrücken
Germany

Abstract

We present a general 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 universally applied and has been successfully tested in a program for assembly planning.

Contents

List of Figures

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. We also discuss similarities with and links to work by Overmars and Švestka, [OS94], in section 1.6.

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 path search 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 arbitrary polyhedra and use enveloping techniques to reduce the average running time dramatically; these approaches go back to [Ca87] and [Sch94]. Quaternions make it possible to describe purely rotational motion planning as a path search problem in a spherical geometry.

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 in [SS82]. Or they confined themselves to solving arbitrary instances of simple motion planning problems efficiently, e.g. for the case of a disc moving 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.

Chapter 1

A Simple Path Search Strategy

1.1 Background

Motion planning is often regarded as part of robotics because it represents the prerequisite for autonomous action of a robot within its environment. From this need it is that the field has developed, with an increasing attention by theoretical computer scientists to abstract problems and especially to their complexity. In this tradition our work is less oriented to practical problems of concrete autonomous robots but very generally to a versatile concept.

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, namely the form of the object and the obstacle.

For the solution of this problem there exists an equally general approach of Schwartz and Sharir [SS82] based on Collins' method for cylindrical algebraic decomposition [Co75]. The running time reached is polynomial in the complexity of the scene, but double exponential in the number of degrees of freedom. In addition to that, the constants involved are rather large, so this procedure is of no practical importance. Canny succeeded in developing a single-exponential algorithm [Ca87], but this is still unacceptable for large input sizes.

Many successful trials have been made to give efficient algorithms for special cases or to explore the inherent complexity of motion planning problems. Because of the known lower bounds, efficient algorithms for general problems cannot be expected. To nevertheless solve at least intuitively simple problems easily and quickly even in complex scenes with many degrees of freedom new ways have to be taken.

One cause for the long running time of "conventional" methods lies in the fact that precise and complete information about the clearance of the moving object within its environment must be computed. For translational motions of a single rigid body, the object is typically shrunk to a point whereas the obstacle is grown accordingly; the difficulty consists in finding the shape of the new obstacle and of the free space for the point. More generally, the positions of all moving parts are described by certain values corresponding to their degrees of freedom, e.g. distances or angles. One point in a possibly high-dimensional space whose coordinates reflect these values can then be used to represent a current configuration of the scene. This is known as the *configuration space* approach and was first presented by [Lo83]. The subset of all points representing forbidden configurations, i.e. those where collision occurs, is called *configuration obstacle*; its complement is named *free-space*. Representation of that information for complex scenes also requires large storage capacity.

We shall not build up such information explicitly, but rather try to find a way for the moving object using only minimal information about the environment. Precisely spoken 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 explain the application thereof, we shall first describe the path search strategy in the plane, both nondeterministic and deterministic versions. The former is mainly used to clarify some ideas. We then extend it to spaces of higher dimension and show how to handle degrees of freedom separately—this will be very important for collision detection.

1.2 Nondeterministic Description

Consider the following simplified 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; this can be described by a continuous function $w : [0,1] \to \mathbb{R}^2$ with $w(0) = \mathbf{a}$, $w(1) = \mathbf{z}$, and

$$\forall\, 0 \le t \le 1 : \; w(t) \notin O$$

For the reasons given above we do not attempt to decide whether such a path exists but merely try to find one. Thus the strategy is allowed to fail or run infinitely although there is a collision-free path from $\mathbf{a}$ to $\mathbf{z}$. We solely require constructed paths to be legal solutions to the problems, i.e. especially to avoid the obstacle. It is in this sense that our heuristic algorithm remains incomplete. By means of this simplification and without the enormous burden of complete algorithms it should often be possible to easily and quickly find solutions.

Furthermore it is very useful to only compute intersections of a line with the obstacle O; this will become clear in the next chapter and is related to the fact that O will in general be represented only implicitly. Therefore paths considered by our strategy are piecewise linear and may be described by a tuple $(\mathbf{v}_0, \ldots, \mathbf{v}_n)$ of points $\mathbf{v}_i \in \mathbb{R}^2$. With $n \in \mathbb{N}$, the path results in

$$v : [0,1] \to \mathbb{R}^2, \; t \mapsto \begin{cases} \mathbf{v}_i & nt = i \\ \mathbf{v}_i + (nt - i)(\mathbf{v}_{i+1} - \mathbf{v}_i) & i < nt < i+1 \end{cases}$$

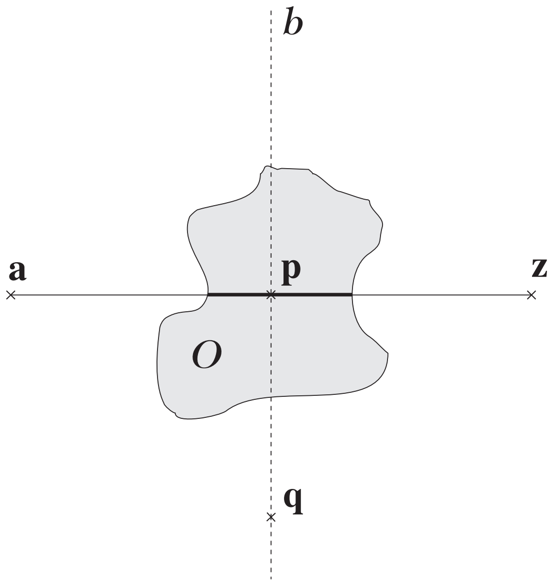


Figure 1.1: A simple motion planning problem

How can such a path be constructed then? Consider the simple example in figure 1.1. The attempt to use $(\mathbf{a}, \mathbf{z})$ as a path fails because the obstacle O blocks the way. The intersection with O is marked on the line segment, a subroutine provides this information as an answer to the strategy's inquiry. For the moment we assume the existence of such a subroutine and will later show how to implement it efficiently. Information about the obstacle is accessible to the planning strategy only through such inquiries which provide considerable abstraction from the representation of O.

The figure also shows a dashed line b which intersects the segment $(\mathbf{a}, \mathbf{z})$ in the point $\mathbf{p}$. This line gives rise to the central idea of this path search strategy:

> *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 crossing point $\mathbf{q}$ on the line b outside of O and try to recursively connect $\mathbf{a}$ to $\mathbf{q}$ and $\mathbf{q}$ to $\mathbf{z}$. As a result, two paths $(\mathbf{a}, \ldots, \mathbf{q})$ and $(\mathbf{q}, \ldots, \mathbf{z})$ should be found and can be combined to the solution $(\mathbf{a}, \ldots, \mathbf{q}, \ldots, \mathbf{z})$. Such an approach is widely known as *divide-and-conquer*.

It is clear that such a method generally does not find an optimal path with regard to any optimization criterion, e.g. path length or safety distance. We are left with two design considerations: how to choose the borderline b and the crossing point q, also known as *via point*. From a theoretical point of view and using a nondeterministic version of the above procedure, a random choice will do—as long as a piecewise linear solution exists, it might be found. We can even construct b deterministically and choose only $\mathbf{q}$ randomly; thus for definiteness let b be the perpendicular bisector of the first interval of $(\mathbf{a}, \mathbf{z}) \cap O$. Experiments with practical implementations suggest that bisectors are a good choice, but it is even better to erect them on an interval close to the centre of $\mathbf{a}$ and $\mathbf{z}$. This is neglected here for simplicity.

The procedure shown in figure 1.2 for a nondeterministic machine 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 $\mathbf{a}$.

given:	
$\quad O \subset \mathbb{R}^2$	obstacle
$\quad \mathbf{a} \in \mathbb{R}^2 \setminus O$	start
$\quad \mathbf{z} \in \mathbb{R}^2 \setminus O$	goal
wanted:	
$\quad path : \mathrm{List}(\mathbb{R}^2)$	computed path
invocation:	
$\quad path \leftarrow \mathrm{connect}(\mathbf{a}, \mathbf{z}, [\mathbf{a}])$	$[\mathbf{a}]$ denotes one-element list

$\mathrm{connect}(\mathbf{a}, \mathbf{z} : \mathbb{R}^2;\ path : \mathrm{List}(\mathbb{R}^2)) : \mathrm{List}(\mathbb{R}^2)$	
$\quad \mathbf{p}, \mathbf{q}, \mathbf{r} : \mathbb{R}^2$	
$\quad L : \mathrm{IntervalList}$	
$\quad t : \mathbb{R}$	
$L \leftarrow \mathrm{intersectSegment}(\mathbf{a}, \mathbf{z})$	collision intervals
if $\mathrm{empty}(L)$ **then**	
$\quad path \leftarrow path \circ [\mathbf{z}]$	straight connection is free
else	
$\quad t \leftarrow \mathrm{mid}(\mathrm{head}(L))$	centre of 1st interval
$\quad \mathbf{p} \leftarrow (1-t)\,\mathbf{a} + t\,\mathbf{z}$	reference point
$\quad \mathbf{r} \leftarrow (\mathbf{z} - \mathbf{a})^{\perp}$	perpendicular to straight path
$\quad L \leftarrow \mathrm{intersectLine}(\mathbf{p}, \mathbf{r})$	collision intervals
$\quad$ randomly choose $t \notin L$	arbitrary free point
$\quad \mathbf{q} \leftarrow \mathbf{p} + t\,\mathbf{r}$	crossing point
$\quad path \leftarrow \mathrm{connect}(\mathbf{a}, \mathbf{q}, path)$	recursive solution ...
$\quad path \leftarrow \mathrm{connect}(\mathbf{q}, \mathbf{z}, path)$	... of subproblems
return $path$	

Figure 1.2: Nondeterministic Path Search Strategy

Two functions, "intersectSegment" and "intersectLine" are used to compute the intersection 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

$$\text{intersectSegment}(\mathbf{a}, \mathbf{z}) = \{t \in [0,1] \mid \mathbf{a} + t\,(\mathbf{z} - \mathbf{a}) \in O\}$$

and analogously

$$\text{intersectLine}(\mathbf{p}, \mathbf{r}) = \{t \in \mathbb{R} \mid \mathbf{p} + t\,\mathbf{r} \in O\}$$

These intervals are called *collision intervals* in contrast to the *free intervals* which make up the rest of $[0,1]$ or $\mathbb{R}$ resp. The corresponding sections on the segment and line resp. are named accordingly.

The programme first checks the straight connection of $\mathbf{a}$ and $\mathbf{z}$ for collisions; if none are found, the path is extended by the goal point and returned. Otherwise b is constructed by a point and a direction which are used for an inquiry. The borderline is then crossed at an arbitrary point outside of O in order to divide the problem.

1.3 Deterministic Description

The next task is to make the algorithm deterministic so that it may be used in practice. This implies to search through the possible computations of the nondeterministic machine for an accepting one, but we cannot consider continuously many potential via points. So we shall restrict ourselves to a small number of promising crossing points on each borderline, which are then tried systematically. If we fail to construct a path we take back some decisions and try again using another via point; this is known as *backtracking*.

Recursion Depth A remaining problem is to diagnose failure, i.e. to decide that a certain computation should no longer be tracked because it will not succeed. This is of course impossible and can only be approximated by a limit k on the depth of recursion involved in the path search scheme. If recursion depth 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.

One situation where endless recursion would occur is if a via point has been chosen within a different connected component of free configuration space, i.e. when there is no way to reach it. This however cannot be determined using only local information, so we must limit recursion. On the other hand we need a certain depth to allow for the twists and turns a path has to take to reach the goal. Therefore we need a way to find a useful value for k.

Three approaches have been implemented:

- Allow the user to specify k. This is most useful for interactive experiments and in cases where a suitable limit is already known.

- Search for k starting from zero and increasing by one. This is simple to implement and never overestimates k, thus avoiding excessively long computations and giving quite elegant paths. It works fast enough because running time grows approximately exponential in k and is therefore dominated by the largest value used.
- Search for k in the same way, but keep a tree of already known information about subproblems. This avoids re-computation of early stages of recursion, but needs a considerable amount of memory, depending on the number of alternative via points considered and the final recursion depth needed. An advantage lies in the fact that different subdivisions of a path planning problem can be tracked simultaneously. We can then rate the partial solutions according to some criterion, e.g. path length or recursion depth, and thus try to improve the quality of the constructed path.

Via Points 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 straight connection $(\mathbf{a}, \mathbf{z})$ and l the length of the free section containing $\mathbf{v}$. We use the valuation function

$$V_\beta : \mathbb{R}_0^+ \times \mathbb{R}^+ \to \mathbb{R}_0^+,\ (d, l) \mapsto \frac{d}{l^\beta}$$

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 the best k points. Thus promising points are evaluated first and the number of alternatives at each recursive stage of the algorithm may be limited to a small number.

For a certain class of problems one could try to find an optimal value of β. In the case of a point moving between discs, $\beta = 2/3$ was found to be the best choice for a large number of instances.

The Algorithm Putting all this together we get the following deterministic algorithm. In addition to the parameters of the non-deterministic strategy, this programme takes a limit r_{max} on the recursion depth. It returns a boolean value indicating success or failure, which is of special importance to recursive calls.

given:

$O \subset \mathbb{R}^2$	obstacle
$\mathbf{a} \in \mathbb{R}^2 \setminus O$	start
$\mathbf{z} \in \mathbb{R}^2 \setminus O$	goal
$r_{max} \in \mathbb{N}$	maximum recursion depth

wanted:

$path : \text{List}(\mathbb{R}^2)$	computed path

invocation:

$path \leftarrow [\mathbf{a}]$	one-element list
$\text{connect}(\mathbf{a}, \mathbf{z}, r_{max}, path)$	returns **true** iff successful

```
connect(a, z : ℝ²; rec : ℕ; var path : List(ℝ²)) : Boolean
    q : ℝ²
    L : IntervalList
    path′, via : List(ℝ²)
    ok : Boolean

L ← intersectSegment(a, z)                                  collision intervals
if empty(L) then
    path ← path ∘ [z]                                       straight path is free
    ok ← true
else if rec ≤ 0 then
    ok ← false                                              recursion too deep
else
    via ← viaPoints(a, z, L)                                get list of via points
    path′ ← path                                            remember previous path
    ok ← false
    while ¬ok ∧ ¬empty(via) do
        q ← head(via)                                       first candidate ...
        via ← tail(via)                                     ... is removed from list
        path ← path′                                        previously computed path
        ok ← connect(a, q, rec − 1, path)                   recursive solution ...
             ∧ connect(q, z, rec − 1, path)                 ... of subproblems

return ok
```

Figure 1.3: Systematic Trial of Via Points

A subroutine "viaPoints" is used to find the list of via points which are to be tested. Using information about start and goal as well as collision intervals on $(\mathbf{a}, \mathbf{z})$, it implements the discussed strategy to construct a borderline and so on.

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

The concept of the path search strategy described above can be easily generalized to dimensions higher than two. We again assume that a point is to be moved between given start and goal positions $\mathbf{a}$ and $\mathbf{z}$ avoiding an obstacle O, which need not be represented explicitly. Using a subroutine to answer queries about the intersection of a segment and line resp. with the obstacle, we proceed as before 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 "borderlines" within that hyper-plane hoping to find useful via points. Note that we can no longer guarantee that a suitable crossing point exists on any fixed borderline. Possible strategies include the following:

- Take the direction of motion $\mathbf{a}-\mathbf{z}$ as one vector and supplement it with $\mathbf{d}_2, \ldots, \mathbf{d}_n$ to obtain an orthogonal vector basis of $\mathbb{R}^n$. These directions then specify lines through a point $\mathbf{p}$ chosen on the straight path $(\mathbf{a}, \mathbf{z})$ as before, e.g. as centre of the first segment contained in O. In this way representative directions are used to cover the hyper-plane equally. Should no via point be found on these lines, then $\mathbf{d}_2, \ldots, \mathbf{d}_n$ may be rotated about $\mathbf{a}-\mathbf{z}$ to find a different set of borderlines.

- Randomly choose directions $\mathbf{d}_i \perp (\mathbf{z}-\mathbf{a})$, either a fixed number or repeatedly until enough via points are found. This is quite simple to implement, but has a subtle disadvantage in practical applications. Real world scenes are full of orthogonal features like the walls of a room, and often the desired motion of an object is somehow guided by these features. Thus it is helpful to prefer directions orthogonal to each other and possibly to the axes of the world coordinate frame.

It is not necessary to choose the reference point of a borderline as a point $\mathbf{p} \in (\mathbf{a}, \mathbf{z})$; furthermore it is quite interesting how a point $\mathbf{q} \notin (\mathbf{a}, \mathbf{z})$ may be looked upon. Imagine the moving point to make a step sidewards from the original path without regard to the obstacle. From that new position, another motion is considered and the intersection of the corresponding line in configuration space with the obstacle O is inquired for in order to find a suitable via point. This is especially useful, if the subroutine "intersectLine" cannot handle arbitrary superpositions of motions directly. For example, if translations are only allowed along coordinate axes, then we might find a via point with both coordinates different from $\mathbf{p}$ by moving $\mathbf{q}$ away from $\mathbf{p}$ in one direction and inquiring about a borderline running in the other direction. This is illustrated in figure 1.4.

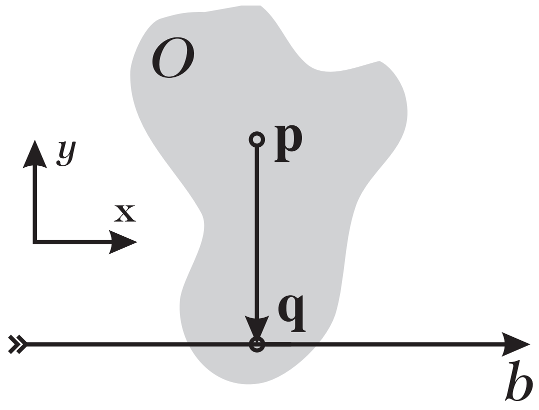


Figure 1.4: Borderline With Reference Point Outside of $(\mathbf{a}, \mathbf{z})$

1.5 Separate Degrees of Freedom

Consider the case where a moving object has distinct kinds of degrees of freedom, e.g. translational and rotational ones, which cannot be used concurrently during a motion. One could also think of a robot arm whose links may only move one at a time; such restrictions can simplify the computation of the intersection of a line with the configuration obstacle remarkably. This means that configuration space is divided accordingly, i.e. d degrees of freedom fall into n groups and we have

$$C = C_0 \times \cdots \times C_{n-1} \subseteq \mathbb{R}^d$$

A feasible path for a moving point must then consist of pieces which each change only coordinates within one group. Another interpretation is that of n coordinates being not necessarily simple numbers but possibly vectors themselves; this makes no difference to the planning strategy and only affects collision detection routines which interpret the coordinates. Thus we shall write configurations as n-tuples, no matter what each component represents.

We now face another design decision: in what order should a "direct path" between $\mathbf{a}$ and $\mathbf{z}$ change these coordinates? The algorithm given here uses a simple cyclic order for reasons of efficiency; this also helps preventing the moving object from running into the obstacle again, because the wrong coordinate is changed first when trying to reach a via point (cf. 1.5). The notion of dodging in a direction perpendicular to the original path is easily conserved; the motion must use another coordinate group or be perpendicular within that group in the previously used 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 reason of clarity, determinism is achieved as demonstrated before and adds a little book-keeping.

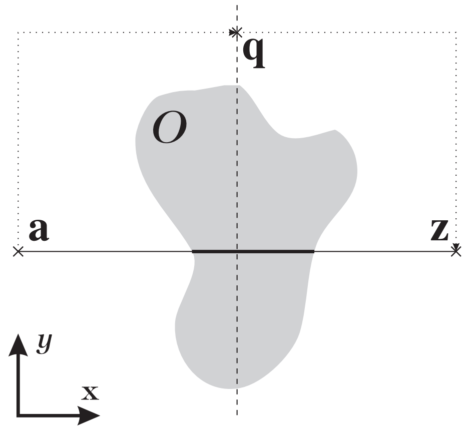


Figure 1.5: Order Is Important When Changing Coordinates

1.6 Related Work

Overmars and Švestka present a probabilistic learning approach to motion planning [OS94] which also aims at avoiding the cost of computing and storing exact information about configuration space while quickly solving practical problems. Their paradigm allows a balance between learning time and success for queries and can easily be applied to robots with non-holonomic constraints. It consists of a global method which constructs a graph representing knowledge about the scene and a *local method* for solving simple motion planning problems.

This local method is required to be a deterministic, symmetrical (with respect to start and goal) motion planner which is allowed to fail "now and then", but is expected to run quickly. Obviously, these requirements are satisfied by the heuristic motion planning scheme described in this work, and hence we propose to use it as the local method in that learning approach. By varying the limit on recursion depth, our strategy can be tuned to run as fast as possible, rarely finding a path, or to be quite effective in path planning for the cost of increased running time.

Overmars and Švestka suggest to use very primitive path planners for best performance of the global method. As a drawback, this clearly requires a larger graph to represent the same amount of knowledge, because the local method is not very smart. Thus nodes in the graph can be saved by a clever planner because the knowledge is not stored explicitly in the global method's data structure but implicitly in the local method's ability to re-compute information if asked to. We think it is desirable to tune the local method in order to balance time versus space according to the needs of a particular application.

Another point of interest is that the local information that our strategy gains through "intersectLine"-calls can be incorporated into the global knowledge stored in the graph. As this information comes at no extra cost and has the suitable structure, i.e. via points and direct connections corresponding to nodes and edges, it can help a lot. Furthermore, using our efficient collision detection scheme for polyhedra connect($a, z : C$; $j : 0 \ldots n-1$; $path$: List(C)) : List(C)

$i, k : \mathbb{N}$
$p, q : C$
L : IntervalList

$i \leftarrow j$ — index of coordinate to change
$p \leftarrow a$ — current position (steps from a to z)

repeat
$p_i \leftarrow z_i$ — change coordinate i
$L \leftarrow$ intersectSegment(a, p)
if empty(L) **then** — piece of direct path is O.K.
$path \leftarrow path \circ [p]$
$a \leftarrow p$
$i \leftarrow i + 1 \pmod n$
until $\neg$ empty(L) $\vee$ $a = z$

if $\neg$ empty(L) **then** — collision in coordinate i
$(q, k) \leftarrow$ viaPoints(a, p, L) — q dodges in coordinate k
$path \leftarrow$ connect($a, q, k, path$) — change coordinate k before i
$path \leftarrow$ connect($q, z, i, path$) — change coordinate i before k

return $path$

Figure 1.6: Planning Strategy for Separate Degrees of Freedom

enlarges the learning scheme's domain to three-dimensional work spaces; the spherical geometry approach helps in guiding the edge adding strategies.

There are some interesting similarities between our work and theirs, too, and these concern the separation of degrees of freedom which seems quite common though often hidden. For example, their local method for car-like robots consists of simply checking two possible "straight connections" of configurations. These consist of sequences of pure translational and rotational motions. Regarding articulated robots, they suggest to define the neighbor configurations in such a way as to use only one degree of freedom at a time. These ideas are in a certain way generalized by our separating degrees of freedom. All in all, our heuristic path search strategy seems to be a good completion of their learning approach.

Chapter 2

Efficient Collision Detection

In the first chapter, a heuristic path search strategy has been introduced and discussed which is applicable to a wide range of problems. For each special case, configuration space will be different and a suitable parameterization has to be found; but apart from that, the planning algorithm remains unchanged. Still lacking for practical use is a subroutine to answer inquiries about the intersection of a line with the configuration obstacle—a segment can be handled analogously. It is there that some work must be done to adapt our strategy to each new kind of motion planning problem.

"intersectLine" needs a description of all objects and obstacles in physical space or *work space*, as opposed to configuration space. We will restrict ourselves to handling polyhedral objects, because real-world bodies can be approximated by them and collision detection will be more efficient. From this implicit description, information about the configuration obstacle will be derived.

The actual input parameters specify the motions of these objects, their interpretation depends on configuration space. For a jointed robot arm this could mean to move all of its joints simultaneously, perhaps each one with its own speed, and the resulting motion could be very complicated. To avoid such difficulties we separate the degrees of freedom and impose 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; it was shown how the path search stage of the algorithm deals with such a separation.

As a drawback this restriction might turn the motion planning problem unsolvable, e.g. because two objects would have to rotate concurrently and such a motion is not considered. However we assume the problem is not too hard to solve and allow the strategy to fail under such circumstances.

We now proceed as follows. Consider only one moving object and one fixed obstacle, to achieve this we may look at each such pair; for concurrently moving objects, we exploit 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]. Enveloping techniques are introduced to reduce average running time substantially. Translational and rotational motions are handled separately, as this is the case within the collision detection algorithm.

2.1 Static Detection of Overlap

Polyhedra We define *"polyhedron"* in such a manner that it means a compact, connected subset of $\mathbb{R}^3$, bounded by flat surfaces which are themselves polygons, and equal to the closure of its open interior. A *boundary representation* is used which lists all vertices, edges, and faces of a polyhedron as well as their adjacency relations. This gives the description complexity $|P|$ of a polyhedron P as the number of all its vertices, edges, and faces: $v + e + f$. Note that $v, e, f = \Theta(|P|)$, so it makes no sense to further distinguish between these numbers for the purpose of asymptotic running time. Convexity is required for all faces and can be achieved in pseudo-linear time with no relevant increase in the polyhedron's complexity.

Furthermore we expect our data structure "polyhedron" to support enumeration of all faces in an arbitrary order. For each face f, the normal equation of its supporting plane is needed, with its normal vector $\mathbf{n}$ pointing to the outside of the polyhedron and thus giving local interior-exterior information. In addition, all vertices of a face must be listed in counter-clockwise order, where f is viewed from the outside, and bear their position in space.

The Predicate Two polyhedra overlap iff their surfaces intersect or one is entirely contained in the other. We derive a 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.

Thus let $\mathbf{p}, \mathbf{q} \in \mathbb{R}^3$ be the head and tail of an edge, which is written as a line segment

$$l_{\mathbf{p},\mathbf{q}} := \{\mathbf{p} + \lambda(\mathbf{q} - \mathbf{p}) \mid 0 \le \lambda \le 1\} \subseteq \mathbb{R}^3$$

Besides, a convex face of the other body be given by a plane

$$P_{\mathbf{n},n_0} := \{\mathbf{x} \in \mathbb{R}^3 \mid \mathbf{n}^{\mathrm{T}}\mathbf{x} = n_0\} \subseteq \mathbb{R}^3,\ \mathbf{n} \in \mathbb{R}^3,\ |\mathbf{n}| = 1,\ n_0 \in \mathbb{R}$$

with normal vector $\mathbf{n}$ and distance n_0 from the origin, and by its distinct vertices $\mathbf{v}_0, \ldots, \mathbf{v}_{k-1} \in P_{\mathbf{n},n_0}$, $k \ge 3$, ordered counter-clockwise as shown in figure 2.1. For simplicity, let $\mathbf{v}_k := \mathbf{v}_0, \mathbf{v}_{k+1} := \mathbf{v}_1$.

The face itself results in the set of points

$$\begin{aligned} f := f_{\mathbf{v}_0,\ldots,\mathbf{v}_{k-1}} &:= \{\sum_{i=0}^{k-1} \lambda_i \mathbf{v}_i \mid \sum_{i=0}^{k-1} \lambda_i = 1,\ \forall\, 0 \le i < k:\ 0 \le \lambda_i \in \mathbb{R}\} \\ &= \{x \in P_{\mathbf{n},n_0} \mid \forall\, 0 \le i < k:\ \det[\mathbf{n}, \mathbf{v}_{i+1} - \mathbf{v}_i, \mathbf{x} - \mathbf{v}_i] \ge 0\} \subseteq \mathbb{R}^3 \end{aligned}$$

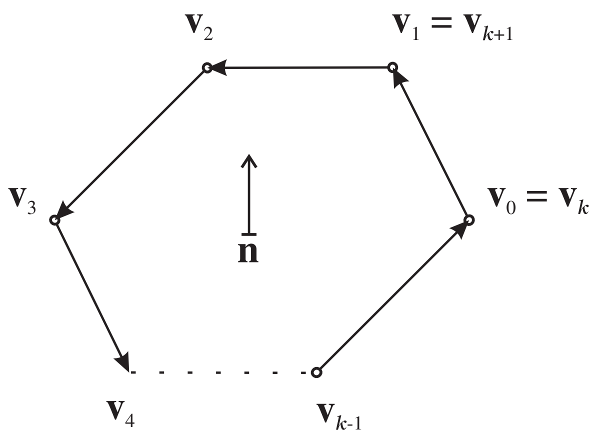


Figure 2.1: Schematic Polygon

where the first form shows f as the convex combination of its vertices, and the second as the set of all points lying "left" of all oriented edges.

We now distinguish three cases according to the relative position of the oriented line through $l_{\mathbf{p},\mathbf{q}}$ with respect to $P_{\mathbf{n},n_0}$.

1. "outward"

   We have $\mathbf{n}^{\mathrm{T}}(\mathbf{q}-\mathbf{p}) > 0$. The line intersects the plane in exactly one point $\mathbf{s}$, which lies on the segment iff the latter starts in the interior and ends in the exterior closed half-space defined by the oriented plane. The intersection point $\mathbf{s}$ belongs to f if it is "left" of all edges. Thus the result for this case is:

$$l_{\mathbf{p},\mathbf{q}} \cap f \neq \emptyset \iff$$

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

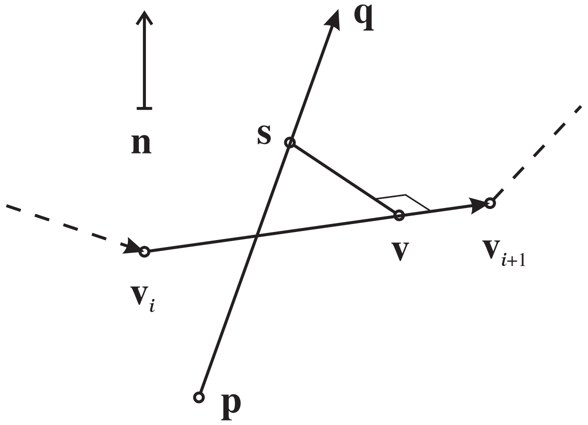


Figure 2.2: Intersection of Line Segment and Face

This can indeed be written without explicitly mentioning $\mathbf{s}$ (cf. fig. 2.2), and by substituting the determinant for the equivalent one with four-dimensional vectors, symmetry becomes more obvious:

$$\forall\, 0 \leq i < k:\ \det \begin{pmatrix} 1 & 1 & 1 & 1 \\ \mathbf{p} & \mathbf{q} & \mathbf{v}_i & \mathbf{v}_{i+1} \end{pmatrix} \geq 0$$

2. "inward"

   We have $\mathbf{n}^{\mathrm{T}}(\mathbf{q}-\mathbf{p}) < 0$ and exchanging $\mathbf{p}$ for $\mathbf{q}$ yields

$$l_{\mathbf{p},\mathbf{q}} \cap f \neq \emptyset \iff$$

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

3. "parallel"

   We have $\mathbf{n}^{\mathrm{T}}(\mathbf{q}-\mathbf{p}) = 0$, a degenerate case. For $\mathbf{p} \notin P_{\mathbf{n},n_0}$, there is no intersection, so assume $\mathbf{n}^{\mathrm{T}}\mathbf{p} = n_0$. In this situation an edge of one polyhedron lies in the same plane as a face of the other one; we could derive a predicate to check for overlap, but we prefer a simpler method.

Lemma 2.1:
This degenerate case can be safely ignored.

Proof:
Assume that $l_{\mathbf{p},\mathbf{q}} \cap f \neq \emptyset$ and $l_{\mathbf{p},\mathbf{q}} \subset P_{\mathbf{n},n_0} =: \mathcal{P}$, i.e. there is a degenerate overlap. We will show that another pair of edge and face exists which also overlaps, but is not contained in a plane. Thus if we consider all edge-face-pairs in our collision test we may neglect these degenerate cases and still get the correct result.

For the purpose of this proof assume that the polyhedra are coloured red and blue resp. and inspect the plane which contains the red edge and the blue face. We expand the red area to the whole (connected component of the) intersection of the red polyhedron with the plane $\mathcal{P}$ and proceed analogously for the blue one. Because our definition of "polyhedron" forbids two-dimensional objects, all outer vertices and edges in $\mathcal{P}$ have adjacent edges and faces which are not contained in that plane.

Ignoring all interior vertices and edges of the coloured areas yields a red and a blue polygon which overlap; we take a closer look at the boundary of that overlapping area, in particular at its vertices (cf. figure 2.3, dashed lines denote interior structure). Such a vertex $\mathbf{v}$ is either a vertex of at least one coloured polyhedron, say the red one, in which case a red edge leaves $\mathcal{P}$ at $\mathbf{v}$ and intersects the blue face containing $\mathbf{v}$. Or $\mathbf{v}$ is the intersection of two coloured edges e_{red} and e_{blue} (which are not collinear), and thus a blue face adjacent to e_{blue} leaves the plane and intersects e_{red}. In both cases, the two objects contain three linearly independent directions and thus they cannot overlap degenerately. ∎

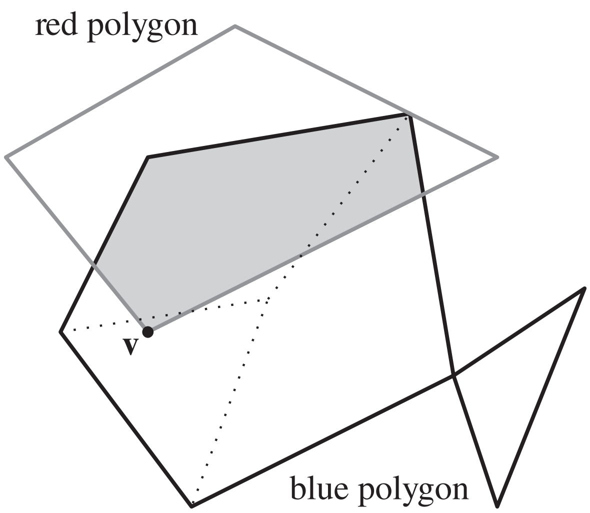


Figure 2.3: Red-Blue-Intersection

Putting all this together, we get the following predicate which is used to test for the intersection of an edge and a face; note that its size is $\Theta(k)$.

$$\begin{aligned} S(l_{\mathbf{p},\mathbf{q}}, f_{\mathbf{v}_0,\ldots,\mathbf{v}_{k-1}}) \;:=\; & \Big(\mathbf{n}^{\mathrm{T}}(\mathbf{q}-\mathbf{p}) > 0 \;\wedge\; \mathbf{n}^{\mathrm{T}}\mathbf{q} - n_0 \geq 0 \;\wedge\; \mathbf{n}^{\mathrm{T}}\mathbf{p} - n_0 \leq 0 \\ & \wedge \bigwedge_{0\leq i<k} \det[\mathbf{q}-\mathbf{p}, \mathbf{v}_{i+1}-\mathbf{v}_i, \mathbf{p}-\mathbf{v}_i] \geq 0\Big) \\ \vee \; & \Big(\mathbf{n}^{\mathrm{T}}(\mathbf{p}-\mathbf{q}) < 0 \;\wedge\; \mathbf{n}^{\mathrm{T}}\mathbf{p} - n_0 \geq 0 \;\wedge\; \mathbf{n}^{\mathrm{T}}\mathbf{q} - n_0 \leq 0 \\ & \wedge \bigwedge_{0\leq i<k} \det[\mathbf{p}-\mathbf{q}, \mathbf{v}_{i+1}-\mathbf{v}_i, \mathbf{q}-\mathbf{v}_i] \geq 0\Big) \end{aligned}$$

If two polyhedra P and Q are given, we denote the sets of their edges by E_P and E_Q and the sets of their faces by F_P and F_Q. The complete predicate for the intersection of surfaces results in an expression of complexity $\Theta(|P||Q|)$:

$$S(P,Q) := \Big(\bigvee_{\{\mathbf{p},\mathbf{q}\}\in E_P}\;\bigvee_{f\in F_Q} S(l_{\mathbf{p},\mathbf{q}}, f)\Big) \vee \Big(\bigvee_{\{\mathbf{p},\mathbf{q}\}\in E_Q}\;\bigvee_{f\in F_P} S(l_{\mathbf{p},\mathbf{q}}, f)\Big)$$

Containment We still have to capture those situations where one polyhedron is completely contained within the other. The test we will derive yields *true* for a superset of the configurations where $P \subset Q$, but only for such with $P \cap Q \neq \emptyset$. We check whether some point of P, say an arbitrary vertex $\mathbf{v}$, lies within Q or not; this is easily decided by looking at a ray from $\mathbf{v}$ into a random direction. If the ray intersects any edges or vertices of Q, we choose another direction, else we count the number of intersections with faces. An odd number means that $\mathbf{v} \in Q \Rightarrow \{\mathbf{v}\} \subset P \cap Q \neq \emptyset$, and vice versa $P \subset Q \Rightarrow \mathbf{v} \in Q$ is correctly ascertained; thus the test meets the requirements mentioned above. Of course this test must also be applied symmetrically to test whether $Q \subset P$.

2.2 Translational Case

The moving body carries out a translation from its current position into direction $\mathbf{r} \in \mathbb{R}^3$, possibly bounded by an amount of $\lambda_{Max} \in \mathbb{R}$. We write such a general translation as a mapping

$$T_{\mathbf{r}}^{\lambda} : \mathbb{R}^3 \to \mathbb{R}^3,\ \mathbf{x} \mapsto T_{\mathbf{r}}^{\lambda}(\mathbf{x}) := \mathbf{x}^{\lambda} := \mathbf{x} + \lambda\mathbf{r}$$

and a motion with constant velocity is given by the continuous function $\Lambda : \mathbb{R} \to \mathbb{R},\ t \mapsto \Lambda(t) := t\lambda_{Max}$. Thus the polyhedron at time t is $T_{\mathbf{r}}^{\Lambda(t)}(P) = P^{\Lambda(t)}$. In the case of a bounded motion ("intersectSegment"), $t = 0$ denotes the beginning and $t = 1$ the end of the motion; else we have one initial position $t = 0$ and simultaneously consider two infinite motions ($\lambda_{Max} := 1,\ t \in \mathbb{R}$).

The Predicate To simplify the evaluation of $S(l_{\mathbf{p},\mathbf{q}}, f)$, we assume that the edge is moving while the face remains fixed. For a given direction $\mathbf{r}$ we now have to determine all relevant times of collision, i.e.

$$\begin{aligned}\mathcal{L} &:= \{\lambda \in \mathbb{R} \mid S(T_{\mathbf{r}}^{\lambda}(l_{\mathbf{p},\mathbf{q}}), f_{\mathbf{v}_0,\ldots,\mathbf{v}_{k-1}})\} \\ &\subseteq \{\lambda \in \mathbb{R} \mid T_{\mathbf{r}}^{\lambda}(l_{\mathbf{p},\mathbf{q}}) \cap f_{\mathbf{v}_0,\ldots,\mathbf{v}_{k-1}} \neq \emptyset\}\end{aligned}$$

This set $\mathcal{L}$ may be restricted to $[0, \lambda_{Max}]$ if appropriate. Note that those points in time are ignored where a degenerate overlap occurs; yet this does not change the final result if all pairs of edges and faces are considered.

With the above notation, $T_{\mathbf{r}}^{\lambda}(l_{\mathbf{p},\mathbf{q}}) = l_{\mathbf{p}^{\lambda},\mathbf{q}^{\lambda}}$. The predicate $S(l_{\mathbf{p},\mathbf{q}}, f)$ consists of only two kinds of inequalities; here $\mathcal{R}$ denotes any comparison symbol $<, \leq, \geq, >$ and $\mathcal{R}^-$ represents the symbol resulting from an exchange of $<$ for $>$.

1. $\mathbf{n}^{\mathrm{T}}\mathbf{x}^{\lambda} - c\ \mathcal{R}\ 0$

   Formulas of this kind inform about the relative direction of two vectors ($\mathbf{n}^{\mathrm{T}}(\mathbf{q} - \mathbf{p}) > 0$) or about which side of a plane a point lies on ($\mathbf{n}^{\mathrm{T}}\mathbf{p} - n_0 \leq 0$). During the motion, f and thus $\mathbf{n}$ remain fixed, $\mathbf{p}$ and $\mathbf{q}$ change.

   $$\begin{aligned}\mathbf{n}^{\mathrm{T}}(\mathbf{x} + \lambda\mathbf{r}) - c\ \mathcal{R}\ 0 &\iff \\ \lambda\underbrace{\mathbf{n}^{\mathrm{T}}\mathbf{r}}_{a} + \underbrace{\mathbf{n}^{\mathrm{T}}\mathbf{x} - c}_{b}\ \mathcal{R}\ 0 &\iff \left\{\begin{array}{llll}\lambda & \mathcal{R} & -b/a & \text{if} \quad a > 0 \\ 0 & \mathcal{R} & -b & \text{if} \quad a = 0 \\ \lambda & \mathcal{R}^- & -b/a & \text{if} \quad a < 0\end{array}\right.\end{aligned}$$

   In each case, at most one interval of feasible λ-values is determined, this can also be $\mathbb{R}$ as a whole or the empty set.

2. $\det[\mathbf{x}, \mathbf{w}, \mathbf{p}^{\lambda} - \mathbf{v}]\ \mathcal{R}\ 0$

As $\mathbf{q}^\lambda - \mathbf{p}^\lambda = \mathbf{q} - \mathbf{p}$ this is the general form of $\det[\mathbf{q}-\mathbf{p}, \mathbf{v}_{i+1}-\mathbf{v}_i, \mathbf{p}-\mathbf{v}_i] \geq 0$ if $\mathbf{p}$ and $\mathbf{q}$ move. $\mathbf{v}$, $\mathbf{w}$, and $\mathbf{x}$ denote fixed points and directions resp. This yields a linear inequality just like above:

$$\det[\mathbf{x}, \mathbf{w}, \mathbf{p}^\lambda - \mathbf{v}] = (\mathbf{x} \times \mathbf{w})^{\mathrm{T}}(\mathbf{p} + \lambda \mathbf{r} - \mathbf{v}) = \lambda \underbrace{(\mathbf{x} \times \mathbf{w})^{\mathrm{T}}\mathbf{r}}_{a} + \underbrace{(\mathbf{x} \times \mathbf{w})^{\mathrm{T}}(\mathbf{p} - \mathbf{v})}_{b}$$

We have shown that the predicate S consists of such inequalities as atomic pieces, whose sets of solutions can be represented by at most one interval. The main idea now is to evaluate S using lists of intervals rather than *true* and *false* as intermediate results, thus yielding all values λ with $l_{\mathbf{p}^\lambda, \mathbf{q}^\lambda} \cap f_{\mathbf{v}_0, \ldots, \mathbf{v}_{k-1}} \neq \emptyset$. 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 balanced tree for $S(P, Q)$ has $\Theta(n)$ leafs because of the predicate's size, $n := |P||Q|$; its depth can be restricted to $O(\log n)$. 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 The test for containment of one polyhedron within the other now has to be adapted to the translational motion of P while Q is assumed to stay fixed. To simplify the test, we only consider the direction of motion $\mathbf{r}$ as a possible direction of the ray. After choosing a suitable point $\mathbf{v} \in P$, we check the line $\{\mathbf{v} + t\mathbf{r} \mid t \in \mathbb{R}\}$ for intersections with faces of Q. The normal vector of each face tells whether $\mathbf{v}$ enters or leaves Q during the motion; we can also exploit that $\mathbf{v}^\infty \notin Q$. In case of a degeneracy, another point $\mathbf{v}$ is tried, but we cannot restrict ourselves to the vertices of P here. We rather 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.

For a fixed direction of motion, we compute such a triangle once for every polyhedron; this takes time $O(|P|)$. The intersection test of the line with Q takes time $O(|Q|)$, if we use sorting to bring the results into order and consider that degeneracies appear with possibility 0, we get $O(|Q| \log |Q|)$ as the running time for one such test. Thus this part is irrelevant for the total running time of the collision detection scheme.

Enveloping Techniques If we evaluate the formula which expresses overlap of two polyhedra naively, a quadratic number of atomic inequalities must be considered. This of course only reflects the worst-case complexity of the problem, for practical applications and especially for simple scenes we observe that a much smaller number suffices. The moving object often stays far away from most obstacles, so that a detailed analysis of collision is superfluous; we can rather use enveloping techniques to quickly state collision-freeness.

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; the advantage lies in the fact that 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. For the latter purpose, the moving object's z-interval is enlarged so as to contain the object during the whole motion. If we describe each interval as a one-dimensional circle with center $p_i^\kappa \in \mathrm{I\!R}$, $i \in \{1,2\}$, $\kappa \in \{x,y,z\}$, and radius $0 \le r_i^\kappa \in \mathrm{I\!R}$, the bounding rectangloids are written as

$$[p_i^x - r_i^x,\ p_i^x + r_i^x] \times [p_i^y - r_i^y,\ p_i^y + r_i^y] \times [p_i^z - r_i^z,\ p_i^z + r_i^z]$$

and do not overlap iff

$$|p_1^x - p_2^x| > r_1^x + r_2^x \ \vee\ |p_1^y - p_2^y| > r_1^y + r_2^y \ \vee\ |p_1^z - p_2^z| > r_1^z + r_2^z$$

If these boxes interfere with each other, we eventually have to consider all edge-face-pairs. In order to speed things up, we also consider bounding rectangloids for individual faces. Only if the intersection of two faces cannot be ruled out, we take a close look at their edges and carry out all edge-face-tests.

The investigation whether one polyhedron is contained within the other 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. Intersection of a line and a face may be ruled out by such tests, as the line is parallel to the z-axis this is especially simple.

Incorporating all these improvements into the collision detection scheme results in a dramatic speed-up for many real-world problems. This matches nicely with the heuristic approach of the path search strategy and achieves the goal of quickly solving simple instances of arbitrary motion planning problems.

2.3 Rotational Case

The moving object now rotates about a fixed axis, either through a given angle or through one complete turn, this corresponds to bounded and unbounded motions resp. It is well known that rotations can be described by matrices $\mathbf{U} \in \mathrm{I\!R}^{3\times3}$, $\mathbf{U}^{\mathrm{T}}\mathbf{U} = \mathbf{I}$, $\det(\mathbf{U}) = 1$, as a mapping

$$R_{\mathbf{U}} : \mathrm{I\!R}^3 \to \mathrm{I\!R}^3,\ \mathbf{x} \mapsto R_{\mathbf{U}}(\mathbf{x}) := \mathbf{U}\mathbf{x}$$

Every rotation $R_{\mathbf{U}}$ is determined by an axis $\mathbf{r} \in \mathbb{R}^3$, $|\mathbf{r}| = 1$ and an angle $\varphi \in \mathbb{R}$; the axis of rotation is assumed to pass through the origin of the coordinate system. This yields

$$\mathbf{U} = \mathbf{U}_{\mathbf{r},\varphi} = (1 - \cos\varphi)\mathbf{r}\mathbf{r}^{\mathrm{T}} + \cos\varphi\mathbf{I} + \sin\varphi \begin{pmatrix} 0 & -r_3 & r_2 \\ r_3 & 0 & -r_1 \\ -r_2 & r_1 & 0 \end{pmatrix}$$

for the matrix and vice versa

$$\varphi = \arccos\left(\frac{1}{2}(u_{1,1} + u_{2,2} + u_{3,3} - 1)\right)$$

$$\mathbf{r} = (2\sin\varphi)^{-1}(u_{3,2} - u_{2,3}, u_{1,3} - u_{3,1}, u_{2,1} - u_{1,2})^{\mathrm{T}}$$

so that we may specify a rotation $R_{\mathbf{r},\varphi} := R_{\mathbf{U}_{\mathbf{r},\varphi}}$ directly by axis and angle. We can then apply it to a vector $\mathbf{x} \in \mathbb{R}^3$ by the function

$$\mathbf{x} \mapsto (1 - \cos\varphi)\mathbf{r}^{\mathrm{T}}\mathbf{x}\mathbf{r} + \cos\varphi\mathbf{x} + \sin\varphi\mathbf{r} \times \mathbf{x} \tag{2.1}$$

which describes the rotation in a coordinate system with base vectors $\mathbf{r}$, $\mathbf{x}$, and $\mathbf{r} \times \mathbf{x}$; we shall make use of this function later.

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.

2.3.1 Motion Planning for Pure Rotations

Quaternions The set $\mathbb{R}^4$ with the usual vector operations and a suitably defined multiplication forms a non-commutative division algebra. Quaternions $\mathbf{Q} = (q_0, q_1, q_2, q_3)^{\mathrm{T}} \in \mathbb{R}^4$ will now be written as $\mathbf{Q} = (q_0, \mathbf{q})$, where q_0 is called scalar part and $\mathbf{q} = (q_1, q_2, q_3)^{\mathrm{T}}$ vector part of $\mathbf{Q}$. This in a natural way gives inclusions of $\mathbb{R}$ and $\mathbb{R}^3$ into the set $\mathbb{R}^4$ of quaternions. Multiplication is defined by the bilinear function

$$(p_0, \mathbf{p}) \cdot (q_0, \mathbf{q}) := (p_0 q_0 - \mathbf{p}^{\mathrm{T}}\mathbf{q},\ p_0\mathbf{q} + q_0\mathbf{p} + \mathbf{p} \times \mathbf{q})$$

In analogy to complex numbers, quaternions are *conjugated* by $\mathbf{Q} = (q_0, \mathbf{q}) \mapsto \mathbf{Q}^* := (q_0, -\mathbf{q})$ with the properties $|\mathbf{Q}|^2 = \mathbf{Q} \cdot \mathbf{Q}^*$ and $\mathbf{Q}^{-1} = \mathbf{Q}^*/|\mathbf{Q}|^2$.

A rotation about axis $\mathbf{r}$ through angle φ is given by the quaternion

$$\mathbf{Q}_{\mathbf{r},\varphi} = \left(\cos\frac{\varphi}{2}, \sin\frac{\varphi}{2}\mathbf{r}\right)$$

describing the mapping $R_{\mathbf{r},\varphi}$ as

$$R_{\mathbf{Q}_{\mathbf{r},\varphi}} = R_{\mathbf{r},\varphi} : \mathbb{R}^3 \to \mathbb{R}^3,\ \mathbf{x} \mapsto \mathbf{Q}_{\mathbf{r},\varphi} \cdot (0, \mathbf{x}) \cdot \mathbf{Q}^*_{\mathbf{r},\varphi}$$

One should note that $\mathbf{Q}$ and $-\mathbf{Q}$ obviously represent the same rotation, so that the mapping of quaternions to the matrices introduced above is not injective, and vice versa we may as well map $\mathbf{r}$ and φ to $\mathbf{Q}_{\mathbf{r},\varphi}$ or to $\mathbf{Q}_{\mathbf{r},\varphi+2\pi}$. This ambiguity results from the fact that one can exchange $\mathbf{r}$ and φ for $-\mathbf{r}$ and $-\varphi$, this still describes the same rotation. If we assume the angle to be positive and take $\mathbf{Q}_{\mathbf{r},\varphi}$ as defined above for the corresponding quaternion, the mapping becomes univocal.

All $\mathbf{Q}_{\mathbf{r},\varphi}$ are unit quaternions, i.e. of Euclidean length one, and form a group under quaternion product. Each such unit quaternion $\mathbf{Q} = (q_0, \mathbf{q})$ represents a rotation through the angle $\varphi = 2\arccos q_0$ about the axis $\mathbf{r} = (\sin\frac{\varphi}{2})^{-1}\mathbf{q}$; the latter is not defined for an angle of zero. Composition of rotations corresponds to multiplication of quaternions, i.e. $R_\mathbf{P} \circ R_\mathbf{Q} = R_{\mathbf{P}\cdot\mathbf{Q}}$.

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 $1 \in \mathbb{R}$ as the unit quaternion $(1, \mathbf{0}) \in \mathcal{S}_4$ and $\mathbf{r} \in \mathcal{S}_3$ as $(0, \mathbf{r}) \in \mathcal{S}_4$ for convenience. Also, we will identify points and their position vectors for simplicity.

If we talk of the orientation of a polyhedron we usually describe it by a rotation with reference to some fixed initial position; thus the initial orientation corresponds to the unity $1 = (1, \mathbf{0})$ of the quaternion algebra which denotes non-rotation. During a rotational motion the object's orientation 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

$$C_{\mathbf{P},\mathbf{Q}} := \{\mu\mathbf{P} + \nu\mathbf{Q} \mid \mu, \nu \in \mathbb{R};\ \mu^2 + \nu^2 = 1\} = \{\cos\varphi\mathbf{P} + \sin\varphi\mathbf{Q} \mid \varphi \in \mathbb{R}\}$$

During a full rotation an object's orientation describes half a great circle on $\mathcal{S}_4$; because of the ambiguity between $\mathbf{Q}$ and $-\mathbf{Q}$ we could also say it describes both halves simultaneously. If a fixed axis $\mathbf{r} \in \mathcal{S}_3$ is chosen, then two turns correspond to the orientations $\{\mathbf{Q}_{\mathbf{r},\varphi} = \cos\frac{\varphi}{2} \cdot 1 + \sin\frac{\varphi}{2} \cdot \mathbf{r} \mid \varphi \in [0, 4\pi)\} = C_{1,\mathbf{r}}$ which form a great circle with axes $1, \mathbf{r} \in \mathcal{S}_4$. To achieve other axes, let an initial orientation of $\mathbf{P} \in \mathcal{S}_4$ be given together with an axis of rotation $\mathbf{r}$; again $\{\mathbf{Q}_{\mathbf{r},\varphi} \cdot \mathbf{P} \mid \varphi \in [0, 4\pi)\} = C_{\mathbf{P},\mathbf{r}\cdot\mathbf{P}}$ forms a great circle, but now with axes $\mathbf{P}$ and $\mathbf{r} \cdot \mathbf{P}$. These are also orthogonal because

$$\mathbf{P}^\mathrm{T}(\mathbf{r} \cdot \mathbf{P}) = (p_0, \mathbf{p})^\mathrm{T}(-\mathbf{r}^\mathrm{T}\mathbf{p},\ p_0\mathbf{r} + \mathbf{r} \times \mathbf{p}) = -p_0\,\mathbf{r}^\mathrm{T}\mathbf{p} + p_0\,\mathbf{p}^\mathrm{T}\mathbf{r} + \underbrace{\mathbf{p}^\mathrm{T}(\mathbf{r} \times \mathbf{p})}_{\det[\mathbf{p},\mathbf{r},\mathbf{p}]=0} = 0$$

Thus complete rotations map to great circles, and vice versa. 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 $\mathbf{Q} = \mathbf{r} \cdot \mathbf{P}$; this results from $\mathbf{Q} \perp \mathbf{P} \iff q_0p_0 + \mathbf{q}^\mathrm{T}\mathbf{p} = 0$ and thus

$$\begin{aligned} \mathbf{Q} \cdot \mathbf{P}^* &= (q_0, \mathbf{q}) \cdot (p_0, -\mathbf{p}) \\ &= (q_0p_0 + \mathbf{q}^\mathrm{T}\mathbf{p}, -q_0\mathbf{p} + p_0\mathbf{q} - \mathbf{q} \times \mathbf{p}) \\ &= (0, \mathbf{r}) \in \mathcal{S}^4 \end{aligned}$$

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*, which meant searching for via points on a hyper-plane perpendicular to the direct connection, also has a simple meaning based on the following lemma.

Lemma 2.2:
Great circles on $\mathcal{S}_4$ are perpendicular iff the corresponding axes of rotation are.

Proof:
Let $\mathbf{P} \in \mathcal{S}_4$ be the single common point of both great circles and $\mathbf{q}, \mathbf{r} \in \mathcal{S}_3$ their respective axes of rotation. Then $\mathbf{Q} := \mathbf{q} \cdot \mathbf{P}$ and $\mathbf{R} := \mathbf{r} \cdot \mathbf{P}$ resp. complete the description of the two circles $C_{\mathbf{P},\mathbf{Q}}$ and $C_{\mathbf{P},\mathbf{R}}$. These are orthogonal iff $\forall \mathbf{S} \in C_{\mathbf{P},\mathbf{Q}} :\ \mathbf{S} \perp \mathbf{R}$ and symmetrically, this follows from $\mathbf{P} \perp \mathbf{R}$ and $\mathbf{Q} \perp \mathbf{R}$ which is obvious because of the following:

$$
\begin{aligned}
\mathbf{Q}^{\mathrm{T}}\mathbf{R} &= (\mathbf{q} \cdot \mathbf{P})^{\mathrm{T}}(\mathbf{r} \cdot \mathbf{P}) \\
&= (-\mathbf{q}^{\mathrm{T}}\mathbf{p},\ p_0\mathbf{q} + \mathbf{q} \times \mathbf{p})^{\mathrm{T}}(-\mathbf{r}^{\mathrm{T}}\mathbf{p},\ p_0\mathbf{r} + \mathbf{r} \times \mathbf{p}) \\
&= \mathbf{q}^{\mathrm{T}}\mathbf{p}\ \mathbf{r}^{\mathrm{T}}\mathbf{p} + p_0^2\mathbf{q}^{\mathrm{T}}\mathbf{r} + p_0(\mathbf{q}^{\mathrm{T}}(\mathbf{r} \times \mathbf{p}) + (\mathbf{q} \times \mathbf{p})^{\mathrm{T}}\mathbf{r}) + \underbrace{(\mathbf{q} \times \mathbf{p})^{\mathrm{T}}(\mathbf{r} \times \mathbf{p})}_{\mathbf{q}^{\mathrm{T}}\mathbf{r}\ \mathbf{p}^{\mathrm{T}}\mathbf{p} - \mathbf{q}^{\mathrm{T}}\mathbf{p}\ \mathbf{r}^{\mathrm{T}}\mathbf{p}} \\
&= \underbrace{(p_0^2 + \mathbf{p}^{\mathrm{T}}\mathbf{p})}_{|\mathbf{P}|^2=1}\mathbf{q}^{\mathrm{T}}\mathbf{r} + p_0\underbrace{(\det[\mathbf{q},\mathbf{r},\mathbf{p}] + \det[\mathbf{r},\mathbf{q},\mathbf{p}])}_{0} \\
&= \mathbf{q}^{\mathrm{T}}\mathbf{r}
\end{aligned}
$$

The other way round, if we start with $\mathbf{Q}$ and $\mathbf{R}$, we define $\mathbf{q} := \mathbf{Q} \cdot \mathbf{P}^*$ and $\mathbf{r}$ analogously and find that $\mathbf{q}^{\mathrm{T}}\mathbf{r} = \mathbf{P}^*(\mathbf{Q}^{\mathrm{T}} \cdot \mathbf{R})\mathbf{P} = \mathbf{Q}^{\mathrm{T}}\mathbf{R}$. ■

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.

2.3.2 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 instead of linear ones. The test for containment and our enveloping strategy are also influenced slightly.

The Predicate For convenience, we choose the notation $\mathbf{x}_\varphi := R_{\mathbf{r},\varphi}(\mathbf{x})$ for a point rotated about a fixed axis through a certain angle, which may vary. We may again assume that the edge is moving and the face is not, to achieve this we might have to swap the role of the object and obstacle while negating the axis of rotation. The two forms of inequalities and their solutions then are

1. $\mathbf{n}^\mathrm{T}\mathbf{x}_\varphi - c \mathrel{\mathcal{R}} 0$

   Here $\mathbf{n}$ remains fixed while $\mathbf{p}$ and $\mathbf{q}$ are moving, but we observe that $\mathbf{q}_\varphi - \mathbf{p}_\varphi = (\mathbf{q}-\mathbf{p})_\varphi$. Refer to equation 2.1, p. 23, for the explanation of $\mathbf{x}_\varphi$.

$$\begin{aligned} && \mathbf{n}^\mathrm{T}\mathbf{x}_\varphi - c &\mathrel{\mathcal{R}} 0 \\ \iff && \mathbf{n}^\mathrm{T}\left((1-\cos\varphi)\mathbf{r}^\mathrm{T}\mathbf{x}\mathbf{r} + \cos\varphi\mathbf{x} + \sin\varphi\mathbf{r}\times\mathbf{x}\right) - c &\mathrel{\mathcal{R}} 0 \\ \iff && \underbrace{(\mathbf{n}^\mathrm{T}\mathbf{x} - \mathbf{n}^\mathrm{T}\mathbf{r}\mathbf{r}^\mathrm{T}\mathbf{x})}_{\alpha}\cos\varphi + \underbrace{(\mathbf{n}^\mathrm{T}\mathbf{r}\times\mathbf{x})}_{\beta}\sin\varphi + \underbrace{(\mathbf{n}^\mathrm{T}\mathbf{r}\mathbf{r}^\mathrm{T}\mathbf{x} - c)}_{\gamma} &\mathrel{\mathcal{R}} 0 \end{aligned}$$

   The original test is thus reduced to $\alpha\cos\varphi + \beta\sin\varphi + \gamma \mathrel{\mathcal{R}} 0$; the corresponding equation describes the intersection of the line $\alpha x + \beta y + \gamma = 0$ with the unit circle. Depending on $\mathcal{R}$, the inequality describes the intersection of a half-plane with the unit circle. Intervals of solution for the inequality naturally arise from the zeros of the equality, which are found as follows.

   For angles $-\pi < \varphi < \pi$ the substitution $t := \tan\frac{\varphi}{2}$ is used and yields $\cos\varphi = \frac{1-t^2}{1+t^2}$, $\sin\varphi = \frac{2t}{1+t^2}$ and after multiplication by $1+t^2$ we get

$$\alpha(1-t^2) + \beta 2t + \gamma(1+t^2) = 0 \iff (\gamma-\alpha)t^2 + 2\beta t + (\alpha+\gamma) = 0$$

   For $\alpha \neq \gamma$ the well known solution formula leads to the condition $\alpha^2+\beta^2 \geq \gamma^2$ for solubility and then to the zeros

$$t_{1,2} = \frac{\beta \mp \sqrt{\alpha^2+\beta^2-\gamma^2}}{\alpha-\gamma}$$

   which are transformed to angles in $[0, 2\pi)$ by $\varphi = 2\arctan t$. The degenerate case of a linear equation is trivial.

   The angle $\varphi = \pi$ solves the equation iff $-\alpha+\gamma = 0$, i.e. the line runs through $(-1, 0)$. A second intersection point exists iff $\beta \neq 0$, else the line is vertical and only touches the circle or for $\alpha = 0$ it degenerates to the plane or the empty set. The second solution can be found by the former approach which leads to $2\beta t + 2\alpha = 0$ in this special case.

2. $\det[\mathbf{x}_\varphi, \mathbf{w}, \mathbf{p}_\varphi - \mathbf{v}] \ \mathcal{R}\ 0$

   This stems from the test $\det[\mathbf{q}-\mathbf{p}, \mathbf{v}_{i+1}-\mathbf{v}_i, \mathbf{p}-\mathbf{v}_i] \geq 0$ for an edge $l_{\mathbf{p},\mathbf{q}}$ moving against a face $f_{\mathbf{v}_0,\ldots,\mathbf{v}_{k-1}}$. We again reduce this to the same kind of inequality as the first case. Therefore we take apart the determinant by $\det[\mathbf{x}_\varphi, \mathbf{w}, \mathbf{p}_\varphi - \mathbf{v}] = \det[\mathbf{x}_\varphi, \mathbf{w}, \mathbf{p}_\varphi] - \det[\mathbf{x}_\varphi, \mathbf{w}, \mathbf{v}]$ into two parts describing the volumes of parallelepipeds. Thus they are obviously invariant under rotation and we may apply $R_{\mathbf{r},-\varphi}$ yielding $\det[\mathbf{x}_\varphi, \mathbf{w}, \mathbf{p}_\varphi] = \det[\mathbf{x}, \mathbf{w}_{-\varphi}, \mathbf{p}] = \det[\mathbf{w}_{-\varphi}, \mathbf{p}, \mathbf{x}]$. Therefore we only deal with determinants of the form

   $$\det[\mathbf{x}_\varphi, \mathbf{v}, \mathbf{w}] = (\mathbf{v} \times \mathbf{w})^{\mathrm{T}}\mathbf{x}_\varphi = \mathbf{s}^{\mathrm{T}}\mathbf{x}_\varphi$$

   where $\mathbf{v}$ and $\mathbf{w}$ and thus $\mathbf{s}$ are fixed while $\mathbf{x}$ is rotating. Application of formula 2.1 immediately yields the desired result

   $$\begin{aligned} \mathbf{s}^{\mathrm{T}}\mathbf{x}_\varphi &= \mathbf{s}^{\mathrm{T}}\left((1-\cos\varphi)\mathbf{r}^{\mathrm{T}}\mathbf{x}\mathbf{r} + \cos\varphi\mathbf{x} + \sin\varphi\mathbf{r}\times\mathbf{x}\right) \\ &= (\mathbf{s}^{\mathrm{T}}\mathbf{x} - \mathbf{s}^{\mathrm{T}}\mathbf{r}\mathbf{r}^{\mathrm{T}}\mathbf{x})\cos\varphi + (\mathbf{s}^{\mathrm{T}}(\mathbf{r}\times\mathbf{x}))\sin\varphi + \mathbf{s}^{\mathrm{T}}\mathbf{r}\mathbf{r}^{\mathrm{T}}\mathbf{x} \\ &= \alpha\cos\varphi + \beta\sin\varphi + \gamma \end{aligned}$$

As shown above, the atomic pieces of our predicate $S(P,Q)$ can be easily computed and represented by at most two intervals. The evaluation of the whole predicate takes place as described in the translational case and gives the list of intervals corresponding to all times at which surfaces collide.

Containment We choose a suitable point $\mathbf{v}$ on the surface of P and test for its containment within Q to cover all situations where P is contained wholly within P. Therefore the circle $\mathbf{v}$ describes during rotation is intersected with the faces of Q; this yields all times when $\mathbf{v}$ enters or leaves the obstacle during its motion. If faces are not oriented or the circle does not intersect Q, this will not suffice. In this case we send a ray from the point's original position to find out whether $\mathbf{v} \in Q$ initially holds or not. Should any degeneracy appear, we randomly pick another $\mathbf{v}$ from a triangle on P's surface that is not perpendicular to the axis of rotation $\mathbf{r}$. Again not every interior point can cause a degenerate situation.

Enveloping Techniques There is only one difference to the translational case. Bounding boxes are computed not in a Cartesian coordinate system but in a cylindrical one 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. It can be neglected in tests to achieve simpler two-dimensional intersection problems or enlarged to contain the moving object throughout the whole rotation.

Appendix A

Some Selected Examples

A.1 Two-dimensional Implementation

We have implemented the two-dimensional path search strategy for a simple motion planning problem, i.e. moving a point between discs within a square part of the plane. In this case, work space and configuration space are the same and one can directly study the heuristic motion planning scheme.

The programme offers an interactive graphical user interface as well as methods for pseudo-random generation of scenes and their statistical evaluation. As a special option the number of via points considered on a borderline can be restricted to two; furthermore the two degrees of freedom may be handled separately to construct paths which are piece-wise parallel to the coordinate axes.

Because our heuristic strategy often yields paths which seem a bit inelegant—especially if the limit on recursion depth is too high—, we incorporated an algorithm for shortening them. A graph is build whose vertices are the corners of the path and whose edges are all direct connections not intersecting an obstacle. The shortest way from start to goal is then taken as a new path. This method can be easily extended to higher dimensions where planning a path takes more time than shortening it.

A.1.1 Separate Degrees of Freedom

Figure A.1 shows two paths for a point moving between randomly distributed discs of equal size in the unit square. Start and goal were determined by hand; the paths were planned considering at most two via points per borderline and with minimal necessary recursion depth. We will use the number of "intersect"-calls as a measure of running time for the path search stage because it does not depend on the implementation of the collision detection subroutine.

The left part of the figure depicts a motion with separated degrees of freedom. During three seconds 625 calls were made, six levels of recursion sufficed to solve the problem. In contrast to that, the same figure on its right side shows a path which was planned without any restriction on the direction of motion. A tree-like structure was used to store information about subproblems in order to speed up the search for a minimal necessary recursion depth. This turned out to be three and only 26 queries to the collision detection were needed for that search.

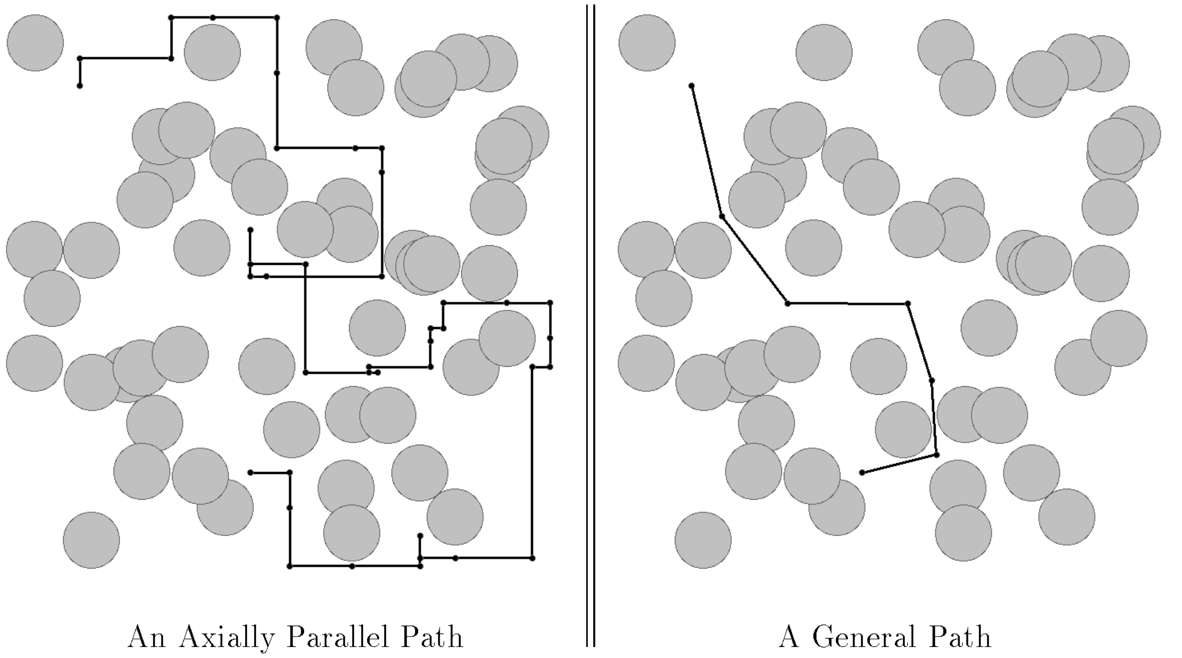


Figure A.1: Comparison of Two Paths

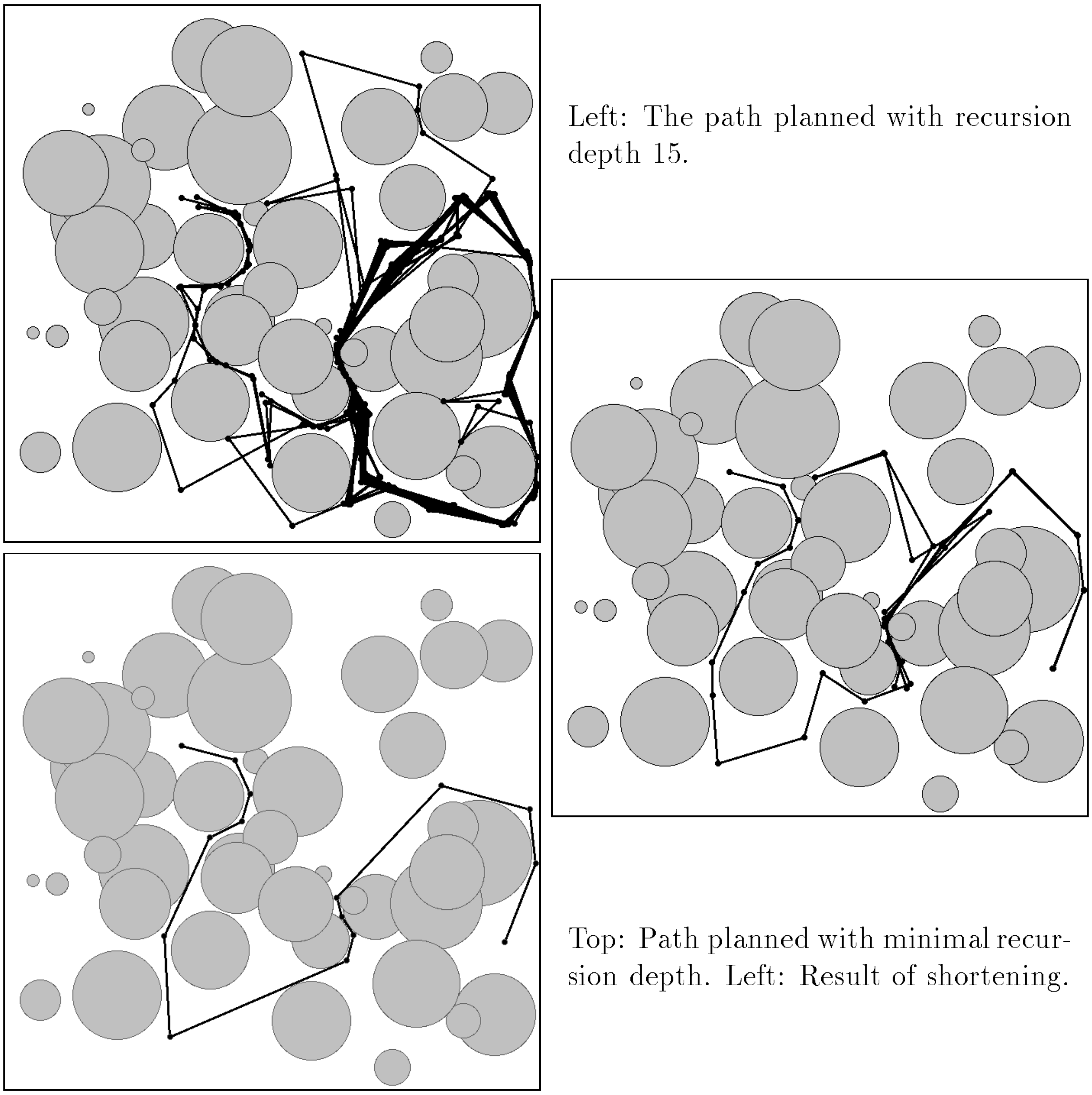
A.1.2 Impact of Recursion Depth

The next example shows a scene which consists of discs of various sizes leaving only small passages between them. A boundary further restricts the free space of the point. In this situation a motion has been planned with a given limit of at most 15 levels of recursion; it is shown in figure A.2 in the upper left corner. The path is very complicated and not only intersects itself several times but also surrounds some obstacles. 13414 calls were needed taking considerable time; if more than two via points per borderline are considered then even 26895 calls are generated.

In contrast to that a path with the optimal, i.e. minimal, recursion depth of eight is remarkably more elegant and faster to construct, taking only 684 "intersect"-calls. Even this relatively good path shows some self-intersections; these are caused by the restrictions on the positions of borderlines and via points. Shortening the path remedies this problem.

A.1.3 Statistical Evaluation

The high speed of the path search strategy for simple problems in the plane enables us to generate and solve a large number of pseudo-random motion planning problems. We



Left: The path planned with recursion depth 15.

Top: Path planned with minimal recursion depth. Left: Result of shortening.

Figure A.2: Impact of Recursion Depth

can then tune certain parameters and analyze their influence on how efficient certain classes of problems are solved. For the following experiments the same set of 10000 problems was generated, each consisting of fifty discs of equal size spread in the unit square, which was to be traversed from upper left to lower right. All instances are guaranteed to be solvable, and the path search strategy of course does not simply choose a path along the border.

We use two measures for the efficiency of a solution: the number of “intersect”-calls made by the path planner, which would dominate running time for realistic three-dimensional scenes, and the minimal recursion depth allowing a solution, which indicates the complexity of the constructed path. We show both the maximal and average

β	at most two via points				all via points			
	recursion depth		"intersect"-calls		recursion depth		"intersect"-calls	
	max.	mean	max.	mean	max.	mean	max.	mean
0	9	4.606	3362	194.4	6	3.812	3688	321.2
1/2	7	4.471	1039	167.0	6	3.812	4246	304.5
2/3	8	4.463	1937	167.6	6	3.812	4367	302.6
1	9	4.573	7655	194.8	6	3.812	3808	308.4
2	20	5.528	$2 \cdot 10^7$	$2 \cdot 10^4$	6	3.812	4545	339.8

Table A.1: Variation Of β With Minimal Recursion Depth

β	at most two via points				all via points			
	recursion depth		"intersect"-calls		recursion depth		"intersect"-calls	
	max.	mean	max.	mean	max.	mean	max.	mean
1/2	7	4.471	557	94.0	6	3.812	3609	236.8
2/3	8	4.463	1084	93.9	6	3.812	3649	234.5
1	9	4.573	2939	106.2	6	3.812	3033	234.6

Table A.2: Variation of β With Minimal Recursion Depth Re-Using Information

(arithmetic mean) values, experiments were made with or without a restriction to two via points per borderline.

Valuation of Via Points In section 1.3, p. 8, we introduced a valuation function V_β to select via points; this function depends on a parameter β. Table A.1 shows results for five different values of β, the minimal recursion depth was determined by repeatedly calling the path search strategy with increasing recursion limit. It turned out that $\beta = 2$ was extremely bad and sometimes took millions of "intersect"-calls and a large recursion depth of twenty to solve certain scenes; the whole experiment thus lasted four days and 11.5 hours. Other values could be analyzed within a few hours, and $\beta \approx 2/3$ yields an optimal balance between distance and length of free interval.

Table A.2 shows results for three of these β-values computed with the help of a tree-like data structure to store information about previous solved sub-problems. This avoids re-computing the early stages of the path construction for each increased recursion limit, and thus speeds up motion planning considerably. Note that it is worth-while keeping recursion not too deeply nested because of the exponential growth in the number of non-deterministic computations. Searching for an accepting one in a depth-first manner avoids memory problems, but a breadth-first approach yields better paths and takes less time. But even with depth-first it usually pays to repeatedly try in vain instead of over-estimating the necessary depth considerably; this was illustrated in figure A.2.

Table A.3 shows results for a fixed limit on recursion depth, which was nine for the left and six for the right column. More values of β have been tried, but $\beta = 2$ was

β	≤ 2 via points "intersect"-calls		all via points "intersect"-calls	
	max.	mean	max.	mean
0	3069	94.74	7306	149.2
2/5	1933	51.04	?	?
1/2	1793	47.26	4178	90.33
4/7	1802	46.41	3961	86.49
2/3	1837	45.97	3633	83.38
1	3159	72.19	2056	90.52

Table A.3: Variation Of β With Fixed Recursion Depth

avoided; we can say that V_2 is a useless valuation function for this class of problems.

Choice of Borderline We mentioned the topic of choosing a borderline which is then searched for suitable via points. The position of that borderline has a great impact on our path search strategy because it determines the position of the via points relatively to the obstacles and to the start and goal. It also determines the sub-division of the original problem into two parts, which should be both easier to solve and approximately of equal difficulty.

Using $\beta = 2/3$, at most two via points and the best recursion strategy—with its tree structure—we varied the base point of the borderline on the straight path $(\mathbf{a}, \mathbf{z})$ but kept it perpendicular. The base point was placed in the midst of a collision interval or randomly, it was also taken as the start of the first collision interval. The intervals considered were the first one, the one in the centre of $(\mathbf{a}, \mathbf{z})$ and of the interval list resp., and a random one. The most elaborate choice was the centre of a collision interval

location of base point	recursion depth		"intersect"-calls	
	max.	mean	max.	mean
start of 1st interval [a]	10	5.972	9086	547.6
randomly in 1st interval	8	4.612	806	106.8
centre of 1st interval	8	4.463	1084	93.9
centre of random interval	7	4.026	440	64.9
centre of middle interval (in list)	7	3.649	591	53.2
centre next to centre of $(\mathbf{a}, \mathbf{z})$	7	3.616	581	51.3
centre of $(\mathbf{a}, \mathbf{z})$	9	4.380	1359	89.6
randomly in $(\mathbf{a}, \mathbf{z})$	12	5.972	5375	211.0

[a] Due to the unfortunate behaviour of this strategy for many problems, recursion depth had to be limited to 10; this restriction prevented the solution of 173 problems. These contribute to the number of calls, but not to recursion depth statistics.

Table A.4: Different Base Points for Borderline closest to the centre of $(\mathbf{a}, \mathbf{z})$.

The results are given in table A.4. It shows that a random point acts quite similar as the corresponding average point, so this gives no advantage. Centres of intervals and points close to the middle of $(\mathbf{a}, \mathbf{z})$ are to be preferred, and the optimum is the synthesis of these two criteria.

A.2 Motion Planning For Polyhedra

We developed a system for **i**nteractive **g**raphical **o**bject assembly and **r**obot simulation—in short IGOR. Its aim is to simulate assembly processes with all participating objects and tools, while a user is interacting with the system to specify, analyze, and correct motions. Objects are modelled by polyhedra which can be rotated and translated freely; emphasis is put on efficient on-line collision detection in this context.

To simplify work with the system, a module for automatic motion planning was added. Thus the user only needs to specify the final position of an object while a collision-free path is computed by the programme. Thus we have implemented the heuristic path search strategy described here and integrated it into the IGOR-system; in this way motions for single polyhedra using rotations and translations can be constructed and incorporated into assembly plans. We will now show some examples of motion planning problems together with their solutions.

A.2.1 Putting a Cube Into a Box

The scene consists of a cube fitting into a square-like box with little clearance; it is considered a typical application for our motion planning scheme. A translational motion is looked for which takes the cube from a place outside the box to its final position inside. Figure A.3 shows a sequence of configurations during the motion, with the cube starting behind the box on the upper right side. The problem was solved within seconds and yielded a path which could be shortened; it is represented by a black line in space. The cube is shown at the vertices of that path, i.e. at those times where the direction of motion changes, and reaches its goal within four steps.

A.2.2 The 'Piano Movers' Problem

We consider a room containing a piano which should be moved from one wall to another; a 90° rotation is necessary as well as avoiding collision with a chair. The free-space in this scene is rather small for this kind of heuristic motion planning; the task is very demanding and needs a few hours to solve. Note that the algorithm does not "know" that the piano should stay inside the room, there is an open door and plenty of space on the outside, but those positions are hard to reach. All via points were considered—so the solution was not missed—, but then every translational borderline yields two via points outside the room. We did not want to change this situation however, because

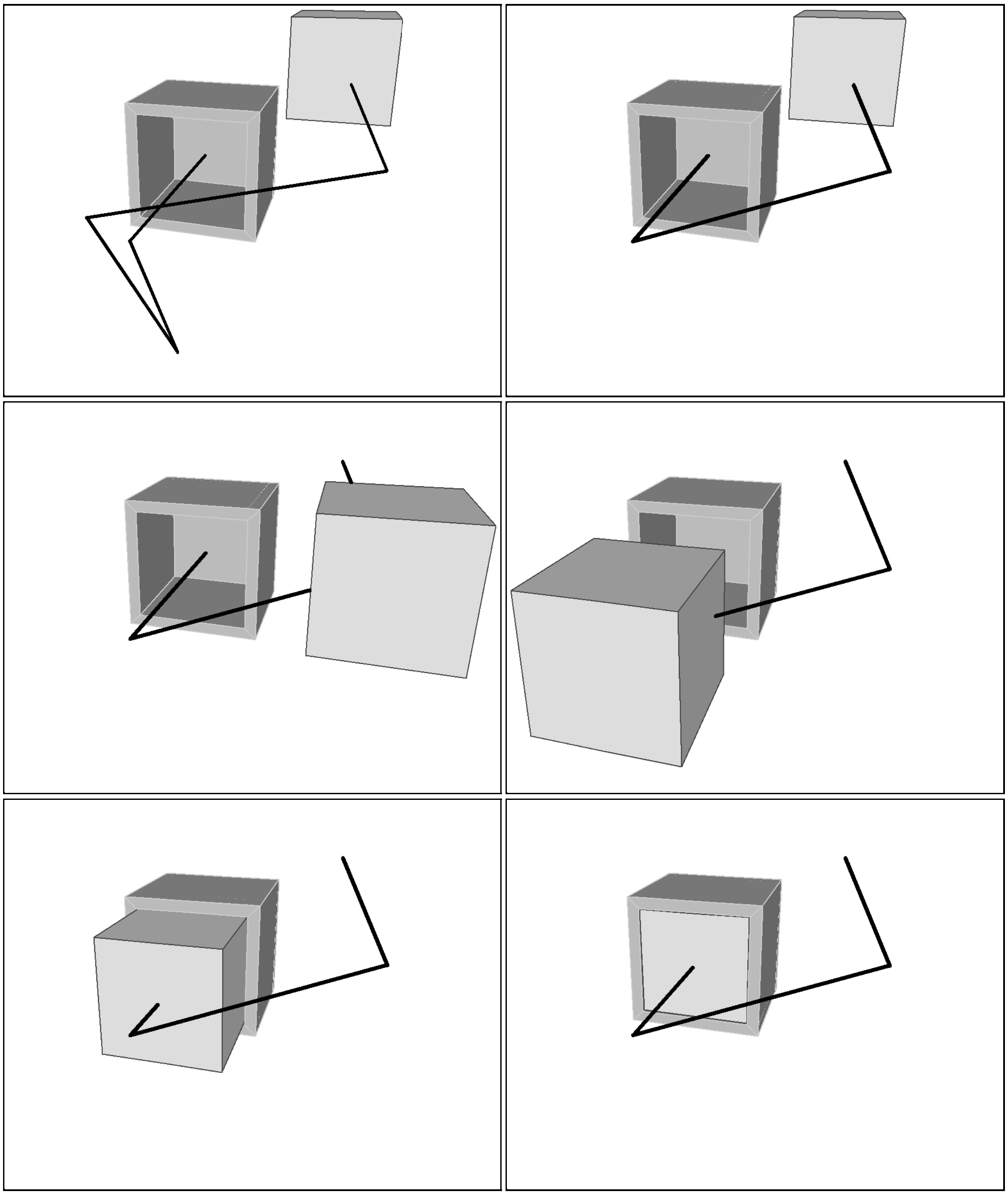


Figure A.3: Putting a Cube Into a Box

the path search strategy is universally applicable, but it certainly shows the limits of the programme.

Figure A.4 presents the computed solution; the simple search for minimal recursion depth was used here. The individual steps of the motion are shown from left to right and top to bottom with every single picture showing a top and a front view of the room. A wire frame representation is used and the translational part of the path can be seen, too. This path can be shortened, yielding a surprisingly elegant solution. It is pictured in A.2.2 as a perspective view from the front, elevated slightly, with some transparent walls.

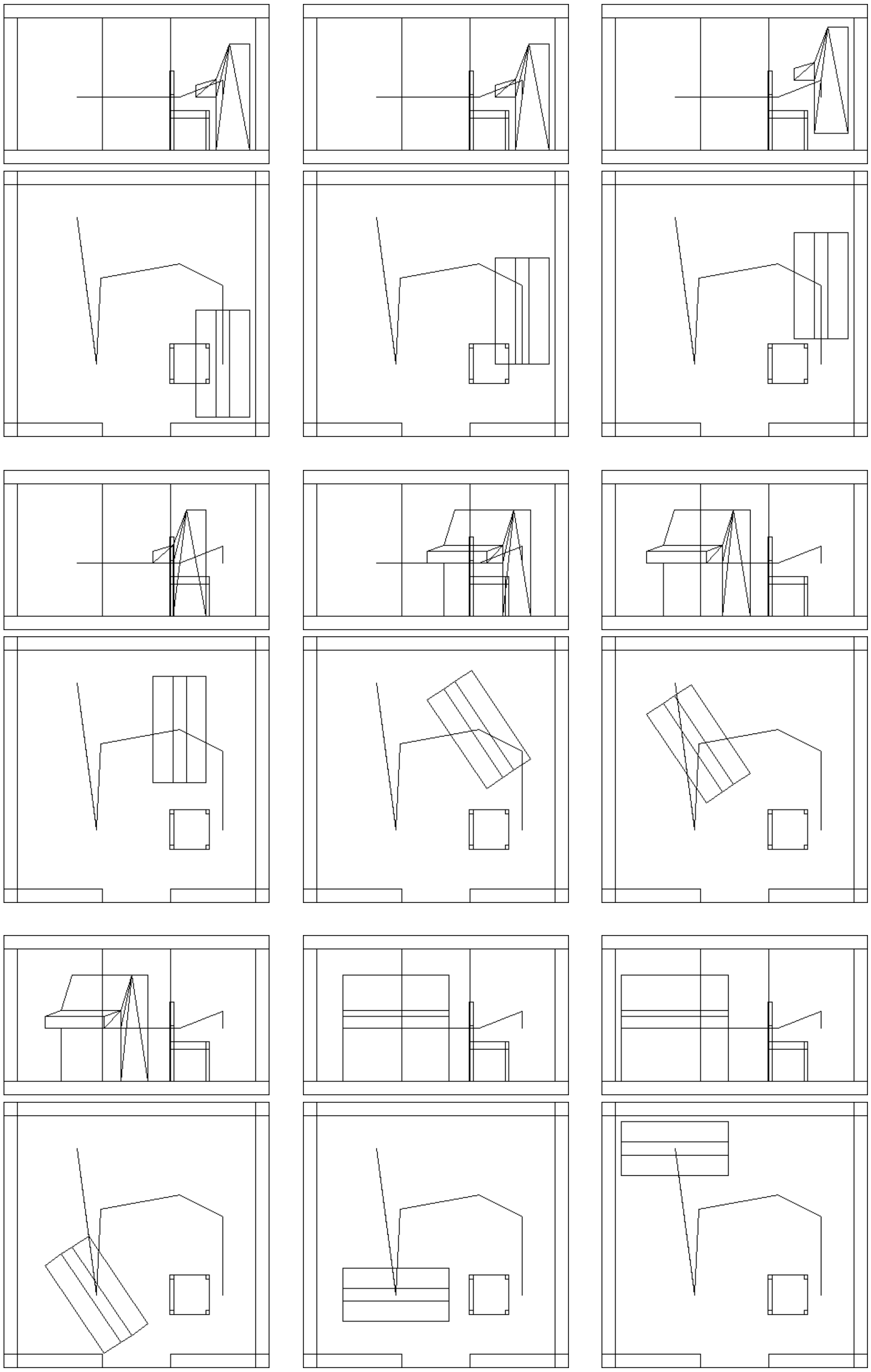
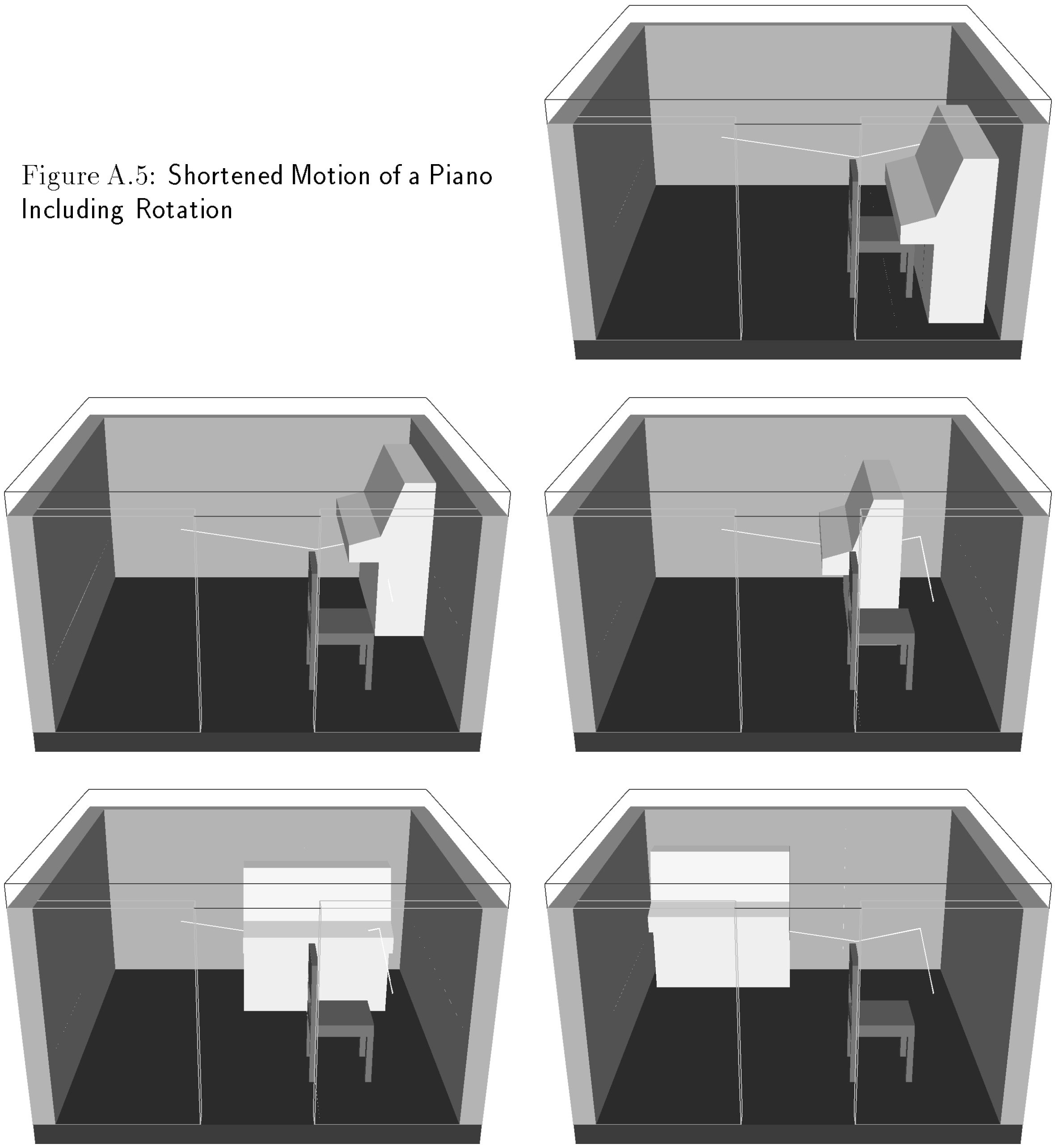


Figure A.4: Motion of a Piano Including Rotation

Figure A.5: Shortened Motion of a Piano Including Rotation



Bibliography

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

[Co75] G. E. Collins
Quantifier Elimination for Real Closed Fields by Cylindrical Algebraic Decomposition
Lecture Notes in Computer Science, No. 33, Springer-Verlag New York, (1975), pp. 135–183

[La91] Jean Claude Latombe
Robot motion planning
Kluwer Academic Publishers, (1991)

[Lo83] Tomás Lozano-Peréz
Spatial Planning: A Configuration Space Approach
IEEE Trans. Computers, C-32, No. 2, (Feb 1983), pp. 108–120

[OS94] Mark H. Overmars, Petr Švestka
A Probabilistic Learning Approach to Motion Planning
Technical Report UU-CS-1994-03, Dept. Comput. Sci., Utrecht Univ., Utrecht, the Netherlands, (1994)

[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 Informatik, Universität des Saarlandes, Saarbrücken, Germany, (1994)

[SS82] J. Schwartz, M. Sharir
On the 'Piano Movers' Problem, II. General Techniques for Computing Topological Properties of Real Algebraic Manifolds

Comp. Sci. Dept., New York University report 41, (1982)

also in: „Planning, Geometry and Complexity of Robot Motion“
J. Hopcroft, J. Schwartz, M. Sharir, Ablex publishing corp. New Jersey, (1987), Ch. 5, pp. 154–186