Fast Planning of a Good Path for a Manipulator Using Potential Fields on a Non-Uniform Grid in C-space.

Dissertation

zur Erlangung des akademischen Grades
"Doktor der technischen Wissenschaften"

Eingereicht von

Carl Van Geem

Mai 1996

Erster Begutachter: Univ.-Doz. Dr. Sabine Stifter
Zweiter Begutachter: Prof. Dr. Witold Jacak

Angefertigt am: Forschungsinstitut für symbolisches Rechnen
Technisch-Naturwissenschaftliche Fakultät
Johannes Kepler Universität Linz

Eidesstattliche Erklärung

Ich versichere, daß ich die Dissertation selbständig verfaßt habe, andere als die angegebenen Quellen und Hilfsmittel nicht verwendet und mich auch sonst keiner unerlaubten Hilfe bedient habe.

Linz, am 8. Mai 1996 Carl Van Geem.

Fast Planning of a Good Path for a Manipulator Using Potential Fields on a Non-Uniform Grid in C-space.

Carl Van Geem
RISC-Linz
Research Institute for Symbolic Computation
Joh. Kepler University
A-4040 Linz, Austria
Internet: Carl.Van.Geem@risc.uni-linz.ac.at

May 1996.

abstract

This thesis considers the Robot Motion Planning Problem for a robot arm using the concept of Configuration Space. The approach described here leads to a path with some particular properties. It also gives rise to a complete solution of the problem under certain conditions.

The thesis tends to be applicable, since the considerations on which the approach is ented are practice oriented. On the other hand only bits and parts are implemented for the moment, and some parts of the work are still in a theoretical phase. We strongly believe that these parts can be implemented and fit together.

Our original results of research are the following.

- We observed that the use of a non-uniform grid in combination with a potential field similar to the Manhattan distance on a uniform grid makes the use of a repulsive potential superfluous when we want to end up with a safe but (in Configuration Space) short path. In order to represent the non-uniform grid, we have chosen to use the so called GET Representation for a 2^k-tree (due to Klaus Burger: [Bur92]).

- As a motivation for the use of a non-uniform grid as opposed to a uniform grid, we counted out for a set of polygons with a particular shape what the gain in number of cells is when using cells of different sizes.

- We observed that it is possible to translate the algorithms due to Irene Gargantini ([Gar82b]) for neighbour computation in quad- and octrees to the GET Representation, and did so. The neighbour computation in the non-uniform grid (looking for adjacent cells of a given cell) is an essential operation for the construction of a potential field on the grid.

- We implemented a non-hierarchical construction of a potential field on a non-uniform grid. We also describe a hierarchical construction. The advantage of a hierarchical construction is that we at first can construct the non-uniform grid and a potential field on it up to a certain dimension and expand the model of the Configuration Space to a higher dimension later on, when necessary. This corresponds more or less to the approach of gross and fine motion planning as it is known in the literature.

- When it is observed during execution of a path that an obstacle is positioned differently compared to its position during the planning faze, the model has to be updated. We describe how to update the representation of the grid and the potential field (without recomputation of the whole model) when slight changes in the position of an obstacle have occured.

- Finally the construction of the non-uniform grid has to be accurate and fast. For this we suggest to use a fast and accurate collision checker. One of the collision checkers that could be used for this is based on the Roider method (see [Sti92a]), particularly well suited for collision detection between two superellipsoids. For a particular geometric model for some robot arms we show how to find sufficiently accurate enclosing superellipsoids for the links using some trigonometry for the computations involved.

Our results as described above can be found respectively in publications and technical reports [VG93] and [VG94b], [VG95b], [VG94a] and [VG96b], [VG95d] and [VG95c], and [VG96a].

Our work restricts itself to finding a path and giving information that can be used by a trajectory planner. We did not look into the trajectory planning aspects, because we felt this would be beyond the scope of this thesis.

Deutsche Kurzfassung.

Diese Dissertation behandelt das Problem der Bewegungsplanung eines Roboters, wobei wir das Konzept eines *Konfigurationsraums* verwenden. Unser Zugang liefert einen Pfad mit besonderen Eigenschaften und eine vollständige Lösung des Problems unter bestimmten Bedingungen.

Bei der Erstellung dieser Arbeit wurden insbesondere Fragen der praktischen Anwendbarkeit vorrangig berücksichtigt. Jedoch sind die Algorithmen bislang nur teilweise implementiert. Die technische Realisierbarkeit steht dabei außer Frage.

Die zentralen Ergebnisse der vorliegenden Dissertation sind die folgenden.

- Es zeigt sich, daß die Kombination von einem nicht-uniformen Gitter und einem Potentialfeld, ähnlich zum Manhattan Abstand, ein abstoßendes Potentialfeld überflüssig macht, wenn wir einen sicheren und im Konfigurationsraum kurzen Pfad suchen. Für die Representation des Gitters verwenden wir die *GET representation* für einen 2^k-Baum (siehe [Bur92]).

- Zur Motivation, nicht-uniforme Gitter statt uniforme zu verwenden, haben wir für gewisse Polygone ausgerechnet, wieviele Zellen weniger man braucht mit nicht-uniformem Gittern.

- Wir zeigen, daß die Algorithmen von Irene Gargantini ([Gar82b]) für die Nachbarschaft von Zellen in einen "quadtree" und "octree" für die GET representation übersetzbar sind. Diese Algorithmen wurden auch implementiert. Für die Gestaltung des Potentialfeldes ist die Berechnung von Nachbarzellen in einem Gitter elementar.

- Ein nicht-hierarchischer Algorithmus wurde implementiert. Darüber hinaus haben wir einen hierarchischen Algorithmus entwickelt, der den Konfigurationsraum und das Potentialfeld zuerst nur teilweise berechnet und später in höhere Dimensionen ausbaut. Diese Technik ist ähnlich zum bekannten "gross" und "fine motion planning".

- Es wird ein einfaches Verfahren beschrieben zur Änderung des Modells für ein leicht versetztes Hindernis.

- Auch die Gestaltung des Gitters muss sorgfältig und schnell gehen. Wir schlagen vor, einen sorgfältigen und schnellen Kollisionsdetektor anzuwenden. Wir haben uns dabei für die *Roider method* (siehe [Sti92a]) entschieden, die besonders geeignet ist für *Superellipsen* und *Superellipsoide*. Für ein bestimmtes geometrisches Modell zeigen wir, daß sich Robotergelenkstücke durch Superellipsen und Superellipsoide beschreiben lassen. Dafür werden trigonometrische Methoden verwendet.

Diese Ergebnisse sind in den folgenden Publikationen und "Technical Reports" dokumentiert: [VG93] und [VG94b], [VG95b], [VG94a] und [VG96b], [VG95d] und [VG95c], und [VG96a].

Die vorgestellten Resultate ergeben zusammengenomen ein Verfahren zum "motion plannung", das darüber hinaus bereits wesentliche Schritte für ein "trajectory planning" vorwegnimmt. Die Beschreibung eines vollständigen "trajectory planner" würde den Rahm dieser Arbeit jedoch sprengen.

Outline.

In the first chapter we describe the problem of Robot Motion Planning as it is known in the literature and we specify the problem tackled here. In the second chapter we give a brief overview of the different approaches to the Robot Motion Planning Problem and we indicate the main aspects of our approach to the particular instance of the problem as we have specified it. In Chapter 3 we give a list of references not aiming at making a complete list of publications related to the field of robot motion planning. By classifying them we ended up with a historical overview of the field. Chapter 4 motivates the use of a non-uniform grid rather than a uniform grid. We come up with some formulas for the number of cells in a grid, and indicate the order of magnitude of the advantage of using a non-uniform grid by a case study of a particular kind of polygons. Then we prove in Chapter 5 that the potential we will construct in the non-hierarchical way on the non-uniform grid is a distance. In Chapter 6 we present the non-hierarchical construction of a potential field on a non-uniform grid in C-space, including the computation of neighbours of a cell in the GET-Representation and some experimental results. In Chapter 7 we present the hierarchical construction of both the non-uniform grid in C-space and a potential field on it. We describe thoroughly the hierarchical aspects of the construction, and we concentrate on the sub-problem of enclosing swept volumes of robot links by superellipses and superellipsoids. In Chapter 8 we address ourselves to slight adaptations of the GET-representation and the potential field in case some obstacle has changed position slightly compared to the expected position when construction of GET-representation and potential field was started. Finally we sum up some open problems related to the topic in the ninth chapter.

Acknowledgements.

I would like to thank:

My parents for their support, but most of all for giving me the chance to study from primary school till university.

Fred Van Oystaeyen for that little push I needed to come to RISC and for telling his students (and thus me) that "pure mathematics" can be applied too,

Rudi Penne for teaching me the first basics of the fields of Robotics and Computational Geometry and for bringing me in contact with these fields of research, and

Herwig Mayr for his efforts in the initial contacts between RISC and me, and Franz Winkler and Franz Lichtenberger for the assistance in the struggle through administration.

Of course, most of all my supervisor at RISC, Sabine Stifter, for her scientific support and constant encouragement, and for organizing the environment for the financial support.

And Witold Jacak, for his willingness to be the second "Gutachter" of this thesis.

The head of RISC, Bruno Buchberger, for his visions and organizational talent, without whom RISC would never have been such a good place to work on a PhD Thesis.

All faculty members and students, and all scientists who visited RISC in the last few years, who all have contributed in some way or another to the development my work – by discussions, by posing questions, by giving lectures and presentations of their recent work, or just by their presence.

And because life is more than studying and working, it is only fair to thank also these people:

My real friends (that is, the ones who stayed in contact although I was and still am far away) just for being friends,

The musicians of the University Orchestra of Linz and of the Kleinmünchner Musikkapelle, who were so kind to let me join their group and let me have a wonderful, relaxing, and musically instructive time, and

The "RISC-members" with whom I spent time also outside the institute.

Contents

Chapter 1

Problem Specification.

In this chapter we first describe the Robot Motion Planning Problem as it is studied in the literature. Then we specify the problem we tackle in this thesis.

1.1 Problem Specification in Literature.

A formal specification of the three-dimensional generalized piano mover's problem $(O, P, p_{start}, p_{goal})$ can be given as follows:

- the obstacle set O consists of a finite set of (rational) polyhedra $O_1, ..., O_{n_1}$,
- the object P to be moved, consists of a finite set of (rational) polyhedra $P_1, ..., P_{n_2}$, which are freely linked at distinguished linkage vertices $v_1, ..., v_{n_3}$,
- p_{start} and p_{goal} are distinguished initial and final rational positions of P

(see [SHS87], page 273).

Motion planning in its most *basic form*, can be defined as finding a collision-free path for an object, or a collection of objects, among stationary objects ([Lat91a]). The objective of Robot Motion Planning is to automatically guide around a robot in an environment filled with obstacles and to choose forces to be applied when assembling objects ([Can88]). This basic form of the problem can be extended.

One could take into account the physical limitations of the robot. These are the so called *kinematic constraints*. Two types of constraints can be distinguished. The *holonomic constraints* can be expressed as equations which can be used to eliminate some parameters and thus reduce the dimension of configuration space (C-space). Examples of such constraints are the limitations in moves of links of robot arms (for example, they cannot rotate around completely, some limitations are present on the joints) or the limitations of moves of car-like robots (for example, they cannot move sidewards). The *non-holonomic constraints*, with which are much harder to deal, can be expressed as non-integrable equations. They involve velocity and do not reduce the dimension of the C-space, but of its velocity space. The working paper [Lau90] explains the mathematical tools for describing and solving non-holonomic motion planning for a multibody car system. A study of this kind of motion and control planning can be found in [Lau90], [Lau94], [WTS+92]. and [TMS93].

One could not only plan the position of the robot on every moment of time, but also the speed and/or acceleration in function of time (the *dynamics*, see [DX89], [SL91]). Interesting problems arise, for example if the end of the robot is to move in a straight line and fixed orientation at a particular line speed then not only must the kinematic trajectories be calculated but also the individual joint control must be able to compensate for the varying loads seen at each joint, otherwise some joints will end up lagging behind their demanded positions during motion, thus preventing the required coordination to achieve the straight line.

One could allow the *obstacles* to *move*. Since we want to plan the motion of the robot, it is reasonable to assume that the obstacles move only slightly. One also can assume that the environment is *uncertain* (see [HHS93] for instance for an analysis of unexpected contacts). In [SM88] and [FS88] and [FS89] the obstacles are moving and can be modelled as time depended objects.

One could also try to plan with *uncertainty*, that is, taking into account the fact that during execution of the path, the robot will not follow the planned path exactly. A robot arm usually oscillates around the planned route. Due to the weight of the robot (gravitation), the arm also tends to bow downwards, also depending on whether or not the robot (robot arm) is carrying an object.

One could ask for finding a path (an arbitrary path), the shortest path, the safest path or for a path that establishes a pay-off between the shortest and safest path. It would be nice in all cases to find a path whenever there exists one. However, sometimes one uses heuristic approaches that do not guarantee this, while one hopes such an approach will always produce a path in real applications and within a reasonably small execution time.

In [Can88], the author states that the most difficult planning problems arise in assembly, where clearances between parts are typically smaller than the robot's positioning and sensing accuracy. A distinction is made between *gross* and *fine* motion planning. Fine motion planning is concerned with the planning for contact between the robot and an obstacle, for instance when a gripper has to grap an object, while gross motion planning deals with moves that may be executed in a 'less careful' way, for example the robot might move slightly away from the planned path without colliding with an obstacle. Sensors of different types are sometimes attached to the robot in order to detect changes in the environment and as a safety measure.

Path planning can also be classified into two other categories: *on-line* path planning and *off-line* path planning. In the off-line case, more complex and time consuming algorithms can be used. Here planning for applications that are scheduled in the future can be done using simulation tools, while the robot in the meanwhile executes some other useful operation (see [CLZ93]). When planning on-line however, fast algorithms are needed, whenever the environment of the manipulator changes unforeseenly, for on-line updating of actual planning data in order to avoid wait states of the manipulator ([FW91]).

1.2 Problem Specification of Our Problem.

We address ourselves to the robot motion planning problem for a manipulator with many degrees of freedom (DOF), given a fixed initial configuration while any configuration is assumed to be a candidate for the goal configuration. We only ask for a path in the kinematic sense.

We propose an approach using a discrete potential function on a non-uniform grid that models the space of all Free Positions in the Configuration Space (C-space) of the robot.

We claim that this approach leeds to the construction of a path that has the property to be a pay-off between short path in C-space and safe path. It also combines certain extensions to what we earlier called the basic form of the robot motion planning problem.

Each point in the Configuration Space (C-space) corresponds to a configuration of the robot in its Workspace and thus the robot motion planning problem can be solved in two steps. In the first step the free space in C-space is computed. In the second step a path is planned for a point in C-space. It is very time consuming to compute the C-space of a robot with many DOF's exactly. For this reason we suggest to approximate the free space in C-space by a non-uniform grid. The robot motion planning problem we want to tackle here demands the complete space of free placements in C-space to be modelled. By doing so it becomes possible to compute for a given fixed initial position of the manipulator a path to any goal position.

Since the representation is done by a non-uniform grid, the size of the cells along the path gives us some information about the allowed error during the execution of the path. So also some information about uncertainty is incorporated in a natural way in the output of our approach. Since the path is a chain of cells, we think a trajectory planner can be designed that uses optimally the information hidden in such a representation of a path: for instance, the size of a cell depends on the distance to the nearest obstacle, in large cells joints can be moved fast and there is much space for the trajectory to be a safe one.

The C-space of a robot is usually high dimensional. The number of cells of a uniform grid on the C-space depends exponentially on the dimension. Using a non-uniform grid, fine nearby obstacles and coarse elsewhere, reduces the number of cells as much as possible. Small changes in the position of an obstacle will effect only the cells nearby, which might be divided into smaller cells or joined together.

We want to consider the problem of the construction of a non-uniform grid on C-space directly from information about the workspace.

We suggest to try out several possibilities for the potential field that can be constructed on the grid, each of those potential fields expressing different properties of the robot arm motions in the Workspace and leading to different planning strategies.

Comparing with the literature, our approach has some of the advantages of two different methods: On the one hand, the method with the wavefront algorithm and the Manhattan distance in [BL91] and [Lat91b] yields a complete solution (that is, always finds a path if there is one and reports failure otherwise). It is a discrete variant of the potential field approach, but seems to have the drawback of the size of the grid, since it grows exponentially with the dimension of the C-space. On the other hand, the potential field method has had reportedly good performance for high dimensional C-spaces. Potential fields also seem to react on slightly moving obstacles by a small perturbation which stabilizes after a short time. However they usually do not lead to complete solutions and can get into expensive computations under particular circumstances.

Our approach leads to a complete solution (if the space of Free Placements in C-space is bounded) and uses the concept of an attractive potential field. The repulsive field is superfluous because the combination of a non-uniform grid and the attractive field models the repulsive force. Hence, local minima in the overall potential field do not occur in our approach.

Finally we want to discuss the problem of adapting the grid and the potential field on it, when changes in the Workspace occurred. The construction of the non-uniform grid on C-space and the construction of the potential field can only be considered as a preprocessing step for the planning problem, when some changes in Workspace can be handled without recomputing completely the grid and the potential field.

The potential field approach on a grid leads in a natural way to parallel implementation and the computation of the potential can be done with a technique known as 'wavefront expansion' (see [Lat91b], [CLZ93]).

Good results are reported in [CLZ93] and [Ver90] using cells of different sizes for finding a path between two configurations and with similar techniques as ours.

Chapter 2

Solution Approaches.

In the first section of this chapter we describe briefly the approaches available in the literature that solve the robot motion planning problem. In the second and third section we give a deeper description of those approaches and while doing so we introduce the main terminology we will use for the description of our own approach. In the last section we present the outlines of our approach, emphasize what aspects of our method are original, and refer to our publications on the different subproblems.

2.1 Approaches in Literature.

The geometric problem, in the previous chapter also referred to as the problem in its most basic form, is solved adequately by Schwartz and Sharir ([SS87]), with a method of time complexity which is double-exponential in the size of the problem and by Canny ([Can88]), with a method of time complexity which is only exponential in the size of the problem.

An interesting introductory article on Motion Planning, giving a nice overview on recent developments in this field, is [Lat91a]. The author states that in essence there are two computational approaches to solve this problem: the Connectivity Representation approach and the Potential Field approach.

The *Connectivity Representation* approach consists of precomputing the connectivity of the space of all positions that are safe for the robot in the form of a non-directed graph, the so called connectivity graph, and then searching in this graph for a path. The methods of Schwartz and Sharir and of Canny are examples of this approach.

Schwartz and Sharir use the exact cell decomposition method (described in [SS87]). Here the space of free positions in the configuration space is divided into a finite collection of semi-algebraic cells. These cells are computed by first finding the projection of the cells in a lower dimensional space and then lifting these projections again to the higher dimension. This can be done recursively, that is, first going down from dimension m (usually bigger than 3) to dimension 1 and then lifting again. The method uses the well-known result in [Col75], for deciding the satisfiability of Tarski sentences.

In his Ph. D. Thesis ([Can88]), Canny introduced the so called roadmap method. The roadmap algorithm he presented, finds all components of a high-dimensional set S by computing its 1-dimensional skeleton (or roadmap) $R(S)$. The roadmap has the property being connected within each component of the original set S. To find a path between two arbitrary points p_1 and p_2 in S, one only has to find a path from each of these points to two points r_1 and r_2 nearby and on the roadmap and find a path along the skeleton. It can already be decided whether a path exists or not, when the two points r_1 and r_2 are found. If they do not lie in the same connected component, there will not be a path between p_1 and p_2 either. The roadmap is constructed by reducing the dimension of the problem by 1, taking slices of S by intersecting S with a "moving" hyperplane. Extremal points in the slices trace out so called silhouette curves while the hyperplane moves. The silhouette curves form the essential part of the roadmap. On some critical moments however, one has to make sure that the roadmap is connected in a 1-1 correspondence with the connectivity of the original set S by adding some appropriate links. Doing so, the problem is reduced to several subproblems of searching paths in the slices of S, which have dimension 1 less than S.

In [OŠ94] the authors use learning techniques to construct a probabilistic roadmap in C-space. Approaches based on Voronoi diagrams are also in the category of the Connectivity Representation approach.

In [FL91] the changes in a Voronoi Diagram for points in the plane are studied when the points move.

The *Potential Field* approach makes the configuration space (the set of all possible configurations, whereas a configuration is a specification of the position of the robot relative to some fixed coordinate system) discrete, by turning it into a fine regular grid, and searches this grid for a free path. The search is then guided by a heuristic that always can be seen as a potential field.

In the potential field approach, in the configuration space the robot, represented as a point, is attracted from start position to the goal position by an attractive potential field, whereas the obstacles are modelled as repulsive potentials. A path is found by computing the gradients of the total potential and following the negative trend from the well towards the minimum point. Local minima are causing some inefficiency or failure of this approach. One can try to define a function without any local minima (or with very few) or one can find ways to escape out of the local minima when the robot gets trapped in them. When this approach is used as a heuristic, usually one cannot guarantee that a path will be found always if there exists one (since such a path might have to move in a direction opposite to the one "downwards").

The potential field approach leads in a natural way to massively parallel implementation, using some kind of a grid to model the C-space.

The book on Robot Motion Planning by J. -C. Latombe ([Lat91b]) gives a thorough overview of the "state of the art" of robot motion planning. Another overview can be found in [HA92].

2.2 Configuration Space.

2.2.1 Position and Orientation.

Let R denote a set (the robot) and A_1,...,A_k a number of obstacles, pairwise disjoint. Let us introduce a coordinate system CS_W for the workspace W of R, with origin o a specific (arbitrary) point of R. A *configuration* of R is defined as the image of R by a rigid body transformation t (also called positive isometry, i.e. preserving distance and orientation) from the workspace to some other space, which can be described as (o, q), where $q = t(o)$. The set of all such transformations is called the configuration space (also written as "C-space", we will also denote the space by $C - space_R$). The dimension of the C-space is equal to the number of degrees of freedom (DOF) of R in the workspace. A position p (or configuration) of R now corresponds to a point in C-space. Indeed, a point of C-space gives us the transformation of the reference point of R in the origin of CS_W to the position of the reference point of R when R is in position p. We will write R_q for the image of R under transformation t where $t(o) = q$. When one considers a coordinate system CS_R moving with R and identical to CS_W when the reference point of R is at the origin of CS_W, the C-space can also be seen as the set of all specifications of the position and orientation of CS_R with respect to CS_W.

A configuration q can be represented bijectively, depending on the choice of CS_R and CS_W, as a pair (t, r), where t determines the position of the origin of CS_R with respect to CS_W and where r determines the orientation of the axes of CS_R with respect to CS_W. Now t can be considered as the N-dimensional vector being the coordinates of the origin of CS_R with respect to CS_W, where N is the dimension of W, and r as the $N \times N$ matrix whose columns are the components of the unit vectors along the axes of CS_R with respect to CS_W. If we denote $SO(N)$ the special orthogonal group of the $N \times N$ matrices, then we have $r \in SO(N)$. In this way, the C-space can be embedded in $\mathbb{R}^M$, with $M = N + N.N$. When we denote M_C for the image of $C - space_R$ to the embedding in $\mathbb{R}^M$, we have:

$$M_C = \mathbb{R}^N \times SO(N) \subset \mathbb{R}^M.$$

It also makes the C-space fitting into the theory of Algebraic Topology. One can show that the C-space is a manifold.

In principle, once defined in C-space, all motion planning problems are equivalent. Also, the C-space has been used to solve some motion planning problems (see for instance [LP81], [KS85], [KS86], [Lat91a], [KL94]).

2.2.2 Joint Parameters.

Another way to look at the concept of C-space, also called Joint Space, pops up when we consider Robot Arms. From a theoretical point of view we can restrict ourselves to rotational and translational joints because all other joints can be seen as several of those two kinds in one.

One parameter expresses the relative position of a link to the previous link in the chain and determines the position of the joint. Each parameter gives rise to a dimension in the C-space of the arm. In this way the configuration of a robot arm is described completely by a point in its C-space. In the following we will denote the Joint Space also by $C - space_R$.

As a space it does not matter whether we deal with a C-space of a car-like robot or with a Joint Space of a robot arm. In the case of a car-like robot often non-holonomic constraints have to be considered, and hence the tangent space to the C-space has to be taken into account. Therefore we restrict ourselves to robot arms with rotational and translational joints.

2.2.3 C-Obstacles

Constraints placed on the motions of a robot arm like joint limits and obstacles in the Workspace are generally hypersurfaces in the Configuration Space.

The $C - space_R$-obstacle of A_i, the expanded obstacle K_i, is defined by:

$$K_i = CO_R(A_i) = \{x \in C - space_R | R_x \cap A \neq \emptyset\}.$$

Obviously, x represents a safe position (a so called free position) for R if and only if x is not in $CO_R(A_i)$ for all obstacles A_i. Denote

$$C_{free} = C - space_R \setminus \bigcup_j CO_R(A_i)$$

for the set of all free positions of R.

A method for the construction of C-space obstacles in a 2-dimensional world is given in [LPW79]. Another method for the computation of obstacle boundaries in 2 and 3 dimensional C-space of robot arms is presented in [NB91] and [MB88]. In [HA92] it is stated that there exist seven basic ways to compute C-space Obstacles: point evaluation, Minkowski difference, boundary equation, needle, sweep volume, template, and a Jacobian-based method.

2.3 Potential Fields.

In the potential field approach, in the configuration space the robot, represented as a point, is attracted from start position to the goal position by an attractive potential field, whereas the obstacles are modelled as repulsive potentials. A path is found by computing the gradients of the total potential and following the negative trend from the well towards the minimum point. Local minima are causing some inefficiency or failure of this approach. One can try to define a function without any local minima (or with very few) or one can find ways to escape out of the local minima when the robot gets trapped in them. When this approach is used as a heuristic, usually one cannot guarantee that a path will be found always if there exists one (since such a path might have to move in a direction opposite to the one "downwards").

The potential field approach leads in a natural way to massively parallel implementation, using some kind of a grid to model the C-space.

2.3.1 The Casual Potential Field.

In general, the potential field approach treats the robot, represented as a point in C-space, as a particle under the influence of a potential field U whose local variations are expected to represent the structure of the free-space. The force $\vec{F}(q) = -\vec{\nabla} U(q)$ induced by the potential field is considered as the most promising direction of motion at every place at any time. The potential field U is considered as the sum of an attractive potential, pulling the robot towards the goal position, and a repulsive potential, pushing the robot away from the obstacles.

Most of the following part of this section comes down to a summary of the overview as it is given in [Lat91b].

Settings

In the following we will use the notation C for the C-space of the robot.

Given a differential potential function $U : C_{free} \to \mathbb{R}$, the artificial forces

$$\vec{F}(q) = -\vec{\nabla}U(q)$$

in C are induced by U, where $\vec{\nabla}U(q)$ stands for the gradient vector of U at q.

The potential U is constructed as the sum of two potentials U_{att} and U_{rep}:

$$U = U_{att} + U_{rep},$$

where U_{att} is the attraction potential associated with q_{goal} and independent from the C-obstacles, and U_{rep} the repulsive potential associated with the C-obstacle region.

Let us assume for simplicity that the C-space is the whole space $\mathbb{R}^N$, for some N.

Attractive Potential

The attractive potential can be defined as a parabolic well:

$$U_{att}(q) = \frac{1}{2}\xi\rho^2_{goal}(q),$$

where ξ is a positive scaling factor and ρ_{goal} denotes the Euclidean distance between q and q_{goal} (if the C-space is $\mathbb{R}^N$ for some N, say 2 or 3). Obviously, the implied attractive force $\vec{F}_{att}(q)$ is then

$$\vec{F}_{att}(q) = -\xi(q - q_{goal}).$$

This force converges linearly towards 0 when the robot's configuration gets closer to q_{goal}. The force also tends towards infinity when $\rho_{goal} \to \infty$.

In order to get rid of this last observation, the attractive potential can also be defined as a conic well:

$$U_{att}(q) = \frac{1}{2}\xi\rho_{goal}(q) = -\xi\frac{(q - q_{goal})}{||q - q_{goal}||}.$$

When $q \to q_{goal}$, the conic-well does not tend toward 0, however.

Repulsive Potential

The repulsive potential is introduced in order to create a "potential barrier" around the C-obstacle region for the reference point of the robot in C-space. However, it would be nice if this repulsive field does not have any influence on the behaviour of the robot on positions which are "sufficiently far away" from the obstacles.

This can, for example, be established by defining U_{rep} as follows: decompose the C-obstacle region $\mathcal{CB}$ into (possibly overlapping) convex components $\mathcal{CB}_k$, $k = 1, ..., r$, and associate to each component a repulsive potential

$$U_{rep,k}(q) = \begin{cases} \frac{1}{2}\eta(\frac{1}{\rho_k(q)} - \frac{1}{\rho_0})^2 & \text{if } \rho_k(q) \leq \rho_0, \\ 0 & \text{if } \rho_k(q) > \rho_0, \end{cases}$$

where η is a positive scaling factor, $\rho_k(q)$ is the Euclidean distance from point q to the C-obstacle $\mathcal{CB}_k$, i.e.

$$\rho_k(q) = \min_{q' \in \mathcal{CB}_k} ||q - q'||,$$

and ρ_0 is called the distance of influence of the C-obstacle region, and then the total repulsive potential field U_{rep} is the sum of these, i.e.:

$$U_{rep}(q) = \sum_{k=1}^{r} U_{rep,k}(q).$$

The artificial repulsive force is then

$$\vec{F}_{rep}(q) = \sum_{k=1}^{r} \vec{F}_{rep,k}(q)$$

where

$$\vec{F}_{rep,k}(q) = -\vec{\nabla} U_{rep,k}(q).$$

A drawback of decomposing the C-obstacle region is that several small components close to each other arbitrarily produce a combined repulsive force bigger than the one produced by a single bigger obstacle. A possible solution is to weight each individual potential according to the size of the obstacle generating it.

Of course, η and ρ_0 might also depend on the obstacle. In fact, when an obstacle is close to q_{goal}, the value of ρ_0 should be not too big in order not to prevent the robot to attain the goal position. When an obstacle is more threatening for the robot than the others, values for η and ρ_0 of this obstacle should be chosen larger.

For More General C-spaces

In case the C-space is of the form $C = \mathbb{R}^N \times SO(N)$, for some N, some extensions to the notions above have to be made.

- The gradient $\vec{\nabla}$ has to be defined on the manifold, using charts of C.
- The Euclidean distance in the previous, is to be replaced by a metric d on C, i.e. $\rho(q) = d(q, q_{goal})$.

For more details, the reader is referred to [Lat91b], pp. 303-305.

Unfortunately, the computation of $\rho(q)$ and $\vec{\nabla}(q)$ for each q is difficult.

2.3.2 Path Planning, Using Potential Fields.

Potential fields are of course just a model in which planning is done, the real planning has to use these potentials for actual path finding. Three simple path planning techniques are available for the potential field approach.

Depth-First Planning

This technique constructs a path as the product of successive path segments starting at the initial point q_{start}. Each segment is oriented along the negated gradient of the potential function computed at the configuration attained by the previous segment. The amplitude of the segment is chosen such that the segment lies entirely in free space. This approach only uses the potential field as such.

Let q_i and q_{i+1} be the origin resp. end extremities of the i-th segment in the path. Locally (in a chart of the C-space manifold, to be a little more precise), let $x_j(q_k)$, $j = 1, .., m$, denote the m coordinates of q_k, for arbitrary k, in the chart. Then

$$x_j(q_{i+1}) = x_j(q_k) + \delta_i \vec{t_j}(q_i),$$

with

$$\vec{t_j}(q_i) = \frac{\vec{F}(q_i)}{||\vec{F}(q_i)||}$$

and considered in the orthonormal basis on the tangent space induced by the chart:

$$\vec{F} = -\vec{\nabla} U$$

(for exact notation and details, see [Lat91b], p. 311), and δ_i denoting the length of the i-th increment (in Euclidean metric on $\mathbb{R}^m$). A good choice of the value of δ_i (small enough to be sure no collision with any of the obstacles occurs) gives us a path. It follows the steepest decent of the potential function and therefore it might get stuck in local minima.

Best-First Planning

For this technique, one first places a fine regular grid over the configuration space. It is assumed that q_{start} and q_{goal} are configurations in this grid, two neighbours in free space can be connected always by a straight line also in free space and the grid is bounded and forms a rectangloid.

First iteratively a tree T is constructed whose nodes are configurations in the grid and whose root is q_{start}. At each iterative step, the algorithm examines the neighbours of the leaf of T that has the smallest potential value, retains the neighbours not already in T at which the potential function is smaller than some large treshold M (so that one doesn't consider going up to some place where the potential function becomes infinite), and installs the retained neighbours in T as successors of the currently considered leaf.

The algorithm terminates when q_{goal} is reached (success) or when all configurations accessible from q_{init} have been fully explored (failure). Each node has a pointer to its parent node, so if q_{goal} is attained, the path can be generated by tracing the pointers from q_{goal} to q_{init}.

This procedure follows a discrete approximation of the negated gradient of the potential function until it reaches a local minimum. It deals with local minima by filling them up until a saddle point is attained.

The approach is reported to be practical only when the dimension of the C-space is small, say less than 5. The filling up of local minima is becoming too expensive for higher dimensions.

Variational Planning

In this approach a functional J of a path τ is constructed and then optimized over all paths. The optimization of the functional is done by integrating the potential along a complete path between the starting and the goal position, using variational calculus.

With U a potential function, constructed for path planning, J could be defined as

$$J(\tau) = \int_0^1 U(\tau(s))ds + \int_0^1 ||\frac{d\vec{r}}{ds}||ds.$$

The first term in J is meant to make the optimization produce a free path. The second term is optional and aimed to produce shorter paths. Other criteria can be encoded as well.

In [SS88b] this approach is used for finding a path which is a pay-off between the safest (more precise, a center line path, staying "in the middle" between the obstacles: this might not be the same as the safest path) and the shortest path. Both travelling distance and robot safety are included in the cost function.

Since this method essentially consists of minimizing a functional J by following its negative gradient, it can get stuck in local minima, just like the depth-first approach. For large dimensions the optimization can be quite costly. The advantage is the possibility to encode extra criteria in J.

2.3.3 The Global Navigation Function.

Fluid Dynamics

Closely related to the potential field approach, is the fluid dynamics metaphor ([KD90]). When a fluid is continuously pomped through a system of conduits, the fluid particles move on trajectories which constitute the shortest path connecting the pump and the outlet. Also when one of the pipes is suddenly closed, thus interrupting the flow, the fluid particles automatically adapt to this change in topology and find a new way. The solution for the motion planning problem (2-D) presented in [KD90] consists in representing the factory's floor plan as a pipe-system with a fluid pump installed at the robot's initial position and an outlet at its destination. The robot itself is simulated as a fluid particle (a point). From some Physics-analysis of the model they deduct that a change in the obstacles should be modelled as waves propagating through the system. When a change is encountered, it is modelled automatically. The initial setting of the floor plan is modelled by prescribing boundary conditions on the equations that describe the fluid for each wall present.

The authors have implemented this model on a massively parallel array processor. The factory floor is represented as a grid of space cells each containing some fluid or a piece of wall. All cells compute in parallel the solution of the wave equation. After some initial time, the solution becomes stable and the moves of particles can start. There is some local communication, exchanging some local information. The implementation done by the authors of the report is said to evolve in real time and to adapt quickly to changes in topology.

Vortex Fields

Another approach to prevent local minima, is the use of vortex fields instead of repulsive potentials. In [DMO91] the authors explain how the vortex fields force the robot (reference point in C-space) to turn around the obstacles. Repulsive potentials make sure that no collision will appear, while vortex fields guide the robot around the obstacles and can be considered to be similar to a velocity field in fluid dynamics. For simplicity reasons, in [DMO91] only planning in 2-dimensional space is described. Suppose that one has a point obstacle located in the origin of the Euclidean xy-coordinate system. The vortex field for this point obstacle is then defined as:

$$V(p) = \begin{bmatrix} V(x) \\ V(y) \end{bmatrix} = \begin{bmatrix} \frac{-y}{x^2+y^2} \\ \frac{x}{x^2+y^2} \end{bmatrix}.$$

This field assigns in each point p a vector which is tangent to the circle with center in the origin, is oriented in counterclockwise direction, and has modulus inversely proportional to the square of its distance to the point obstacle, so to rapidly decrease as p "moves away" from it. Adding such vortex fields to the attractive potential field, will establish a total potential without local minima, and the vortex fields will guarantee obstacle avoidance.

Three remarks were made by the authors. First, to stop the robot turning around the obstacle for ever, a vortex remover procedure is introduced. The angle between the direction of the vortex vector and the attractive vector becomes zero when the obstacle is passed. Second, the selection of the direction of vortex rotation has to be done because it influences the length of the path and in some cases (when moving a manipulator with fixed basis is planned) a bad choice can even lead to not finding an existing path. In the latter case one can choose for the direction which generates moves towards the robot base. Third, in the case of articulated manipulators, link interaction has to be taken into account. The links will also feel an influence by the vortex field.

L^1-distance

Another navigation function is described in [Lat91b], using a so called wavefront technique. In fact, the C-space is modelled as a uniform grid, where some cells are "filled" by the obstacle and where the "empty" cells represent a free position in C-space. At initialization time, the "filled" cells are labeled with "1" and the others with "0". It is assumed that q_{start} and q_{goal} both are in an "empty" cell. The navigation function used here is the L^1-distance, also called Manhattan distance, to the cell with goal position in it. This distance function U can easily be computed by a so called "wavefront expansion" technique. At first, U is set to 0 at q_{goal}. In the next step, in all neighbouring empty cells U is set to 1. Then recursively all empty neighbours of cells with $U = d$, except for those which already have a value, will get value $d + 1$ for U. When q_{start} is reached, a path can be planned by following the decent of distance.

A drawback of this method is that the path always grazes the obstacles, like the visibility graph approach in the pure geometric version of the planning problem. To improve this, a skeleton can be constructed first. This skeleton S has one dimension less than the C-space and is a subset of the set of free positions. Starting from the boundary of this free space, the L^1-distance from each position in free space to the boundary is computed using the wavefront approach. Then S is the set of positions where these waves meet, i.e. in the "middle" between the obstacles. This looks somewhat like the Voronoi diagram approach in the pure geometric problem. Obviously, the complexity of the L^1-wavefront approach is linear in the number of empty cells in the C-space. Therefore in the higher dimensional C-spaces this approach is used with some modification. One chooses some points in the workspace of the manipulator. With each of these points a potential field is associated in the set of free placements in the workspace. Then the potential function U is computed in the C-space in those cells where it seems reasonable to have U (the number of such cells is much smaller than the total number of cells), with an arbitration function depending on the values of the potential fields in the workspace. Since the workspace is 2 or 3 dimensional, it is reasonably easy to compute the potential fields. Since the number of cells in C-space is smaller, planning can be done more efficiently also in high dimensional C-spaces.

We refer the reader to [Bor86] for a thorough study of different possible definitions of *neighbours* for the construction of the Manhattan distance on the uniform grid in 2-dimensional space by the wavefront approach (at least, one could interpret the paper in such a way). The author compares the geometric behaviour of the expansion of the area of cells to which a distance value is assigned.

The wavefront expansion algorithm above has been implemented in a way that reduces the memory needed for the representation of C-space and some reduction of the complexity of the problem using a "hierarchical multilevel discretisation" and a "multisource navigation function", as a part of a simulation system, so is reported in [CLLZ93] and [CZ93]. Doing a step in C-space along one of the directions of a grid means that the robot moves along only one of the DOF's at a time (e.g. only one joint of a robot arm changes position at each moment), as is pointed out in [CLZ93]. Visiting smaller cells will result in a higher number of changes in moves of the point in C-space and thus in the follow-up of joints that are moving.

Another article about a similar approach to the robot motion planning problem is found in [Ver90]. There the author models robot and obstacles by *bubbles* and does a collision check for bubbles in two large blocks (or cells) in C-space (namely one cell containing the initial configuration, and one containing the goal configuration). Both cells are treated in the same way. When a first free cell is found, free cells adjacent to this cell are searched and get distance value 1. When a cell seems to be too big to be free, it is split. These cells are ordered heuristically for further investigation. A path is found when the two lists of free cells meet.

2.4 Our Approach.

We choose to split the motion planning problem into two subproblems: the construction of a model for the C-space and planning a path for a point in C-space.

Since exact modelling of C-space obstacles for highly dimensional C-spaces is computationally expensive, in our approach we model them with a non-uniform grid. Near the obstacles small cells are used for adequate modelling, while in areas without obstacles the cells are large. The 2^k-tree is a well-known data structure, used extensively in geometric modelling. Such a tree can be represented by a list with binary entries (one of the two lists out of which the GET Representation consists).

We counted out for a set of polygons with a particular shape what the gain in number of cells is when using cells of different sizes. This has been reported in [VG95b]

The path we get as output is a list of adjacent cells connecting the cell comprising the goal configuration to the cell comprising the initial configuration. Thus the path is a corridor in C-space.

Since potential field approaches have reportedly been successful in planning paths, we choose to use such an approach for the second subproblem.

The cells of the non-uniform grid, used in our planning algorithm, can be classified in two categories: *white cells* and *black cells*. *White cells* are those cells in which all configurations correspond to free positions in workspace; *black cells* are cells in which some (or all) configurations are forbidden positions in workspace (physically not feasible for the robot, or colliding with an obstacle). We colour all cells containing a forbidden configuration black because we want to be sure that the path we get as output is safe regardless which part of the cells on the path are really visited during the execution of the path. We have to consider the computation of the GET Representation of such a non-uniform grid on C-space, when given a robot arm and a number of obstacles in the workspace of the robot. We found a hierarchical construction of the GET Representation for the grid on C-space (reported on first in [VG95e]). For this we model the robot links as in [VG95a] for the first link, or the swept volumes – swept over some range for each of the joint parameters – of the other links by superellipses and superellipsoids as in [VG96a]. In the latter case we can use a collision detector for each cell in the grid in order to decide if the cell contains a forbidden configuration. If it does, we can still refine the cell by dividing it into smaller cells. In the second list of binary entries of the GET Representation each entry corresponds to a cell (a leaf of the 2^k-tree) and describes if the cell is white or black.

When we model one link at a time, we only have to consider the joint parameters for joints between the currently considered link and the base support of the robot arm. Starting with the link closest to the base support, we can construct the grid by one dimension at a time. If for the i th link a i-dimensional cell contains a forbidden cell, then for any value of the joint parameters not yet considered this cell will be black. On the next hierarchical step we only consider the i-dimensional cells that were not coloured black and test again for the $i + 1$ th link. On this approach was reported during the 4 th International Workshop on Robotics in Alpe-Adria Region ([VG95c]) and in [VG95d].

The model of the C-space allows small changes in the position of obstacles (that is, when an obstacle is not positioned exactly on the spot we assumed it to be in in the Workspace). In this way the computation of the free space in C-space can be considered as a pre-processing step that can be executed in reasonable time.

For our approach we want to put an "attractive potential field" on the non-uniform grid on C-space, using the wavefront expansion technique as it is done in [BL91] on a uniform grid. To each cell we therefore assign a u-value (a natural number, a value of the potential field). It suffices to make a list of natural numbers organized like the second list of the GET Representation.

To each cell c of the grid, we assign a value of the potential field that expresses the number of cells that have to be visited in the shortest sequence of cells between c and the goal position. The path from the start to the goal position follows the descent of this potential field. The path is a good pay-off between a safe and a short path because the path preferably visits large cells which are far away from the obstacles and will visit the lowest number of cells possible for a path. Moreover, since the path goes along large cells and a cell corresponds to a move of one joint at a time, this joint can move over a long distance while the large cell is visited.

For the wavefront expansion we have to find all neighbouring cells of a cell. We have observed that the algorithms given in [Gar82b] can be generalized to any 2^k-tree and can be translated in simple algorithms for manipulating the GET-Rep (see [VG94a]). It describes the non-hierarchical construction of a potential field.

It is also possible to construct a potential field hierarchically, that is, when the grid is constructed up to the j th dimension we can model the last $k-j-1$ links by the swept volume with respect to the last $k-j-1$ joints and determine which cells are certainly white. Then we can compute a potential field with the wavefront technique but with different increments for different directions. When we consider an extra link to improve the grid, we can extend this potential field to the new white area or newly reachable free cells. This has been described for the first time in [VG95e], [VG95d], and [VG95c].

The hierarchically constructed potential field additionally expresses the preference to move the joints of the manipulator closest to its base support.

Similarly, the GET Representation and this potential field can be adapted in case some slight changes in the Workspace occured. We also described these adaptations in [VG95e] and [VG95d].

Our approach delivers a complete solution (that is, always finds a path if there is one and reports failure otherwise if the space is bounded). Since the representation is done by a non-uniform grid, the size of the cells along the path gives us some information about the allowed error during the execution of the path. So also information for planning with uncertainty is incorporated in a natural way in this approach.

The potential field approach on a grid leads in a natural way to parallel implementation and the computation of the potential can be done with a technique known as "wavefront expansion" (see [Lat91b], [CLZ93]).

Good results are reported in [CLZ93] and [Ver90] using cells of different sizes for finding a path between two configurations and with similar techniques as ours. Our sequential implementation of the non-hierarchical algorithm for the subproblem of path planning indicates that our approach is satisfactory.

Some aspects of the approach in [Ver90] are revisited in our work. The author of that article came to his planning approach considering metrics, while we came up with similar ideas when looking at the uniform grid and wavefront approach in [Lat91b]. Essential differences of our approach in respect to the method presented in [Ver90] are the following:

- we construct a total model of C-space and solve the motion planning problem for any goal position and a give initial position,
- the GET Representation we use (due to Klaus Burger) is compact and computation of neighbouring cells is easy,
- we observed the fact that a repulsive potential is superfluous when we use a non-uniform grid and some particular attractive potential functions,
- we need a fast collision checker (for instance, using the Roider method – see [Sti92a]) and a good approximation of the swept volumes of robot links (using superellipsoids),
- we worked out a hierarchical construction of the grid and of a potential field on it.

Chapter 3

Some More Literature

During our search through literature for publications closely related to the topic of the thesis, we also came accross other papers related to the field of robot motion planning. We here give a list of publications that came to our attention during literature search.

We have classified them by problem and approach, and doing so we came to a historical overview of the field as well as to a list of elementary references. However we did not attempt to make the list into a complete survey of the field of robot motion planning.

Some of the references are related more to the topic of this thesis than other and therefore are commented on in the list below. The other references are only briefly commented on or not at all.

Books on Robotics

- For an overview on Robotics as a field, books like [Cra86] and [Cri85] are very suitable. One of the most important references in that respect certainly is [Pau81].
- Concentrating on algorithmic and geometric aspects as well as on the complexity of algorithms concerning Robot Motion Planning the books [SHS87] and [SY87] are good references.
- The book [BHJ+82] is dedicated to dynamics, planning and control.
- A tutorial overview of the field of robot motion planning certainly is [Lat91b].

Survey Papers

- [Hop86] discusses robotics as a field in science and research.
- [SS88a]: on motion planning and related geometric algorithms.

Techniques Introduced

- Grid: A first discrete representation in the field has been used in [Udu77].
- C-space: The concept of Configuration Space was first introduced in [LPW79] and further discussed in [LP81].
- Potential fields: In [Tho84] a potential field based on distance has been used for an optimal path for a circular robot in the plane. Other early papers where a potential field approach is used are the independently developed methods in [Kha86] and [MA84], [PV84].

Collision detection and obstacle avoidance

- [Udu77] combines planning in empty space with collision detection and obstacle avoidance. Hierarchical decomposition is used.
- [WH92]: optimal trajectory planning with collision detection by geometric reasoning and geometric computation, and obstacle avoidance.

- [Kha86]: obstacle avoidance using the artificial potential field concept.
- [Pou91]: solving the inverse kinematics problem using a mathematical model for avoiding obstacles.

Planning in Workspace

- [NNA89] represents the 2D workspace of a robot by a quadtree and uses closest points search and a distance measure for the motion planning.

Exact and Theoretical Approaches

- [ABF88]: an exact computation of the set of free placements.
- [KS86]: attempts have been made towards constructing a unified descriptive and computational framework for path planning, as stated and contributed to in [DMT91].

Potential Field Approach

- [Sat93]: solving Laplace's equation, it's solution does not have local minima.
- [War88]: global planning, reducing the presence of local minima.
- [RK88]: a theoretical treatment of navigation functions without local minima.
- [KV88] presents a potential field that closely models obstacles and does not generate local minima in the workspace of the manipulator.

Other Heuristic Approaches

- [IT90]: a heuristic search through a graph, on an implicitly defined C-space.

Interactive Path Planning

- [Shi89]
- [DHX91]: off-line planning, both kinematic and dynamic.

CAD Systems

- [SdPD88]: Object offset (shrinkage or expansion) is realized to support many applications in CAD/CAM. Offsetting techniques have been used in "shrinkage" of obstacles in workspace to C-space obstacles, as in the piano movers problem.

Following the faces of objects in path planning (contact is wished)

- [WCWC87]
- [BK90]: Generating the boundary of C-space obstacles arising from the translational motion of curved objects among curved obstacles, where the boundaries of the curved objects and obstacles are patches of algebraic surfaces.

Computing C-space Obstacles

- [Bro89]

Friction modelled in C-space

- [Erd94]

Randomization of actions

- [Erd92]
- [Ove92b]: random network of possible motions.

Task Planning

- [AFB91]
- [Sch92]

Artificial Intelligence

- [ZM88]

Inverse Kinematics

- [LPF89]
- [PL89]: theoretical study of the generic singularities of the manipulator Jacobian.

Overall Structure

- [AM91] gives a description of a robotics architecture.

Pick and Place

- [LP81]: planning a move of an arm with an object that was picked up.

Travelling Salesman Problem

- [PCH91]: Planning the visit of several points, for instance in spot welding on a car body, taking in account the joint limits.

Planning with non-holonomic constraints

- In [JC89] an algorithm is presented for planning paths for a mobile robot subject to non-holonomic constraints. The technique is to define a set of canonical trajectories with safety constraints. It is shown that considering only these trajectories is sufficient to find a path if one exists.

Graphics and image processing, CAD, data structures

- [Gar82b]
- [Gar82a]
- [ST85]
- [GA84]
- [CCV85]

Computational Geometry

- [Ove92a]: Point Location.
- [NP92]: Intersection.
- Some books on the use of Computational Geometry in Robotics are [Boi88], [PS85], and [Ede87].

At RISC, with their abstract

- [Man93]: Many problems in geometric modeling and robotics require solving systems of polynomial equations. We give an overview of these problems and a guide to the relevant literature, having in mind the symbolic methods for solving polynomial systems.

- [Sti87]: an annotated bibliography on the programming of robot manipulators.

- [Sti93]: For quite a few years, robotics and, more generally, softautomation has become one of the main research topics at RISC-Linz. Conditional on the fields of expertise of RISC-Linz, the concentration is on geometric problems like collision detection, collision free path planning, path tracking, forward and inverse kinematics, and software technological aspects as they appear e.g. in simulation systems, supervision systems, and control systems. Both, basic research as well as applications within industrial cooperations are studied.

- [Pfa92]: Some selected applications of mathematical modeling methods in the AI field robotics are presented in survey style by examples of geometric reasoning, topological reasoning and so-called fibered logical spaces for logical reasoning in robotics. The main perspective is on interaction and combination of different fields and methods from symbolic mathematical computation and AI and the mutual stimulation given by the various disciplines.

Recent work

- The paper [KL94] presents a path planning approach for robots with many DOF's operating in known static environments. The approach consists of a preprocessing and a planning stage. Preprocessing, which is only done once for a given environment, generates a network of randomly, but properly selected, collision-free configurations (nodes) and interconnections. Planning then connects any given initial and final configurations of the robot to two nodes of the network and computes a path through the network between these two nodes.

- [GZ95]: The n-dimensional problem of planning collision-free motions for an n-link manipulator is decomposed into a sequence of smaller m-dimensional sub-problems, with backtracking, each of which correspond to planning the motion of $m-1$ links. Numeric potential fields over a bitmap-based representation of sub-spaces are used for the m-dimensional sub-problems.

- [GMN95]: A gross motion planner, with geometric model, approximating obstacles, and reasoning techniques, works in a not completely computed C-space and reduces the planning of a trajectory avoiding fixed obstacles to a planer geometric reasoning problem.

- [QCM95]: The claim is made that many path planning problems can be solved by a mixture of global and local planning. They propose to use an approximation technique based on efficient distance computations for the construction of a Generalized Voronoi Diagram in order to tackle the global motion planning, and the new concept of *virtual springs* and complex geometric models for manipulators and workspace for local planning. This leads to an incomplete algorithm that however can be applied to practical and geometrically more involved problems.

Chapter 4

Counting Cells in a Grid

4.1 Introduction.

Considering the use of grids in the Robot Motion Planning Problem (e.g. [BL91], [CLZ93], [VG94a], [VG95e], [FT88]), we are interested in finding out how much better it is to use a non-uniform grid instead of a uniform grid. This chapter is addressing this question.

In Section 2 some definitions are given, needed throughout this chapter. Then we discuss the number of cells in uniform and non-uniform grids in Sections 3 and 4. In Section 5 we give a first example. Section 6 considers some properties of quantities introduced in the preceding sections and relations between the introduced parameters are studied. In Section 7 we consider for the parameters some values which are corresponding to some practical applications carried out over the last few years at the Research Institute for Symbolic Computation (as also discussed in [VG95e]). Finally in Section 8 we consider a particular family of polygons in 2-dimensional space. We compute the number of cells in non-uniform grids modelling these polygons and compare for some particular polygons these amounts with the number of cells needed in a uniform grid.

4.2 Preparations.

In the following we will use some terminology which we now define. We will consider both some uniform and non-uniform grids on a space of dimension $k \geq 1$.

A grid consists of k-dimensional *cubes*. We only consider grids that can be represented by a 2^k-tree. These grids can be constructed in the following way. We start with an initial k-dimensional cube. This cube can be subdivided in 2^k smaller cubes of equal size (in each dimension edges of the initial cube are cut in half). Any of these smaller cubes can be subdivided again in 2^k even smaller cubes of equal size, and so on. Each node of the corresponding 2^k-tree represents a cube. The initial cube is represented by the root of the tree, the children of node N represent the cubes of smaller size in which the cube corresponding to node N was subdivided. The leaves of the tree correspond to cubes that were not subdivided. We call these cubes *cells*. When all cells are of equal size, we call the grid a *uniform* grid. In the other case we call the grid a *non-uniform* grid.

When a cube is represented by a node on height s in the 2^k-tree, we call the cube a *cube of size s*. If the cube is a cell, we call it a *cell of size s*.

Cells C_1 and C_2 in a uniform grid are called *neighbours* when C_1 has a face in common with C_2. Cells C_1 and C_2 in a non-uniform grid are called *neighbours* when a face of C_1 coincides partially or completely with a face of C_2. Note that we consider a face to be open, that is, C_1 and C_2 are not considered to be neighbours when they coincide only in (part of) an edge.

We assume a coordinate system with axes parallel to the edges of the initial cube. In each dimension we consider an orientation along the coordinate axis. We now say that for each dimension there are two *directions*, considering the orientation on the corresponding axis. In this sense, we also say that a cell has a *neighbour in a particular direction.*

4.3 Uniform Grid.

We consider the k-dimensional uniform grid to be represented by a 2^k-tree and denote h for the height of the tree and d for the number of cells along an axis ($d = 2^{h-1}$). A straightforward computation gives us the following results. The number of cells in the grid is equal to d^k, the number of neighbours of a cell in the grid is $2k$, and the total number of neighbours is of order of $2kd^k$.

4.4 Non-Uniform Grid.

We still denote k for the dimension of the grid. We denote h for the height of the tree. By h is also expressed the smallest size of a cell in the non-uniform grid. An upper bound for the number of cells in the non-uniform grid is given by $(2^{h-1})^k$.

4.4.1 Number of Cells.

Assume that we construct the non-uniform grid by subdividing stepwise the k-dimensional cubes, so that in step s we only subdivide cubes of size s in cubes of size $s+1$. We denote the number of subdivisions during step s of cubes of size s in cubes of size $s+1$ by σ_s ($s = 1, ..., h-1$).

Then the number of cells of size $s+1$, denoted by A_{s+1}, is equal to

$$A_{s+1} = \sigma_s 2^k - \sigma_{s+1}, \text{ where } \sigma_h = 0, \text{ for } 1 \leq s < h-1,$$

since by subdividing a cube of size s one cell (considering the 2^k-tree as it was before executing step s) of size s is replaced by 2^k cells (considering the 2^k-tree as it is after executing step s) of size $s+1$ and in the next step σ_{s+1} of those will be subdivided further. Obviously, when $\sigma_1 = 0$, we have $A_1 = 1$, and when $\sigma_1 = 1$, we have $A_1 = 0$.

The total number A of cells in a non-uniform grid corresponding to the tuple $(\sigma_1, ..., \sigma_{h-1})$, is equal to

$$A = 1 + (2^k - 1) \sum_{s=1}^{h-1} \sigma_s,$$

whereby the sum appearing in the formula is read 0 when $h = 1$.

4.4.2 Acceptable Grids.

Not every tuple $(\sigma_1, ..., \sigma_{h-1})$ of $h-1$ natural numbers corresponds to a grid. A necessary and sufficient condition for such a tuple to represent a grid is:

$$\sigma_1 \in \{0, 1\},$$

$$\sigma_{s+1} \leq 2^k \sigma_s \text{ (for } s = 1, ...h-2).$$

Denote the minimum desired amount of cells of smallest size in the non-uniform grid by D. Considering a tuple $(\sigma_1, ..., \sigma_{h-1})$, σ_t (for $t = 2, ..., h-2$) must at least be equal to $\lceil \frac{\sigma_{t+1}}{2^k} \rceil$ in order to end up with the minimum desired amount of cells after $h-1$ steps. More cells than the minimum desired amount can be created, so σ_t can have all natural numbers in the interval

$$[\lceil \frac{\sigma_{t+1}}{2^k} \rceil, \sigma_{t-1} 2^k] \text{ for } t = 2, ..., h-2$$

as its value. Obviously, if the desired amount of cells is strictly more than 1, then $h \geq 1$ and $\sigma_1 = 1$. The value for σ_{h-1} is $\lceil \frac{D}{2^k} \rceil$, since we want a grid with at least D cells of smallest size, but with not more than the absolutely necessary amount of cells of smallest size.

When for some t the interval $[\lceil \frac{\sigma_{t+1}}{2^k} \rceil, \sigma_{t-1} 2^k]$ is empty, the tuple $(\sigma_1, ..., \sigma_{h-1})$ is not acceptable, since then the inequation $\sigma_{t+1} \leq 2^k \sigma_t$ does not hold, which was the necessary and sufficient condition of above.

4.4.3 Number of Cubes.

When we consider the tuple $(\sigma_1, ..., \sigma_{h-1})$, and all its subtrees corresponding to the tuples $(\sigma_1, ..., \sigma_s)$ for $s = 1, ..., h-1$. Then the number C_s of cells in the grid corresponding to tuple $(\sigma_1, ..., \sigma_s)$ is equal to

$$C_1 = 1 + \sigma_1(2^k - 1),$$

$$C_s = C_{s-1} - \sigma_s + 2^k\sigma_s \text{ (for } s = 2, ..., h-1).$$

If we construct the grid corresponding to tuple $(\sigma_1, ..., \sigma_{h-1})$, stepwize we go through all the grids corresponding to tuples $(\sigma_1, ..., \sigma_s)$ for $s = 1, ..., h-1$. For tuple $(\sigma_1, ..., \sigma_{h-1})$, the number of cells that are not subdivided in step s is equal to $\sigma_{s-1}2^k - \sigma_s$, and the number of cells of sizes 1 up to $s-1$ in the non-uniform grid is equal to $C_{s-1} - \sigma_s$.

4.5 An Example.

Let $k = 3, h = 5$. Assume that the number of cells of smallest size in the non-uniform grid has to be 400. We look for all tuples $(\sigma_1, ..., \sigma_4)$, using the notation of above.

We observe that $\sigma_1 = 1$ and $\sigma_4 = 50$ must hold. Several values for σ_2 and σ_3 are possible here. For σ_2 all values between 1 and 8 are acceptable. For a given σ_2, σ_3 can have all values between 7 and U_{σ_2}, where $U_{\sigma_2} = \sigma_2 2^k$.

The maximal number of cells in the non-uniform grid is 862, which occurs for the highest acceptable values for σ_2 and σ_3, namely $\sigma_2 = 8$, $\sigma_3 = 64$.

4.6 Some Properties and Numbers.

4.6.1 Realizable Precision.

Here we assume that k and h are given fixed values. We assume that D can be written as $D = 2^{k(h-1)-p}$. We address ourselves to the question what the upper and lower bounds for p are.

Obviously the lower bound for p is 0, to which corresponds the uniform grid with $2^{k(h-1)}$ cells.

The grid corresponding to the upper bound of p is the non-uniform grid with minimal amount of cells. This grid corresponds to the $h-1$-tuple $(\sigma_1, ..., \sigma_{h-1}) = (1, ..., 1)$. The minimal value for D therefore is equal to

$$D_{\text{min}} = \sigma_{h-1}2^k = 2^k.$$

Thus, p cannot be larger than $k(h-2)$.

So for a non-uniform grid, p must be an integer between 1 and $k(h-2)$. When D is given, we can compute p from the formula: $p = k(h-1) - \log_2 D$.

4.6.2 Acceptable Height.

Here we assume that both k and D are given. For minimal height the tree has to be built as wide as possible. So the number of cells of smallest size in a tree of height h corresponding to a uniform grid must be larger than D and for minimal height we must have that the number of cells of smallest size in a tree of height $h-1$ corresponding to a uniform grid is strictly smaller than D. Hence

$$2^{k(h-2)} < D \leq 2^{k(h-1)}$$

must hold for minimal h, and we get

$$1 + \frac{\log_2 D}{k} \leq h_{\text{min}}.$$

Thus the minimal h for which D can be realised is then $h_{\text{min}} = \lceil 1 + \frac{\log_2 D}{k} \rceil$.

An upper bound for h does not exist. Indeed, if $(\sigma_1, ..., \sigma_{h-1})$ is a tuple for which D is realised, then the tuple $(1, \sigma_1, ..., \sigma_{h-1})$ also realises D and is larger.

4.6.3 Grids With Many Cells.

From the paragraph on acceptable grids it is clear that the highest amount H of cells in a k-dimensional non-uniform grid with at least D cells of smallest size and corresponding to a 2^k-tree with given height h is reached for the highest acceptable values for $(\sigma_2, ..., \sigma_{h-2})$. Note that H is the upper bound for the number of cells in a non-uniform grid with D cells of smallest size, which is a smaller upper bound than $(2^{h-1})^k$.

Denote $\tau_{h-1} = \lceil \frac{D}{2^k} \rceil$, and for $t = 1, ..., h-2$: $\tau_t = \lceil \frac{\tau_{t+1}}{2^k} \rceil$. Then, providing that $D \leq (2^k)^{h-1}$, we have

$$\tau_t \leq (2^k)^{t-1}.$$

Therefore, considering the tuple $(1, \sigma_2, ..., \sigma_{h-2}, \lceil \frac{D}{2^k} \rceil)$ with $\sigma_t = (2^k)^{t-1}$ for $t = 2, ..., h-2$,

$$H = 1 + (2^k - 1)1 + (2^k - 1)(\sum_{s=2}^{h-2} (2^k)^{s-1}) + (2^k - 1)\lceil \frac{D}{2^k} \rceil$$

$$= 1 + (2^k - 1)(1 + \lceil \frac{D}{2^k} \rceil) + 2^k((2^k)^{h-3} - 1).$$

When we assume that $D = 2^{k(h-1)-p}$, then straightforward rewriting gives us the following result:

$$H = 1 + (2^k - 1)2^{k(h-2)-p} + 2^k(2^{k(h-3)} - 1)$$

$$= 1 - 2^k + 2^k(2^{k(h-2)-p} + 2^{k(h-3)}) - 2^{k(h-2)-p}$$

$$= 1 - 2^k + 2^{k(h-2)-p}(2^p + 2^k - 1).$$

4.6.4 Grids With Few Cells.

The fewest number L of cells in a non-uniform grid with D cells of smallest size is equal to

$$L = 1 + (2^k - 1)\tau_1 + (\sum_{s=2}^{h-2} \lceil \frac{\tau_{s+1}}{2^k} \rceil)(2^k - 1) + (2^k - 1)\tau_{h-1}$$

$$= 2^k + (\sum_{s=2}^{h-2} \lceil \frac{\tau_{s+1}}{2^k} \rceil)(2^k - 1) + |lceil \frac{D}{2^k} \rceil (2^k - 1),$$

where $\tau_{h-1} = \lceil \frac{D}{2^k} \rceil$ and $\tau_s = \lceil \frac{\tau_{s+1}}{2^k} \rceil$ for $s = h-2, h-3, ..., 3$.

We now compute the value for L for fixed h and for $D = 2^{k(h-1)-p}$. We now have

$$\tau_{h-1} = \lceil 2^{(h-2)k-p} \rceil = \max(1, 2^{(h-2)k-p}).$$

Assuming that $(h-2)k - p > 1$,

$$\tau_{h-2} = \lceil 2^{(h-3)k-p} \rceil = \max(1, 2^{(h-3)k-p}).$$

As long as $(h-t)k - p > 1$ holds, we get

$$\tau_{h-t} = \lceil 2^{(h-t-1)k-p} \rceil = \max(1, 2^{(h-t-1)k-p}).$$

In general, $\tau_s = \max(1, 2^{(s-1)k-p})$. Define q such that $q < h - \frac{p+1}{k}$ and $q + 1 \geq h - \frac{p+1}{k}$. This gives us

$$L = 1 + (\sum_{s=2}^{h-(q+1)} 1 + \lceil 2^{(h-q)k-p} \rceil + \sum_{s=h-q+1}^{h-1} 2^{(s-1)k-p})(2^k - 1)$$

$$= 1 + (h - q - 2 + \lceil 2^{(h-q)k-p} \rceil)(2^k - 1) + \frac{(2^k)^{h-q+1}((2^k)^{q-1} - 1)}{2^{k+p}}.$$

4.6.5 Fixed Number of Cells.

In contrast to the previous, in this subsection we assume the total amount of cells A in the non-uniform grid to be given. We also assume that k and p (and thus $D = 2^{k(h-1)-p}$) are given. We now look for acceptable values for h. Observe that from the formula for H, we now can deduce the minimal acceptable value for h if we assume that A is equal to H (i.e. the cells of smallest size are spread as widely as possible over the grid). Analogously, from the formula for L we can deduce the maximal acceptable value for h, assuming that $A = L$ (corresponding to the most densely distribution of the cells of smallest size). We do get an upper bound for h here, since the total amount A of cells in the grid is given.

We thus get from the formula for H:

$$h_{\min} = 3 + \frac{1}{k} \log_2 \frac{A + 1 - 2^k}{1 - 2^{-2k-p} - 2^{-k-p}}.$$

4.7 Reasonable D, k, and h

In practical examples k typically might be large, say $k = 6, 7, 8, \ldots$. We assume h usually is equal to $2, 3, 4$ or 5. Assuming that D can be written as $D = 2^{kh-p}$, then typically p will be 2 or 3, although acceptable values are up to $k(h-1)$.

Assuming that the amount of cells in the non-uniform $(k =)6$-dimensional grid is $(A =)15000$, we get from the formula above for $p = 2$ and for $p = 3$, $h_{\min} = 6$. For $k = 7$ and $p = 2$ or 3, $h_{\min} = 5$.

4.8 A Particular Family of Polygons.

In this section, we consider the 2-dimensional case. We study the amount of cells needed for modelling a particular kind of polygons.

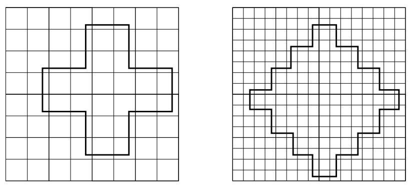

Figure 4.1: Polygons: Cases $j = 1$ and $j = 2$.

The family of polygons that we consider here, have the following characteristics. The edges are all parallel to the sides of the cells of the grid and they are all of equal length. The length of the edges is equal to the length of a cell of size $h - 2$, where h is the height of the quadtree corresponding to the grid. We assume that the grid is not larger than necessary for modelling the polygon. The polygon is concave and the number of edges is equal to $16j - 4$, where $j = 1, 2, \ldots$. In Figure 4.1 the polygons for $j = 1$ and $j = 2$ are drawn. Here, the polygons are drawn in the uniform grid.

This family of polygons can be constructed with the building blocks drawn in Figures 4.2 and 4.3. These building blocks can be glued together only in some particular combinations. In order to build a staircase, we can only glue two identical building blocks together. For this, an additional vertex has to be inserted, as demonstrated in Figure 4.4. In order to glue two staircases together, only type one and two, and type two and three can be combined, as demonstrated in Figure 4.5.

In order to construct the 12-gon ($j = 1$) of the family, we have to glue building block one to building block two (rotated over $\frac{\pi}{2}$), followed by building block three (rotated over π) and building block two (its mirror image). In general, for a $(16j - 4)$-gon of the family, we need a staircase of j copies of building block one, followed by a staircase of j copies of building block two (rotated over $\frac{\pi}{2}$), followed by a staircase of j copies of building block three (rotated over π) and connected by a staircase of j copies of building block two (its mirror image).

This allows us to compute the $h-1$-tuple $(\sigma_1, \ldots, \sigma_{h-1})$ and thus A and D. Indeed, when considering the inside of the obstacle on the left side of the contour in Figures 4.2 and 4.3, all building blocks contribute 1

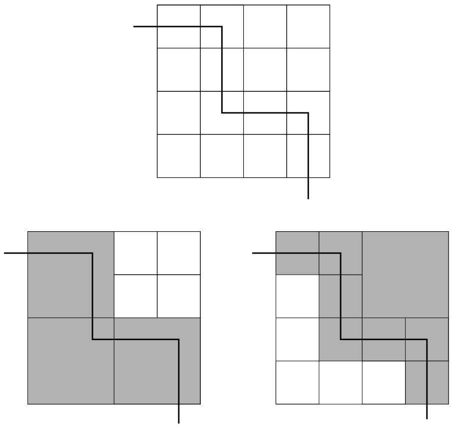

Figure 4.2: Building Block of Type One.

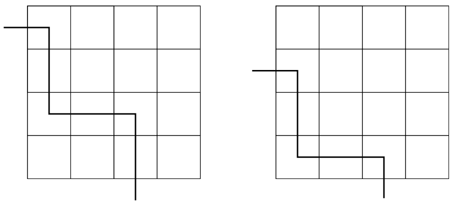

Figure 4.3: Building Blocks of Types Two and Three.

to σ_{h-2}, building block one contributes 0 to σ_{h-1}, building block two contributes 2 to σ_{h-1}, and building block three contributes 3 to σ_{h-1}. (When the inside is on the right hand side of the contour in the figures, then building block one contributes 3 and building block three contributes 0 to σ_{h-1}, all the other contributions stay the same.) Gluing together two building blocks to a staircase in all cases contributes 1 to σ_{h-2}. In the cases where two building blocks of type one or two are glued, there is a contribution of 1 to σ_{h-1}, and for gluing two building blocks of type three there is no contribution to σ_{h-1}. Gluing two building blocks of different type does not affect the $h-1$-tuple further. The relation between j and h is given by:

$$j = h - 2.$$

Thus, for the 12-gon we get the tuple $(1, 4, 7)$, $A = 37$, $D = 28$. For the 28-gon we get the tuple $(1, 4, 9, 15)$, $A = 88$, $D = 60$. And in general, for the $(16j-4)$-gon we get $(1, 4, ..., 4^{j-1}, 5j-1, 8j-1)$, $A = 4^j + 39j - 6$ and $D = 32j - 4$. The uniform grid consists of $U = 4^{j+2}$ cells. The ratio $\frac{A}{U}$ for $j = 1, ..., 13$ is given in following table.

j	$\frac{A}{U}$	j	$\frac{A}{U}$
1	0.57812500	8	0.06279182
2	0.34375000	9	0.06258225
3	0.17089844	10	0.06252289
4	0.09912109	11	0.06250630
5	0.07403564	12	0.06250172
6	0.06597900	13	0.06250047
7	0.06351852		

Considering the same family of polygons, but with longer edges, corresponds to adding another level to the quadtree. If we now suppose that the length of the edges is equal to the length of the cells of size $\hat{h} - 3 - t$, where $\hat{h}$ is the height of the quadtree, then we get a tuple that is t longer than in the previous

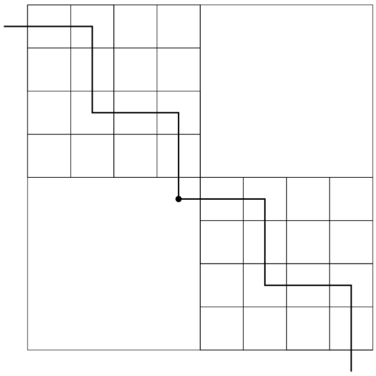

Figure 4.4: Staircase.

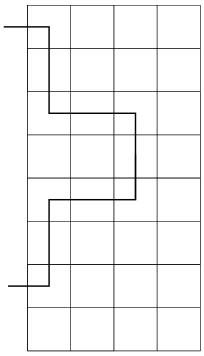

Figure 4.5: Gluing Two Different Kinds of Building Blocks.

($\hat{h} = h+t$), and for which the first part of the tuple is the same a the tuple we had before. In fact, we add σ_{h-1+s}, for $s = 1, ..., t$. Assuming that every vertex of the polygon with n vertices lies in the SW-cube (south-west, the third branch in the quadtree), we get the following formula:

$$\sigma_{h-1+s} = 1 + 2^s \frac{n}{2}.$$

Indeed, for each edge for which we have to refine the grid, on s levels lower than the original one we split 2^s cells along that edge. In the staircase of type one, we get $\frac{\frac{n}{4}-1}{2}$ horizontal and $\frac{\frac{n}{4}-1}{2}$ vertical edges, each for which 2^s cubes have to be split. In the staircase of type two, we have $\frac{\frac{n}{4}+1}{2}$ edges for which 2^s cubes have to be split. The same amount is to be considered for the staircase of type two in mirror image. And in the staircase of type three no split has to be performed. Then for the vertices, there is one convex vertex more for which a cube has to be split on s levels, than there are concave vertices for which we counted a cube double in the discussion of the edges above. So, we get $\sigma_{h-1+s} = 2^s(2\frac{\frac{n}{4}-1}{2} + 2\frac{\frac{n}{4}+1}{2} + 0) + 1$, which can be rewritten in the formula presented earlier.

Counting that out for various values for j and t, we get the results of the next table.

j	t	number cells uniform grid	number cells non-uniform grid	Non-uni/uni * 100
1	1	256	184	71.875000
1	2	1024	475	46.386719
1	3	4096	1054	25.732422
1	4	16384	2209	13.482666
1	5	65536	4516	6.890869
1	6	262144	9127	3.481674
1	7	1048576	18346	1.749611
2	1	1024	763	74.511719
2	2	4096	2110	51.513672
2	3	16384	4801	29.302979
2	4	65536	10180	15.533447
2	5	262144	20935	7.986069
2	6	1048576	42442	4.047585
2	7	4194304	85453	2.037358
3	1	4096	2290	55.908203
3	2	16384	6517	39.776611
3	3	65536	14968	22.839355
3	4	262144	31867	12.156296
3	5	1048576	65662	6.262016
3	6	4194304	133249	3.176904
3	7	16777216	268420	1.599908
4	1	16384	6169	37.652588
4	2	65536	17692	26.995850
4	3	262144	40735	15.539169
4	4	1048576	86818	8.279610
4	5	4194304	178981	4.267240
4	6	16777216	363304	2.165461
4	7	67108864	731947	1.090686
5	1	65536	15808	24.121094
5	2	262144	44995	17.164230
5	3	1048576	103366	9.857750
5	4	4194304	220105	5.247712
5	5	16777216	453580	2.703547
5	6	67108864	920527	1.371692
5	7	268435456	1854418	0.690825
6	1	262144	39655	15.127182
6	2	1048576	110314	10.520363
6	3	4194304	251628	5.999303
6	4	16777216	534256	3.184414
6	5	67108864	1099507	1.638393
6	6	268435456	2230006	0.830742
6	7	1073741824	4491001	0.418257
7	1	1048576	99598	9.498405
7	2	4194304	265489	6.329751
7	3	16777216	597268	3.559995
7	4	67108864	1260823	1.878773
7	5	268435456	2587930	0.964079
8	1	4194304	256309	6.110883
8	2	16777216	637240	3.798246
8	3	67108864	1399099	2.084820
8	4	268435456	2922814	1.088833
8	5	1073741824	5970241	0.556022
9	1	16777216	692572	4.128051
9	2	67108864	1552735	2.313755
9	3	268435456	3273058	1.219309
9	4	1073741824	6713701	0.625262

We observe that the number of cells in the uniform grid grow rapidly, while for the non-uniform grid the relative growth of the number of cells is less. When we assume that the total amount of cells in the grid is fixed, for increasing j the highest possible t is decreased. However, with the same amount of cells the non-uniform grid can be much more precise in modelling the polygons than the uniform grid can. If we assume that the amount of cells that can be stored is about 30000 at most, we see that with increasing j the highest possible t decreases. However, the highest possible value for t for non-uniform grids is considerably larger than the highest possible value for t for the uniform grid. This is demonstrated by the next table.

j	Non-Uniform Grid, t	Uniform Grid, t
1	7	4
2	5	3
3	4	2
4	2	1
5	1	-

4.9 Summing Up.

In this chapter we have addressed ourselves to the question how many cells a non-uniform grid can have when the representation of the k-dimensional grid is done using a 2^k-tree.

For this, we introduced several parameters and quantities:

- We found bounds on the acceptable amounts of subdivision in each level of the tree.
- For the most widely spread distribution of cells of smallest size over the grid, for given dimension k and given height h of the tree, we found a formula for the total amount H of cells in the grid. This is an upper bound for the amount of cells. Similarly, we found a lower bound L, the total amount of cells in the grid when the cells of smallest size are most densely distributed (all positioned in one branch of the tree) over the grid.
- We looked for a formula giving us the height a tree must have in order to model a grid with given amount of cells of smallest size and given total amount of cells.

We also considered some particular values for the parameters corresponding to some practical applications carried out over the last years at the Research Institute for Symbolic Computation, in the fields of geometric modelling and obstacle avoidance.

Finally we studied the amount of cells needed in a non-uniform grid for modelling a particular family of polygons in 2 dimensions. We compared this amount with the number of cells in a uniform grid and observed a considerable reduction of the total number of cells needed when using a non-uniform grid.

Chapter 5

Our Potential is a Distance

Here we consider a grid of arbitrary dimension k and define the distances we are going to use in the following chapters. Throughout this chapter, denote S the white cell enclosing the initial position, and C_1 and C_2 are some arbitrary white cells. A path P from C_1 to C_2 is a list of white cells $(A_0, ..., A_m)$, where $A_0 = C_1$, $A_m = C_2$, and A_j is adjacent to A_{j+1} for all $j = 0, ..., m-1$. The length $l(P)$ of path P is defined equal to m.

Corresponding to the potential we use in our non-hierarchical approach we define

$$d(C_1, C_2) = \min_{P \in \mathcal{P}} \{l(P)\},$$

where $\mathcal{P}$ is the set of all paths from C_1 to C_2. In case the set $\mathcal{P}$ is empty, we set $d(C_1, C_2) = \infty$. The function $d(.,.)$ is defined on the set of white cells. It is easy to show that $d(.,.)$ is a distance.

The value for cell C of the non-hierarchically constructed potential field on the non-uniform grid can thus be written as $d(C, S)$.

In case the grid is uniform it is clear from the definition that $d(.,.)$ equals the L^1-distance.

Corresponding to the potential we use in our hierarchical approach we define

$$D(C_1, C_2) = \min_{P \in \mathcal{P}} \{l_h(P)\},$$

where $l_h(P)$ is the weighted length of path P from C_1 to C_2. The weighted length of path $P = (A_0, ..., A_m)$ can be written as

$$l_h(P) = \sum_{i=1}^{m} w_i,$$

where w_i is the weight of the move from A_{i-1} to A_i. We will consider a weight of a move that depends on the direction of the move: for the j th direction the weight is equal to c^{k-j} $(j = 1, ..., k)$, where c is a constant (in the following chapters we will use $c = 2$ in the examples). Notice that $w_j > 0$. In case the set $\mathcal{P}$ is empty, we set $D(C_1, C_2) = \infty$. The length of a path can thus also be written as

$$l_h(P) = \sum_{j=1}^{k} m_j d_j^{k-j},$$

where m_j is the number of times a move in the j th direction is to be executed along path P. The function $D(.,.)$ is defined on the set of white cells. It is easy to show that $d(.,.)$ is a distance.

The value for cell C of the hierarchically constructed potential field on the non-uniform grid can thus be written as $D(C, S)$.

The second formula for $l_h(P)$ shows that the direction and the size of cells both play a role in the length of the path and thus in the value of the potential field at cell C.

Chapter 6

Computing Neighbouring Cells and Non-Hierarchical Approach

6.1 Introduction

In order to find a collision-free path from a given start position to a given goal position amid obstacles, we propose an approach using a discrete potential function on a non-uniform grid that models the space of all Free Positions in the Configuration Space (C-space) of the robot. The resulting path is a good pay-off between safest and shortest path and the grid should allow us to deal with obstacles moved slightly away from the position they were in when modelling began.

In this chapter we present a non-hierarchical construction of a Manhattan distance-like potential function on a non-uniform grid in C-space.

On the non-uniform grid, we build a potential field that looks similar to the Manhattan Distance on a uniform grid. To each cell c of the grid, we assign a value of the potential field that expresses the number of cells that have to be visited in the shortest sequence of cells between c and the goal position. The path from the start to the goal position follows the descent of this potential field.

This approach seems to be suitable for a combination of possible extensions to the general Robot Motion Planning Problem. The descent of the potential function is steeper along cells of bigger size, and thus the robot tends to move away from the obstacles to "safe" cells. But the approach also leeds to a "fast" path through the Configuration Space, due to the definition of the Manhattan distance-like function. However, this approach does not have the main disadvantage of the Potential Field Approach as known in the literature, since local minima do not occur for this Manhattan distance-like function. Furthermore, this approach delivers a complete solution (i.e., always finds a path if there is one and reports failure otherwise if the space is bounded). Since the representation is done by a non-uniform grid, the size of the cells along the path gives us some information about the allowed error during the execution of the path. So also uncertainty seems to be incorporated in a natural way in this approach.

The potential field approach on a grid leads in a natural way to parallel implementation and the computation of the potential can be done with a technique known as "wavefront expansion" (see [Lat91b], [CLZ93]).

In the next section, we introduce the well-known notion of 2^k-tree and two representations for it. In Section 3, we describe the problem tackled in this article. In Sections 4 and 5, we explain the way in which the problem is solved. In Section 4 we describe our algorithm to compute the neighbours of a cell in the non-uniform grid. In Section 5 we give our algorithm to compute the Manhattan distance-like potential function. In Section 6, we briefly describe an implementation of these algorithms and discuss some experimental results. Finally we conclude with some considerations for future work on the approach presented here.

6.2 Representation of a Non-Uniform Grid

In modelling, the 2^k-tree is a well known technique for representing objects in k-dimensional space (called quadtree for $k = 2$ and octree for $k = 3$). Its properties are well studied and discussed in the literature (e.g. in [Sam90]). There are several ways to represent the structure of such a tree by lists of numbers. We are concerned with two of these representations (presented in [Gar82b] and [Bur92]).

6.2.1 2^k-Tree

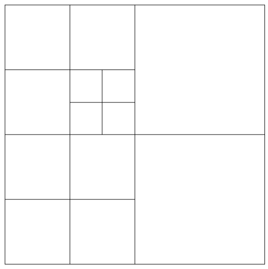

Figure 6.1: A Partitioning of a Rectangular Area .

A quadtree is a tree representation of a partition of an area into smaller subareas, possibly of different sizes. Assume a rectangle divided into smaller rectangles as in Figure 6.1, where the partitioning is done by cutting a rectangle into 4 smaller rectangles of equal size.

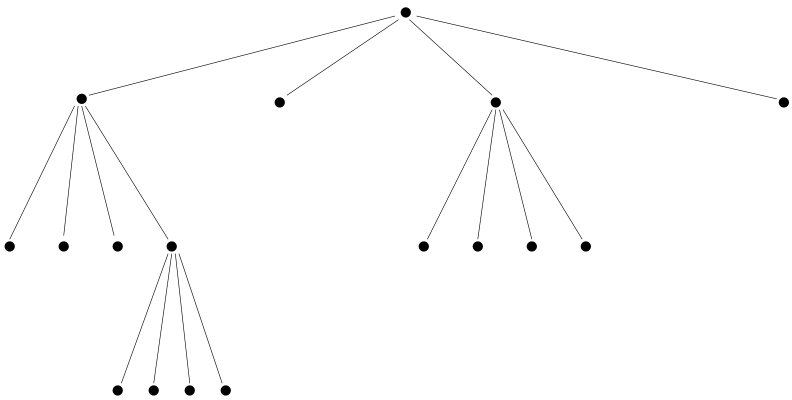

Figure 6.2: The Quadtree Corresponding to the Partitioning in Figure 1.

We represent this partitioning by a tree, where each node corresponds to a particular rectangle. The root corresponds to the original rectangle. Each child of a node n corresponds to one of the 4 smaller rectangles in which the rectangle corresponding to n was divided. The quadtree corresponding to the partitioning in Figure 6.1, is depicted in Figure 6.2.

An octree is the straightforward generalization of the quadtree to 3 dimensions. Here one starts with a cube and partitions it into 8 smaller cubes of equal size. In this case, every parent node will have 8 children. A 2^k-tree is the straightforward generalization of the quadtree to k dimensions.

6.2.2 IG-tree Representation

In [Gar82b], a representation of octrees is described, which was a generalization of a representation of quadtrees also due to Gargantini. We will refer to this representation in the rest of the chapter by the name *IG-tree representation*. With this representation, neighbouring cells are easy to find. Once having the concept of IG-trees in 2D, it is straightforward to see how it generalizes to higher dimensions. So, we next give the IG-tree representation for a quadtree.

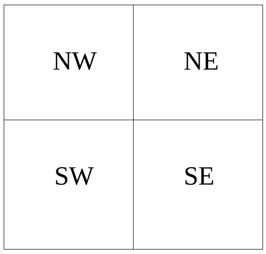

Figure 6.3: Denotation for Cells.

For the IG-tree representation of a quadtree we assume that the children of a node are always listed from left to right according to the relative position of the corresponding partitioning. We can denote the cells as in Figure 6.3, where NW, NE, SW, and SE stands for north-west, north-east, south-west, and south-east respectively, and we assume that the first child of a node in the tree always corresponds to the NW cell, the second child to the NE cell and so on. The tree in Figure 6.2 is already in this format.

Let us number the children of each leaf from left to right, with 0 (NW), 1 (NE), 2 (SW), and 3 (SE). We now represent a leaf in the tree by a tuple

$$(a_0, a_1, ..., a_h),$$

where h is the height of the tree and each of the a_i is an element of the set $\{0, 1, 2, 3, X\}$. Roughly stated, an X is used to indicate leaves in the tree with height less than the height of the tree. In the following we will call such a tuple a *code* for the corresponding cell. When we go from the root to a leaf following branches of the tree, we will in general pass by other nodes. On each level i, we set a_i to the value corresponding to this intermediate node. When we reach the leaf, we fill up the tuple with symbols "X", that is, when a leaf has height m in the tree and $m < h$, then for all $i > m$, we set a_i to X. Doing so for the tree of Figure 6.2, we get the tuples as depicted in Figure 6.4.

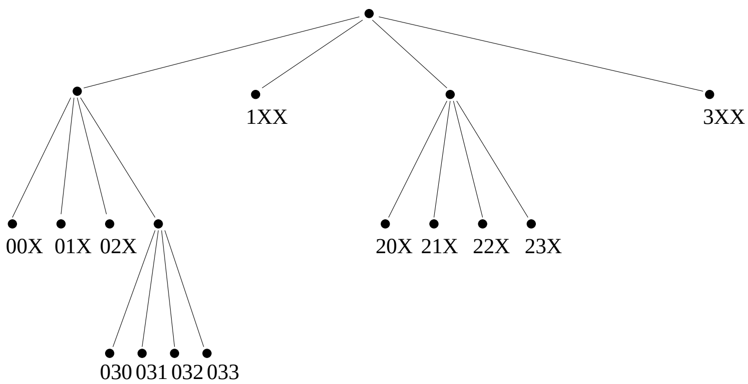

Figure 6.4: The IG-tree Representations for the Quadtree of Figure 6.2.

The generalization to octrees is now done by introducing a new direction. A cubic cell will be partitioned in 8 smaller cubes. The 4 cubes on the "front" can be encoded by NWF, NEF, SWF, and SEF, the 4 cubes on the "back" by NWB, NEB, SWB, and SEB. We assume that the children of the nodes are always ordered according to the sequence NWF, NEF, SWF, SEF, NWB, NEB, SWB, and SEB, and we number the children from left to right by 0, 1, ..., 7. The IG-tree representation is then obtained by proceeding in the same way as for the quadtree.

The same construction can be done for 2^k-trees, coming from $k-1$ to k dimensions, where we need k directions and 2^k possible values for a_i (besides "X"). Further we will consider this general case also. The size of the data, however, grows very rapidly for increasing k.

The partitioning is then represented by a list of codes, one code for each cell.

6.2.3 GET Representation

Another, but very compact, representation for quad- and octrees, the so-called *GET*[1], has been invented (see [Bur92]). Starting from the tree, where we assume as in the previous that the children of each node are ordered according to the position of the respective cells, here one constructs two lists with binary entries. The first list, called *DFA*[2], is constructed by running through the tree in a breadth-first transversal order. When passing through a node, a "0" is added to the list when the node is a leaf, and a "1" when the node has children. In the case of the tree in Figure 6.2, we get the following DFA:

1 1010 0001 0000 0000.

In fact, we can begin with recording on the level (height) in the tree where the first leaf will be encountered, i.e. we can drop the leading "1" in the example above, since it represents the root. In the following, we will assume that the DFA does only start recording on the level where the first leaf is encountered. In particular, we will always drop the first entry, representing the root. We will call an element ("0" or "1") in the DFA-list an *entry*.

As the spaces in the DFA of the example insinuate already, a way of looking at the DFA corresponding to a quadtree is by taking parts of length 4 out of this representation of the quadtree. We will call these parts of length 4 *4-tuples* of entries. Indeed, one sees that the first "1" in the list (say, representing node n) gives rise to a new 4-tuple of entries representing the children of node n. This new 4-tuple is in this case the second 4-tuple in the DFA. The second "1" in the DFA gives rise to the third 4-tuple, the third "1" to the last 4-tuple.

Let us now look to the case of higher dimensions. In the case of an octree, the DFA can be seen as a list of tuples of length 8 and the two previous remarks will also hold in this case. For a 2^k-tree, moreover, we will look for tuples of length 2^k.

Note that the whole partitioning is now represented by this one DFA-list. The second list, called *DFB*[3], is not of any importance when we are only interested in the partitioning of an area. It describes for each leaf, i.e. for each zero in the DFA, whether the cell is coloured black or white, for example, "inside" or "outside" an obstacle when we consider robot environments. The m-th entry in DFB corresponds to the m-th zero in the DFA, and black or white can also be represented with one digit (1 or 0). The DFB list does not play a role in the partitioning of the C-space, but will be needed in the motion planning. The potential field will be kept in a list similar to the DFB list (see below).

Looking more closely to one of the 4-tuples in the DFA of a quadtree and reminding ourselves that the children of a node are ordered always in the same way according to the position of the corresponding cell in the partitioning, we observed that the first entry corresponds to NW, the second to NE, and so on. This observation is the essential link between the IG-tree representation and the GET representation, which enables us to compute the entries of neighbours in a DFA-list also in a simple way.

6.3 General Description of the Problem

The reason for dealing with the problem of finding neighbours in the GET representation of 2^k-trees is, that we want to plan a collision-free path for a robot arm by going through following steps:

- Convert the path planning problem in the workspace of the robot to the path planning problem for a point in the joint space (or configuration space) of the robot.
- Put a non-uniform grid on the C-space. This grid is fine grained close to the boundary of the set of free placements in the C-space (i.e. close to the obstacles in C-space), and coarse grained elsewhere.

[1] Stands for "Grundelement" in [Bur92].
[2] Stands for "Datafield A" in [Bur92].
[3] Stands for "Datafield B" in [Bur92].

- Assign the value "0" to the cell c_g with the goal position lying in it. Compute a new value for all neighbouring cells c_m (all in the free space, subscript m takes values 0,1,...), based on the value for cell c_g. In this way we compute a discrete potential function u on all cells of the free space. As a first attempt, we will just add "1" to the value for cell c_g and give this as the new values for the neighbours (in this case, when the grid would be uniform, we would get the Manhattan distance to the goal position). Iterate this process for the neighbours of the cells c_m (when we would have a uniform grid, this would be the wavefront expansion approach).

- If we have been given an initial position of the robot, we can now create a path by going from cell c_s in which the initial position lies to a neighbouring cell c_p with value less than the value in c_s. We then iterate this process for c_p until we reach c_g.

- We might want to smooth out the path obtained.

The advantages of this approach we mentioned in earlier chapters already and they were also summarized in [VG93]. Of course, this is just a rough sketch of a general setting, and the aim of this chapter is not to cover all details of all these steps. We will, however, look more closely to the third and forth item on the list above.

The main advantage of the GET representation of the 2^k-tree and its underlying partitioning of an area, is the compactness of the representation (for a discussion on this aspect, see [Bur92]). It is however not clear at all how and where to find the neighbouring cells in the GET representation. The IG-tree representation is very well suited for the computation of neighbouring cells, and algorithms to find the neighbours for an octree are given in [Gar82b]. These algorithms can easily be generalized for 2^k-trees for arbitrary large k.

Another advantage of the GET representation lies in the second list, the DFB. This list will be replaced by a list of all values assigned by the discrete potential function u to the cells (which are represented by a zero in the DFA).

6.4 Computing Neighbouring Cells

Looking at the partition in Figure 6.1 again, we can see that some of the neighbours of a cell may be cells of the same partitioning step (i.e. cells of equal size, nodes with the same parent node). Sometimes however, a big neighbour is partitioned itself and gives rise to some smaller neighbouring cells. This phenomenon can occur even recursively. Also, not all neighbours are cells represented by nodes with the same parent node. In Figure 6.5, all neighbours of one particular cell are marked.

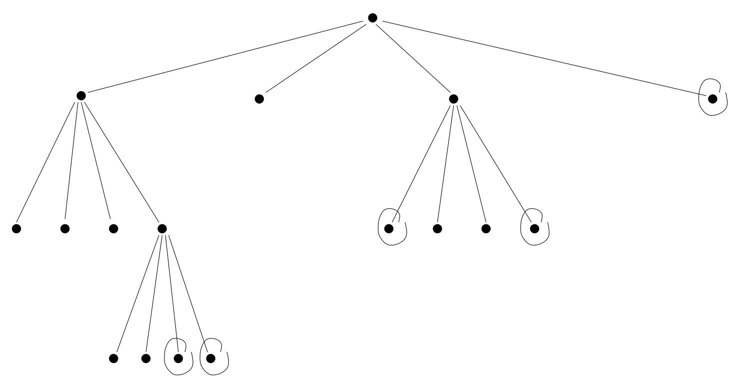

Figure 6.5: Neighbours of a Cell, Scattered About the Tree.

The central observation on which the algorithms in [Gar82b] are based, is that the neighbouring cells in the partition are represented by IG-tree codes equal up to a shift of some of the entries. For IG-tree representations, shifting in a particular direction means adding a fixed number (depending only on the direction) modulo 2^k. For example, in the example of Figure 6.1, the neighbour at the east of the cell with IG-tree code “21X” is “20X”, the neighbours at the East of the cell with IG-tree code “1XX” are “01X”, “031” and “033”, that is, the last digit before the “X” is decreased by 1 and then the children at the westside (with “1” or “3”) are found in the list of IG-tree representation codes. We observed that an analogous statement holds for the representation as DFA-list, and that the key idea of the algorithms computing neighbours in [Gar82b] can also be applied to the GET representation. This results in the algorithms described in the following subsections.

6.4.1 The Neighbours With Same Parent Node

Let us first look at the neighbours that are children of the same parent node. We look at the neighbours in a given direction, that is, the neighbour of a cell when looking in a certain given direction. We assume for the moment that all these children are leaves.

In the case of a quadtree, we have, for a given direction, two nodes that have a neighbour with the same parent node. When we look to the West, the nodes in the NE and SE have respective neighbours NW and SW at their westside. When we look to the North, the nodes SW and SE have respective neighbours NW and NE. Similar statements hold for the eastside and southside neighbours. In the DFA-list, we look at one particular 4-tuple, since the cells have the same parent node. All entries in this 4-tuple are “0”, as they all represent leaves. Let us number these entries from 0 to 3. For the westside, the entries with odd number have a neighbour with the same parent, namely the one represented by the even number left of them (i.e. entry 1 has neighbour with entry 0, entry 3 has neighbour with entry 2). For the eastside, entry 0 has neighbour 1, entry 2 has neighbour 3. For the northside, entry 2 has neighbour 0, entry 3 has neighbour 1. For the southside, entry 0 has neighbour 2, entry 1 has neighbour 3.

In the case of an octree, a similar thing happens. The relative position of the cells determine the ordering of the entries in the DFA in such a way that we can see the octree as two quadtrees: one quadtree on the front and one at the back. Therefore, the same shift operations as in the case of a quadtree hold, but we now have to use them for 8, that is twice for 4, entries. So, for example, when we look for the neighbours on the westside, the cells NEF, SEF, NEB, and SEB have respective neigbours NWF, SWF, NWB, and SWB. In the 8-tuple (consisting of “0”-s and let us number them from 0 to 7), the odd numbered entries have the even numbered entries as there westside neighbours, i.e. 1 and 0, 3 and 2, 5 and 4, 7 and 6. Similarly, for the neighbours on the northside, we get the shifts from 2 to 0, from 3 to 1, from 6 to 4 and from 7 to 5. Moving East and South is just the other way round. However, for the octree, we have a new dimension. We can also move to the front or to the back. In case of neighbours in front (moving to the front) we have for NWB, NEB, SWB and SEB, the respective neighbours NWF, NEF, SWF, and SEF. In the DFA-list, we get a shift in the 8-tuple from 4 to 0, from 5 to 1, from 6 to 2, and from 7 to 3. In the opposite direction it is moving to the back.

In general, for the k-dimensional case, we have all neighbours like in the $(k-1)$-dimensional case twice (a 2^k-tree can be seen as two 2^{k-1}-trees “behind each other”, taking into account the convention on the ordering according to the relative position of the cells in the partitioning of the area). These neighbours can be found in the 2^k-tuple by splitting it up in two 2^{k-1}-tuples, by taking the first 2^{k-1} entries as one 2^{k-1}-tuple and the remaining entries as another 2^{k-1}-tuple, and applying the rules for the 2^{k-1} case. However, two additional directions have to be considered as well. Let us denote them as A_k and B_k, and add them to the order as follows: W, E, N, S, F, B, A_4, B_4, A_5, B_5, ..., A_k, B_k (where we have the correspondence $W = A_1$, $E = B_1$, $N = A_2$, $S = B_2$, $F = A_3$, $B = B_3$). For neighbours on the A_k-side (when moving in the direction A_k), the nodes represented by the second part (second 2^{k-1}-tuple) have neighbours in the first part (first 2^{k-1}-tuple). Let us number the entries in the 2^k-tuple from left to right, beginning with 0 up to 2^k-1. Then for all $l \geq 2^{k-1}$, entry with number l has A_k-neighbour $l-2^{k-1}$.

The nice thing about it is, that this iteration also holds for $k=2$, that is, starting from a binary tree, representing two cells on an interval, only considering W and E as directions.

6.4.2 The Neighbours With Other Parent Node

Given a cell c, we now assume that the neighbour of c for which we are looking, is larger in size than c. This means that the neighbour of c cannot be found in the 2^k-tuple in which c is represented, but has to be found in a 2^k-tuple that occurs earlier in the DFA-list. We can apply the following algorithm.

input: The DFA-list, the entry for cell c.

output: The 2^k-tuple in which the entry in the DFA-list of the parent node p of the node representing cell c lies, and the entry of p itself.

step a: Divide the DFA-list into levels of the tree (left to right transversal, counting the encountered "1"-s).

step b: Divide the level on which the entry of c occurs, into 2^k-tuples.

step c: When c is in the i-th 2^k-tuple on level m, then the parent p of c is the i-th "1" in level $m-1$.

step d: Divide level $m-1$ (of parent p) in 2^k-tuples (same procedure as in step b).

step e: Return the i-th entry and the 2^k-tuple (on level $m-1$) in which this entry lies.

This case of the search for neighbours is reduced to the previous case, with the following observation. The larger neighbour of c on the Γ-side, where Γ is an element of a subset of $\{A_1, ..., A_k, B_1, ..., B_k\}$, is the neighbour of p when we neglect the fact that p is not a leaf in the 2^k-tree. (I.e. we forget about the children of p. This can be done, since these children never will be a neighbour for which we are looking in the case under consideration, here.)

A final remark about this case; The procedure described above, might have to be executed recursively on p. We might look for a neighbour that corresponds to a node several levels higher in the tree than the node corresponding to c. Then the node representing this neighbour lies several levels higher than the representation of c in the DFA.

6.4.3 Going Down To Smaller Sized Neighbours

Till now, we assumed that neighbours were always of equal size or larger than cell c. This is of course not always the case. However, in the previous, we never had to check whether the entry of a possible neighbour was "0" or not, and all the reasoning of above holds still up to the following small adaptation. Let us call nodes n that are a parent node of neighbours of c and not a parent node of c itself, *parent neighbours* of c. In the previous subsections, we could find such nodes n. So, we reduced the remaining problem of finding neighbouring cells of smaller size than parent neighbours of c represented by leafs, to finding the leafs which have parents who are parent neighbours of c. From now on, we assume that we are looking for these leafs.

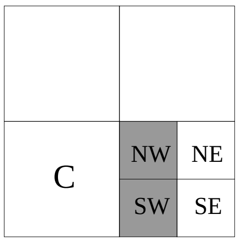

Figure 6.6: The eastside Neighbours of Cell c Are westside Children.

Let us first look at quadtrees. If we look for the eastside neighbours of cell c, we first find the parent neighbour of the neighbouring cells in the East. When this parent cell n is represented by a "1", it is divided into subcells, and we have to select the two children of n on the westside of the partition. This is illustrated in Figure 6.6. So, in the DFA, we have to look for the first and the third entry in the 4-tuple corresponding to the "1" representing node n. When these two children are leaves, we are done. When one of them, say node w is represented by a "1", we have to look for the westside children of this cell w, that is, to the first and the third entry in the 4-tuple corresponding to the "1" representing node w. When these are leaves, we are done, otherwise we have to iterate this procedure again.

When looking for the southside neighbours of c, we have to look for the northside children of the southside parent neighbour, and iterate when necessary. The analogue holds for west- and northside neighbours.

In the case of an octree, we can apply the same rules for N, S, W and E as in the case of a quadtree, only we have to apply this twice: again, we divide the 8-tuple in two 4-tuples, as described previously. For the directions F and B, we have to look for the B-, resp. F-, children of the F-side, resp. B-side, neighbours.

In the case of a general 2^k-tree, the same method of generalization can be applied. So we divide the 2^k-tuple in two 2^{k-1}-tuples for the directions A_1, ...,A_{k-1} and B_1, ...,B_{k-1}, and we look for the A_k-, resp. B_k-, children of the B_k-side, resp. A_k-side, neighbours.

6.5 The Discrete Potential Function u

We now come to the computation of the discrete potential function u by a wavefront expansion approach. The "0"-s in the DFA-list represent leaves in the 2^k-tree, and therefore cells in the partitioning of the area. When applying such a partitioning to the robot motion planning problem using the C-space of a robot arm, this partitioning is a non-uniform grid over the set of feasible configurations in C-space. So, some cells will be forbidden area for the robot (due to an obstacle in the Workspace or due to some physical limitations on the feasible motions of the robot arm), small cells will occur close to these forbidden areas. The goal position of the robot will lie in some cell c_g, the initial position usually in some other cell c_s. The path through this grid, consisting of a list $c_0, c_1, ..., c_q$, where $c_0 = c_g$, $c_q = c_s$, and c_p is a neighbouring cell of c_{p+1} for all $p = 0, ..., q-1$.

Then the following algorithm computes a discrete potential function u on the non-uniform grid, when calling it for $c = c_g$.

step a: Find the entry for cell c in the DFA-list.

step b: Find the entries for all neighbours of c.

step c: For each of the entries $\mathcal{E}$ of all neighbours $N_{\mathcal{E}}$ of c, do

- If the corresponding cell is not feasible (i.e. if the corresponding entry in the DFB is a "1"), put "X" on the corresponding position in the u-list,
- Else, $u(N_{\mathcal{E}}) \leftarrow u(c) + 1$, and put this value on its place in the u-list.
- Call this algorithm recursively for $N_{\mathcal{E}}$.

The path starts from c_s, and can be built up by choosing at each step a neighbour c_{p+1} of the cell c_p added one step earlier, for which $u(c_{p+1}) < u(c_p)$ holds, until c_g is reached. In the rest of this chapter we assume that such a path exists. If such a path does not exist and the connected component of the C-space in which c_g lies is bounded, the wavefront expansion algorithm will stop at this boundary. In the case when a path does exist, the wavefront algorithm may stop as soon as c_s is reached and $u(c_s)$ is computed, or when the Workspace is bounded, when all cells c (finitely many) are reached and $u(c)$ is computed for all cells. This remark is exactly the same as in the wavefront expansion approach for a uniform grid in [Lat91b].

This potential function u has the following meaning. For each feasible cell c, the value $u(c)$ is the minimal number of cells that has to be crossed for reaching cell c_g. In particular, $u(c_g) = 0$, $u(c_{p+1}) - u(c_p) = 1$ for each $p = 0, ..., q-1$.

6.6 Implementation, Experimental Results

We implemented the computation of neighbours, the construction of the potential field and pathfinding on a Silicon Graphics machine in the programming language $C++$, using extensively some standard $C++$-classes available in the *gnu*-library ([Lea91]). This program is a sequential algorithm, which already performs quite well for a decomposition of the space into up to 8,000 cells. The computation of the path goes as follows: For cell c_i on the path, found in the previous iterative step, we compute all neighbours of c_i. Then, we run through the list of all these neighbours and select the cell first appearing in the list, with lowest value of u assigned to it. We wanted to have a look at the outcome of the algorithm, by implementing a prototype with few efforts, using extensively tools already available. The goal was not to make a program that runs as fast as possible. A better sequential implementation, in the C

programming language, running on a powerful PC, will certainly perform much better than the current version. Therefore, we did not compare computing time with results known in the literature yet. However, this first implementation does have a good performance, as we will describe in the following.

The paths computed as results of the experiments, indeed prefer to go along large cells. The number of cells for which neighbours have to be computed in order to find the path is relatively small, namely equal to the number of cells along the path. Since the time complexity depends mainly on the number of times neighbours have to be computed, the computation of the path goes very fast. We could also observe this when comparing the time needed for computing the potential field and the time needed for computing a path.

Experiments show that the time necessary to compute the potential field increases with the number of cells in the decomposition of the C-space. Keeping the total number of cells fix, we observed that the height of the tree (i.e. the refinement of the grid) also has a slight influence on execution time.

We also observed that with dimension higher than 6, the computation of all neighbours of a cell c sometimes takes about 16 milliseconds, that is, some noticable time. The number of directions in which to look for neighbours seems to become significantly large for high dimensions.

The method described above needed about 15 seconds for 4 dimensional cases with 900 cells, about 1.5 minutes for 6 dimensional cases with 2,000 cells, about 23 minutes for 7 dimensional cases with 8,000 cells, about 2 hours for 6 dimensional cases with 15,000 cells, and almost 7 hours for an 8 dimensional case with 32,896 cells.

Of course, the number of cells depends on the dimension of the space, not only for uniform grids but here also. An expedited version can be expected by parallelization of the computation of neighbours, for example by computing neighbours of a cell in each direction by another processor (and to be expected speed up proportional to the dimension of the space, also depending on the number of available processors). The computation of the path could also be made faster by storing in memory which cells are neighbours of each other. However, this would increase drastically the amount of memory needed.

In some real applications, carried out over the last years at the Research Institute for Symbolic Computation, in the fields of geometric modelling and obstacle avoidance, the number of "cells" used was 20 to 50 per object in 3D, and an estimate of 10,000 cells for a quadtree representation. A reasonable amount of cells to be expected seems to be between 5 and 20 cells along each dimension. So, the real cases the sequential program as described above can handle (in less than 2 hours) are limited to 3 dimensions if the number of cells in one dimension is 20 ($20^3 = 8,000$) and to 6 dimensions if the number of cells in one dimension is 5 ($5^6 = 15,625$). However, a speed-up with an estimated factor of 10 to 100 can be expected by implementing the algorithm in C, and by parallelizing the algorithm.

In real applications, the tasks of a robot usually result in moving from a given start position to another place and then returning to the start position. So, as long as the environment of the robot does not change from a geometric point of view, it suffices to compute the potential field u only once, building up u from the start position. A path to different goal positions can be computed with the same u. In this way, the computation of u is part of the modelling phase, and might take several minutes or even hours. The computation of a path however is just a matter of seconds, because it only depends on the computation of neighbours of the cells along the path and because the path consists of the lowest number of cells between start and goal position.

6.7 Conclusion

The approach presented here to solve the general robot motion planning problem, uses a representation of the C-space that does not take a lot of memory. Up to the case of a 6 dimensional C-space, the method presented here is very fast. The resulting path is a pay-off between shortest path in C-space and safest path.

A more expedited time frame can be expected from a better sequential implementation, by just using the programming language C and by writing all programming structures by ourselves. One could also look for another - more involved - algorithm to compute neighbours in different branches of the 2^k-tree, using the GET-representation, or for a more involved way to compute the potential field.

The most time consuming task is the computation of the neighbours; not so much because it would take long to find a neighbouring cell, but for the fact that so many neighbours have to be computed. We therefore are optimistic that a further speed-up can easily be obtained by parallelizing this computation.

We will dedicate some of our time in future to the implementation of a parallel version of this algorithm. Furthermore, we will experiment with assigning other potential fields. When an obstacle has moved slightly, the 2^k-tree representation will also change. Therefore we are especially interested in a potential field that can be adapted in a small environment of such a slightly moved obstacle. We will also look for a potential field that favours a particular direction (i.e. moving a particular joint of a robot arm). It would be interesting to look for a potential function which would result in a path that is a good pay-off between safe and shortest path in the workspace of the robot.

Chapter 7

Hierarchical Approach.

7.1 Grid Construction.

We can easily merge two GET-Rep's for the same manipulator and two different obstacles, as we will describe in the next subsection. We propose to compute GET-Rep for each obstacle in parallel and to merge the resulting GET-Rep's with a parallel cascading strategy. For the same reason we can also restrict ourselves to convex obstacles. In the following subsections we will concentrate on the computation of a GET-Rep for just one convex obstacle.

7.1.1 Merge Two GET-Rep's.

For the computation of the merge of two GET-Rep's, we use the following algorithm. We consider a function dfaTOdfb() to be given, computing for entry i in a DFA, the corresponding entry in the DFB. This function is easy to write.

For both DFA's, each entry in one of them has to correspond to an entry in the other. If such a corresponding entry does not exist we introduce a dummy entry: \$ (in algorithm: [A]).

When a large cell occupied by an obstacle is discovered in one of the DFA's by checking in the DFB's, all children in the 2^k-tree corresponding to the other DFA have to be removed. We replace them by dummy 2^k-tuples, consisting of \$-entries only (in algorithm: [B]).

The resulting DFA entry is computed with a logical OR. For the dummy entry \$ we define: \$ OR \$ = \$, \$ OR 1 = 1, \$ OR 0 = 0, 1 OR \$ = 1, and 0 OR \$ = 0 (in algorithm: [C]).

IN: (DFA1,DFB1),(DFA2,DFB2),

OUT: merge (DFAr,DFBr).

A: introduce dummy 2^k-tuples

*: for (i=0; i < max(length(DFA1),length(DFA2)); i++)

  #: if (((DFA1(i)==0) and (DFB1(dfaTOdfb(i))==1)) or
  ((DFA2(i)==0) and (DFB2(dfaTOdfb(i))==1)))

    *: DFAr(i)=0;

    *: DFBr(dfaTOdfb(i))=1;

    B: adapt next levels: replace in DFA2 or DFA1 resp. the corresponding 2^k-tuple by \$...\$ (2^k times).

  #: else

    C: DFAr(i)=(DFA1(i) OR DFA2(i));

    *: if (DFAr(i)==0)
    DFBr(dfaTOdfb(i)) =
    (DFB1(dfaTOdfb(i)) OR DFB2(dfaTOdfb(i)));

*: return (DFAr,DFBr);

7.1.2 Hierarchical Model of Robot Arm.

In order to construct the non-uniform grid on C-space, we propose a hierarchical method on the robot arm. In each step we consider only one link, starting with the link closest to the base support. In case the first joint is a rotational one and the first link is modelled as a cylinder then for a convex polyhedron as obstacle we can approximate easily the extreme still acceptable values for the angle of the first joint also by a projection method as we will describe further.

For the other links we propose to approximate them by superellipsoids. Superellipsoids are well-known primitives in solid geometric modelling and enclose objects that are rather symmetric with respect to an axis very accurately (see [Bar81] and [Zar85]). The Roider method, introduced in [RS87a] and [RS87b] for 2D and generalized to 3D in [Sti89], [Sti88b], and [Sti92b], allows a very fast collision checking for two convex objects and especially also for two superellipsoids or for a polyhedron and a superellipsoid (see [Sti92a]).

In [Sti92a] a method is given for enclosing polyhedrons (amongst other geometrical objects, see also [Sti88a]). Therefore it suffices to approximate the usually non-convex swept volume by a polyhedron. Fortunately most of the area where the swept volume is locally concave is already swept by the previous link. Since we only want to check for collisions of the i th link in the i th step, we only consider the cells that are free of collisions for the first $i-1$ links and we will only observe collisions or near-collisions for the swept volume of the i th link. In some cases we do not have to consider the whole swept volume of the i th link, since it might happen that a previously studied link has swept part of the area already. When a robot arm is given, we propose to study in a similar way the swept volumes of the links of this robot. This only has to be done once and can be considered as a preprocessing step. The kinematics of robot arms has already been studied in a general way, e.g. [GR88] and [CV95]. We believe that the swept volume of a link can be studied easily and classified geometrically in a few cases so that for all swept volumes classified in the same set the procedure of enclosing them in a superellipsoid is the same. We will address ourselves to this enclosure problem further.

7.1.3 Hierarchical Grid Construction.

So, the hierarchical construction of the non-uniform grid for a robot arm with k DOF's consists of k steps. Each previously not examined part of the swept volume of a link of the robot arm is modelled by one (possibly more) superellipsoids. In the i th step the i th dimension of C-space is added to the non-uniform grid constructed in the previous steps. In the first step the link closest to the base support is considered. For this link the free configurations can be computed adequately and these determine the first subdivision along the first coordinate axis in C-space. In step i we only examine the area that was free up to the $i-1$ th step. We add a dimension, subdivide the new dimension in such a way that it can easily be represented as a GET-Rep (see also the next subsection), and do a collision check for each cell. A cell determines an interval for each of the first i joints, and thus a swept volume of the i th link with respect to the first i joints.

Observe that if we model the last $k-i$ joints by their swept volume and also do a collision check for it (colouring some more cells as forbidden region), we can try to find a path in the part of the C-space that is built up only for the first i joints. If a path is found, it will only need movements in the first i joints. If it is not found, we only need to consider the cells where the swept volume of the last $k-i$ links collides (the 'grey' cells) for further collision checking. After the next step in the construction of the grid we can expand the potential field that was already computed.

Briefly, the policy of our method is that for the first few links the GET-Rep is constructed and U is not. From a certain link on (e.g. the third or fourth of a 6 or 7 DOF manipulator) we also consider the swept volume of the last few links and determine the grey cells and compute U for the first few dimensions, using different increments for different directions (joints). If no path is found yet, a new dimension is added and the already computed U can be extended to the new dimension. As long as no grey cells are computed, all white cells for the first few links have to be investigated. As soon as the grey cells are determined, the white cells are free for all links and thus only the grey cells have to be investigated further.

7.1.4 Hierarchically Building the GET-Rep.

In order to construct the GET-Rep in a hierarchical way, we need two routines. The first one adds an extra dimension to the C-space, the second one processes the information from the computation of new black and white cells in the cells under investigation (the cells for which no collision occurs for the first i links or the cells known to be grey in the previous hierarchical step).

Suppose we start with the GET-Rep of an i-dimensional C-space. Since we want to keep the cell construction symmetric (we want the cells to be $i+1$ dimensional cubes), we first have to 'double' the GET-Rep on each level of the corresponding 2^i-tree. Since we have to 'double' all children of a 'doubled' '1' entry in the DFA of the GET-Rep, we best 'double' the GET-Rep level by level. For example, starting from a 1 dimensional C-space with DFA

10, 00

we double the first level to

1010, 00 00

and then the second level to

1010, 0000 0000.

'Doubling' a level, consists of two actions. First we transform a 2^i-tuple T into a 2^{i+1}-tuple by concatenating two copies of the original 2^i-tuple. And second, we insert on each level a copy of the 2^i-tuples corresponding to the subtrees of the '1' entries in tuple T on the appropriate place. When this is done for all 2^i-tuples on level j, we repeat the process for all 2^i-tuples on level $j+1$. This is the first of the two routines. We illustrate the routine once more with an example of a GET-Rep of a 2 dimensional grid to a 3 dimensional one. Say we start with

1010, 0000 1000, 0000.

Then doubling on the first level results in

10101010, 0000 1000 0000 1000, 0000 0000.

Doubling on the second level results in

10101010, 00000000 10001000 00000000 10001000, 0000 0000 0000 0000.

Finally we also double the third level, resulting in

10101010, 00000000 10001000 00000000 10001000,

00000000 00000000 00000000 00000000.

This process has to be carried out analogously for the DFB of the GET-Rep.

The second operation consists of splitting up some $i+1$ dimensional cells in smaller ones. Here we have to turn some '0' entries in the DFA into '1' entries, corresponding to the extreme values for the $i+1$ th parameter computed during the hierarchical construction of the $i+1$ th dimension of C-space. In the DFB some cells will be marked white and others black.

7.2 U Construction.

As in [BL91] we use a wavefront expansion approach for the computation of a potential field (U) on the non-uniform grid.

A hierarchical construction not only of the GET-Rep but also of the potential field results in a potential field with a slightly different but also very useful meaning than the potential function in our non-hierarchical approach. A path along the descent of this potential field prefers moves of the links closest to the base support. For each link it still has the advantage of being a good pay-off between safe and short path, still preferring the larger sized cells.

The main motivation for thinking of a hierarchical way of constructing U is that then it might happen that we can already find a path in a low dimensional C-space. Since C-space is constructed starting from the joint closest to the fixed link of the robot arm, executing a path found in a low dimensional C-space will result in moves of the first links only. These links are usually the ones that are moved preferably. This planning approach leeds in a natural way to gross motion planning, by specifying the goal position of the robot configuration as a large cell. Additional fine motion could then be planned locally, in and about the large cell that was specified as the goal in the gross motion planning faze. And this can be repeated on line for the not yet needed goal configurations until all free C-space is modelled.

7.2.1 The Hierarchical Way.

In a grid there are three kinds of cells: white, grey and black cells. If we would give a U-value to grey cells, a path could be planned through a grey cell. This can cause the well known problems of planning with grey cells and heuristic approaches, since we are not sure that a path through a grey cell is going to be feasible: this depends on the links that were not yet considered for the construction of the C-space. Theoretically it can even happen that a grey cell turns black completely. Therefore we decided not to give a U-value to the grey cells.

We can start computing a potential field as soon as the initial configuration of the robot is in a white cell. We then can give a U-value to each white cell that can be connected with the cell where the initial configuration is by a path through white cells. Some white cells may not be reached: they might lie in another connectivity component of the free space in C-space or currently unreachable because some grey cell is lying in the way.

When a new dimension is added to the C-space and grey cells turned (partially) into white, grey or black (only grey cells turn into grey ones), we can determine in which cell (for the new dimension) the initial configuration lies.

7.2.2 Algorithm for Extending the Potential Field.

As in the non-hierarchical approach, the potential field is stored in a list like the DFB list. Here the entries are the integer number of the potential field assigned to the corresponding cell in the grid. During the operations for the extension of the grid to the $i+1$ th dimension, this list changes in the same way as the DFB.

Let k denote the number of degrees of freedom of the robot arm and d the maximum number of cells along an axis in C-space. There are two kinds of waves that have to be performed for the U-value assignment. Firstly, during the doubling of white cells that were reachable from the initial configuration, they all get the value of the i-dimensional cell as a temporary value. When the $i+1$-dimensional cell containing the initial configuration is found, we perform a wave starting from that cell but only expanding in the new direction. In all other white areas generated from reachable white cells in the i-dimensional grid, we perform a wave in the new direction only, visiting only cells generated from the same i-dimensional cell, starting from the cell with same coordinate in the new dimension as the cell containing the initial configuration.

Secondly, we perform waves for the assignment of a U-value to all newly reachable white cells, generated from a previously unreachable white cell or from a grey cell. During whitening a cell in a grey area, we look for its white reachable neighbours in the old directions. We assign to this new white cell the smallest U-value of these neighbours, added to the appropriate increment. These new white cells are stored as candidate sources, ordered in sets of cells with the same U-value.

From these sources on we perform a wavefront expansion in any direction, through all previously grey and all previously unreachable white areas. We start with a wave from those candidate sources with smallest U-value. Each cell that inherits a U-value through this wave is removed as candidate source. When the wave stops and some candidate sources are left, we repeat the process.

The increment depends on the direction of inheritance. When a cell inherits its U-value from a neighbour in the j th direction, then the increment is equal to c^{k-j}, where c is a constant which in our examples will be equal to 2.

After these computations, all reachable white cells in the $i+1$-dimensional grid will have a U-value, which finally allows us, to use the same path planning technique as in the non-hierarchical approach, following the descent of the potential field.

Note that we can start computing a potential field in the $i+1$-dimensional subspace of C-space if the projection of the initial configuration of the robot onto the $i+1$ first dimensions is in a $i+1$-dimensional white cell. We then can give a U-value to each white cell that can be connected with the cell where the initial configuration is by a path through white cells. When in the $i+1$-dimensional grid the initial configuration is in a white cell while it was not yet in a white cell in the i-dimensional grid, then the first kind of waves is not performed and in the second kind of waves the cell containing the initial configuration is the only candidate source with potential value equal to '0'.

7.3 Modelling the First Link: Intersection Problem.

7.3.1 Specification of the Problem.

In the previous we already pointed out that, as in our approach, Robot Motion Planning using the Configuration Space (C-space) consists of two subproblems: the construction of C-space and planning a path for a point in C-space. This section tackles the problem of the construction of a grid on C-space, modelling the border of the set of free placements. Here we concentrate on a robot link following a rotational joint.

7.3.2 The Computation of a Grid.

We reduce the 3D intersection problem to a number of calls of a Basic Operation that solves a 2D intersection problem. With this Basic Operation we find candidates for the extreme values of the angle by simple computations on the input.

Modeling Scheme.

We use the same modelling scheme for the robot arm as the one described in [FT88]. In that article, a hierarchical solid modelling technique is used to model the robot arm. A robot arm can be seen as a chain of links, each modelled by solid bodies, each of them connected to the next link by a joint. At one of the ends of this chain, the robot arm is fixed to the environment. We start counting the links and the joints from this fixed link on. That is, we denote A for the robot arm, consisting of links A_0, A_1,..., A_k, where A_{j-1} and A_j are connected to each other by the j th joint (for $j = 1, ..., k$) and A_0 is fixed to the environment (e.g. stands on the floor of the workcell).

On the j th hierarchical level, the links A_0,...,A_{j-1} are modelled by j solid bodies. The other links are modelled by their swept volume with respect to the joints following link A_{j-1}. The cells on the free space in C-space are computed stepwise. In the j th step, the j th dimension of the C-space is added. The $j-1$-dimensional white cells computed in the previous step are expanded to j-dimensional white cells; additional free space is computed for the $j-1$-dimensional black cells. Indeed, the swept volume considered in the previous step has led to area defined as forbidden for all values of the j th parameter, while usually for some values of the j th parameter there is no collision with the obstacle. So, in each step we have to compute, in those areas considered as forbidden in the previous step, the intervals for the j th parameter for which no collision occurs. In [FT88], these intervals are approximated using a heuristic approach for the computation, and regroupment of very small cells is necessary for the construction of the generalized quadtree. We approximate the cells geometrically, immediately looking for the GET-representation, and keeping track of the rate of approximation of the obstacle by the grid. Another difference with the approach in [FT88] is, we do not allow the robot to access partially free cells ("grey cells") in our planning phase.

For simplicity reasons, we consider the obstacle in 3D workspace to be a convex polyhedron. We assume that the links of the robot arm are modelled by cylinders. In the j th step, we are looking for a safe but accurate approximation for particular extreme values of the j th parameter, for each $j-1$-dimensional black cell. These extreme values will be the endpoints of the intervals along the boundary of the j-dimensional white cells, in the j th direction. In order to find these extreme values, we have to compute an approximation of the values of the j th parameter for which an intersection occurs. We simplify this problem in the following way. We have to consider the intersection of the polyhedral obstacle first with the cylinder modelling the j th link and secondly with the swept volume of the following links.

Intersecting a Polyhedron with a Cylinder.

The cylinder rotates around an axis through the center of one of its circular faces. For simplicity reasons we assume that the axis lies in the plane of the face, but for successful use of the approach we will describe this is not a necessary condition. Consider the center of this circle to be the origin of a fixed coordinate frame, and the rotation axis to be the Y-axis. We choose the symmetry axis of the cylinder to be the Z-axis and we choose the X-axis in such a way that we get a righthand oriented coordinate frame (see Figures 7.1 and 7.2).

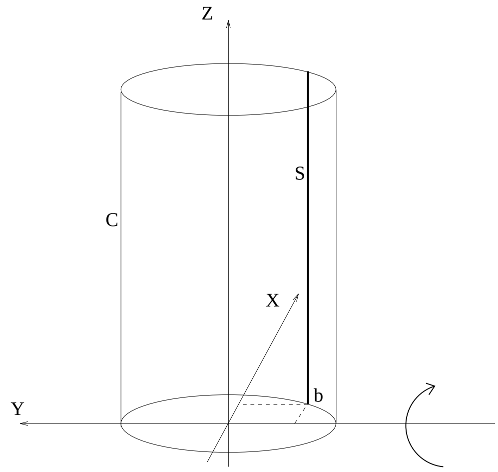

Figure 7.1: Rotating Cylinder C.

We then consider the interval along the Y-axis that intersects the cylinder. We perform a subdivision strategy along this interval in order to find the extreme values for the angles of rotation for which intersection occurs. For each step in the subdivision, we look for intersections in a plane perpendicular to the Y-axis.

The 2D Intersection Problem. In each step of the subdivision, we choose some particular y-value, and we only consider what happens in plane Φ parallel to the Z-axis and through the point on the Y-axis with the chosen y-value.

The intersection of plane Φ with the cylinder is a segment S parallel to the Z-axis. (Due to symmetry, we only consider wlog. the halfplane with positive x-coordinates.) We compute the distance in plane Φ of the endpoints of this segment to the origin. For the endpoint b lying in the XY-plane, this is just its x-coordinate x_b. We denote the other value by r_{max}.

When rotating the cylinder, each point p of segment S describes a circle $C(o, r)$ with center in the origin and radius the distance r for p to the origin. All possible values for r are between x_b and r_{max}. Figure 7.3 shows the situation in plane Φ.

The intersection of plane Φ with the polyhedron can be empty, or some polygon (possibly just one point). If the intersection is non-empty, some interesting points w on the polygon are the vertices lying on a distance from the origin between x_b and r_{max}, and the intersection points of the edges adjacent to only one such a vertex with the circle $C(o, r_{max})$ or with the circle $C(o, x_b)$. In the following, we denote the finite set of these points w by W.

Observe that for a given point w in the circular region swept by S when rotated, there is exactly one point s on the segment that collides with a given point w during the sweep. Indeed, this point s has coordinates $(x_b, y_b, \sqrt{r^2 - x_b^2})$, where r is the radius of the circle with center in the origin and through w. So, for any points w on the polyhedron, we can easily determine the unique angle α_w for which the segment, rotated over that angle, intersects with the polyhedron at w. This computation is the Basic Operation in our approach.

We could also find the extreme values for the angle in plane Φ. Since W is a finite set, we can easily determine the extreme values among the angles α_w corresponding to points w in W.

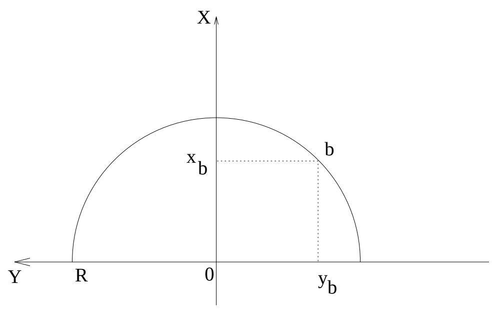

Figure 7.2: Situation in the XY-plane.

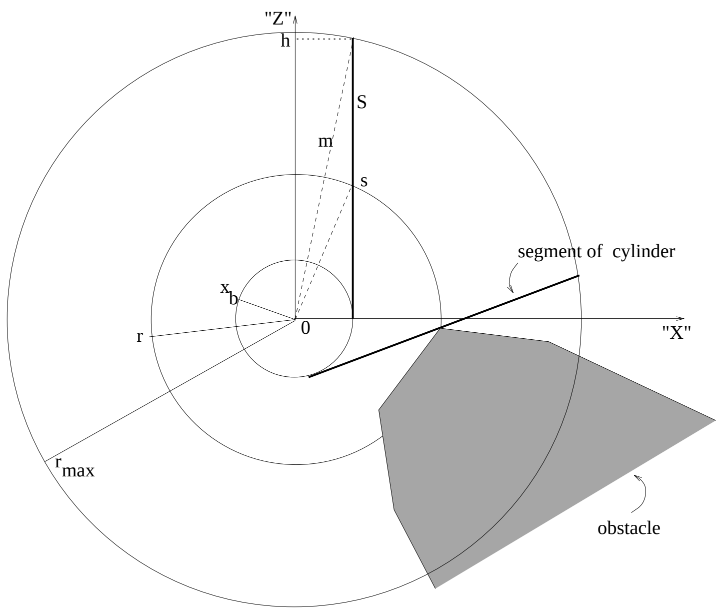

Figure 7.3: Line L on the cylinder sweeps in plane Φ.

Subdivision. In the following, we will need some notations. Let C denote the cylinder in its initial position (angle of rotation $\alpha = 0$), C_α for the cylinder rotated over angle α, P the polyhedron and e an edge of P. Let us write R for the radius of the circle at the basis of cylinder C. Then we start off with the interval [-R,+R] along the Y-axis.

Note that an intersection of P with an infinite cylinder, when the latter rotates, happens first at an edge of P. For the finite cylinder C, the first intersection might also happen at a face. This observation was already encountered in the previous: in the 2D intersection problem we mentioned the vertices of the slice that is cut out of P by plane Φ (these correspond to edges of P) and the intersection points of this slice with the circles $C(o, x_b)$ and $C(o, r_{max})$ (these correspond to the faces of P).

In the following, in the case of a first intersection with a face F, we have to consider an artificial edge e being the intersection of F with the plane through the circle at the top or the bottom of C_α (respectively for the cases where the contact was made with $C(o, x_b)$ or $C(o, r_{max})$). We can now make a few observations.

Observation 1 *Suppose that the cylinder intersects polyhedron P in vertex v of P, for angle α, when rotating by increasing the angle. Let e be an edge of P incident to v. Then three cases can occur:*

case 1: e intersects the cylinder "earlier" than v, i.e. for smaller angle than α (this is the case in which we are interested most).

case 2: e intersects the cylinder "later" than v, i.e. for larger angle than α.

case 3: e does not intersect the cylinder for any angle.

Observation 2 *In the same settings as above, the three cases can be distinguished from each other. The relative position of edge e w.r.t. cylinder C_α must be one of the following three possibilities:*

pos. 1: The line through edge e intersects the cylinder in more than one point but only v is on e.

pos. 2: The edge e intersects the cylinder in at least one point p other than v ($p \neq v$).

pos. 3: The line through e is a tangent line of C_α in v.

Possibility 1 corresponds to case 1, possibility 2 corresponds to case 2, possibility 3 corresponds to case 3.

A similar observation can be made for points anywhere on an edge of the polyhedron:

Observation 3 *Let Φ_1 be a plane parallel to the XZ-plane, intersecting cylinder C in line L. Let α_1 be the angle for which line L, when rotating from low to high values for the angle, touches the polyhedron for the first time (a result of the computation as described in the previous subsection). Let p_1 be the point of intersection of L and edge e of the polyhedron. Then we can distinguish two kinds of positions of e w.r.t. the rotated cylinder C_{α_1}:*

- *$e \cup C_{\alpha_1} = \{p_1\}$: then p_1 is the first intersection of the rotating cylinder with edge e and α_1 is an extreme value for the angle along edge e.*
- *$|e \cup C_{\alpha_1}| > 1$: then there must be another point p_2 ($\neq p_1$) along edge e for which the rotating cylinder intersects edge e at angle α_2, where $\alpha_2 < \alpha_1$.*

Observation 4 *Let P be a polyhedron, let C denote a cylinder. Denote I_P for the interval of y-coordinates of the points in P, I_C for the interval of y-coordinates of the points in C. When P intersects C, we distinguish the following situations and define the "search region" SRY accordingly:*

- *$I_P \subset I_C$, define $SRY = I_P$,*
- *$I_C \subset I_P$, define $SRY = I_C$,*
- *non-empty intersection, but not one of the cases above, define $SRY = I_P \cap I_C$.*

Intersections between the rotating cylinder and polyhedron P can only occur in points of P and C with y-coordinate in "search region" SRY. The interval SRY is always contained in $[-R,R]$. The y-coordinates of points on an edge or a face of P may all ly in SRY. If this is not the case, then we say that the edge or face is "clipped by C".

Observation 5 *When sweeping plane Φ (defined as in the previous subsection) in the y-direction along interval SRY, the angle for which first contact occurs changes due to two changes:*

- *the intersection of Φ with the cylinder,*
- *the intersection of Φ with polyhedron P.*

These contributions to the change of the extreme value along an edge may annihilate each other when sweeping along the edge. At a vertex of P, encountered by Φ sweeping by, a dramatic change can be expected. The edge (or artificial edge) where the first contact occurs on will change only when one of its endpoints (a vertex or an artificial vertex of P) is reached by the sweeping plane Φ.

The observations of above lead to the following algorithm.

Step 1: Compute SRY.

Step 2: Subdivide SRY in intervals $[y_i, y_{i+1}]$ such that all (artificial) edges of P with at least one point with y-coordinate in SRY are intersected by at least one plane Φ_i (parallel to the XZ-plane, with y-coordinate one of the y_i's).

Step 3: Compute for each Φ_i the extreme values α_i (where first contact occurs in Φ_i) with the Basic Operation. We now have for each of these α_i's a corresponding edge (or artificial edge) e_i of P.

Step 4: Find for each α_i the endpoints of the interval of extreme values around it, when sweeping a plane Φ along the corresponding e_i. When edge e_i is clipped we only have to consider the part of e_i with y-coordinates in SRY.

Step 5: Find among all these endpoints the smallest and largest value for α.

Since the polyhedron is convex, we have exactly one minimum and maximum angle. The size of the smallest segment considered for the angle is a measure for its accuracy. In Step 3, we check all edges with some point with y-coordinate in SRY because we do not know a priori which edges are going to give rise to the extreme values of α. In Step 5, we only have to find the minimum and the maximum value in a finite set of numbers. We will concentrate now on Step 4.

Step 4. We propose two different approaches for the computation in Step 4. In the first approach the problem becomes discrete, the other approach is analytic with a difficult symbolic computation problem.

Discrete. We must make sure that the value for α is a safe one, because we want to be sure that the free cells in the non-uniform grid are really empty. I.e. we have to know about the behaviour of the values for α along edge e when sweeping plane Φ along. We can use Observation 3 for this purpose.

Indeed, define plane Ψ through edge e and through the over α rotated segment S. Plane Ψ is tangent to the cylinder iff α is an extreme value along e. Thus, for each (artificial) edge e, we can proceed in the following way.

i: Take arbitrary point w on edge e (say, w is the midpoint of interval $[p_a,p_b]$, where p_a and p_b are the endpoints of e).

ii: Compute α_w for this point w (using the notation of the previous subsection).

iii: Rotate points p_a and p_b over angle $-\alpha_w$, resulting in points e_1 resp. e_2.

iv: We now check whether the line through e_1 and e_2 is tangent to cylinder C. We can do this check in the XY-plane by projecting the points e_1 and e_2 and determining whether the line through their projections is tangent to the circle at the bottom of the cylinder. So, project e_1 and e_2 to the XY-plane, compute the square of the distance of the line E through these projected points to the origin and compare the result with R^2 (the radius of cylinder C). This is shown in Figure 7.4. If these values are identical, α_w is the extreme value for angle α along edge e.

v: If these values are different, then the extreme value for α along edge e occurs at another point w on e.

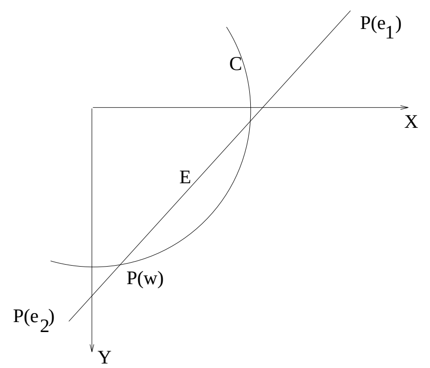

Figure 7.4: Check for tangency after projecting to the XY plane.

We now concentrate on this last step. If indeed point w, chosen as above, is not the good one, we adapt α_w by a constant $\sigma > 0$. We can consider the angle $\beta_1 = \alpha_w - \sigma$ as a new candidate for the extreme value. We now proceed with a procedure similar to steps [ii] to [v], but with fewer computational effort and without computing the to β_i $(i = 1, 2, ...)$ corresponding point $\hat{w}$. That is, we compute the projection E_i of edge e after it has been rotated over $\beta_i = \beta_{i-1} - \sigma$ (where $\beta_0 = \alpha_w$), and check for the number of intersection points with the circle. If there is one intersection point, then we reached the extreme value for the angle. If there are two intersection points, then we repeat the adaptation of α_w (we consider $\beta_{i+1} = \beta_i - \sigma$). If there is no intersection, we repeat the adaptation of α_w in the opposite direction by $\frac{1}{2}\sigma$ (we consider $\beta_{i+1} = \beta_i + \frac{1}{2}\sigma$). Obviously, we again use the technique of computing the distance from the origin to E_i, in order to determine which of the above cases apply. We should keep in mind that here we compute numerically the distance and the comparison with radius R, so some inaccuracy in the data may occur.

In order to compute E_i, we can use a formula. Rotating p_a and p_b over angle κ and then projecting the resulting points on the XY-plane, we get two dimensional coordinates

$$(x_{pa} \cos\kappa - z_{pa} \sin\kappa, y_{pa}) \text{ and}$$

$$(x_{pb} \cos\kappa - z_{pb} \sin\kappa, y_{pb})$$

respectively. So, the slope s_i of E_i in the XY-plane is then:

$$s_i = \frac{y_{pb} - y_{pa}}{(x_{pb} - x_{pa})\cos\kappa - (z_{pb} - z_{pa})\sin\kappa}.$$

And thus, the equation for E_i is then:

$$(y - y_{pa}) = s_i\ (x - (x_{pa}\cos\kappa - z_{pa}\sin\kappa)).$$

Having this equation, we can now compute the distance of the origin $(0, 0)$ to E_i:

$$d((0,0), E_i) = \frac{y_{pa} - s_i\ (x_{pa}\cos\kappa - z_{pa}\sin\kappa)}{\sqrt{(y_{pb} - y_{pa})^2 + ((x_{pb} - x_{pa})\cos\kappa - (z_{pb} - z_{pa})\sin\kappa)^2}}.$$

The value of σ will be chosen appropriately to get an adequate subdivision for a 2^k representation of the C-space and thus suited for the GET-representation.

Symbolic. The problem of finding the appropriate α can be reduced to the problem of finding a minimum of a function f. The angle α can be written as a function of some parameters. An extreme value for α is then also an extreme value for that function f. In the following, such a function f is constructed. However, we did not find a suitable way for the determination of the extreme points of f.

Given the coordinates of the two (artificial) endpoints of edge e, and an arbitrary point p on e in between these two endpoints, we can compute the equation of plane Φ through p and parallel to the XZ-plane. We parameterize p in function of the two endpoints p_a and p_b:

$$p = \lambda p_a + \mu p_b,$$

where $\lambda, \mu \in [0,1]$. For this plane Φ we can compute the extreme values for angle α. Doing so, we get a function f in two variables λ and μ. The minimum and the maximum value of f for $\lambda, \mu \in [0,1]$ are the extreme values of α along edge e. This can be done as a preprocessing step.

However, f is a highly non-linear function. Indeed, when we write (x_{pa}, y_{pa}, z_{pa}) and (x_{pb}, y_{pb}, z_{pb}) for the coordinates of points p_a and p_b respectively, then the equation of plane Φ through point p and parallel to the XZ-plane is

$$\Phi : y = \lambda y_{pa} + \mu y_{pb}.$$

Line L being the intersection of Φ with C gives us an equation for x_b:

$$x_b^2 + (\lambda y_{pa} + \mu y_{pb})^2 = R^2.$$

Let r be the radius of the circle in Φ as in the algorithm of the previous subsection, on which the contact point p lies. That is,

$$r = d(o_\Phi, p), \text{ where } o_\Phi = (0, \lambda y_{pa} + \mu y_{pb}, 0).$$

Observe that $r \in [x_b, r_{max}]$. Let $\hat{r} = \sqrt{r^2 - x_b^2}$, and point $c = (x_b, \lambda y_{pa} + \mu y_{pb}, \hat{r})$. We know that point p is the point reached by point c after rotation by angle α. The angle α is the angle between the lines $o_\Phi p$ and $o_\Phi c$ in plane Φ.

We now express α as a function in λ and μ. Let γ be the angle such that

$$\tan \gamma = \frac{\lambda z_{pa} + \mu z_{pb}}{\lambda x_{pa} + \mu x_{pb}}.$$

(The angle γ can be seen as the angle of rotation from one coordinate frame to another needed for a straightforward computation of $\tan \alpha$.) Then

$$\tan \alpha = \frac{x_b \sin \gamma - \hat{r} \cos \gamma}{x_b \cos \gamma + \hat{r} \sin \gamma}$$

$$= \frac{x_b \tan \gamma - \hat{r}}{x_b + \hat{r} \tan \gamma}.$$

This gives us the function f. (In fact, we can use $\tan \alpha$ instead of α as the parameter for the angle of rotation of the joint of the robot, when we assume that the rotational joint cannot rotate freely over 2π.)

Intersection of a Polyhedron with a Swept Volume.

The discrete method in the previous is a good method for the first link and rotational joint, the symbolic method seems to be too involved. In the following we will describe another approach for the decisions on the "extreme angles" for swept volumes. However, the swept volume is a more complicated object considered as geometric model. Therefore the approach of above will lead to some more complicated computations. Instead, we will approximate the swept volumes by superellipsoids and use collision detection for the construction of the grid.

7.4 Enclosing Robot Links By Superellipses and Superellipsoids.

Superellipses and superellipsoids are useful and extremely versatile primitives for constructive solid models. They have been studied thoroughly in the literature (see [Bar81], [Zar85]). A particular application of these solids in collision detection in robot simulation systems is described in [Sti92a], where the so called Roider Method for collision checking is used (see also [Sti88b]).

As described in the previous, we want to construct a non-uniform grid over the Configuration Space of a robot arm, and therefore have to be able to decide whether or not in a given cell there is a configuration for which the robot arm collides with an obstacle. In order to do this collision check with the Roider Method for superellipses (in the 2-dimensional case) or superellipsoids (in the 3 dimensional case), we have to enclose the area swept by a link of the robot arm (when joints are moved over the intervals determining the given cell) by a superellipse or superellipsoid. On the grid we put a potential field in order to solve the motion planning problem for a robot arm (for more details on our planning approach, see [VG94a]).

In [Dit87], [Sti88b], and [Sti88a] the problem of enclosing cylinders, cones, etc. is discussed. In essence, given a particular object, one just takes a set of *extreme points* whose convex hull enclose the object, and then puts a superellipsoid around these points. The same approach can be used here, for the swept volumes in which we are interested. We assume that the robot arm is built up as in the previous section, that is, as in [FT88].

We will describe in very precise way how to proceed for the swept volumes of some particular kind of robot topologies. Our main goal in doing so was to prove our feeling that it is possible to find good enclosing superellipses or superellipsoids using some easy geometrical and trigonometrical tools. We need particularly good enclosures for the application we have in mind, because too much area within the superellipses or superellipsoids will give rise to too many blocked cells in the grid over Configuration Space, and in consequence there would be cases in which we would not be able to find a path although it exists during motion planning. We do not feel satisfied with general approaches for modelling the swept volumes.

We will also compare the volume of the swept area with the volume of the enclosing superellipse or superellipsoid. The rate between them is a measure for the accurateness of the enclosure.

In Subsection 2 we consider a robot arm with two links and two rotational joints in 2-dimensional space. We study the swept volume of the second link. In Subsection 3 we describe the extreme points for the enclosure of the swept volume of the second link of that robot topology. In Subsection 4 we consider a robot with a rotational joint at the base support, and then three more joints: a translational joint followed by a rotational joint and finally a translational joint. We also describe the extreme points for this topology in Subsection 5. In Subsection 6 we compute the volumes of the enclosing superellipses, and the volumes of the swept areas for both the topologies considered in the previous subsections. In Subsection 7 we consider the same topologies as before, but this time we model the links by rectangles instead of by line segments. We determine the extreme points and compute the volumes of the swept areas. In Subsection 8 we thicken the rectangular links so that they become cylinders. We then are in a three dimensional case, and thus have to enclose the swept area by a superellipsoid. The joints rotate as before in a plane. We again look for the extreme points and compute the volume of the swept area.

7.4.1 Shapes of the Swept Volume of a Second Link in 2D.

In the following we will use the notation as introduced in Figure 7.5. We write o for the point at the base support, the second joint is placed in point a and rotated over angle α to point d, the furthest point of the second link is positioned originally in point b, which is rotated over angle β to position c. Points b and c are rotated over angle α to points e and f respectively. We denote ζ for the angle between lines oa and ab, and ξ for the angle between lines oa and ac, so $\xi = \zeta + \beta$. There are four different cases for

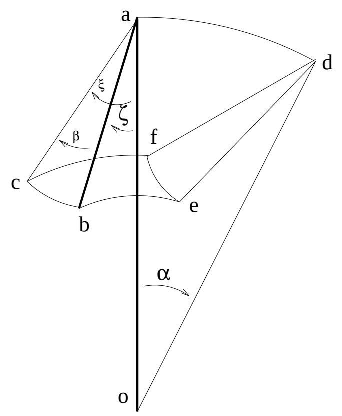

Figure 7.5: Notation.

the shape of the swept volume. First of all notice the symmetry w.r.t. $\zeta + \xi = 2\pi$ (measuring the angles in the opposite direction, interchanging the roles of ζ and ξ). This reduces the cases by half. We further also assume that $\zeta - \xi < \frac{\pi}{2}$, which is a reasonable assumption for our application.

We can determine in which case we are in the following way, using the law of cosines for arbitrary triangles: Suppose ξ and ζ are both smaller than π. Then:

- For small values for angle ζ, point f is in the inside of the swept volume. If $\cos\xi \geq \frac{m}{2l}$ then we are in case 1,

- When ζ becomes sufficiently large, point a will lie in the interior of the swept volume and point f will lie on the boundary. If $\cos\xi < \frac{m}{2l}$ and $\cos\zeta \geq \frac{m}{2l}$, we are in case 2,

- When ζ is large, point a will lie on the boundary again and point e moves to the inside. If $\cos\zeta < \frac{m}{2l}$ we are in case 3,

- If $\cos\xi = \frac{m}{2l}$, then c and a sweep along the same line and thus we can proceed as in case 1,

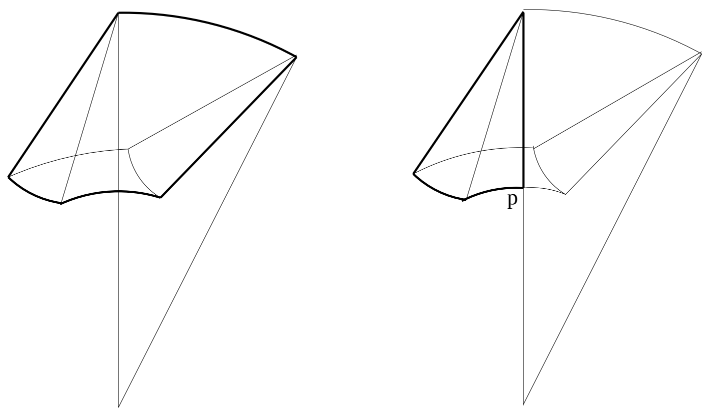

Figure 7.6: The first case: swept volume of second link, and point p.

- If $\cos\zeta = \frac{m}{2l}$, then a is on the same circle with center o and radius $\hat{n}$, where $\hat{n}$ is the distance between o and b. So this case is identical to case 2.

If $\frac{\pi}{2} < \zeta < \pi < \xi$, we are in case 4.

We now prove the completeness of this case distinction by moving ξ and ζ along the unit circle and checking all possible cases. As in the previous we assume $\zeta - \xi < \frac{\pi}{2}$. In the first quadrant of the unit circle, for $0 < \zeta < \xi < \frac{\pi}{2}$, we see that $\cos\zeta > \cos\xi$. So, when $\cos\xi \geq \frac{m}{2l}$ we are in case 1. Since l and m will usually be of the same order of magnitude, $\frac{m}{2l}$ might be smaller than 1 rather often. For $\zeta, \xi < \frac{\pi}{2}$ it can happen that $\cos\xi < \frac{m}{2l}$, while $\cos\zeta \geq \frac{m}{2l}$. Then we are in case 2. When we increase the angles so that ξ ends up in the second quadrant of the unit circle, we get for $\xi > \frac{\pi}{2}$, $\cos\xi < 0 < \frac{m}{2l}$. Then we can have $\cos\zeta \geq \frac{m}{2l}$, corresponding to case 2. When instead $\cos\zeta < \frac{m}{2l}$, we are in case three. When both angles are in the second quadrant, that is $\frac{\pi}{2} < \xi < \zeta < \pi$, we naturally are in case 3. When $\frac{\pi}{2} < \xi < \pi$ and $\pi < \zeta < \frac{3\pi}{2}$, we end up with case 4.

7.4.2 Enclosing the Swept Volume of a Second Link in 2D.

We now come to the way how to enclose the shapes of the swept volume of the second link in the four essentially different cases:

case 1: First shape. Find point p, the intersection of line oa and circle segment bc (as in Figure 7.6; find point q, the intersection of the tangent in c to circle segment bc; find the superellipse (as in [Sti92a], [Sti88a]) enclosing the polygon $capbq$. The locally concave area and the rest of the swept volume are swept by the first link already.

For small values for α, the line oa does not intersect circle segment be. In that case we just take the polygon $bqcape$, where p now stands for the point of intersection of lines oa and de.

For uniformity to the other cases we can also consider the polygon $bqcarde$, where r is the intersection of the tangent to curve segment ad in a and the tangent to curve segment ad in d.

case 2: Second shape. Find point p, the intersetion of circle segment be and line oa. We denote s for the intersection of the tangent to curve segment cf in c and the tangent to curve segment cf in f. We denote t for the intersection of the tangent to curve segment bc in b and the tangent to curve segment bc in c. If the polygon $dpbtcsf$ is convex, then we use it, else we use polygon $debtcsf$. This check has to be performed only on the area d, p, b. Find the enclosing superellipse for the polygon. Remark: There is a nasty area created by the fact that c and f are not on circle segment ad but lie above it, i.e. an area not swept by the first link. Therefore we prefer modelling the whole area swept by the second link in this case.

Again for small values for angle α oa does not intersect circle segment be. We then just consider the polygon $debtcsf$.

case 3: Third shape. Find point p, the intersection of the tangent in b of circle segment bc and the tangent in c of circle segment bc; q the intersection of the tangent in c on circle segment cf and the tangent in f on circle segment cf. Enclose polygon $adfqcpb$ by a superellipse.

case 4: Fourth shape. Find point s, the intersection of the circle segment bc and the line oa. Find t, the intersection of the circle segment ef and the line od. Denote p_1 for the intersection of the tangent to circle segment bs in b and the tangent to circle segment bs in s. Denote p_2 for the intersection of the tangent to circle segment st in s and the tangent to circle segment st in t. Denote p_3 for the intersection of the tangent to circle segment tf in t and the tangent to circle segment tf in f. Enclose the polygon $bp_1sp_2tp_3fda$ by a superellipse.

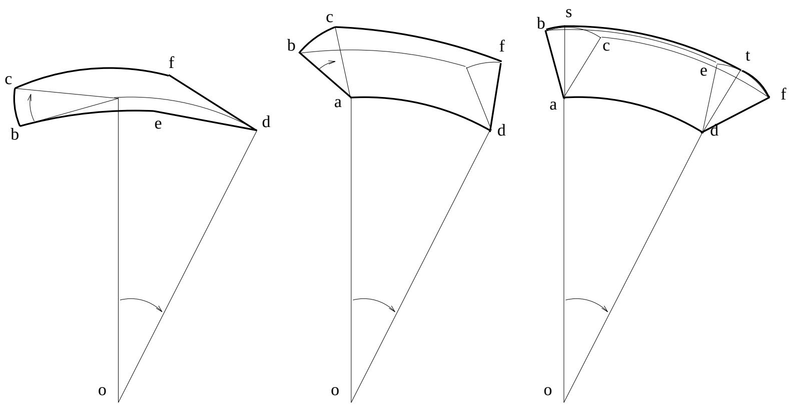

Figure 7.7: The cases 2, 3 and 4.

The vertices of the enclosing polygons constructed in the four cases are the extreme points for the algorithm in [Sti92a]. However, also the direction of the axis of the cylinder from which the superellipse is constructed, is free to choose in that algorithm. We suggest to choose for case 1 the line through the centroid of the extreme points and parallel to ab, and in the other cases the line through the centroid of the extreme points and parallel to ad.

7.4.3 Shapes of Swept Volume of a Third and Fourth Link in 2D.

Only small adaptations of the previous have to be made for extending it to two rotational joints alternated with two translational joints, in 2-dimensional Workspace.

We will denote the shortest possible distance between points o and a by l, between a and b by m. Any additional distance created by a move in the first translational joint will be denoted by l_e, and when created by a move in the second translational joint by m_e.

The first rotational joint is responsible for the creation of other shapes. Considering a coordinate frame in o with X-axis along the first link and Y-axis perpendicular to it. When $l_e = 0$, we have the shapes for the swept volume as in the previous subsections. The extreme point g of that swept volume w.r.t. the Y-coordinate will give rise to a different shape in all four cases discussed in the previous subsections. In the following we will denote $\tilde{x}$ for the point to which x is swept by a move over distance l_e in the first translational joint, $\bar{x}$ for the point to which x is swept by moving the first rotational joint over the total angle range considered. The notations are now like in Figure 7.8. We will denote x' for the point to which point x is swept by moving the second translational joint over a distance m_e.

Ignoring the first rotational joint for now, each point in the swept volume of the third link over the considered angle range for the second rotational joint will be swept over an interval of length l_e parallel to the first two links. This happens in each position of that swept volume, when it is swept over the considered angle range for the first rotational joint. Roughly, the resulting swept volume is thickened by l_e compared to the previous case. Starting from the four cases found in the previous subsections, we come to the four cases as depictured in Figure 7.8, of which easily can be checked they are the only cases up to some symmetry considerations.

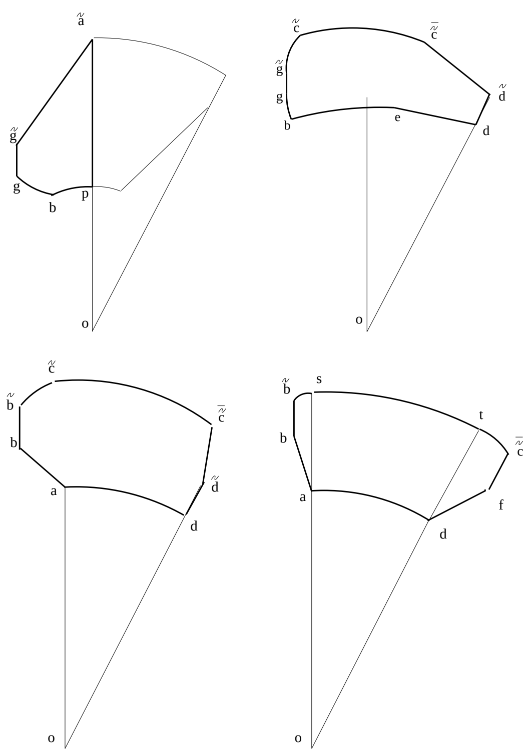

Figure 7.8: The four cases for third (and fourth) link.

To conclude this subsection, we observe that the second translational joint does not induce any change to the shape of the swept volume. Indeed if we assume that the initial distance between a and b was m, we then computed the swept volume and then want to take into account a change in the distance between a and b by m_e, then we will end up with the swept volume as if we would have computed it with initial distance between a and b equal to $m + m_e$. So by this we are also done with the fourth link.

7.4.4 Enclosing the Swept Volumes of a Third and Fourth Link in 2D.

For the third and fourth link, we have still four cases for the shapes, but the sets of extreme points (vertices of an enclosing polygon) are different. For the third link we get the following situation.

case 1: First Shape. Since $\xi, \zeta < \frac{\pi}{2}$, g has to be c. Take the polygon $bqc\tilde{c}\tilde{a}p$, where q is the intersection of the tangent to bc in b and the tangent to bc in c.

For small values for α for which oa will not intersect circle segment be, we consider $bc\tilde{c}q\tilde{a}pe$, where now p stands for the intersection point of lines oa and de.

case 2: Second Shape. Again, if b, p, d makes a convex area, we use polygon $dpbg\tilde{g}\tilde{c}\bar{\tilde{c}}\tilde{d}$, else we use $debg\tilde{g}\tilde{c}\bar{\tilde{c}}\tilde{d}$. Note that $\bar{\tilde{c}}$ can also be written as $\tilde{f}$.

Again for small values for α for which oa will not intersect circle segment be, we consider here the points $b, g, \tilde{g}, \tilde{c}, \bar{\tilde{c}}, \tilde{d}, d, e$.

case 3: Third Shape. Take the polygon $dab\tilde{b}p\tilde{c}q\bar{\tilde{c}}f$, where p is the intersection of the tangent to $\tilde{b}\tilde{c}$ in $\tilde{b}$ and the tangent to $\tilde{b}\tilde{c}$ in $\tilde{c}$, and q the intersection of the tangent to $\tilde{c}\bar{\tilde{c}}$ in $\tilde{c}$ and the tangent to $\tilde{c}\bar{\tilde{c}}$ in $\bar{\tilde{c}}$. Note that g now coincides with b.

case 4: Fourth Shape. Take the polygon $dab\tilde{b}st\bar{\tilde{c}}f$, where s is the intersection of $\tilde{b}\tilde{c}$ and oa, and t is the intersection of $\tilde{b}\bar{\tilde{c}}$ and od. Note that g now coincides with b.

Considering the axis of the cylinder, we suggest to choose for case 1 the line through the centroid of the extreme points and parallel to ab, and in the other cases the line through the centroid of the extreme points and parallel to ad, as in the previous.

For the fourth link we replace x by x' in the previous, for $x \in \{b, c, e, f\}$.

7.4.5 Volume: Error Estimate.

In order to get an impression of the accuracy of the approximation of the swept volume of a link by the superellipse we constructed in the previous subsections, we can compute the volumes of both the swept volume of the link and of the superellipse and compare them.

Volume of Superellipse.

We will derive a formula for the volume of a superellipse, in the way done in [Zar85] for superellipsoids. A superellipse is determined by its center and three parameters A, B, and n. The superellipse centered in the origin of a carthesian coordinate system is given by the set

$$\{(A\cos^{1/n}\psi, B\sin^{1/n}\psi)\ ;\ \psi \in [0, 2\pi[,\}$$

for fixed A, B, n. This also corresponds to the equation $\left(\frac{x}{A}\right)^{2n} + \left(\frac{y}{B}\right)^{2n} = 1$.

Introducing the polar coordinates r, θ by

$$\begin{cases} \bar{x} = \frac{x}{A} = r\cos\theta, \\ \bar{y} = \frac{y}{B} = r\sin\theta, \end{cases}$$

we find the equation

$$(r\cos\theta)^{2n} + (r\sin\theta)^{2n} = 1,$$

from which we find for r:

$$r = (\cos^{2n}\theta + \sin^{2n}\theta)^{\frac{-1}{2n}}.$$

Taking slices of equal size for θ, the area of such a slice equals $\frac{r^2 d\theta}{2}$ and therefore the area of the superellipse will be equal to

$$8\int_0^{\frac{\pi}{4}}((\cos^{2n}\theta + \sin^{2n}\theta)^{\frac{-1}{n}})d\theta.$$

For particular values of n a sufficiently good approximation of this integral is computed by the computer algebra system Maple. This result has to be multiplied still with A and B in order to get the volume of the superellipse enclosing the swept volume.

In order to be able to make use of this formula we have to determine A, B and n for the superellipses constructed in the previous subsections. If we allow a tolerance ϵ as in [Sti92a], the algorithm there (simplified doing only the step for a superellipse) will provide a value for A, B and n.

Volume of Swept Volume for Second Link in the 2 DOF Case.

We now come to the computation of the volume taken by the swept volume of the second link. In the following we will denote $Vol(SV_\delta([u, v]))$ for the volume taken by sweeping the interval $[u, v]$ over angle δ revolving around center u. Denoting D for the length of interval $[u, v]$, this area is equal to $\frac{\delta}{2}D^2$.

case 1: Here the volume is equal to $Vol(SV_\beta([a, b])) + (\ Vol(\Delta a, b, o) - Vol(SV_\gamma([o, b])))$, where $Vol(\Delta a, b, o)$ is the volume of the triangle with vertices a, b and o and γ the angle between lines oa and ob.

For small angles a slight adaptation has to be made to this computation. In this case we get the volume by $Vol(SV_\beta([a, b])) + Vol(\Delta abo) - (Vol(SV_\alpha([o, b])) + Vol(\Delta oep))$, where p now is the intersection of lines de and oa.

case 2: Let q be the intersection point of circle segment cf and line oa, p the intersection point of oa and circle segment be as in the previous. Then the volume is equal to ($Vol(SV_\alpha[o, q])$ - $Vol(SV_\alpha[o, p])$) + $Vol(SV_\beta([a, b]))$. This indeed is the volume since the area enclosed by ac, cq and qa is equal in size to the area enclosed by fd, $d\bar{q}$ and $\bar{q}f$ where $\bar{q}$ is the point to where q would be swept when swept over angle α (and analogous for a, b, p).

For small angles we can compute the volume by:
$Vol(SV_\beta([a,b])) + (Vol(SV_\alpha([o,c])) - Vol(SV_\alpha([o,b])))$.

case 3: Denote q for the intersection point of line oa and circle segment cf. Then the volume is $Vol(SV_\alpha([a,q]))$ + $Vol(SV_\beta([a,b]))$. Again we use the similar property that the area between a, c, and q is equal to the area between d, f, and $\hat{q}$ where $\hat{q}$ is the point to which q would be swept when swept over angle α.

case 4: The volume now can be computed by $Vol(SV_\alpha([a,s])) + Vol(SV_\beta([a,c]))$ (similar to case 3).

Volume of Swept Volumes for Third and Fourth Link in the 4 DOF Case.

We can use the same technique here as in the previously described cases for the second link. In the following we will describe the volume computation for the fourth link. For the volume computation for the third link we just have to drop the accent ($'$) in the text.

case 1: Splitting up the volume to be computed in three now, we get similar to the 2 DOF case the volumes $Vol(CS_\beta([a,b']))$ and $Vol(\Delta ab'o) - Vol(SV_{\gamma'}([o,b'])$, where γ' is the angle between oa and ob', and the additional volume $Vol(\diamondsuit c'\tilde{c}'ha)$. The area of that parallelogram is equal to $l_e(m+m_e)\sin\xi$, since the sharp angle of the parallelogram is equal to ξ.

For small angles we compute the volume by formula: $Vol(SV_\beta([a,b])) + Vol(\Delta abo) - (Vol(SV_\alpha([o,b]))$ + $Vol(\Delta oep)) + Vol(\diamondsuit ac\tilde{c}\tilde{a})$. Again the area of the parallelogram is equal to $l_e(m+m_e)\sin\xi$.

case 2: The volume of the swept volume can be computed by $(Vol(SV_\alpha([o,q]))$ - $Vol(SV_\alpha([o,p])))$ + $Vol(SV_\beta([a,b'])) + Vol(\diamondsuit(a,g',\tilde{g}',\tilde{a}))$ where p is the intersection point of oa and be, q the intersection point of oa and $\tilde{c}'\bar{\tilde{c}}'$. The volume of the rectangle $\diamondsuit(a,g',\tilde{g}',\tilde{a})$ is exactly $l_e(m+m_e)$.

For small angles we compute the volume as follows: $Vol(SV_\alpha([o,q])) - Vol(SV_\alpha([o,b']) +$
$Vol(SV_\beta([a,b'])) + Vol(\diamondsuit g\tilde{g}'\tilde{a}'a) - (Vol(\Delta epo) - Vol(SV_\gamma([o,e])))$, where q is the intersection of the line oa and circle segment $c'f'$, p is the intersection of the lines de and oa, and γ is the angle between the lines oa and oe.

case 3: Apart from the volume $Vol(SV_\alpha([o,q]))$ - $Vol(SV_\alpha([o,a]))$, where q is the intesection point of oa and $\tilde{c}'\bar{\tilde{c}}'$, we have to add $Vol(SV_\beta([a,b']))$ and $Vol(\diamondsuit(\tilde{c}',\tilde{b}',c',b'))$.

case 4: Apart from the volume $Vol(SV_\alpha([o,\hat{s}]))$ - $Vol(SV_\alpha([o,a]))$, where $\hat{s}$ is the intesection point of oa and $\tilde{c}'\tilde{b}'$, we have to add $Vol(SV_\beta([a,b']))$ and $Vol(\diamondsuit(\tilde{c}',\tilde{b}',c',b'))$.

In cases 3 and 4, we still have to compute the area of a parallelogram. All three of them have a basis of length l_e. The computation of the height can be done in the same way for both of them, as we will describe now. The length of one of the edges of the parallelogram clearly is l_e. The other edge has length $2(m+m_e)\sin(\frac{\beta}{2})$, since the $\Delta a, b', c'$ is an isosceles triangle. The sharp angle ϵ of the parallelogram is $\zeta - (\frac{\pi-\beta}{2})$ and thus the area of the parallelogram is equal to $2l_e(m+m_e)\sin(\frac{\beta}{2})\sin\epsilon$.

7.4.6 Assuming Some Thickness for the Links: Line Extended to Rectangle.

In the previous we modelled the links by straight lines. In this section we model the links by rectangles. For the different shapes no major changes have to be made. The volume computation gets slightly more involved.

Extreme Points for Enclosure.

Let us introduce some new notations for the vertices of the rectangle. We thicken the line segment modelling a link to a rectangle. Each end of the line segment will now become the midpoint of the shorter edge of the rectangle. We will denote the vertices as depictured in Figure 7.9. The angle between ab'_1 and ab'_2 will be denoted by $\hat{\beta}$.

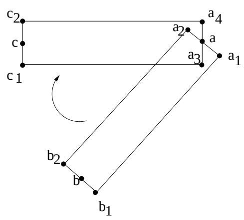

Figure 7.9: Notation.

The four cases we had before give rise to the following extensions considering a thickening of the line segment. Note however that in this situation Case 3 ends when the angle between oa and ab_2', i.e. angle $\xi + \hat{\beta}$, is larger than π.

case 1: We now take for the extreme points the points b_1, e_1, d_1, $\tilde{d_1}$, $\tilde{d}$, $\tilde{d_4}$, p_1, $\tilde{a_4}$, $\tilde{c_2}$, c_2, p_2, where p_1 and p_2 are the intersection points of the tangents to the circle segments between the two points $\tilde{d_4}$ and $\tilde{a_4}$, respectively c_2 and b_1.

case 2: We take for extreme points b_1, p, e_1, d_1, u, v, $\tilde{d_4}$, $\tilde{f_2}$, p_1, $\tilde{c_2}$, p_2, $\tilde{g}$, g, p_3 if their polygon is convex, where p is the intersection point of oa with circle segment between b_1 and e_1. In the non-convex case and for small angles for α (for which oa does not intersect the circle segment between b_1 and e_1), we take for extreme points b_1, e_1, d_1, u ,v, $\tilde{d_4}$, $\tilde{f_2}$, p_1, $\tilde{c_2}$,p_2, $\tilde{g}$, g, p_3. The points p_1, p_2, and p_3 are the intersection points of the tangents to the circle segments between the two points $\tilde{f_2}$ and $\tilde{c_2}$, $\tilde{c_2}$ and $\tilde{g}$, respectively g and b_1. Depending on the angle for β, u is $\tilde{d_1}$ and v is the intersection point of lines $\tilde{d_1}\tilde{d}$ and $d_4\tilde{d_4}$ for small β, or u is the intersection point of lines $d_1\tilde{d_1}$ and dd_4 and v is d_4. Here a large value for β means that $\beta > \pi - 2\zeta$, and small value for β means that $\beta < \pi - 2\zeta$. (When equality holds, d_1, $\tilde{d_1}$, d_4, and $\tilde{d_4}$ are collinear, in which case we can just forget about u and v.) Indeed, if and only if $\beta + 2\zeta > \pi$, the distance of d_1 to line od is smaller than the distance of d_4 to line od, and thus the line $d_1\tilde{d_1}$ will intersect dd_4.

case 3: Here we take $a_1, d_1, d, d_4, \tilde{d_4}, \tilde{f_2}, p_1, \tilde{c_2}, p_2, \tilde{b_1}, b_1$ where p_1 and p_2 are the intersection points of the tangents to the circle segments between the two points $\tilde{f_2}$ and $\tilde{c_2}$, respectively $\tilde{c_2}$ and $\tilde{b_1}$.

case 4: And in this case we take $a_4, a, a_1, b_1, \tilde{b_1}, p_1, \tilde{s}, p_2, \tilde{t}, p_3, \tilde{f_2}, f_2, d_4$, where p_1,p_2,and p_3 are the intersection points of the tangents to the circle segments between the two points $\tilde{b_1}$ and $\tilde{s}$, $\tilde{s}$ and $\tilde{t}$, and $\tilde{t}$ and $\tilde{f_2}$ respectively at these points. The points s_1 and s_2 are the vertices of the rectangle on the short edge of which s is the midpoint (i.e. after some rotation, point b was moved to s). Similarly points t_1 and t_2 correspond to t.

Volume Computation.

We can compute the volume swept by the rectangle in a similar way as in the case of a line. The main observation is that we now will consider the volumes swept by line segment ab_2, instead of by the line segment ab. On the appropriate places we then only have to add the area of the rectangle itself to the volumes computed for the line segment.

7.4.7 A 3D Case: Rectangle Extended to Cylinder.

We now extend the model of the links to cylinders. To the rectangles in the previous section we now add a height.

Extreme Points for Enclosure.

We will use the algorithm for enclosure of polygons by superellipsoids described in [Sti92a]. For this algorithm we need a set of extreme points of a polygon.

We consider a Z-axis perpendicular to the plane in which the rectangle lies. We keep to all extreme points we already used in the previous section for the swept volume of the rectangle (we consider them now as points in 3-dimensional space with zero Z-coordinate). In each case we now add some points with non-zero Z-coordinate. These new points lie above and below (considered in the Z-direction) some points in the plane. For each such a point u in the plane we construct two new points u_+ and u_- with the same X- and Y-coordinates as u and with Z-coordinate h and $-h$ respectively, where h is the radius of the cylinder. The points we take for u are given in the following for each of the four cases.

case 1: $\tilde{a}, \tilde{d}, c, \tilde{c}, b, e, d$.

case 2: $b, e, d, \tilde{d}, \tilde{f}, \tilde{c}, g, \tilde{g}$.

case 3: $a, d, \tilde{d}, \tilde{f}, \tilde{c}, \tilde{b}, b$.

case 4: $a, d, f, \tilde{f}, \tilde{b}, b, \tilde{t}, \tilde{s}$.

Volume Computation.

We now have to compare the volume of the swept volume of a link with the volume of the enclosing superellipsoid. The volume of a superellipsoid can be approximated as described in [Zar85].

For the volume of the swept volume of the link modelled by a cylinder, we proceed as follows. Observe that the volume of the swept rectangle times the height of the cylinder is equal to the volume of a swept rectangular solid. This volume minus the difference in volume of the rectangular solid and the cylinder is the volume wanted. The volume of a rectangular solid and of a cylinder is trivial to compute, the volume of the swept rectangle in two dimensional space is done above.

First and Second Link. We can compute the volume of the swept volume by adding $Vol(SV_{\alpha+\hat{\alpha}}([a, b_2']))$, $Vol(\Delta a b_2' a_2)$, and $Vol(\Delta a b_1' a_2)$. Here $\hat{\alpha}$ stands for the angle between ab_1' and ab_2'. The areas of the two triangles are the same and equal to $\frac{1}{2}h(l + l_e)\cos(\frac{\hat{\alpha}}{2})$.

Third and Fourth Link.

case 1: We add to the volume of the swept volume of line segment ab_2', which can be computed as in the case of a robot arm modelled by line segments, the following areas: $Vol(\Diamond a_1 b_1 b_2 a_2)$, $Vol(\Diamond \tilde{d}\tilde{d}_1 d_1 d)$, and the differences $Vol(SV_\alpha(oq)) - Vol(SV_\alpha(o\tilde{a}))$ and $Vol(SV_\alpha(op_1)) - Vol(SV_\alpha(op_2))$. Here q is the intersection point of line oa and circle segment $\tilde{a}_4\tilde{d}_4$, $p_1 = b_1 e_1 \cap o\tilde{a}$, and $p_2 = b_2 e_2 \cap o\tilde{a}$. Notice that the distance from o to p_1 is the same as the distance from o to b_1, and that the distance from o to p_2 is the same as the distance from o to c_1, which simplifies the computations.

In case angle α is small, q, p_1 or p_2 may not be in the swept volume. However, the computations can be done in the same way here, since we do not have to use these points in the computations themselves.

case 2: Here the volume can be computed by adding to the swept volume of line segment ab_2' the volumes: $Vol(\Diamond a_1 b_1 b_2 a_2)$, $Vol(\Diamond uv)$, and $(Vol(SV_\alpha([o, p_2]) - Vol(SV_\alpha([o, p_1]))))$. Here $p_1 = oa \cap b_1 e_1$, and $p_2 = oa \cap b_2 e_2$. With $Vol(\Diamond uv)$ we mean a parallelogram with u and v as some of the vertices. There are two cases. If $u = \tilde{d}_1$ and $v = \tilde{d}_1\tilde{d} \cap d_4\tilde{d}_4$, then we have to compute the area of the parallelogram $d_1 uv\tilde{v}$. This area is equal to $(l + l_e)y_1$, where $y_1 = h(\sin(\frac{\pi}{2} - \zeta) - \sin(\xi + \beta - \frac{\pi}{2}))$. In case $u = d_1\tilde{d}_1 \cap dd_4$ and $v = d_4$, we have to compute the area of parallelogram $v\tilde{d}_4 u\tilde{u}$ which is equal to $(l + l_e)y_2$, where $y_2 = h(\sin(\xi + \beta - \frac{\pi}{2}) - \sin(\frac{\pi}{2} - \zeta))(= -y_1)$. We illustrate the computation of y_2 in Figure 7.10.

case 3: We add to the swept volume of line segment ab_1 the following areas: $Vol(\Diamond a_1 b_1 b_2 a_2)$, and the differences $Vol(SV_\alpha(oa)) - Vol(SV_\alpha(oa_1))$ and $Vol(SV_\alpha(o\tilde{c}_2)) - Vol(SV_\alpha(o\tilde{c}_1))$.

case 4: We add to the volume swept by line segment ab_2 the following areas: $Vol(SV_{\beta+\hat{\beta}}([a, b_1]) - Vol(\Delta a b_1 b_2)$, $Vol(\Diamond b_1 b_2 \tilde{b}_2 \tilde{b}_1)$, $Vol(\Delta a_1 a b_1)$, $Vol(\Delta f_2 d_4 d)$, and the volume swept by the line segments $a_1 a$ and aa_4 when swept over α.

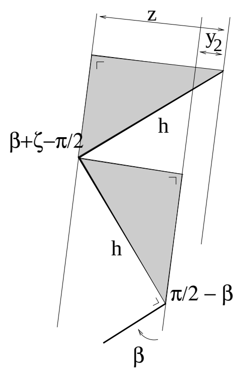

Figure 7.10: Computing y_2.

For this last area there are two cases. If circle segment a_1d_1 intersects aa_4, then we compute the intersection point (we denote it by r), and then add volumes $Vol(SV_\alpha(oa)) - Vol(SV_\alpha(oa_4))$, and $Vol(\Delta a_1ar) - (Vol(SV_\gamma(oa)) - Vol(\Delta a_1or))$, where γ is the angle between oa and or. In the other case, d_1d intersects circle segment a_4d_4 and we can proceed in a similar way.

7.4.8 Counting It Out.

In this section we will compute and compare the volumes for some of the cases we have discussed. For this we have to consider a coordinate frame. In the two dimensional case we will take the first rotational joint (point o) as the origin of the frame, (the symmetry axis of) the first link oa as Y-axis and the X-axis perpendicular to it. In the following we will denote the coordinates for point p by (px, py).

2 DOF's, case 1.

Straightforward computations result in the following formulas for the coordinates of the extreme points.

p	px	py
a	0	l
b	$-m\sin\zeta$	$l - m\cos\zeta$
c	$-m\sin(\zeta+\beta)$	$l - m\cos(\zeta+\beta)$
d	$l\sin\alpha$	$l\cos\alpha$
e	$-l\sin\alpha - m\sin(\zeta-\alpha)$	$l\cos\alpha - m\cos(\zeta-\alpha)$
q	$\frac{m+\frac{\cos(\zeta+\beta)(\sin\zeta-\sin(\zeta+\beta))}{sin\beta}}{\sin(\zeta+\beta)}$	$l + \frac{m(\sin\zeta-\sin(\zeta+\beta))}{\sin\beta}$
r	$\frac{l}{\tan\alpha}(\sin\alpha+\cos\alpha-1)$	l

The centroid M of these 7 points can now be computed: x_M is the sum of the x-coordinates of the extreme points divided by 7, y_M is the sum of the y-coordinates of the extreme points divided by 7.

The main axis of the superellipse, through M and parallel to ad, is given by the equation:

$$(\cos\alpha - 1)x - \sin\alpha y + y_M\sin\alpha - (\cos\alpha - 1)x_M = 0.$$

This allows us to compute the distance of a point to the main axis. When we do so for the extreme points, B will be defined as the largest distance among these increased by the tolerance ϵ.

For A, we proceed in the same way as for B but this time we compute the distance to the other axis of the superellipse, namely:

$$\sin\alpha x + (\cos\alpha - 1)y - x_M\sin\alpha + (\cos\alpha - 1)y_M = 0.$$

2 DOF's, case 2.

In this case we can have two different sets of extreme points. The coordinates of all these points can be computed in a straightforward way. The table below summarizes the results.

p	px	py
b	$-m\sin\zeta$	$l - m\cos\zeta$
c	$-m\sin(\zeta+\beta)$	$l - m\cos(\zeta+\beta)$
d	$l\sin\alpha$	$l\cos\alpha$
e	$-l\sin\alpha - m\sin(\zeta-\alpha)$	$l\cos\alpha - m\cos(\zeta-\alpha)$
f	$-l\sin\alpha - m\sin(\xi-\alpha)$	$l\cos\alpha - m\cos(\xi-\alpha)$
p	0	$\sqrt{l^2+m^2-2lm\cos\zeta}$
s	$\frac{(l^2+m^2-2lm\cos(\zeta+\beta))(yc-yf)}{xf.yc-xc.yf}$	$\frac{(l^2+m^2-2lm\cos(\zeta+\beta))(xf-xc)}{xf.yc-xc.yf}$
t	$xa+\frac{m^2(yc-yb)}{(yc-ya)(xb-xa)-(yb-ya)(xc-xa)}$	$ya+\frac{m^2(xb-xc)}{(yc-ya)(xb-xa)-(yb-ya)(xc-xa)}$

The coordinates of point p are obtained by considering the triangle aob, and those of points s and t by intersecting the appropriate tangent lines to a sphere. For point s we also considered the triangle aoc. For the centroid of these points and for the first axis (and thus for the determination of A and B) we proceed as in case 1.

2 DOF's, case 3.

We have for the coordinates of the extreme points in this case the following table.

p	px	py
a	0	l
b	$-m\sin\zeta$	$l - m\cos\zeta$
c	$-m\sin(\zeta+\beta)$	$l - m\cos(\zeta+\beta)$
d	$l\sin\alpha$	$l\cos\alpha$
f	$-l\sin\alpha - m\sin(\xi-\alpha)$	$l\cos\alpha - m\cos(\xi-\alpha)$
p	$xa+\frac{m^2(yc-yb)}{(xb-xa)(yc-ya)-(xc-xa)(yb-ya)}$	$ya+\frac{m^2(xb-xc)}{(xb-xa)(yc-ya)-(xc-xa)(yb-ya)}$
q	$\frac{(l^2+m^2-2lm\cos(\zeta+\beta))(yc-yf)}{xf.yc-xc.yf}$	$\frac{(l^2+m^2-2lm\cos(\zeta+\beta))(xf-xc)}{xf.yc-xc.yf}$

For the centroid of these points and for the first axis (and thus for the determination of A and B) we proceed as in case 1.

2 DOF's, case 4.

Finally in the fourth case we have the following coordinates for the extreme points.

p	px	py
a	0	l
b	$-m\sin\zeta$	$l - m\cos\zeta$
d	$l\sin\alpha$	$l\cos\alpha$
f	$-l\sin\alpha - m\sin(\xi-\alpha)$	$l\cos\alpha - m\cos(\xi-\alpha)$
s	0	$l+m$
t	$-(l+m)\sin\alpha$	$(l+m)\cos\alpha$

For the centroid of these points and for the first axis (and thus for the determination of A and B) we proceed as in case 1.

The Exponent.

Finally we come to the determination of the exponent. We can use the approach in [Sti92a] where the exponent is adapted until all points are inside the superellipse. However, we can compute for all extreme points a safe approximation of the exponent for which that particular point is inside a superellipse with given A and B and then choose the biggest one of those computed exponents (so that all points are inside). For this we have to solve the equation of the superellipse to exponent n. We can approximate the solution by computing the Taylor series. Note that without loss of generality we can assume that the extreme points all lie in the first quadrant (if for point p this is not the case, we take the point $(|px|, |py|)$ instead). Also without loss of generality we can assume that A and B are 1, by adapting the coordinates of the extreme points. We let Computer Algebra Systems compute the Taylor series for the equation $x^m + y^m - 1$ around point $(\frac{1}{2}, \frac{1}{2})$ so that we have an expression for m in function of x and y. This value then can be used as an initial guess for the exponent, and can be adapted as in [Sti92a].

If we want to be sure that the resulting exponent is a safe one, we also have to look at the (general) error estimate for the Taylor expansion. Also for checking its convergence we need the error estimate. Currently we did not succeed in computing the error term for the equation we considered with any of the Computer Algebra Systems with which we tried. For this we have to consider the result of the Taylor series as an initial guess rather than a safe approximation for the exponent.

Chapter 8

Operations on Constructed C-space and U.

The operations discussed in this chapter appear whenever some change in the environment of the robot is observed. We start with a given model of the C-space, in the form of a GET-Rep and a potential field U on the non-uniform grid. Overall we can say that there are two kinds of updates for U. The first one involves only local changes to the grid and to U. The second one involves the impact of the change on long distance from the place where the event took place. When we only apply the first one, we still get a potential field without local minima and with the certainty that a path will be found if there exists one. However, we loose the nice property that the path will be a pay-off between safe and short path. The second kind of update of U will repair this property by recomputing over a long distance the values of the potential function. We will call the 'wavefront' computation for the first kind of update the *local wave* and for the second kind of update the *update wave*. We now come to the four operations.

8.1 Delete an Obstacle.

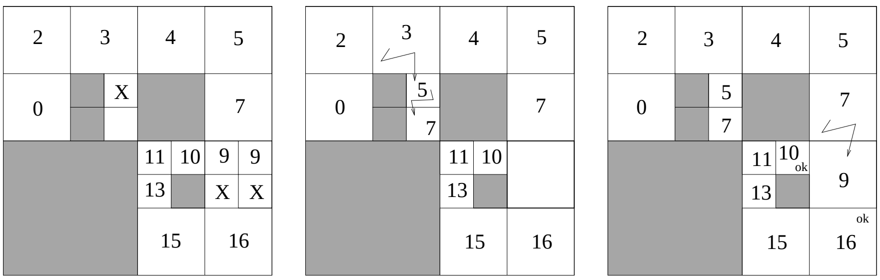

Figure 8.1: Local wave.

When an obstacle has to be deleted, some black cells turn into white ones. These new white cells form a new white area. Sometimes deleting an obstacle also implies that 2^k cells can be merged to one larger cell, and this can happen subsequently on several levels. Locally the potential field can be 'repaired', in the sense that we can assign a U-value to the new white cells just by looking at its direct neighbours. This is just a local wave.

The local wave works in the following way. Find among all neighbours of the area of new white cells, the one with the lowest value and start up a wave in the new white cells from there. The values assigned to the new white cells have to lie between the lowest and the highest U-value among the neighbouring cells of the new white area. The increment has to be small enough in order to make sure that the U-values for the new white cells do not go over the higher bound. This increment can be determined by considering the amount of new white cells and their size. If the whole new white area corresponds to exactly one new cell (after merging together several smaller cells), the U-value from the local wave will simply be the minimum value of all values on the merged smaller sized cells.

To perform the update wave, we start from the cell(s) in the new white area with a neighbour with lowest U-value of all neighbours of the new white region and proceed in the usual way. The update wave does not update those cells (outside the new white area) that have already a U-value equal or smaller than the value the wave would assign to them. The wave does not go on from such cells.

It is irrelevant of how many connected components the new white area consists since the update wave disperses first from all cells with the current U-value and only after that step from the cells with lowest U-value larger than the current one. In Figures 8.1 and 8.2 we show on an example such local and update waves. In Figure 8.1 the first adaptation of the potential field is done for three cells that came to be free (the ones with 'X' in it). First four smaller cells are merged together, then the new white cells get a suitable U-value. In Figure 8.2 the potential field is restored completely as if the cells that were freed had never been blocked.

8.2 Insert an Obstacle.

When a new obstacle has been observed, its insertion into C-space will result in new black cells. These black cells might replace white cells completely or partially. In the latter case, the white cell has to be split up and part of the subcells will become black.

In some cases the new black cells will block a passage in C-space, in other cases no such dramatic change will occur. These two possibilities have to be treated separately. When a passage is blocked, we cannot expect that a local wave is able to adapt U, since U has to be changed drastically. This can only be done by an update wave. When no passage is blocked, a very easy and very local change will be sufficient for fixing up the potential field in order to find 'a' path.

8.2.1 Fixing: No Blocked Passage.

Deciding on whether or not we are in this case can be done simply by looking at the neighbours of the new black area. If they all can inherit their U-value from one of their neighbouring cells that were uneffected by the insertion of the obstacle, then we are in this case. The white cells resulting from a subdivision of a white cell that turned black partially, are considered in this test as if they had the U-value of the large cell from which they originate. This search is the first step of the treatment for the local wave anyway, as described in the next section. Intuitively, in this case there is still a passage to all the neighbours of the new black region.

The only cells to worry about are the previously white cells that are subdivided and partially turned black. We treat each of these cells as if they were the only cell that was subdivided by the insertion of the obstacle. We look for the largest and the smallest U-value of all previously white (and partially or completely white still) neighbouring cells of the previously white cell under treatment. All subcells of the previously white cell under treatment will get a U-value between these largest and smallest U-values. A

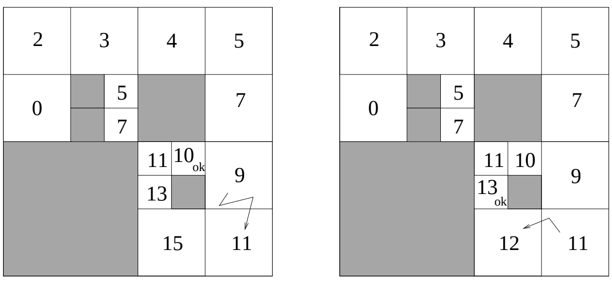

Figure 8.2: Update wave.

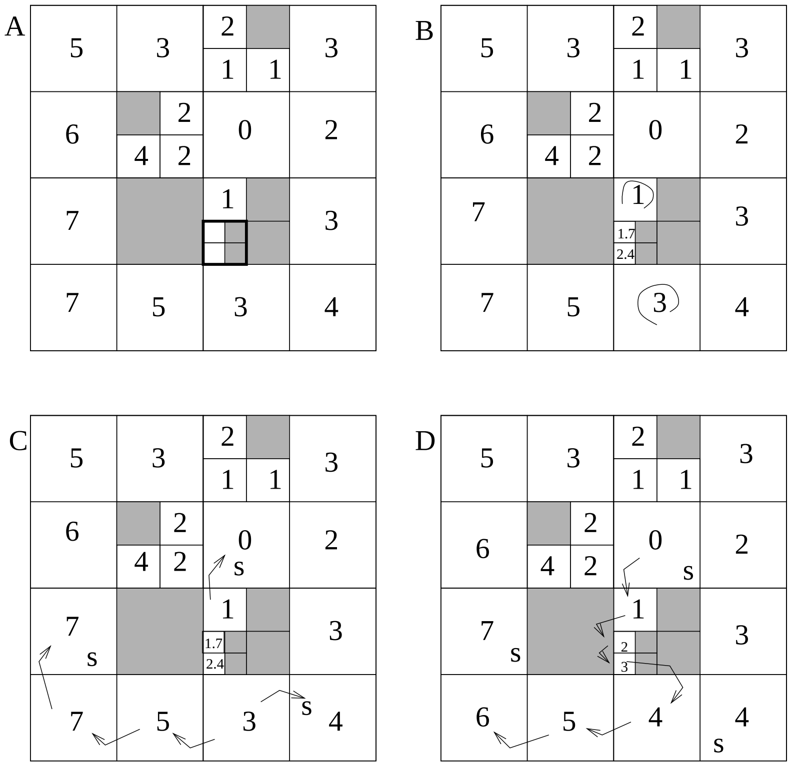

Figure 8.3: A: new black cells, B: fixing, C: wave in opposite direction, D: update wave from sources (s)

wave is started up from the cell(s) where this smallest U-value was (were) found. Only the new white subcells get a U-value. The increment used in this local wave depends on the size of the subcells (how high the new subtree in the corresponding 2^k-tree is), and the strategy used for the construction of the original potential field U. This can be computed with the information available. This is sketched in Figure 8.3, A and B.

8.2.2 Update Wave.

Some neighbours of the new black area inherited their U-value from one of the white cells that turned (partially) black and cannot inherit the same value from any of their other neighbours. Some new cells appeared and have to get an appropriate U-value. These cells have to be treated with the update wave we describe in this section.

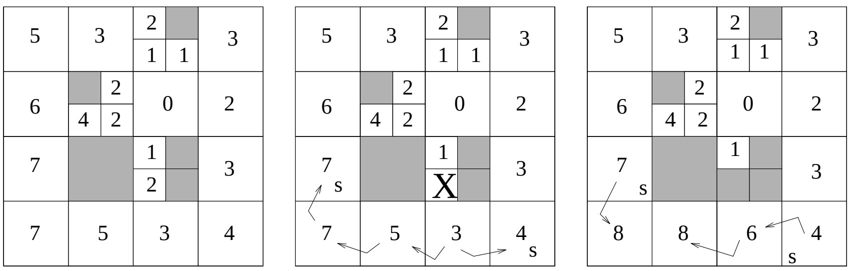

Figure 8.4: Passage Blocked: Wave in Opposite Direction, Update Wave. (s: candidate sources)

First we search for all the neighbours of the new black area that have to be treated with the update wave. Then we start looking for cells (potential sources for the update wave) from where these cells can inherit a new value. Sometimes such a potential source does not exist. In that case we have an area in C-space that cannot be reached from the initial position of the robot (we have another connected component, created by the inserted obstacle). In order to decide whether such a source cell exists and, if there exists one, to find sufficiently many of them, we start up a *wave in the opposite direction* (away from the new black area). We start from the neighbouring cells of the new black area with lowest U-values. We call this process also a 'wave' because we have to consider all the neighbours of these cells first before we move to the cells with larger U-values neighbouring to the new black area or previously treated cells. We only treat cells that cannot get their U-value from any other cell than from one of those treated before or from a white cell now (partially) turned into a black one. For this reason, the wave has to stop, either at white cells which are potential sources or at black cells. When it stopped, we may have found potential sources for the update wave in those cells where the 'wave in the opposite direction' died out. We now let the update wave start at the potential sources, adapting the U-values. A potential source is activated only when the inheritance of the U-values reach the value of U at that potential source.

Figure 8.4 illustrates this for the case in which a passage is blocked. An example in case no passage is blocked can be found in Figure 8.3.

8.3 Model More Accurately.

Starting from an obstacle and a construction algorithm for C-space, we get a more accurate subdivision of some black cells. That is, some black cells are subdivided into smaller cells and some of those smaller cells become white. We can assume that no complete black cell turns into a white cell, since we assume that the information we had in the previous grid was correct (i.e. in the black cells some configurations are forbidden, but maybe not all of them). The adjustment of U can now be done exactly like in the approach for deleting an obstacle, described above.

8.4 Move an Obstacle Slightly.

Assume that a non-uniform grid on the C-space of a robot arm in an environment with several obstacles is given and, that it is observed that an obstacle is not exactly on the position for which the grid was computed. This slight move of this obstacle results in minor changes in C-space: some larger white cells might be subdivided and turn partially black, some black cells might turn partially white and sometimes get merged with some neighbouring white cells to result in a larger white cell. The adjustment of U can be done in a similar way as for the previously discussed operations, by consecutively adding and deleting black cells.

Chapter 9

Further Research Challenges.

Although this thesis stands on its own and is selfcontained as it is, solving a particular problem with a particular approach, further considerations upon the topic can be taken into vision. Here we list several open problems that were encountered during the work that finally lead to this thesis as well as indications toward methods that might solve them. This chapter therefore lists problems which can be taken as future topics for research.

9.1 Dynamics

In the previous we have stated that the information from our kinematic path planner can be used as input for a dynamics planner. The information indeed contains not only a chain of cells as a description of a path, also the size of the cells is known (height in the 2^k-tree) and thus at least some minimal bound on the distance to the nearest obstacle. Building a new dynamics planner or adapting an existing one to the information coming out of our kinematic path planner can be done.

9.2 Shapes of Grid

The representation of the non-uniform grid with a 2^k-tree does not depend on the shape of the cells. In fact the cells can very well be rectangular rather than square in our approach already, but the representation only demands a constant subdivision of big blocks into smaller blocks. One could look into different shapes that allow more accurate modelling of the C-obstacles or that already contain some preference for a particular direction for a move merely by its geometry.

9.3 Transformations of Grid

Other transformations than the ones we have looked at can be thought of. In [Gar83] some transformations of quadtrees are studied. However it is known in Geometric Modelling and Vision that quad- and octrees are computationally not the best representations for these kind of transformations.

9.4 The Taylor Series Problem

In the previous we wanted to determine the exponent of the enclosing superellips or superellipsoid for a swept volume of a link of a robot arm with the use of a Taylor series. However, in order to be sure that the value we get from the computation of the Taylor series is a good one, we have to know that the series is convergent and that the value is a safe approximation of the optimal exponent. This theoretical problem is open.

9.5 Longest path in a Cube

An interesting and not straightforward combinatorial problem pops up when we think of inheritance by a wavefront approach of a value in the non-uniform grid.

When we consider a k-dimensional cube corresponding to a parent node of leafs in the corresponding 2^k-tree, and we assume that at least one of those leafs is black, we can ask for the longest "path" through the white cells in this cube. A path of length l consists of a list of cells $c_1,...,c_l$ where c_i is a neighbouring cell of c_{i-1} for $i = 2, ..., l$. We define *a path through a cube* starting in cell c_1 by the following properties: we start in the white cell c_1 in the cube; all neighbours of c_i other than c_{i+1} will not occur in the rest of the path (that is, in $c_{i+2}, ..., c_l$). We now look for the largest l for which such a path through a cube exists. For $k = 2$ the largest l is 3. It is not difficult to find the maximal length for other values of k too, but it is not straightforward to find a general answer to this combinatorial question.

The answer to this question however is related to bounds for the length of a path in a non-uniform grid. Indeed, the maximal length of a path in a cube is an upper bound for the increment induced by refining a cell (turning it into a cube with cells of smaller size corresponding to 1 level in the 2^k-tree).

9.6 Enclosing Swept Volumes

In [BL92] swept volumes are analyzed with mathematical tools. We believe that this information can be used to find a better way for enclosing the swept volumes of manipulator links as we encountered the problem in the previous.

9.7 Sensor Information

In [LC94] a grid is placed over workspace and when a sensor attached to the robot encounters an obstacle it assigns values of probability to the cells corresponding to the place where the sensor observes the existence of the obstacle. We would like to look for a similar but more elaborate approach for the construction of a potential on C-space with sensor information in Workspace.

9.8 Distance Measure in C-space, Computation in Workspace

Along the same lines of thoughts we would like to look for a distance measure in C-space that can be computed in Workspace. A paper that describes some attempt in that direction is [Fer94].

The article [NNA89] suggests that maybe we can compute some distance via levels in the 2^k-tree.

Curriculum Vitae.

Carl Van Geem

- Institutional address: Research Institute for Symbolic Computation, Johannes Kepler University Linz, A-4040 Linz, Austria,
  tel.: +43 / 7236 / 32 31 26, fax: +43 / 7236 / 32 31 30,
  E-mail: Carl.Van.Geem@risc.uni-linz.ac.at

- Nationality: Belgian
- Gender: Male
- Date and place of birth: July 13, 1968 in Schoten, Belgium

Education:

- September 1974 - June 1980, Primary School, Stedelijke Instelling Lager Onderwijs 4, Antwerpen
- September 1980 - June 1986, Secundary School, Koninklijk Atheneum II Antwerpen
- October 1986 - July 1990, Licentiaat Wiskunde (Mathematics Studies), University of Antwerpen (RUCA, UIA)
- 1 April 1990 - 1 July 1990, Erasmus student at the department of Mathematics, Rijksuniversiteit Utrecht (RUU), The Netherlands (attended a course of Prof. W. Van der Kallen, met Prof. T. A. Springer and studied his book "Invariant Theory", [Spr77])
- October 1990 - January 1993, Bijzondere Licentie Informatica en Bedrijfswiskunde richting Informatica (Special Studies programme in Computer Science), University of Antwerpen (UIA)

Fun:

- 1986: Finalist of the first Vlaamse Wiskunde Olympiade (Flemish Mathematics Olympiad)
- July 1986, Free University Brussel (VUB), entrance examination for studies Civil Engineer

Teaching Experience:

- December 1990 - September 1993, Part-time Teaching Assistant at the Department of Mathematics, Free University Brussel (VUB). Assisted in teaching the exercise sessions for courses:
  - Analytische meetkunde en lineaire algebra (Analytic Geometry and Linear Algebra), Prof. Dr. Valette,
  - Beginselen van de analytische meetkunde en van de infinitesimaalanalyse (Rudiments of Analytic Geometry and Analysis), Prof. Dr. De Grande - De Kimpe,
  - Hogere Algebra (Higher Algebra), Prof. Dr. Bingen.

Research Motivation:

- The application of mathematics in computer science, in particular the use of computational geometry in the field of robotics.

Research Interests:

- Computational Geometry, Parallel Computational Geometry,
  CAD/CAM, Robot Motion Planning.

Licentiaats Theses ("Master's Theses"):

- Mathematics: Invariant Theory, main reference: [Spr77], supervisor: Prof. F. Van Oystaeyen.
- Computer Science: An Implementation of An Algorithm for a Simplified Translational Motion Planning Problem in 2-Dimensional Space, main reference: [KS85], [KS86], [NP92], [OWW82], supervisor: Dr. R. Penne.

Technical Reports and Publications:

- Towards a Fast Solution Method for the General Robot Motion Planning Problem using a Manhattan-like Distance Function on a Non-uniform Grid in Configuration Space. (Proposal for Ph.D. Thesis), RISC-Report 93-66, December 20, 1993.
- On Using a Manhattan Distance-Like Function for Robot Motion Planning on a Non-Uniform Grid in Configuration Space, to appear in the International Journal of Flexible Automation and Integrated Manufacturing (FAIM); also RISC-Report 94-74, October 1994.
- Preparations with Workspace Information for Meaningful Robot Motion Planning in Configuration Space, RISC-Report 95-14, March 16, 1995.
- Counting Cells in a Grid, RISC-Report 95-29, June 29, 1995. Path Planning for Manipulators with Many Degrees of Freedom Using a Discrete Representation of Configuration Space Extensively, submitted for publication and RISC-Report 95-45, October 27, 1995.
- Enclosing Robot Links by Superellipses and Superellipsoids, RISC-Report 96-01, January 1996.

Presentations on Workshops and Conferences:

- Towards a Fast Solution Method for the General Robot Motion Planning Problem using a Manhattan-like Distance Function on a Non-Uniform Grid in Configuration Space, 10th European Workshop on Computational Geometry (CG'94), Universidad de Cantabria, March 17-18, 1994.
- Computation of a Non-Uniform Grid on the Configuration Space of a Robot Arm Amidst Obstacles: Intersection Problem, 11th European Workshop on Computational Geometry (CG'95), RISC-Linz, March 23-24, 1995.
- Hierarchical Construction of a Potential Field on a Non-Uniform Grid on the Configuration Space of a Robot Arm, 4th International Workshop on Robotics in Alpe-Adria Region (RAA95), Poerschach, July 6-8, 1995.

Workshops, Conferences, Schools Attended:

- ACPC - Conference of the Austrian Center for Parallel Computation, 3-6 October 1993,Gmunden, Austria.
- Winter School on Real Geometry, 3-7 January 1994, Madrid, Spain.
- Paragraph'94, Workshop on Parallel Graphics, 21 March 1994, Hagenberg, Austria.
- Congress of the Belgian Mathematical Society-1994: Mathematical Methods of Technology, 4-6 May 1994, Antwerp, Belgium.
- (Meeting on) Geometry and Robotics, 13-18 June 1994, Pisa, Italy.
- Will attend: First joint meeting of the AMS and the mathematical societies of the BeNeLux countries (Belgium, the Netherlands and Luxemburg), in Antwerp (Belgium), May 22-25 1996.
- Will attend: 2nd International Workshop on Algorithmic Foundation of Robotics (WAFR96), Toulouse, 3-5 July 1996.

Membership:

- Member of the Belgian Mathematical Society.

Projects and Fundings:

- 1990: Erasmus Grant (European Exchange Program).
- at RISC:
  - PARAGRAPH (Parallel Computer Graphics, supported by the Austrian ministry for science and research),
  - Roider Method (supported by Austrian National Science Foundation),
  - CEI/geometric modeling of technical systems (CEI = Central European Initiative, supported by the Austrian ministry for science and research),
  - Stereovideometry and spacial object recognition (within Austrian research on robot vision, supported by Austrian National Science Foundation).

Knowledge of Languages

- Mother tongue: Dutch,
- Very good knowledge of English and French (in reading, speaking and writing),
- Fairly good notion of: German (very good in reading and understanding, good in speaking).

List of Figures

Bibliography

[ABF88] Francis Avnaim, Jean Daniel Boissonnat, and Bernard Faverjon. A practical exact motion planning algorithm for polygonal objects amidst polygonal obstacles (extended abstract). In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1656–1661, 1988.

[AFB91] R. S. Aylett, A. N. Fish, and S. R. Bartrum. Task planning in an uncertain world. In *IEE International Conference on Control, Control-91*, Edinburgh, March 1991.

[AM91] Ruth Aylett and Chris Milton. Achievable autonomy in next-generation robots. In ARRL, editor, *ESA/ESTEC AI and KBS in Space Workshop*. Noordwijk, May 1991.

[Bar81] A. H. Barr. Superquadrics and angle-preserving transformations. *IEEE Computer Graphics and Applications*, 3:11–23, January 1981.

[BHJ+82] Michael Brady, John M. Hollerbach, Timothy L. Johnson, Tomás Lozano-Pérez, and Matthew T. Mason, editors. *Robot Motion: Planning and Control*. MIT Press, Cambridge, Massachussetts and London, England, 1982.

[BK90] Chanderjit Bajaj and Myung-Soo Kim. Generation of configuration space obstacles: Moving algebraic surfaces. *The International Journal of Robotics Research*, 9(1):92–112, February 1990.

[BL91] J. Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, December 1991.

[BL92] Denis Blackmore and M. C. Leu. Analysis of swept volume via Lie groups and differential equations. *The International Journal of Robotics Research*, 11(6):516–537, December 1992.

[Boi88] Geometry and robotics, proceedings of workshop in toulouse. In Jean-Daniel Boissonnat and Jean-Paul Laumond, editors, *Lecture Notes in Computer Science*. Springer-Verlag, May 1988.

[Bor86] Gunilla Borgefors. Distance transformations in digital images. *Computer Vision, Graphics, and Image Processing*, 34:334–371, 1986.

[Bro89] Randy C. Brost. Computing metric and topological properties of configuration-space obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 170–176, 1989.

[Bur92] K. C. Burger. A kernel-based construction system (KBC). Technical Report diploma thesis, RISC-Linz, Joh. Kepler University Linz, Austria, Juli 1992.

[Can88] John F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, 1988. ACM Doctoral Dissertation Award 1987.

[CCV85] Ingrid Carlbom, Indranil Chakravarty, and David Vanderschel. A hierarchical data structure for representing the spacial decomposition of 3-D objects. *IEEE CG & A*, pages 24–31, April 1985.

[CLLZ93] G. Conte, T. Leo, S. Longhi, and R. Zulli. Emulation of articulated robots. In *Preprints of Second International Workshop on Robotics in Alpe-Adria Region (RAA'93)*, Krems, Austria, June 13-15 1993.

[CLZ93] G. Conte, S. Longhi, and R. Zulli. Cad-based generation of collision-free paths for robotic manipulators. (preprint, will be published in Control Engineering Practice), December 1993.

[Col75] George E. Collins. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. *Lecture Notes in Computer Science*, 33:135–183, 1975.

[Cra86] John J. Craig. *Introduction to Robotics, Mechanics and Control.* Addison-Wesley Publishing Company, 1986.

[Cri85] Arthur J. Critchlow. *Introduction to Robotics.* Macmillan Publishing Company, New York, 1985.

[CV95] Marco Ceccarelli and Adalberto Vinciguerra. On the workspace of general 4R manipulators. *The International Journal of Robotics Research*, 14(2):152–160, April 1995.

[CZ93] G. Conte and Romolo Zulli. Hierarchical path planning in a multi-robot environment with a simple navigation function. Submitted to IEEE Transactions on SMC, December 1993.

[DHX91] Han Ding, Lou Lin Huang, and You Lun Xiong. Computer-aided off-line planning of robot motion. *Robotics and Autonomous Systems*, 7:67–72, 1991.

[Dit87] Kurt Dittenberger. How to enclose boxes, cylinders, cones, and pyramids using superellipsoids. Technical Report 87-34.1, RISC-Linz, Joh. Kepler University Linz, Austria, 1987.

[DMO91] Claudio De Medio and Giuseppe Oriolo. Robot obstacle avoidance using vortex fields. In S. Stifter and J. Lenarčič, editors, *Advances in Robot Kinematics.* Springer-Verlag, 1991.

[DMT91] Leo Dorst, Indur Mandhyan, and Karen Trovato. The geometric representation of path planning problems. *Robotics and Autonomous Systems*, 7:181–195, 1991.

[DX89] Bruce Donald and Patrick Xavier. A provably good approximation algorithm for optimal-time trajectory planning. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 958–963, 1989.

[Ede87] Herbert Edelsbrunner. *Algorithms in Combinatorial Geometry.* Springer-Verlag, 1987.

[Erd92] Michael Erdmann. Randomization in robot tasks. *The International Journal of Robotics Research*, 11(5):399–436, October 1992.

[Erd94] Michael Erdmann. On a representation of friction in configuration space. *The International Journal of Robotics Research*, 13(3):240–271, June 1994.

[Fer94] Vincenzo Ferrucci. Measuring in configuration space via computing in the workspace. Technical Report TR 09-94, Dip. di Informatica e Sistemistica, Università di Roma 'La Sapienza', 1994.

[FL91] Jyh-Jong Fu and R. C. T. Lee. Voronoi diagrams of moving points in the plane. *International Journal of Computational Geometry and Applications*, 1(1):23–32, 1991.

[FS88] Kikuo Fujimura and Hanan Samet. Path planning among moving obstacles using spacial indexing. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1662–1667, 1988.

[FS89] Kikuo Fujimura and Hanan Samet. A hierarchical strategy for path planning among moving obstacles. *IEEE Transactions on Robotics and Automation*, 5(1):61–69, February 1989.

[FT88] B. Faverjon and P. Tournassoud. Motion planning for manipulators in complex environments. In *Lecture Notes in Computer Science 391, Geometry and Robotics*, pages 87–115, 1988.

[FW91] B. Fink and H.-D. Wend. Fast collision-free motion planning for robot manipulators based on parallelised algorithms. In IFAC/IFIP/IMACS, editor, *Preprints, Symposium on Robot Control (Syroco) '91*, pages 681–686, Wien, Austria, 16-18 September 1991.

[GA84] Irene Gargantini and H. H. Atkinson. Linear quadtrees: A blocking technique for contour filling. *Pattern Recognition*, 17(3):285–293, 1984.

[Gar82a] Irene Gargantini. An effective way to represent quadtrees. *Communications of the ACM, Graphics and Image Processing*, 25(12):905–910, December 1982.

[Gar82b] Irene Gargantini. Linear octtrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, 20:365–374, 1982.

[Gar83] Irene Gargantini. Translation, rotation and superposition of linear quadtrees. *Int. J. Man-Machine Studies*, 18:253–263, 1983.

[GMN95] G. Gini, R. Massa, and R. Negretti. Toward efficient path-planning for articulated robots. *Journal of Robotic Systems*, 12(2):93–104, 1995.

[GR88] B. Gorla and M. Renaud. Models of robot manipulators. In J.-D. Boissonnat and Jean-Paul Laumond, editors, *Geometry and Robotics, Lecture Notes in Computer Science*, pages 306–335, Proceedings of Workshop, Toulouse, 1988.

[GZ95] Kamal Kant Gupta and Xinyu Zhu. Practical global motion planning for many degrees of freedom: A novel approach within sequential framework. *Journal of Robotic Systems*, 12(2):105–117, 1995.

[HA92] Y. K. Hwang and N. Ahuya. Gross motion planning - a survey. *ACM Computing Surveys*, 24(3):219–292, September 1992.

[HHS93] K. Hörmann, T. Hübner, and M. Spreng. Contact analysis and verification in a 2D environment with uncertainties. *Journal of Intelligent and Robotic Systems*, 7:175–193, 1993.

[Hop86] John E. Hopcroft. The impact of robotics on computer science. *Communications of the ACM*, 29(6):486–498, 1986.

[IT90] Joan Ilari and Carme Torras. 2D path planning: A configuration space heuristic approach. *The International Journal of Robotics Research*, 9(1):75–91, February 1990.

[JC89] Paul Jacobs and John Canny. Planning smooth paths for mobile robots. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 2–7, 1989.

[KD90] Didier Keymeulen and Jo Decuyper. A reactive navigation system based on a fluid dynamics metaphor. Technical Report AI MEMO nr 90-5, Artificial Intelligence Laboratory (VUB), Vrije Universiteit Brussel, 1990.

[Kha86] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–99, Spring 1986.

[KL94] Lydia Kavraki and Jean-Claude Latombe. Randomized preprocessing of configuration space for fast path planning. In *Proceedings of the IEEE International Conference of Robotics and Automation*, pages 2138–2145, 1994.

[KS85] K. Kedem and Micha Sharir. An efficient algorithm for planning collision-free translational motion of a convex object in 2-dimensional space amidst polygonal obstacles. *ACM*, 1985.

[KS86] K. Kedem and Micha Sharir. An efficient motion planning algorithm for a convex polygonal object in 2-dimensional polygonal space. Technical Report Technical Report No. 253, Robotics Report No. 90, Courant Institute of Mathematical Sciences, New York University, Dept. of Computer Science, October 1986.

[KV88] Pradeep Khosla and Richard Volpe. Superquadric artificial potentials for obstacle avoidance and approach. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1778–1784, 1988.

[Lat91a] Jean-Claude Latombe. Motion planning: Theory and applications. *Computer Applications in Production and Engineering: Integration Aspects (IFIP)*, pages 23–32, 1991.

[Lat91b] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, 1991.

[Lau90] Jean-Paul Laumond. Nonholonomic motion planning versus controllability via the multibody car system example. Technical Report STAN-CS-90-1345, Department of Computer Science, Stanford University, Stanford, California 94305, December 1990.

[Lau94] Jean-Paul Laumond. Non-holonomic motion planning. Presentation on the UIA-Symposium, Mathematical Methods of Technology, May 4-6 1994.

[LC94] Jang Gyu Lee and Hakyoung Chung. Global path planning for mobile robot with grid-type world model. *Robotics and Computer-Integrated Manufacturing*, 11(1):13–21, 1994.

[Lea91] Doug Lea. *User's Guide to the GNU C++ Library*. Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA, version 2 edition, 1991.

[LP81] Tomás Lozano-Pérez. Automatic planning of manipulator transfer movements. *IEEE Transactions on Systems, Man and Cybernetics*, SMC-11(10):681–698, October 1981.

[LPF89] Gregory L. Long, Richard P. Paul, and William D. Fisher. The Hamilton wrist: A four-revolute-joint spherical wrist without singularities. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 902–907, 1989.

[LPW79] Tomás Lozano-Pérez and M. A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Communications of the ACM*, 22(10):560–570, October 1979.

[MA84] Fumio Miyazaki and Suguru Arimoto. Sensory feedback based on the artificial potential for robots. In *Proceedings 9th IFAC, Budapest*, Hungary, 1984.

[Man93] Ana Maria Mandache. Applications of polynomial systems solving in geometric modeling and robotics – an annotated bibliography –. Technical Report 93-65, RISC-Linz, Johannes Kepler University, Linz, Austria, 1993.

[MB88] Walter Meyer and Powell Benedict. Path planning and the geometry of joint space obstacles. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 215–219, 1988.

[NB91] Wyatt S. Newman and Michael S. Branicky. Real-time configuration space transforms for obstacle avoidance. *The International Journal of Robotics Research*, 10(6):650–667, December 1991.

[NNA89] Hiroshi Noborio, Tomohide Naniwa, and Suguru Arimoto. A feasible motion-planning algorithm for a mobile robot on a quadtree representation. In *IEEE Conference on Robotics and Automation*, pages 327–332, 1989.

[NP92] J. Nievergelt and F. P. Preparata. Plane-sweep algorithms for intersecting geometric figures. *Communications of the ACM*, 25(10):739–747, October 1992.

[OŠ94] Mark H. Overmars and P. Švestka. A probabilistic learning approach to motion planning. Technical Report RUU-CS-1994-03, Department of Computer Science, Utrecht University, January 1994.

[Ove92a] Mark H. Overmars. Point location in fat subdivisions. *Information Processing Letters*, 44:261–265, 1992.

[Ove92b] Mark H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Department of Computer Science, Utrecht University, October 1992.

[OWW82] Thomas Ottman, Peter Widmayer, and Derick Wood. A fast algorithm for boolean mask operations. Technical Report Bericht 112, Institut für Angewandte informatik und formale beschreibungsverfahren, Universität Karlsruhe, June 1982.

[Pau81] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control.* MIT Press, Cambridge, Massachusetts and London, England, 1981.

[PCH91] J.-F. Petiot, P. Chedmail, and J.-Y. Hascoet. Optimal scheduling for functional points of a trajectory generalization to cluttered environment. In IFAC/IFIP/IMACS, editor, *Preprints, Symposium on Robot Control (Syroco) '91*, pages 617–621, Vienna, Austria, 16-18 September 1991.

[Pfa92] Jochen Pfalzgraf. On mathematical modeling in robotics. Submitted to Proceedings of the AI and Symbolic Math. Comp. Conference, Karlsruhe, 1992.

[PL89] Dinesh K. Pai and M. C. Leu. Generic singularities of robot manipulators. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 738–744, 1989.

[Pou91] Behnam Pourbabai. A note on avoiding obstacles. *Robotics and Autonomous Systems*, 7:73–76, 1991.

[PS85] Franco P. Preparata and Michael Ian Shamos. *Computation Geometry, an introduction.* Springer-Verlag, 1985.

[PV84] V. V. Pavlov and A. N. Voronin. The method of potential functions for coding constraints of the external space of an intelligent mobile robot. *Soviet Automated Control*, 6, 1984.

[QCM95] Caigong Qin, Stephen Cameron, and Alistair McLean. Towards efficient motion planning for manipulators with complex geometry. In IEEE, editor, *ISATP'95*, Pittsburg, 10-11 August 1995.

[RK88] Elon Rimon and Daniel E. Koditschek. Exact robot navigation using cost functions: The case of distinct spherical boundaries in e^n. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1791–1796, 1988.

[RS87a] B. Roider and S. Stifter. Collision of convex objects. In *Proceedings of the Tenth Tagung Berichte aus Informatikinstituten*, Vienna, Austria, 1987. Oldenburg Publishing Company.

[RS87b] B. Roider and S. Stifter. Collision of convex objects in the plane. In *Proceedings EUROCAL'87*, pages 258–259, Leipzig, GDR, 1987. Springer LNCS 378.

[Sam90] Hanan Samet. *Applications of Spacial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[Sat93] Keisuke Sato. Deadlock-free motion planning using the Laplace potential field. *Advanced Robotics*, 7(5):449–461, 1993.

[Sch92] Brian Schmult. Autonomous robotic dissassembly in the blocks world. *The International Journal of Robotics Research*, 11(5):437–459, October 1992.

[SdPD88] Saif E. O. Saeed, Alan de Pennington, and John R. Dodsworth. An efficient 2D solid offsetting algorithm (geometric modelling project). Technical report, Department Of Mechanical Engineering, The University OF Leeds, Leeds LS2 9JT, England, 1988.

[Shi89] Zvi Shiller. Interactive time optimal robot motion planning and work-cell layout design. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 964–969, 1989.

[SHS87] J. T. Schwartz, J. Hopcroft, and M. Sharir. *Planning, Geometry, and Complexity of Robot Motion.* Ablex Publishing Corporation, Norwood, New Jersey, 1987.

[SL91] S. K. Singh and M. C. Leu. Manipulator motion planning in the presence of obstacles and dynamic constraints. *The International Journal of Robotics Research*, 10(2):171–187, April 1991.

[SM88] K. Sutner and W. Maass. Motion planning among time dependent obstacles. *Acta Informatica*, 26:93–122, 1988.

[Spr77] T. A. Springer. *Invariant Theory.* Number 585 in Lecture Notes in Mathematics. Springer-Verlag, 1977.

[SS87] J. T. Schwartz and M. Sharir. On the 'piano movers' problem. In J. T. Schwartz, J. Hopcroft, and M. Sharir, editors, *Planning, Geometry, and Complexity of Robot Motion*, pages 51–96, Norwood, New Jersey, 1987. Ablex Publishing Corporation.

[SS88a] J. T. Schwartz and M. Sharir. A survey of motion planning and related geometric algorithms. *Artificial Intelligence*, 37:157–169, 1988.

[SS88b] Suk-Hwan Suh and Kang G. Shin. A variational dynamic programming approach with a distance-safety criterion. *IEEE Journal of Robotics and Automation*, 4(3):334–349, June 1988.

[ST85] Hanan Samet and Markku Tamminen. Bintrees, CSG trees, and time. *Siggraph '85*, 19(3):121–130, 1985.

[Sti87] Sabine Stifter. On programming of robot manipulators (an annotated bibliography). Technical Report 87-8.2, RISC-Linz, Joh. Kepler University Linz, Austria, 1987.

[Sti88a] Sabine Stifter. An algorithm to enclose objects in a robot working cell by superellipsoids. Technical Report 88-72.0, RISC-Linz, Joh. Kepler University Linz, Austria, 1988.

[Sti88b] Sabine Stifter. A generalisation of the Roider method to solve the robot collision problem in 3D. In *Proceedings of International Symposium on Symbolic and Algebraic Computation (ISSAC '89), Rome, Italy*, pages 332–343, LNCS 358, 1988. Springer.

[Sti89] Sabine Stifter. *A Medley of Solutions to the Robot Collision Problem in 2 and 3 Dimensions.* PhD thesis, University of Linz, 1989. Also: RISC-Linz Technical Report Series, No. 88-12.0.

[Sti92a] Sabine Stifter. Collision detection in the robot simulation system SMART. *International Journal of Advanced Manufacturing Technology*, 7:277–284, 1992.

[Sti92b] Sabine Stifter. The Roider method: A method for static and dynamic collision detection. *Issues in Robotics and Non-Linear Geometry*, 1992.

[Sti93] Sabine Stifter. Research activities in robotics at RISC-Linz. Technical Report 93-25, RISC-Linz, Joh. Kepler University Linz, Austria, May 1993.

[SY87] Jacob T. Schwartz and Chee-Keng Yap, editors. *Algorithmic and Geometric Aspects of Robotics*, volume 1. Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey and London, 1987.

[Tho84] C. E. Thorpe. Path relaxation: Path planning for a mobile robot. In *Proceedings of the AAAI, Austin, Texas*, pages 318–321, Los Altos, CA, August 6-10 1984. Morgan Kaufmann Publishers.

[TMS93] Dawn Tilbury, R. Murray, and Shankar Sastry. Trajectory generation for the N-trailer problem using goursat normal form. In *32nd IEEE Conference on Decision and Control*, San Antonio, Texas, December 15-17 1993.

[Udu77] Shriram M. Udupa. Collision detection and avoidance in computer controlled manipulators. In *Proceedings of the Fifth International Joint Conference on AI (IJCAI-77)*, pages 737–748, Massachusetts, August 1977.

[Ver90] B. J. H. Verwer. A multiresolution work space, multiresolution configuration space approach to solve the path planning problem. In *Proceedings of the IEEE Conference on Robotics and Automation*, pages 2107–2112, Cincinnati, Ohio, May 1990.

[VG93] Carl Van Geem. Towards a fast solution method for the general robot motion planning problem using a Manhattan-like distance function on a non-uniform grid in configuration space. Technical Report 93-66, RISC-Linz, Joh. Kepler University Linz, Austria, December 1993.

[VG94a] Carl Van Geem. On using a Manhattan distance-like function for robot motion planning on a non-uniform grid in configuration space. Technical Report 94-74, RISC-Linz, Joh. Kepler University Linz, Austria, October 1994.

[VG94b] Carl Van Geem. Towards a fast solution method for the general robot motion planning problem using a Manhattan-like distance function on a non-uniform grid in configuration space. In *10th European Workshop on Computational Geometry*, Santander, March 17-18 1994.

[VG95a] Carl Van Geem. Computation of a non-uniform grid on the configuration space of a robot arm amidst obstacles: Intersection problem. In *11th European Workshop on Computational Geometry*, RISC-Linz, Johannes Kepler University Linz, Austria, March 23-24 1995.

[VG95b] Carl Van Geem. Counting cells in a grid. Technical Report 95-29, RISC-Linz, Joh. Kepler University Linz, Austria, 1995.

[VG95c] Carl Van Geem. Hierarchical construction of a potential field on a non-uniform grid on the configuration space of a robot arm. In *Proceedings of the 4th International Workshop on Robotics in Alpe-Adria Region (RAA'95)*, Poertschach, Austria, 6-8 July 1995.

[VG95d] Carl Van Geem. Path planning for manipulators with many degrees of freedom using a discrete representation of configuration space extensively. Technical Report 95-45, RISC-Linz, Joh. Kepler University Linz, Austria, 1995.

[VG95e] Carl Van Geem. Preparations with workspace information for meaningful robot motion planning in configuration space. Technical Report 95-14, RISC-Linz, Joh. Kepler University Linz, Austria, 1995.

[VG96a] Carl Van Geem. Enclosing robot links by superellipses and superellipsoids. Technical Report 96-01, RISC-Linz, Joh. Kepler University Linz, Austria, 1996.

[VG96b] Carl Van Geem. On using a manhattan distance-like function for robot motion planning on a non-uniform grid in configuration space. *International Journal of Flexible Automation and Integrated Manufacturing (IJFAIM)*, 1996.

[War88] Charles W. Warren. Global path planning using artificial potential fields. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 316–321, 1988.

[WCWC87] H. Wang, H. Chang, R.A. Wysk, and A. Chandawarkar. On the efficiency of NC tool path planning for face milling operations. *Journal of Engineering for Industry*, pages 370–376, May 15 1987.

[WH92] Deming Wang and Yskandar Hamam. Optimal trajectory planning of manipulators with collision detection and avoidance. *The International Journal of Robotics Research*, 11(5):460–468, October 1992.

[WTS+92] G. Walsh, Dawn Tilbury, Shankar Sastry, R. Murray, and Jean-Paul Laumond. Stabilization of trajectories for systems with nonholonomic constraints. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1999–2004, Nice, 1992.

[Zar85] M. Y. Zarrugh. Display and inertia parameters of superellipsoids as generalized constructive solid geometry primitives. In *Proceedings of the 1985 ASME International Computers in Engineering Conference and Exhibition, Boston, MA*, volume 1, pages 317–328, August 4-8 1985.

[ZM88] T. Zrimec and P. Mowforth. An experiment in generating deep knowledge for robots. In J. S. Gero, editor, *Artificial Intelligence in Engineering: Robotics and Processes*, pages 21–33. Elsevier and Computational Mechanics Publications, 1988.