Path Planning using Probabilistic Cell Decomposition

FRANK LINGELBACH

Licentiate Thesis
Stockholm, Sweden 2005

TRITA-S3-REG-0501
ISSN 1404-2150
ISBN 91-7283-961-9

KTH Signaler Sensorer och System
SE-100 44 Stockholm
SWEDEN

Akademisk avhandling som med tillstånd av Kungl Tekniska högskolan framläges till offentlig granskning för avläggande av teknologie licentiatexamen torsdagen den 10 februari 2005 i Kollegiesalen, Administrationsbyggnaden, Kungl Tekniska högskolan, Valhallavägen 79, Stockholm.

Tryck: Universitetsservice US AB

Abstract

The problem of path planning occurs in many areas, such as computational biology, computer animations and computer-aided design. It is of particular importance in the field of robotics. Here, the task is to find a feasible path/trajectory that the robot can follow from a start to a goal configuration. For the basic path planning problem it is often assumed that a perfect model of the world surrounding the robot is known. In industrial robotics, such models are often based on, for example, CAD models. However, in applications of autonomous service robotics less knowledge about the environment is available. Efficient and robust path planning algorithms are here of major importance. To be truly autonomous, a robot should be able to plan all motions on its own. Furthermore, it has to be able to plan and re-plan in real time, which puts hard constraints on the acceptable computation time.

This thesis presents a novel path planning method called Probabilistic Cell Decomposition (PCD). This approach combines the underlying method of cell decomposition with the concept of probabilistic sampling. The cell decomposition is iteratively refined until a collision-free path is found. In each immediate step the current cell decomposition is used to guide probabilistic sampling to important areas.

The basic PCD algorithm can be decomposed into a number of components such as graph search, local planning, cell splitting and probabilistic sampling. For each component different approaches are discussed. The performance of PCD is then tested on a set of benchmark problems. The results are compared to those obtained by one of the most commonly used probabilistic path planning methods, namely Rapidly-exploring Random Trees. It is shown that PCD efficiently solves various kinds of path planning problems.

Planning for autonomous manipulation often involves additional path constraints beyond collision avoidance. This thesis presents an application of PCD to path planning for a mobile manipulator. The robot has to fetch a carton of milk from the refrigerator and place it on the kitchen table. Here, opening the refrigerator involves motion with a pre-specified end-effector path. The results show that planning the different motions for the high-level task takes less time than actually executing them. The whole series of subtasks takes about 1.5 seconds to compute.

Acknowledgements

First of all, I want to thank my academic advisors, Professor Bo Wahlberg and Professor Henrik I. Christensen for giving me the opportunity to work at S3 and CAS. Your doors are always open for a discussion. Thanks for all the helpful comments and letting me do my thing even if the pre-planned path of my PhD studies looked quite different from the one I am currently following.

Many thanks go to Danica Kragić, for pointing in the right directions, for her enthusiasm and valuable help regarding all issues from proof reading to taming Obelix.

Special thanks go to Morten and Daniel for all the discussions on path planning and related stuff. Not to forget: Daniel, thanks for all debugging, driver fixing, cvs support and so on. I want to thank all my colleagues at S3 and CAS for creating such a pleasant atmosphere. Thanks to Karin, Stex and support@s3 for making life at KTH a lot easier.

Finally, I want to thank my girlfriend, Tanja, for being there even when being far away, for all her support and love.

This research has been sponsored by the Swedish Foundation for Strategic Research through the Centre for Autonomous Systems (CAS). The support is gratefully acknowledged.

Contents

List of Figures

Chapter 1

Introduction

In this chapter, we give a brief introduction to the thesis. After presenting some motivating examples where path planning methods are used to solve problems from very different fields of research, we present the outline of the thesis and state the main contributions.

1.1 Motivation

Applications of path planning exist in many areas. The field of application we have in focus when designing PCD is path planning for an autonomous service robot in a home or office environment.

Mobile Robotics

In the field of autonomous, mobile robotics, the ability to plan its own motions is essential for a robot to be truly autonomous. It has to react to its environment and carry out user-defined instructions. Nearly every task the robot has to accomplish involves a motion from one configuration to another. Each time, this raises the question of how to move each joint in a feasible manner, such that the robot reaches the goal state and avoids collisions with obstacles on its way. It is evident that predefined controls are not sufficient to comply with these requirements in a dynamic environment. On the other hand, the user does not want the robot to mull over a feasible path for minutes when assigning a simple task to it. There is an obvious need for efficient path planning algorithms.

To be able to plan its motions, the robot needs an accurate model of itself and the world surrounding it. The first demand is often possible to meet. CAD-models of the robot exist and sensors are precise enough to state the actual configuration of the robot. In other cases, as can be seen from Figure 1.1, parts of the robot might be

very hard to model correctly. In this example, the cables reaching from the Barrett Hand to the arm are flexible and thus difficult to be taken into consideration. Here, one could build a conservative model that includes all points that the cables could possibly reach or a more optimistic model that disregards the cables at the planning stage.

The problem of obtaining a model of the environment is much harder. In a home or office environment, almost all objects are movable. Some are quasi-static, like big pieces of furniture, some objects change status within a well defined set of possible configurations, like doors. If the robot shares the environment with humans, some objects – besides the human itself – will dynamically change their positions. Obviously, a static model will not suffice for path planning, but the robot has to react to sensor information. With substantial progress in the field of stereo vision and a big drop in prices for 3D sensors such as, for example, laser scanners, the practicability of path planning within mobile manipulation will increase significantly.

Figure 1.1: Left Figure: Obelix platform with PUMA 560 arm on XR4000 base; Right Figure: Computer model of Obelix platform used for path planning

Industrial Robotics

For industrial robots the focus of demands on the path planning algorithm is shifted towards other issues. Often a robot is programmed only once and then operates for

years carrying out the same task. In this field, CAD models of the robot and the work cell it is working in are most often available. The work cell is specially designed to contain as few obstacles as possible. There might exist dynamic obstacles in the vicinity of the robot but their motions are known beforehand. An evident example is a work piece on a conveyor belt that has to be processed by the robot.

For industrial applications, a major aspect is optimality of the obtained path. As the robot repeats the same motion over and over again, savings in execution time, energy consumption or similar measures sum up to huge amounts of money. Optimality is typically not considered by probabilistic path planning algorithms. Instead, the first feasible path that is found is returned and modified in a subsequent optimization step. Time efficiency of the path planning algorithm is clearly important – even if there are no real-time requirements. Using automatic path planning algorithms for robot programming can save many working hours of experienced engineers. Another key aspect is that the robot can be programmed off-line if an accurate model is available. The assembly line does not have to be stopped, which can result in notable savings.

Computational Biology

Applications for path planning algorithms exist also in fields that are rather unrelated to robotics at a first glance. In computational biology, path planning algorithms are applied on a molecular scale. Ligands (molecules that bind to a receptor protein), for example, have structural similarities with articulated robots. If a reference atom is kept fixed, the configuration of the ligand is determined by the state of several rotational degrees of freedom. Singh *et al.* (1999) use this fact to apply robot path planning algorithms to study the dynamics of the process of ligand binding. These dynamics are important in the context of drug design.

Another application within the field of computational biology is the investigation of protein folding mechanisms. Long protein molecules naturally appear in a folded low-energy state. Song and Amato (2001) study the folding pathways of proteins with help of path planning methods. As the function of a protein is determined by its three-dimensional structure, understanding the folding process is significant.

Computer Animations

In the area of computer animations, there exist several applications for path planning algorithms. As a natural advantage in this area a perfect world model exists. Thus, path planning algorithms can be applied without being confronted with the sensing and perception problems of mobile robotics. Computer games is a challenging field within this area. Characters that should follow the user's instructions have to be animated in realtime. Pre-computed animations are often feasible but they restrict the free hand of the user. Beneath requirements on a path that are

similar to those of the field of robotics, such as collision avoidance, character motions should look natural to the human eye. Nieuwenhuisen and Overmars (2002) compute smooth paths for camera movements in virtual environments. Kuffner and Latombe (2000) use path planning techniques to animate manipulation tasks. Pettré *et al.* (2002) present a method based on path planning for animation of human characters.

Assembly Planning

Assembly planning has several applications, for example, in virtual prototyping or product maintenance. Here, path planning can answer questions like: Is it possible to assemble the product from its pieces? In which order must the pieces be assembled? Closely related is the problem of disassembly planning (Sundaram *et al.* 2001). If some part of an assembled product is broken, what is the best way to replace it? Those problems can be solved using path planning techniques. Each piece of the assembly is treated as a free-flying rigid body. Then, a path is planned from some initial configuration to the assembled goal state.

1.2 Thesis Outline

Chapter 2

Path planning has been a very active field of research for more than two decades. The motivating examples in this chapter show the need for efficient path planning methods. In Chapter 2, we present the path planning problem. We give a brief overview over the most important classical methods. Then, we present the two probabilistic approaches that have drawn major attention during the last years. Both approaches, Probabilistic Roadmap Methods (PRM) and Rapidly-exploring Random Trees (RRT) have been shown to be able to solve high dimensional problems in acceptable computation times. We review both methods and present current research topics regarding these two methods.

Chapter 3

In Chapter 3, we introduce the basic ideas of Probabilistic Cell Decomposition (PCD). It combines the concept of probabilistic sampling with the underlying method of cell decomposition. The elementary components of the algorithm are presented in detail. The performance of the algorithm is tested on the set of benchmark problems presented in Appendix A. These problems are chosen from different fields of applications including maze-like problems, rigid body problems and path planning for a mobile manipulator. The problem dimension ranges from two to forty-eight.

Chapter 4

Possible modifications to the elementary components of the basic algorithm of PCD are discussed in Chapter 4. The components we investigate are graph search, probabilistic sampling, cell shape and cell splitting. For each modification, we compare the computation results with those of the basic algorithm.

Chapter 5

Chapter 5 presents an application of the general path planning method PCD developed in Chapters 3 and 4 to the field of mobile manipulation. In a task specified by the high level command "place the milk on the kitchen table" the mobile robot has to plan several consecutive motions. It has to open the refrigerator, get the milk, place it on the table and close the refrigerator again. Here, we neglect related problems like, for example, task decomposition and focus on the path planning problem. Opening and closing the refrigerator door involves a motion with constrained end-effector movement. We discuss this subproblem briefly.

Chapter 6

Chapter 6 concludes this thesis by discussing results of the preceding chapters. Additionally, we give some prospects for future research. We outline directions of future research regarding PCD as a general path planning method and in particular for using PCD for path planning for mobile manipulation.

1.3 Main Contributions

The main contributions of this thesis are:

- A novel approach to probabilistic path planning called Probabilistic Cell Decomposition (PCD); This new approach combines probabilistic sampling with the underlying method of cell decomposition
- Extensive studies on different components of PCD, such as graph search, probabilistic sampling, cell shape and cell splitting

The results contained in this thesis have been presented at several international conferences:

- Frank Lingelbach, "Path Planning using Probabilistic Cell Decomposition", *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, New Orleans, LA, 2004

- Frank Lingelbach, "Path Planning for Mobile Manipulation using Probabilistic Cell Decomposition", *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, Sendai, 2004

- Daniel Aarno, Frank Lingelbach, Danica Kragić, "Constrained Path Planning and Task-consistent Path Adaptation for Mobile Manipulators", submitted to *IEEE International Conference on Advanced Robotics (ICAR)*, Seattle, 2005

An additional paper that is not considered in this thesis as it is not related to path planning:

- Vikram Krishnamurthy, Bo Wahlberg, Frank Lingelbach, "A Value Iteration Algorithm for Partially Observable Markov Decision Process Multi-armed Bandits", *Sixteenth International Symposium on Mathematical Theory of Networks and Systems (MTNS)*, Leuven, 2004

Chapter 2

Path Planning – an Overview

In this chapter, we will give a broad overview on the field of path planning. After a short introduction to the problem, we will present the most important classical methods and recent work on probabilistic path planning.

2.1 Problem Formulation

The problem of path planning appears in many different forms that are only loosely connected at first. An instance of the problem could, for example, be a point like agent that has to traverse a maze, an articulated robot that has to move from one configuration to another or path planning for a free-flying rigid body. This last instance is also known as the *Piano Movers' Problem* (Schwartz and Sharir 1983): Given a current and a desired final position of the piano, find a continuous collision-free path for the piano, connecting these two positions. A more complicated variant of this problem includes multiple free-flying rigid bodies that have to avoid collisions with static obstacles and themselves. This kind of problem is typical for the field of assembly planning. Additional examples from the previous chapter include planning for large molecules or animated characters.

2.1.1 The Configuration Space

All example problems given above have in common that some kind of agent has to move from some initial to a goal configuration without colliding with obstacles on its way. This leads to the configuration space concept.

The configuration of the agent is completely determined by n values, where n is the number of the agent's degrees of freedom (dof). For example, the state of a point-like agent in a 2D maze is completely determined by two parameters, the x- and y-coordinates. A free-flying rigid body in a three-dimensional workspace has six

dof, three translational and three rotational. Thus, each configuration corresponds to a point q in an n-dimensional space which is called the configuration space $\mathcal{C}$. The space where the agent "lives" is called the workspace $\mathcal{W}$. Typically, $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$. The image of the agent at the configuration q in $\mathcal{W}$ is denoted by $\mathcal{A}(q)$.

A requirement on a feasible path is naturally that it is collision-free. The agent in a maze example is not allowed to tunnel through the walls, a robot should better not collide with any obstacle. Thus, the obstacles in the work space $\mathcal{W}$ have to be translated to the configuration space $\mathcal{C}$. A configuration $q \in \mathcal{C}$ is called *colliding* if $\mathcal{A}(q)$ intersects with an obstacle in $\mathcal{W}$. Let the closed set $\mathcal{O} \subset \mathcal{W}$ denote the obstacle region, namely the set of all points in the work space $\mathcal{W}$ that belong to an obstacle. Then, the set of configuration space obstacles $\mathcal{C}_{\text{obst}}$ is defined as all colliding configurations in $\mathcal{C}$. The complement of $\mathcal{C}_{\text{obst}}$ in $\mathcal{C}$ is the set of collision-free configurations $\mathcal{C}_{\text{free}}$, called collision-free configuration space or simply *free space*,

$$\mathcal{C}_{\text{obst}} = \{q \in \mathcal{C} \mid \mathcal{A}(q) \cap \mathcal{O} \neq \emptyset\} \tag{2.1}$$

$$\mathcal{C}_{\text{free}} = \mathcal{C} \setminus \mathcal{C}_{\text{obst}} \tag{2.2}$$

As both $\mathcal{A}(q)$ and $\mathcal{O}$ are closed sets, also $\mathcal{C}_{\text{obst}}$ is a closed set. This definition has to be extended, for example, for multi rigid body problems or planning for some articulated robots. Depending on the structure of the robot, it might be possible that for some configurations one link collides with another. These *self-collisions* can simply be added to $\mathcal{C}_{\text{obst}}$. Two examples for possible workspace - configuration space pairs are shown in Figure 2.1. For the 2D robot, colliding configurations occur, when the inner or the outer link penetrates the platform or the ceiling. The colliding region in the center of $\mathcal{C}$ corresponds to a collision of the outer link with the ceiling. Only if the inner link points almost straight upwards, the outer link can reach the ceiling. The colliding regions to the left and to the right correspond to collisions with the platform. Obviously, if for some configuration the inner link collides with the platform, the outer joint angle is irrelevant – the state is colliding.

For the lower problem in Figure 2.1, we assume that the triangular agent may translate freely in the plane but it is not allowed to rotate. Thus, also this problem has two degrees of freedom, namely the x- and y-coordinate of the agent. To construct the configuration space $\mathcal{C}$, we choose a reference point on the agent and define a state q in $\mathcal{C}$ as free if the agent does not collide with any obstacle in $\mathcal{W}$, when the reference point is placed at q. As it can be seen in Figure 2.1, the shapes of the obstacles in $\mathcal{C}$ are here strongly related to the shape in $\mathcal{W}$ – just blown up by the size of the agent. To be more specific, the $\mathcal{C}$-obstacles are obtained by taking the Minkowsky sum of the $\mathcal{W}$-obstacles and the agent. Consequentially, one of the passages between the two large open regions in $\mathcal{W}$ is not traversable, as the agent is too large.

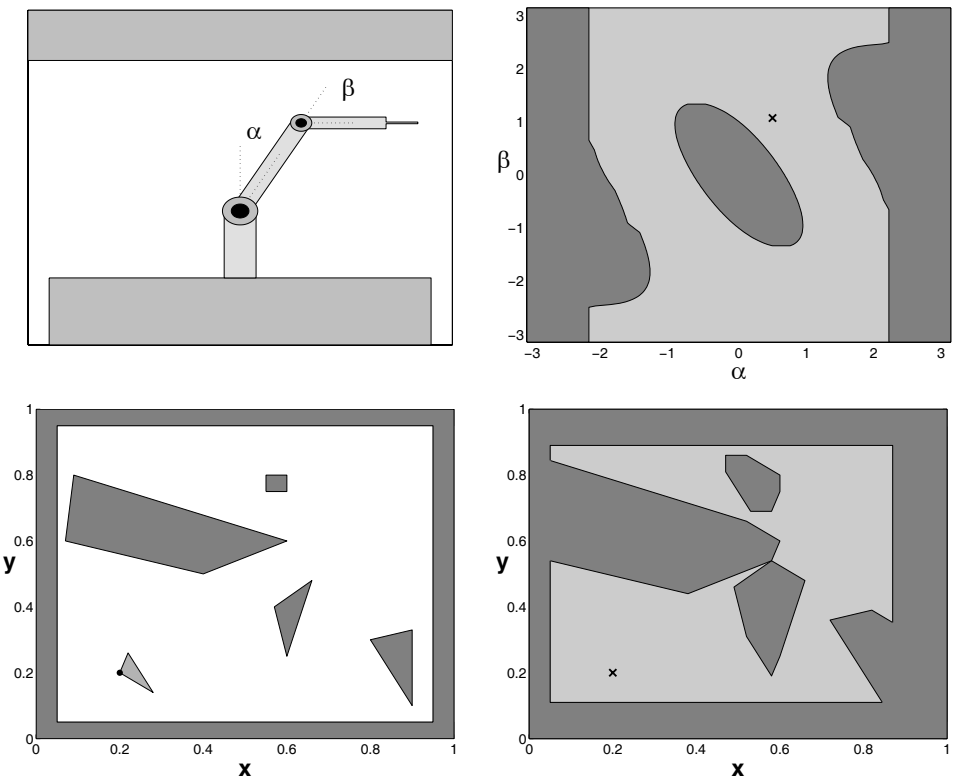


Figure 2.1: Upper Left Figure: Articulated 2 dof Robot, work space $\mathcal{W}$; Upper Right Figure: Corresponding 2D configuration space, $\mathcal{C}_{\text{obst}}$: dark gray, $\mathcal{C}_{\text{free}}$: light gray, configuration shown in $\mathcal{W}$ marked with x; Lower Left Figure: Triangular rigid body allowed to translate but not to rotate; reference point marked with $\bullet$; Lower Right Figure: Corresponding 2D configuration space, $\mathcal{C}_{\text{obst}}$: dark gray, $\mathcal{C}_{\text{free}}$: light gray, configuration shown in $\mathcal{W}$ marked with x;

Not only the dimension but also the topology of $\mathcal{C}$ is of particular importance. For the robot problem, it has to be decided at the modeling stage whether the outer link is allowed to rotate freely several turns or if there are hard boundaries such that the link is allowed to rotate only in the interval $[-\pi, \pi]$. If the outer link is allowed to rotate freely, the upper boundary for β is identified with its lower boundary. Thus, a path in $\mathcal{C}$ can "leave" the configuration space at $\beta = \pi$ and re-enter at $\beta = -\pi$ at the same α value. For a real robot, the internal structure typically gives rise to limits also on the rotational degrees of freedom. In contrast,

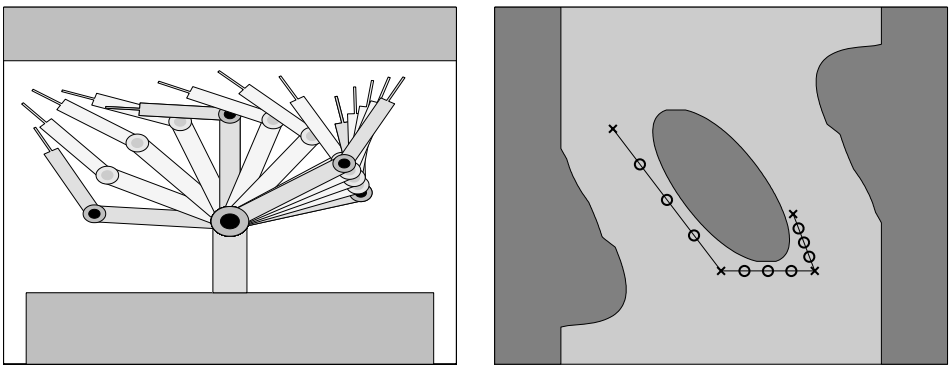


Figure 2.2: Left Figure: Feasible path for 2D robot in work space $\mathcal{W}$; Right Figure: Corresponding continuous path in the collision-free configuration space $\mathcal{C}_{\text{free}}$; Configurations marked with x and o in $\mathcal{C}_{\text{free}}$ are plotted dark and light in $\mathcal{W}$, respectively.

a free-flying rigid body is usually allowed to rotate unlimited. Thus, the three-dimensional configuration space of a free-flying rigid body in the plane is given as $\mathcal{C} = \mathbb{R}^2 \times \mathbb{S}$ which can be visualized as a hollow cylinder. The configuration space of a free-flying rigid body in $\mathcal{W} = \mathbb{R}^3$ is six-dimensional, namely $\mathbb{R}^3 \times \text{SO}(3)$. For an in-depth discussion of $\mathcal{C}$-topology, see (LaValle 2004).

Returning to the problem of path planning, the initial and the goal configuration of the agent are represented by two points in $\mathcal{C}$. A continuous path in $\mathcal{C}$ corresponds to a continuous motion of the agent in $\mathcal{W}$ and, accordingly, a continuous path in $\mathcal{C}_{\text{free}}$ corresponds to a collision-free motion in $\mathcal{W}$, connecting the initial with the goal configuration. Thus, all path planning problems, that looked so different at first glance, can be solved by finding a continuous path for a point like agent in $\mathcal{C}_{\text{free}}$. Figure 2.2 shows snapshots of a continuous motion of the 2D robot and the corresponding continuous path in $\mathcal{C}_{\text{free}}$.

An important concept of path planning is completeness. A path planning algorithm is said to be *complete* if, for any planning task, it returns a feasible path or the correct answer that no feasible path exists in finite time. Complete algorithms exist, but their area of application is limited to very simple problems as the underlying "generalized mover's problem" is shown to be PSPACE hard (Latombe 1991). A weaker concept is probabilistic completeness. A path planning algorithm is called *probabilistically complete* if, for the case that a feasible path exists, the probability that the algorithm solves the problem approaches one as computation time goes to infinity. The biggest drawback of a method which is only probabilistically complete is the fact that it can not decide whether a problem is not solvable. If a proba-

bilistically complete method is used on an unsolvable problem, it will simply run forever.

2.1.2 Holonomic vs. Nonholonomic Path Planning

In the basic path planning problem, dynamics of the agent are neglected. The time dimension is modelled implicitly by implying that successive configurations of a planned path have to be traversed one after another. As there are no constraints on the dynamics, the agent is free to move in any direction in $\mathcal{C}$, aside from possible restrictions due to configuration space obstacles. This is referred to as holonomic path planning. In this thesis, we will only consider planning for holonomic path planning problems.

There might exist kinematic constraints that restrict the possible motions of an agent. In the field of path planning, typically three kinds of constraints are considered:

- holonomic constraints
- nonholonomic constraints
- kinodynamic constraints

Holonomic constraints are equality constraints among the parameters of a configuration. A holonomic constraint reduces the dimension of $\mathcal{C}$ by one but the problem stays holonomic. The concept of holonomic constraints can be visualized using the articulated 2D robot shown in Figure 2.1. The base including the shoulder of the robot is not able to move by any means. Thus, we do not have to plan any actions for the base. The remaining part of the robot consists of two rigid links – two rigid bodies. Consequentially, we can think of the planning problem as a multi rigid body problem, where the two links are able to translate and rotate freely in $\mathcal{W} = \mathbb{R}^2$. According to this, each link has 3 dof giving rise to a configuration space of dimension 6. Though, the revolute joint between the inner link and the shoulder determines two holonomic constraints, namely that the x- and y-coordinates of the joint position on the link have to coincide with the x- and y-coordinates of the joint position on the shoulder. If the reference point on the inner link is chosen at the joint location, this is easy to see. Otherwise, if the reference point is chosen, for example, as the center of the link, at a distance of dx_1 and dy_1 from the joint in upright position, the holonomic equality constraints can be solved for the position of the reference point.

$$\begin{aligned} x_{rp_1} &= x_{bj} + dx_1 \sin(\alpha) \\ y_{rp_1} &= y_{bj} + dy_1 \sin(\alpha) \end{aligned}$$

where x_{rp_1}, y_{rp1} is the position of the reference point and x_{bj}, y_{bj} is the position of the base joint. When α is given and the position of the reference point calculated, the position of the joint connecting link one and link two is also fixed (x_{lj}, y_{lj}). Similar to the first link, the position of an arbitrarily chosen reference point on the second link can be calculated using

$$\begin{aligned} x_{rp_2} &= x_{lj} + dx_2 \sin(\alpha + \beta) \\ y_{rp_2} &= y_{lj} + dy_2 \sin(\alpha + \beta). \end{aligned}$$

Nonholonomic constraints involve not only the state but also state derivatives. According to the definition in (Latombe 1991) a nonholonomic constraint is a non-integrable scalar constraint of the form

$$G(q, \dot{q}, t) = 0,$$

where G is a smooth function of the configuration q, its derivative $\dot{q}$ and time. Nonintegrability implies that the derivative parameters $\dot{q}$ can not be eliminated by integration. A nonholonomic constraint does not reduce the dimension of $\mathcal{C}$ but restricts the set of possible motions in $\mathcal{C}$. An illustrative example is a car-like robot as can be seen in Figure 2.3. It can take any position and any orientation in the plane. Thus, its configuration space is three-dimensional. But, due to nonholonomic constraints, the space of possible differential motions in $\mathcal{C}$ is only two-dimensional, namely position changes along the steering direction and orientation changes.

Kinodynamic constraints additionally include constraints on state accelerations. Path planning problems involving nonholonomic or kinodynamic constraints are called nonholonomic or kinodynamic path planning problems, respectively.

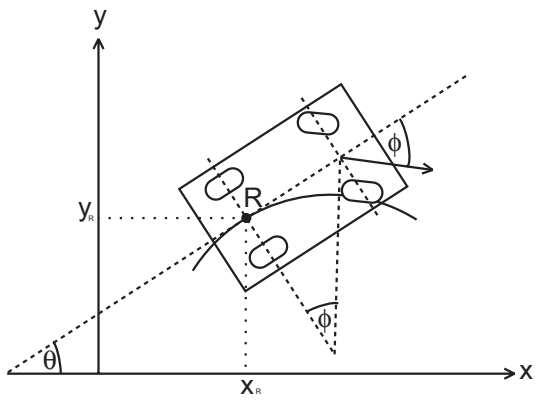


Figure 2.3: Car-like robot with nonholonomic constraint: $-\sin\theta\, dx + \cos\theta\, dy = 0$

2.2 Classical Path Planning Methods

Here, we present the three most successful classical approaches to robot path planning,

- Cell Decomposition
- Roadmap Methods
- Potential Field

For each approach, we present the basic ideas and a few different realizations. For these methods, an explicit representation of $\mathcal{C}$ is assumed to be known. See (Latombe 1991), for an extensive overview on the field of classical path planning.

2.2.1 Cell Decomposition

The idea behind cell decomposition methods is to decompose the configuration space $\mathcal{C}$ into a number of disjoint sets, called cells. An important element of cell decomposition methods is the connectivity graph G that captures the structure of $\mathcal{C}$. Each cell is represented as a node in this graph. Two nodes are connected by an edge if and only if the two corresponding cells are adjacent.

Cell decomposition methods can be classified as *exact* or *approximate*. The major difference is that exact methods generate an exact decomposition of the free configuration space $\mathcal{C}_{\text{free}}$, whereas approximate methods try to approximate the structure of $\mathcal{C}$ with cells that have a simple shape like, for example, rectangloids. Using these simple shapes, it is rarely possible to capture the exact shape of $\mathcal{C}_{\text{free}}$.

Exact Cell Decomposition

Characteristic for exact cell decomposition methods is that the union of all cells equals the closure of the collision-free configuration space. A path connecting an initial with a goal configuration is found by searching G for a cell path connecting the cell containing the initial configuration with the cell containing the goal configuration. Such a cell path is also called *channel*. Then, as the interior of the channel lies entirely in free space, a continuous path from the initial to the goal configuration can be found. Figure 2.4 shows an example of trapezoidal decomposition.

Approximate Cell Decomposition

In approximate cell decomposition methods, all cells have a simple predefined shape.The standard cell shape is the rectangloid, in n dimensions defined by

$$\{(x_1, \ldots, x_n) \mid x_1 \in [x_1', x_1''], \ldots, x_n \in [x_n', x_n'']\}$$

The most commonly used decomposition technique is the 2^m-tree decomposition, where m is the dimension of the configuration space. A cell that is not entirely contained in $\mathcal{C}_{\text{free}}$ or the complement of $\mathcal{C}_{\text{free}}$ in $\mathcal{C}$, $\mathcal{C}_{\text{obst}}$, is called *mixed* and is split up into 2^m subcells. These are then recursively refined in the same manner until a path in G has been found or a given minimum resolution is reached. This indicates that the complexity of these algorithms grows exponentially in m constraining the applicability for planning in high-dimensional spaces. A visualization of this approach is shown in Figure 2.4.

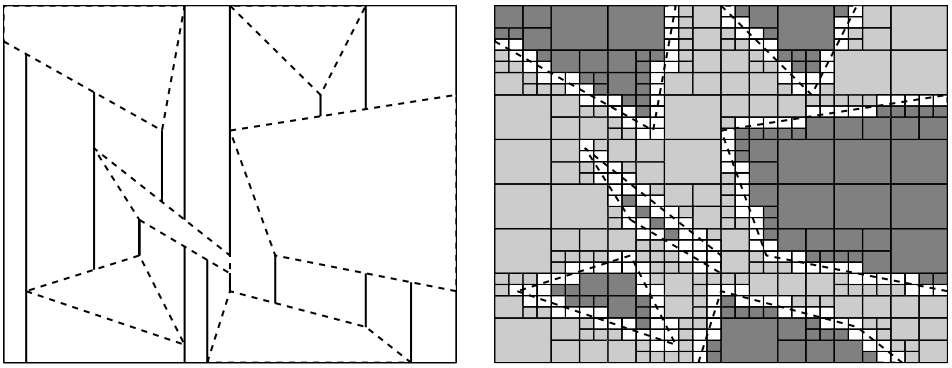


Figure 2.4: Left Figure: Exact cell decomposition, $\mathcal{C}_{\text{free}}$ exactly decomposed into trapezoids; Right Figure: Approximate cell decomposition, mixed cells are divided until a series of free cells connects the start with the goal cell; free cells: light gray, obstacle cells: dark gray, mixed cells: white, obstacles (- -)

2.2.2 Roadmap Methods

The basic idea of roadmap methods is to create a roadmap that reflects the connectivity of $\mathcal{C}_{\text{free}}$. If such a roadmap exists for a given configuration space, the problem of solving a path planning query is reduced to connecting the initial and the goal configuration to the roadmap. If both configurations can be connected to the same connected component of the roadmap, a feasible path is found by simply following the path in the roadmap. Otherwise, if the connectivity of the roadmap correctly reflects the connectivity of $\mathcal{C}_{\text{free}}$, no feasible path exists. To construct the roadmap, several approaches have been proposed.

For a set of polygonal configuration space obstacles S in the plane it can be shown that the shortest path from any initial configuration to any goal configuration is polygonal and its inner vertices are vertices of S (de Berg *et al.* 1997). Taking the vertices of S as the nodes of the roadmap and connecting two nodes by a straight

line path if this is collision-free, yields an *optimal* roadmap. Optimal in the sense that the inner edges of the optimal polygonal path are contained in the roadmap and it remains only to connect the initial and the goal configuration to the "right" vertices. The roadmap built in that way is called the *visibility graph* of S. It is actually sufficient to plan on the *reduced visibility graph*, which is obtained by only considering those edges that are tangent to the obstacle at both vertices. An edge is called tangent to an obstacle at a vertex if the infinite line through this edge is tangent to the obstacle in a neighborhood around the vertex. An example is shown in the left part of Figure 2.5. It clearly depends on the path planning problem at hand whether this definition of an optimal roadmap is suitable or not. Aside from trivial problems, where the straight line path between initial and goal configuration is collision-free, the minimum clearance between the agent and an obstacle along the shortest path is zero, by construction of the roadmap. This is clearly undesirable, for example, for robotic path planning problems, where it is often reasonable to keep a safety margin.

For this kind of problems, it might be preferable to choose a path that maximizes the clearance. The so called Voronoi diagram of the configuration space is the set of collision-free configurations, whose minimal distance to $\mathcal{C}_{\text{obst}}$ is achieved with at least two points on the boundary of $\mathcal{C}_{\text{obst}}$ (Latombe 1991).

$$\mathcal{V} = \Big\{ q \in \mathcal{C}_{\text{free}} \quad | \quad d = \min_{q' \in \mathcal{C}_{\text{obst}}} \text{dist}(q, q'), \\ \exists q', q'' \in \mathcal{C}_{\text{obst}} \quad q' \neq q'', d = \text{dist}(q, q') = \text{dist}(q, q'') \Big\}$$

As can be seen from the example shown in the right part of Figure 2.5, if the agent moves along the Voronoi diagram, it keeps a maximum distance to all $\mathcal{C}$-obstacles.

2.2.3 Potential Field Methods

Potential field methods try to guide the agent from the initial configuration q_{init} to the goal configuration q_{goal} using an artificial potential field. The agent can follow a virtual force defined at each point by the gradient of the potential field. Typically, the potential field is composed of one field attracting the agent to the goal configuration and one field repelling the agent from configuration space obstacles. Figure 2.6 shows a simple example using a parabolic field with minimum in the goal state and one repelling from obstacles. As can be seen from the color coding, the global optimum is located at the goal state. Nevertheless, the agent might not find its way to the goal but get stuck in a local minimum. If the attractive and the repelling field are simply added together, the total gradient might sum up to zero at some configurations. If the agent is attracted to such a configuration, it will stay there, as the virtual force driving the agent vanishes in this point. A number of different potential fields have been proposed to reduce the number of

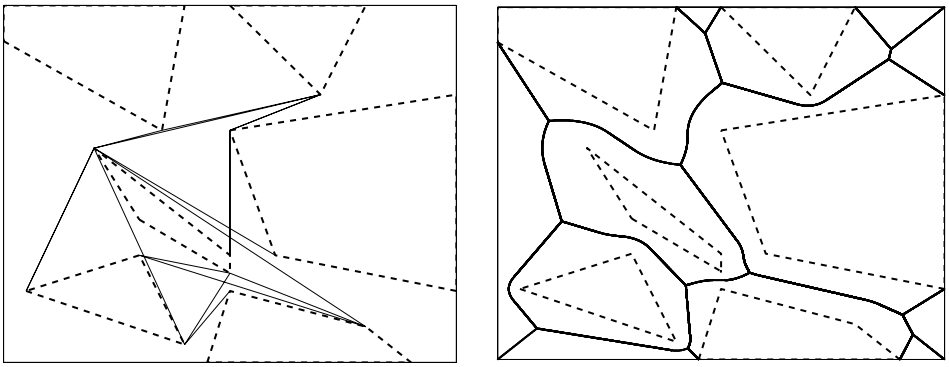


Figure 2.5: Left Figure: Roadmap obtained from reduced visibility graph, edges running along obstacles not shown, all obstacle edges that are not in contact with the configuration space boundary belong to the roadmap; Right Figure: Roadmap obtained from Voronoi diagram where configuration space boundary is regarded as obstacle

local minima and the size of their region of attraction, see (Latombe 1991). Another approach to deal with local minima is the *randomized path planner* (RPP) proposed by Barraquand and Latombe (1990). Here, randomization is used in a way different from the probabilistic methods presented in the next section. The RPP starts a *random walk* each time it encounters a local minimum.

2.3 Probabilistic Path Planning Methods

For high-dimensional path planning problems, it is computationally too expensive to calculate an explicit representation of the configuration space. Probabilistic path planning techniques have achieved substantial attention throughout the last decade as they are capable of solving high-dimensional problems in acceptable execution times. As no explicit representation of $\mathcal{C}$ exists, probabilistic methods invoke a binary collision checker to test whether a specific configuration is feasible. For a given configuration q, the collision checker simply verifies whether $\mathcal{A}(q)$ collides with an obstacle in $\mathcal{W}$. The two methods that attracted most attention during the last years are Probabilistic Roadmaps and Rapidly-exploring Random Trees. Both are probabilistically complete.

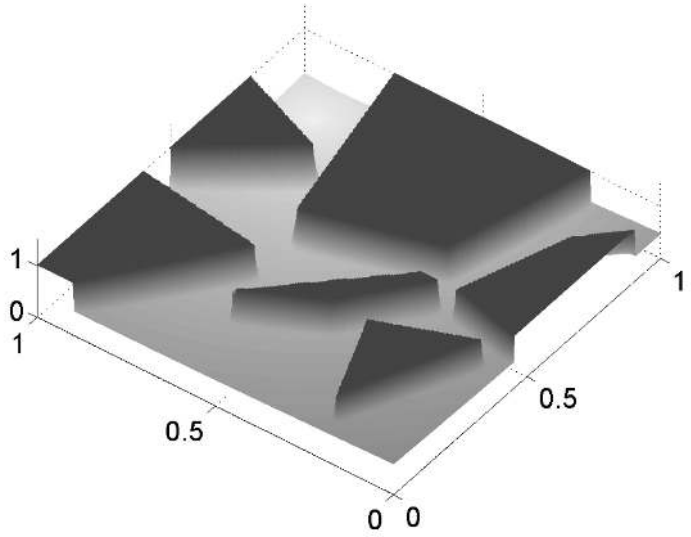


Figure 2.6: Potential field for path planning, attractor at goal state (0.9, 0.9) and repelling from obstacles

2.3.1 Probabilistic Roadmaps

Probabilistic Roadmap Methods (PRM) (Kavraki *et al.* 1996, Amato *et al.* 1998 *b*, Dale and Amato 2001) usually distinguish between a learning phase and a query phase. In the learning phase the roadmap is built by randomly sampling the configuration space. Those samples that correspond to collision-free configurations form the vertices of the roadmap. Neighboring vertices are then connected by edges if a local planner is able to connect the corresponding two configurations by a continuous path in $\mathcal{C}_{\text{free}}$. The most common local planner simply checks a straight line path connecting these configurations. In the query phase, the initial and the goal state are connected to the roadmap, which is then searched for a path connecting these two states. Figure 2.7 shows a simple 2D example of the learning and the query phase.

If the environment is static, the roadmap can be reused for further queries. Therefore, most PRM methods are classified as multiple query methods. Bohlin and Kavraki (2000) introduced a single query variant called Lazy PRM. In this approach, the roadmap validation is postponed. The roadmap is built not in the collision-free configuration space $\mathcal{C}_{\text{free}}$, but in the whole configuration space $\mathcal{C}$. First after a path has been found in the query phase, this path is checked whether

it is feasible or not. Thereby, the number of collision checks needed is reduced drastically, making Lazy PRM favourable especially if collision checking is very costly. If no path could be found, the roadmap has to be extended.

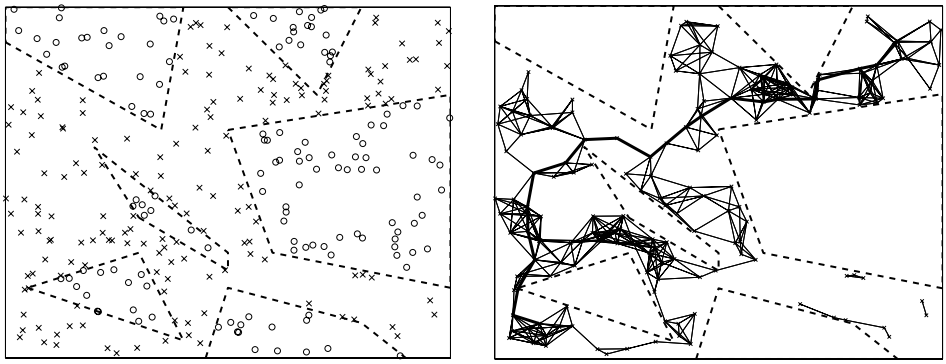


Figure 2.7: Left Figure: Learning phase – the configuration space is sampled; collision-free samples x, colliding samples o; Right Figure: Still in the learning phase, neighboring collision-free samples are connected by a straight line path to form the roadmap, colliding samples are discarded; query phase – the initial and goal configuration are connected to the roadmap, which is then searched for the shortest path connecting these two configurations.

Difficulties arise for PRM methods when the solution path has to pass through a narrow passage. The probability that a random sample falls inside this passage is very low. Thus, the number of samples that have to be drawn gets prohibitively large. Obviously, the connectivity of the roadmap has to reflect the connectivity of the free configuration space $\mathcal{C}_{\text{free}}$ to be able to solve path planning problems. Thus, current research focusses on improving the connectivity of the roadmap while at the same time keeping down the number of nodes.

To cope with this problem, several variations and extensions have been proposed. Morales *et al.* (2003) propose to add a connected component connection step after the roadmap is constructed in the learning phase. The main focus is not on guiding new samples to difficult areas, but trying to connect nodes in different connected components. Hsu *et al.* (1998) try to find narrow passages by dilating the free space. They first allow for a small penetration of the agent into an obstacle to construct the roadmap. Then, the colliding nodes are retracted to $\mathcal{C}_{\text{free}}$ by local re-sampling.

Another important aspect is that the number of nodes in the roadmap should not grow too large. Many samples in large open regions of $\mathcal{C}_{\text{free}}$ typically do not enhance the connectivity of the roadmap and are therefore almost useless. If the

connectivity of the roadmap is not sufficient, uniform random sampling will mainly produce more samples in large open regions while only rarely finding a valuable sample in a difficult part of $\mathcal{C}_{\text{free}}$. Siméon *et al.* (2000) propose visibility based roadmaps, where a collision-free sample is added as a node to the roadmap only if it is not visible by any other node or if it is visible to at least two nodes from different connected components. In this context, two nodes are visible for each other if a local planner is able to connect them by a feasible path, e.g., the straight line path is collision-free.

In (Boor *et al.* 1999) a Gaussian sampling strategy is introduced to keep only free samples that are close to configuration space obstacles. A maximum distance is chosen according to a Gaussian probability distribution. In a related approach, Aarno *et al.* (2004) bias the probabilistic sampling using a potential field. As the potential field is generated in the workspace, it does not suffer from the possibly high dimensionality of $\mathcal{C}$. Each sample in $\mathcal{C}$ is not only checked for collision, but for its correspondence in $\mathcal{W}$, the workspace potential field is integrated over some control points. In this way, a configuration space potential is approximated which is then used to decide whether or not a sample is important enough to keep. Hsu *et al.* (2003) classify a sample as important only if it passes the *bridge test*. The bridge test stands for finding colliding samples to both sides of a collision-free sample in a random direction at a short distance. Thus, the line segment connecting the two colliding configurations can be seen as a bridge from one $\mathcal{C}$-obstacle to another hovering over the collision-free sample. Consequentially, mainly samples in narrow regions of $\mathcal{C}_{\text{free}}$ remain. Kurniawati and Hsu (2004) try to identify narrow passages in $\mathcal{C}$ based on a tetrahedralization of $\mathcal{W}$ and sample these regions to find collision-free samples.

2.3.2 Rapidly-Exploring Random Trees

Another approach to probabilistic planning are Rapidly-exploring Random Trees (RRTs)(LaValle 1998). In the basic RRT algorithm a tree is grown from the initial configuration to explore $\mathcal{C}_{\text{free}}$. In each step a random sample in $\mathcal{C}$ is taken. Starting from the nearest vertex in the tree a new edge pointing at the sample is added. This nearest vertex expansions implicitly adds a Voronoi bias to the configuration space exploration. As the vertices on the boundary of a tree have the largest Voronoi regions, they will frequently be chosen for expansion. Thus, large free areas are explored rapidly by RRT methods. Figure 2.8 shows a 2D example for a bi-directional RRT method. Two trees grow from the initial and goal configuration to explore the free configuration space $\mathcal{C}_{\text{free}}$. When both trees can connect to the same collision-free configuration, a feasible path is found.

One of the most efficient variations of this algorithm is RRTConCon (LaValle and Kuffner 2000). Here, one tree is grown from the initial configuration and another from the desired goal configuration. In every step both trees try to connect

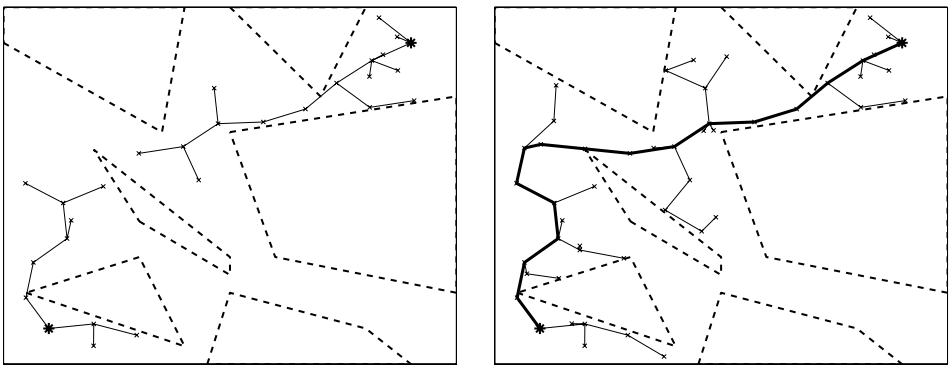


Figure 2.8: Left Figure: trees grow from the initial and goal configuration to explore $\mathcal{C}_{\text{free}}$; Right Figure: after some more iterations, both trees can connect to the same sample – a feasible path is found

to the same state in $\mathcal{C}_{\text{free}}$. Thus, a path is often found very quickly at the cost of diverging a lot from an optimal path. This can be corrected by a subsequent smoothing step. The major advantage of RRT-like methods is that they are perfectly suitable for nonholonomic and kinodynamic planning problems (LaValle and Kuffner 2001). For these types of problems, PRM and the proposed PCD method face the difficulty that connecting two states by an edge raises a possibly nontrivial control problem.

Also RRT planners experience difficulties if narrow passages are crucial to the connectivity of the configuration space. However, the problems faced by RRTs differ slightly from those for PRM. For RRT planners the main difficulty is to find an entrance to the passage. Once it contains a sample inside the passage, it can grow incrementally through it. But, on the other hand, the incremental process makes the problem of finding an entrance a lot harder. If a collision-free sample inside the passage is found, which can not be connected to the existing trees, it is thrown away. Strandberg (2004 *a*) proposes to keep these valuable samples and let them spawn so called local trees which will potentially connect to the global start or goal tree later on.

Related concepts that also explore the configuration space by growing a tree are Expansive Space Trees (Hsu *et al.* 2002) and the Ariadne's Clew algorithm (Mazer *et al.* 1998).

2.3.3 Roadmaps of Trees

In an attempt to combine PRM and RRT, Akinc *et al.* (2003) propose a powerful multi-level path planner. Here, RRT serves as a local planner to a PRM-like planner that keeps track of the global planning problem. The high level planner divides the problem into subtasks which are than passed to the local – low level – planner for solving. As the relatively large subproblems can be assigned to different processors, the method achieves high parallel performance. Impressive results where obtained on a number of challenging path planning problems.

2.3.4 De-randomizing Probabilistic Path Planning Methods

Noticing the success and performance of probabilistic path planning methods, a question that naturally arises is: To what extent is the performance of these methods due to the randomness involved?

In the case of PRM, LaValle *et al.* (2004) showed that the original probabilistic approach does not provide any advantages over deterministic sampling. Both, quasi-random sampling with Hammersley or Halton sequences and lattice based roadmaps are beneficial in terms of discrepancy and dispersion. These benefits are due to the fact that standard uniform samples tend to accumulate in some areas, while other areas are sparsely covered.

Lindemann and LaValle (2004) presented RRT-like planners that are based on deterministic sampling. In this context the choice of deterministic sequence that provides the samples is very important to maintain the Voronoi bias. Halton points were used in this case as well.

Chapter 3

Probabilistic Cell Decomposition

In this chapter, we present a novel approach to probabilistic path planning called Probabilistic Cell Decomposition. It combines the concept of probabilistic sampling with the underlying method of cell decomposition.

3.1 Introduction

As stated in Section 2.2.1, the idea behind cell decomposition methods is to partition $\mathcal{C}$ into disjoint cells. Unfortunately, the deterministic methods rely heavily on an explicit representation of the configuration space. Though, for high-dimensional problems, the computation of this representation is too expensive to be practical. Cell decomposition methods can be classified as *exact* and *approximate*. PCD is based on the latter, where all cells have a predefined shape. The most commonly used decomposition technique is the 2^m-tree decomposition, where m is the dimension of the configuration space. A cell that is not entirely contained in $\mathcal{C}_{\text{free}}$ or the complement of $\mathcal{C}_{\text{free}}$ in $\mathcal{C}$, $\mathcal{C}_{\text{obst}}$, is called *mixed* and is split up into 2^m subcells. This subdivision technique is impractical for high-dimensional problems. A mixed cell in a, for example, twenty-dimensional configuration space had to be split up into more than one million cells.

As no explicit representation of the configuration space is available, it is never known, whether a cell is entirely free or entirely contained in $\mathcal{C}_{\text{obst}}$. To handle this problem in PCD, a cell is assumed to be free until disproval. A cell is called *possibly free*, as long as all collision checks of samples in this cell have negative outcome. Accordingly it is called *possibly occupied* if all checks are positive. If both collision-free and colliding samples occur in the same cell, it is named *known to be mixed* and has to be split up into possibly free and possibly occupied cells. This implies unfortunately that a cell path in G does no longer automatically deduce an existing

```
WHILE ( !success )
    IF ( path <- findCellPath(G) )
        IF ( checkPath(path) )
            success <- true
        ELSE
            splitMixedCells
    ELSE
        q <- randomState(pOccCells)
        IF ( !collision(q) )
            splitMixedCells
```

Figure 3.1: The basic algorithm of PCD

path in $\mathcal{C}_{\text{free}}$, but the states along a path through this channel have to be checked for collision.

3.2 Notation

Let us introduce the basic notation. A cell κ_i might be

- possibly free, i.e. $P(\kappa_i \subset \mathcal{C}_{\text{free}}) > 0$
- possibly occupied, i.e. $P(\kappa_i \subset \mathcal{C}_{\text{obst}}) > 0$
- known to be mixed, i.e. $\kappa_i \not\subset \mathcal{C}_{\text{free}} \wedge \kappa_i \not\subset \mathcal{C}_{\text{obst}}$

Let κ_{init} and κ_{goal} denote the possibly free cells containing the initial and the goal configuration, q_{init} and q_{goal}, respectively. Note that throughout the decomposition process these labels might be passed to different cells. The nodes of the connectivity graph G are the possibly free cells. Two nodes are connected by an edge if and only if the corresponding cells are adjacent. The set of cells corresponding to the connected component of G containing κ_{init} is called the start region $\mathcal{R}_{\text{init}}$, accordingly for the goal region $\mathcal{R}_{\text{goal}}$. A path in G connecting κ_{init} and κ_{goal} is called a *channel* or cell path.

3.3 The Basic Algorithm

The basic algorithm of PCD is given in Figure 3.1. The algorithm is initialized with $\kappa_{\text{init}} = \kappa_{\text{goal}} = \mathcal{C}$. Thus, initially the whole configuration space is covered by

one cell. As the only two samples inside this cell, q_{init} and q_{goal}, are collision-free, the cell is labelled as possibly free.

In every iteration, G is searched for a channel connecting κ_{init} with κ_{goal}. Whenever such a channel exists, states along a path through these cells connecting q_{init} with q_{goal} are checked for collision. If no collision occurs, a feasible path has been found. Otherwise, a cell that was marked as possibly free contains free and colliding states. Thus, this cell has to be marked as known to be mixed and split up into possibly free and possibly occupied cells.

If no such channel exists in G, check random states in the possibly occupied cells for collision. If the test is negative, a cell that was marked as possibly occupied contains free and colliding states and has to be relabelled and split up. In this manner, the possibly occupied cells are refined until a path in G has been found. All states that are checked for collision, either in the sampling step or while path checking, are stored as samples in the respective cell. In the following, the important parts of the algorithm are described in some more detail.

3.3.1 Cell Shape

We choose to decompose the configuration space into rectangloid cells as the advantages of this shape clearly outrange existing drawbacks. Trivially, any rectangloid is convex. Furthermore, it is defined by only $2n$ values (one upper and the opposing lower vertex), which is beneficial with respect to both memory consumption and execution time. In contrast, an arbitrary n-dimensional polytope with the same number of vertices as the corresponding rectangloid is defined by $n \times 2^n$ values. Starting with rectangloid cells aligned to the coordinate axes, the cells obtained by splitting a cell orthogonal to any coordinate axis are trivially also rectangloid. The check, whether two cells are adjacent, is simplified considerably as it is divided up into two easily verifiable parts. First checking if one of the coordinates of the upper vertex of one cell is equal to the respective coordinate of the lower vertex of the other cell and - if so - check if the measure of the shared area on the (n-1)-dimensional hyperplane orthogonal to this coordinate axis is larger than zero.

An obvious drawback of the rectangloid cell shape is that even geometrically simple $\mathcal{C}$-obstacles can hardly ever be made up by a finite number of cells. Additionally, the resulting path planner is not independent of the internal alignment of the problem. This second argument will vanish if the obstacles in $\mathcal{C}$ are mainly not oriented in a common direction.

3.3.2 Graph Search

The possibly free cells form the vertices in a non-directed, possibly not connected graph, the connectivity graph G. Two adjacent cells are connected by an edge. Adjacency in this context is defined as indicated above. The (n-1)-dimensional

measure of the shared area on the boundary between two cells has to be strictly positive. This allows for multiple edges connecting the same two vertices depending on the topology of the configuration space. The cost assigned to an edge is the distance between the two centers of the cells according to some metric. Figure 3.2 shows a cell decomposition and the corresponding connectivity graph G. In each iteration, A*-search (Luger and Stubblefield 1989) is used to find an optimal channel ϕ connecting κ_{init} with κ_{goal}.

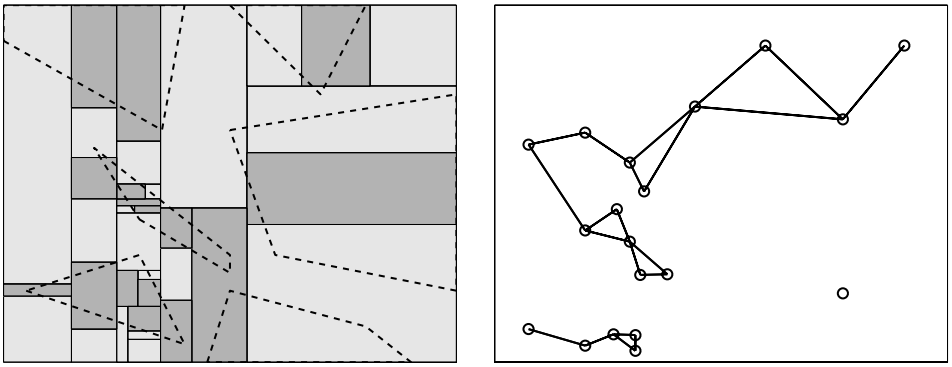


Figure 3.2: Left Figure: Cell decomposition; Right Figure: Corresponding connectivity graph G with three connected components.

The main idea of A* is to employ an estimate of the goal distance – given by the heuristic h – for each node in the graph to reduce the number of nodes that have to be evaluated. It maintains a priority queue of already visited nodes, sorted by the sum of the cost along the shortest known path from the start cell to the node (g) and its goal distance estimate (h).

In each iteration of A*, the node with the lowest key value in the queue gets *expanded*. All of its neighbors that have not been visited yet get placed in the priority queue. Those that have been visited before are checked whether a shorter path to the start node has been found. In this case, their position in the priority queue gets updated.

To ensure that A* finds the optimal path, h has to be a underestimator of the true goal distance, it has to fulfill $h(\kappa_i) \leq h^*(\kappa_i)$ for all i, where h^* is the actual cost along the shortest path in the graph connecting κ_i with κ_{goal}. The heuristic used in PCD is $h(\kappa_i) = M(\kappa_i, \kappa_{\text{goal}})$, where M is the Euclidian metric, which obviously satisfies this condition. Another heuristic $\bar{h}$ is called to be *more informed* than h if $\bar{h}(\kappa_i) > h(\kappa_i)$ for all i. When using the least informed heuristic $h(\kappa_i) = 0$ for all i, A* coincides with the ordinary breadth first search.

Initially, the start cell gets expanded. An optimal path from κ_{init} to κ_{goal} is found when the goal cell is visited.

3.3.3 Local Path Planning

Whenever a channel $\phi = \{\kappa_{\text{init}} = \kappa_{p_0}, \kappa_{p_1}, \ldots, \kappa_{p_n} = \kappa_{\text{goal}}\}$ connecting κ_{init} with κ_{goal} in G has been found in the graph search step, a local path planner has to find a path in $\mathcal{C}_{\text{free}}$ connecting q_{init} with q_{goal}. To maintain the iterative structure of the algorithm, this continuous path may not traverse possibly occupied cells. Otherwise, when checking this path for feasibility, a collision might be found in a possibly occupied cell. Then, the path is not feasible, but the new information about the colliding sample will not lead to a cell split as the cell was already labelled as possibly occupied. Thus, in the next iteration, the graph search will return the same channel and the local path planner will check the same continuous path through these cells. The iterative algorithm gets stuck.

As can be seen in Figure 3.3, connecting the centers of adjacent cells by a straight line path is ruled out according to these restrictions. The direct connection of two centers of adjacent possibly free cells might traverse a possibly occupied cell. Instead, the centers of the shared boundary between two successive cells will be connected by straight line paths. In an n-dimensional configuration space, two adjacent cells are separated by an (n-1)-dimensional hyperplane. As the cells are convex, the path connecting the centers of shared boundaries by a straight line is guaranteed to lie completely inside the channel of possibly free cells. Thus, to find a path through a possibly free cell κ_{p_i}, the local path planner simply tries to connect the center of the shared area on the hyperplane separating $\kappa_{p_{i-1}}$ and κ_{p_i} with the center of the shared area on the hyperplane separating κ_{p_i} and $\kappa_{p_{i+1}}$ by a straight line. Similarly q_{init} and q_{goal} are connected to the center of the shared boundary to the next cell on ϕ.

This path in $\mathcal{C}$ is then checked for collisions to determine whether it is entirely lying in $\mathcal{C}_{\text{free}}$ or not. For the moment, this check is done at a number of r discrete points along the path. Observe that for any finite step size between these points and for arbitrary robot shapes and obstacles this can not guarantee that the continuous path is collision-free as well. Here adaptive step size methods using distance measures (Schwarzer *et al.* 2002) could give certainty at the cost of higher computational effort.

Discrete point collision checking has been studied in the context of PRM (Amato *et al.* 1998 *a*), (Geraerts and Overmars 2002). Mainly two techniques, incremental and binary checking, have drawn major attention. The incremental method checks a path by successively checking configurations at a given step size along the path. In contrast, the binary method checks the configuration midway on the path and then recursively uses this technique to check the first and the second half until a given step size has been reached. In general the binary checking detects a collision

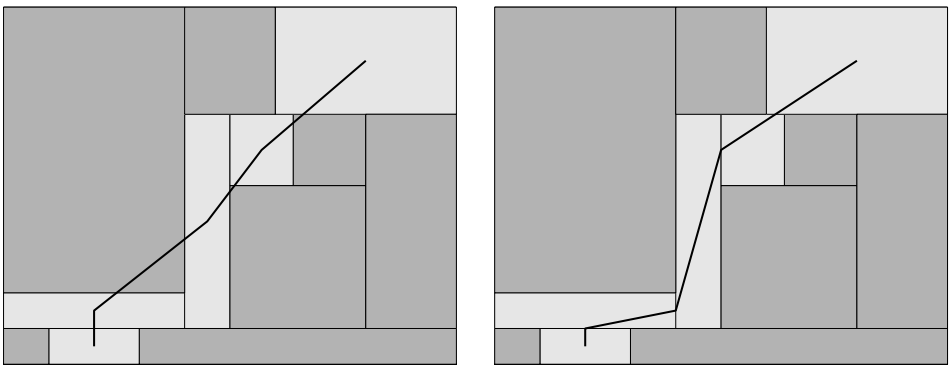


Figure 3.3: Left Figure: Connecting the centers of adjacent cells might lead to a path not lying entirely in possibly free cells.; Right Figure: Connecting the centers of the shared boundary of adjacent cells in a channel yields a path entirely contained in the channel.

faster as it is more likely that colliding states lie in the middle of the path than close to the collision-free ends. Therefore, binary checking is used within PCD.

3.3.4 Cell Splitting

Whenever a collision-free state is found within a possibly colliding cell or a colliding state is found within a possibly free cell, this cell has to be marked as known to be mixed and split up into possibly free and possibly colliding cells.

The requirement for rectangloid cells restricts us to cuts along the coordinate axes. A cell that is known to be mixed might not be divided into one possibly free and one possibly colliding cell only, but further splitting could be necessary. An n-dimensional rectangloid cell that is known to be mixed containing m collision-free samples and one colliding sample (accordingly vice versa) can be split up into at most $\min(2n, m)$ free cells and one colliding cell.

The strategy used to determine where to cut a cell was finding the nearest existing sample and then cut orthogonal to the dimension of the largest distance right in the middle between those two samples. Figure 3.4 shows an example where a collision-free sample was found in a possibly occupied cell containing several colliding samples.

3.3.5 Probabilistic Sampling

Whenever a channel is not found, the possibly occupied cells are sampled in order to refine them and make them better adopt the obstacles in $\mathcal{C}$. In the basic algorithm

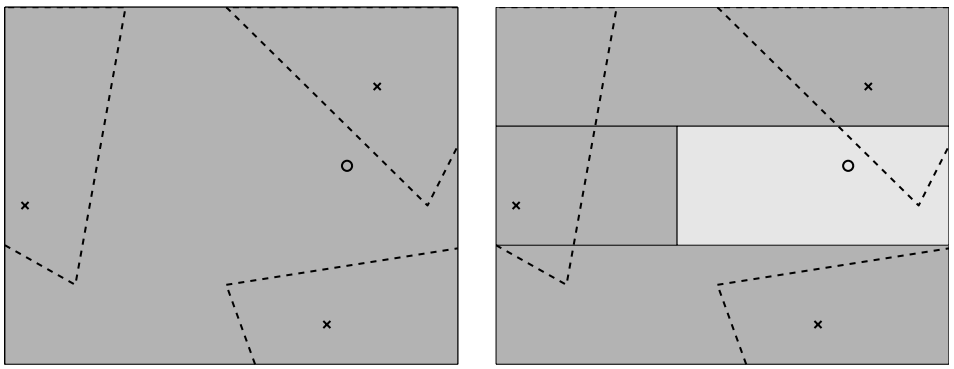


Figure 3.4: Left Figure: Collision-free sample (o) found in possibly occupied cell containing several colliding samples (x). Cell is known to be mixed and has to be split up into possibly free and possibly occupied cells. Right Figure: First split with respect to the closest colliding sample (upper right), then wrt. lower right sample, finally wrt. left sample

one sample is drawn in each possibly occupied cell. If this sample happens to be collision-free, the corresponding cell is known to be mixed and has to be split up into possibly occupied and possibly free cells.

3.4 On Probabilistic Completeness

In (Lingelbach 2004 *b*) it was stated that PCD is probabilistically complete and an outline of the proof was given. Unfortunately, the proof did not formally hold under the given assumptions. However, we are confident that the conjecture is correct in a sense that probabilistic completeness can be proven following those lines of reasoning.

3.5 Example

We will now apply PCD to the example problem frequently used in Chapter 2. Figures 3.5 and 3.6 show twelve snapshots of the solution process.

Figure 3.5.a Initially, the whole configuration space is covered by one possibly free cell that contains two collision-free samples, the initial and the goal configuration.

Figure 3.5.b Trivially, a series of possibly free cells is found that connects the cell containing the initial configuration with the cell containing the goal configu-

ration. The cell path consists of only one cell. Thus, the local planner tries to connect the initial with the goal configuration by a straight line path. The first three checked samples are collision-free, the fourth sample is colliding.

Figure 3.5.c A colliding sample has been found in a possibly free cell. Thus, this cell is no longer possibly *entirely* free but known to be mixed – it contains both, colliding and collision-free samples. Consequentially, it has to be split up into possibly free and possibly occupied cells.

Figure 3.5.d After the cell splitting step, one possibly occupied and two possibly free cells remain. The left possibly free cell is the new start cell κ_{init}, the right possibly free cell is the new goal cell κ_{goal}. As the possibly free cells are not adjacent, the connectivity graph G is not connected and κ_{init} and κ_{goal} are in different connected components of G. Thus, no cell path can be found connecting κ_{init} with κ_{goal}. As no cell path could be found, the possibly occupied cells have to be sampled until a collision-free sample is found. After finding this collision-free sample, the cell is no longer possibly *entirely* occupied but known to be mixed.

Figure 3.5.e The mixed cell is split up into a possibly occupied and a possibly free cell. Fortunately, incorporating the new possibly free cell, the connectivity graph G is connected and a cell path from κ_{init} to κ_{goal} can be found. The local planner tries to find a continuous path from q_{init} to q_{goal} through the cell path. Again it finds a collision, leading to a cell split. Notice the three collision-free samples found earlier in this cell.

Figure 3.5.f The cell is split up into two possibly free cells and one possibly occupied cell. The local planner tries to find a continuous path through the channel and, again, finds a collision.

Figures 3.6.a, 3.6.b, 3.6.c, 3.6.d, 3.6.e After some more iterations, it can be clearly seen, that the cell decomposition adapts to the obstacles.

Figure 3.6.f Finally, the local planner successfully connects q_{init} with q_{goal}.

3.6 Simulation Results

In this section, experimental results are presented for various kinds of problems followed by an interpretation of these results. The PCD planner presented in this chapter is tested on maze like problems, rigid body motion planning and path planning problems for articulated robots. The set of benchmark problems is presented in Appendix A.

3.6.1 Experimental Setup

The computation times were measured on a 1700 MHz machine with 512 MB of physical RAM. The algorithm was implemented and tested in the Motion Strategy Library (MSL) (LaValle 2000). Since the method is based on probabilistic sampling, the results are not directly repeatable. Therefore, the data we present here is based on 200 independent runs for each problem. For the execution time we present the values after which 25%, 50% (median) and 75% of the runs returned a feasible path. These values are less susceptible to outliers than, for example, the mean. Furthermore, the median number of collision checks and the median number of possibly free and possibly occupied cells used is given. As can be seen from the basic algorithm given in Figure 3.1, the execution time can mainly be divided into time spent on graph search, local path planning including path verification and probabilistic sampling. Splitting of mixed cells is accounted for within path verification and probabilistic sampling depending on which action led to the cell split. We present the median times spent on these three parts, respectively.

3.6.2 Summary of the Results

The results are shown in Table 3.1. The two 2D problems are solved almost instantaneously. The multi rigid body problem needed more than half an hour execution time in more than 50 % of the runs. Interestingly, the apportioning of computation time to graph search, local path planning and probabilistic sampling varies very much for the different problems. This can be explained by the structure of the problems. The configuration space of the multi rigid body problem, for example, is forty-eight-dimensional. Thus, a cell has very many neighboring cells and, consequentially, the connectivity graph is highly connected. In a highly connected graph, a cell path from the start to the goal cell is very likely to be found. As the sampling step is only processed if no cell path was found, the time spent on probabilistic sampling is negligible compared to the overall execution time. For the two 2D problems, the average number of neighboring cells is much lower and thus, after a local path planning step, the probability that G is no longer connected is much higher.

Another observation that can be made is the difference, whether there are more possibly free or possibly occupied cells. If the majority of cell splits occurs during local path planning, there will be more possibly occupied cells. This is due to the fact that every time a possibly free cell is split, one possibly occupied cell and at least one – but most often only exactly one – new possibly free cells occur. Else, when cells are split after probabilistic sampling, more possibly free cells arise.

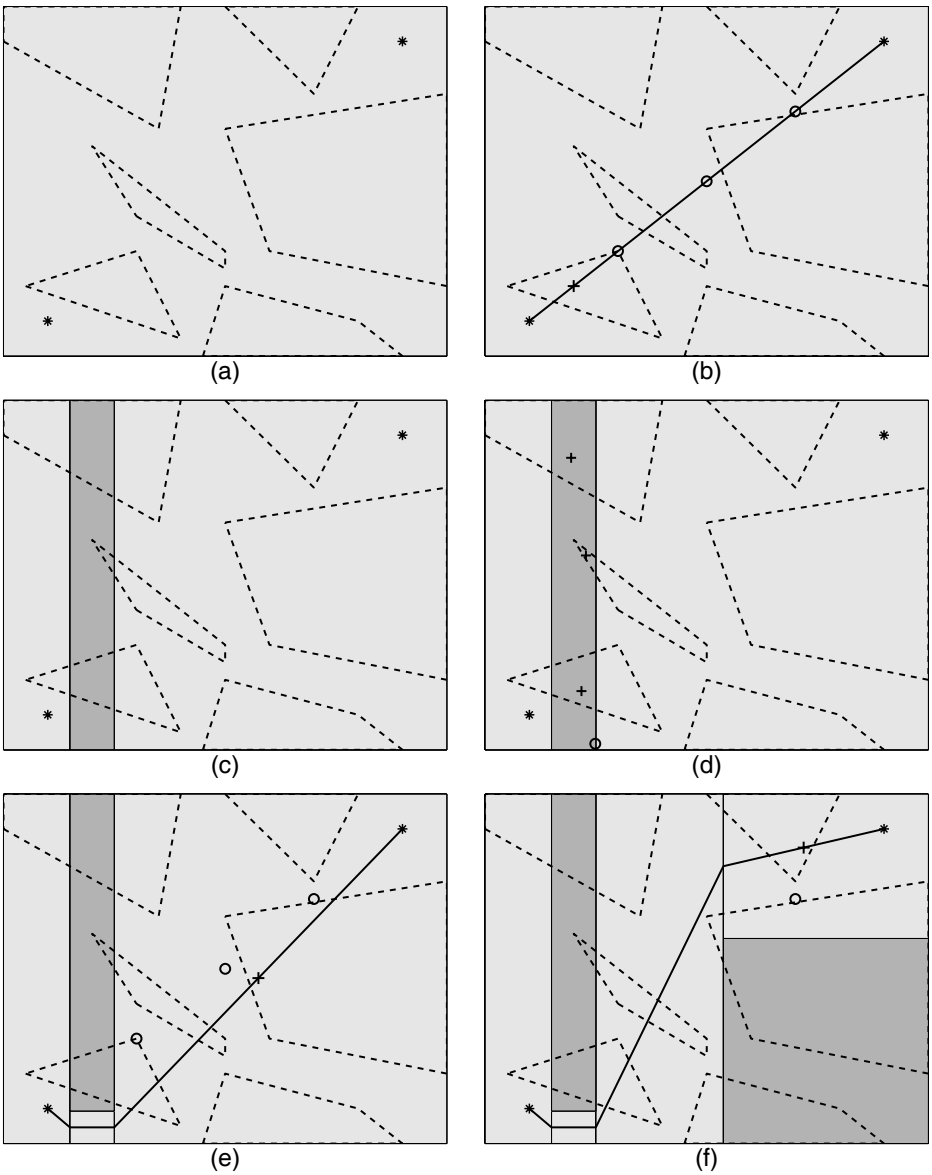


Figure 3.5: These figures show the progress of PCD on the example problem. A detailed explanation of all instances is given in Section 3.5. Possibly free cells: light gray; possibly occupied cells: dark gray; path checked by local planner: black line; colliding samples: $+$; collision-free samples: o; q_{init}, q_{goal}: $*$; obstacles: - -

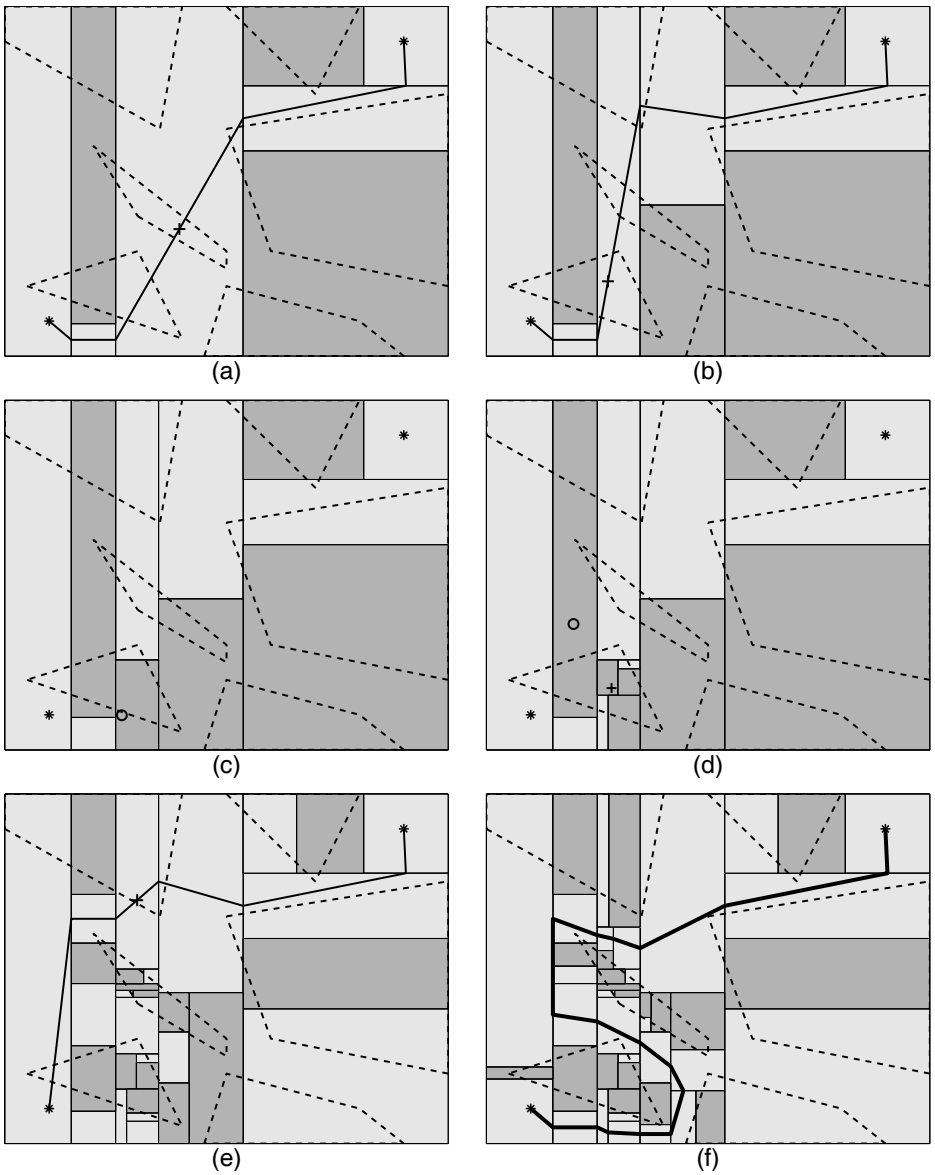


Figure 3.6: These figures show the progress of PCD on the example problem. A detailed explanation of all instances is given in Section 3.5. Possibly free cells: light gray; possibly occupied cells: dark gray; path checked by local planner: black line; colliding samples: $+$; collision-free samples: o; q_{init}, q_{goal}: $*$; obstacles: - -

	Problem I	Problem II	Problem III	Problem IV	Problem V
	maze	corridor	cage	mobmanip	multi
$\dim(\mathcal{C})$	2	2	6	9	48
t_{25}	0.12	0.09	0.45	0.08	447.6
t_{50}	0.15	0.12	1.16	0.16	1915.1
t_{75}	0.19	0.20	3.36	0.54	4072.3
Checks	3091	3212	4273	238	19821
$\#\kappa_{\text{free}}$	167	124	598	16	1432
$\#\kappa_{\text{occ}}$	86	66	497	23	2916
t_{gs50}	0.02	0.01	0.78	0.01	1472.7
t_{lpp50}	0.08	0.06	0.22	0.13	433.2
t_{ps50}	0.05	0.06	0.16	0.03	7.3

Table 3.1: Experimental Results; Execution time, number of collision checks and number of cells; median time spent on graph search, local path planning and probabilistic sampling, respectively

Chapter 4

Modifications of the Elementary Components of PCD

In this chapter, we investigate the elementary components of PCD further and propose possible modifications and improvements. For each component, the performance of the different alternatives is then compared using the set of benchmark problems presented in Appendix A. The components we will investigate are

- Probabilistic sampling
- Graph search
- Cell splitting
- Cell shape.

4.1 Probabilistic Sampling

Whenever a channel is not found, the possibly occupied cells are sampled in order to refine them and make them better adapt to the obstacles in $\mathcal{C}$. Different ways of distributing the samples are possible. An evident drawback of the sampling scheme used in the basic algorithm is that a big possibly occupied cell might block a narrow passage between the start and the goal configuration. If samples are taken one per cell, the probability of finding this narrow passage is very low.

4.1.1 Uniform Sampling over Possibly Occupied Volume

Instead, samples could be drawn uniformly distributed over the accumulated volume of all possibly occupied cells. Using this method, the accumulated volume of all possibly occupied cells is sampled until a collision-free sample is found. Thus, the

cell where the sample was found has to be split up into possibly free and possibly occupied cells.

4.1.2 Sampling Interesting Cells

Another point worthy of consideration is if really all possibly occupied cells have to be sampled. The sampling step is due each time there was no path in G connecting the start with the goal cell. Thus, the main task for the sampling step is to connect the two components in G containing the start and the goal cell. The idea here is two sample only *interesting* cells. In (Verwer 1990) an uncertain cell (in our case a possibly occupied cell) is called interesting if it is adjacent to $\mathcal{R}_{\text{init}}$ or $\mathcal{R}_{\text{goal}}$. As it turned out in the experiments, for the chosen problems almost all possibly occupied cells are adjacent to at least one of these regions. Tightening the requirements, a possibly occupied cell is called interesting if it is adjacent to $\mathcal{R}_{\text{init}}$ *and* $\mathcal{R}_{\text{goal}}$. To maintain probabilistic completeness (see Section 3.4), with a given probability of $p > 0$ a sample is drawn in every possibly occupied cell.

Figure 4.1 shows a 2D example where the circle in the lower left corner has to be connected to the circle in the upper right corner. Those cells marked dark gray in the left illustration will be omitted by this sampling scheme as they are not separating the goal from the start region.

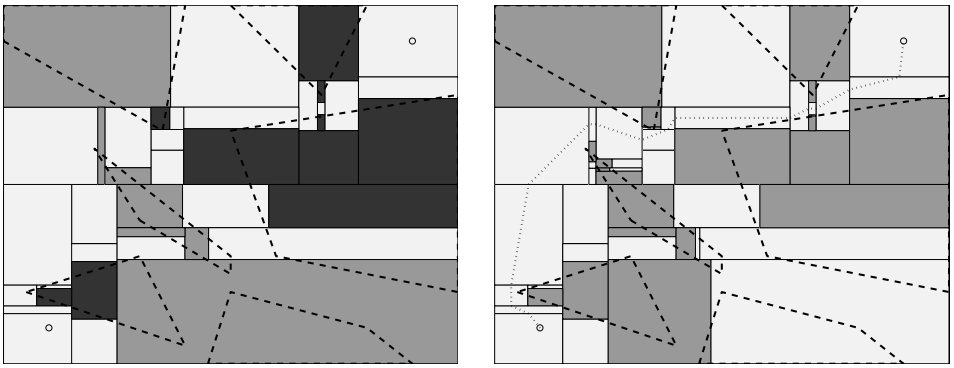


Figure 4.1: Left Figure: Cell decomposition after some iterations; Right Figure: Path found after a few more iterations; obstacles: $(--)$, possibly free cells: white; interesting / not interesting possibly occupied cells according to 4.1.2: light gray / dark gray; path $(\cdots)$

	Problem I maze	Problem II corridor	Problem III cage	Problem IV mobmanip	Problem V multi
$\dim(\mathcal{C})$	2	2	6	9	48
Sampling over the accumulated volume					
t_{25}	0.10	0.64	1.15	0.13	189.23
t_{50}	0.14	1.46	4.83	0.24	789.34
t_{75}	0.17	6.20	11.18	0.72	2347.83
Checks	2437	13212	11983	272	16163
$\#\kappa_{\text{free}}$	108	462	920	18	1203
$\#\kappa_{\text{occ}}$	68	703	1234	26	2680
Sampling interesting cells.					
t_{25}	0.09	0.09	0.28	0.06	165.76
t_{50}	0.12	0.12	0.78	0.12	624.86
t_{75}	0.15	0.17	1.59	0.39	1674.03
Checks	2388	2369	3313	238	12074
$\#\kappa_{\text{free}}$	106	78	279	16	1095
$\#\kappa_{\text{occ}}$	65	49	415	23	2407

Table 4.1: Experimental results for modifications on probabilistic sampling; Execution time, number of collision checks and number of cells for various methods and problems

4.1.3 Experimental Results

Table 4.1 shows the results for the different sampling schemes. Sampling *interesting* cells gave clearly the best performance for all benchmark problems. Sampling over the accumulated volume does not lead to any efficiency improvements. For the narrow corridor problem the performance got much worse. This can be explained by the fact that, while trying to find a path through the narrow corridor, many small cells are created in the vicinity of the narrow passage. Thus, sampling in interesting cells and also drawing one sample per cell will lead to many samples in the region of the narrow corridor. The main advantage of sampling in interesting cells is the fact that splitting an interesting cell will most likely connect the start with the goal region, whereas the other two sampling schemes also sample in regions of $\mathcal{C}$ that are not of major importance for the solution process.

4.2 Graph search

The examples from the preceding chapter indicate that up to 75% of the overall computation time is spent on repeatedly finding a channel connecting κ_{init} with κ_{goal}. Thus, the efficiency of PCD can be improved considerably by improving the graph search part.

4.2.1 The Connectivity Graph

The possibly free cells form the nodes of the non-directed connectivity graph G. In the following we will denote both a cell and its corresponding node in G by κ_i. Two nodes κ_i, κ_j are connected by an edge $e_{i,j}$ if the corresponding cells are adjacent. The cost assigned to an edge $e_{i,j}$ is $\mathrm{c}(e_{i,j}) = M(\kappa_i, \kappa_j)$, where M is a metric reflecting the distance between the centers of κ_i and κ_j.

The structure of G depends very much on the particular planning problem and the topology of $\mathcal{C}$. Let $n_{\mathrm{neighb}}(i)$ denote the number of possibly free cells adjacent to κ_i and n_{neighb} the average number of possibly free neighbors in the graph.

Splitting of possibly occupied cells

Possibly occupied cells only get split in the sampling step, that is, if no channel was found connecting κ_{init} with κ_{goal}. Thus, it is known that prior to incorporating the new cells, G is disconnected and κ_{init} and κ_{goal} belong to two different connected components, $\mathcal{R}_{\mathrm{init}} \neq \mathcal{R}_{\mathrm{goal}}$. Furthermore, it is clear that no nodes are deleted from G but only new nodes are added for each newly arisen possibly free cell. Depending on the sampling strategy used, such a recent possibly free cell might be further split up within the same sampling step. This has not to be taken care of as a *splitting of possibly free cells* as described later, since G is first evaluated after the complete sampling step.

Splitting of possibly free cells

Free cells get split when a collision was detected while checking a path through a channel of possibly free cells. Here, a possibly free cell is split up into one possibly occupied and up to $2n$ possibly free cells, where n is the dimension of the configuration space. Thus, one node is deleted and between one and $2n$ nodes are added to G for each split.

4.2.2 Optimal Graph Search

A*-search

In the basic algorithm presented in Chapter 3 in each iteration A*-search (Luger and Stubblefield 1989) was used to find a path in G.

Between two iterations, only a few cells get split and G does not change very much. By restarting A*-search in each iteration, a lot of information gathered in the previous graph search is discarded that might be used in the following run. Thus, iterative methods like *lifelong planning A** (LPA*) (Koenig and Likhachev 2002) are expected to give improvements compared with repeated A*.

Incremental A*-search

In their paper Koenig and Likhachev (2002) present performance improvements of factor five to ten of LPA* compared to repeated A*. Unfortunately, the setup for these experiments does not fit the graph search problem occurring in PCD. While they used a regular eight-connected grid world, where often exist many parallel paths of the same length, in PCD G is irregular and ties in path length occur only accidentally. Another difference in the setup is that in (Koenig and Likhachev 2002) edge costs changed for randomly chosen edges in the graph. Thus, often the optimal path was not effected at all. In PCD the graph often changes due to collisions in a previously found channel. Thus nodes along the optimal path are deleted.

In PCD, two different actions will effect G and thus call for a re-computation of the optimal path. If no path was found in the previous graph search, $\mathcal{R}_{\text{init}}$ got completely expanded and the shortest path from every node in $\mathcal{R}_{\text{init}}$ to κ_{init} is known. This knowledge can be taken over to the current graph search step. The new nodes get placed on the priority queue, if and only if they are adjacent to the prior $\mathcal{R}_{\text{init}}$. By expanding these nodes an optimal path will be found if it exists. Otherwise, $\mathcal{R}_{\text{init}}$ will again get completely expanded, which will be important for the following sampling step.

When possibly free cells get split after a collision has been found, nodes have to be deleted from G in addition to the incorporation of new nodes. Since only cells in a found channel are checked, the corresponding nodes in the graph lie on the shortest path connecting κ_{init} with κ_{goal}. Thus, at least the start distances of all cells on the shortest path subsequent to a split cell have to be updated. Depending on the structure of G, this *shadow* can be very wide, containing a large number of cells whose start distance changed. Obviously, the number of cells to be updated is larger, if the split cell was close to the start cell.

Incremental A* with occasional restart

As stated above, the structure of G depends strongly on the particular planning problem. Especially the informedness of the heuristic h can vary over a wide range. Since our goal is to present a general path planner, choosing a suitable heuristic for a particular problem is not an option. Instead, we propose an adaptive method to decide, if starting from scratch might be advantageous over incremental search.

If the changes in G are due to splitting of a possibly occupied cell in the sampling step, new cells have to be incorporated into the graph. No increased start distances have to be propagated through the graph and thus, the number of nodes expanded by LPA* is not greater than that of A*. Hence, LPA* is used for graph search after a sampling step.

When possibly free cells get split and deleted from G, the start distances of all nodes have to be updated whose shortest path to the start cell passed one of these

deleted nodes. The complexity of the propagation of increased start distances is quadratic in the number of neighbors n_{neighb}, which itself increases with the dimension of $\mathcal{C}$. Thus, if a node far away from the goal node gets deleted and the increased start distances have to be propagated over many cells, the performance of incremental A* decreases rapidly.

Since a deleted node κ_i lied on a previously found optimal path, the shortest goal distance in the graph is known and we can define the *relative informedness* of the heuristic h at this node as

$$\hat{h}_i = \frac{h_i}{h_i^*} = \frac{h_i}{g_{\text{goal}} - g_i}$$

which is simply the estimated goal distance of node κ_i using the heuristic h divided by the shortest path from κ_i to κ_{goal} found in the graph.

The relative informedness can be seen as a measure of how useful the heuristic h is, to guide the graph search towards the goal. For a node κ_i, the optimal value of 1 is obtained only if the path from κ_i to κ_{goal} is a straight line. Then after expanding this node, no other nodes will be expanded than those lying on the shortest path. In general, the relative informedness is higher for nodes closer to the goal.

When the relative informedness of a node to be deleted drops below a certain threshold, restarting the A*-search might be advantageous over propagating the changed information through the whole graph. Unfortunately, $\hat{h}$ is defined using information that is revoked by deleting the respective node. In general, the changes made in the graph are not too extensive and $\hat{h}_i$ still gives a good measure for the usefulness of h in this region of the graph.

4.2.3 Suboptimal Graph Search

As the graph search algorithm is used very frequently, the question arises whether it is necessary to search for an *optimal* channel every time. It might be advantageous to trade optimality for speed and use a fast but suboptimal graph search algorithm like a less informed best-first search (BFS) that utilizes only goal distance to guide its search. Typically, a close-to-optimal path is found much faster than using A*. Though, in a worst case, BFS might take more time to obtain a longer path than A* would need to find the optimal one. This, however, will only happen exceptionally and the more significant drawback is that the longer path obtained by BFS is in general more likely to be in collision with obstacles than a shorter one.

Especially for probabilistic methods, changes in one part of the algorithm might have significant side-effects which, in a worst case, let the overall performance decrease. Using Incremental A* will not show any of those, since it will give exactly the same result – the optimal channel with respect to some heuristic. The channel found by BFS will in general not coincide with the one obtained by A*.

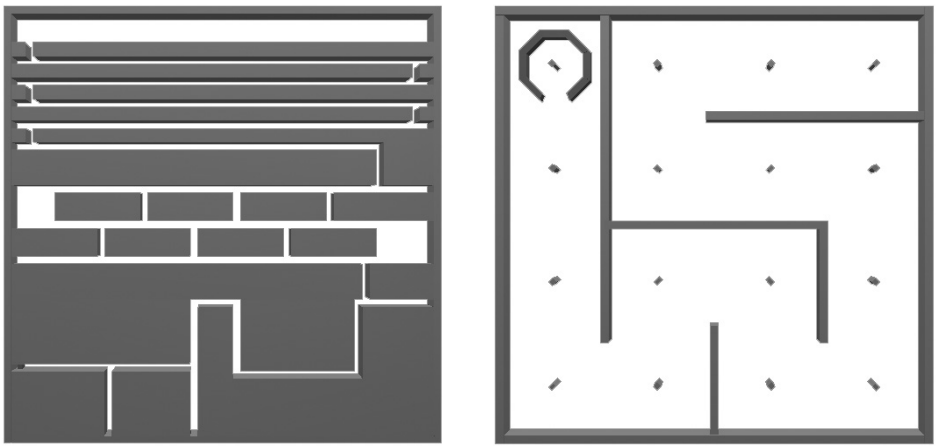


Figure 4.2: Left Figure: The point-like agent has to traverse the long narrow corridors from the lower left to the upper right corner; Right Figure: The C-shaped agent has to reach the upper right corner of the maze.

4.2.4 Experimental results

As the time spent on graph search is negligible for the two 2D problems and the mobile manipulation benchmark, we test the different graph search methods on two other benchmark problems instead. Figure 4.2 shows the two problems taken from MSL, namely `2dpoint2` and `2dmaze4c`, referred to as (i) and (ii), respectively. In the first problem, the point-like agent has to traverse the long narrow corridor from the lower left to the upper right corner. The agent is not allowed to rotate. Thus, the configuration space is two-dimensional. In the second problem, the C-shaped agent has to reach the upper right corner. The pillars in the middle of the corridor force the agent to rotate around them. The agent is allowed to translate and rotate in the plane, giving rise to a three-dimensional configuration space.

We use "sampling in interesting cells" for all tests. As A*, incremental A* and incremental A* with occasional restart return exactly the same path, we test them in parallel. Thus the number of cells, the number of collision checks needed and the path length is the same for all three methods. The results of the tests can be seen in Table 4.2. The median execution time and the median time spent on graph search are given for all four methods and the four problems. Additionally, the path length is given for the optimal graph search methods and for best-first search.

The performance of incremental A* depends very much on the structure of G. For the first two problems G is winded, following the shape of the maze. Thus, the heuristic used by the optimal graph search methods does not provide valuable

	Problem i	Problem ii	Problem III	Problem V
	long corridor	C-maze	cage	multi
$\dim(\mathcal{C})$	2	3	6	48
A*-search				
t_{50}	24.41	26.07	0.78	624.86
$t_{\text{gs}50}$	3.29	12.52	0.47	458.54
path length	751.2	1393.3	322.2	953.7
Incremental A*-search				
t_{50}	24.63	34.87	0.66	453.19
$t_{\text{gs}50}$	3.56	21.34	0.34	287.73
Incremental A*-search with occasional restart				
t_{50}	22.68	24.74	0.68	460.56
$t_{\text{gs}50}$	1.90	11.31	0.37	294.79
Best-first search				
t_{50}	26.55	21.52	0.52	713.28
$t_{\text{gs}50}$	2.41	6.43	0.18	112.81
path length	748.5	1528.3	349.7	1284.5

Table 4.2: Experimental results for modifications on graph search; Median execution time, median time spent on graph search and path length

information until the search has come close to the goal cell. Thus, for these problems, incremental A* is even slower than searching from scratch at each iteration. Using the switch to restart the incremental A*-search if the relative informedness is below a certain threshold, gives rise to small improvements compared to repeatedly running A* and discard the information gained in the previous iteration. For the rigid body problems, the heuristic provides enough information for incremental A* to perform better than repeated A*.

For all but the first problem, the path obtained by best-first search is about 10% to 30% longer of that obtained using an optimal graph search method. For the multi rigid body problem, it can be observed that the overall planning time increases while the fraction of time spent on graph search drops from 75% to 15%. This can, again, be explained with the structure of G for this problem. Here, the connectivity graph is highly connected, leading to a large number of parallel cell paths connecting the start cell with the goal cell. Thus A* has to expand a very large number of nodes, whereas best-first search directly heads for the goal cell. The drawback is a longer path returned by best-first search. As a longer path in general corresponds to a higher probability of collision, the time gained on graph search is lost in local path planning and a higher number of iterations needed to solve the path planning problem.

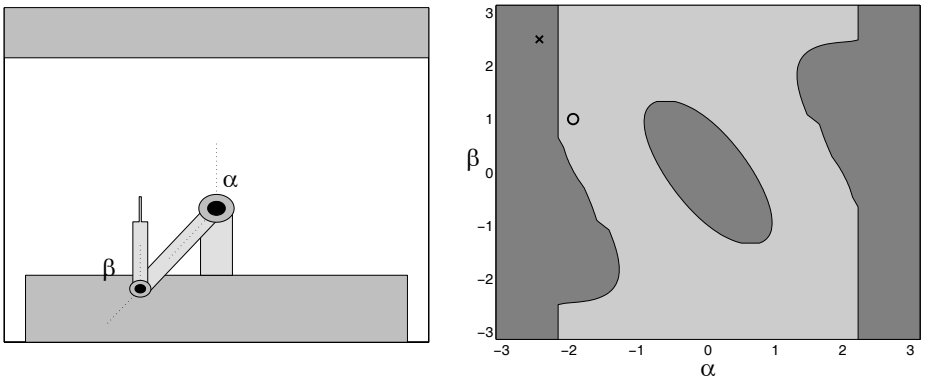


Figure 4.3: Left Figure: Articulated 2 dof Robot colliding with the podium; Right Figure: Configuration space $\mathcal{C}$, collision-free (light gray) and colliding (dark gray) states, collision-free sample (o), colliding sample (x) shown in $\mathcal{W}$

4.3 Cell Splitting

The cell splitting step can be adapted if there is some information available about $\mathcal{C}$-obstacles. This is most evident for multi rigid body problems or path planning for an articulated robot. If, in a multi rigid body problem, one body collides with an obstacle, all other configurations corresponding to the same state of this colliding body will neither be feasible. In particular, it does not help to change the state of any other body to get rid of this collision. In the same way, the structure of an articulated robot can be utilized throughout the cell splitting step. A mobile manipulator usually has a chain-like structure, where a manipulator with multiple links is mounted on a mobile base. It is intuitively clear that, if for some configuration the base is in collision with an obstacle, all other configurations with the same base state will also be colliding. More general, a collision of a link with the environment can be eliminated only by a movement of this link or one that is closer to the base. In the same way, a self-collision can only be dealt with by a relative motion of the two colliding links.

Figure 4.3 shows a 2D example. A 2 dof robot is mounted on a solid podium. The possible collisions are those of the two links with the podium or the ceiling. To the right in the same figure, the configuration space is shown. The area of colliding states around the origin corresponds to collisions of the outer link with the ceiling. For $\alpha < -2.2$ or $\alpha > 2.2$ the inner link collides with the podium. Thus, regardless of β, the state is colliding. Assume now the whole configuration space being only one possibly free cell. The colliding sample (x) has been found in the cell, the closest existing collision-free sample is marked (o). Thus the cell has to

	Problem IV mobmanip	Problem V multi
$\dim(\mathcal{C})$	9	48
t_{25}	0.05	9.11
t_{50}	0.06	21.38
t_{75}	0.08	61.13
Checks	155	4213
$\#\kappa_{\text{free}}$	6	288
$\#\kappa_{\text{occ}}$	4	857

Table 4.3: Experimental results for modifications on cell splitting; Execution time, number of collision checks and number of cells

be marked as known to be mixed and split into possibly free and possibly occupied cells. According to Section 3, the cell has to be split orthogonal to the dimension of the largest distance right in the middle between these two samples. This would lead to a horizontal split orthogonal to the β-axis. Obviously, a vertical split orthogonal to the α-axis would be much more convenient.

For cell splitting this leads to the following strategy: Find the nearest existing sample. Split the cell orthogonal to the dimension of the largest distance between these two samples, where only those dimensions are considered that might lead to an elimination of the collision. To be able to adapt the cell splitting, more information about the collision has to be available than just a binary answer. The planner needs to know which body actually was colliding. The results of tests applying this cell split adaptation can be seen in Table 4.3.

The number of cells needed to solve the problems is reduced significantly. As a consequence thereof, the execution time is decreased, too. For the mobile manipulation problem it is advantageous, that the main obstacles like the table and the chair are low enough. Thus, only the base can collide with the obstacles and this part of the planning is reduced to the 2D problem of finding a path for the base towards the shelf.

4.4 Cell Shape

The advantages of a rectangloid cell shape, mentioned in Section 3.3.1, seem to outweigh possible drawbacks. Thus, instead of proposing alternative cell shapes and testing their effect on the performance, we will investigate the influence of the alignment of rectangular obstacles with the split directions. Especially for Problems I and II, the obstacles are perfectly aligned with the coordinate axes.

In order to investigate the influence of the alignment on the planning performance, we rotate a slightly modified version of the two 2D benchmark problems

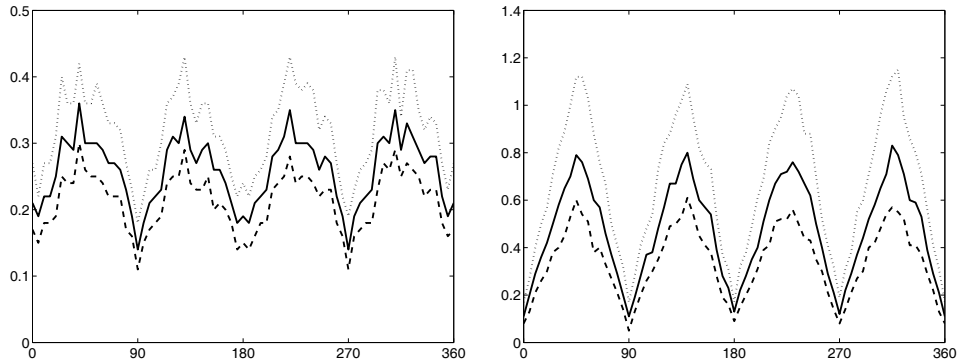


Figure 4.4: Left Figure: Rotated benchmark problem I; Right Figure: rotated benchmark problem II; computation time in seconds plotted vs. rotation angle in degrees: t_{25}: dashed; t_{50}: solid; t_{75}: dotted

at a resolution of 5 degrees. The modification comprises an inflation of the outer obstacle ring such that the interior can be rotated. This inflation causes the small differences in computation time for a rotation angle of zero compared to the results presented before. For each rotation angle, Figure 4.4 shows t_{25}, t_{50} and t_{75} for 200 runs. The sampling method used is sampling in interesting cells and A* is used for graph search. It can be clearly seen from Figure 4.4 that the planning time is minimal for a rotation angle of $\alpha = k\pi/2, \quad k = 0, \ldots, 4$. This is rather unsurprising as for these angles the $\mathcal{C}$-obstacle boundaries are perfectly aligned with the split directions.

Consequentially, if information about the main orientations of configuration space obstacles is available, the performance of PCD might be increased by a transformation of $\mathcal{C}$ such that the boundaries of configuration space obstacles get aligned with the coordinate axes. As could be seen in Section 4.3, parts of the configuration space obstacles of multi rigid body problems and articulated robots are naturally aligned with the coordinate axes.

4.5 Summary of the Results

Here, we will summarize the results of the preceding sections and propose the elementary components for a general PCD planner. It should be general in a sense that for a given problem class, no further information about the problem is given. We will then compare the performance of the general algorithm to that of RRTConCon, a very successful variant of Rapidly-exploring Random Trees.

Restricting the set of cells for sampling to those that are adjacent to the start region $\mathcal{R}_{\text{init}}$ *and* the goal region $\mathcal{R}_{\text{goal}}$ gave clearly the best results for the chosen benchmark problems. We assume that this holds true for most path planning problems. Therefore, we propose to use sampling in *interesting cells* in the general planner.

For the graph search part, the method of choice may depend on the application at hand. The results obtained for the different methods do not clearly favor any of them. Best-first search gave good performance at the cost of a longer path. The performance of the incremental A* search is very much dependent on the graph structure of the connectivity graph G of the problem. The heuristic switch deciding when to restart the incremental A* algorithm improved the results for the degenerated graph search problems. However, the quality of the threshold used may be problem dependent. As general path planners should contain as few parameters as possible, that have to be tuned to obtain a good performance, we propose to use A* even at the cost of slightly worse efficiency.

The cell split adaptation gave rise to major performance improvements, where applicable. If the problem class reveals information about C-obstacles, as for the multi rigid body problem and the mobile manipulation benchmark, one clearly benefits from using this information.

We compare the results obtained by the general PCD planner to those obtained by one of the most successful variants of Rapidly-exploring Random Trees, namely RRTConCon. This method was presented briefly in Section 2.3.2. An implementation of the algorithm is distributed with the Motion Strategy Library (LaValle 2000). Table 4.4 shows the execution time and the number of collision checks needed for both methods on the five benchmark problems. Additionally, we give the number of cells needed for PCD and the number of nodes in the trees for RRTConCon.

The results show that PCD performs well compared with RRTConCon. The most significant difference appears for the 2D narrow corridor problem. Here, RRTConCon faces the difficulty to incorporate a sample inside the corridor into one of the trees. Even if it finds a valuable sample inside the narrow passage, it tries to connect to it most probably through the nibs attached to both ends of the corridor. Thus, the sample can not be reached by one of the trees. PCD is also considerably faster, where the split adaptation is applicable. One could argue that PCD needs more information from every collision check to obtain the higher performance. This is true, but using recent collision checking packages as, for example, PQP, this information is available at no higher cost.

Compared to the results obtained by the basic algorithm, presented in Chapter 3, major improvements could be made for the three higher-dimensional problems. The results on the two 2D problems differ not very much.

	Problem I	Problem II	Problem III	Problem IV	Problem V
	maze	corridor	cage	mobmanip	multi
$\dim(\mathcal{C})$	2	2	6	9	48
PCD					
t_{25}	0.09	0.09	0.28	0.05	9.11
t_{50}	0.12	0.12	0.78	0.06	21.38
t_{75}	0.15	0.17	1.59	0.08	61.13
Checks	2388	2369	3313	155	4213
$\#\kappa_{\text{free}}$	106	78	279	6	288
$\#\kappa_{\text{occ}}$	65	49	415	4	857
RRTConCon					
t_{25}	0.38	582.93	1.09	1.36	66.58
t_{50}	0.67	1499.32	2.08	1.81	124.94
t_{75}	1.15	2431.66	3.57	2.42	226.3
Checks	3768	114900	23185	2945	75392
#Nodes	422	25764	550	79	3695

Table 4.4: Experimental Results; Execution time, number of collision checks and number of cells for a general PCD planner. For RRTConCon, the number of tree nodes is given instead of cells.

Chapter 5

Path Planning for Mobile Manipulation – an Application

In this chapter, we present an application of PCD from the field of autonomous mobile manipulation in a home or office environment. In this field, the ability to plan its own motions is essential for a robot to be truly autonomous. It has to react to its environment and carry out user-defined instructions. Nearly every task, the robot has to accomplish, involves a motion from one configuration to another. Each time, this raises the question of how to move each joint in a feasible manner, such that the robot reaches the goal state and avoids collisions with obstacles on its way. Planning performance is of particular importance as the user wants the robot to react on a given command without mulling over a feasible path for a while.

5.1 Framework

A typical task for an autonomous service robot could be: "Get the milk from the refrigerator!". The high level "get milk" task can be decomposed into a number of subtasks on a lower level like:

- open the refrigerator
- get the milk
- place the milk on the table
- close the refrigerator

Each of these tasks can then be divided into even finer grained tasks. The "open the refrigerator" task, for example, can be divided into:

- locate the refrigerator

- locate the door handle
- approach the door handle
- grasp the door handle
- move the door handle along a specific trajectory
- release the door handle

Executing such tasks requires, among other things, knowledge of different areas such as software and hardware architectures, low-level and real-time programming, modeling, automated control, fault-control, human-robot interaction, vision and perception, localization, mapping and path planning.

At the Centre for Autonomous Systems (CAS) an architecture for integrating research from different areas into a complete system is under development. The NoMan (**No**vel **Man**ipulation) architecture provides a simple to use architecture for testing ideas on real robots. One of the short term goals of NoMan is to combine relevant research from different areas into a system capable of navigating an office environment and performing the "get milk" task. The NoMan project uses a deliberate/reactive architecture. A simplified version of the NoMan architecture is shown in Figure 5.1. The architecture presented here excludes some parts that are not needed for the "get milk" task.

The top layer of the NoMan architecture is the specification layer where tasks are specified. A task specification can be as simple as a predefined task (for example, a surveillance robot) or it can be more complex, originating from a human-robot interaction (HRI) sequence (for example, a verbal speech command or a gesture), see (Christensen *et al.* 2001) and (Topp *et al.* 2004). From the specification layer a *task description* is propagated to the task planner. Such a task description could be `Fetch(MILK)`. Using the *knowledge base* the task planner then decomposes the task into sub-tasks. For instance the knowledge base knows that the milk is likely to be in the refrigerator and the refrigerator is in the kitchen. The task planner can then use the world model and the path planner to compose a necessary low-level task description for performing the `Goto(KITCHEN)` sub-task. This low-level task description is then fed to the task execution layer. From the low-level task description the task execution layer starts a set of reactive behaviors and monitors their progress. In the `Goto(KITCHEN)` example the reactive behavior would be the *path adaptation* behavior, which is composed of an obstacle avoiding and a path following behavior.

5.2 Path Planning for a Redundant Platform

The platform used in our experiments is composed of a mobile XR4000 base (3 dof), a PUMA 560 arm (6 dof) and a Barrett hand (4 dof). For a given grasp,

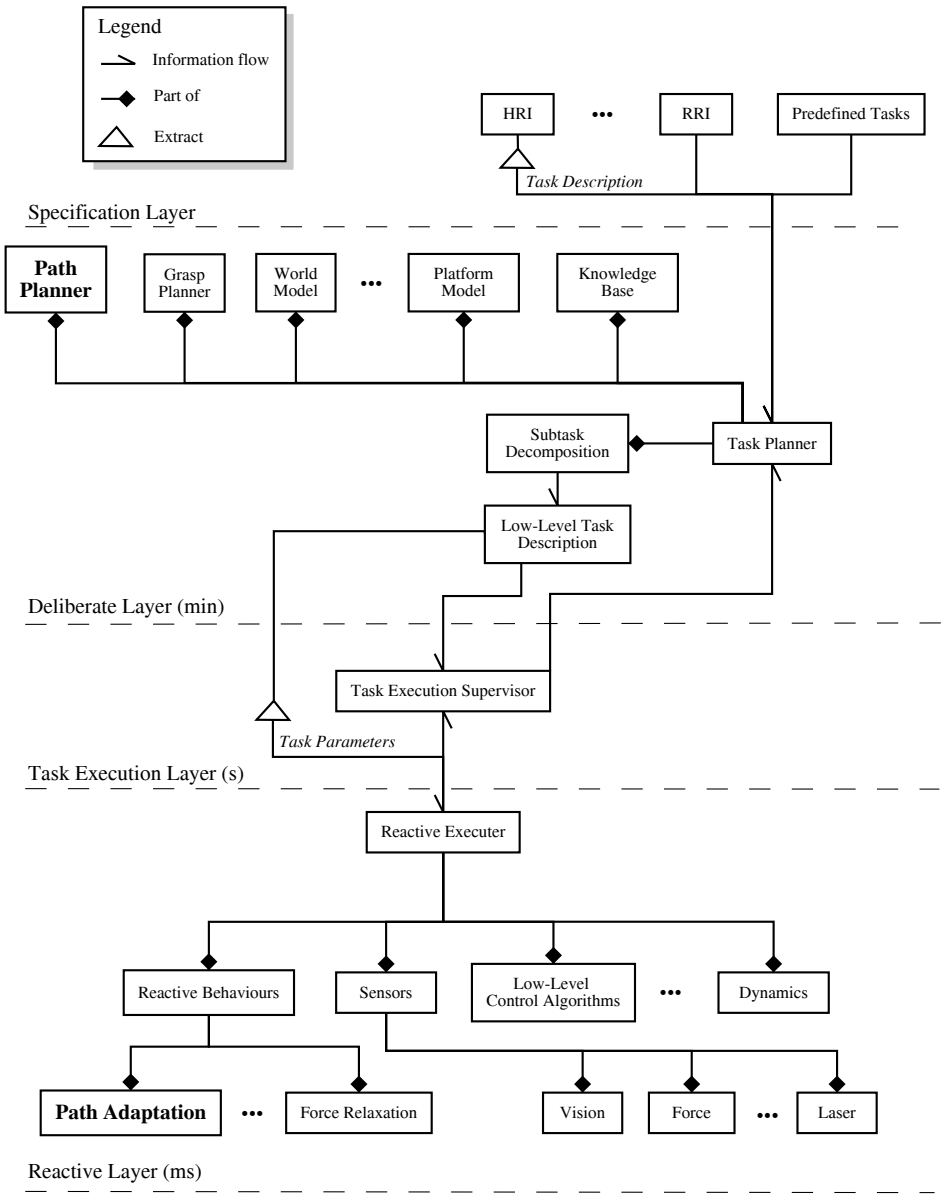


Figure 5.1: Overview of the architecture for the NoMan system.

the joints of the hand are fixed. Thus, only 9 dof remain. For some tasks that the robot has to accomplish, there may be additional constraints on a feasible path. A very common one is, for example, a given end-effector path. If the pose of the end-effector is given, the inverse kinematics of the platform can be used to calculate a feasible joint configuration. For a redundant platform, there exist infinitely many solutions to the inverse kinematics. These can be grouped into a finite number of self-motion manifolds (Burdick 1989). The joint configuration can change continuously along such a manifold without affecting the pose of the end-effector. We therefore propose to perform probabilistic path planning in the space $\mathcal{C}_p = s \times p_s$, where s is the progress along the end-effector path and p_s is a parameterization of the self-motion manifold for a given s.

Using the example of opening a refrigerator from Section 5.1, a given grasp, obtained from the grasp planner (Miller *et al.* 2003), fixes the position of the end-effector relative to the door handle. Thus, a given opening angle of the door determines the pose of the end-effector and consequentially restricts the feasible configurations of the platform. To open the door, the end-effector has to follow a path that is defined by the position of the door hinge, the distance between the door handle and the hinge and the chosen grasp. For the platform described above, there remain 3 dof after fixing 6 dof by the holonomic constraints of the given end-effector pose. The rotational axis of the base coincides with the axis of the first joint of the arm. Together with the fact that the shape of the base is rotational invariant, if the small gearbox on top of the base cylinder is neglected, we assume all configurations equivalent for which holds $x_{\text{rot}} - x_0 = \text{const}$ (where x_{rot} is the base rotation and x_0 is the state of the first joint) Consequentially, 2 dof remain for the self-motion manifold.

The problem can be parameterized in the following way:

- α is the door opening angle, i.e., the progress along the end-effector path
- ϕ, r give the position of the base relative to the end-effector in polar coordinates

We denote this 3-dimensional configuration space by $\mathcal{C}_p$. For each collision check, the configuration in $\mathcal{C}_p$ has to be mapped to the natural configuration space $\mathcal{C}_n$. For a given triple (α, ϕ, r), the position of the end-effector and the base are fixed. The inverse kinematics of the manipulator then maps the relative base – end-effector position to a feasible configuration of the manipulator. Thus, in addition to the binary collision check, there arises another requirement for a configuration $x \in \mathcal{C}_p$ to be feasible: the inverse kinematics must have a solution for x on the chosen manifold. For the PUMA 560 there exist eight self-motion manifolds characterized by elbow-up / elbow-down, right-arm / left-arm and flip / no-flip.

There may be problems that are not solvable without switching to another manifold. Thus, a path planning algorithm that maps a $\mathcal{C}_p$ configuration to only

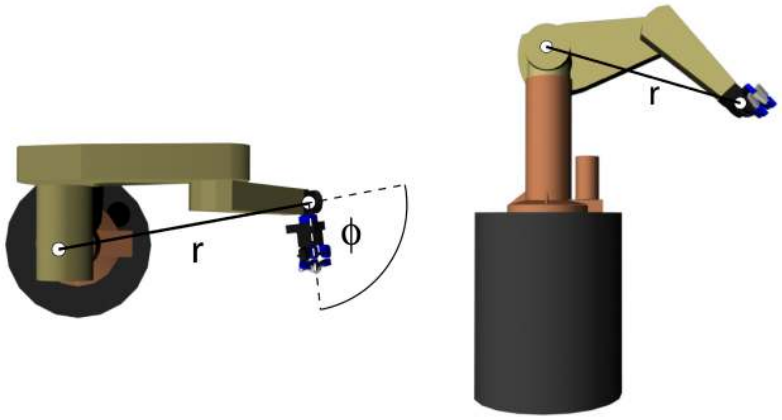


Figure 5.2: Top and side view of the platform with parameterization used for constrained path planning

one specific manifold can not find a feasible path for this kind of problems. This can be seen as a limitation of our approach. On the other hand, to switch to another manifold, the manipulator has to pass a singular configuration like an outstretched arm. However, in such a configuration the manipulator can not react to external forces in all directions. Thus, unexpected deviations from the path may be harmful to the platform. Therefore, it seems natural to avoid these singular configurations and, consequentially, remain on one manifold. The choice of manifold is up to the task planner that provides the path planner with a feasible start and goal configuration. Here, a possible approach could be starting with a standard manifold or checking random samples on the different manifolds for collision to obtain a rating of the manifolds.

5.3 Experimental Results

In order to test PCD on the "fetch milk" problem, we try to solve a series of subtasks, as possibly given from a task planner, using PCD. The task planner is not implemented yet, but the subgoals were chosen by hand. This ensures also that all subtasks are solvable. In a setting with an automated task planner, it had to be taken into account, that a subgoal might not be reachable from the given initial position. PCD – as many other probabilistic path planning algorithms – is not able to decide whether a problem is solvable or not. It simply runs until a feasible path is found. If no feasible path existed, it would in theory never stop but refine the cell decomposition. Thus, in a completely automated setting, each path planning

process had to be monitored and possibly terminated after some time concluding that no path could be found.

The following low level tasks involve a motion of the robot

(i) approach the refrigerator

(ii) open the refrigerator door

(iii) approach the milk

(iv) transfer the milk to the table

(v) approach the open refrigerator

(vi) close the refrigerator door

(vii) return to initial position

Here, opening and closing the refrigerator door implies motions with pre-defined end-effector paths as mentioned in Section 5.2. In this example, the milk carton has not been opened yet. Thus there are no additional constraints on the transfer path (iv). If the motions are sufficiently slow, an open milk carton could have been modelled using holonomic constraints stating that the carton has to be upright along the path. For faster motions, the dynamics of the milk in the carton would imply nonholonomic constraints.

Several snapshots from some of the subtasks are shown in Figure 5.3. In Figure 5.3.a, the robot is located in its the middle of the kitchen. This is the initial configuration for subtask (i) and the goal configuration for subtask (vii). Figure 5.3.b shows the robot opening the door. Observe that the robot maneuvers itself into a tight corner between the wall, the refrigerator door and the chair. This gives rise to a relatively narrow passage which the robot has to pass through when approaching the milk. In Figure 5.3.c, the robot stands in front of the refrigerator and grasps the milk carton. The next figure shows the robot placing the milk on the table. Then it returns to the refrigerator (Figure 5.3.e). Here, it has to maneuver around the chair and, again, pass through the narrow passage between the chair and the refrigerator door. The last figure shows the robot closing the door.

To compute the different paths, we use the Components for Path Planning (CoPP), a framework recently developed by Strandberg (2004 *b*). A* is used for graph search and the sampling scheme is sampling in interesting cells. For the unconstrained problems, the cell split adaptation presented in Section 4.3 is exploited.

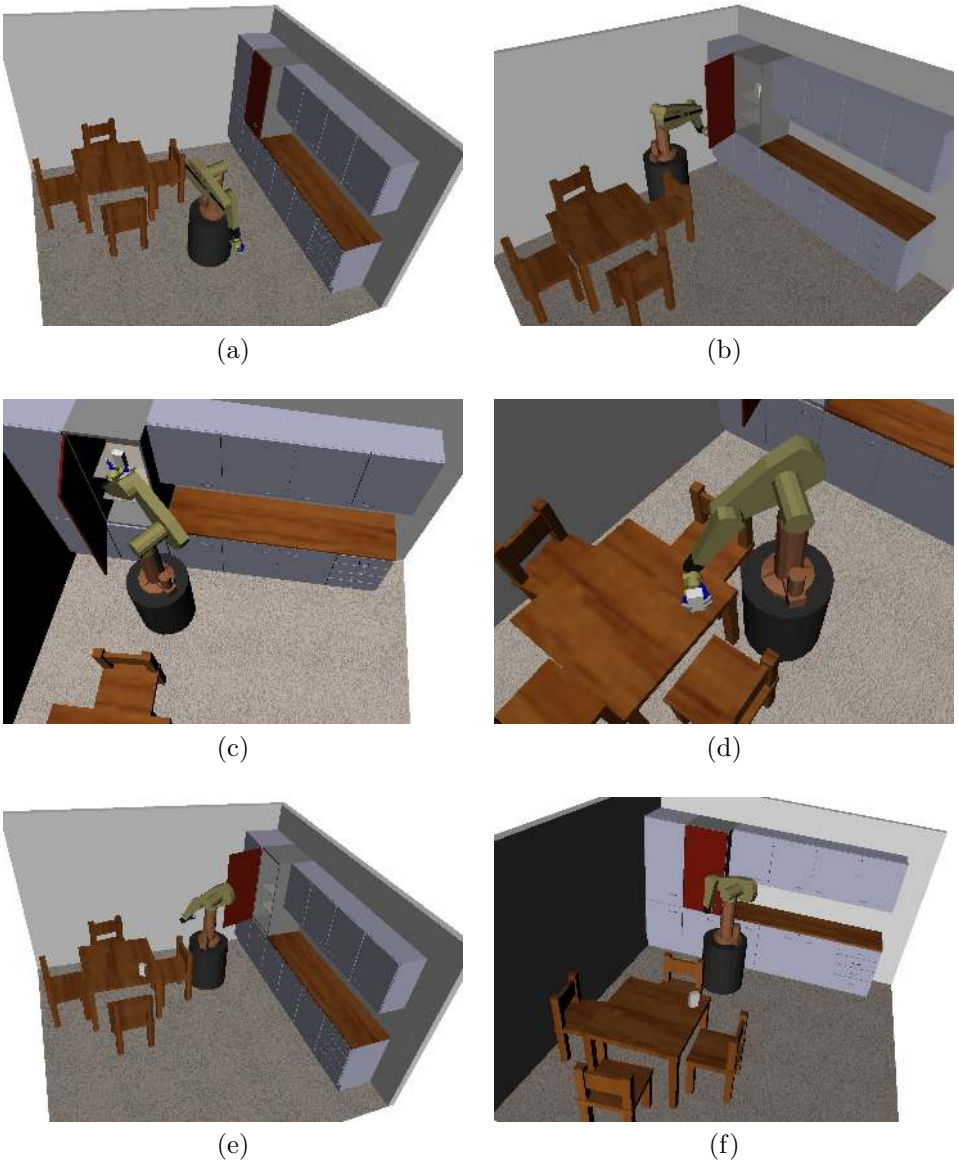
Table 5.1 shows the execution times for the different subtasks. It can be seen that planning the whole series takes between 0.7 and 2.5 seconds. That is far less than executing this series of motions would take at a speed suitable for a service robot in a home or office environment. Though, as we stated earlier in this chapter,

Table 5.1: This table shows the execution times for the different subtasks of the "fetch milk"-task.

subtask	(i)	(ii)	(iii)	(iv)	(v)	(vi)	(vii)
$\dim(\mathcal{C})$	9	3	9	9	9	3	9
t_{25}	0.05	0.08	0.23	0.10	0.18	0.09	0.05
t_{50}	0.06	0.12	0.43	0.13	0.41	0.13	0.07
t_{75}	0.07	0.20	0.91	0.25	0.83	0.20	0.08

the subgoals where carefully chosen by hand. If an automated task planner had been involved, planning of the whole sequence could have taken considerably longer because of subgoals that are very difficult to reach or even not reachable at all.

The most difficult subtasks are those where the robot has to move through the narrow passage formed by the chair and the open refrigerator door. Especially the door is here causing troubles, as it is most probably colliding with one of the outer links, such that the cell split adaptation can not provide any help.



(a) (b)

(c) (d)

(e) (f)

Figure 5.3: Figure (a): initial position; Figure (b): opening the refrigerator; Figure (c): grasping the milk carton; Figure (d): placing the milk on the table Figure (e): returning to refrigerator; Figure (f): closing the refrigerator

Chapter 6

Summary and Future Work

In this chapter we will summarize the thesis and give some directions for future research.

6.1 Summary

Chapter 3

In Chapter 3, we introduced the basic ideas of Probabilistic Cell Decomposition (PCD). It combines the concept of probabilistic sampling with the underlying method of cell decomposition. The elementary components of the algorithm are presented in detail. The performance of the algorithm is tested on the set of benchmark problems presented in Appendix A. These problems are chosen from different fields of application including maze-like problems, rigid body problems and path planning for a mobile manipulator. The problem dimension ranges from two to forty-eight.

Chapter 4

In Chapter 4 we proposed several modifications and improvements to the elementary components of PCD and tested their effect on the overall computation time.

Graph search In the basic algorithm, in each iteration A* is used on the connectivity graph G to compute an optimal channel connecting the start with the goal cell. As the graph changes only gradually between successive iterations incremental A* seems to be an option. Unfortunately, due to the structure of G and the type of changes between successive iterations, using an incremental A* variant led only to marginal improvements. For some problems, the performance even declined. A heuristic switch when to restart the incremental algorithm gave rise to an increased efficiency. Also non-optimal

best-first search has been tested. Lower computation times face the cost of longer solution paths. As none of the methods gave rise to considerable improvements and the best method – incremental A* with occasional restart – involves tuning a threshold parameter, we propose to use A* further on.

Probabilistic sampling We investigated several possible ways of distributing samples in the possibly occupied cells. Best results were obtained by sampling in *interesting* cells, where those cells are called *interesting* that divide the start region from the goal region.

Cell shape The choice of cell shape is crucial to the performance of the method. Changes in cell shape would directly effect all other components of the algorithm. Substantial arguments such as simplicity and memory consumption let us choose rectangloid cells. Therefore, we did not test different cell shapes but tested the effect of obstacle alignment with the split directions. We rotated the two 2D benchmark problems gradually and measured computation time and number of cells needed. Not surprisingly, best performance was achieved when the obstacles where perfectly aligned with the split directions.

Cell splitting For those problems where applicable, adjusting the directions of cell splits gave major improvements in computation time and the number of cells needed to solve the problem. For the mobile manipulation benchmark the articulated structure of the robot can be utilized to guide the splitting. The idea that can be exploited in this case is that a collision between a link and an obstacle can only be removed by a relative motion between this link and the obstacle. In the multi rigid body problem, similar considerations give insight that a collision between two free-flying rigid bodies calls for a split orthogonal to one of the dimensions affecting the pose of these two bodies.

We proposed a general PCD algorithm based on sampling in interesting cells, A*-search and, where applicable, split adaptation. The performance of this algorithm was then compared to one of the most successful variants of Rapidly-exploring Random Trees, namely RRTConCon. The experimental results showed significant improvements compared with the basic algorithm presented in Chapter 3. On the set of benchmark problems chosen, PCD was considerably faster than RRTConCon.

Chapter 5

In Chapter 5, we applied the modified PCD algorithm to a series of mobile manipulation tasks emanating from the given high level command "place the milk on the kitchen table". Here, we focussed on the path planning problem and neglected issues like manipulation planning, grasp planning and sensor based planning. For a specific subtask – opening and closing the refrigerator door – the path of the

end-effector was given. We discussed this subproblem briefly and proposed a parameterization of the constrained problem for the Obelix platform. Planning the whole series of subtasks takes about 1.5 seconds.

6.2 Future Work

Future work may consider both work on PCD as a general path planning method and modifications tailored to the field of mobile manipulation.

6.2.1 The PCD Algorithm

In Chapter 4, some modifications to the elementary components of PCD are discussed. This list is by no means conclusive, but many more modifications are conceivable. As each change in one component may have negative side effects on the overall performance, it is very hard to predict whether these modifications will lead to improvements.

Local Path Planning For a given cell path, the shortest path through this channel is obtained by solving a convex optimization problem. A shorter path corresponds in general to a lower risk of collision. It will be interesting to investigate, whether it is beneficial to spend the time on solving this optimization problem to obtain the shorter path.

Probabilistic Sampling So far, we looked at static features only when deciding which possibly occupied cell to choose for sampling. The superior sampling scheme was sampling in those cells that divide the start region from the goal region. It might be practical to look also at the "dynamics" and choose cells in the vicinity of the last cell split or along a previously found but discarded path.

Furthermore, it is worth investigating whether improvements can be achieved by not uniformly sampling a chosen possibly occupied cell. Here, one could think of sampling on or close to the boundary of the cell. The sampling area can be confined further by only taking those parts of the boundary into account, where the cell adjoins to a possibly free cell, leading to a higher possibility of finding a collision-free sample, or to a possibly occupied cell, may be leading to a reduction of splits nonrelevant to the solution progress.

Cell Splitting In Section 4.3, we discussed a possible cell split adaptation for problems where information about configuration space obstacles is available. This adaptation concerns the direction of the split and leads to great improvements where applicable. Another parameter to explore is the location of the split. In the current algorithm, the mixed cell is split in the middle between

the two closest samples of opposing type. It might be advantageous not to split exactly in the middle but shift the split location to one sample or the other. If the cell is split closer to the collision-free sample, the cell decomposition gets more conservative. If the cell is split closer to the colliding sample, the cell decomposition gets more optimistic. We assume that, by varying this location parameter, more load can be put on the graph search and sampling component or on local path planning and verification of these local paths.

It might also be beneficial to decide with respect not only to the type of sample, but to take also the type of the cell that gets split into account. The cell split location can be shifted towards the "wrong" sample, for example, a colliding sample found in a possibly free cell. This reduces the number of splits needed on average per mixed cell at the possible cost of worse adaptation to configuration space obstacles.

6.2.2 Planning for Mobile Manipulation using PCD

Planning under differential constraints So far, PCD is not particularly suitable for nonholonomic path planning. As stated in Section 2.1, differential constraints naturally arise from the dynamics of an agent. In contrast to the *Obelix* example, these differential constraints can not be neglected for many systems. Even if we look only at the field of robotics, there exist numerous examples of nonholonomic systems.

- **UAV** The dynamics of Unmanned Aerial Vehicles (UAV) obviously impose nonholonomic or kinodynamic constraints.
- **Robotic platforms** Most robotic platforms have a differential drive and support wheels limiting their motions to rotations and forward/backward driving. Car-like robots additionally have a maximum turning radius constraining their maneuverability.

The local planner used within PCD has to connect specific configurations by a feasible path. In presence of nonholonomic constraints, this poses a possibly nontrivial control problem for each local path. RRT methods avoid this problem due to their incremental growth. This idea might be applicable to PCD. Given a channel of possibly free cells, there is no need to connect pairs of specific configurations but the control problem might be easier to solve if it incrementally has to connect a specific configuration to a goal region - the shared boundary between the current and the next cell in the channel. Only one point to point connection remains to connect to the goal configuration.

Sensor based planning Sensor based planning includes both planning using sensor data and planning such that available sensors can be utilized in an optimal

way. The second point can probably taken care of by a subsequent optimization step. For example, an eye-in-hand camera should point on the target object or a 2D laser scanner could be swept orthogonal to the scanning plane to obtain 3D information.

Planning using sensor data is a challenging task, even if the world is assumed to be static. The inaccuracy of the sensors has to be taken into account. The world model obtained from sensor data has to reflect these uncertainties. Thus, a collision check might no longer be binary but return a probability that the given configuration is collision-free. It is not clear at first glance, how this can be incorporated into PCD. The concept of *probably* free and *probably* occupied cells might however provide handy opportunities.

Sensor data obtained while following the path have to be incorporated in the world model and the remaining path has to be validated and, if needed, modified.

For dynamic environments, the problem gets even harder. Are the new sensor data due to changes in the environment, uncertainties in sensor readings or even wrong sensor readings? What fraction of the old data is invalidated by the new data? Building a dynamic world model is not a task for a path planning algorithm but it has to cope with the problems arising from not knowing exactly, how the environment looks like or from changes in the world model. It is a moot point whether the straight forward way of reacting to an updated world model would work for PCD. First, affected saved configurations had to be updated and then, the cell decomposition had to be revised according to the changes. A cell might turn from possibly occupied to mixed or even to possibly free. In exceptional cases, it might be possible, to merge two cells of the same type, but in general the complexity of the cell decomposition will increase.

Appendix A

Benchmark Problems

In this appendix, we present the set of benchmark problems that was used frequently throughout this thesis. The problems are taken from different fields such as maze like problems, rigid body planning, planning for a mobile manipulator and multi rigid body planning.

A.1 2D Maze

Figure A.1 shows the agent in the start configuration. The goal configuration is located in the upper right corner of the maze. The configuration space for this problem is only two dimensional as the agent is only allowed to translate but not to rotate. For this low dimensional problem with polygonal obstacles and a polygonal agent, it is actually possible to easily calculate $\mathcal{C}_{\text{free}}$ and the optimal path. This problem is distributed with the Motion Strategy Library (LaValle 2000) as `2dpoint1`.

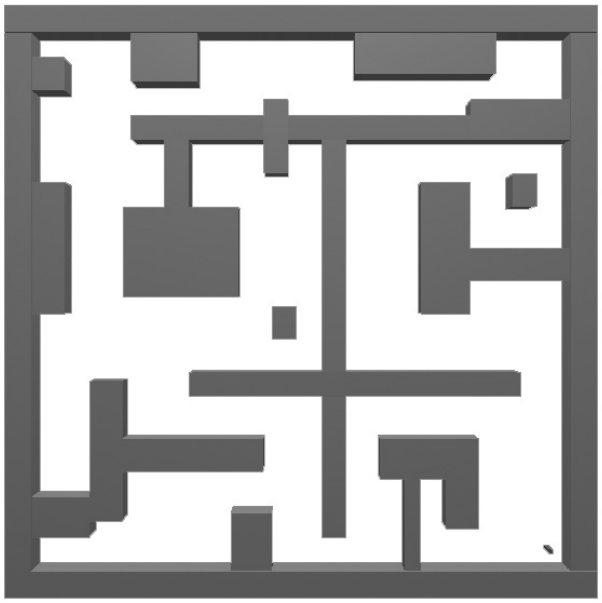


Figure A.1: Benchmark problem I (2D): The small quadratic agent has to traverse a maze.

A.2 2D Corridor

Here, the small quadratic agent has to pass through a corridor which is only 1.5-times wider than the agent itself. The nibs on both sides of the corridor make it very hard for RRT-like methods to find a path through the narrow passage. In the rare case that a collision-free sample is found inside the corridor, RRT will try to connect to this sample from the closest existing node in the tree. This will most likely be separated from the sample by a nib. Thus, the tree can not connect to the sample and the very valuable sample inside the corridor is discarded.

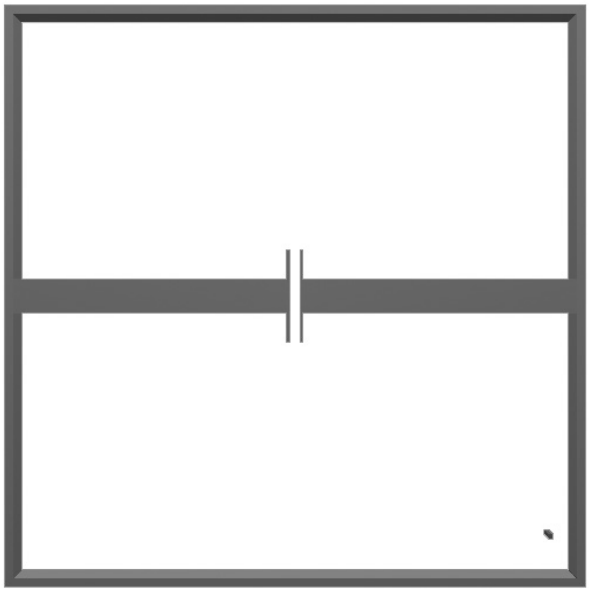


Figure A.2: Benchmark problem II (2D): The small quadratic agent has to pass through the narrow corridor.

A.3 6D Cage

In this problem, the L-shaped rigid body has to escape from the cage. It can translate within some bounds and rotate freely in the 3D workspace. Thus, its configuration space is $\mathcal{C} = [0, 1]^3 \times \mathrm{SO}(3)$. As can be seen from Figure A.3, there exist a lot of different paths the rigid body can take. This problem is taken from the Motion Strategy Library (`2dpoint1`).

Figure A.3: Benchmark problem III (6D): The eight L-shaped rigid body has to escape from the cage.

A.4 9D Pick'n'Place

In this mobile manipulation benchmark, the robot has to maneuver around the table and place the tool on the uppermost shelf. On its way to the shelf it has to pass through a somewhat narrow passage made up by the the chair and the table. The robot consists of a PUMA 560 arm mounted on a XR4000 base. The base can rotate and translate freely in the plane. The arm has six degrees of freedom. Thus, the configuration space of the whole mobile manipulator is nine-dimensional.

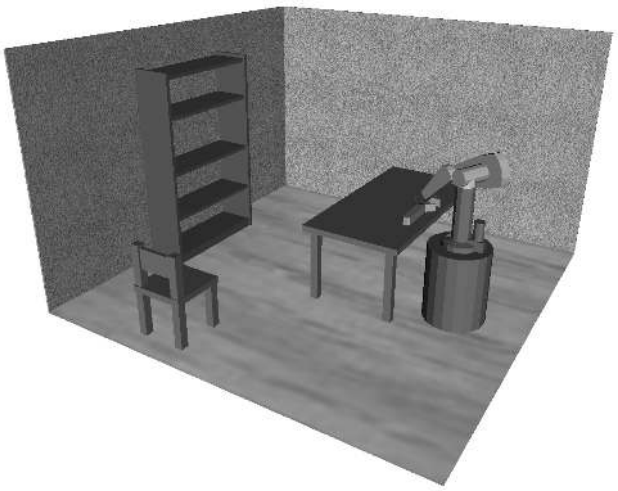


Figure A.4: Benchmark problem IV (9D): The mobile robot has to place the large tool on the uppermost shelf.

A.5 48D Multi Rods

This high dimensional problem was also taken from MSL (`multi3`). The eight rigid bodies have to exchange their positions without colliding with each other on their path. Each rigid body can move freely in space. Thus, the configuration space for this problem is composed of the eight individual configuration spaces: $\mathcal{C} = SE(3)^8$. The challenge of this problem is the very high dimensionality of $\mathcal{C}$. Even if the problem looks trivial to the human eye, it is very tough for a path planning algorithm. A problem that occurs especially for PCD is that due to the structure of the configuration space, a cell has very many neighboring cells. Thus, the connectivity graph $\mathcal{G}$ is heavily connected which slows down the graph search step.

Figure A.5: Benchmark problem V (48D): The eight rigid rods have to exchange their positions.

Bibliography

Aarno, Daniel, Danica Kragić and Henrik I. Christensen (2004). Artificial potential biased probabilistic roadmap method. In: *Proceedings of the IEEE International Conference on Robotics and Automation*.

Aarno, Daniel, Frank Lingelbach and Danica Kragić (2005). Constrained path planning and task-consistent path adaptation for mobile manipulators. Submitted to: *IEEE International Conference on Advanced Robotics (ICAR)*.

Akinc, Mert, Kostas E. Bekris, Brian Y. Chen, Andrew M. Ladd, Erion Plakue and Lydia E. Kavraki (2003). Probabilistic roadmaps of trees for parallel computation of multiple query roadmaps. In: *Proceedings of International Symposium of Robotics Research (ISRR)*.

Amato, N. M., O. B. Bayazit, L. K. Dale, C. V. Jones and D. Vallejo (1998 *a*). Choosing good distance metrics and local planners for probabilistic roadmap methods. In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. pp. 630–637.

Amato, N., O. Bayazit, L. Dale, C. Jones and D. Vallejo (1998 *b*). OBPRM: An obstacle-based PRM for 3D workspaces. In: *Proc. Int. Workshop on Algorithmic Foundations of Robotics (WAFR)*.

Barraquand, J. and Jean-Claude Latombe (1990). A Monte-Carlo algorithm for path planning with many degrees of freedom. In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*.

Bohlin, Robert and Lydia Kavraki (2000). Path planning using lazy PRM. In: *Proceedings of the International Conference on Robotics and Automation*. Vol. 1. pp. 521–528.

Boor, V., M.H. Overmars and F. van der Stappen (1999). The Gaussian sampling strategy for probabilistic roadmap planners. In: *Proceedings of the International Conference on Robotics and Automation*. pp. 1018–1023.

Burdick, J. W. (1989). On the Inverse Kinematics of Redundant Manipulators: Characterization of the Self-Motion Manifolds. In: *International Conference on Robotics and Automation*.

Christensen, Henrik I., Danica Kragić and F. Sandberg (2001). Vision for interaction. In: *Sensor Based Intelligent Robotics* (G.D. Hager, H.I. Christensen, H. Bunke and R. Klein, Eds.). pp. 51–73. Lecture Notes in Computer Science. Springer.

Dale, Lucia K. and Nancy M. Amato (2001). Probabilistic roadmaps - putting it all together. In: *Proceedings of the International Conference on Robotics and Automation (ICRA)*. pp. 1940–1947.

de Berg, Mark, Marc van Kreveld, Mark Overmars and Otfried Schwartzkopf (1997). *Computational Geometry: Algorithms and Applications*. Springer. BER m2 97:1 1.Ex.

Geraerts, R. and M.H. Overmars (2002). A comparative study of probabilistic roadmap planners. In: *Proceedings of the 5th International Workshop on Algorithmic Foundations of Robotics (WAFR)*.

Hsu, David, Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani and Stephen Sorkin (1998). On finding narrow passages with probabilistic roadmap planners. In: *Proceedings of the 3rd International Workshop on Algorithmic Foundations of Robotics (WAFR)*.

Hsu, David, Robert Kindel, Jean-Claude Latombe and Stephen M. Rock (2002). Randomized kinodynamic motion planning with moving obstacles.. *I. J. Robotic Res.* **21**(3), 233–256.

Hsu, David, Tingting Jiang, John Reif and Zheng Sun (2003). The bridge test for sampling narrow passages with probabilistic roadmap planners. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Kavraki, Lydia, Petr Svestka, Jean-Claude Latombe and Mark Overmars (1996). Probabilistic roadmaps for path planning in high dimensional configuration spaces. *IEEE Transactions on Robotics and Automation* **12**(4), 566–580.

Koenig, Sven and Maxim Likhachev (2002). Incremental A*. In: *Advances in Neural Information Processing Systems 14* (T. G. Dietterich, S. Becker and Z. Ghahramani, Eds.). MIT Press. Cambridge, MA.

Kuffner, James J. and Jean-Claude Latombe (2000). Interactive manipulation planning for animated characters. In: *Proceedings of the Pacific Graphics*.

Kurniawati, Hanna and David Hsu (2004). Workspace importance sampling for probabilistic roadmap planning. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Latombe, Jean-Claude (1991). *Robot Motion Planning*. Kluwer Academic Publishers.

LaValle, Steven M. (1998). Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11. Computer Science Dept., Iowa State University.

LaValle, Steven M. (2000). Motion Strategy Library. Available at http://msl.cs.uiuc.edu/msl/.

LaValle, Steven M. (2004). *Planning Algorithms*. [Online]. Available at http://msl.cs.uiuc.edu/planning/.

LaValle, Steven M. and James J. Kuffner (2000). Rapidly-exploring random trees: Progress and prospects. In: *Proceedings of the 4th International Workshop on Algorithmic Foundations of Robotics (WAFR)*.

LaValle, Steven M. and James J. Kuffner (2001). Randomized kinodynamic planning. *International Journal of Robotics Research* **20**, 378–400.

LaValle, Steven M., Michael S. Branicky and Stephen R. Lindemann (2004). On the relationship between classical grid search and probabilistic roadmaps. *IEEE Journal of Robotics Research* **23**(7-8), 673–692.

Lindemann, S.R. and Steven M. LaValle (2004). Steps toward derandomizing RRTs. In: *Proceedings of the Fourth International Workshop on Robot Motion and Control (RoMoCo)*. pp. 271–277.

Lingelbach, Frank (2004 *a*). Path planning for mobile manipulation using Probabilisitic Cell Decomposition. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*.

Lingelbach, Frank (2004 *b*). Path planning using Probabilisitic Cell Decomposition. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Luger, George F. and William A. Stubblefield (1989). *Artificial Intelligence and the Design of Expert Systems*. Benjamin/Cummings.

Mazer, E., J. M. Ahuactzin and P. Bessiere (1998). The Ariadne's Clew algorithm. *Journal of Artificial Intelligence Research* **9**, 295–316.

Miller, Andrew T., Steffen Knoop, Henrik I. Christensen and Peter K. Allen (2003). Automatic grasp planning using shape primitives. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Taipai.

Morales, Marco, Samuel Rodríguez and Nancy M. Amato (2003). Improving the connectivity of PRM roadmaps. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. Taipai. pp. 4427–4432.

Nieuwenhuisen, D. and M.H. Overmars (2002). Motion planning for camera movements in virtual environments. Technical Report UU-CS-2003-004. Utrecht University: Information and Computing Sciences. Utrecht, the Netherlands.

Pettré, J., Thierry Siméon and Jean Paul Laumond (2002). Planning human walk in virtual environments. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. Vol. 3. pp. 3048–3053.

Schwartz, J.T. and M. Sharir (1983). On the piano movers' problem: I. the case if a two-dimensional rigid polygonal body moving amidst polygonal bariers. *Communications on Pure and Applied Mathematics* **36**, 345–398.

Schwarzer, F., M. Saha and J.C. Latombe (2002). Exact collision checking of robot paths. In: *Proceedings of the 5th International Workshop on Algorithmic Foundations of Robotics (WAFR)*.

Siméon, T., Jean-Paul Laumond and C. Nissoux (2000). Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics Journal*.

Singh, Amit Pal, Jean-Claude Latombe and Douglas L. Brutlag (1999). A motion planning approach to flexible ligand binding. In: *Proceedings of the Seventh International Conference on Intelligent Systems for Molecular Biology*. pp. 252–261.

Song, Guang and Nancy M. Amato (2001). Using motion planning to study protein folding pathways. In: *Proceedings of the fifth annual international conference on Computational biology*. ACM Press. pp. 287–296.

Strandberg, Morten (2004 *a*). Augmenting RRT-planners with local trees. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Strandberg, Morten (2004 *b*). Robot Path Planning: An Object-Oriented Approach. PhD thesis. KTH, Royal Institute of Technology.

Sundaram, S., I. Remmler and N. Amato (2001). Disassembly sequencing using a motion planning approach. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Topp, Elin A., Danica Kragić, Patric Jensfelt and Henrik I. Christensen (2004). An interactive interface for a service robot. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*.

Verwer, B J H (1990). A multiresolution work space, multiresolution configuration space approach to solve the path-planning problem. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. pp. 2107–2112.