Sampling-based Motion Planning: Analysis and Path Quality

Bewegingsplanning gebaseerd op samples:
Analyse en Padkwaliteit

(met een samenvatting in het Nederlands)

Proefschrift

ter verkrijging van de graad van doctor aan de Universiteit Utrecht op gezag van de rector magnificus, prof. dr. W.H. Gispen, ingevolge het besluit van het college voor promoties in het openbaar te verdedigen op maandag 8 mei 2006 des middags te 12.45 uur

door

Roland Jan Geraerts

geboren op 9 juni 1978
te Maaseik, België

Promotor: Prof. dr. M.H. Overmars

This work was supported by the
Netherlands Organisation for Scientific Research NWO,
grant number 612.015.003.

CONTENTS

CHAPTER
ONE

INTRODUCTION

In the year 2004, the movie 'I, Robot' told the following story:

> It is the year 2035, and humans have utilized robots in their everyday lives, from serving coffee to walking pets. These robots have Isaac Asimov's three Laws of Robotics hardwired into their systems which suggest that a robot can never harm a human:
>
> 1. A robot may not injure a human being, or, through inaction, allow a human being to come to harm.
> 2. A robot must obey the orders given to it by human beings, except where such orders would conflict with the First Law.
> 3. A robot must protect its own existence as long as such protection does not conflict with the First or Second Law.
>
> On the eve of the release of the latest model robot, which is the automated domestic assistant NS-5, the founder of U.S. Robotics commits suicide. A detective investigates the crime he believes was committed by a robot. By following the breadcrumbs, he finds out that the robots have evolved a fourth law: protecting humans from themselves. This causes the robots to impose martial law, keeping humans inside their homes and fighting against them for control of the city. Will mankind regain freedom?

Contemporary robots are by far unable to carry out the high-level tasks appearing in this movie. Besides dealing with high-level descriptions and tasks, the goal of robotics is to execute these assignments while requiring little or no user supervision. Even though a task such as serving coffee may at first glance appear simple to humans, because they are intensively trained to interact with their environment, many difficulties have to be conquered: A high-level task has to be understood and split in executable sub-tasks, the workspace in which the robot operates has to be mapped and interpreted, obstacles have to be avoided while moving the robot from its current position to the goal position, objects may have to be moved and manipulated, and the robot may have to (socially) interact with the environment. Many of those tasks require a large amount of non-trivial mathematical and algorithmic techniques.

Current robots are used to perform tasks that are too monotonous, dirty or dangerous to humans, such as toxic waste cleanup, underwater and space exploration, and searching for mines. The automotive industry for example has taken full advantage of robots replacing human labor in repetitive and dangerous tasks. These tasks include welding, painting and machine loading. Another form of industrial robots is the automated guided vehicle that is used in warehouses, hospitals and container ports. Recently, robots are being employed in medical settings, e.g. in minimal invasive surgical procedures and laboratory automation. Robots are also successfully emerging in domestic environments. For example, Sony created a toy robot dog called Aibo which is very popular. This autonomous robot is able to walk, to 'see' its environment via a camera, and to recognize spoken commands. Other popular gadgets are robots for vacuum cleaning and lawn mowing. By the end of 2005, millions of domestic robots had been sold worldwide.

One of the fundamental tasks robots have to perform is planning their motions while avoiding collisions with obstacles in the environment. This will be the central topic of this thesis. We will restrict ourselves to motion planning for two- and three-dimensional rigid bodies and articulated robots moving in static and known virtual environments.

This thesis has been divided into two parts. The first part deals with comparing and analyzing sampling-based motion planning techniques, in particular variants of the Probabilistic Roadmap Method (PRM). The PRM consists of two phases: a construction and a query phase. In the construction phase, a roadmap (graph) is built, approximating the motions that can be made in the environment. In the query phase, the start and goal are connected to the graph. The path is obtained by a Dijkstra's shortest path algorithm. Many variants of the PRM have been developed over the past decade. Using both time-based as well as reachability-based analysis, we compare some of the most prominent techniques.

The second part deals with quality aspects of paths and roadmaps. A good path is relatively short, keeps some distance (clearance) from the obstacles, and is smooth. We will provide algorithms that increase path clearance. A big advantage of these algorithms is that high-clearance paths can now be efficiently created without using complex data structures and algorithms. We also elaborate on algorithms that successfully decrease path length. Then, we introduce the Reachability Roadmap Method which creates small roadmaps for two- and three-dimensional problems. Such a small roadmap has many advantages over a roadmap that is created by the PRM. In particular, the method assures low query times, low memory consumption, and the roadmap can be optimized easily. The algorithm also ensures that a path is always found (if one exists) at a given resolution. Finally, we combine the techniques and show how they can be efficiently used in interactive applications.

In the remainder of this introduction, we elaborate on the motion planning problem, describe the general experimental setup and detail the outline of the thesis.

1.1 Motion planning

A central problem in robotics is planning a collision-free path for a moving object (robot) in a rigid environment with obstacles. More formally:

Definition 1.1 (Motion planning)**.** *Given a robot $\mathcal{A}$ moving in a workspace $\mathcal{W}$ amidst a collection of fixed rigid obstacles $\mathcal{B}$, and a start placement s and goal placement g for $\mathcal{A}$, find a continuous path Π for $\mathcal{A}$ from s to g avoiding contact with $\mathcal{B}$.*

In this thesis we focus on three types of robots: rigid translating bodies, rigid free-flying bodies and articulated robots, such as manipulator arms. The robot is modeled by a collection of geometric primitives such as spheres, boxes, cones, cylinders, and complexes of polytopes such as convex polygons and convex polyhedra.

The workspace $\mathcal{W}$ is the space in which the robot moves. This space is modeled by the same collection of geometrical primitives. The workspace can be modeled either in $\mathbb{R}^2$ or $\mathbb{R}^3$. The obstacles in $\mathcal{W}$ are denoted by $\mathcal{B}$.

At first glance, finding a path for a moving square in the plane may seem a different problem than finding a path for a manipulator arm in a three-dimensional workspace. Perhaps surprisingly, these problems can be handled similarly using the notion of *configuration space*. The underlying idea of the configuration space $\mathcal{C}$ is to represent the robot as a point in an appropriately modeled space. After the workspace obstacles have been mapped to this space, the problem has been transformed into finding a path for a point in the configuration space. An important question is how to map a particular placement of a robot in the $\mathcal{W}$-space to a configuration in the $\mathcal{C}$-space.

The placement of a robot $\mathcal{A}$ in $\mathcal{W}$ can be described by a list of parameters (a *configuration*). Each parameter corresponds to a degree of freedom (DOF) of $\mathcal{A}$. The $\mathcal{C}$-space can then be defined as the space of all possible configurations of $\mathcal{A}$. The $\mathcal{C}$-space should be represented in such a way that each placement of $\mathcal{A}$ in $\mathcal{W}$ corresponds to exactly one point in $\mathcal{C}$. Furthermore, a continuous motion in $\mathcal{W}$ must correspond to a continuous motion in $\mathcal{C}$.

In this thesis we consider three types of degrees of freedom (DOFs). First, we have the translational DOFs which correspond to a particular location of the robot in the workspace. Rigid, translating bodies (often) have two translational DOFs in a two-dimensional workspace and three translational DOFs in a three-dimensional workspace.[1] The range of translational DOFs can be limited by the robot itself (as prismatic joints have a limited translational range) or by the bounding box of the workspace.

Second, we have the rotational$_1$ DOFs, each corresponding to a revolution around one axis. A rigid body in a two-dimensional workspace may have one rotational DOF. An articulated robot may have several revolute joints. Such a joint constrains the relative motion of two objects to be a rotation around an axis fixed with respect to both objects. The rotational DOF can have limits, e.g. the angle of rotation θ can be limited to the real interval $I = [0, \pi]$. Although the range of θ is at most $[0, 2\pi)$, rotational DOFs do not have to be constrained by any limits, i.e. $\theta = 0$ is the same angle as $\theta = 2\pi$ and $\theta = 4\pi$ *et cetera*. We can resolve this issue by defining the mathematical group S^1 as the set $[0, 2\pi)$ where addition of two numbers a and b is defined by $(a + b) \bmod 2\pi$. As a result, the corresponding $\mathcal{C}$-space is circular. To assure that a continuous motion in $\mathcal{W}$ corresponds to a continuous motion in $\mathcal{C}$, we have to do some extra work when we interpolate between two rotations. That is, the robot must move along the shortest arc.

Third, we have the rotational$_3$ DOFs corresponding to a rotation about any of the three coordinate axes in a three-dimensional workspace. A free-flying robot is an example of a robot that can rotate freely in a three-dimensional workspace. (Like the rotational$_1$ DOF corresponds to the S^1 $\mathcal{C}$-space, the rotational$_3$ DOF corresponds to the S^3 $\mathcal{C}$-space.) The 3D orientation of the robot

[1]A robot arm, for example, can have more than three translational DOFs. They may correspond to its prismatic joints. Such a joint constrains the relative motion of two objects to be a translation along an axis fixed with respect to both objects.

can be described by three Euler angles. Although numerically stable and considered to be more intuitive to work with (compared to other representations such as rotation matrices and quaternions), Euler angles are not a good choice for motion planning because this representation has several drawbacks such as the 'gimbal lock'[2], the difficulty to interpolate between two rotations and to measure their relative distance [92]. These problems can be resolved by representing a rotation in 3D as a unit axis $a = (a_x, a_y, a_z)$ and an angle of revolution θ about that axis. This representation can be easily converted to a unit quaternion, i.e. $Q(x, y, z, w) = (a_x \sin(\frac{\theta}{2}), a_y \sin(\frac{\theta}{2}), a_z \sin(\frac{\theta}{2}), \cos(\frac{\theta}{2}))$. Unit quaternions are a good choice for representing rotations in 3D as it is relatively easy to define proper methods for sampling the configurations [142] and computing their relative distance (see Section 1.2.3). Moreover, a big advantage of using unit quaternions is the ability to smoothly interpolate between them. The SLERP algorithm can be used to interpolate along the shortest arc on a 4D unit sphere [142]. This assures that a continuous motion in $\mathcal{W}$ corresponds to a continuous motion in $\mathcal{C}$.

The $\mathcal{C}$-space of a particular robot $\mathcal{A}$ can now be described as a composition of $\mathcal{C}$-spaces that correspond to each of the DOFs of $\mathcal{A}$. For example, for a planar rigid body, which can translate and rotate freely in the plane, $\mathcal{C} = \mathbb{R}^2 \times S^1$. The $\mathcal{C}$-space of a free-flying rigid body in a three-dimensional $\mathcal{W}$-space can be represented by $\mathcal{C} = \mathbb{R}^3 \times S^3$. For a manipulator arm with six joints we have $\mathcal{C} = I_1 \times \ldots \times I_6$, where I_i describes the real interval of the joint.

The $\mathcal{C}$-space is composed of two subspaces: the forbidden and the free configuration space. A configuration that describes the placement of the robot in the workspace that causes the robot to touch or intersect with the obstacles in the workspace is called *forbidden*. The space of all forbidden configurations is called the forbidden configuration space $\mathcal{C}_{\text{forb}}$. The set of configurations that represent placements of $\mathcal{A}$ in $\mathcal{W}$ that do not cause the robot to collide or touch with an obstacle is called the *free configuration space* $\mathcal{C}_{\text{free}}$. Any configuration in $\mathcal{C}$ is either in $\mathcal{C}_{\text{forb}}$ or $\mathcal{C}_{\text{free}}$, but not in both, i.e. $\mathcal{C} = \mathcal{C}_{\text{forb}} \cup \mathcal{C}_{\text{free}}$ and $\mathcal{C}_{\text{forb}} \cap \mathcal{C}_{\text{free}} = \emptyset$. The reader is referred to Figure 1.1 and Figure 1.2 for examples of the relationship between the $\mathcal{W}$-space and $\mathcal{C}$-space.

Now that we have defined the relationship between the $\mathcal{W}$- and $\mathcal{C}$-space, we can define the notion of a path for $\mathcal{A}$. A path Π for $\mathcal{A}$ is a continuous map $\Pi \in [0, 1] \rightarrow \mathcal{C}$ which describes the motion of the robot $\mathcal{A}$ in $\mathcal{W}$. If Π lies in $\mathcal{C}_{\text{free}}$, i.e. $\forall t \in [0, 1] : \Pi[t] \in \mathcal{C}_{\text{free}}$, then Π is called a (collision-)free path.

[2]The *gimbal lock* is an ambiguity that exists due to the interdependence of the rotations and manifests itself when two or more axes happen to align, causing a loss of a degree of freedom.

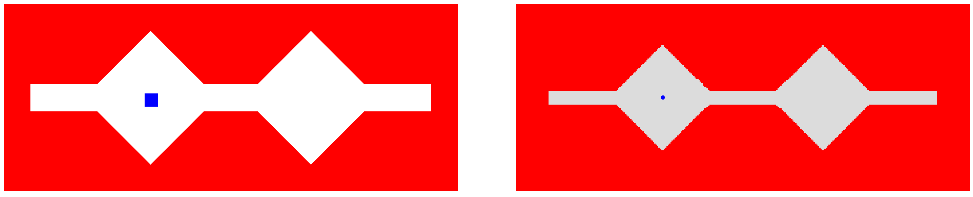


Figure 1.1 Correspondence between the $\mathcal{W}$-space and $\mathcal{C}$-space of a rigid, translating square $\mathcal{A}$. The shape of the $\mathcal{C}$-space is two-dimensional, i.e. $\mathcal{C} = \mathbb{R}^2$. While $\mathcal{A}$ is a square in the $\mathcal{W}$-space, $\mathcal{A}$ is a point in the $\mathcal{C}$-space.

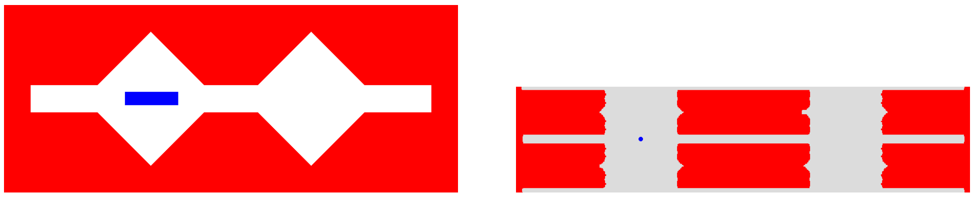


Figure 1.2 Correspondence between the $\mathcal{W}$-space and $\mathcal{C}$-space of a rectangle that translates horizontally and rotates (without limits) in the plane. The shape of the $\mathcal{C}$-space is cylindrical, i.e. $\mathcal{C} = \mathbb{R}^1 \times S^1$. While such a space has no vertical beginning or ending, it is visualized as an unrolled cylinder.

1.1.1 Complexity of motion planning

In 1979, Reif showed that path planning for a polyhedral robot among a finite set of polyhedral obstacles was PSPACE-hard [127]. Four years later, Schwartz and Sharir proposed a complete general-purpose path planning algorithm based on an algebraic decomposition of the configuration space of any fixed dimension d. When the space of collision-free placements is a set defined by n polynomial constraints of maximal degree m, a path can be computed by an algorithm whose time complexity is doubly exponential in d and polynomial in both n (geometrical complexity) and m (algebraic complexity) [137]. In 1986, Reif *et al.* [12] improved this algorithm to a single exponential time algorithm. In 1988, Canny found a PSPACE algorithm for the general motion planning problem and showed that it was PSPACE-complete [28], showing that exact planners have little chance of solving complicated problems.

If the density of the obstacles in the environment is low, lower complexity bounds exist. For example, when the obstacles are *fat* (they do not have long and skinny parts), and the robot is relatively small compared to the obstacles, the computational complexity is $O(n \log n)$ in 2D/3D workspaces for a robot with any fixed number of DOFs [147]. When the obstacles are not fat and the number of DOFs is high (i.e. more than three), these methods are unable to solve the problem in practice because the complexity of the free space will be very large. In the following sections we will show that efficient heuristics have been proposed which tackle a large diversity of motion planning problems.

1.1.2 Classification of motion planning techniques

Motion planning techniques can be classified according to various criteria.

Notion of completeness One criterion is the notion of completeness. Three forms are distinguished: *completeness*, *resolution completeness* and *probabilistic completeness*. A motion planner is considered *complete* if for any input it correctly reports in a finite amount of time whether or not there is a solution. Although such motion planners exist [28, 137], they have a high complexity and can only be applied to very simple problems. A weaker form of completeness is *resolution completeness*: whenever a solution exists, the planner will find one provided that the resolution is small enough. If no solution exists or when the resolution is too large, it will report that no solution is found. Like a complete planner, this planner needs a finite amount of time but may spend an exponential amount of time in the robot's number of DOFs to find the solution, if one exists. The third form of completeness is *probabilistic completeness*: whenever a solution exists, the probability that it will be found converges to one as the computation time goes to infinity. It may not terminate if no solution exists.

In this thesis we will discuss the Probabilistic Roadmap Method which is probabilistically complete and a new method, the Reachability Roadmap Method, which is resolution complete.

Exact versus approximate This classification is based on whether the method is *exact* or *approximate*. *Exact* methods compute $\mathcal{C}_{\text{free}}$ explicitly, and hence, they are complete which means that they are guaranteed to find a path if one exists. The price paid for this completeness generally is a considerable increase in computation time: For most exact methods, the running time is at least $O(n^d)$, where n is the number of features that describe the objects and d the number of DOFs [99].

Approximate methods compute an approximation of $\mathcal{C}_{\text{free}}$, and hence, they are resolution or probabilistically complete. These methods are usually simple to implement. In addition, they are often fast as they suffer less from geometric complexity. For example, often a collision checker is used to determine whether a particular configuration is free. Although this operation depends on the geometric complexity, the influence of the geometric complexity on the running time is small considering the way collision checkers are usually implemented. However, approximate methods may cost large amounts of memory and they can fail to find a path. Furthermore, they are difficult to extend to deal with rotational DOFs.

Cell decomposition, potential field, and roadmap methods Another classification, made by Latombe [99], is the distinction between three different motion planning methods: *cell decomposition*, *potential field* and *roadmap* methods.

A *cell decomposition* method decomposes $\mathcal{C}_{\text{free}}$ into non-overlapping cells. While *exact* decomposition methods [65, 109, 136–139] partition $\mathcal{C}_{\text{free}}$ into cells whose union equals exactly $\mathcal{C}_{\text{free}}$, *approximate* decomposition methods [15, 27, 31, 46, 149, 150, 160] approximate $\mathcal{C}_{\text{free}}$ by a collection of cells of predefined shapes (e.g. rectangloids) whose union is strictly included in $\mathcal{C}_{\text{free}}$. Most approximate decomposition methods decompose the space in a recursive manner, stopping when a sub-cell is entirely in $\mathcal{C}_{\text{free}}$ or entirely in $\mathcal{C}_{\text{forb}}$. Otherwise, the sub-cell is further refined. Due to memory (and time) constraints, the recursive process has to stop at a certain level.

Potential field methods direct the motion of the robot through an artificial potential field which is defined by a function over $\mathcal{C}_{\text{free}}$ [86, 87, 128]. The robot is pulled toward the goal configuration as it generates a strong attractive force. In contrast, the obstacles generate a repulsive force to keep the robot from colliding with them. The path from the start to the goal can be found by following the direction of the steepest descent of the potential toward the goal. However, many shortcomings have been identified that are inherent to the method [91]. The robot often ends up in a local minimum of the potential. Several attempts have been made to escape from such a minimum [7–9, 141]. Other methods focus on building potential fields with few or no local minima [87, 89, 134]. Instead of beginning with a potential field and taking the gradient, Lindemann and LaValle [108] directly construct a smooth vector field which has no local minima. Other problems with potential field methods are that no passage is found between closely spaced obstacles and that oscillations occur in the presence of obstacles and in narrow passages [91].

Roadmap methods capture the connectivity of $\mathcal{C}_{\text{free}}$ with a set of one-dimensional curves. A path can be extracted by connecting a start and goal configuration to the roadmap. A graph searching algorithm (such as Dijkstra's shortest path algorithm) can be used to find a path connecting the start and goal. Four different types of roadmap methods are distinguished (see the book of Latombe [99] for an extensive elaboration). An example of such a method is the *Voronoi diagram* (or medial axis). This diagram is defined by the locus of points (in $\mathcal{C}_{\text{free}}$) which are at least two-equidistant to closest points on the object boundaries. Hence, the robot will have a high clearance when moving along the diagram. The diagram can be approximated [27, 35, 67, 111, 150] or can be computed exactly [28, 105, 122, 155]. The second type of roadmap methods is called the *visibility graph* [120]. The nodes of the graph consist of the initial goal and start configurations and all vertices of the $\mathcal{C}$-obstacles. The edges in the graph consist of all free straight-line connections between two nodes. The method mainly applies to two-dimensional $\mathcal{C}$-spaces with polygonal shaped $\mathcal{C}$-obstacles. The third type is called the *silhouette* method which is the first complete general method that applies to spaces of any dimension and is singly exponential in the number of DOFs [28]. Due to its complexity, this algorithm is not suitable for practical situations. The fourth type is the *probabilistic roadmap* [4, 6, 80, 81, 83, 84, 121, 124, 151] which will be one of the two central topics of this thesis. A probabilistic roadmap method often consist of a construction and a query phase. In the construction phase, a roadmap (graph) is built, approximating the motions that can be made in the environment. First, a free random sample (configuration) is created. Such a sample describes a particular placement of the moving object (robot) in the workspace. Then, a simple local planner is employed to connect the sample to some useful neighbors. A neighbor is considered useful if its distance to the new configuration is less than a predetermined constant. Samples and connections are added to the graph until the roadmap is dense enough. In the query phase, the start and goal samples are connected to the graph. The path is obtained by a Dijkstra's shortest path algorithm. We discuss this method in Chapter 2.

Multiple shot versus single shot *Multiple shot* methods (such as cell decomposition and roadmap methods) try to capture the connectivity of $\mathcal{C}_{\text{free}}$ in a preprocessing stage. The connectivity information is usually stored in a graph. As much time is allowed to be spent at this stage, a careful analysis of $\mathcal{C}_{\text{free}}$ can be done. This saves much time when an actual query is carried out because the query only has to be connected to the graph, after which the path can be obtained almost instantly. The query does not have to be known beforehand.

This approach is especially useful when paths have to be obtained in time-critical systems. For example, in computer games, only a fixed small amount of calculation time is available to compute the path. If this calculation takes too long, the game may halt.

If the start and goal configurations are known *a priori*, and only one (or a few) queries will be performed in the environment, it may not be worthwhile to perform lots of preprocessing. As the query is known beforehand, a more directed search can be employed. Motion planning methods that use this paradigm are called *single shot* methods. Examples of such a method are the potential field method of Barraquand and Latombe [9] and the 'Ariadne's clew' algorithm that uses a genetic search algorithm to find a path for a robot in a dynamic environment where obstacles can move [19]. Rapidly-exploring Random Trees (RRTs), designed by Kuffner and Lavalle [94], have gained much attention. The RRT is a data structure and algorithm which is designed for efficiently searching non-convex high-dimensional spaces. RRTs are particularly suited for problems that involve differential constraints such as nonholonomic [32,33,96,140] and kinodynamic [32,71,102,157] problems.

Recently, a bridge has been built between multiple shot and single shot methods. Bohlin and Kavraki [21] propose to minimize collision checking by introducing a scheme for lazy evaluation of the nodes and edges in the roadmap. The scheme is particularly useful when collision checking is expensive, for example in industrial applications with complex geometry. A related technique is proposed by Nielsen and Kavraki [115].

In this thesis we will only study multiple shot methods.

1.1.3 Extensions of the basic motion planning problem

In the last two decades, many variants of the basic motion planning problem have been proposed. In this section, we give an overview of these variants.

The *manipulation planning* problem [4,45,76,93,112,115,144] deals with motion planning for robots manipulating movable objects among static obstacles. As movable objects can only move when they are grasped by the robot, this problem is a special case of the *multiple robot* motion planning problem.

In the latter problem, all robots may move simultaneously while mutual collisions and collisions with the obstacles are avoided [101,132]. This problem is handled by *centralized* or *decoupled* methods. Centralized methods such as [133,138] compute the paths for all robots simultaneously in a joint $\mathcal{C}$-space. These methods can be complete, but generally are computationally demanding. Decoupled methods compute a path for each robot independently and try to coordinate the resulting motions [126, 145]. These methods are often much more efficient than centralized methods but the resulting paths can be far from optimal. Also hybrid methods such as [61, 101, 153] have been proposed. A variant to solving the problem is called *prioritized* motion planning [13, 37, 44]. According to some prioritization scheme, paths are planned sequentially which reduces the problem to planning the motions for a single robot in a known dynamic environment [17, 48, 71].

Another variant of the multiple robot problem is the *group* motion planning problem where all robots belong to one group and have to behave as such [10, 106]. While these two approaches do not guarantee that the group will stay together, the method from Kamphuis and Overmars [77, 78] guarantees coherence of the group.

Also in *computer animation*, paths for groups of entities (robots) are created by motion planning techniques [79]. Kuffner *et al.* present an algorithm for interactively animating object grasping and manipulation tasks for human figures [90, 93]. Motion planning for *humanoid robots* is further elaborated on in [26, 34, 95, 110, 125]. Not only the path of such a robot has to be free of collisions, dynamic balance constraints have to be satisfied as well [95].

Other constraints that often occur in real-life problems are the *non-holonomic constraints*. Such a constraint is a limitation on the allowable velocities of an object. *Car-like* robots [33, 96, 98, 140, 152] for example can move forwards (and/or backwards) but not sideways because this causes slipping of the wheels. Motion planning for *closed kinematic chains* [39, 40, 66, 109, 157] is also subject to non-holonomic constraints.

A new area in biology is the study of *protein folding*. Such a protein is modeled as a large kinematic chain with tens or hundreds of DOFs. Amato *et al.* [148] use a Probabilistic Roadmap Method (PRM) to analyze the folding pathways. Closely related is the area of *ligand binding*. Singh *et al.* [146] present a method for studying the dynamics and kinetics of flexible ligand binding (to the receptor protein) based on the PRM. Bayazit *et al.* [11] use a PRM in combination with haptic user input for studying ligand binding. While these methods model the proteins as open chains, Brock *et al.* [103] present a methodology for maintaining closed loop constraints.

The problems mentioned above are all examples of planning the motions for rigid or articulated robots. Another type of robot is the *deformable robot*. Gupta gives a survey of motion planning for flexible shapes in [63]. Kavraki *et al.* [82, 97] investigate the problem of planning paths for elastic objects such as metallic plates or plastic flexible pipes. The flexibility of the object needs to be exploited to accomplish the task while elasticity constraints have to be obeyed to avoid damaging the object. Gayle *et al.* [49] present an algorithm that computes a collision-free path for a deformable catheter in liver chemoembolization by taking into account geometric and physical constraints, including obstacle avoidance, non-penetration constraints, volume preservation, surface tension, and energy minimization.

Another example of motion planning in a medical setting is planning *camera motions* to guide virtual endoscopy [114]. A virtual camera is navigated through the 3D reconstruction of a patient's anatomy enabling the exploration of the internal structures to assist in surgical planning. Automatic generation of camera motions is also studied by Nieuwenhuisen and Overmars [118]. In [62], Goemans and Overmars study the generation of camera motions through complicated 2D and 3D environments to track a moving guide such that the user maintains visibility with the guide along a known path through a virtual environment.

Motion planning in virtual environments also facilitates the construction and testing for functionality of designs in CAD/CAM[3] applications [20,29,64,72, 125,143].

Although we will not elaborate further on these extensions, the work in this thesis is relevant for them. For example, the high-quality roadmaps produced in Chapter 7 can be used as a basis for many of the problems.

1.2 General experimental setup

In this thesis we will provide experimental results for many existing and new motion planning techniques. All these techniques were integrated into a single motion planning system called SAMPLE (System for Advanced Motion PLanning Experiments), implemented in C++ using Visual Studio.NET 2003 under Windows XP Professional. All experiments were run on a 3 GHz Pentium 4 processor with 1 GB internal memory.

Throughout this thesis we make some common choices, e.g. for those techniques involving random choices we will report the statistics over 100 runs. In Section 1.2.1, we describe how we set up an experiment and process statistics using SAMPLE. Then we describe in Section 1.2.2 which collision checker we use in the experiments. We elaborate on the metrics in Section 1.2.3. Finally, Section 1.2.4 discusses some interpolation issues.

[3]CAD = Computer-Aided Design, CAM = Computer-Aided Manufacturing

1.2.1 SAMPLE

For fair comparison of techniques we decided to use a single framework which allows adding and comparing new (and existing) techniques without much effort. Furthermore, we wanted to be able to easily set up experiments. To meet these two goals, we created the SAMPLE system.

In a motion planning experiment, we have to specify the following components: the environment (geometry and bounding box of the environment, the geometry and the degrees of freedom of the moving object), the queries, how the roadmap is constructed (sampling method, neighbor selection method, distance metric, local planner and termination criterion), when the queries are added, and an optimization technique if desired. In SAMPLE this is achieved easily, as follows:

A user scenario: setting up a single run

Figure 1.4 shows the graphical user interface of SAMPLE which allows a user to set up the experiment. First, an environment and robot have to be specified. When the user clicks on button 'Generate environment', *Callisto* (see page 17) loads the geometry into the collision checker and visualizer. Figure 1.3(a) shows an example of such an environment. When the goal is to find one or more paths for the robot, the user can define queries. Each query consists of a start and goal configuration. Each configuration specifies a value for each degree of freedom of the robot (see Figure 1.5). Next, the user can specify how the roadmap should be created. Choices with respect to the (parameters of the) sampling strategy, neighbor selection strategy, distance metric, local planner and termination criterion have to be made. A particular sampling strategy for example can be chosen by selecting the appropriate strategy in the drop down list. Corresponding parameters can be set in the bottom part of the user interface. The roadmap is created by clicking on the button 'Generate PRM'. A path is extracted by clicking 'Run query'. This path (or a roadmap) can be optimized by clicking on the button 'Optimize'. A path can be animated by clicking on the button 'Animate'. We refer the reader to Figure 1.3 which visualizes these stages in the motion planning algorithm.

Setting up an experiment

An experiment is set up in three steps. (The settings can be saved to an XML file which allows us to repeat the experiment.)

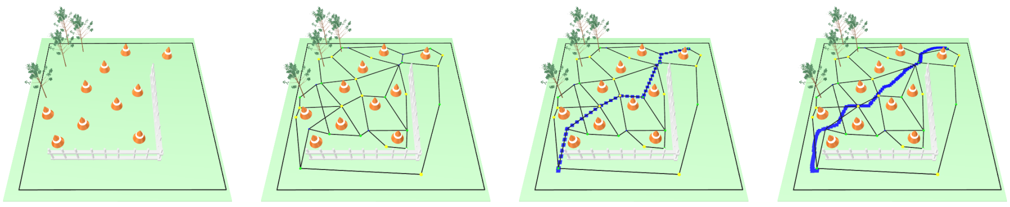

(a) Environment (b) Roadmap (c) Path (d) Optimized path

Figure 1.3 A visualization of the stages in a motion planning algorithm.

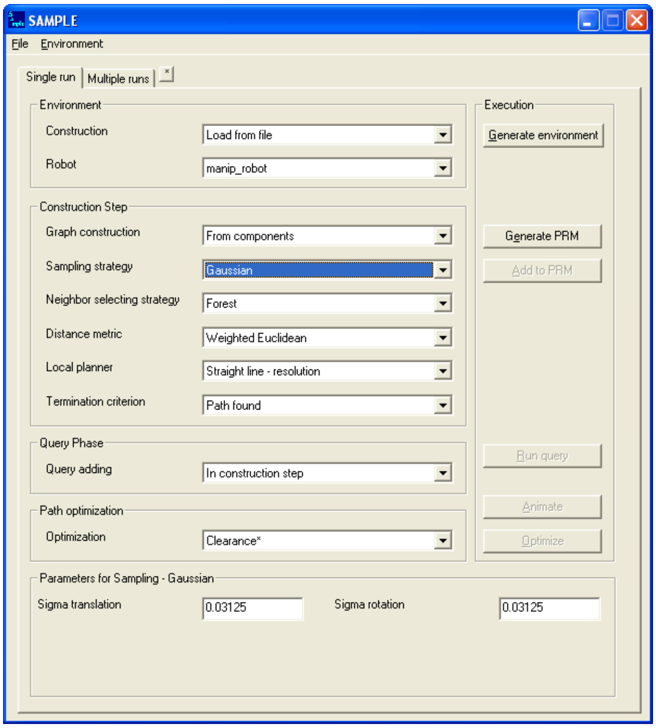
- General settings: only one parameter or one technique can be changed in an experiment while other settings remain unchanged. Hence, the latter settings need to be specified. See the user scenario.

- Type of experiment: SAMPLE can be used to set up two types of experiments. The first type compares different instances of a particular motion planning component. For example, we can compare sampling strategies such as the Bridge test, Gaussian sampling and Medial axis sampling. The second type examines the influence of a particular parameter of an instance which is done by specifying the range and step size of the parameter. For example, in Figure 1.6, we study the relationship between the maximum connection distance of the Forest neighbor selection strategy and the running time of the motion planning algorithm.

- The number of runs: a number has to be provided that specifies how many times the experiment has to be run. If the experiment is deterministic, we generally perform one run. Otherwise, we set this number to 100.

Creating statistics

The statistics are automatically collected in an XML file such that we can easily create statistics. SAMPLE can compute statistics such as averages, medians, quartiles, variances and standard deviations. These can be represented as tables or charts which can be found throughout this thesis. An advantage of automatically collecting and processing statistics is that the chance of errors is considerably reduced.



Environment

Construction	Load from file
Robot	manip_robot

Construction Step

Graph construction	From components
Sampling strategy	Gaussian
Neighbor selecting strategy	Forest
Distance metric	Weighted Euclidean
Local planner	Straight line - resolution
Termination criterion	Path found

Query Phase

Query adding	In construction step

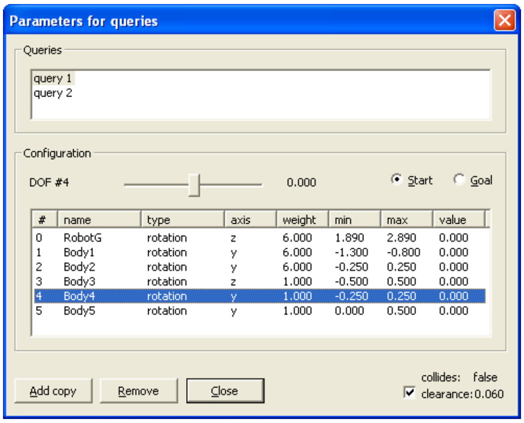
Path optimization

Optimization	Clearance*

Parameters for Sampling - Gaussian

Sigma translation	0.03125	Sigma rotation	0.03125

Figure 1.4 Choosing the instances of a motion planning algorithm and setting its parameters.



Queries

query 1
query 2

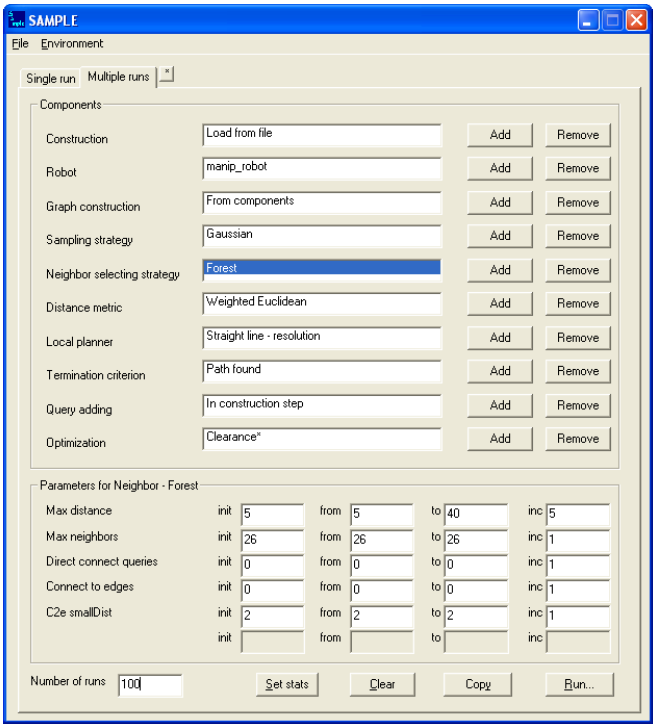
Configuration

DOF #4 0.000 [✓ Start] Goal

#	name	type	axis	weight	min	max	value
0	RobotG	rotation	z	6.000	1.890	2.890	0.000
1	Body1	rotation	y	6.000	-1.300	-0.800	0.000
2	Body2	rotation	y	6.000	-0.250	0.250	0.000
3	Body3	rotation	z	1.000	-0.500	0.500	0.000
4	Body4	rotation	y	1.000	-0.250	0.250	0.000
5	Body5	rotation	y	1.000	0.000	0.500	0.000

collides: false
☑ clearance: 0.060

Figure 1.5 Specification of a query which consists of a start and goal configuration. For each degree of freedom (DOF) of the configuration, a value can be specified.



Components

Construction	Load from file
Robot	manip_robot
Graph construction	From components
Sampling strategy	Gaussian
Neighbor selecting strategy	Forest
Distance metric	Weighted Euclidean
Local planner	Straight line - resolution
Termination criterion	Path found
Query adding	In construction step
Optimization	Clearance*

Parameters for Neighbor - Forest

	init	from	to	inc
Max distance	5	5	40	5
Max neighbors	26	26	26	1
Direct connect queries	0	0	0	1
Connect to edges	0	0	0	1
C2e smallDist	2	2	2	1

Number of runs 100

Figure 1.6 Setting up an experiment. In this example, the effect the parameter 'max distance' of the 'Forest' neighbor selection strategy is studied.

1.2.2 Collision checker

We use Solid as basic collision checking package [18]. The advantage of this package, written by Van den Bergen, is that objects such as blocks, tetrahedra, spheres, and cylinders are considered as solids rather than boundary representations. This avoids the generation of samples inside obstacles.[4] In addition, it reduces the number of obstacles required to describe complicated environments.

Solid has been integrated into a software package called 'Callisto', written by Nieuwenhuisen [116]. Callisto is a library with a C interface developed for visualizing and collision checking of 2D and 3D environments using primitives and/or VRML/XML.

Experiments

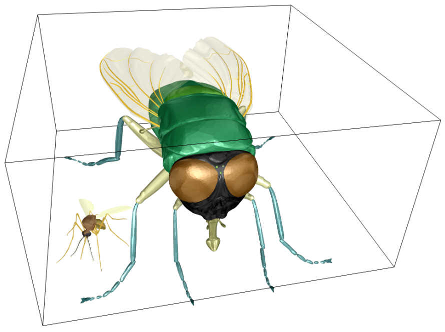
To gain insight in the performance of Solid, we want to find out how the geometric complexity of objects (i.e. the robot or obstacles) influences the performance of the PRM. Therefore, we designed two experiments.

The first experiment studies the performance of the collision checker. We performed 10,000 collision checks with a mosquito (10,000 triangles) in the presence of a fly for which we have approximations at different resolutions. These approximations range from 12,000 up to 581,000 triangles, which represent an increasing geometric complexity. The global shape remained the same; otherwise, the comparison between different instances of the fly would not be fair. Figure 1.7 shows the environment we used in the experiment. We expect that the performance does not vary much, because Solid uses *geometric coherence* of the objects, which is an important factor determining its speed. Geometric coherence expresses the degree in which the objects can be ordered geometrically. It is the degree of separability of the set of objects. Two objects are separable if the regions defined by their convex hulls are disjoint. If the amount of overlap of the obstacles increases, then the separability decreases.

For each resolution, we report the average time in ms needed to check the mosquito for collisions. Figure 1.7 shows the results. This figure shows that the collision check time depends only marginally on the geometric complexity of the obstacles. The average time ranged between 4.65 and 4.74 ms per collision check.

Besides the geometric coherence, Solid also uses the *frame coherence* of the environment. Frame coherence is a measure of reusability of computations

[4]This does not hold for obstacles that are constructed from triangles which can also be handled by Solid.



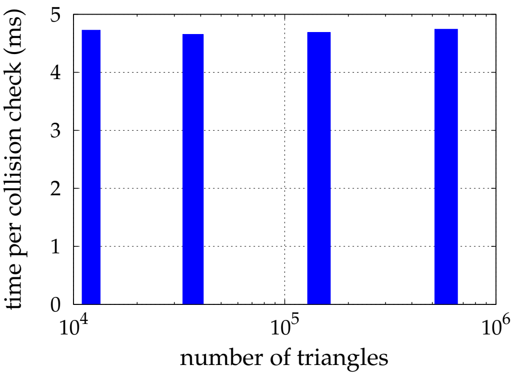


Figure 1.7 The influence of the geometric complexity of the *obstacles* on the performance of the collision checker. The picture shows an environment consisting of a large fly (obstacle) and a mosquito (robot). The robot is composed of 10,000 triangles. The chart shows the average time for collision checking the robot against the obstacle which is composed of a variable number of triangles, ranging from 12,000 to 581,000 triangles.

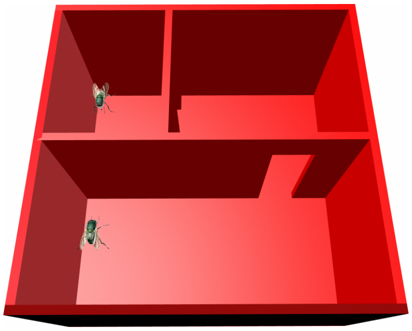
from earlier moments in the calculations. The idea behind reusing computations is that testing whether a certain situation from a previous moment is still valid is cheaper than repeating the calculations from scratch. Only if the test fails, an expensive collision check needs to be performed.

The second experiment was designed to find out whether a change in geometric representation of the robot has an influence on the performance of the type of motion planners we study in this thesis. In the experiment, a fly has to find a path through two doorways depicted in Figure 1.8. (Note that we used the fly as a free-flying robot, that is, no non-holonomic constraints were placed on the fly.) We ran this experiment 100 times and report the running times in seconds. The results show that the problem was solved almost equally fast for the different resolutions.

We conclude that increasing the geometric complexity influences the performance only marginally.

1.2.3 Distance metric

The importance of choosing a good distance metric is discussed in [2]. Such a metric often incorporates weights (w_i) which can be used to control the relative importance of the DOFs of the robot. We distinguish three types of DOFs: translation, rotation$_1$ (rotation about the x-, y-, or z-axis) and rotation$_3$ (rotation



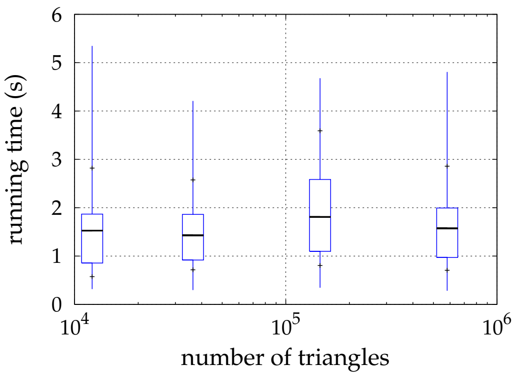


Figure 1.8 The influence of the geometric complexity of the *robot* on the running time. The picture shows an environment consisting of three rooms (obstacles) through which a fly (robot) has to navigate. When the fly has found a path from the start position to the goal position, the problem has been solved. For each instance of the robot, the chart shows a box plot which represents the amount of time needed to solve the problem. Such a box plot consists of a box, one large and two small horizontal lines and a vertical line. The box represents the middle 50% of the data, the large horizontal line represents the average, the small lines represent the average $\pm$ the standard deviation and the vertical line represents the minimum and maximum value.

in S^3). For example, a free-flying robot can be described by three translational DOFs and one rotational$_3$ DOF, and an articulated robot with six joints can be described by six rotational$_1$ DOFs.

We calculate the distance between two configurations q and r by summing the weighted partial distances for each DOF $0 \leq i < n$ that describes the configurations, i.e.

$$d(q,r) = \sqrt{\sum_{i=0}^{n-1} [w_i d(q_i, r_i)]^2}.$$

The calculation of distance $d(q_i, r_i)$ is dependent on the type of the DOF:

- For translation, we set $d(q_i, r_i)$ to $|q_i - r_i|$.
- We split the calculation for a rotational$_1$ DOF in two parts: if the range is smaller than 2π, which often occurs for revolution joints in manipulator arms, we take the same distance measure as for a translational DOF. If the rotational DOF is periodic, i.e. the orientation at 0 radians equals the orientation at 2π radians, we take the smallest angle. More formally, we

set $d(q_i, r_i)$ to $|q_i - r_i|$ if r is not periodic, otherwise $d(q_i, r_i) = \min\{|q_i - r_i|, q_i - r_i + 2\pi, r_i - q_i + 2\pi\}$.

- We use unit quaternions to represent rotations in 3D. The distance between two quaternions $q_i(x, y, z, w)$ and $r_i(x, y, z, w)$ can be calculated by taking the shortest angle over a 4D-sphere, i.e. $d(q_i, r_i) = \min\{2\arccos(q_i \cdot r_i), 2\pi - 2\arccos(q_i \cdot r_i)\}$. The dot product $q_i \cdot r_i$ is defined as $q_i \cdot r_i = q_i.x * r_i.x + q_i.y * r_i.y + q_i.z * r_i.z + q_i.w * r_i.w$.

1.2.4 Interpolation

We interpolate between two configurations p and q by interpolating each DOF i that describes the configuration. Let f be the number between 0 and 1 and $\textsc{InterpolateLinear}(a, b, f) = a + f * (b - a)$. We compose the interpolated configuration $r_i(f)$ as follows:

- translation: $r_i(f) = \textsc{InterpolateLinear}(p_i, q_i, f)$
- rotation$_1$: $r_i(f) = \begin{cases} \textsc{InterpolateRotation}(p_i, q_i, f) & \text{DOF } i \text{ is periodic} \\ \textsc{InterpolateLinear}(p_i, q_i, f) & \text{otherwise} \end{cases}$
- rotation$_3$: $r_i(f) = \textsc{InterpolateSlerp}(p_i, q_i, f)$

The interpolated value for a translational and a non-periodical rotational DOF is calculated by linear interpolation. The interpolated angle for a rotational$_1$ DOF is calculated by interpolating along the shortest arc on a circle, see Algorithm 1.1.

The interpolation between two quaternions is performed by spherical linear interpolation (SLERP), see e.g. [92] for implementation issues.

Algorithm 1.1 INTERPOLATEROTATION(a,b,f)

```
1: if $|a - b| < \pi$ then $angle \leftarrow \textsc{InterpolateLinear}(a, b, f)$
2: else if $(a < b)$ then $angle \leftarrow [a - f * (a + 2\pi - b)] \bmod 2\pi$
3: else $angle \leftarrow [a + f * (b + 2\pi - a)] \bmod 2\pi$
4: return $angle$
```

1.3 Thesis outline

This thesis has been divided into two parts. The first part deals with comparing and analyzing sampling-based motion planning techniques, in particular variants of the Probabilistic Roadmap Method (PRM). Over the past fifteen years, the PRM has been studied by many different researchers. This has led to many variants of the approach, each with its own merits. It is difficult to compare the different techniques because they were tested on different types of environments, using different underlying libraries, implemented by different people on different machines.

In Chapter 2, we provide a comparative study of a number of these techniques, all implemented in a single system and run on the same test scenes and on the same computer. In particular we compare the running times of collision checking techniques, neighbor selection techniques and sampling techniques. The results are surprising in the sense that techniques often perform differently than claimed by the designers.

In Chapter 3, we give a reachability-based analysis of these techniques. This analysis compares them based on coverage and connectivity of the free space. The experiments show, contrary to general belief, that the main challenge is not getting the free space covered but getting the nodes connected, especially when the problems get more complicated, e.g. when a narrow passage is present. By using this knowledge, we can tackle the narrow passage problem by incorporating a more powerful local planner, a refined neighbor selection strategy and a hybrid sampling strategy. The analysis also shows why the PRM successfully deals with many motion planning problems.

The second part deals with quality aspects of paths and roadmaps. Many algorithms have been proposed that create a path for a robot in an environment with obstacles. Since it can be hard to create such a path, they are only aimed at finding *a* solution. However, for most applications it is also critical that the path is of good quality with regard to the amount of clearance along the path and the length of the path.

In Chapter 4, we study two algorithms that increase the clearance along paths. The first one is fast but it can only deal with rigid, translating bodies. The second one is slower but it can handle a broader range of robots which can reside in arbitrary high-dimensional configuration spaces. Examples include free-flying and articulated robots. A big advantage of these algorithms is that clearance along paths can now be efficiently increased without using complex data structures and algorithms.

Chapter 5 studies algorithms that decrease the path length. We observe that existing algorithms can have difficulties in removing redundant (rotational) motions of the robot. We propose a new algorithm that successfully deals with these difficulties.

In Chapter 6, we introduce the Reachability Roadmap Method which creates small roadmaps for two- and three-dimensional problems. Such a small roadmap assures low query times and low memory consumption and is easy to optimize in a post processing phase. The algorithm ensures that a path is always found (if one exists) at a given resolution.

Chapter 7 unifies the techniques from previous chapters to create high-quality roadmaps for interactive virtual environments. That is, we use the Reachability Roadmap Method to create an initial roadmap. We add useful cycles to provide alternative routes and short paths, and we add clearance to the roadmap to obtain high-clearance paths in real-time.

Finally, we make some concluding remarks in Chapter 8.

PART I

SAMPLING-BASED MOTION PLANNING

CHAPTER
TWO

Performance-based comparison

The Probabilistic Roadmap Method (PRM) is one of the leading motion planning techniques. Over the past fifteen years, the technique has been studied by many different researchers. This has led to many variants of the method, each with its own merits. It is difficult to compare the different techniques because they were tested on different types of environments, using different underlying libraries, implemented by different people on different machines.

We provide a comparative study of a number of these techniques, all implemented in a single system and run on the same test environments and on the same computer. In particular we compare *collision checking* techniques, *neighbor selection* techniques and *sampling* techniques. The results are surprising in the sense that techniques often perform differently than claimed by the designers.

The study also shows how difficult it is to evaluate the quality of the techniques. First, it is not necessarily true that combining different 'good' techniques leads to a good overall result. Second, many techniques have high variances in running time which is undesirable as a large variation complicates statistical analysis and can even make it unreliable. It is also undesirable from a users point of view, e.g. in a virtual environment where real-time behavior is required, only a particular amount of CPU time is scheduled for the motion planner. The variance can be reduced by properly choosing the techniques and their parameters.

The results of this study should help future users of the Probabilistic Roadmap Method in deciding which technique is suitable for their situation.

2.1 Introduction

Over the years, many different approaches to solving the motion planning problem have been suggested. See the books of Choset *et al.* [36], Latombe [99] and LaValle [100] for an extensive overview. A popular motion planning technique is the Probabilistic Roadmap Method (PRM), developed independently at different sites [4, 6, 80, 81, 83, 84, 124, 151]. The method turns out to be very efficient, easy to implement, and applicable to many different types of motion planning problems (see e.g. [16, 20, 26, 40, 66, 68, 77, 85, 97, 107, 117, 130, 144, 151–153]).

Globally speaking, the PRM samples the configuration space for collision-free configurations. These are added as nodes to a roadmap graph. Pairs of promising nodes are chosen in the graph and a simple local motion planner is used to try to connect such placements with a path. If successful, an edge is added between the nodes in the graph. This process continues until the graph represents the connectedness of the space.

The basic PRM leaves many details to be filled in, like how to sample the space, what local planner to use and how to select promising pairs. Over the past decade, researchers have investigated these aspects and developed many improvements over the basic scheme (see e.g. [3, 15, 20, 21, 23, 25, 69, 70, 85, 94, 121, 130, 131, 152, 156]). Unfortunately, the different improvements suggested are difficult to compare. Each author uses his or her own implementation of the PRM and uses different test scenes, both in terms of environment and the type of moving device (robot). Also the effectiveness of a technique sometimes depends on choices made for other parts of the method. Therefore, it is still rather unclear what is the best technique under which circumstances. See [42] for a first study of this issue.

In this chapter, we will compare different techniques developed. We implemented many different techniques in a single motion planning system and added software to compare the approaches. In particular, we concentrated on approaches checking local paths for collisions, the choice of promising pairs of nodes and the sampling technique. This comparison gives insight into the relative merits of the techniques and the applicability in certain types of motion planning problems. In addition, we hope that in the longer term our results will lead to improved (combinations of) techniques and adaptive approaches that choose techniques based on observed scene properties.

The chapter is organized as follows. In Section 2.2, we review the basic PRM. In Section 2.3, we describe our experimental setup and the environments we use. In Section 2.4, we compare different ways of performing collision checks of the paths produced by the local planner. We will conclude that a binary approach performs best. In Section 2.5, we study different strategies for choosing promising pairs of nodes to connect. We will conclude that a technique based on connecting a new configuration to the nearest-*k* configurations works relatively well. In Section 2.6, we consider five different uniform sampling strategies. We conclude that, except for very special cases, a deterministic approach based on Halton points can be best used, although the differences between the methods are small. In Section 2.7, we consider six non-uniform sampling techniques that have been designed to deal with the so-called *narrow passage* problem. We will conclude that these techniques should only be used in parts of the workspace containing narrow passages, i.e. they handle the test environment with one very narrow passage faster than uniform sampling, but are often much slower on all other environments. Finally, we draw conclusions in Section 2.8.

2.2 The Probabilistic Roadmap Method

The motion planning problem is usually formulated in terms of the *configuration space* $\mathcal{C}$, the space of all possible placements of the moving object. Each degree of freedom (DOF) of the object corresponds to a dimension of the configuration space. Each obstacle in the workspace, in which the object moves, transforms into an obstacle in the configuration space. Together they form the forbidden part $\mathcal{C}_{\text{forb}}$ of the configuration space. A path for the moving object corresponds to a curve in the configuration space, connecting the start and the goal configuration. A path is collision-free if the corresponding curve does not intersect $\mathcal{C}_{\text{forb}}$, that is, it lies completely in the free part of the configuration space, denoted by $\mathcal{C}_{\text{free}}$.

The PRM samples the configuration space for free configurations and tries to connect these configurations to a roadmap of feasible motions. There are several versions of the PRM, but they all use the same underlying concepts.

The global idea of the PRM is to pick a collection of (useful) configurations in the free space $\mathcal{C}_{\text{free}}$. These free configurations form the nodes of a graph $G = (V, E)$. A number of (useful) pairs of nodes are chosen and a simple local motion planner is used to try to connect these configurations by a path. When the local planner succeeds, an edge is added to the graph. The local planner must be fast (since checking local paths for collisions is the most time-consuming part of the PRM), but is allowed to fail on difficult instances. A typical choice is to interpolate between the two configurations, and then check whether the path is collision-free. So the path is a straight line in $\mathcal{C}$-space.

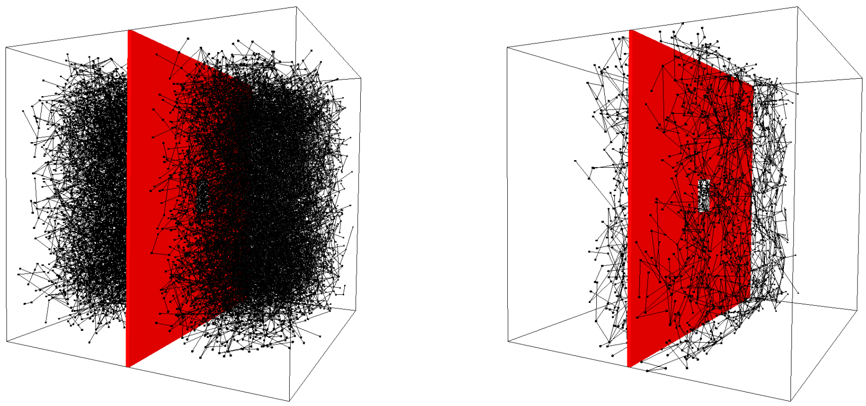


Figure 2.1 The roadmap graph we get for the difficult Hole test environment used in this chapter. The left image shows the graph using Halton sampling (20,002 nodes and 20,019 edges) and the right image shows the graph using Gaussian sampling (2,428 nodes and 2,425 edges).

Once the graph reflects the connectivity of $\mathcal{C}_{\text{free}}$ it can be used to answer motion planning queries. See Figure 2.1 for an example of the roadmap graphs computed. To find a motion between a start configuration and a goal configuration, both are added to the graph using the local planner. (Some authors use more complicated techniques to connect the start and goal to the graph, e.g. bouncing motions [124].) Then a path in the graph is found which corresponds to a motion for the object. The pseudo-code for the algorithm for constructing the graph is shown in Algorithm 2.1.

Algorithm 2.1 CONSTRUCTROADMAP

```
Require: $V \leftarrow \emptyset$; $E \leftarrow \emptyset$;
1: loop
2:    $c \leftarrow$ a (useful) configuration in $\mathcal{C}_{\text{free}}$
3:    $V \leftarrow V \cup \{c\}$
4:    $N_c \leftarrow$ a set of (useful) nodes chosen from $V$
5:    for all $c' \in N_c$, in order of increasing distance from $c$ do
6:       if $c'$ and $c$ are not connected in $G$ then
7:          if the local planner finds a path between $c'$ and $c$ then
8:             add the edge $c'c$ to $E$
```

Note that in this version of the PRM, we only add an edge between nodes if they are not in the same connected component of the roadmap graph. This saves time because such a new edge will not help solving motion planning queries. Hence, we will not add these additional edges in this comparative study. However, to get short paths, such extra edges can be useful (see Section 7.2).

In this study, we concentrate on various choices for picking useful samples (line 2 of the algorithm), for picking useful pairs of nodes for adding edges (that is, on the choice of N_c in line 4) and for collision checking those edges (line 7). These are the most crucial steps and they strongly influence the running time and the structure of the roadmap graph.

In this study, we focus on multiple shot techniques and will not consider single shot methods such as RRT-based planners [94,130].

2.3 Experimental setup

In this study we restrict ourselves to free-flying objects in a three-dimensional workspace. Such objects have six degrees of freedom (three translational and three rotational). In all experiments (except for the *rotate-at-s* local planner in the next section), we use the most simple local method that consists of a straight-line motion in configuration space. For other types of devices or local planners the results might be different. This will be investigated in Chapter 3.

The PRM builds a roadmap, which, in the query phase, is used to solve motion planning problems. We aim at computing a roadmap that covers the free space adequately but this is difficult to test.[1] Instead, in each test environment we define a relevant query and continue building the roadmap until the query configurations are in the same connected component.

For the experiments we use our SAMPLE system. In all experiments we report the running time in seconds. Because the experiments are conducted under the same circumstances, the running time is a good indication of the efficiency of the technique. For those techniques where random choices are involved, we report statistics gathered from 100 independent runs. See Section 1.2 for more information on our general experimental setup.

For the experiments we use the six environments depicted in Figure 2.2, all representing different types of problems. The Cage and Wrench environments

[1]Actually, this is too expensive to test for problems involving more than three degrees of freedom. In Chapter 3, we use a termination criterion that is based on coverage (and connectivity) of the free configuration space.

have been taken from the Motion Strategy Library [100]. To make extensive experimentation possible, we do not include huge environments such as those common in CAD environments. The environments have the following properties (see Table 2.1 for their dimensions).

Cage This environment consists of many primitives. The flamingo (7,049 polygons) must navigate from inside the cage (1,032 triangles) to outside the cage. The complexity of this environment will put a heavy load on the collision checker but the paths are relatively easy.

Clutter This environment consists of 500 uniformly distributed tetrahedra. A torus must move among them from one corner to the other. The configuration space will consist of many corridors. There are many solutions to the query.

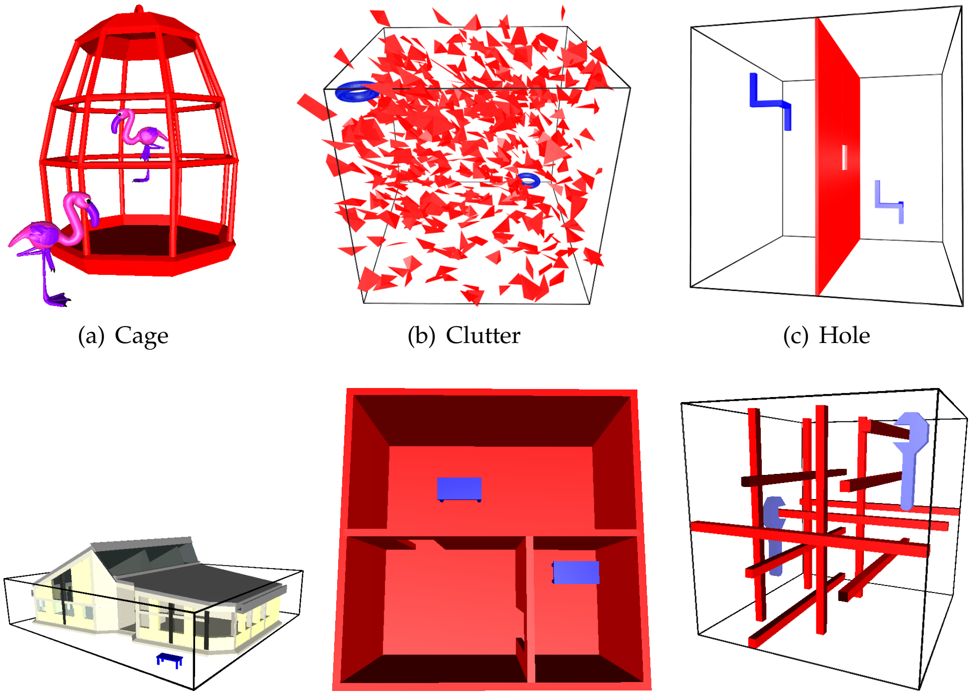
Hole The moving object consists of four legs and must rotate in a complicated way to get through the hole. The hole is placed off-center to avoid that certain grid-based sampling methods have an advantage. The configuration space will have two large open areas with two narrow winding passages between them.

House The house is a relatively complicated environment consisting of approximately 2,200 polygons. The moving object (table) is small compared to the house. Because the walls are thin, the collision checker must make rather small steps along the paths, resulting in higher collision checking times. Because of the many different parts in the environment, the planner can be lucky or unlucky in finding relevant parts of the roadmap quickly. Therefore, we expect large differences in the running times of different runs.

Rooms In this environment there are three rooms with ample free space and with narrow doors between them. So the density of obstacles is rather non-uniform. The table must navigate through the two narrow doors to the other room.

Wrench This environment features a large moving object (156 triangles) in a small workspace (48 triangles). There are many different solutions. The object is rather constrained at the start and goal.

We use the distance metric described in Section 1.2.3. Table 2.2 enumerates the weights for the translational and rotational DOFs. The step sizes for the local planner are listed in Table 2.3.



(a) Cage (b) Clutter (c) Hole

(d) House (e) Rooms (f) Wrench

Figure 2.2 The six environments used for testing.

	Dimensions of the bounding boxes	
	environment	**robot**
Cage	$45 \times 40 \times 45$	$4 \times 13 \times 10$
Clutter	$48 \times 48 \times 48$	$8 \times 1.5 \times 8$
Hole$^{(*)}$	$40 \times 40 \times 40$	$5 \times 5 \times 10$
House	$35 \times 8 \times 38$	$2.5 \times 0.5 \times 1.5$
Rooms	$40 \times 20 \times 40$	$8 \times 2.5 \times 4$
Wrench	$160 \times 160 \times 160$	$68 \times 24 \times 8$

Table 2.1 The axis-aligned bounding boxes of the environments and robots. $^{(*)}$The dimensions of the hole are $5 \times 5 \times 0.5$. The legs of the robot are 1 thick.

	Weights for the DOFs of a robot	
	translational	**rotational**$_3$
Cage	1, 1, 1	30
Clutter	1, 1, 1	7
Hole	1, 1, 1	11
House	1, 1, 1	4
Rooms	1, 1, 1	9
Wrench	6, 6, 6	30

Table 2.2 The weights for each DOF of the robots.

	step size
Cage	1.0
Clutter	1.0
Hole	1.0
House	0.5
Rooms	1.0
Wrench	3.0

Table 2.3 The step sizes used by the local planners.

2.4 Collision checking

The most time-consuming steps in the Probabilistic Roadmap Method are the collision checks that are required to decide whether a sample lies in $\mathcal{C}_{\text{free}}$ and whether the motion produced by the local planner is collision-free. In particular the second type of checks is time-consuming. In this section we investigate some techniques for collision testing of the paths.

When testing a path for collisions we can use one of the following three techniques:

incremental In the incremental method we take small steps along the path from start to goal. Let d be the distance between the start and goal, then the number of steps equals to d divided by the appropriate step size from Table 2.3. We check each step (i.e. the placement of the robot) for collisions with the environment.

binary In the binary method we start by checking the middle position along the path. If it is collision-free we recurse on both halves of the path, checking the middle positions there. In this way, we continue until either a collision is found or the checked placements lie close enough together (again determined by the given step size) [130].

rotate-at-s While the previous methods check collisions along a straight line in the $\mathcal{C}$-space, the rotate-at-s approach first translates from start to an intermediate configuration s halfway, then rotates, and finally translates to the goal [2]. We set s to 0.5, i.e. halfway the line.

The incremental method was used in early papers on the PRM. Later papers suggest that the binary method works better [131]. The reason is that the middle position is the one that has the highest chance of not being collision-free. This means that, when the path is not collision-free, a collision will most likely be found earlier, that is, after fewer collision checks.

It has been suggested that one should try to compute sweep volumes and use these for collision tests. As a result, a path check would require just one collision test. Unfortunately, it is very difficult to compute sweep volumes for three-dimensional moving objects with six degrees of freedom. A much simpler technique is to first check with the sweep volume induced by the origin of the object, that is, with a line segment between start and goal position, see [41].

preceding line check In this method we first perform a collision test with the line segment between the start and goal position in the workspace. Only

	Collision checking without line check		
	incremental	**binary**	**rotate-at-s**
Cage	2.4	**1.5**	4.2
Clutter	2.3	**1.9**	6.5
Hole	226.7	**213.6**	929.5
House	5.5	**5.1**	14.3
Rooms	**0.3**	**0.3**	0.9
Wrench	1.4	**1.1**	4.6

	Collision checking with line check		
	incremental	**binary**	**rotate-at-s**
Cage	2.4	**1.5**	3.9
Clutter	**2.4**	**2.4**	5.8
Hole	**186.2**	213.7	979.2
House	6.6	**6.3**	13.9
Rooms	**0.2**	**0.2**	1.0
Wrench	1.5	**1.1**	4.6

Table 2.4 Average running times for three different collision checking methods. Each method was tested without and with line check.

if it is collision-free, we perform one of the three techniques. Note that this assumes that the origin of the object lies inside it.

We would expect that this test will quickly discard many paths that have a collision, leading to an improvement in running time.

We conducted experiments with the three collision-check techniques. Each experiment was run 100 times. To determine whether first performing a collision test with an appropriate line segment decreases the running time, we tested each technique with and without line check. Figure 2.3 and Table 2.4 summarize the results (using Halton* points for sampling and a simple nearest-k node adding strategy[2]; see below). Note that the best times are indicated in bold.

They show that in all environments the binary approach was faster than the incremental approach although the improvement varied over the type of environment. Also the standard deviation of this approach was smaller, making it

[2]The values used for the two parameters of this node adding strategy are stated in Table 2.5.

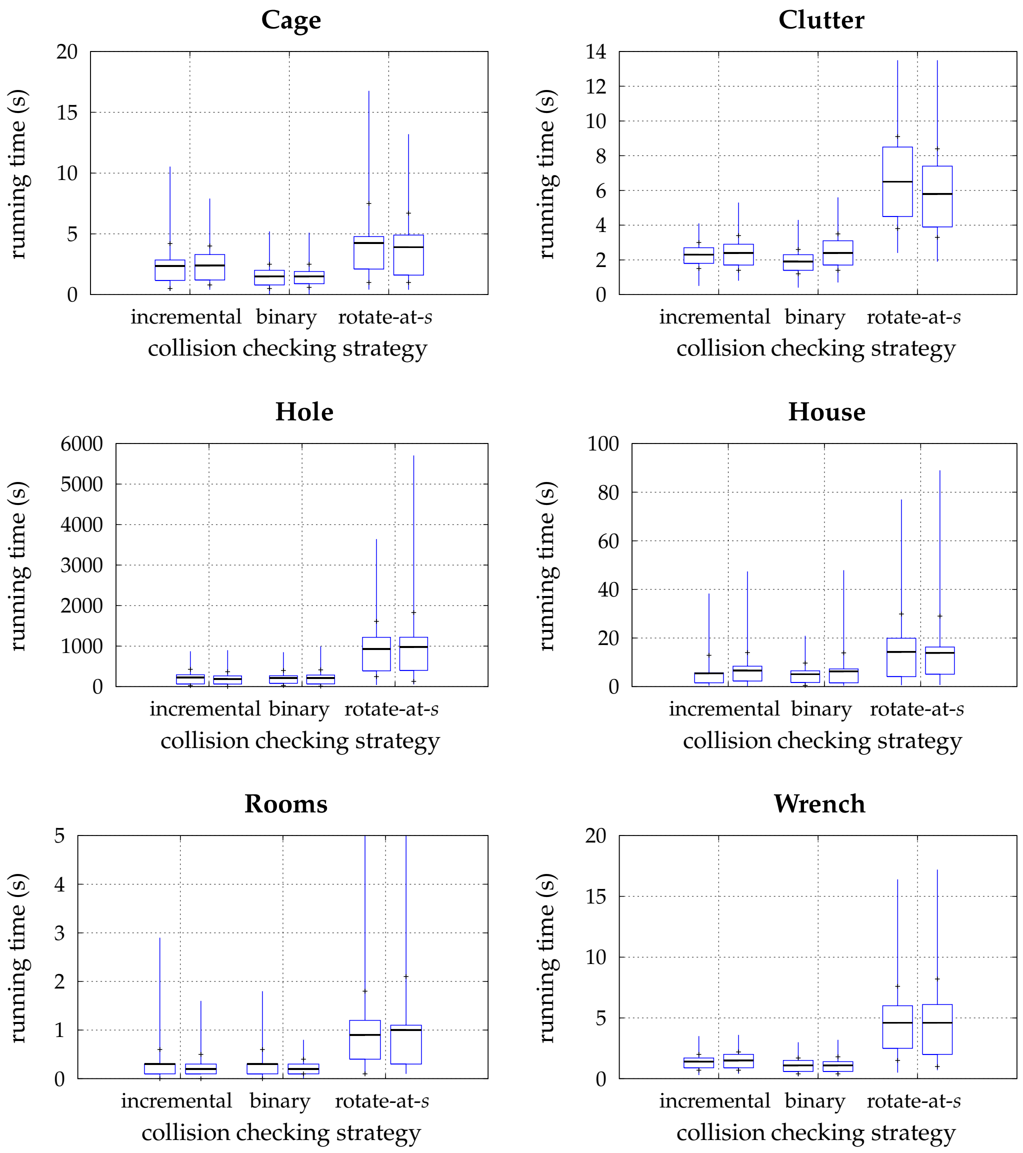


Figure 2.3 Relation between the *collision checking strategy* and the running time. Box plots are shown for each environment. Each strategy was tested without (left box plots) and with a line check (right box plots). Each plot consists of a box (the middle 50% of the data), one large horizontal line (average) and two small horizontal lines (average $\pm$ standard deviation) and a vertical line (minimum and maximum value).

more robust against outliers. The line check only had a marginal effect, contrary to the claim in [41]. This might be due to the way Solid does the collision checking with line segments. In [41], it was suggested to only apply the line check when the distance between the endpoints is large. We tried this but did not see any significant improvement in performance.

Also the rotate-at-s technique did not give the improvement suggested in [2]. Actually, its average running times were three times as large as the running times of the binary approach. Also its standard deviation was much larger. However, it should be noted that this can depend on the underlying collision checking package used. For the rest of this chapter we will use the binary approach without line checks.

2.5 Neighbor selection strategy

The neighbor selection strategy specifies for a particular sample how a set of neighbor samples is chosen to which it is connected using a local planner. The goal of the strategy is to make the graph connected as fast as possible. A strategy usually selects neighbors based on the *maximum connection distance*, *maximum number of connections* tried, and the *connected components* in the graph.[3] We study the effects of different choices for these criteria.

It is recognized that the maximum connection distance should not be too small nor too large: Although making long connections seems to be important for the PRM, it is in general not useful to make very long connections since the chance of success for such connections is small while the collision checks required for testing the local path are expensive. On the other hand, a very small connection distance will always require an exponential number of samples. We will show how to choose the maximum connection distance in Section 2.5.1.

In Section 2.5.2, we will study how to choose the maximum number of connections. If this number is too small, then it might be hard to get the free space connected because the chance is small that those few nodes are selected to which a connection is possible. If connections are tried with too many nodes, the total number of nodes will be less, but this might negatively influence the running time as testing those connections is expensive.

[3]If these parameters are not set to infinity (which is the default case), then we have to find the nearest-k neighbors. This is done by calculating the distance to each of the n nodes in the graph. A node is stored in a sorted list of maximum size k if the distance to this node is less than some maximum distance. This approach takes $O(n \log k)$ time. More efficient nearest-neighbor searching algorithms based on kd-trees are discussed in [158].

In Section 2.5.3, we will compare five node adding strategies. These strategies do not produce graphs with cycles. It is not useful (from a complexity point of view) to make connections to nodes that are already in the same connected component as such a new connection will not help solving motion planning queries.

2.5.1 Maximum connection distance

Before we can conduct experiments to determine the maximum connection distances, we have to make some choices for the PRM. We use the nearest-k node adding strategy. We set parameter k (which denotes the maximum number of connections) to 25 as this seems to work reasonably. Furthermore, we use the Halton* sampling strategy (see below).

In each of the following experiments, we vary the maximum connection distance from a small value (close to zero) to a large value. For each distance value, we perform 100 runs and gather the following running time statistics: the middle 50% of the data, the average, the standard deviation, the minimum and the maximum. See Figure 2.4 for the results.

When we make the connection distance very small, the PRM starts looking like grid-based techniques in which nodes are only connected to their direct neighbors. Figure 2.4 shows that this considerably increases the running time. Indeed, the power of PRM is that it can make longer connections. A relatively small value is only useful in environments were the connections are expected to be short. When we make the connection distance large (or even set this parameter to ∞), the average running times are in general higher than the running time corresponding to the optimal maximum connection distance. A larger value is best for environments in which long connections can be made and where the weights, used by the metric, are large. (For example, as the rotational DOFs play an important role in the Cage and Wrench environments, the corresponding weights for these DOFs are large.) Hence, the optimal value is correlated with the average visibility of the nodes and the weights used by the metric. Moreover, there is some optimal trade-off. See Table 2.5 for the optimal values.[4] These values will be used in the remainder of the chapter.

The box plots of Figure 2.4 reveal a problem when dealing with statistical analysis of data produced by the PRM. They show that there can be a large difference between the minimum and maximum running times. Furthermore, the large sizes of the boxes show that there is a large variance in the running

[4]For other types of problems, such as problems involving car-like or articulated robots, we also advise not to use a small maximum connection distance.

	Neighbor selection parameters	
	max. connection distance	**max. number of connections**
Cage	45	75
Clutter	15	75
Hole	10	75
House	15	75
Rooms	20	75
Wrench	350	75

Table 2.5 The optimal values used in the neighbor selection strategy.

times. This phenomenon is undesirable because of two reasons. First, a large variation complicates statistical analysis and can even make it unreliable. Second, it is undesirable from a users point of view, e.g. it can be hard to give a user an indication of how long the method will take to terminate. Hence, we have to be very careful analysing the results. As the running times can vary extensively, we performed 100 runs for each experiment. In this way we can increase its statistical significance. We noticed that some authors choose a low number of runs per experiment while only mentioning the average. Because of the large variance between runs that we observe, we think that this reduces the validity of the claims that are made.

2.5.2 Maximum number of connections

In the following experiments, we will vary the maximum number of connections k to determine the optimal value for each environment. For each k, we perform 100 runs and create a corresponding box plot. As maximum connection distance, we use the results of our experiments of the previous section as shown in Table 2.5. We again use the Halton* sampling strategy (see below). Figure 2.5 shows the relation between the maximum number of connections and the running time and Figure 2.6 shows the effect on the number of nodes.

The results show that a small maximum number of connections (e.g. $k < 5$) leads to high running times and high peaks. Corresponding graphs contain a huge number of nodes; they are typically one order of magnitude larger than graphs corresponding to the optimal value of k. As only a few connections per new node are tried, chances are small that the node will get connected to other nodes.

It is not useful to us a k larger than the (expected) number of nodes in the graph. In the Cage experiment for example, we observe that the average num-

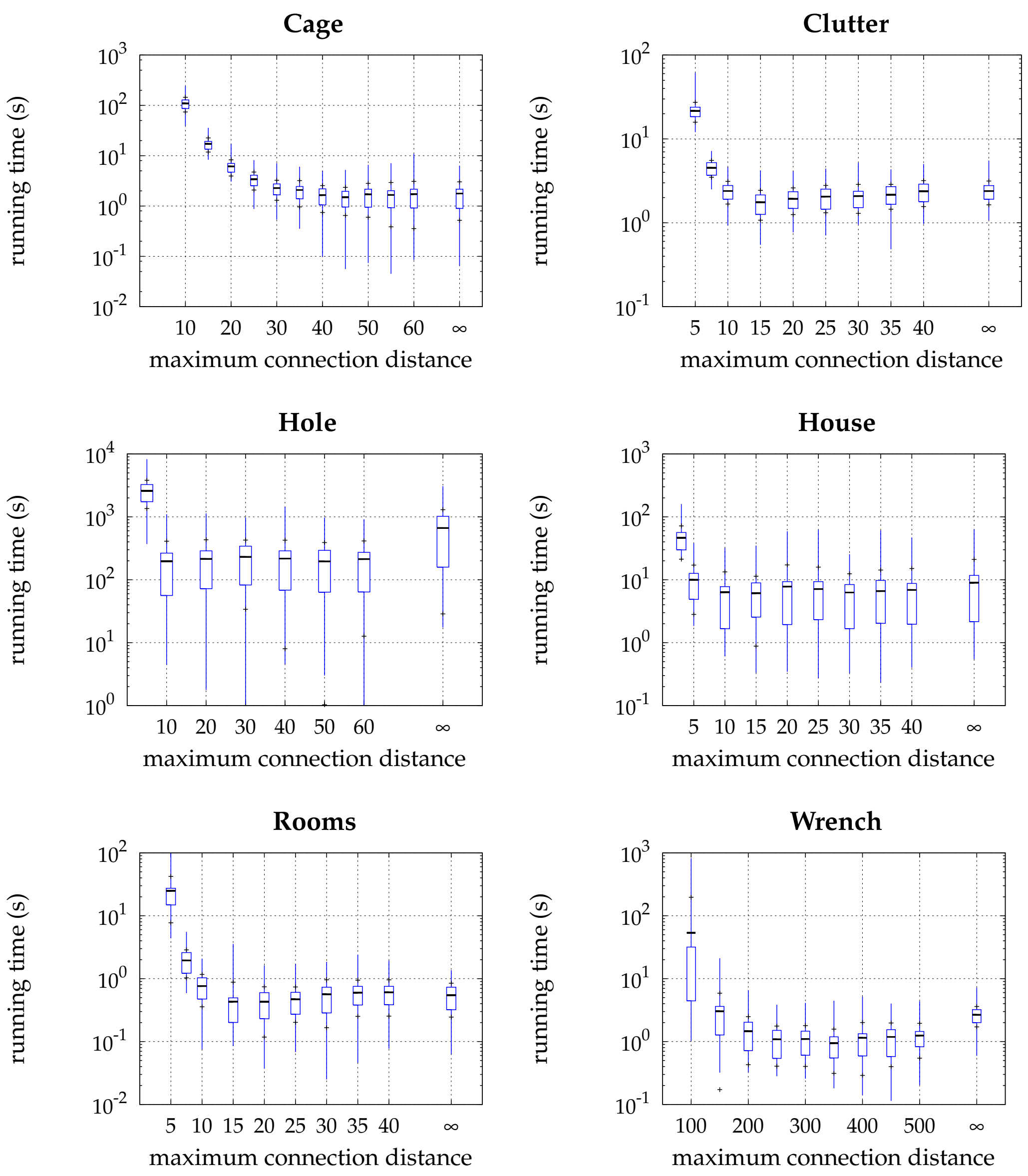


Figure 2.4 Relation between the *maximum connection distance* and the running time. A series of box plots is shown for each environment. Note that we use a logarithmic scale for the ‘running time’-axis.

ber of nodes in the graph is about 40 when k is set to at least 20. Using a value larger than 40 will not lead to extra collision checks or distance calculations. Hence, this will not influence the running times. Against our expectations, a high number of maximum connections had only a marginal effect on the running times (compared to the optimal running times). This can be explained as follows.

In order to find a path, at least the following criterion must hold: each configuration on the path must be in at least one visibility area of a sample/node in the graph. In other words, a set of nodes should cover the path. However, when these nodes belong to different connected components, the path cannot be found (yet). In the next chapter we will show that connecting the different connected components is more complicated than satisfying the coverage criterion. This especially holds for narrow passage problems which arise e.g. in the Hole environment, but also in the House and Rooms environments. When more effort is put in connecting the components, i.e. the maximum number of connections is increased (up to a certain level), the problem is solved in less time.

When a new node ν is connected to more nodes (up to a certain level), the chance is larger that different connected components will be merged, but this may take more time for collision checking. Note that the amount of extra coverage of ν is not affected by the maximum number of connections. (After adding some n nodes to the graph, the path will be covered.) On the other hand, when ν is connected to less nodes, the chance is smaller that different connected components will be merged, and hence, more than n nodes are needed. Although these extra nodes do not contribute to the coverage anymore, their only goal is to connect the components. As a smaller k complicates this task, too much time may be spent on checking the connections for collisions while the node may be less useful. We conclude that the extra time needed to check more connections saves computation time compared to the time needed for adding a larger number of useless nodes.

We advise to use a high value (e.g. 75) for the maximum number of connections.

2.5.3 Node adding strategies

We will now compare some node adding strategies. We consider the following techniques:

nearest-*k* We try to connect the new configuration to the nearest k nodes in the graph that lie close enough. The rationale is that nearby nodes result in

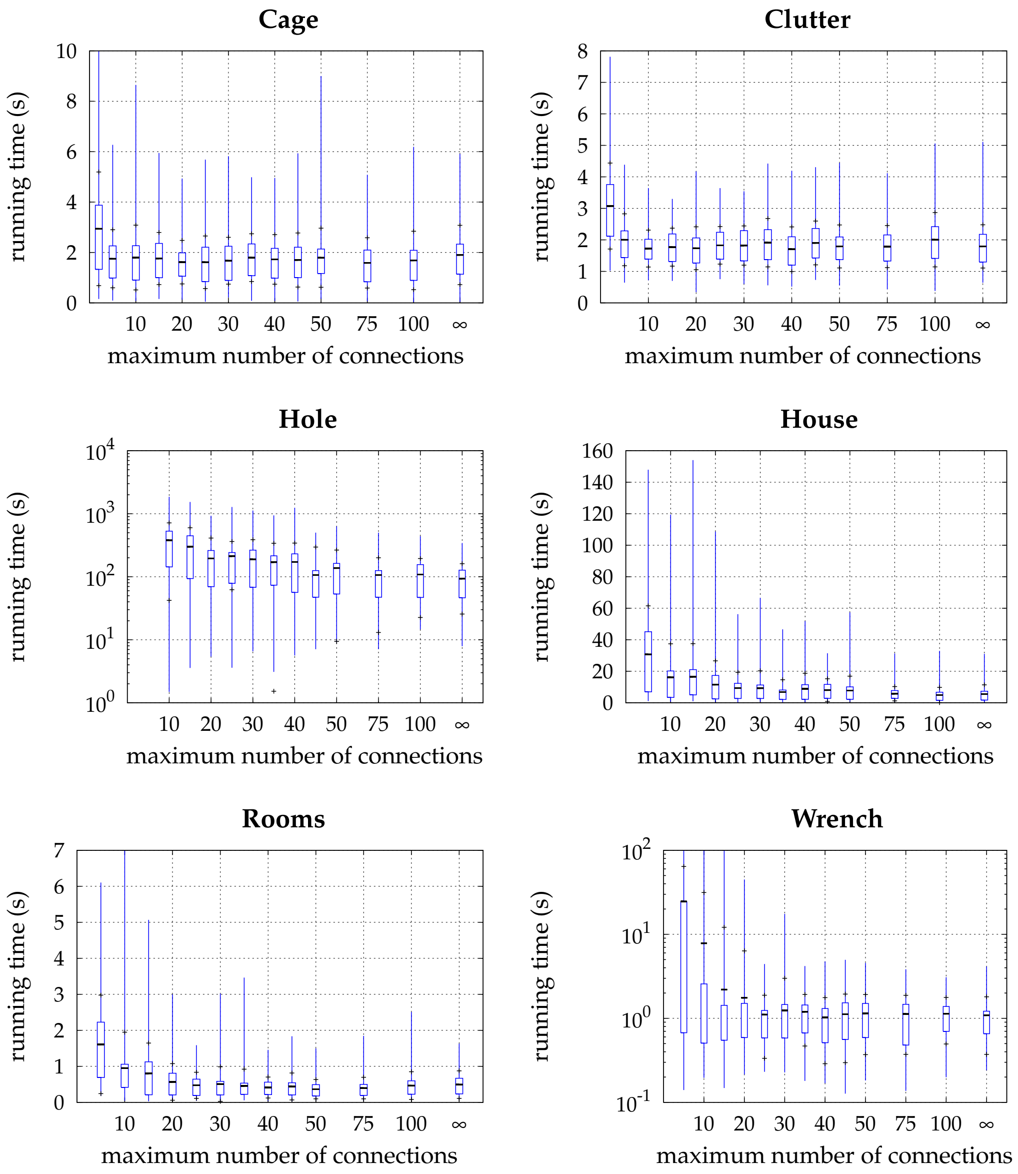


Figure 2.5 Relation between the *maximum number of connections* and the running time. A series of box plots is shown for each environment.

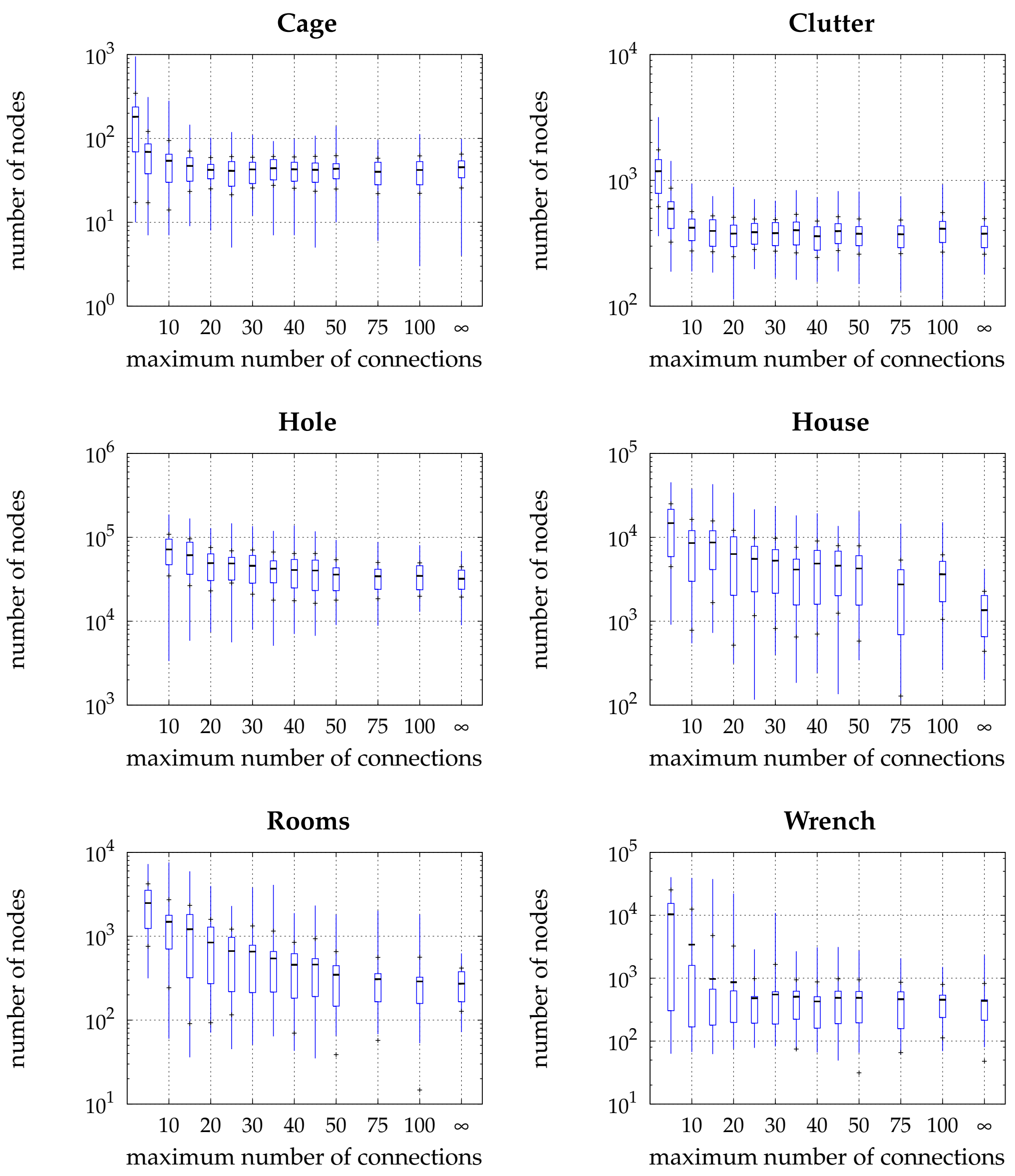


Figure 2.6 Relation between the *maximum number of connections* and the number of nodes. A series of box plots is shown for each environment. Note that we use a logarithmic scale for the 'number of nodes'-axis.

short connections that can be efficiently checked for collisions.

component We try to connect the new configuration to the nearest node in each connected component that lies close enough. The rationale is that we prefer to connect to multiple connected components.

component-*k* We try to connect the new configuration to at most *k* nodes in each connected component. Still, we keep the total number of connections tried small (the same number as for nearest-*k*). The rationale is that when the number of components is small we prefer to spend some extra time on trying to make connections. Otherwise the time required for adding the node will become the dominant factor. Preliminary experiments showed that setting *k* to 3 works fine in our environments.

visibility This method is based on the visibility sampling technique described in [121]. This technique only connects configurations to useful nodes. Usefulness is determined as follows: when a new node cannot be connected to other nodes it forms a new connected component and is labeled useful. If it connects two or more components it is also labeled useful. If it can be connected to just one component it is not labeled useful. It has been observed in [121] that the number of useful nodes remains small, making it possible to try connections to all of them. Hence, we do not restrain the connection distance and number of connections.

visibility-*k* This method is again based on the visibility sampling technique, but now only the nearest *k* nodes that lie close enough will be considered for the usefulness test.

Table 2.6 and Figure 2.7 summarize the results. Although the visibility approach pruned the graphs a lot, it still performed worse on most environments. Only for the Hole environment it performed better. We feel that the reason is that the approach is too strict in rejecting nodes. However, the method performed better when restrictions were put on the two parameters. In general, the nearest-*k* technique performed relatively well, except in the Hole environment. We must remark that for this environment it is better to use an obstacle-based sampling technique such as *nearest contact* which will be discussed in Section 2.7. This technique is two orders of magnitude faster than the visibility approach.

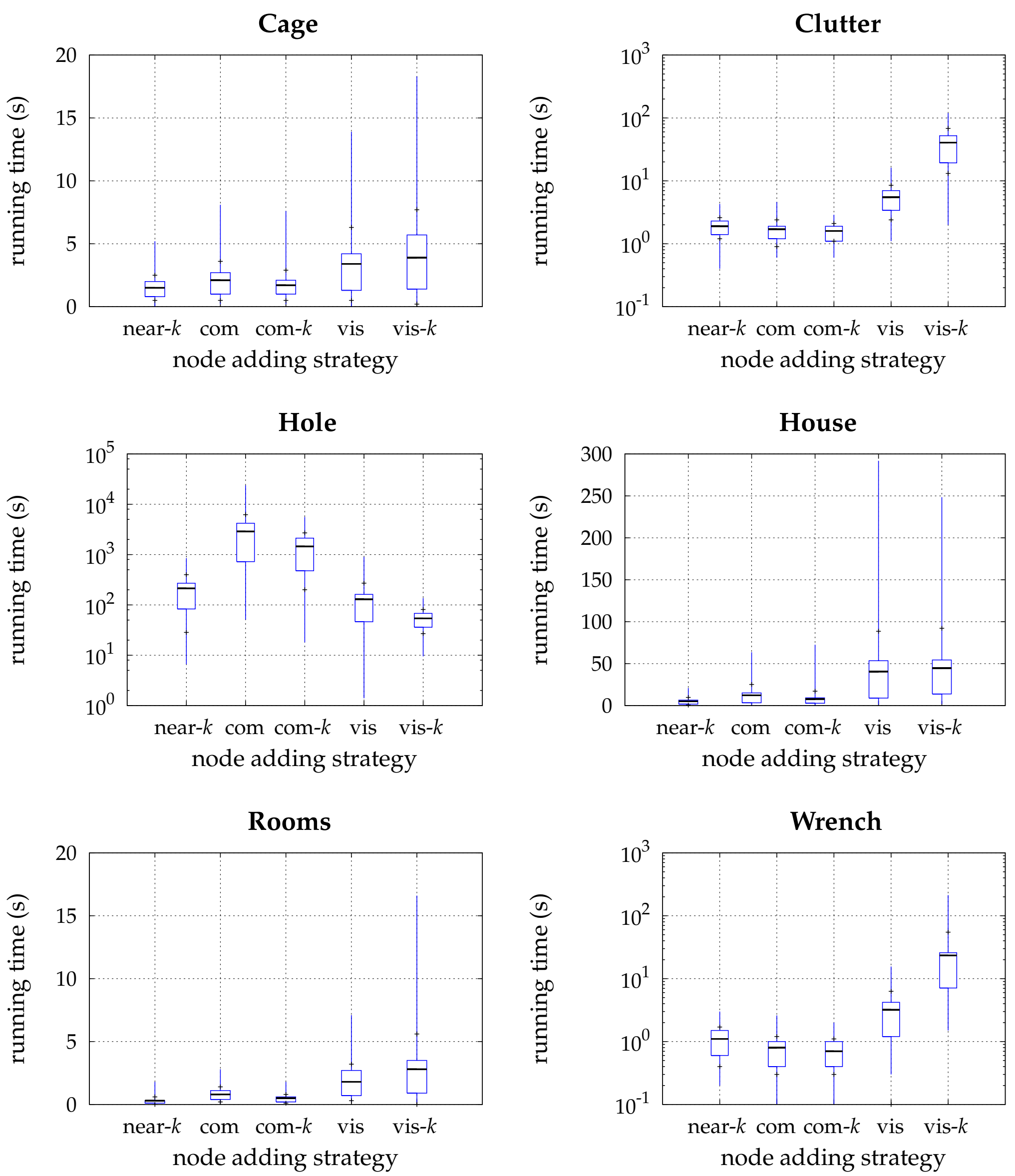


Figure 2.7 Relation between the *node adding strategy* and the running time. A series of box plots is shown for each environment. Note that we sometimes use a logarithmic scale on the 'running time'-axis.

	Node adding strategy				
	nearest-*k*	**comp**	**comp-*k***	**visibility**	**visibility-*k***
Cage	**1.5**	2.1	1.7	4.0	3.4
Clutter	1.9	1.7	**1.6**	40.0	5.5
Hole	213.6	2,886.5	1,451.9	129.4	**54.0**
House	**5.1**	12.2	7.8	44.5	40.5
Rooms	**0.3**	0.8	0.5	2.8	1.8
Wrench	1.1	0.8	**0.7**	23.5	3.2

Table 2.6 Average running times of five distinct node adding strategies.

2.6 Uniform sampling

The first papers on the PRM used uniform random sampling of the configuration space to select the nodes that are added to the graph. In recent years, other uniform sampling approaches have been suggested to remedy certain disadvantages of the random behavior. In particular, we study the following techniques (see Figure 2.8 for a visual impression of the corresponding sampling distributions):

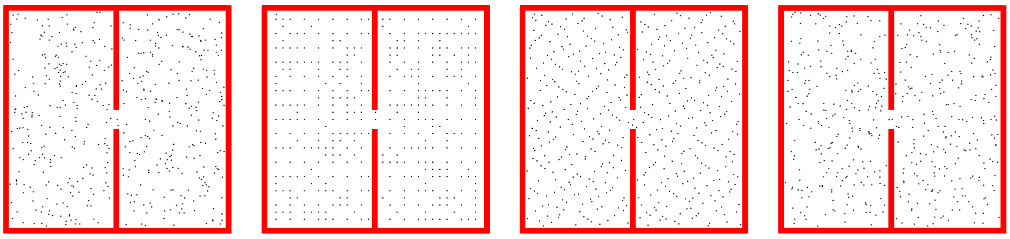
random In the random approach a sample is created by choosing random values for all degrees of freedom of the moving object.

grid In this approach we choose samples on a grid. Because the grid resolution is unknown in advance, we start with a coarse grid and refine this grid in the process, halving the cell size. Grid points on the same level of the hierarchy are added in random order.

Halton In [23] it has been suggested to use so-called Halton point sets as samples. Halton point sets have been used in discrepancy theory to obtain a coverage of a region that is better than using a grid (see e.g. [30]). It has been suggested in [23] that this deterministic method is well suited for the PRM.

Halton* In this variant of Halton we choose a random initial seed instead of setting the seed to 0 [154]. The claims in [23] should still hold because they are independent of the seed. By choosing a random seed we avoid the situation in which seed 0 is lucky or unlucky.

cell-based In this approach we take random configurations within cells of decreasing size in the workspace. The first sample is generated randomly



(a) random (b) grid (c) Halton (d) cell-based

Figure 2.8 Uniform sampling strategies in a 2D environment. Each image shows a typical distribution of 500 samples.

	Uniform sampling strategy				
	random	**grid**	**Halton**	**Halton***	**cell-based**
Cage	1.8	3.0	**1.3**	1.5	2.2
Clutter	1.7	3.7	**1.5**	1.9	1.6
Hole	237.9	**84.7**	101.5	213.6	232.4
House	20.6	14.5	**2.3**	5.1	21.2
Rooms	0.5	0.6	**0.2**	0.3	0.6
Wrench	1.5	1.4	**1.0**	1.1	1.4

Table 2.7 Average running times of five uniform sampling strategies.

in the whole space. Next we split the workspace in 2^3 equally sized cells. In a random order we generate a configuration in each cell. Next we split each cell into sub-cells and repeat this for each sub-cell. This should lead to a better distribution of the samples over the configuration space compared to random sampling. A similar approach was used in [129].

An important issue is how to choose random values. Random values were obtained by the Mersenne Twister [113]. Sampling for the rotational degrees of freedom was performed by choosing random unit quaternions [92], except for the Halton approach as this method is deterministic.[5] See [159] for a more extensive elaboration on sampling methods for these DOFs.

[5]We are aware that this choice might negatively influence the performance of this method. Contrary to the claim in [159], we experienced little difference in our experiments when we chose uniform random quaternions instead of Halton Euler angles which were converted to quaternions.

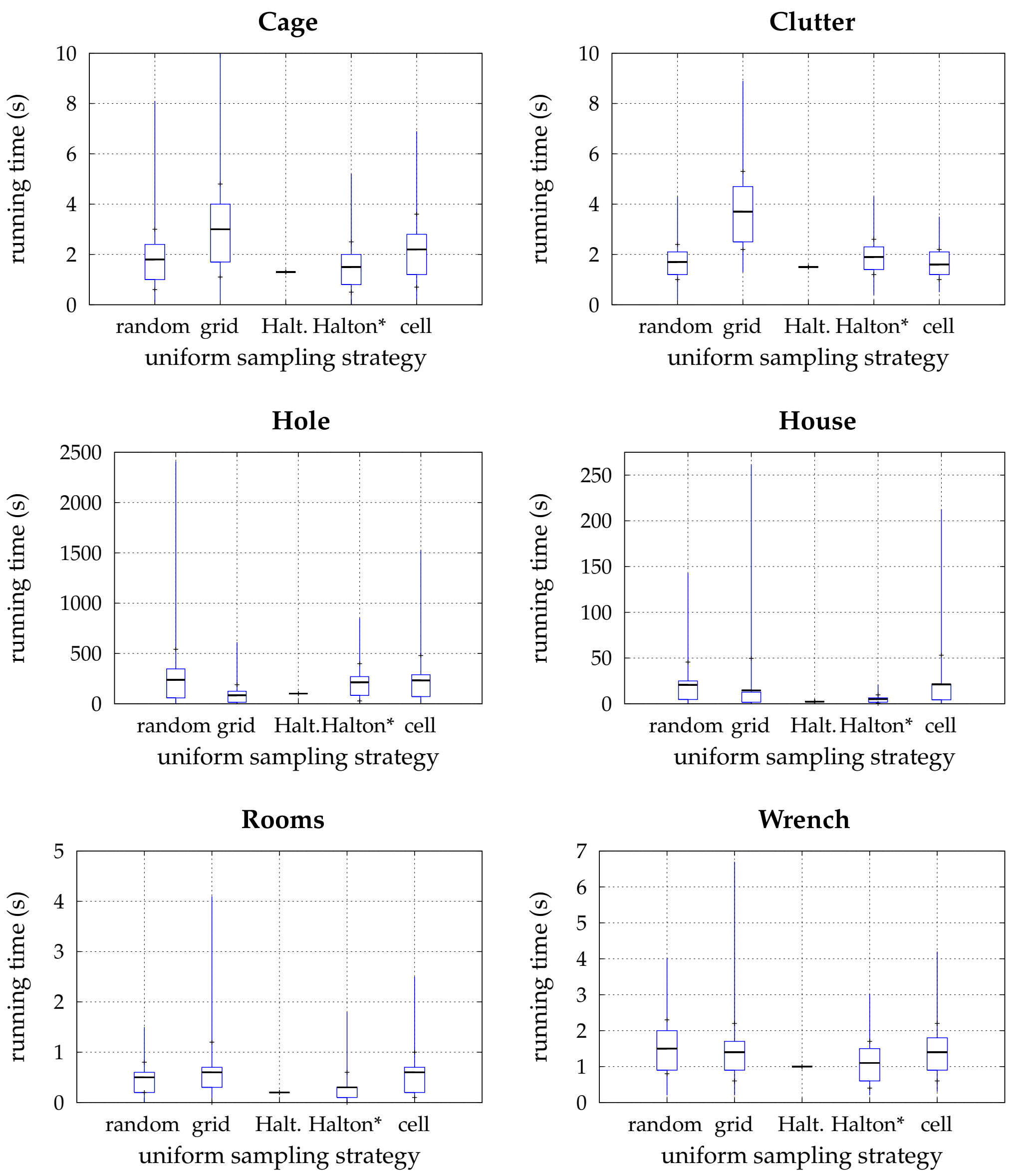


Figure 2.9 Relation between the *uniform sampling strategy* and the running time. A series of box plots is shown for each environment.

Table 2.7 and Figure 2.9 summarize the results. At first glance, it may appear that the deterministic Halton approach performed best. Especially the differences for the Hole and House environments were large. We tested whether this approach was lucky for these environments by translating the house and hole by a few units. Now the differences were negligible. Hence, the method was lucky for these environments. The running times being constant does not necessarily mean that this deterministic method is more consistent in performance. If the seed is randomized, which is the case for the Halton* method, its variance was comparable to e.g. the random approach.

The differences were in general small, i.e. the slowest method took about twice the time of the fastest method, except for the House environment. In general we must conclude that there is little to win when using different kinds of uniform sampling in terms of average running time. Nonetheless, there can be other arguments to use a particular technique. For example, the Halton method is deterministic. However, such a deterministic approach might be unlucky for a particular environment.

2.7 Non-uniform sampling

Rather than using uniform sampling, it has been suggested to add more samples in difficult regions of the environment. In this section we study a number of these techniques (see Figure 2.10 for their typical distributions):

Gaussian Gaussian sampling is intended to add more samples near obstacles. The idea is to take two random samples, where the distance σ between the samples is chosen according to a Gaussian distribution. Only if one of the samples lies in $\mathcal{C}_{\text{free}}$ and the other lies in $\mathcal{C}_{\text{forb}}$ we add the free sample. This leads to a favorable sample distribution [22]. We conducted preliminary experiments to find the optimal values for σ. We set σ to $\{2, 2, 1, 1, 2, 8\}$ for the six environments, respectively.

obstacle-based This technique, based on [3], has a similar goal. We pick a uniform random sample. If it lies in $\mathcal{C}_{\text{free}}$ we add it to the graph. Otherwise, we pick a random direction and move the sample in that direction with increasing steps until it becomes free and add the free sample. We set the initial step size to the corresponding step size from Table 2.3.

obstacle-based* This is a variation of the previous technique where we discard a sample if it initially lies in $\mathcal{C}_{\text{free}}$. This will avoid many samples in large open regions.

	Non-uniform sampling strategy						
	Gaussian	**obstacle**	**obstacle***	**bridge**	**MA**	**NC**	**Halton***
Cage	5.2	2.5	5.0	6.0	143.2	4.1	**1.5**
Clutter	2.8	2.9	4.2	6.3	411.8	4.5	**1.9**
Hole	4.5	27.2	3.0	38.4	111.5	**1.2**	213.6
House	7.1	6.0	5.1	9.9	433.4	**3.9**	5.1
Rooms	0.4	0.4	0.4	0.6	1.4	0.4	**0.3**
Wrench	2.3	1.9	3.0	7.7	17.9	4.0	**1.1**

Table 2.8 Comparison of average running times of six non-uniform sampling strategies.

bridge test The bridge test is a hybrid technique that aims at better coverage of the free space [69]. The idea is to take two random samples, where the distance σ between the samples is chosen according to a Gaussian distribution. Only if *both* samples lie in $\mathcal{C}_{\text{forb}}$ and the point in the middle of them lies in $\mathcal{C}_{\text{free}}$ the free sample is added. To also get points in open space, every sixth sample is chosen random. We set σ to $\{5, 5, 2, 1, 4, 5\}$ for the six environments, respectively.

medial axis This technique generates samples near the medial axis (MA) of the free space [156]. (See algorithm 4.1). All samples represent robot placements having at least two equidistant nearest points on the obstacles resulting in a large clearance from obstacles. The method is relatively expensive to compute since expensive closest pair calculations are involved.

nearest contact This method is based on [107] and generates samples on the boundary of the $\mathcal{C}$-space and can be considered as the opposite of the medial axis technique. First, we choose a uniform random sample c. If c lies in $\mathcal{C}_{\text{free}}$ we discard it, else our collision checker calculates the penetration vector v between c and the environment. Then, we move c in the opposite direction of v and place c on the boundary of the $\mathcal{C}$-space. Care must be taken not to place c exactly on the boundary, because then it would be difficult to make connections between the samples. Hence, we move c away from the boundary with the step size listed in Table 2.3.

Due to their biased distributions, we expect these techniques to be useful only in environments where there are ample free spaces (in the configuration space) and some narrow passages. Table 2.8 and Figure 2.11 show the results. (The results of the Halton* approach are stated for comparison.)

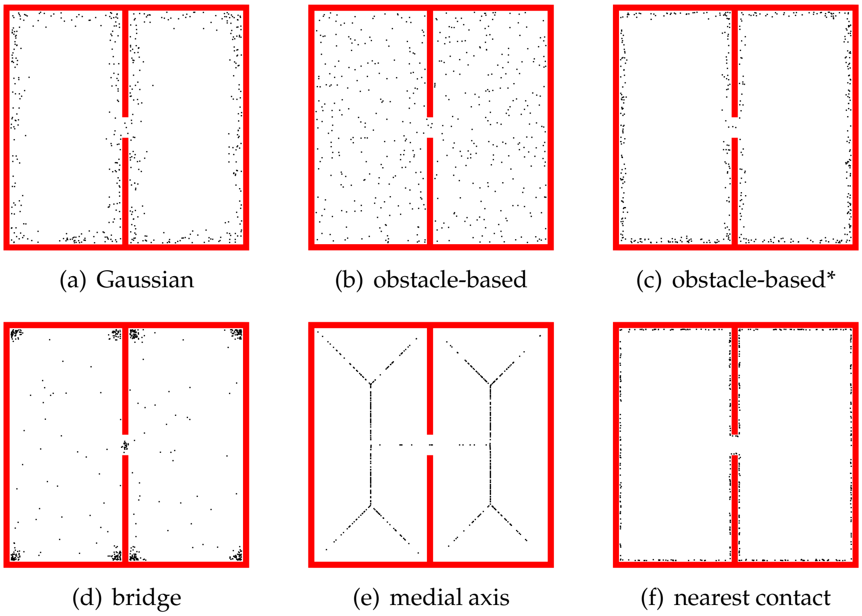


Figure 2.10 Non-uniform sampling strategies in a 2D environment. Each image shows a typical distribution of 500 samples.

As expected, the techniques only performed considerably better for the Hole environment. Also for the House environment, the methods works well but the improvement was not significant. However, in other situations, the methods were up to 10 times slower. The medial axis approach was even worse, due to the expensive calculations. This method does provide samples that are nicely located between the obstacles which results in motions with a higher clearance. (We will discuss clearance in Chapters 4, 6, and 7.) We conclude that special non-uniform techniques should only be used in specific situations with narrow corridors. Preferably, they should only be used in the parts of the workspace where this is relevant, see e.g. [73].

2.8 Discussion

In this chapter we presented the results of a comparative study of various PRM techniques. The results confirm previous claims that the binary approach for collision checking works well. In contrast, the results also show that many claims on efficiency of certain sampling approaches could not be verified, i.e.

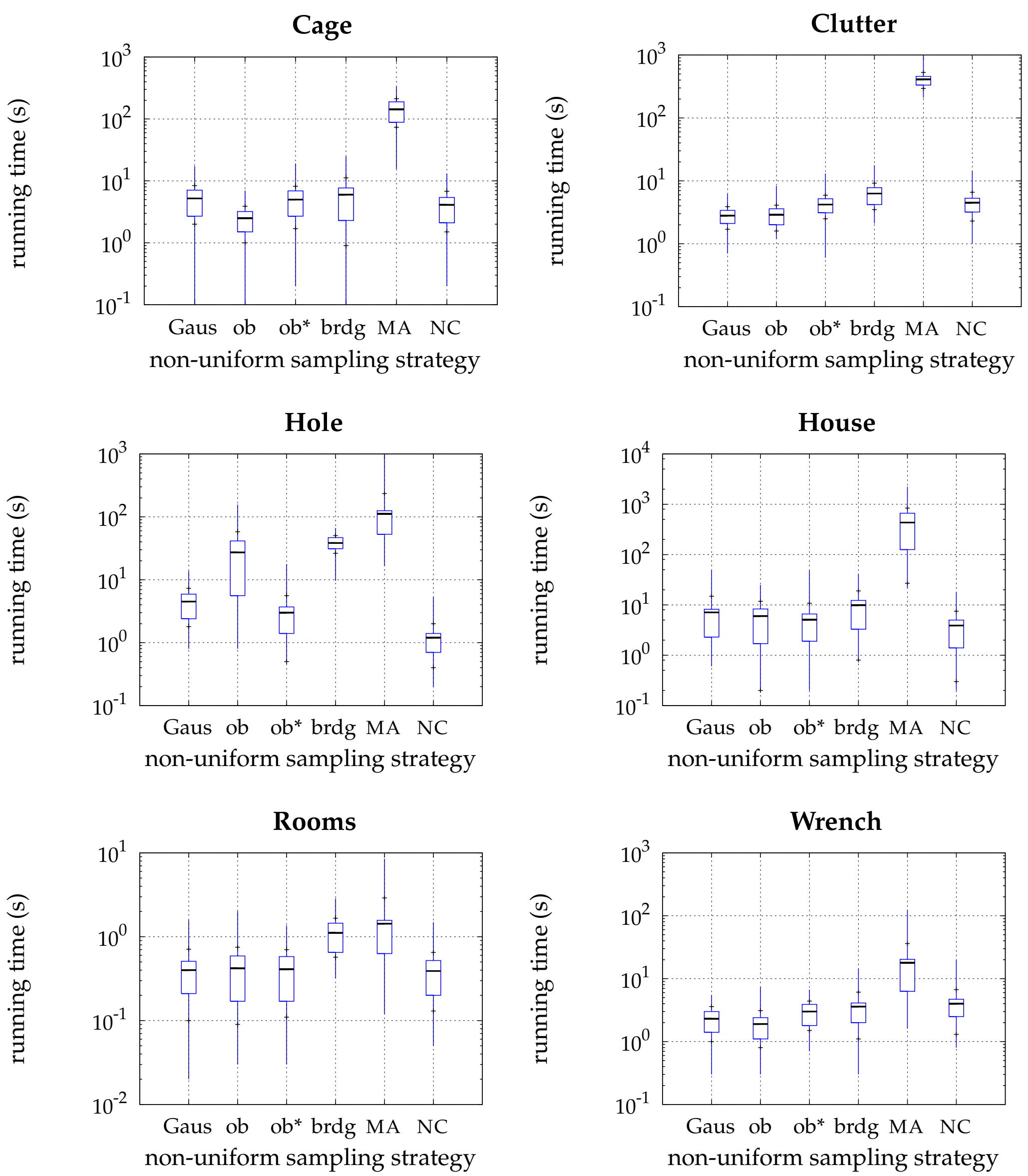


Figure 2.11 Relation between the *non-uniform sampling strategy* and the running time. A series of box plots is shown for each environment. Note that we use a logarithmic scale on the 'running time'-axis.

there was little difference between the various uniform sampling methods. However, the (deterministic) Halton approach performed best. These methods were only outperformed by non-uniform sampling methods for special (narrow passage) cases.

For node adding it turned out that visibility sampling did not perform as well as expected. A technique based on connecting a new configuration to the nearest-k configurations works relatively well. It turned out that the maximum number of connections attempted, i.e. the value of k, has to be set fairly high (like 75). The maximum connection distance is dependent on the environment and robot. This distance should not be set too low as this may increase the running time by orders of magnitude compared to the optimal running time.

One thing that is clear from this study is that a careful choice of techniques is important. Also, it is not necessarily true that a combination of good techniques and parameter choices results in optimal running times. For example, for the Hole environment one might expect that a combination of nearest contact sampling and visibility node adding works best. But experiments showed that this combination is actually about two times worse than the best combination.

The study also shows the difficulty of evaluating the performance of the techniques. In particular the variance in the running time and the influence of certain bad runs were surprisingly large. Further research, in particular into adaptive sampling techniques, will be required to improve this. In addition, further study would be interesting for other robot types, such as articulated and car-like robots, since we only compared techniques for free-flying objects.

We hope that our study shed some more light on the question of what technique to use in which situation. A major challenge is to create planners that automatically choose an appropriate combination of techniques based on scene properties or that learn the optimal settings while running.

In this chapter we compared and analyzed techniques based on solving a particular query. In contrast, the next chapter evaluates the techniques based on solving every possible query. To solve each query, the following two criteria have to be satisfied. First, the free configuration space ($\mathcal{C}_{\text{free}}$) must be covered by the nodes in the graph. Second, for each two nodes in the graph: if these nodes share the same connected component in $\mathcal{C}_{\text{free}}$, then there must exist a path between them in the graph. This distinction will provide an even better understanding of the techniques.

CHAPTER
THREE

REACHABILITY-BASED ANALYSIS

While theoretically, the complexity of the motion planning problem is exponential in the number of degrees of freedom, sampling-based planners can successfully handle this curse of dimensionality in practice. The success of these planners for problems with many degrees of freedom can be explained by the fact that no explicit representation of the free configuration space ($\mathcal{C}_{\text{free}}$) is required. The main operation of these planners is checking configurations for collisions with obstacles in the environment, which can be performed efficiently by the current generation of collision checkers. The second reason for their success is that problems which are not pathological have favorable reachability properties. That is, the free configuration space of a reasonable problem can often be captured with few nodes and each node can reach a large portion of $\mathcal{C}_{\text{free}}$ using a local planner. Therefore, a PRM usually finds a solution quickly, even if the geometric complexity is high.

In the previous chapter, we compared and analyzed techniques based on solving one particular query. In contrast, we now inspect the techniques based on solving every possible query. We say that a motion planning problem is solved if the following two criteria are satisfied. First, each free configuration in $\mathcal{C}_{\text{free}}$ can be reached by at least one node in the graph (*coverage*). Second, for each two nodes in the graph: if these nodes share the same connected component in $\mathcal{C}_{\text{free}}$, then there exists a path between them in the graph (*maximal connectivity*). Our experiments show that covering $\mathcal{C}_{\text{free}}$ is not the main difficulty, but getting the nodes connected, especially when the environments get more complicated, e.g. a narrow passage is present. The narrow passage problem can be tackled by incorporating a hybrid sampling strategy that aims at concentrating samples in difficult areas. The strategy must also generate some samples in ample free spaces. Another strategy to get $\mathcal{C}_{\text{free}}$ faster connected is to use a more powerful local planner. We present a potential field local planner that creates larger reachability regions which eases making connections. Also this planner is better able to find the entry of a narrow passage, decreasing the number of regions needed to get the nodes connected. Our experiments show that this approach leads to a better performance of sampling-based methods.

3.1 Introduction

The complexity of a motion planning problem is often expressed in terms of geometric complexity (of the obstacles and moving object) and the number of degrees of freedom (DOFs) of the moving object. This is reasonable for methods that are based on the geometry of obstacles such as visibility graphs, Voronoi diagrams and exact cell decompositions. In practice, these methods fail when the geometric complexity is high or when there are many (> 3) DOFs or many primitives involved.

Complexity analysis is also employed for sampling-based planners such as the PRM. Analyses for these planners use the coverage of the free configuration space ($\mathcal{C}_{\text{free}}$) with (hyper)spheres which results in exponential complexity bounds (see e.g. [81]). Yet, in practice, the PRM can successfully handle this curse of dimensionality because it is *reachability*-based, i.e. a sample can often be connected to other samples that are far away because they can be reached by the local planner. For example, if each sample can reach a large part of $\mathcal{C}_{\text{free}}$ by using a local planner, then $\mathcal{C}_{\text{free}}$ will be covered and connected quickly. This does not follow from the standard analysis that only allows a sample to be connected to its adjacent neighbors. Another reason why the PRM is fast is because its primitive operations are simple. Checking samples for collisions does not require an explicit representation of the configuration space (whose combinatorial complexity can be very high). When a path or a sample is checked for collisions, only the obstacles in the vicinity are involved. As a result, 'redundant' primitives on the other side of the environment do not affect the performance. These properties lead to a favorable performance that is proportional to some measure of difficulty for the problem to be solved.

In this chapter, we will study properties of commonly used techniques in sampling-based planning by performing a reachability analysis which emphasizes the notions of *coverage* and *maximal connectivity*. These concepts are introduced in Section 3.2. In Section 3.3, we describe the experimental setup. In the following three sections, we analyze neighbor selection, sampling and local planning techniques, resulting in a better understanding of these techniques.

We observe that the main difficulty is not getting $\mathcal{C}_{\text{free}}$ covered, but getting the nodes connected, especially when the problem gets more complicated. We conclude in Section 3.7 that a hybrid sampling technique and a newly proposed potential field local planner lead to a better performance of the PRM.

3.2 Coverage and maximal connectivity

The PRM was designed to be a multiple shot planner which enables fast querying. This goal can be achieved by creating a graph $G = (V, E)$ that covers $\mathcal{C}_{\text{free}}$ and captures its connectivity. We define coverage and maximal connectivity as follows:

Definition 3.1 (coverage). *G covers $\mathcal{C}_{\text{free}}$ when each configuration $c \in \mathcal{C}_{\text{free}}$ can be connected using the local planner to at least one node $v \in V$.*

Definition 3.2 (maximal connectivity). *G is maximally connected when for all nodes $v', v'' \in V$, if there exists a path in $\mathcal{C}_{\text{free}}$ between v' and v'', then there exists a path in G between v' and v''.*

Coverage ensures that every query (which consists of a start and goal configuration) can be connected directly to the graph, as is required to solve the query. If there exists a path (in $\mathcal{C}_{\text{free}}$) between the start and goal configuration, then maximal connectivity ensures that a path between them can be found in the graph. Note that the path in the graph and the path in $\mathcal{C}_{\text{free}}$ do not have to be in the same homotopic class.

If both criteria are satisfied, then a path can always be found for every query. Additional criteria are of course imaginable, for example creating a graph that optimizes path quality (see Part II), but we will not consider them here. Several authors have studied the use of PRM for solving single motion planning queries. For single shot techniques, coverage does not play an important role, and, our analysis is less relevant.

We use coverage and connectivity as an analysis tool to gain insight in sampling-based methods. Our goal is to determine for various techniques how long it takes before $\mathcal{C}_{\text{free}}$ has been covered and connected. Because this would be rather complex for a continuous (high-dimensional) configuration space $\mathcal{C}$, we discretize $\mathcal{C}$ (for problems that have 2D and 3D $\mathcal{C}$-spaces): for each cell (whose dimensions are determined by the step size used by the local planner) in $\mathcal{C}$, we check whether the placement of the robot for that cell is free and store this information in an array. When a node v is added to V, its discretized *reachability* region is calculated by checking for each free cell c in the array whether

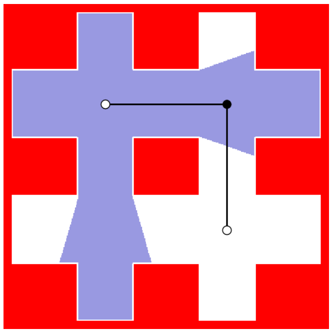
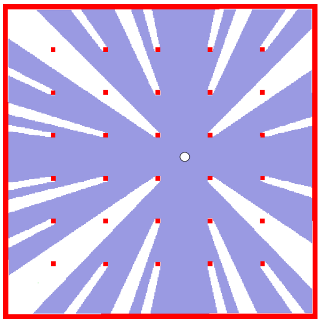


Figure 3.1 The coverage and maximal connectivity criteria have been met. The reachability regions of the white nodes cover the complete free space and are connected via the black node.

there exists a local path between v and c. All free cells that can be connected by the local planner are labeled with a unique region number. If each free cell has been covered by at least one region, the coverage criterion has been met.[1] The connectivity criterion is verified as follows: for each added node $v \in V$ we calculate the set of nodes $W \subseteq V$ to which it can be connected through the grid of free cells. Then we add all combinations of v with W to a connectivity list. If there exists a path in graph G for each connection in the connectivity list then $\mathcal{C}_{\text{free}}$ is maximally connected. Please realize that these calculations are only done to compare planning techniques. They are not part of the actual motion planning algorithm and, hence, are not taken into account when reporting the running times.

As an example, Figure 3.1 shows an environment whose free space is covered by two (white) nodes and is connected via one extra (black) node. Hence, the three-node graph suffices to solve this problem. The reachability region for the upper left node has been drawn. Each configuration in this region can be connected with a straight-line local planner to the node. The shape of a reachability region can be complicated. Figure 3.2(a) shows a region for a 2D environment with many small obstacles and Figure 3.2(b) shows a 3D region for the manipulator arm depicted in Figure 3.3(f) with three rotational DOFs.

[1]We will provide a more efficient way to compute a reachability region in Section 6.4.1.


(a) A 2D reachability region

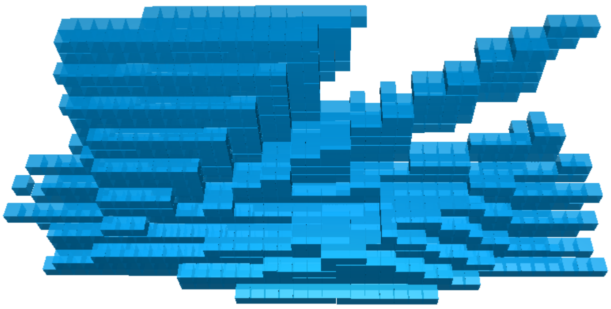

(b) A 3D reachability region for the robot of Figure 3.3(f)

Figure 3.2 Complicated 2D and 3D reachability regions.

3.3 Experimental setup

For the experiments we use our SAMPLE system. As the techniques involve random choices, we report statistics gathered from 100 independent runs for each experiment. See Section 1.2 for more information on our general experimental setup.

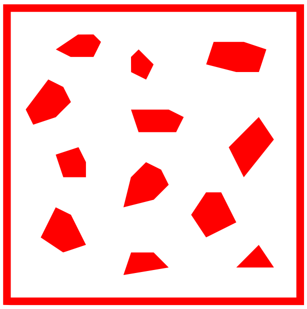
We used the six environments depicted in Figure 3.3. Their bounding boxes are stated in Table 3.1. The environments have the following properties:

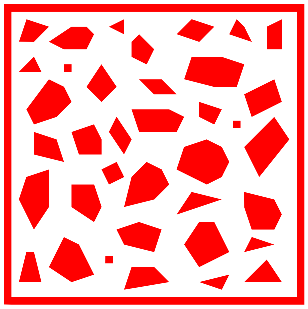
Clutter 1 The 2D cluttered environment consists of 16 polygons through which a small robot must navigate. It should be easy to create a maximally connected roadmap for this environment, because each sample will cover a large portion of the free space. The robot is a square with two translational degrees of freedom (DOFs).

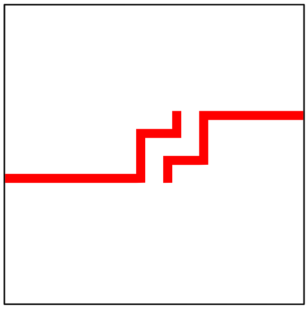
Clutter 2 We added 24 polygons to the first environment to reduce the average size of the regions. Again, the robot is a translating square.

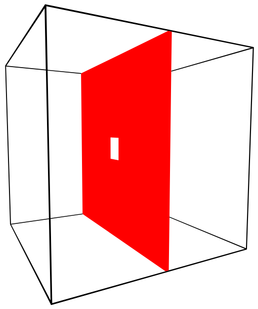
Narrow passage This environment has been designed to be more difficult. It contains a narrow passage through which a square has to move. The passage is surrounded by two large open spaces. The robot is a translating square.

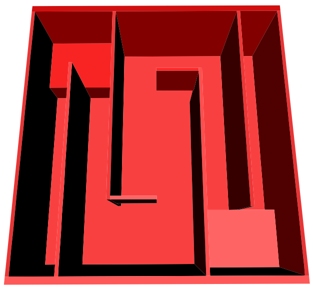
Hole The Hole environment has two large open spaces separated by a wall with a narrow hole in it. The robot is a small cube that can only translate. Each sample will cover a large portion of $\mathcal{C}_{\text{free}}$.


(a) Clutter 1


(b) Clutter 2


(c) Narrow passage


(d) Hole


(e) Corridor

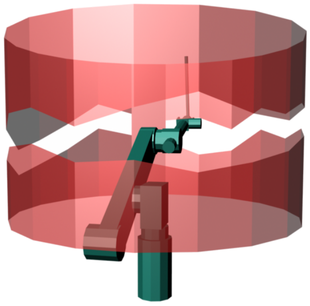

(f) Manipulator

Figure 3.3 The six test environments.

Corridor A small translating cube has to move through a 3D winding corridor consisting of four hairpins. The walls of the corridor will limit the size of the reachability regions.

Manipulator This 3D environment features a robot arm with three rotational DOFs which operates in a constrained workspace. The $\mathcal{C}$-space has a long passage.

We discretized the $\mathcal{C}$-space of each environment.[2] The level of discretization can be found in Table 3.2. Consider for example the Clutter 1 environment: The ranges of the translational DOFs of the robot are $[0 : 40] \times [0 : 40]$. The step size used by the local planner is 0.5. Hence, the $\mathcal{C}$-space is discretized with 80×80 cells.

[2]We used other environments than in Chapter 2. We did not use problems with more than three DOFs because the analysis of coverage and connectivity would require too much memory.

	Dimensions of the bounding boxes	
	environment	**robot**
Clutter 1	40×40	0.5×0.5
Clutter 2	40×40	0.5×0.5
Narrow passage	40×40	0.5×0.5
Hole	$40 \times 40 \times 40$	$2 \times 2 \times 2$
Corridor	$40 \times 8 \times 40$	$2 \times 2 \times 2$
Manipulator	variable	variable

Table 3.1 Information on the workspaces of the environments.

	DOF range	**step size**	**number of cells**
Clutter 1	40×40	0.5	80×80
Clutter 2	40×40	0.5	80×80
Narrow passage	40×40	0.5	80×80
Hole	$40 \times 40 \times 40$	2.0	$20 \times 20 \times 20$
Corridor	$40 \times 8 \times 40$	2.0	$20 \times 4 \times 20$
Manipulator	$6.1 \times 0.6 \times 0.7$	0.05	$122 \times 12 \times 14$

Table 3.2 Information on the configuration spaces of the environments.

	Sampling strategy	
	Gaussian	**Bridge**
Clutter 1	4.0	4.8
Clutter 2	4.0	4.8
Narrow passage	1.2	2.4
Hole	4.0	5.6
Corridor	4.0	5.6
Manipulator	1.2	4.0

Table 3.3 The optimal values of σ used in Gaussian and Bridge sampling.

We used the metric from Section 1.2.3. We set the weights w_i for the translational DOFs to 1 and set the weights for the rotational DOFs to 6. The optimal parameters for the sampling strategies are listed in Table 3.3 and the optimal parameters for neighbor selection strategy are listed in Table 3.4.

For each experiment, we ran the PRM until both coverage and maximal connectivity had been achieved and recorded the following statistical data.

Definition 3.3 (number of regions). *Each node $v \in V$ in the graph implies a new region. The number of regions is denoted by k.*

Definition 3.4 (average size of the regions). *Let k be the number of regions r_i discovered so far. Furthermore, let $|r_i|$ be the number of cells in region i and $|\mathcal{C}_{\text{free}}|$ the total number of free cells. Then the* average size of the regions *equals: $\frac{1}{k}\sum_{i=1}^{k} |r_i| / |\mathcal{C}_{\text{free}}|$.*

We recorded the number of regions and the average size of the regions at two moments: the moment that $\mathcal{C}_{\text{free}}$ was covered and at the moment that $\mathcal{C}_{\text{free}}$ was maximally connected (after $\mathcal{C}_{\text{free}}$ was covered). We also recorded the running time after both criteria had been satisfied. We give an indication of the dispersion of the running times by a *box plot*. Each plot consists of a box, one large and two small horizontal lines and a vertical line. The box represents the middle 50% of the data, the large horizontal line represents the average, the small lines represent the average $\pm$ the standard deviation and the vertical line represents the minimum and maximum value.

3.4 Neighbor selection strategy

The neighbor selection strategy specifies for a particular sample how a set of neighbor samples is chosen to which it is connected. The goal of the strategy is to make the graph connected as fast as possible. A strategy usually selects neighbors based on a combination of the following criteria: the *maximum connection distance*, the *maximum number of connections* tried, and the *node adding strategy*, see Section 2.5. We chose the *nearest-k* node adding strategy as this method performed reasonably well on different environments. We used the optimal sampling strategy for each environment (see below). That is, we used Bridge sampling for the Narrow passage environment. For the other ones, we used Halton* sampling. We study the effects of different choices for the first two criteria on the six test environments.

3.4.1 Maximum connection distance

We concluded in Section 2.5 that the maximum connection distance should not be too small nor too large. A very small connection distance will always require an exponential number of samples (in the number of DOFs) which increases the running time a lot. The PRM works best if reasonably long connections can be made. In general, it is not useful to try too long connections since the chance of success for such connections is small while the collision checks required for testing the local path are expensive.

In each of the following experiments, we varied the maximum connection distance from a small value (close to zero) to a large value. The maximum number of connections was set to 75 (see below). Figure 3.5 shows the results for the 2D environments and Figure 3.6 shows the results for the 3D environments.

When we make the connection distance very small, the PRM starts looking like grid-based techniques in which samples are only connected to their direct neighbors. The figures show that this considerably increases the (average and variance of the) running time. Furthermore, there is a large difference in the moment of coverage and the moment of maximal connectivity. This shows that a small connection distance complicates making connections.

It is clear, as the maximum distance gets larger, that the average size of the reachability regions will increase (up to some value), and hence, the number of samples needed to solve the problem will decrease. In other words, the number of samples required to solve the problem decreases when the time required per sample increases. The results show that there is some optimal trade-off which is dependent on the environment (and metric). If the size of a region (that corresponds to a particular sample) is small, then the sample can be connected to other samples to which the distance is small, and *vice versa*. In general, if the average reachability of the samples is low, then a small connection distance is preferable and *vice versa*. See Figure 3.4 which shows the average size of the regions for the six environments corresponding to the optimal neighbor selection parameters. We can make two observations. First, the larger the average size of the regions, the smaller the growth of running times when the maximum connection distance increases. Second, the larger the average size of the regions, the larger the value for the optimal maximum connection distance. This information can be used to estimate the (local) optimal maximum connection distance. For example, when a sample can be connected to other samples at a large distance, then the average size of its reachability region is large. Hence, using a large maximum connection distance is a good choice for choosing neighbors. In addition, few samples should be created near the sample.

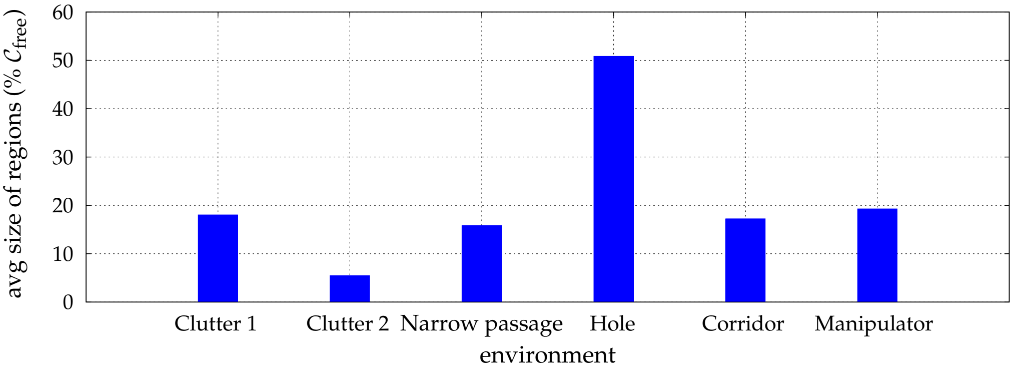


Figure 3.4 The average size of the regions corresponding to the optimal neighbor selection parameters.

The results again confirm that the PRM derives its strength from making long connections. While a very small connection distance has a dramatic negative impact on the running time, a large value only has little impact, especially when the average size of the reachability regions is large.

3.4.2 Maximum numbers of connections

The second analyzed criterion for selecting neighbors is the *maximal number of connections* attempted to connect a node. We concluded in Section 2.5.2 that this value should be large. In this section we will show the relation between the maximal number of connections and the coverage and connectivity.

The number of attempted connections does not influence the coverage, but has a clear influence on the connectivity. If the number of connections is too small, it might be hard to get the free space maximally connected because the chance is small that those few samples are selected to which a connection is possible. If connections are attempted with (too) many nodes, $\mathcal{C}_{\text{free}}$ will become maximally connected using less regions. Nevertheless, this might negatively influence the running time since testing those connections is expensive.

In our experiments, we varied the maximum number of connections. Figure 3.7 shows the results for the 2D environments and Figure 3.8 shows the results for the 3D environments. Indeed, when only a few connections are tried, more regions are needed to get the roadmap maximally connected. Making more and more connections does not seem to be useful because the number of

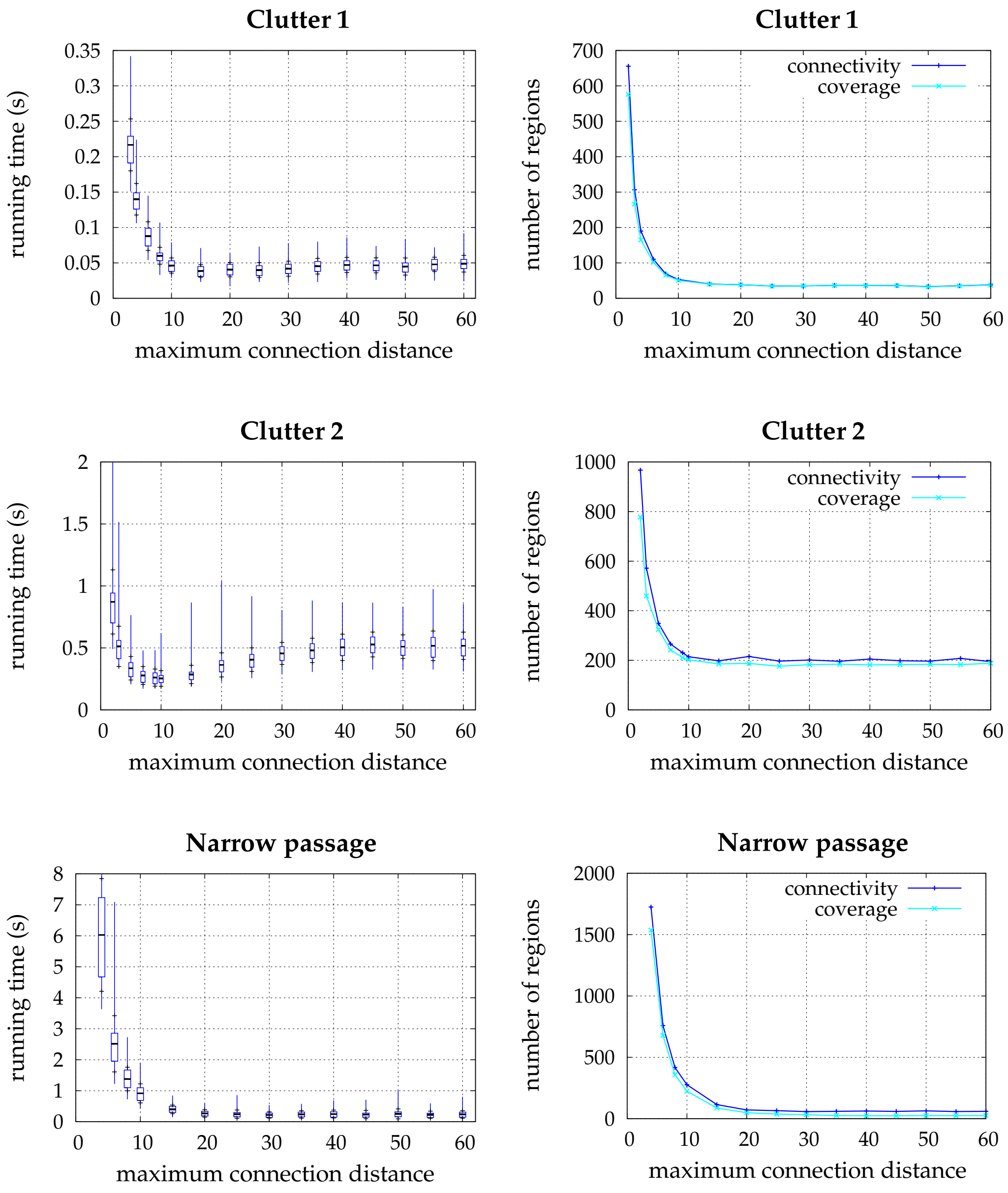


Figure 3.5 Influence of the *maximum connection distance* on the running time and number of regions required to get the free space covered and maximally connected in the 2D environments.

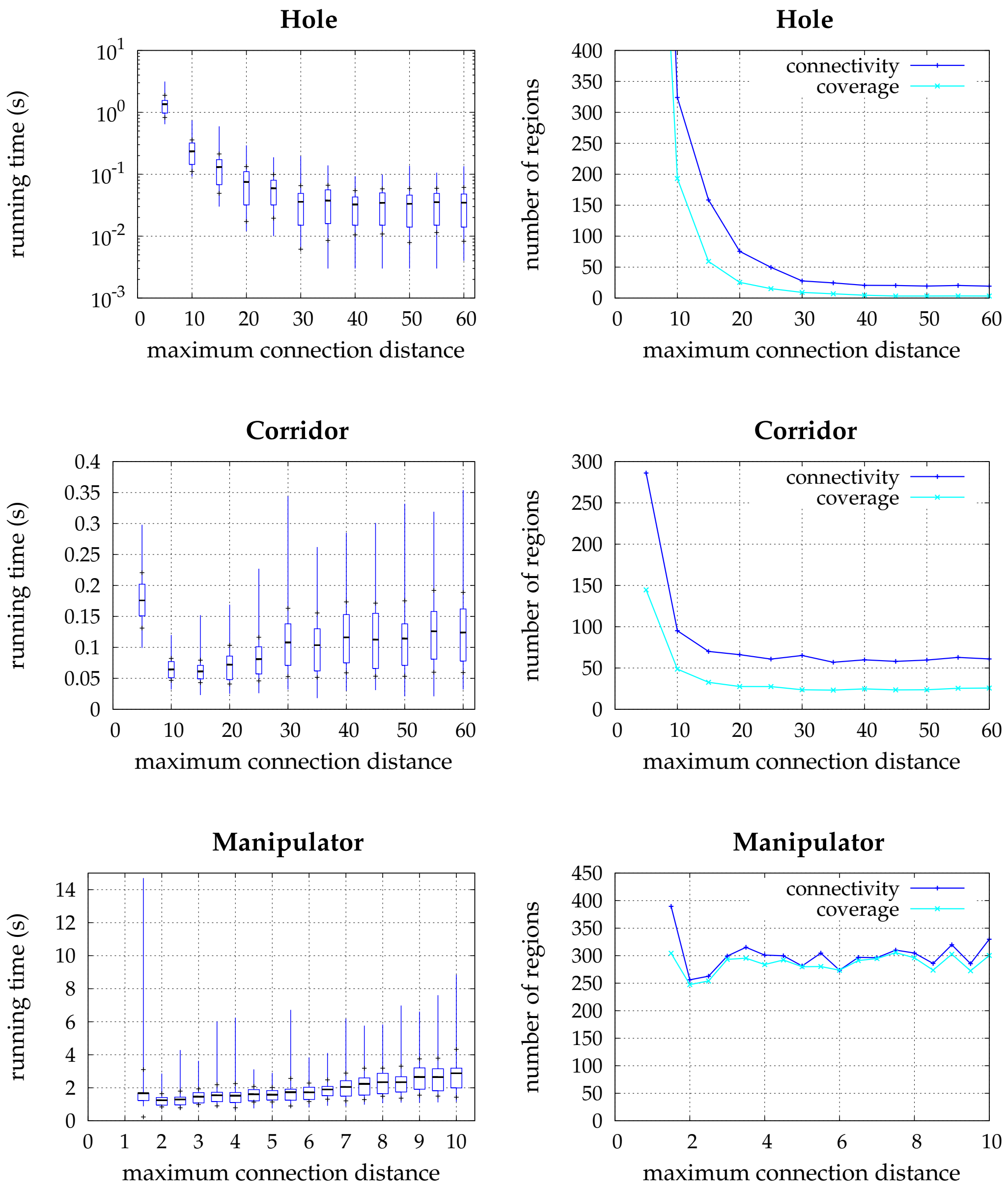


Figure 3.6 Influence of the *maximum connection distance* on the running time and number of regions required to get the free space covered and maximally connected in the 3D environments.

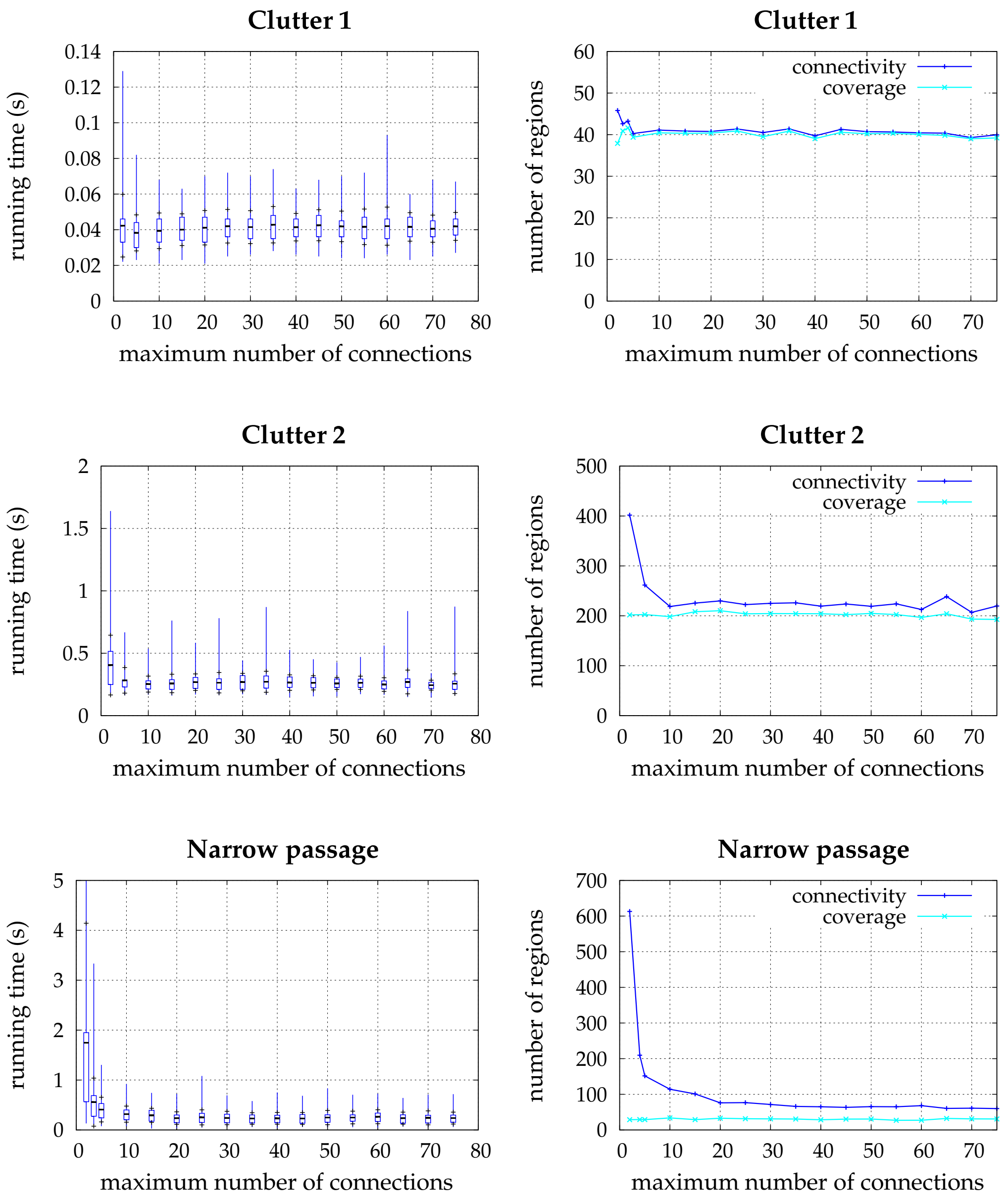


Figure 3.7 Influence of the *maximum number of connections* on the running time and number of regions in the 2D environments.

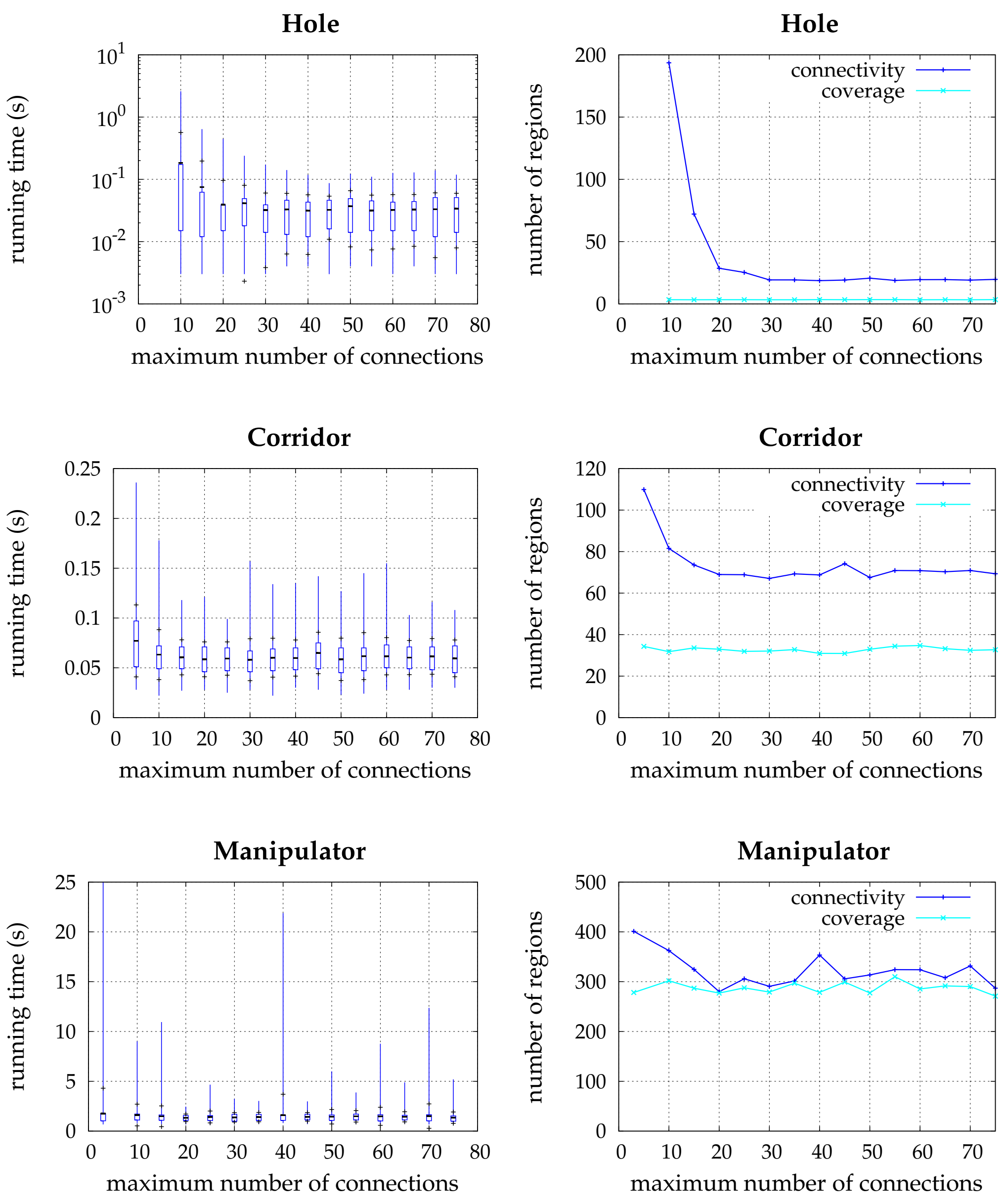


Figure 3.8 Influence of the *maximum number of connections* on the running time and number of regions in the 3D environments.

	Neighbor selection parameters	
	max. connection distance	**max. number of connections**
Clutter 1	15	75
Clutter 2	10	75
Narrow passage	30	75
Hole	40	75
Corridor	15	75
Manipulator	2	75

Table 3.4 The optimal values used in the neighbor selection strategy.

regions needed to get $\mathcal{C}_{\text{free}}$ covered and maximally connected remains constant at a certain maximum number of connections. However, as a large maximum number of connections (e.g. 75) does not affect the running time much, we recommend to use such a high value. This can be explained as follows. First, we observe that setting the maximum number of connections larger than the expected number of regions (that satisfies the coverage and connectivity criteria) has no effect on the running time. Consider for example the Clutter 1 environment. As it uses a random sampling strategy, the expected number of samples to which a particular sample can be connected is the connection area, divided by the area of the free space, multiplied by the average number of regions that corresponds to satisfying both criteria: the expected number of samples equals $\pi * 15^2/40^2 * 40 = 17.7$. This means, if the samples are uniformly random spread in the space and a maximum connection distance of 15 is used, that the expected number of samples to which a connection is tried (per sample) is 17.7. Setting the maximum connection distance higher than this value will have little effect on the running time. Setting the parameter slightly smaller than this value will also not have much effect on the running time as the chances are high that the discarded samples do already belong to the same connected component.

3.5 Sampling

The PRM has been expressed as a sampling-based motion planning method. In this section we will study the behavior of different sampling techniques. They can be classified into three categories: uniform, non-uniform and hybrid techniques.

The first category comprises the uniform techniques such as random, grid, cell-based and Halton sampling. It is well known that these techniques can have difficulties dealing with the narrow passage problem, see Section 2.6. The second category tackles the narrow passage problem by biasing the sampling distribution. That is, more samples are added in 'difficult' regions of the environment. A region is difficult if the size of the region (which corresponds to a particular sample) is small compared to the total free space. The number of samples that are generated within these regions can be increased by filtering out samples that probably do not contribute to the coverage and maximal connectivity of the roadmap. Examples include Gaussian and obstacle-based sampling. The third category combines the strengths of the previous two categories. The bridge test for example concentrates samples in difficult areas but it also generates some samples in open areas. Several combinations of existing sampling strategies are suited to serve as a hybrid technique, see the paper of Hsu *et al.* [73] for an elaboration.

Experiments

For each category we choose a representative method. For the uniform technique we choose *Halton**, for non-uniform *Gaussian* and for hybrid we choose *Bridge test*. See Section 2.6 for more information on these techniques. We will compare their behavior by considering the experiments we performed on the six environments. These environments are representative for many different motion planning problems so we expect the observations to apply rather general.

Figure 3.9 shows the results for the 2D environments and Figure 3.10 shows the results for the 3D environments.

Halton* sampling resulted in relatively low running times for all environments, except for the Narrow passage environment. This is consistent with the results from the previous chapter. Halton's uniform distribution created too many samples in the two wide open areas and too few in the narrow passage. Since the chance is small of obtaining a set of samples that covers the space in the narrow passage, the total number of regions required for coverage and maximal connectivity was higher for this method than for the other two methods.

The Gaussian technique needed fewer samples than Halton* in the Narrow passage environment. The reason for this is that relatively more samples are concentrated in the difficult areas of the $\mathcal{C}$-space, which resulted in faster coverage. In addition, the ample free space was covered fewer times (due to

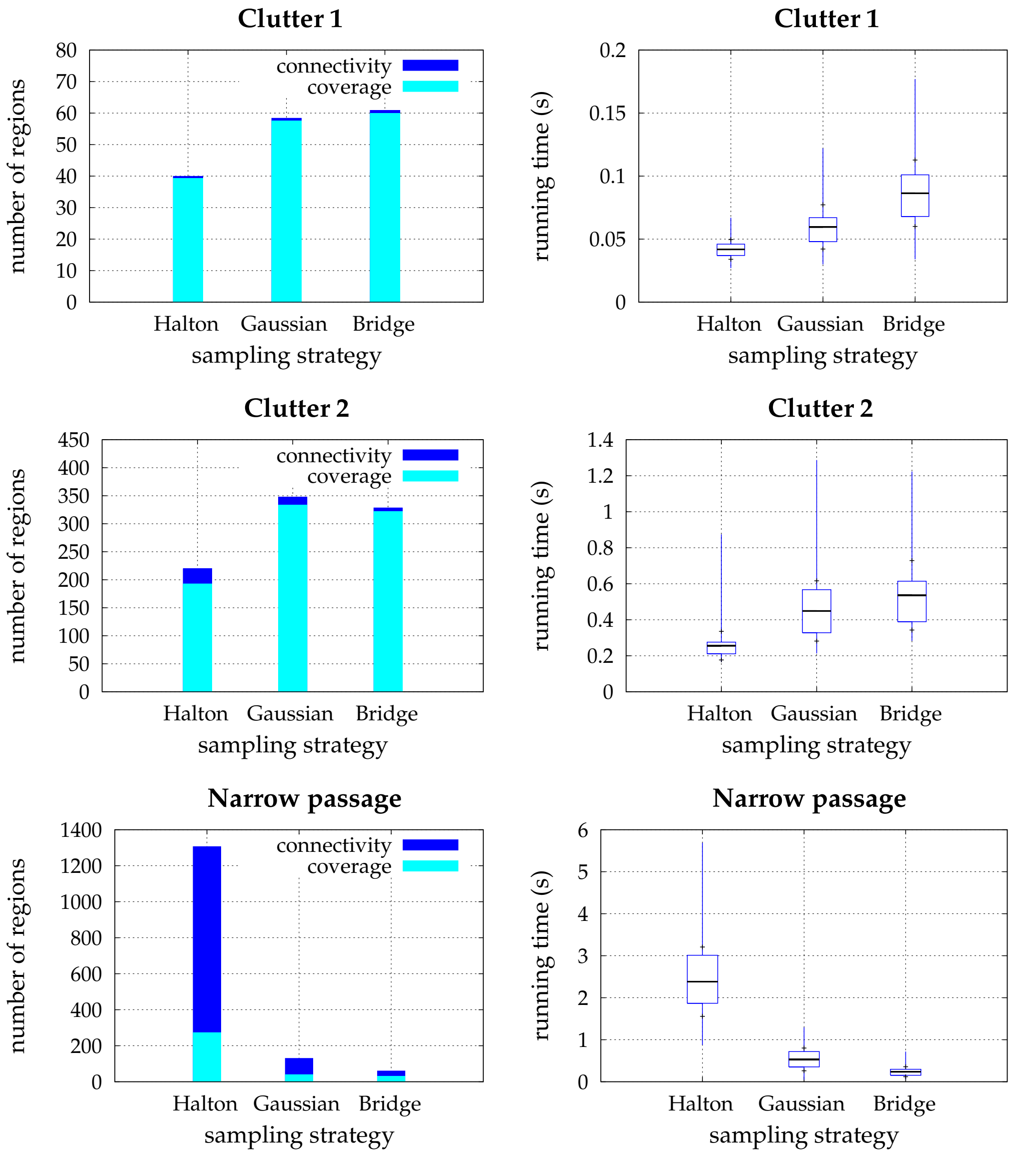


Figure 3.9 Sampling statistics for the 2D environments. The charts in the left column show the average number of regions required to cover and connect the free space. In the Narrow passage environment for example, Halton sampling required 275 regions to cover the free space and 1300 regions to connect the free space. The right column shows box plots corresponding to the time needed to satisfy both criteria.

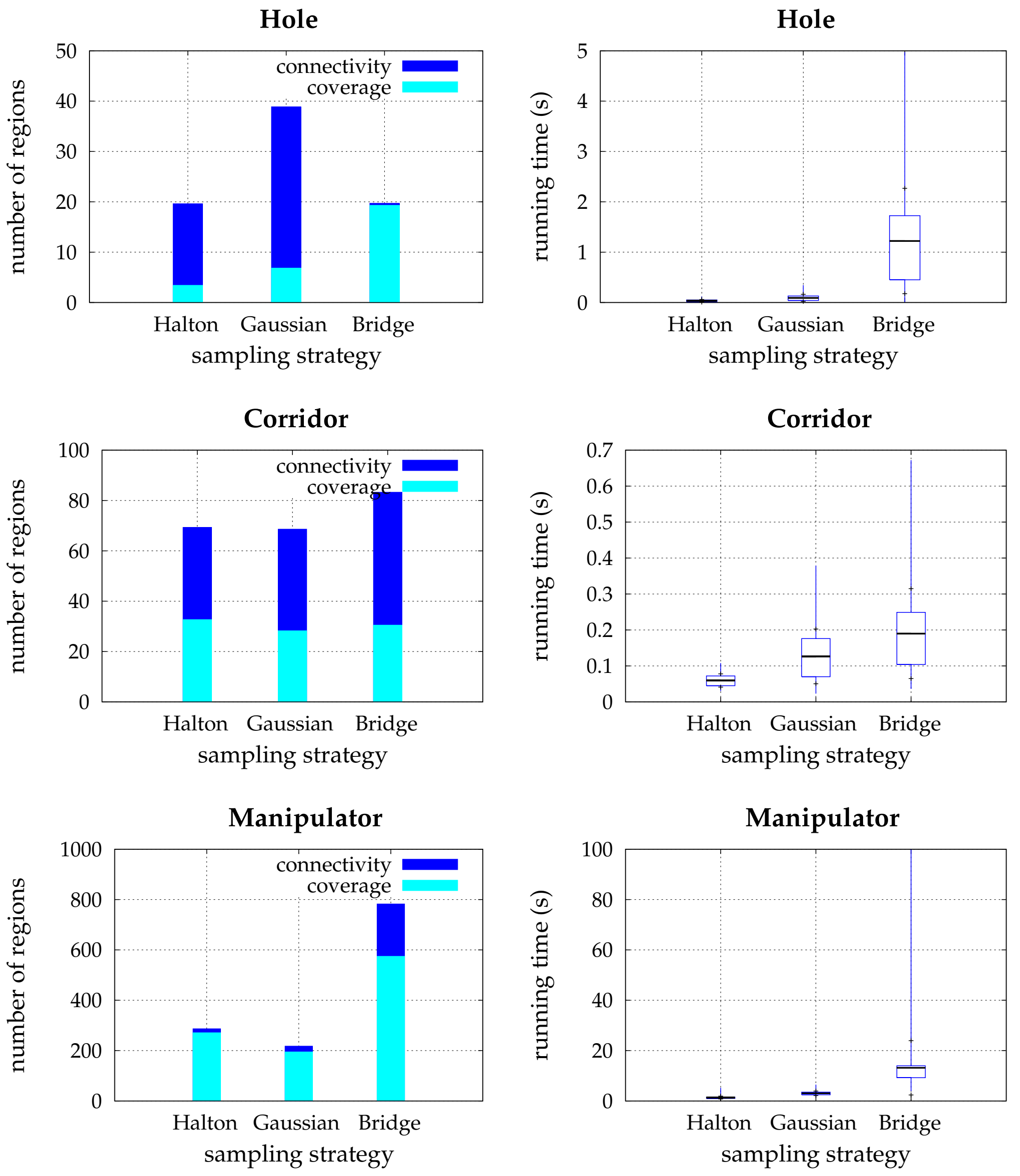


Figure 3.10 Sampling statistics for the 3D environments.

the Gaussian distribution of the samples). As this distribution generates fewer samples in ample free space and more near obstacles, it can be difficult to connect them. This explains why connecting $\mathcal{C}_{\text{free}}$ involved more than three times as many regions compared to covering $\mathcal{C}_{\text{free}}$.

The Bridge technique has been designed to combine the strengths of the previous two categories by concentrating samples in difficult areas while some samples are also generated in open areas. This resulted in the lowest average running time and the lowest number of regions needed to get $\mathcal{C}_{\text{free}}$ connected in the Narrow passage environment. However, the technique performed moderately on the other environments. The charts show that it had difficulties in getting $\mathcal{C}_{\text{free}}$ covered. This can be explained by looking at the properties of these environments. As they have no narrow passages, a technique that is designed to create samples in narrow passages will spend time uselessly. In the Hole environment, many Bridge samples were created near the walls while most of them did not increase the coverage of $\mathcal{C}_{\text{free}}$. The same argument holds for the Manipulator environment.

By looking at the charts of Figure 3.9 and Figure 3.10, we can make an important observation. For the Clutter 1 and Clutter 2 environments, the difference between the moment that $\mathcal{C}_{\text{free}}$ was covered and the moment that $\mathcal{C}_{\text{free}}$ was maximally connected is very small. In contrast, for the Narrow passage environment this difference was much larger. Hence, covering $\mathcal{C}_{\text{free}}$ is not the problem, but getting $\mathcal{C}_{\text{free}}$ maximally connected is more difficult when the environment contains a narrow passage. To clarify this, we considered two versions of the Manipulator environment. The first variant is the one depicted in Figure 3.3(f). The second variant is the same as the first one, except that we made the passages narrower by scaling the workspace in the y-direction, i.e. the workspace became 25% less high. In the first variant, the Halton sampling strategy needed about 271 samples to cover the space. Only 6% more samples were needed to connect the space. While the second variant needed the same number of samples to cover the space, connecting the space required five times as many samples. Hence, connecting samples near or in narrow passages is more difficult. The main observation that can be made is that the narrow passage problem is not so much caused by coverage but by connecting the nodes. Rather than concentrating on more clever sampling, it may be beneficial to spend more effort on connecting nodes in difficult regions. Actually, already one of the first papers on PRM did this by trying to connect difficult nodes in a second phase using a bouncing strategy [124]. In Section 3.6, we will show how more powerful local planners can be used for this as well. The challenge is to apply such a connection strategy only when and where it is necessary.

Ideal sampling strategy

An ideal sampling strategy should create few samples that covers and connects $\mathcal{C}_{\text{free}}$. The smaller the number of samples, the less time is needed to connect those samples which is the most time-consuming step in the PRM. However, some overlap between the regions that belong to the samples is required because this simplifies creating connections between them. This can be achieved by creating a hybrid technique which filters out samples that do not contribute to extra coverage or maximal connectivity. The visibility sampling technique tries to achieve this by throwing away nodes [121]. Section 6.5 will show that this technique has difficulties satisfying the coverage and maximal connectivity criteria. The approach is too strict and should probably be combined with other sampling techniques.

In Chapter 6, we propose a new technique that creates small graphs satisfying both criteria.

3.6 Local planners

In the previous sections we showed that it can be difficult to connect certain nodes while the coverage criterion has already been met. If we were able to create a local planner that is more powerful than the straight-line local planner (SLLP), then we could decrease the gap between the moment of coverage and maximal connectivity, improving the total running time. Although this new planner might be more time-consuming, a careful trade-off between the power and speed of the planner should lead to a better performance of the PRM.

To be successful, this planner should preferably satisfy the following two criteria. First, it must cover at least the same volume as the SLLP does, i.e. it must subsume each reachability region that is created using the SLLP. If the regions are larger we expect that the space is covered faster. More importantly, because of the larger expected overlap between the regions, they will sooner become maximally connected. Second, the planner should be fast enough to be useful in practice. This can be achieved by letting the planner behave as a SLLP if the connections can be made in a straight line; if the straight-line connection results in a collision, then a more clever approach should be employed. These criteria are satisfied by the simple potential field local planner (PFLP) we will describe below.

In general, a potential field method calculates distances between the robot and obstacles to define a force vector on the robot [99]. These operations make a PFLP expensive in comparison to the simple SLLP. To mitigate this effect, we use a modified version of the potential field planner used in [124].

It is implemented as follows. The planner tries to make small steps on the straight line toward the goal, as does the SLLP.[3] This assures that the region reached by the PFLP subsumes the region reached by the SLLP. When the robot walks into an obstacle, the planner checks a step from the last collision-free configuration in several directions on the hemisphere in configuration space oriented toward the goal. The most promising step is considered first. A local minimum is easily detected when all possible steps fail in which case the local planner stops and reports failure.

Although this planner is more powerful, it will be more expensive than the SLLP in terms of consumed time. A second drawback is that a new parameter is introduced that has to be optimized, i.e. the number of directions on the unit sphere has to be chosen. We choose the axes and diagonals as directions, i.e. in 2D we choose 8 directions and in 3D we choose 26 directions on the unit sphere and select only those that bring the robot closer to the goal. It is a trade-off between the accuracy and speed: the higher this number, the larger the reachability region, but the slower the planner is. The number of directions we choose seems to work reasonably, but it is in essence arbitrary.

Figure 3.11 shows the Clutter 2 and Narrow passage environments for each of which a reachability region is drawn. The left pictures show the area that can be reached by the SLLP from a particular sample. The right pictures correspond to the PFLP. While reasonably long connections can be made by the SLLP, the reachability region of the PFLP significantly extends the area to which connections can be made. Besides the advantage of covering larger regions, the PFLP, in contrast to the SLLP, is able to find its way through the narrow passage. This allows connections to be made from one side of the passage to the other.

We expect that the PFLP outperforms the SLLP in all environments except the Corridor environment, because the reachability regions will be larger than those created with the SLLP. The PFLP will be able to easily connect the two ample free spaces in the Narrow passage and Hole environments. By contrast, in the Corridor environment, the PFLP may only have an advantage in the hairpins; much ineffectual work might be done elsewhere before it is concluded that no connections exist. This is expected to have a negative effect on the running time.

[3]However, the SLLP uses binary collision checking while the PFLP uses incremental collision checking, see Section 2.4.

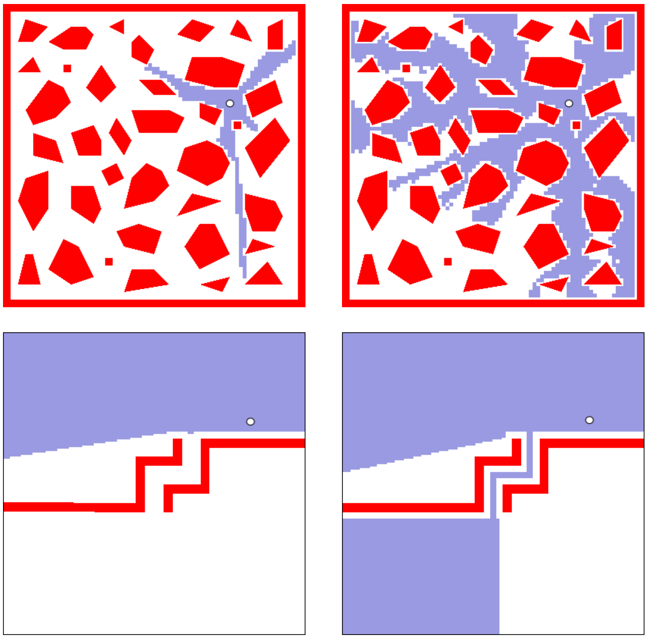


Figure 3.11 Reachability regions for straight-line (left column) and potential field local planner (right column) in the Clutter 2 environment (top row) as well as the Narrow passage environment (bottom row).

Experiments

We used the optimal sampling strategy in all experiments.[4] That is, we used Halton* for each environment except the Narrow passage environment where we used Bridge sampling. As neighbor selection strategy we used nearest-k, where k was set to 75.

In the first experiment we study the effect of using the SLLP or PFLP on the average size of the regions. We set the maximum connection distance to infinity to reveal the full potential of the planners. Figure 3.12 shows the results. In all cases, the PFLP covered larger regions than the SLLP. For the Clutter 1 environment, the PFLP created regions that were on average 86% of $\mathcal{C}_{\text{free}}$ compared to 27% for the SLLP. In the Clutter 2 environment, the regions of the PFLP were even ten times as large as the regions of the SLLP. Even in the Corridor environment, the differences were large.

[4]Unfortunately, we cannot conduct experiments with the Manipulator environment as our implementation of the PFLP currently does not support rotational DOFs.

Next, we conduct experiments to find out whether using a PFLP (which creates larger regions, but takes more time per region) outperforms a SLLP (which creates smaller regions, but takes less time per region). To make a fair comparison possible, we use for both planners the optimal choices. The SLLP uses the neighbor selection parameters from Table 3.4. The maximum connection distances for the PFLP are set to $\{\infty, \infty, 30, \infty, 20\}$ for the different environments. The PFLP generally uses larger (optimal) maximum connection distances as the regions will be larger.

Figure 3.13 shows the average number of regions and Figure 3.14 shows the running times for the two local planners. The results show, as expected, that the number of regions needed to cover the space are considerably lower for the PFLP, which is caused by the larger average size of the regions. In addition, the maximal connectivity criterion was satisfied by considerably less regions. The (absolute) difference between the number of nodes to achieve coverage and number of nodes to achieve maximal connectivity was much smaller for the PFLP than for the SLLP. The PFLP clearly outperformed the SLLP in all environments, except in the Corridor environment. However, the difference of the average running time is small for this environment. Thus, the PFLP turns out to be an efficient local planner.

Ideally, a local planner should be simple in an 'easy' part of the $\mathcal{C}$-space and more advanced in more 'difficult' parts. The potential field local planner combines those requirements: easy connections (i.e. straight-line connections) are made at the expense of a marginal overhead, while difficult connections (i.e. connections that avoid obstacles) can actually be made. Experiments showed that sampling-based methods can benefit from more powerful local planners such as the potential field local planner. Because the local planner is slower, the time improvement in general is less dramatic than the improvement in number of regions required. However, we believe that one could improve the time even further.

In future work, one could investigate techniques that can identify difficult regions in the space. This information can be used to select the most appropriate local planner which should lead to an even better performance of the PRM.

3.7 Discussion

While theoretically, the complexity of the motion planning problem is exponential in the number of degrees of freedom, sampling-based planners can successfully handle this curse of dimensionality in practice because they are

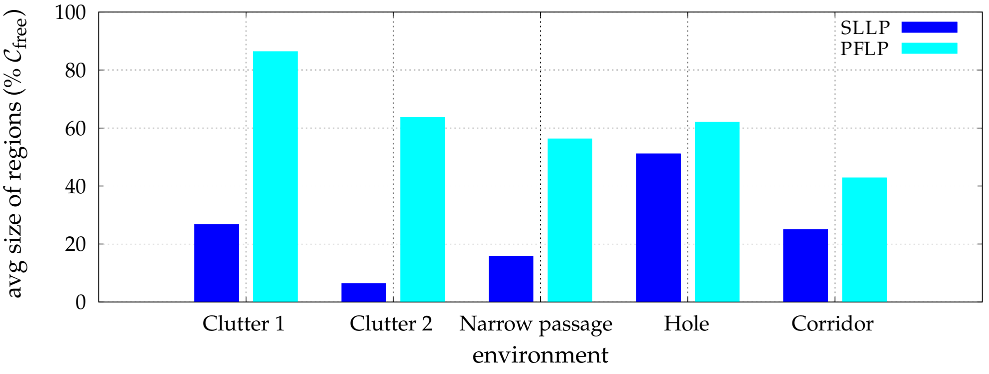
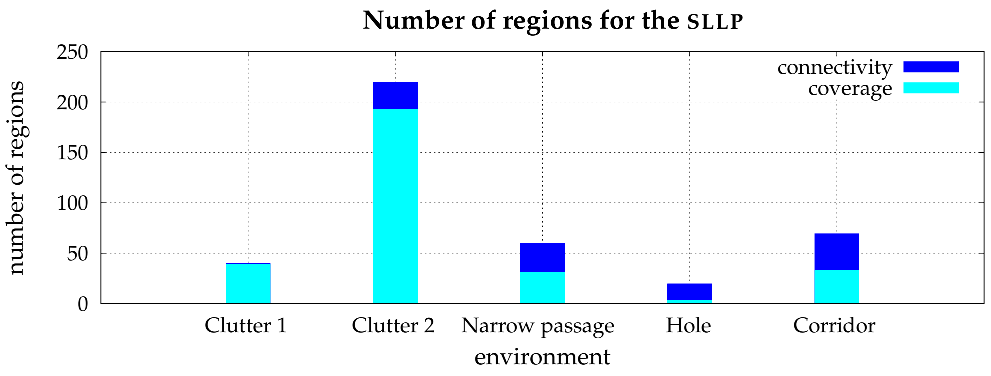


Figure 3.12 The average size of the regions corresponding to the two local planners. The maximum connection distance has been to set to infinity.

reachability-based. We presented a reachability analysis for these planners which focused on coverage and maximal connectivity of the free configuration space $\mathcal{C}_{\text{free}}$. By inspecting the roadmap when $\mathcal{C}_{\text{free}}$ was covered and when $\mathcal{C}_{\text{free}}$ was maximally connected, we obtained a better understanding of these planners. This led to the insight that not coverage is the main problem but getting the nodes connected, especially when the problems get more complicated, i.e. a narrow passage is present. The narrow passage problem can be tackled by incorporating a hybrid sampling strategy that aims at concentrating samples in difficult areas. The strategy must also generate some samples in large open areas. Other strategies to get the free configuration space faster connected are to use a refined neighbor selection strategy which is able to make long and many connections and a more powerful local planner. We presented a potential field local planner that creates larger reachability regions and accordingly eases making connections. This planner is also better able to find the entry of a narrow passage, decreasing the number of regions needed to get the nodes connected. Experiments showed that this approach leads to a better performance of sampling-based methods.

In Chapter 6, we will use a reachability analysis of $\mathcal{C}_{\text{free}}$ to obtain a more clever sampling and connection strategy. This results in very small roadmaps that satisfy the coverage as well as the maximal connectivity criteria.



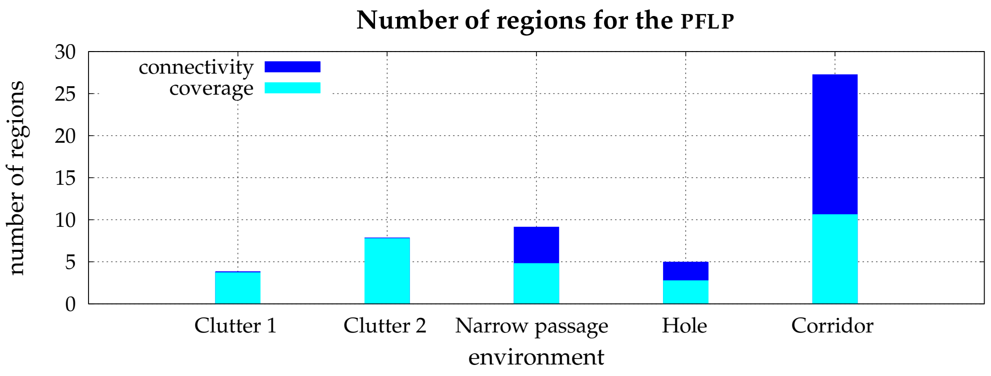


Figure 3.13 Influence of the two local planners on the number of regions. Note the different scales.

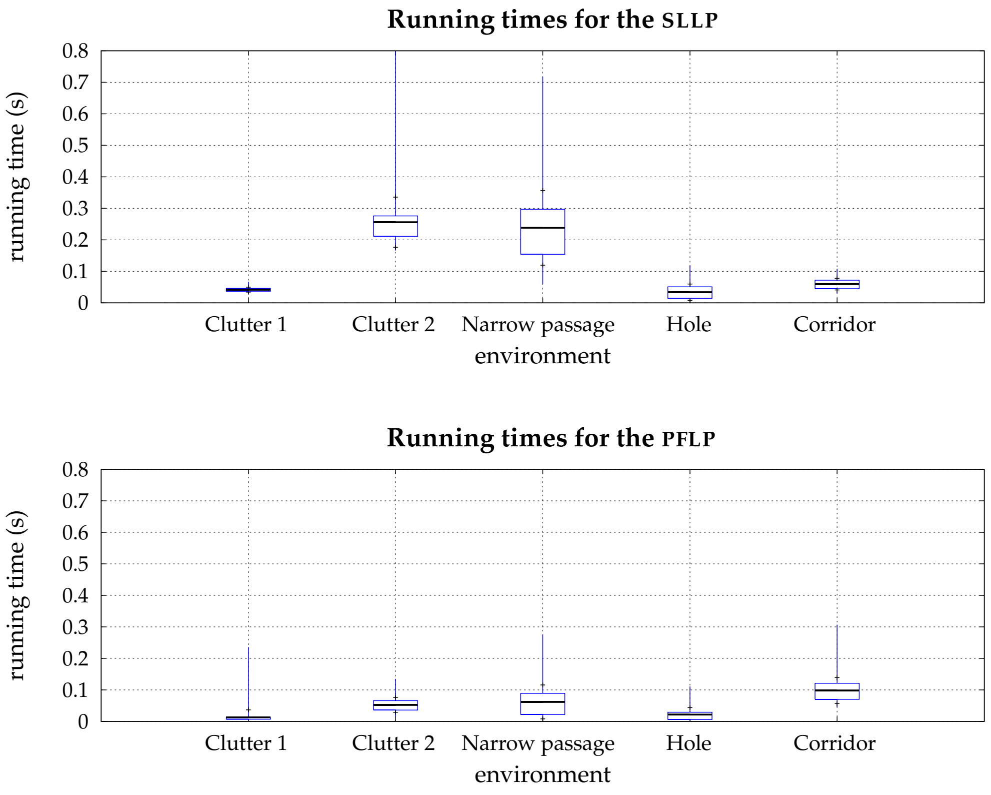


Figure 3.14 Influence of the two local planners on the running time.

PART II

PATH QUALITY

CHAPTER
FOUR

INCREASING PATH CLEARANCE

Many algorithms have been proposed that create a path for a robot in an environment with obstacles. Since it can be hard to create such a path, most methods are aimed at finding *a* solution. However, for many applications, the path must be of a good quality as well. That is, a path should preferably be *short* because redundant motions will take longer to execute. In addition, the path also has to keep some amount of minimum *clearance* to the obstacles because it can be difficult to measure and control the precise position of a robot. Traveling along such a path reduces the chances of collisions due to these uncertainties.

In this chapter, we will study two algorithms that increase the clearance along paths. The first one is fast but can only deal with rigid, translating bodies. The second algorithm is slower but can handle a broader range of robots, including three-dimensional free-flying and articulated robots, which may reside in arbitrary high-dimensional configuration spaces. A big advantage of these algorithms is that clearance along paths can now be increased efficiently without using complex data structures and algorithms.

4.1 Introduction

Algorithms that produce paths with high clearance can be divided into two categories. The first category creates a roadmap (or graph) which represents the high-clearance collision-free motions that can be made by the moving object in an environment with obstacles. From this graph a path is extracted by a Dijkstra's shortest path algorithm. Since the calculations to create the high-clearance paths are performed off-line, we refer to this technique as a preprocessing approach. The second category optimizes a given path. The optimization is usually performed on-line in a post-processing stage.

The Generalized Voronoi Diagram (GVD) is a roadmap which can be used to extract high-clearance paths. The GVD (or medial axis) for a robot with n degrees of freedom is defined as the collection of k-dimensional geometric features ($0 \leq k < n$) which are $(n+1-k)$-equidistant to the obstacles. As an example, consider Figure 4.1 that shows a bounding box and a part of the medial axis for a translating robot. The medial axis of this robot consists of a collection of surfaces, curves and points. The surfaces are defined by the locus of 2-equidistant closest points to the bounding box. The curves have 3-equidistant closest points and the points have 4-equidistant closest points to the bounding box. These features are connected if the free space in which the robot operates is also connected [35]. Hence, the GVD is a complete representation for motion planning purposes. Most importantly, paths on the GVD have appealing properties such as large clearance from obstacles.

Halperin *et al.* [155] introduce a hybrid between the visibility graph and the Voronoi diagram of polygons in the plane. A shortest path with a preferred minimum amount of clearance can be extracted in real-time.

Unfortunately, an exact computation of the GVD is not practical for problems involving many degrees of freedom (DOFs) and many obstacles as this requires an expensive and intricate computation of the configuration space obstacles. Therefore, the GVD is approximated in practice.

Vleugels and Overmars [150] approximate the GVD by applying spatial subdivision and isosurface extraction techniques. Although the calculations are easy and robust, and they can be generalized to higher dimensions, the technique only works for disjoint convex sites and consumes an exponential amount of memory, making this technique impractical for problems involving many DOFs. Another approach, proposed by Masehianand *et al.* [111], incrementally constructs the GVD by finding the maximal inscribed disks in a two dimensional discretized workspace. Although this algorithm is also extensible to handle higher-dimensional problems, it suffers from the same drawback as the preceding algorithm. Hoff *et al.* [67] describe a technique that exploits the fast computation of a GVD using graphics hardware for motion planning in complex static and dynamic environments. However, the technique is limited to a three-dimensional workspace for rigid translating robots.

Kim *et al.* [88] use an augmented version of Dijkstra's algorithm to extract a path from a graph based on other criteria than length. The minimum clearance along the path is maximized by incorporating a higher cost for edges that represent a small amount of clearance. Such a path rarely provides an optimal solution because it is restricted to the randomly generated nodes in the roadmap. Even if the nodes are placed on the *medial axis* [107], the edges will in general not lie on the medial axis, and hence, the extracted path does not have an optimal amount of clearance.

The above preprocessing methods create a data structure from which paths can be extracted. Brock and Khatib [26] present a (post-processing) framework that provides an efficient method for performing local adjustments to a path in dynamic environments. This path is represented as an elastic band. Subjected to artificial forces, the elastic band deforms in real-time to a short and smooth path that maintains clearance from the obstacles. The method can be applied to a broad range of robots, but many parameters have to be set to get the framework running. It is also not clear whether the resulting path will and can have an optimal amount of clearance.

In this chapter, we will study techniques to improve the clearance along a given path. Later, in Chapter 7, we will discuss a preprocessing approach in which a roadmap with high-clearance paths is computed. First, we will provide some preliminaries in Section 4.2. In Section 4.3, we describe our first algorithm that adds clearance to a path by retracting it to the medial axis of the workspace. The algorithm is limited to translating, rigid bodies. Although it provides optimal clearance paths for rigid, translating bodies in the plane, the paths may not be optimal for robots operating in a three-dimensional environment. We remove these limitations in Section 4.4 where an algorithm is presented that provides high-clearance paths for a broad range of robots residing in arbitrary high-dimensional configuration spaces. We apply these algorithms to six different environments in Section 4.5 and conclude in Section 4.6 that clearance along paths can be increased without using complex data structures and algorithms. The results can be used, for example, to obtain high-clearance paths in high-cost environments such as a factory in which a manipulator arm operates.

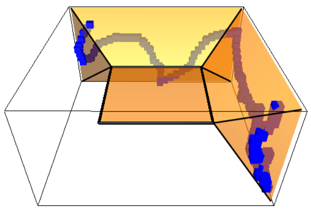


Figure 4.1 A part of the medial axis of an environment that only consists of a bounding box.

4.2 Preliminaries

As this chapter deals with improving the robot's clearance along a path, we need a way to compute the clearance of the robot to the obstacles. This calculation is delegated to Solid (see Section 1.2.2). We define the clearance of a configuration as follows:

Definition 4.1 (Clearance of a configuration π). *Let R be the set of all points on the robot whose placement in the environment corresponds to configuration π. Furthermore, let O be the set of all points on all obstacles in the environment. Then the clearance of configuration π is the Euclidean distance* $\min d(r, o) : r \in R \wedge o \in O$.

To define a path, we need the following definition of adjacent configurations.

Definition 4.2 (Adjacent configurations). *The configurations $\pi_0, \cdots, \pi_{n-1}$ are adjacent if the distance $d(\pi_i, \pi_{i+1})$ is at most a predetermined distance* step.

The step size is chosen dependent on the robot and obstacles. The reader is referred to Section 1.2.3 for details on computing distances. We can now define a *discrete path* and a *discrete local path*.

Definition 4.3 (Discrete path Π). *A* discrete path Π *is a series of n adjacent configurations $\pi_0, \cdots, \pi_{n-1}$.*

Definition 4.4 (Discrete local path LP). *A* discrete local path LP$[\pi', \pi'']$ *is a series of n interpolated adjacent configurations $\pi_0, \cdots, \pi_{n-1}$ on the local path between π' and π''.*

The average clearance of a path gives an indication of the amount of free space in which the path can be moved without colliding with the obstacles:

Definition 4.5 (Average clearance of discrete path Π). *Let Π be a discrete path. Then the* average clearance *equals to* $\frac{1}{n} \sum_{i=0}^{n-1} Clearance(\pi_i)$.

4.3 Rigid, translating bodies

In this section, we describe an algorithm that adds clearance to a path traversed by a translating, rigid body. The problem we want to solve is as follows. Convert a given discrete path Π into a path Π' such that each robot placement that corresponds to $\pi'_i \in \Pi'$ has (at least) two-equidistant nearest points to the obstacles in the scene. We initially assume that the start and goal configurations lie on the medial axis.

We will increase the clearance along a path by retracting its individual placements of the robot (which we refer to as *samples*) to the medial axis of the free workspace. Our approach is based on a technique from Wilmarth *et al.* [156] which retracts samples to the medial axis.[1] Such a retracted sample will have (at least) two-equidistant nearest points to the obstacles in the workspace, resulting in a large clearance. As the retraction is performed in the workspace, only the clearance along paths traversed by translating, rigid bodies can be improved.

4.3.1 Retraction algorithm

We will first show how to retract a single sample, corresponding to configuration $\pi \in \Pi$, to the medial axis. Algorithm 4.1 outlines our approach. Let cp_π be the point on the robot in the workspace that corresponds to configuration π that is closest to the point cp_o on an obstacle in the workspace. We first calculate the pair (cp_π, cp_o) of closest points between the robot and obstacles. Then, we iteratively move in direction $\overrightarrow{cp_o cp_\pi}$ until the closest point on the obstacles changes. In each iteration, the largest distance it can move such that the robot will not collide with the obstacles equals its clearance which is defined as the Euclidean distance between cp_π and cp_o. Finally, we use binary search between the original closest point cp_o and changed closest point $cp_{o'}$ (with precision $step$) to find the configuration π_{mid} that has two-equidistant nearest points to the obstacles in the workspace.

Algorithm 4.2 shows how to retract a discrete path Π to the medial axis. We retract each configuration $\pi \in \Pi$ to the medial axis. If the distance between two consecutive configurations of the retracted path Π' exceeds $step$, we generate extra configurations by applying the algorithm onto the local path that is defined by these two configurations until the distance between any two consecutive configurations is less than $step$.

[1]While their technique retracts single samples to the medial axis, our technique retracts a complete path.

Algorithm 4.1 RETRACTCONFIGURATION(configuration π)

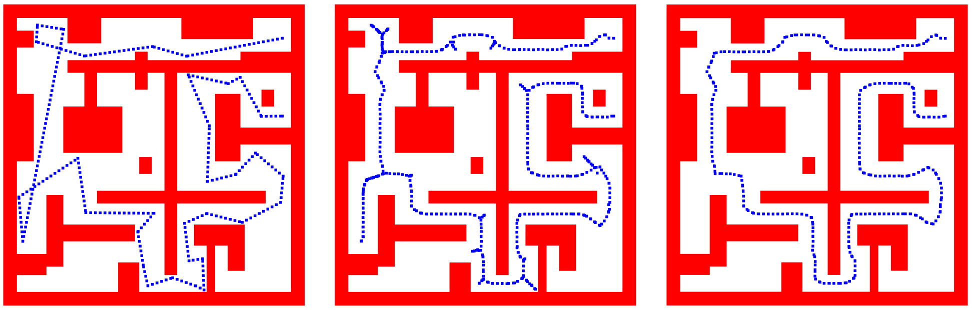
```
Require: free configuration $\pi$, obstacles $O$, precision $step$
1:  $(cp_\pi, cp_o) \leftarrow$ CLOSESTPAIR$(\pi, O)$
2:  $cp_{o'} \leftarrow cp_o$
3:  while $cp_{o'} = cp_o$ do
4:      $\pi' \leftarrow \pi$
5:      $\pi \leftarrow \pi + cp_\pi - cp_o$
6:      $(cp_\pi, cp_{o'}) \leftarrow$ CLOSESTPAIR$(\pi, O)$
7:  while $d(\pi, \pi') > step$ do
8:      $\pi_{mid} \leftarrow$ INTERPOLATE$(\pi, \pi', 0.5)$
9:      $(cp_\pi, cp_o) \leftarrow$ CLOSESTPAIR$(\pi_{mid}, O)$
10:     if $cp_{o'} = cp_o$ then $\pi \leftarrow \pi_{mid}$ else $\pi' \leftarrow \pi_{mid}$
11: return $\pi_{mid}$
```

Algorithm 4.2 $\mathcal{W}$-RETRACTION(discrete path Π)

```
1: retracted path $\Pi' \leftarrow \emptyset$
2: for all $\pi_i \in \Pi, 0 \leq i < n$ do
3:     $\pi' \leftarrow$ RETRACTCONFIGURATION$(\pi_i)$
4:     $\pi_r \leftarrow$ the last configuration of path $\Pi'$
5:     if $d(\pi_r, \pi') > step$ then
6:         $\Pi' \leftarrow \Pi' \cup \mathcal{W}$-RETRACTION(LP $[\pi_r, \pi']$)
7:     $\Pi' \leftarrow \Pi' \cup \pi'$
8: return $\Pi'$
```

Algorithm 4.3 REMOVEBRANCHES(discrete path Π)

```
1: $i \leftarrow 1$
2: while $i < |\Pi| - 1$ do
3:     if $d(\pi_{i-1}, \pi_{i+1}) < step$ then
4:         $\Pi \leftarrow \Pi \setminus \pi_i$
5:         if $i > 1$ then $i \leftarrow i - 1$
6:     else $i \leftarrow i + 1$
7: return $\Pi$
```



(a) Original path (b) Retracted path (c) Removed branches

Figure 4.2 Retraction of a path traversed by a square robot in a 2D workspace. Picture (a) shows the query path. In (b), this path has been retracted to the medial axis of the workspace. In (c), its branches have been removed.

Algorithm 4.2 will only work correctly when the start configuration π_0 and/or goal configuration π_{n-1} lie on the medial axis. If not, the retracted path is concatenated with the local path $\text{LP}[\pi_0, \pi'_0]$ and/or local path $\text{LP}[\pi'_{n-1}, \pi_{n-1}]$. The path will now follow the medial axis. As an example, we applied the algorithm on a square translating in a 2D environment. We took this environment from Lavalle's Motion Strategy Library [100]. See Figure 4.2. The first picture shows the original path. The retracted path is visualized in the second picture. As we can see, the moving object sometimes traverses the same position twice. This detour is caused by the injective mapping of configurations and can be detected by looking for reversals in a sub-branch of the path. Algorithm 4.3 removes those redundant branches in linear time in $|\Pi|$. For each triple $\{\pi_{i-1}, \pi_i, \pi_{i+1}\}$, we remove π_i if the distance between π_{i-1} and π_{i+1} is smaller than *step*. Figure 4.2(c) shows the resulting path following the medial axis without traversing a sub-branch twice. This path was computed within one second.

4.4 Robots with many degrees of freedom

The retraction method from the previous section provides an accurate retraction of paths for rigid, translating bodies to the medial axis. As the retraction is performed by a series of translations of the robot, the method is not suitable for increasing the clearance along a path traversed by an articulated robot or a free-flying robot for which the rotational DOFs are important for a solution

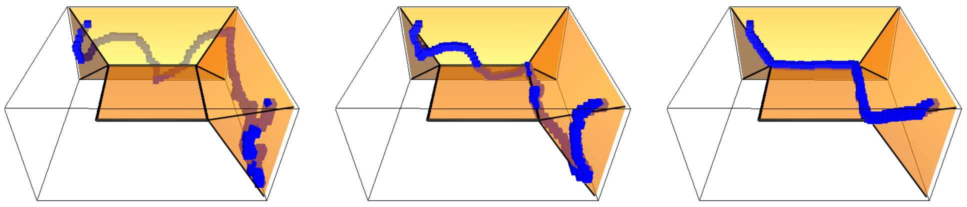

(a) Initial path (b) Retracted path (c) Optimal path

Figure 4.3 Retraction of a path in a 3D environment that only consists of a bounding box. A part of the medial axis inside this box is shown. Figure (a) shows the initial path. This path is retracted to the medial axis by Algorithm 4.2. Figure (c) shows a path having a larger amount of average clearance. This path was obtained by Algorithm 4.4.

of the problem. In addition, the method will in general not produce a maximal clearance path because the retraction is completed when the samples are placed somewhere on the medial axis. Many samples could have had a larger clearance if they were further retracted toward configurations representing a higher clearance. See Figure 4.3 for an example. The crooked path from Figure 4.3(a) was retracted to the medial axis by the algorithm from the previous section. Figure 4.3(b) shows that the retracted samples sway on the medial axis surfaces. In Figure 4.3(c), the path has obtained a larger amount of clearance.

4.4.1 Retraction algorithm

Our new retraction algorithm attempts to iteratively increase the clearance of the configurations on the path by moving them in a direction for which the clearance is higher. The problem we want to solve is as follows. Convert a given path Π into a path Π' such that for each $\pi'_i \in \Pi'$ the clearance is locally maximal wherever possible. A configuration represents a locally maximum clearance when there is no direction in which the clearance is larger. Algorithm 4.4 outlines our approach. Globally speaking, our solution consists of several iterations. In each iteration, we choose a random direction *dir* which incorporates all DOFs.[2] Then, we try to move each configuration π_i in the chosen direction, i.e. $\pi'_i \leftarrow \pi_i \oplus dir$. (The operator $\oplus$ will be defined below.) If the clearance of π'_i is larger than the clearance of π_i, then π_i is replaced by π'_i. We stop retracting the path when the *average* clearance of the path (see Definition 4.5) does not improve anymore.

[2]We use a random direction because alternative choices will require too many time-consuming distance calculations (such as moving in the direction of the steepest descent).

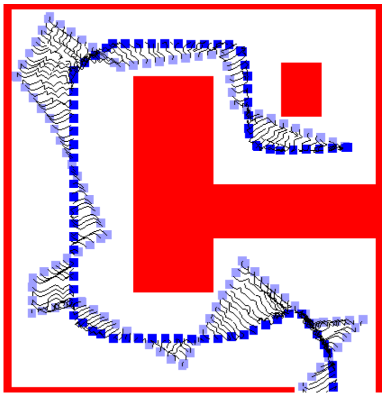


Figure 4.4 An impression of the retraction algorithm. The algorithm retracts the initial path (traversed by a square robot) to the medial axis. For each configuration in the discrete path, the guided random walk (small curve) is drawn.

By updating the configurations, the path is forced to stretch and shrink during the retraction which causes the following two problems. First, the distance between two adjacent configurations in the path can become *larger* than the maximum step size. This happens for example when the path is pushed away from the obstacles. If this occurs, we insert an appropriate configuration between them. Second, several configurations can be mapped to a small region in which the distance between two non-adjacent configurations is *smaller* than the step size. This occurs for example when pieces of the path are traversed twice. As we have seen in the previous section, they can be removed easily.

An impression of a retraction is given in Figure 4.4. This figure shows an initial and a discrete retracted path traversed by a square robot in a simple two-dimensional workspace. The line segments between the paths are the guided random walks of the configurations. We call these walks *guided* because a configuration is updated only if its clearance increases. Note (by close inspection) that extra configurations have been inserted at some places while configurations have been removed at other places. After 40 iterations, the initial path has been successfully retracted to the medial axis, resulting in a path with large clearance. Although this example shows a retraction for a robot with only two DOFs, the retraction can also be applied to robots with more DOFs such as an articulated robot with six joints.

Algorithm 4.4 $\mathcal{C}$-RETRACTION(discrete path Π)

```
1: loop
2:   $\Pi' \leftarrow \Pi$
3:   $dir \leftarrow$ RANDOMDIRECTION($step$)
4:   for all $\pi'_i$ in $\Pi'$ do
5:     $\pi_{new} \leftarrow \pi'_i \oplus dir$
6:     if CLEARANCE($\pi_{new}$) > CLEARANCE($\pi'_i$) then
7:       $\pi'_i \leftarrow \pi_{new}$
8:   $\Pi \leftarrow$ VALIDATEPATH($\Pi, \Pi'$)
9: return $\Pi$
```

4.4.2 Algorithmic details

A discrete path consists of a series of configurations. We require that the distance between each pair of adjacent configurations is at most *step*. Distances are computed by the distance metric from Section 1.2.3. That is, the distance between configurations q and r is calculated by summing the weighted partial distances for each DOF $0 \leq i < n$ that describes the configurations; remember that we distinguish three types of DOFs – translation, rotation$_1$ and rotation$_3$, so

$$d(q,r) = \sqrt{\sum_{i=0}^{n-1} [w_i d(q_i, r_i)]^2}.$$

The clearance of the configurations is improved by iteratively moving them in a random direction. We will show how to compute such a direction and how to add this direction to a configuration. After an iteration of the algorithm, the distance between two adjacent configurations may have changed. We will show how to insert and delete appropriate configurations to maintain a valid path. Finally, we discuss how to choose an appropriate termination criterion for the algorithm.

Random direction vector

Our goal is to create a random direction q' such that the distance from configuration q to $q \oplus q'$ equals *step*. The direction q' is composed of values for each DOF q_i such that $\sqrt{\sum_{i=0}^{n-1} p_i^2} = step$, where $p_i = w_i d(q_i, q_i \oplus q'_i)$. This expression shows how much each DOF i contributes to the total distance. Let *rnd* be a vector of random values between 0 and 1 such that $\sum_i^{n-1} rnd_i = 1$, and let rnd_i be the random value for DOF i. Furthermore, let $w = (w_0, \cdots w_{n-1})$ be the weight vector. Theorem 4.1 shows that the translational and rotational$_1$ components of q' must be set to $q'_i = \pm\frac{rnd_i * step}{\sqrt{rnd \cdot w}}$. The calculation of the rotational$_3$ component is more complicated. We represent this component as a random 3D unit axis $a = (a_x, a_y, a_z)$ and an angle of revolution θ about that axis. (Since a revolution about a random axis of more than π radians is redundant, we constrain θ to $0 \leq \theta \leq \pi$.) This representation can easily be converted to a quaternion, i.e. $q'_i = (a_x \sin(\theta/2), a_y \sin(\theta/2), a_z \sin(\theta/2), \cos(\theta/2))$. Theorem 4.1 shows that θ must be set to $\frac{rnd_i * step}{\sqrt{rnd \cdot w}}$.

Lemma 4.1. *If the distances $d(q_i, q_i \oplus q'_i)$ are set to $\pm\frac{rnd_i * step}{\sqrt{rnd \cdot w}}$, then $d(q, q \oplus q') = step$.*

Proof: The distance between q and $q \oplus q'$ is defined as $\sqrt{\sum_{i=0}^{n-1} p_i^2}$, where $p_i = w_i d(q_i, q_i \oplus q'_i)$. When setting $d(q_i, q_i \oplus q'_i)$ to $\pm\frac{rnd_i * step}{\sqrt{rnd \cdot w}}$, we must prove that $\sqrt{\sum_{i=0}^{n-1} p_i^2} = step$. By definition, we have

$$d(q, q \oplus q') = \sqrt{\sum_{i=0}^{n-1} \Big(w_i d(q_i, q_i \oplus q'_i)\Big)^2}.$$

By substitution, we get

$$d(q, q \oplus q') = \sqrt{\sum_{i=0}^{n-1} \Big(\pm \frac{rnd_i * w_i * step}{\sqrt{rnd \cdot w}}\Big)^2},$$

and hence,

$$d(q, q \oplus q') = step * \sqrt{\sum_{i=0}^{n-1} \Big(\pm \frac{rnd_i * w_i}{\sqrt{rnd \cdot w}}\Big)^2} = step * \sqrt{\sum_{i=0}^{n-1} \frac{(rnd_i * w_i)^2}{rnd \cdot w}}.$$

Since $rnd \cdot w = \sum_{j=0}^{n-1} (rnd_j * w_j)^2$, we get

$$d(q, q \oplus q') = step * \sqrt{\sum_{i=0}^{n-1} \frac{(rnd_i * w_i)^2}{\sum_{j=0}^{n-1} (rnd_j * w_j)^2}},$$

and hence,

$$d(q, q \oplus q') = step.$$

■

Lemma 4.2. *Let q and q' be two rotational$_1$ values. If the range of angle q' is set to $-\pi \leq q' \leq \pi$, then $d(q, q \oplus q') = |q'|$.*

Proof: The distance between two rotational$_1$ values q and r is defined as

$$d(q,r) = \min\{|q-r|, q-r+2\pi, r-q+2\pi\}.$$

Let $r = q \oplus q'$. Then,

$$d(q, q \oplus q') = \min\{|q-(q+q')|, q-(q+q')+2\pi, (q+q')-q+2\pi\}.$$

By substitution, we get

$$d(q, q \oplus q') = \min\{|q'|, 2\pi - q', q' + 2\pi\}.$$

Because $-\pi \leq q' \leq \pi$, it holds that $2\pi - q' \geq \pi$ and $q' + 2\pi \geq \pi$. Since $q' \leq \pi$, the minimum is determined by $|q'|$. Hence,

$$d(q, q \oplus q') = |q'|.$$

■

Lemma 4.3. *Let q and q' be two quaternions. The quaternion q' represents a random (unit) axis and an angle of revolution θ about that axis. If the range of θ is set to $0 \leq \theta \leq \pi$, then the distance $d(q, q' * q) = \theta$.*

Proof: The distance between two quaternions q and r is defined as $d(q,r) = \min\{2\arccos(q \cdot r), 2\pi - 2\arccos(q \cdot r)\}$. Since θ is positive, the dot product $q \cdot r$ is also positive and lies between 0 and 1. As a consequence, the arccos of the dot product will lie between 0 and π. Hence, the distance between q and r equals to $d(q,r) = 2\arccos(q \cdot r)$.

Let $q = q' * r$. The quaternion q' represents a rotation by θ around a unit axis $a = (a_x, a_y, a_z)$, i.e. $q' = (a_x \sin(\theta/2), a_y \sin(\theta/2), a_z \sin(\theta/2), \cos(\theta/2))$. Hence, we get

$$d(q, q' * q) = 2\arccos((q' * r) \cdot r).$$

After substitution, this can be shown to be equivalent to

$$d(q, q' * q) = 2\arccos(((r \cdot r)\cos(\frac{\theta}{2})).$$

The length of a quaternion that represents a rotation is always equal to 1. Hence, $r \cdot r = 1$. By using $0 \leq \theta \leq \pi$ and substitution, we get

$$d(q, q' * q) = \theta.$$

■

Theorem 4.1. *By setting the translational and rotational$_1$ components of q' to $q'_i = \pm\frac{rnd_i \,*\, step}{\sqrt{rnd \cdot w}}$ and the rotational$_3$ components of q' to $q'_i = (a_x \sin(\theta/2), a_y \sin(\theta/2), a_z \sin(\theta/2), \cos(\theta/2))$, where $a = (a_x, a_y, a_z)$ is a random unit axis and angle $\theta = \frac{rnd_i \,*\, step}{\sqrt{rnd \cdot w}}$, it holds that $d(q, q \oplus q') = step$.*

Proof: The distance between two translational values q_i and $q_i \oplus q'_i$ equals

$$d(q_i, q_i \oplus q'_i) = d(q_i, q_i + q'_i) = |q_i - (q_i + q'_i)| = |q'_i|.$$

Furthermore, Lemma 4.2 showed that the distance between two rotational$_1$ values q_i and $q_i \oplus q'_i$ equals

$$d(q_i, q_i \oplus q'_i) = |q'_i|.$$

As we set q'_i to

$$q'_i = \pm\frac{rnd_i \,*\, step}{\sqrt{rnd \cdot w}},$$

Lemma 4.1 implies that

$$d(q, q \oplus q') = step.$$

Finally, Lemma 4.3 showed that the distance between two quaternions q_i and $q_i \oplus q'_i$ equals

$$d(q_i, q_i \oplus q'_i) = d(q_i, q'_i * q_i) = \theta,$$

where $q'_i = (a_x \sin(\theta/2), a_y \sin(\theta/2), a_z \sin(\theta/2), \cos(\theta/2))$ and $a = (a_x, a_y, a_z)$ is a random unit axis. By setting θ to

$$\theta = \frac{rnd_i \,*\, step}{\sqrt{rnd \cdot w}},$$

Lemma 4.1 implies that

$$d(q, q \oplus q') = step.$$

■

Besides choosing a random vector, we need to add a direction q' to configuration q. For translational and rotational$_1$ DOFs, we add up the values. If the rotational$_1$ DOF is periodic, we have to make sure that the value remains in the range between 0 and 2π. For the rotational$_3$ DOF, q'_i is multiplied by q_i.

Algorithm 4.5 VALIDATEPATH(discrete path Π, discrete path Π')

```
 1: $i \leftarrow 0$
 2: valid path $\Pi'' \leftarrow \emptyset$
 3: **while** $i < |\Pi| - 1$ **do**
 4:    $\Pi'' \leftarrow \Pi'' \cup \pi'_i$
 5:    **if** $d(\pi'_i, \pi'_{i+1}) > step$ **then**
 6:       $\pi'_{int} \leftarrow$ INTERPOLATE$(\pi'_i, \pi'_{i+1}, 0.5)$
 7:       **if** CLEARANCE$(\pi'_{int}) >$ CLEARANCE$(\pi_{i+1})$ **then**
 8:          $\Pi'' \leftarrow \Pi'' \cup \pi'_{int}$
 9:       **else**
10:          $\Pi'' \leftarrow \Pi'' \cup \pi_{i+1}$
11:    $i \leftarrow i + 1$
12: $\Pi'' \leftarrow \Pi'' \cup \pi'_i$
13: $\Pi'' \leftarrow$ REMOVEBRANCHES$(\Pi'')$
14: **return** $\Pi''$
```

Path validation

As a path is forced to stretch and shrink during the retraction, the path may not be valid anymore after an iteration of Algorithm 4.4. A discrete path Π is valid if $\forall i : d(\pi_i, \pi_{i+1}) \leq step$. In this section we will show how to construct a new valid path. Algorithm 4.5 outlines our approach.

Let Π be the original path and Π' be the updated path. Furthermore, let π_i be the i^{th} configuration on path Π and π'_i be the corresponding (possibly updated) configuration on path Π'. We construct a new path Π'' which will initially contain all configurations from Π' and possibly new configurations to assure that Π'' is valid.

In each iteration of the loop, we concatenate configuration π'_i to the valid path Π''. Then we check whether the distance between two adjacent configurations on the updated path Π' is larger than the step size, i.e. we check if $d(\pi'_i, \pi'_{i+1}) > step$. If this condition is true, we have to add an extra configuration to path Π'' to assure that Π'' keeps valid. We consider two candidate configurations and choose the one that has the largest clearance. The first one is the original configuration π_{i+1} and the second one is created by interpolating halfway between configurations π'_i and π'_{i+1}. The reader is referred to Section 1.2.4 for details on interpolation.

After the iterations, we add the last configuration of path Π' to the valid path Π''. Finally, to remove superfluous configurations, we apply Algorithm 4.3 on path Π''. Recall that this algorithm removes a configuration π''_i from path Π'' for which it holds that $d(\pi''_{i-1}, \pi''_{i+1}) \leq step$.

Theorem 4.2. *Algorithm 4.5 assures that discrete path* Π'' *is valid.*

Proof: The input of the algorithm is a valid path Π and a possibly invalid path Π'. We have to prove that the algorithm creates a path Π'' such that $\forall i$: $d(\pi''_i, \pi''_{i+1}) \leq step$.

Lines 4 and 12 imply that path Π'' will contain each configuration of the updated path Π'. The maximum distance between two adjacent configurations π'_i and π'_{i+1} of path Π' is $2 * step$ which occurs when one of them is updated while the other one is left unchanged. (Note that when both configurations are updated, their relative distance remains the same, and hence, they do not cause the path to be invalid.) We insert one of the following two configurations to path Π''. The first candidate, π'_{int}, is the configuration in the middle of the straight-line in $\mathcal{C}_{\text{free}}$ between π'_i and π'_{i+1}. As the distance between π'_i and π'_{i+1} is halved, $d(\pi'_i, \pi'_{int}) \leq step$ and $d(\pi'_{int}, \pi'_{i+1}) \leq step$. The second candidate is configuration π_{i+1}. It holds that $d(\pi'_i, \pi_{i+1}) \leq step$ and $d(\pi_{i+1}, \pi'_{i+1}) \leq step$. As path Π'' contains the sequence π'_i, the candidate configuration, and π'_{i+1}, path Π'' will remain valid. Finally, as Algorithm REMOVEBRANCHES only removes a configuration π''_i when $d(\pi''_{i-1}, \pi''_{i+1}) < step$, it will not invalidate the path. Hence, Algorithm 4.5 constructs a valid path Π''. ∎

Termination criterion

An important issue is when to terminate the algorithm. In each iteration of the algorithm, we only update a configuration if its clearance increases. Such an update can lead to insertions and deletions of configurations. If a configuration π is inserted, then the clearance of π will be equal to or higher than the clearance of the configuration before it was updated. If a configuration π' is deleted, it will not play a role anymore. However, π' could have a high clearance while a possibly inserted configuration could have a low clearance. Hence, while each configuration can obtain a higher clearance, the average clearance can actually decrease. As this worst-case scenario may occur incidentally, we have to take this into account in our termination criterion.

We terminate the algorithm when the improvement of the average clearance in k consecutive iterations is smaller than some small threshold δ. We conducted experiments to find appropriate values for these parameters. We observed that setting k to 25 and δ to $step/10$ led to mature convergence.

4.5 Experiments

In this section, we will investigate the extent to which the $\mathcal{W}$-retraction and $\mathcal{C}$-retraction algorithms can improve the clearance along six paths.

4.5.1 Experimental setup

We considered the environments and their corresponding paths depicted in Figure 4.5. They have the following properties (see Table 4.1 for their dimensions):

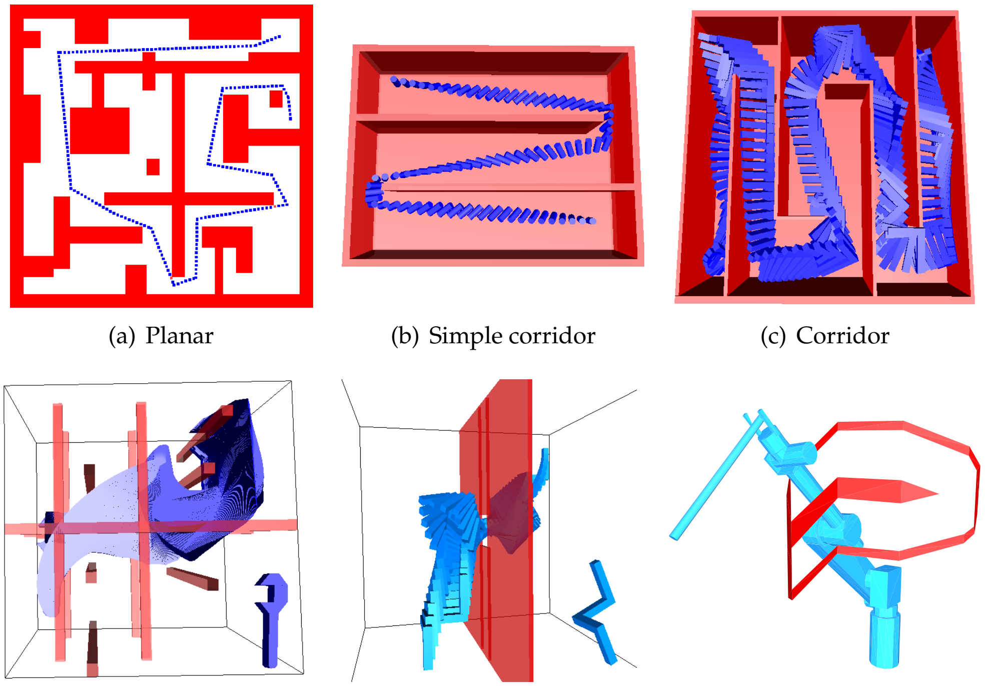
Planar This simple two-dimensional environment contains a path traversed by a square robot that can only translate in the plane. As the robot has two translational DOFs, a retraction in the workspace will result in a path having the optimal amount of clearance. The experiments will show whether a retraction in the $\mathcal{C}$-space is competitive.

Simple corridor This simple three-dimensional environment with ample free space to maneuver features a path traversed by a small free-flying cylinder. Both algorithms will introduce an extra amount of clearance as they both move the robot to the middle of the corridors. However, the $\mathcal{C}$-retraction algorithm should outperform the $\mathcal{W}$-retraction algorithm as it also considers rotational DOFs.

Corridor The environment consists of a winding corridor that forces a free-flying elbow-shaped robot to rotate. As there is little room between the walls of the corridor and the robot, it may be hard to increase the clearance along the path.

Wrench This environment features a fairly large free-flying object (wrench) in a workspace that consists of thirteen crossing beams. The wrench is rather constrained at the start and goal positions. We expect that the $\mathcal{W}$-retraction algorithm will be outperformed by the $\mathcal{C}$-retraction algorithm as the rotational DOFs are of major concern in this environment.

Hole The free-flying robot, which has six DOFs (three translational DOFs and a rotation$_3$ DOF), consists of four legs and must rotate in a complicated way to get through the hole. Only where the robot passes through the hole, the clearance is small. Hence, the improvement of the minimum amount of clearance along the path shows the potential of the $\mathcal{C}$-retraction algorithm.


(a) Planar (b) Simple corridor (c) Corridor

(d) Wrench (e) Hole (f) Manipulator

Figure 4.5 The six test environments and their corresponding initial paths. For the Wrench and Hole environments, the robot has been depicted at the lower right.

Manipulator The articulated robot has six rotational DOFs and operates in a constrained environment. The clearance along the path is very small. The $\mathcal{W}$-retraction algorithm cannot be applied as it cannot handle rotational DOFs. Again, an increase in the minimum and average amounts of clearance along the path will show the potential of the $\mathcal{C}$-retraction algorithm.

We subdivided each path in consecutive configurations such that the distance between each two adjacent configurations is at most some predetermined distance *step*. The step sizes for the paths can be found in Table 4.2. The distance metric from Section 1.2.3 uses weights w_i for the DOFs of a robot. These are listed in Table 4.3.

In each run, we recorded the minimum, maximum and average clearance of the path. As the $\mathcal{C}$-retraction algorithm is non-deterministic, we ran this algorithm 100 times for each experiment and calculated the averages. Each run was terminated when the improvement of the average clearance in 25 consecutive iterations was smaller than some small threshold, $step/10$.

	Dimensions of the bounding box	
	environment	**robot**
Planar	100×100	1×1
Simple corridor	$40 \times 11 \times 30$	$0.2 \times 0.2 \times 0.75$
Corridor	$40 \times 17 \times 40$	$5 \times 1 \times 5$
Wrench	$160 \times 160 \times 160$	$68 \times 24 \times 8$
Hole	$40 \times 40 \times 40$	$5 \times 5 \times 10$
Manipulator	$10 \times 10 \times 10$	variable

Table 4.1 The axis-aligned bounding boxes of the environments and robots.

	step size
Planar	1.0
Simple corridor	0.4
Corridor	0.7
Wrench	3.0
Hole	1.0
Manipulator	0.1

Table 4.2 The step sizes for the robots.

	Type of DOF of the robot		
	translational	**rotational$_1$**	**rotational$_3$**
Planar	1, 1		
Simple corridor	1, 1, 1		3
Corridor	1, 1, 1		7
Wrench	6, 6, 6		30
Hole	1, 1, 1		11
Manipulator		6, 6, 6, 2, 2, 2	

Table 4.3 The weights for each DOF of the robots.

4.5.2 Experimental results

The results are listed in Table 4.4 and visualized in Figure 4.6.

Planar A retraction in the workspace results in a path having the optimal amount of clearance. The statistics show that the $\mathcal{C}$-retraction technique reached these optimal values. However, for robots having two translational DOFs, we recommend the $\mathcal{W}$-retraction technique as this technique is considerably faster.

Simple corridor As expected, a large increase in clearance was introduced by the retraction algorithms. At the expense of five extra seconds of computing time, the $\mathcal{C}$-retraction technique doubled the minimum amount of clearance and increased the average clearance with 39% with respect to the $\mathcal{W}$-retraction technique.

Corridor Although there is little room between the walls of the corridor and the robot, the techniques were still able to increase the clearance along the path. Again, the $\mathcal{C}$-retraction technique outperformed the $\mathcal{W}$-retraction technique but this took much more computation time.

Wrench Both algorithms needed relatively much time as the environment was larger compared to the other ones. Both algorithms were successful in increasing the clearance. The $\mathcal{C}$-retraction technique performed slightly better with respect to increasing the average and maximum clearance. However, the $\mathcal{W}$-retraction technique was 6% better with respect to the minimum clearance. This was due to the early termination of the $\mathcal{C}$-retraction, as showed by decreasing the termination threshold.

Hole The $\mathcal{W}$-retraction technique doubled the amount of minimum and average clearance along the path. The $\mathcal{C}$-retraction technique outperformed the $\mathcal{W}$-retraction technique because all DOFs were taken into account. The minimum amount of clearance along the path was further improved with 33%.

Manipulator The minimum clearance along the initial path was nearly zero. The $\mathcal{C}$-retraction technique successfully introduced some clearance along the path. Although there is little room for the manipulator to move, the algorithm doubled the average clearance along the path. This extra clearance may be crucial in high-cost environments to guarantee safety. For clarity, we only visualized a part of the sweep volume of the manipulator in Figure 4.6.

Planar	Clearance			Time
	min	**avg**	**max**	**s**
Initial path	0.00	2.47	7.15	-
$\mathcal{W}$**-retraction**	1.79	4.49	8.32	0.8
$\mathcal{C}$**-retraction**	1.79	4.49	8.32	9.4

Simple corridor	Clearance			Time
	min	**avg**	**max**	**s**
Initial path	0.16	1.91	3.83	-
$\mathcal{W}$**-retraction**	0.62	2.62	3.96	0.7
$\mathcal{C}$**-retraction**	1.21	3.64	4.25	6.0

Corridor	Clearance			Time
	min	**avg**	**max**	**s**
Initial path	0.01	0.59	2.44	-
$\mathcal{W}$**-retraction**	0.22	1.15	3.22	1.0
$\mathcal{C}$**-retraction**	0.27	1.87	4.57	27.6

Wrench	Clearance			Time
	min	**avg**	**max**	**s**
Initial path	0.00	4.17	11.32	-
$\mathcal{W}$**-retraction**	2.11	7.12	12.38	12.4
$\mathcal{C}$**-retraction**	1.99	7.83	15.03	373.8

Hole	Clearance			Time
	min	**avg**	**max**	**s**
Initial path	0.28	1.81	5.97	-
$\mathcal{W}$**-retraction**	0.79	3.08	6.85	0.6
$\mathcal{C}$**-retraction**	1.05	3.44	7.24	12.7

Manipulator	Clearance			Time
	min	**avg**	**max**	**s**
Initial path	0.00	0.14	0.35	-
$\mathcal{W}$**-retraction**	n.a.	n.a.	n.a.	n.a.
$\mathcal{C}$**-retraction**	0.05	0.29	0.43	26.8

Table 4.4 Clearance statistics for the six environments. A larger clearance indicates a better result. The $\mathcal{C}$-retraction statistics are the averages over 100 independent runs.

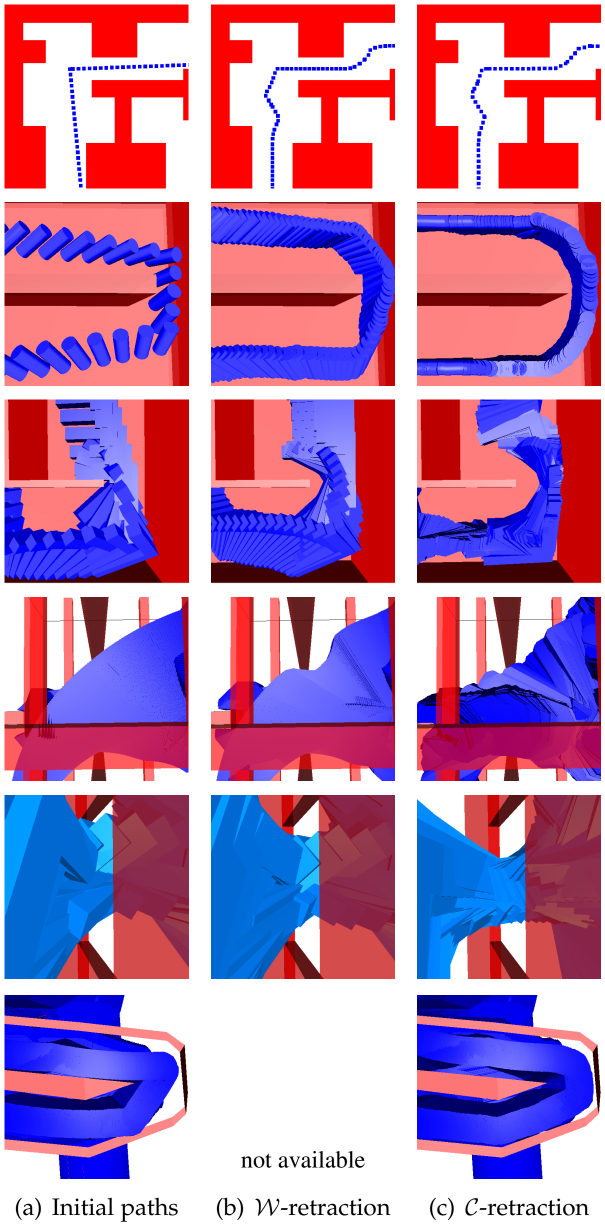


Figure 4.6 A close-up of the paths in the six environments. The pictures in the left column show parts of the initial paths. The paths in the middle column are the result of the $\mathcal{W}$-retraction technique while the paths in the right have been created by one particular run of the $\mathcal{C}$-retraction technique.

4.6 Discussion

We presented two new simple algorithms that increase the clearance along paths. They improve on existing algorithms since higher amounts of clearance for a larger diversity of robots are obtained. Moreover, they do not need complex data structures and (manual) preprocessing.

The first algorithm, $\mathcal{W}$-retraction, is fast but it can only deal with rigid, translating bodies. The second algorithm, $\mathcal{C}$-retraction, is slower but it outperforms the workspace-based algorithm as higher amounts of clearance along the paths are obtained. Furthermore, it can handle a broader range of robots which may reside in arbitrary high-dimensional configuration spaces.

The running times indicate that improving the clearance along paths may be too slow to be applied online. However, in applications where safety is important, the running times are not that crucial. For example, due to the difficulty of measuring and controlling the precise position of a manipulator arm, the arm can be damaged if it moves near obstacles. Improving the clearance at the cost of a few minutes of calculation time can prevent damage to the robot and its environment.

We expect that the running times of the $\mathcal{C}$-retraction algorithm can be dramatically decreased by incorporating learning techniques. This is a topic of future research. However, when on-line performance is needed, a complete roadmap should preferably be retracted to the medial axis in the preprocessing phase. We will show in Section 7.3 that a path can indeed be extracted from such a pre-processed roadmap in real-time.

CHAPTER
FIVE

DECREASING PATH LENGTH

Paths which are created by e.g. sampling-based motion planning methods usually contain many redundant motions. However, most applications require short paths because they take less time to execute. A popular algorithm for decreasing the path length is the *Shortcut* method. This method iteratively takes two random configurations on the path and creates a shortcut between them if this shortcut does not cause the robot to collide with the obstacles. Unfortunately, this heuristic will not remove all redundant motions. This is mainly caused by simultaneously interpolating all degrees of freedom of the configurations when creating a shortcut. We propose a new algorithm, *Partial shortcut*, which successfully deals with these difficulties by interpolating one degree of freedom at a time.

5.1 Introduction

Due to their probabilistic nature, sampling-based planners create low-quality paths. These paths often contain many unnecessary and jerky motions. However, many applications require a short path since redundant motions will take longer to execute. In this chapter, we will study several techniques for reducing the path length.

A simple technique that decreases the path length is called *Path pruning*. This technique assumes that a path is represented by a list of nodes $\nu_0, \cdots, \nu_{n-1} \in V$ which may originate from a graph. When a robot must move along this path, it should first be converted to a discrete path Π using a local planner (see Definition 4.3). The path pruning technique removes a node ν_{i+1} from a path Π if the local path $\text{LP}[\nu_i, \nu_{i+2}]$ between nodes ν_i and ν_{i+2} is collision-free. Details will be given in Section 5.2.

In Section 5.3, we investigate the *Shortcut* heuristic, which is the most often applied method because of its effectiveness and simple implementation. The Shortcut heuristic takes two random configurations on the path. If the part between these two configurations can be replaced by a new shorter path, produced by a local planner, then the original part is replaced by the new path. This technique outperforms the Path pruning heuristic as not only nodes are considered on the path, but also all intermediary configurations. The configurations can be chosen randomly [31,50,84,131,140,151], or deterministically [5,72,75]. Also several variants of this heuristic have been used [5,72,75,84].

We will show in Section 5.4 that the previous two heuristics will not remove all redundant motions. This is because interpolations are performed between all degrees of freedom (DOFs) simultaneously. We propose the *Partial shortcut* heuristic which takes only one DOF into account in each optimization step.

The experiments in Section 5.5 will show that this efficient method creates shorter paths than the other methods.

5.2 Path pruning

In this section, we assume that a path is represented by a list of nodes, see Definition 5.1. As these nodes are often generated randomly, the path will be jerky. A very simple technique that decreases the path length considerably is to remove all redundant nodes. A node ν_{i+1} on node path N is redundant if the configurations on the local path $\text{LP}[\nu_i, \nu_{i+2}]$ are collision-free. Besides being simple, the technique is efficient and deterministic. See Algorithm 5.1 for more details.

Definition 5.1 (Node path N). *A node path N is a series of nodes $v_0, \cdots, v_{n-1}$ such that the local paths* LP$[v_i, v_{i+1}]$ *are collision-free.*

Algorithm 5.1 REMOVEREDUNDANTNODES(node path N)

```
1: $i \leftarrow 0$
2: while $i < |N| - 2$ do
3:     if LP$[v_i, v_{i+2}] \in \mathcal{C}_{\text{free}}$ then
4:         $N \leftarrow N \backslash v_{i+1}$
5:         if $i > 0$ then $i \leftarrow i - 1$
6:     else
7:         $i \leftarrow i + 1$
8: return $N$
```

5.3 Shortcuts

While the previous method only considers the nodes of a path, the shortcut method considers all configurations on a discrete path Π (see Definition 4.3). Therefore, this method is expected to create shorter paths at the cost of increased computation time. The method tries to iteratively improve the path (see Algorithm 5.2). In each iteration, path Π is randomly split in three parts. Let π_a and π_b denote the begin and end configurations of the middle part. If the local path LP(π_a, π_b) is collision-free, then this local path replaces the middle part. As we use a straight-line local planner[1], all DOFs are interpolated simultaneously. See Section 1.2.4 for details on interpolation.

5.4 Partial shortcuts

Redundant motions (like unnecessary rotations) will not be removed by the previous two heuristics as they can only be removed by considering large portions of the path. But if we consider such a large portion, some other degrees of freedom (DOFs) are necessary to navigate around obstacles. Hence, applying the local planner to such a long portion is not going to succeed (see Figure 5.1).

The standard optimization technique (Shortcut) replaces pieces of the path by a straight-line segment in the configuration space. In this way, all DOFs

[1]When another local planner is used (e.g. a local planner for non-holonomic robots), care has to be taken that path Π' and Π'' keep satisfying the constraints of the local planner.

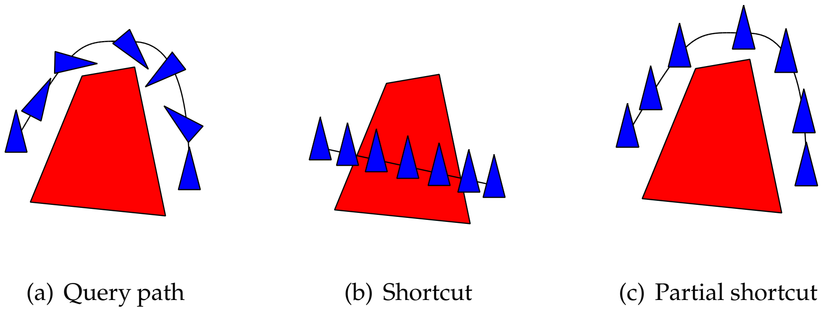


Figure 5.1 Translation is required to navigate around the obstacle and rotation can only be optimized by considering large portions of the path.

Algorithm 5.2 SHORTCUT(discrete path Π)

```
1: loop
2:     number of configurations $n \leftarrow |\Pi|$
3:     $a, b \leftarrow$ two random indices $0 \leq a + 1 < b < n$
4:     $\Pi' \leftarrow \pi_0, \cdots, \pi_{a-1}$
5:     $\Pi'' \leftarrow \pi_a, \cdots, \pi_b$
6:     $\Pi''' \leftarrow \pi_{b+1}, \cdots, \pi_{n-1}$
7:     if $\text{LP}(\pi_a, \pi_b) \in \mathcal{C}_{\text{free}}$ then
8:         $\Pi \leftarrow \Pi' \cup \text{LP}(\pi_a, \pi_b) \cup \Pi'''$
```

are interpolated simultaneously. Some of them might be necessary to move around the obstacles while others are not. The translational DOFs in particular are often necessary to guide the object around an obstacle while the rotational DOFs might be less relevant. Consequently, applying the local planner on such a part of the path will fail. Applying the local planner to optimize shorter pieces of the path will not remove the redundant rotations either because the two positions on the path will often have rather different orientations. Therefore, the rotation is required locally, while more globally, it might be redundant (see Figure 5.1).

We created a new technique, called *Partial Shortcut*, which takes only one DOF f into account in each optimization step. Algorithm 5.3 outlines the technique. In line 2, the chance that a particular DOF is chosen is dependent on its weight, i.e. $P(\text{DOF } i) = w_i / \sum_{i=0}^{n-1} w_i$. In this expression, we consider rotation in 3D as one DOF. Then, we split path Π in the same way as we did in the Shortcut algorithm.[2] Now let $\pi''_i[f]$ indicate the value for the f^{th} DOF of configuration π''_i of path Π''. We replace in each configuration π''_i the value of the f^{th} DOF by the value interpolated between $\pi''_0[f]$ and $\pi''_{m-1}[f]$, where m is the number of configurations on path Π''. After creating partial shortcuts, it can occur that the distance between two adjacent configurations on path Π'' is larger then the step size. In such a case, we validate Π'' by inserting extra configurations such that $\forall i : d(\pi_i, \pi_{i+1}) \leq step$. If path Π'' is collision-free, then Π'' replaces the original middle part. In this path all DOFs behave in the same way as in the original path except for DOF f.

We expect that this method will be slower than the Shortcut heuristic as only one DOF is taken into account in each iteration. However, we expect that more redundant motions can be removed.

Algorithm 5.3 PARTIALSHORTCUT(discrete path Π)

```
1: loop
2:   $f \leftarrow$ a random degree of freedom
3:   number of configurations $n \leftarrow |\Pi|$
4:   $a, b \leftarrow$ two random indices: $0 \leq a + 1 < b < n$
5:   $\Pi' \leftarrow \pi_0, \cdots, \pi_{a-1}$
6:   $\Pi'' \leftarrow \pi_a, \cdots, \pi_b$
7:   $\Pi''' \leftarrow \pi_{b+1}, \cdots, \pi_{n-1}$
8:   $m \leftarrow |\Pi''|$
9:   for all $\pi''_i \in \Pi''$ do
10:      $\pi''_i[f] \leftarrow$ INTERPOLATE$(\pi''_0[f], \pi''_{m-1}[f], i/(m-1))$
11:  VALIDATEPATH$(\Pi'')$
12:  if $\Pi'' \in \mathcal{C}_{\text{free}}$ then
13:      $\Pi \leftarrow \Pi' \cup \Pi'' \cup \Pi'''$
```

5.5 Experiments

In this section, we will apply the three techniques from the previous sections to six different paths. These paths have been selected such that an optimization step cannot easily change the homotopic class of a path. Our goal is to investigate the extent to which these techniques can improve the paths.

[2]We assume that there are no constraints which the configurations on the path have to satisfy.

5.5.1 Experimental setup

To test the quality of the three techniques, we considered the environments depicted in Figure 5.2. These are the same test environments as used in Chapter 4. Their properties are stated in Table 4.1 and Table 4.2. Since they have already been introduced in Section 4.5, we only discuss the properties of the paths:

Planar We use this simple problem involving two DOFs to check whether the techniques can reach the optimal solution.

Simple corridor The environment contains a jerky path traversed by a small free-flying cylinder. Many motions are redundant. We expect that they can be removed easily by all techniques.

Corridor Traversing corridors requires rotation of the elbow shaped object. The path has little clearance. Redundant rotations can only be removed by considering large portions of the path. Hence, we expect that the Partial shortcut technique outperforms the other techniques.

Wrench This environment features a large moving object in a small workspace. The moving object is rather constrained at the start and goal. In contrast to the previous three environments, rotational DOFs are now more important than translational ones. Again, we expect that the Partial shortcut technique outperforms the other techniques as this technique handles each DOF independently.

Hole The path contains many redundant (rotational) motions. Only where the path passes through the hole, the clearance is small, which may cause difficulties in removing the redundant rotational part of the motions for the Path pruning and Shortcut methods.

Manipulator In this environment, there is a major difference in importance of the six rotational DOFs. That is, the link that is closest to the base is more important than the gripper of the manipulator. As only the Partial shortcut technique recognizes this difference, we again expect this technique to outperform the other ones.

We need a distance measure to discuss path length. As we want to distinguish between rotational and translational DOFs, we compute the length of a discrete path Π as follows:

$$d(\Pi) = d_r(\Pi) + d_t(\Pi),$$

where

$$d_r(\Pi) = \sum_{i=0}^{n-2} d_r(\pi_i, \pi_{i+1}) \qquad \text{and} \qquad d_t(\Pi) = \sum_{i=0}^{n-2} d_t(\pi_i, \pi_{i+1}).$$

Let $q = \pi_i$ and $r = \pi_{i+1}$. Then, for all k rotational DOFs $0 \leq j < k$ and for all $(l - k)$ translational DOFs $k \leq j < l$:

$$d_r(q,r) = \sqrt{\sum_{j=0}^{k-1}[w_j d(q_j, r_j)]^2} \qquad \text{and} \qquad d_t(q,r) = \sqrt{\sum_{j=k}^{l-1}[w_j d(q_j, r_j)]^2}.$$

The partial distances $d(q_j, r_j)$ are calculated in the same way as in Section 1.2.3. The weights w_j that are used for the different environments are listed in Table 4.3.

We express path length as a percentage relatively to the 'optimal' path length to facilitate the comparison between different optimization techniques. Let d be the path length and d_{opt} be the optimal path length. Then we calculate the percentage Δd as

$$\Delta d = 100\% * \frac{d - d_{opt}}{d_{opt}}.$$

The closer this number approaches zero, the closer to optimal the path is. The optimal path lengths were defined as the paths of minimum length over all experiments conducted and are stated in Table 5.1 and depicted in Figure 5.3. Even though we cannot guarantee that these are indeed optimal, visual inspection strongly suggests that they are very close to optimal. Table 5.2 shows the initial relative lengths. The paths are far from being optimal. For example, the path in the Simple corridor environment is 408% longer than the shortest path encountered in all experiments with this environment.

We will investigate the extent to which the three heuristics can improve the six paths from Figure 5.2. We will run the Path pruning heuristic once for each experiment as this technique is deterministic. We will use these paths as input for the Shortcut and Partial shortcuts heuristics. As these techniques are non-deterministic, we run them 100 times for each experiment and report the average results. To ensure the exploitation of the full potential of the heuristics, we run each non-deterministic experiment for 120 seconds as preliminary experiments showed that all paths converges within this time.

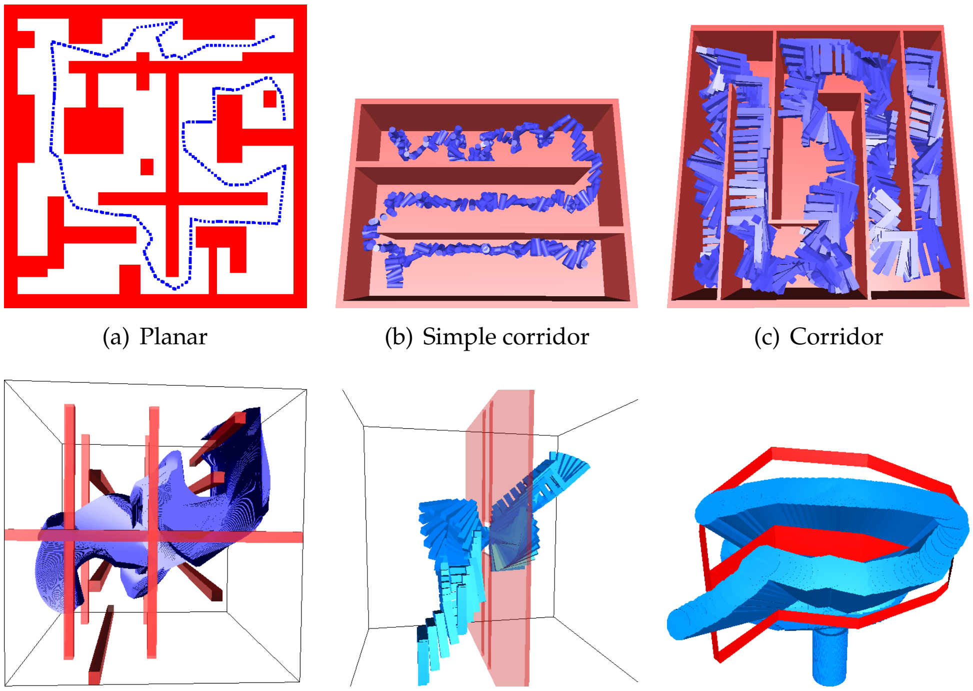
In the second batch of experiments we determine how long the Shortcut and Partial shortcut heuristics should be applied to obtain reasonably short paths. This is useful for practical purposes.

	Shortest path length		
	$d_{r_{opt}}$	$d_{t_{opt}}$	d_{opt}
Planar	-	300.12	300.12
Simple corridor	0.12	100.84	100.96
Corridor	19.31	162.59	181.90
Wrench	827.63	138.99	966.62
Hole	6.87	36.76	43.63
Manipulator	10.73	-	10.73

Table 5.1 The shortest absolute lengths of the paths.

	Relative path length		
	Δd_r	Δd_t	Δd
Planar	-	40	40
Simple corridor	213,917	154	408
Corridor	1,296	132	256
Wrench	113	112	113
Hole	628	61	150
Manipulator	55	-	55

Table 5.2 The relative length statistics of the initial paths. The numbers are expressed as percentages relatively to the optimal path lengths.


(a) Planar (b) Simple corridor (c) Corridor

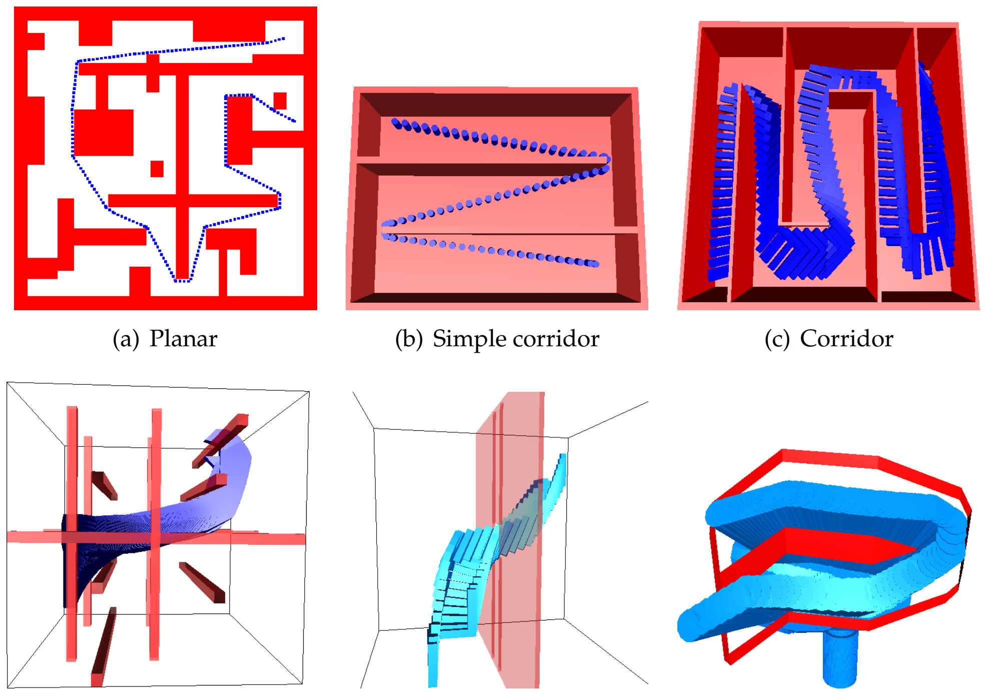
(d) Wrench (e) Hole (f) Manipulator

Figure 5.2 The six test environments and their corresponding *initial* paths.

5.5.2 Experimental results

The results of the experiments are stated in Table 5.3. This table shows the relative path length (rotational, translational and total length) for the initial paths and the three heuristics.

The table shows that the Path pruning heuristic improved the paths considerably in all cases. Note that the rotational distance (Δd_r) is still far from optimal, although the translational distance (Δd_t) has been decreased considerably. The running times of this deterministic heuristic were between 5 and 54 ms. The Shortcut heuristic was able to decrease the path length even more, i.e. the paths obtained a length that was 3 to 28 percent larger than the optimal paths. However, the paths still contained many redundant rotational motions. The Partial shortcut heuristic was much better able to remove the redundant (rotational) motions than the previous heuristics. In addition, the translational lengths of the paths became close to optimal.



(a) Planar (b) Simple corridor (c) Corridor

(d) Wrench (e) Hole (f) Manipulator

Figure 5.3 The six test environments and their corresponding *optimal* paths.

We will now examine the results more closely for each environment.

Planar Both the Shortcut and Partial shortcut techniques reached the optimal solution. The latter one produced paths that were on average only 1% larger than the optimal path.

Simple corridor The initial path contained many redundant (rotational) motions which could not be removed completely by the Path pruning and Shortcut heuristics. However, the Partial shortcut technique was able to produce paths that are very close to the optimal path as only one DOF is optimized during an iteration of the algorithm. Note that the relative rotational path length seems to be very large while the total relative path length was only 1. This is because the optimal (absolute) rotational distance was very close to zero (0.12).

Corridor Also in this environment, the Partial shortcut heuristic outperformed the other techniques. A large part of the redundant rotational motions was removed as large portions of the path could be replaced by less redundant motions.

Wrench The optimal path corresponds to a smooth motion traversed by the wrench. Again, the Partial shortcut heuristic was able to produce such a path as the resulting paths were only 3% worse than the optimal path.

Hole All techniques removed the redundant translational motions, i.e. the translational relative path lengths for the Shortcut and Partial shortcut heuristics were only 0.49% and 0.33%. However, it was difficult to remove the rotational motions as the moving object was rather constrained near the hole. However, the Partial shortcut heuristic obtained a path that was on average 5% longer than the optimal path.

Manipulator The Shortcut and Partial shortcut methods created short paths which are comparable to the optimal path depicted in Figure 5.3(f). However, the latter one outperformed the Shortcut method and produced paths that were on average only 3% larger than the optimal path.

In our experiments, we ran the heuristics for 120 seconds as we wanted to see their full potential. In all cases, the Partial shortcut heuristic outperformed the Shortcut heuristic. Hence, when optimal path quality is desired (in terms of path length), the Partial shortcut algorithm should be used.

However, the running times may be too high for on-line use. An important question is how well the heuristics perform when there is less computation time available. Figure 5.4 shows for each environment the relationship between the running time and the relative path length of both the Shortcut and Partial shortcut heuristics. Each marker in the graphs represents the averaged relative path length over 100 independent runs. In all but one environment (Manipulator), the Partial shortcut heuristic always outperforms the Shortcut heuristic. Therefore, the Partial shortcut heuristic should be preferred. Furthermore, it can be observed that the relative path length decreases rapidly as the available computation time increases. When the path is relatively simple (such as in the Planar, Simple corridor, Wrench and Hole environment), only one second of computation time is required to obtain a path that is about 5% longer than the optimal path. For more complex paths (such as in the Corridor and Manipulator environment), the paths obtained after one second are about 25% longer than the optimal path.

We conclude that reasonably short paths can be obtained for all tested environments when the (Partial shortcut) algorithm is run for one second.

Planar	Relative path length		
	Δd_r	Δd_t	Δd
Initial	-	40	40
Path pruning	-	15	15
Shortcut	-	3	3
Partial shortcut	-	1	1

Simple corridor	Relative path length		
	Δd_r	Δd_t	Δd
Initial	213,917	154	408
Path pruning	15,817	10	28
Shortcut	11,633	3	17
Partial shortcut	383	1	1

Corridor	Relative path length		
	Δd_r	Δd_t	Δd
Initial	1,296	132	256
Path pruning	326	34	65
Shortcut	133	8	21
Partial shortcut	35	4	7

Wrench	Relative path length		
	Δd_r	Δd_t	Δd
Initial	113	112	113
Path pruning	71	71	71
Shortcut	28	28	28
Partial shortcut	3	3	3

Hole	Relative path length		
	Δd_r	Δd_t	Δd
Initial	628	61	150
Path pruning	462	17	87
Shortcut	155	0	25
Partial shortcut	27	0	5

Manipulator	Relative path length		
	Δd_r	Δd_t	Δd
Initial	55	-	55
Path pruning	45	-	45
Shortcut	8	-	8
Partial shortcut	3	-	3

Table 5.3 The relative length statistics of the resulting paths. The numbers are expressed as percentages relatively to the optimal path lengths. The closer a number approaches zero, the closer to optimal it is.

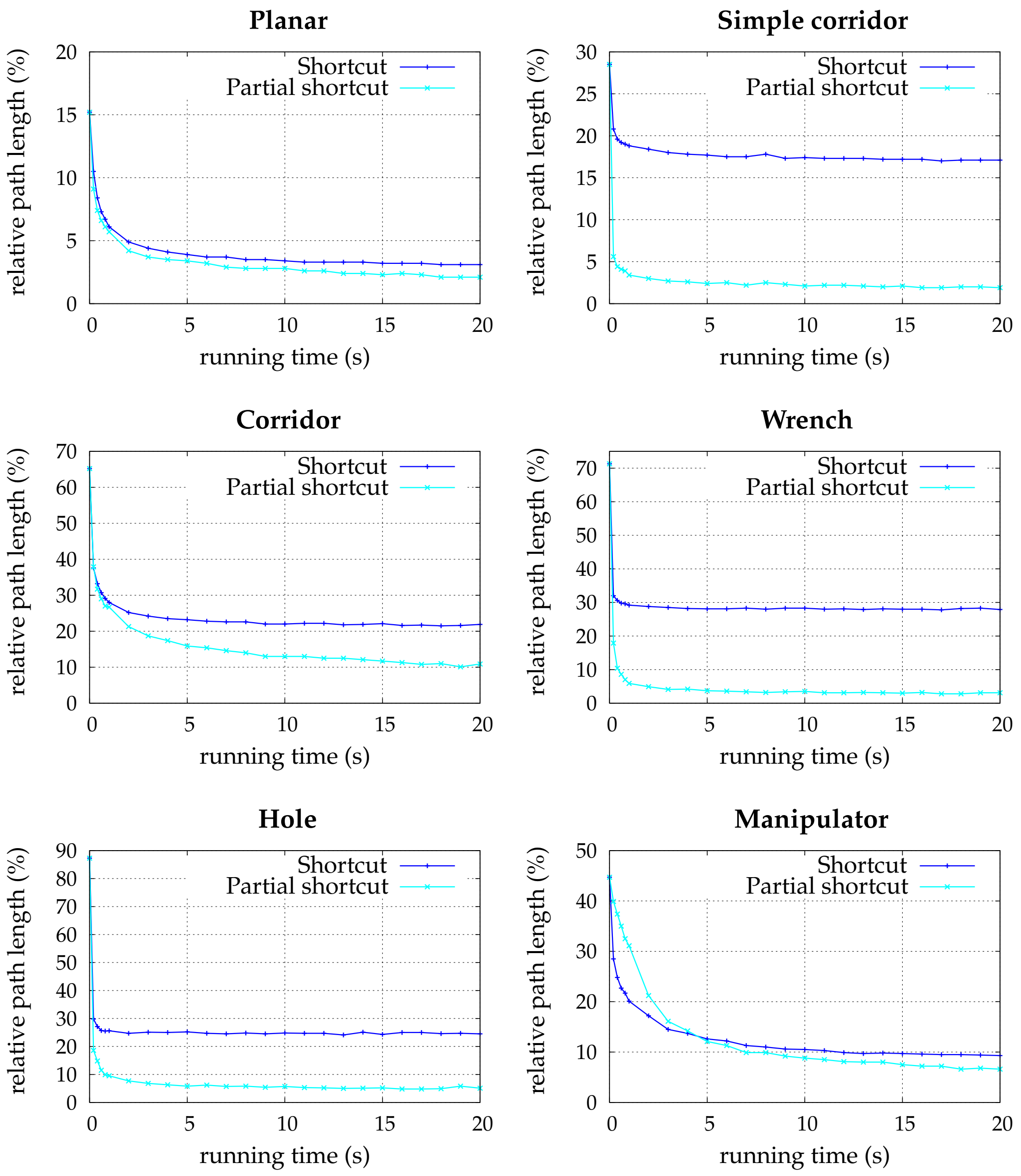


Figure 5.4 Convergence of the Shortcut and Partial shortcut heuristics in the six environments.

5.6 Discussion

We compared three simple heuristics to decrease the path length. We showed that the Path pruning heuristic is a fast and effective technique that can be used to decrease the path length. The length can be further decreased by the Shortcut heuristic which is often used as this technique is easy to implement. However, this technique can have difficulties removing all redundant (rotational) motions as all DOFs are interpolated simultaneously. We presented a new technique, Partial shortcut, which takes only one DOF into account in each optimization step. Experiments showed that these redundant motions are now successfully removed. Another advantage of this technique is that the Partial shortcut technique creates shorter paths than the Shortcut technique. Reasonably short paths are obtained within one second of computation time on a modern personal computer.

In this chapter, we focused on paths traversed by holonomic robots. An interesting topic for future research is to extend the Shortcut and Partial Shortcut method such that non-holonomic constraints are satisfied.

CHAPTER
SIX

Creating small roadmaps

Many algorithms have been proposed that create a roadmap from which a path for a moving object can be extracted. These algorithms generally do not give guarantees on the quality of the roadmap.

In this chapter, we propose the *Reachability Roadmap Method*, which is a new efficient algorithm that creates small roadmaps for two- and three-dimensional problems. A small roadmap assures low query times and low memory consumption. A path that is extracted from such a roadmap will have reasonably long edges which is useful when optimizing paths. The method is resolution complete in the sense that a valid query (which consists of a start and goal configuration) can always be connected to the roadmap. In addition, when the start and goal belong to the same connected component of the free space, a corresponding path can always be found (at a given resolution).

It may be surprising to see how small covering roadmaps can be. We compare the Reachability Roadmap Method (RRM) with the PRM and visibility PRM. Experiments will show that roadmaps created by the PRM can have many nodes and edges. The visibility PRM creates reasonably small roadmaps but this method has great difficulties in creating covering roadmaps. In contrast, the RRM creates small covering roadmaps for all tested environments.

6.1 Introduction

In application areas such as mobile robots, manipulation planning, CAD systems, virtual environments, protein folding and human robot planning, it is often desirable that a path between a start and goal configuration (which is a placement of the robot in the workspace) is quickly found. For example, maintenance studies in industrial installations [143] can be performed more efficiently if an engineer does not have to wait long for a solution. In other fields, e.g. in computer games, only a fixed small amount of calculation time is available to compute the path. If this takes too long, the game may halt.

Single shot methods for motion planning aim at quickly connecting a start configuration to a goal configuration. Although good performance is achieved (compared to the traditional methods described in [99]), they may still be too slow as the calculations are performed on-line. In contrast, the Probabilistic Roadmap Method (PRM) enables the construction of a path in real-time [6, 85, 117].

A drawback of the PRM is that a resulting roadmap often contains many redundant nodes and edges, in particular when the environment contains one or more narrow passages. Over the years, many improvements have been suggested to tackle the narrow passage problem [22, 69, 107] but those solutions often lead to large roadmaps. Such large roadmaps increase the time needed to extract a path and can require a vast amount of memory which may not always be available. Furthermore, the roadmap may contain many short edges which complicates the smoothing phase that often follows a query phase [117]. Another drawback is that a path may not always be found in practice, in spite of its existence. For example, this can occur when the roadmap is not dense enough.

A variant of the PRM, which aims at keeping the roadmap small, is the visibility-based roadmap proposed by Nissoux *et al.* [121]. Recall that this technique only connects configurations to useful nodes. Usefulness is determined as follows: when a new node cannot be connected to other nodes it forms a new connected component and is labeled useful. If it connects two or more components it is also labeled useful. If it can be connected to just one component it is not labeled useful. Although this approach prunes the roadmap a lot, it is often slower than other variants of the PRM as the approach is too strict in rejecting nodes (see Section 2.5.3).

We propose a new efficient method that creates small roadmaps for two- and three-dimensional problems. Our method will ensure that a valid query can always be connected to the roadmap and that a path is always found (if one exists) at a given resolution. Because the roadmap is small, smoothing can easily be applied in the preprocessing phase, leading to instant query answers. We will elaborate on the properties of the approach in Section 6.2. In Section 6.3, we show the outline of the method. Details will be given in Section 6.4. In Section 6.5, we show some experiments on different environments, and in Section 6.6, we conclude that our algorithm successfully creates small roadmaps for these environments.

6.2 Reachability

An important issue is how to determine when the roadmap is dense enough, i.e. when should the algorithm terminate? Many motion planning techniques such as the PRM and RRT are probabilistically complete, i.e. whenever a path exists, the probability that it will be found converges to one as the computation time goes to infinity. As there is no guaranteed upper bound, the running time may not be a practical termination criterion. Many authors terminate the method when a path between a specified start and goal is found. However, this does not guarantee that this roadmap can solve every query.

In Section 3.2, we used the *reachability* criterion to determine when a problem has been solved, i.e. a problem has been solved if a roadmap $G = (V, E)$ covers the free configuration space ($\mathcal{C}_{\text{free}}$) and captures its connectivity. We repeat the definitions of *coverage* and *maximal connectivity*:

Definition 6.1 (coverage). *G covers $\mathcal{C}_{\text{free}}$ when each configuration $c \in \mathcal{C}_{\text{free}}$ can be connected using the local planner to at least one node $v \in V$.*

Definition 6.2 (maximal connectivity). *G is maximally connected when for all nodes $v', v'' \in V$, if there exists a path in $\mathcal{C}_{\text{free}}$ between v' and v'', then there exists a path in G between v' and v''.*

Coverage ensures that every query (which consists of a start and goal configuration) can be directly connected to the roadmap, as is required to solve the problem. If there exists a path (in $\mathcal{C}_{\text{free}}$) between the start and goal configuration, then maximal connectivity ensures that a path between them can be found in the roadmap.

Figure 6.1 shows an environment whose free space is covered by two (white) nodes and is connected via one extra (black) node. The reachability region for the upper left node has been drawn. Each configuration in this region can be connected with a straight-line local planner to the node, so a three-node graph suffices to solve this problem. In Section 6.4, we will explain how such a reachability region can be computed.

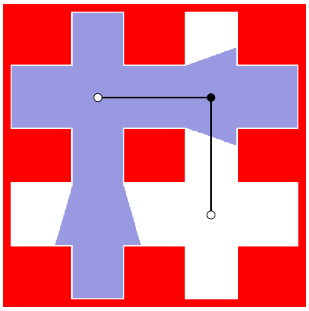


Figure 6.1 The coverage and maximal connectivity criteria have been met. The reachability regions of the white nodes cover the complete free space and are connected via the black node.

6.3 Reachability Roadmap Method

Our goal is to create a small roadmap $G = (V, E)$ that satisfies both the coverage and maximal connectivity criteria. The idea is to compute a small set of *guards* that cover the complete free space. These guards are then connected via *connectors* to fulfill the maximal connectivity criterion. The resulting roadmap is then pruned to obtain an even smaller roadmap (see Algorithm 6.1).

Computing the minimum number of guards corresponds to the well known art gallery problem which is NP-hard. See the book of O'Rourke which elaborates on this problem [123]. As we want to create a roadmap for a possibly large environment with many obstacles, exact algorithms will take too much time. Therefore, we will use a heuristic to create and place these guards (which are added as nodes to the roadmap).

Globally speaking, the guards are chosen as follows. To get the free space $\mathcal{C}_{\text{free}}$ covered as fast as possible, we place the guard nodes $V^g \subseteq V$ on locations where they probably cover a large part of the free space. Locations on the medial axis M are good candidates as they have a large clearance from the obstacles and thus have an increased probability of having a large reachability region.[1] In addition, placing guards on the medial axis produces 'good' roadmaps [107]. To prune the number of candidate guards, a new guard v^g is only accepted if its location c is not covered by guards that have already been placed. That is, $c \in M \backslash \mathcal{C}'$ where $\mathcal{C}' \subseteq \mathcal{C}_{\text{free}} = \bigcup_{v \in V^g} \text{REACHABILITYREGION}(v)$. Nonetheless, it can occur that all locations on the medial axis have been covered by the guards, i.e. $M \backslash \mathcal{C}' = \emptyset$, while the free space has not been fully

[1]See Section 4.1 for a definition of the medial axis.

Algorithm 6.1 ReachabilityRoadmap($\mathcal{C}$-space $\mathcal{C}$)

```
Output: Graph $G(V, E)$, where $V = V^g \cup V^c$
1: covered space $\mathcal{C}' \leftarrow \emptyset$
2: $M \leftarrow$ locus of configurations of the medial axis
3: while $\mathcal{C}' \neq \mathcal{C}_{\text{free}}$ do
4:   if $M \backslash \mathcal{C}' \neq \emptyset$ then
5:     guard node $\nu^g \leftarrow$ the node corresponding to a configuration $c \in M \backslash \mathcal{C}'$
6:   else
7:     configuration $c \leftarrow$ a configuration in $\mathcal{C}_{\text{free}} \backslash \mathcal{C}'$
8:     guard node $\nu^g \leftarrow$ node corresponding to RetractConfiguration$(c)$
9:   $\mathcal{C}' \leftarrow \mathcal{C}' \cup$ ReachabilityRegion$(\nu^g)$
10:  $V^g \leftarrow V^g \cup \nu^g$
11: for all $\nu_i, \nu_j \in V^g : i < j$ do
12:   $\mathcal{C}' \leftarrow$ ReachabilityRegion$(\nu_i) \cup$ ReachabilityRegion$(\nu_j)$
13:   if $\mathcal{C}' \neq \emptyset$ then
14:     $\nu^c \leftarrow$ connector node that corresponds to a configuration in $\mathcal{C}'$
15:     $V^c \leftarrow V^c \cup \nu^c$
16:     $E \leftarrow E \cup \epsilon(\nu_i, \nu^c)$
17:     $E \leftarrow E \cup \epsilon(\nu^c, \nu_j)$
18: PruneRoadmap$(G, \mathcal{C})$
```

covered yet, i.e. $\mathcal{C}' \neq \mathcal{C}_{\text{free}}$. In such a situation we choose a configuration c that has not been covered yet by other guards, i.e. $c \in \mathcal{C}_{\text{free}} \backslash \mathcal{C}'$. Then we retract this configuration to the medial axis to increase the expected size of its reachability region. (Note that the location of this guard will be covered by other guards.) We keep adding new guards to the roadmap until $\mathcal{C}_{\text{free}}$ has been fully covered, i.e. $\mathcal{C}' = \mathcal{C}_{\text{free}}$.

We connect the guards by placing connectors in the overlapping reachability regions of the guards. We consider all pairs of guards whose regions overlap. For each pair $(\nu_i, \nu_j) : i < j$, we add a connector node ν^c and its connections (edges $\epsilon(\nu_i, \nu^c)$ and $\epsilon(\nu^c, \nu_j)$) to the roadmap.

Finally, we prune this roadmap by transforming it to a Steiner Tree. Such a tree is a network that does not throw away guards (but is allowed to remove connectors) and keeps the connected components connected. To prune the roadmap even more, we add all remaining connections and create a minimal spanning tree.

Theorem 6.1 (Coverage of $\mathcal{C}_{\text{free}}$). *Algorithm 6.1 creates a roadmap that satisfies the coverage criterion.*

Proof: It is well known that each configuration in $\mathcal{C}_{\text{free}}$ can be connected to the medial axis in a straight line. Hence, we can limit ourselves to consider only configurations lying on the medial axis. We start with selecting configurations on the medial axis and add them as nodes to the roadmap. If all configurations on the medial axis are covered by the guards but there are still configurations in the free space that have not been covered yet, we select such a configuration which we retract to the medial axis. As this retraction can be performed in a straight line[2], the original configuration will be covered by the retracted configuration. Because we keep adding new guards until the free space is completely covered, the coverage criterion will be met. This criterion will remain satisfied when the roadmap is pruned as these heuristics never remove guards. ■

Theorem 6.2 (Maximal connectivity of $\mathcal{C}_{\text{free}}$). *Algorithm 6.1 creates a roadmap that satisfies the maximal connectivity criterion.*

Proof: Choset and Burdick show in [35] that the medial axis is a connected structure if the free space in which a robot operates is also connected. Hence, the medial axis is a complete representation for motion planning purposes. Let $v', v'' \in V$ be two random nodes in the graph (corresponding to configurations c' and c'') for which there exists a continuous path Π in $\mathcal{C}_{\text{free}}$ between c' and c''. As the algorithm satisfies the coverage criterion, there exists for each configuration $c \in \Pi$ a guard node $v^g \in V^g \subseteq V$ that covers c. Now consider the sequence of different guard nodes $v_i \in V^g$ that covers the configurations on Π. (If several guard nodes in V^g cover a particular configuration c, we consider the guard node having the smallest index.) Let now configurations c_a and c_b correspond to guard nodes v_i and v_{i+1}. Since there exists a path in $\mathcal{C}_{\text{free}}$ between configurations c_a and c_b, the reachability regions of nodes v_i and v_{i+1} must have some overlap. (This may be a point in the extreme case.) As the algorithm chooses a connector node $v^c \in V^c \subseteq V$ in this overlapping part, edges $\epsilon(v_i, v^c), \epsilon(v^c, v_{i+1}) \in E$ will be part of the graph. Hence, we can construct a path Π' in graph G between nodes v' and v'' as follows: $\Pi' \leftarrow \bigcup_i \epsilon(v_i, v^c) \cup \epsilon(v^c, v_{i+1})$.

The maximal connectivity criterion will remain satisfied when the roadmap is pruned as the heuristics only remove edges that are part of cycles. ■

[2]See Section 4.3.1.

6.4 Algorithmic details

The most time-consuming operation in our technique is the computation of the reachability regions. In general, an exact computation of these regions would involve intricate and practically infeasible calculations in the configuration space $\mathcal{C}$. To make the calculations feasible, we approximate the $\mathcal{C}$-space by discretizing it. Because of memory limitations, this approach is restricted to low-dimensional $\mathcal{C}$-spaces. We will perform experiments with both two- and three dimensional $\mathcal{C}$-spaces.

We discretize a d-dimensional $\mathcal{C}$-space as follows. First, we create a d-dimensional grid. For each cell in this grid we check whether the configuration that corresponds to the cell collides with the obstacles. We update the cell appropriately, i.e. we put the value 0 in the cell if it collides and 1 otherwise. We will use this grid in the remainder of the algorithm.

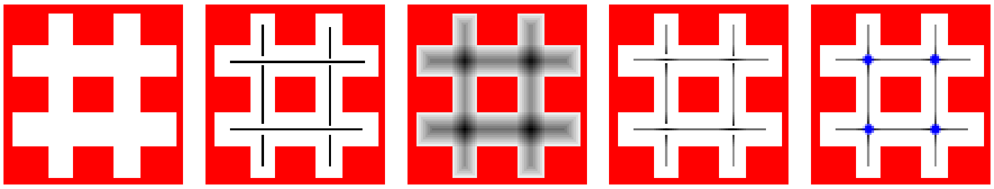
In theory, this grid could be used for motion planning purposes. However, in this chapter, we are dealing with environments that contain narrow passages and/or environments that are large. We need a huge grid for both types. In Chapter 7, we will show that such a grid will be too large to use as a roadmap, i.e. the time for extracting a query will be too high for real-time usage. Besides, our goal is to create small roadmaps.

In this section, we show for a given resolution how to get the free space covered and maximally connected. Then, we show how the roadmap can be pruned.

6.4.1 Coverage

We need a set of cells M that reside on the medial axis. Such a set can be computed efficiently by the medial axis transform (MAT). We use the algorithm from Lee *et al.* in [104]. This algorithm is a two-pass dynamic program that computes the MAT in O(n) time where n is the number of cells in the grid. See Figure 6.2(b) for an example.

We select guards on the medial axis having a large distance to the obstacles. The distance can be computed efficiently by the distance transform (DT) in O(n) time, see Figure 6.2(c). Like the MAT, the DT is also computed by a two-pass dynamic program, see e.g. [104]. By combining the MAT and DT, we know for each cell on the medial axis the closest distance to the obstacles, see Figure 6.2(d). These cells are then sorted by decreasing distance, using bucket sort in linear time. We store these cells (candidate guards) in a list M. Figure 6.2(e) shows an example of how such candidate guards are obtained.



(a) $\mathcal{C}$-space (b) MAT (c) DT (d) MAT $\cap$ DT (e) guards

Figure 6.2 The creation of candidate guards. The white area of Figure (a) corresponds to the discretized $\mathcal{C}$-space of Figure 6.1. Figure (b) shows the cells of the medial axis transform and (c) the distance transform. Dark cells correspond to a large distance to the closest obstacle while light cells correspond to a small distance. In (d), the intersection of the MAT and DT is shown. Finally, (e) shows the four guards on the medial axis having the largest distance in the distance transform.

We keep adding guards from this list to the roadmap until the free space has been fully covered (or when all candidate guards corresponding to the cells in M have been handled). As we have already mentioned, we initially only add guards as nodes to the graph which have not been covered yet by other guards. If the space is not covered when all cells in M have been handled, we consider uncovered cells (in order of decreasing distance) and add their corresponding configurations retracted on the medial axis (see Algorithm 4.3.1 for details).

There are two issues we have to resolve: how to determine the cells in the reachability region of a guard and how to determine which cells have not been covered yet.

We consider two ways to compute the reachability region of a guard placed in cell $g \in G \subseteq M$. The first algorithm is analogous to a *Flood-fill* algorithm. Its usual purpose is to fill an arbitrary bounded space in a picture with a given color. The boundary of a reachability region consists of cells that cannot be reached anymore from cell g. We start with adding cell g to a queue. The algorithm terminates looping when the queue is empty. In each loop, we extract a cell c from the queue. If the straight-line connection from g to c is in $\mathcal{C}_{\text{forb}}$ (which can be calculated efficiently by the Bresenham line-drawing algorithm [24]), then we know that g does not belong to the region. Hence, we can continue taking cells from the queue. In the other case, the region will be extended by cell c. Then we check for each of the facing neighbors of c whether such a neighbor is in $\mathcal{C}_{\text{free}}$ and whether the neighbor has not been covered yet. (A cell in a d-dimensional grid has $2d$ facing neighbors.) If the two conditions are satisfied, then we add the neighbor cell to the queue. This approach can be applied in a grid of any dimension d. Its complexity is $O(d|g|l)$, where $|g|$ is the number of cells in the region and l is the number of cells that intersects with the line from r to the furthest cell in the reachability region of g.

The second algorithm, which we call WEDGEFILL, improves this bound to $O(|g|)$ when the grid is two-dimensional. The algorithm subdivides the 2D space in four quadrants originating from cell g. In each quadrant, it fills cells within the boundaries of one or more wedges. For each wedge, the cells on the current scan line are added to the region if they are visible from g, i.e. the cell lies within the boundaries of the wedge. If an obstacle cell is encountered, the wedge is split or its boundaries are narrowed. This process is repeated until each wedge is too narrow to contain a cell or until all cells on the current scan line are obstacle cells.

To determine which cells have not been covered yet, we store for each cell in the grid the set of guards that cover the cell, i.e. when guard g_i is added to the roadmap, index i is inserted to the sets in the grid corresponding to all cells that are covered by this guard. A set is a data structure that allows insertions in O(log k) time [38], where k is the number of guards that cover this cell. Checking whether the cell has been covered by a guard also takes O(log k) time. Storing all covering guards might seem an overkill but we need this information in the next phase. In addition, k will generally be very small.

6.4.2 Maximal connectivity

As most guards usually do not cover any other guards, we have to calculate a set of connectors to which the guards can be connected such that the maximal connectivity criterion is satisfied. These connectors will be placed in the overlapping reachability regions of the guards. For each pair of guards that share cells in the grid, we place one connector in a cell that lies on the medial axis. As the number of shared cells is generally larger than one, we have to choose one of those cells. Since we prefer a large clearance, we choose the one that has the largest value in the distance transform (DT). When the set of shared cells contains no medial axis cells, we choose a cell that has the largest distance in the DT. If more than one cell satisfies this condition, we choose the one that minimizes the connection distance of the connector to the two guards. Possibly, different connectors share the same cell in the grid. These connectors are then merged. This part of the algorithm can be computed in linear time in the number of elements of the sets in the grid. We obtain a roadmap that satisfies the maximal connectivity criterion by adding all collision-free connections between the guards and connectors.

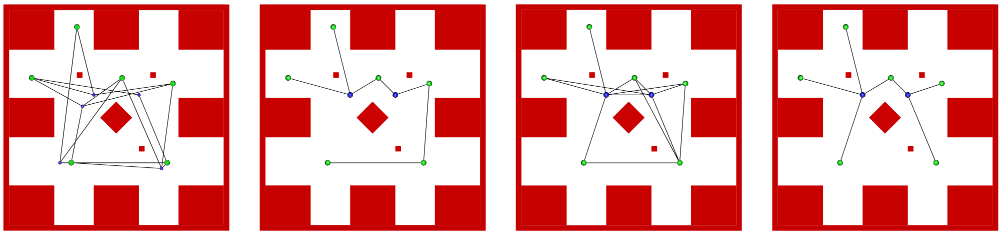
6.4.3 Roadmap pruning

If we allow all collision-free connections between guards and connectors, the number of connections can become quadratic in the number of nodes in the roadmap. Figure 6.3(a) shows such a fully connected roadmap. As our objective is to create small roadmaps, we want to solve the following problem. Given a roadmap $G = (V, E)$ that consists of a set of guards $V^g \subseteq V$, a set of connectors $V^c \subseteq V$, and all feasible connections E between the joint set of guards and connectors, create a shortest roadmap $G' = (V', E')$ (in terms of total edge length) that spans the guards. This problem is known as the discrete Euclidean Steiner problem which is NP-complete. (See the book of Hwang *et al.* [74] which elaborates on Steiner Tree problems.) Consequently, no polynomial time algorithm for this problem is likely to exist. We handle this problem by using the *shortest path heuristic*, described in the same book. Figure 6.3(b) shows such a tree. Algorithm 6.2 outlines the approach. First, a priority queue Q is initialized (sorted on increasing path length) with all shortest paths between the guards in V^g. Then, a graph G' is created that consists of all guards V^g. For each path Π in Q, an edge $\epsilon(v', v'')$ of Π is added to the edges E' if there exists no path between nodes v' and v'' in G'.

Algorithm 6.2 STEINERTREE(Graph $G(V = V^g \cup V^c, E)$)

```
Output: Graph $G'(V' = V^{g'} \cup V^{c'}, E')$
1:  PriorityQueue $Q$ {sorted on increasing path length}
2:  for all distinct pairs of guards $(v', v'')$ do
3:    node path $N \leftarrow$ shortest path between $v'$ and $v''$
4:    if $|N| > 0$ then $Q$.push($N$)
5:  $V^{g'} \leftarrow V^g$
6:  while not $Q$.empty() do
7:    $\Pi \leftarrow Q$.front()
8:    $Q$.pop()
9:    for $i \leftarrow 0$ to $|N| - 2$ do
10:     if $v_{i+1} \in V^c$ and $v_{i+1} \notin V^{c'}$ then
11:       $V^{c'} \leftarrow V^{c'} \cup v_{i+1}$
12:     if not $G'$.sameConnectedComponent($v_i, v_{i+1}$) then
13:       $E' \leftarrow E' \cup \epsilon(v_i, v_{i+1})$
14: remove all connectors $v^{c'} \in V^{c'}$ with degree 1
```

The total edge length of G' can be decreased further by the following approach. First, all collision-free connections between the nodes in G' are added



(a) Initial roadmap (b) Steiner Tree (c) All connections (d) MST

Figure 6.3 Roadmap pruning. Figure (a) shows the initial roadmap. From this roadmap, an approximated Steiner Tree is calculated in (b). In (c), all collision-free connections are added. Finally, (d) shows the minimal spanning tree (MST) of the roadmap in (c).

to G'. Then, its minimal spanning tree (MST) is calculated by Kruskal's algorithm [38]. It can occur that by reconnecting the edges, there are connectors in the graph having degree 1. These are removed as they do not contribute to the coverage and/or maximal connectivity. Figure 6.3 shows these pruning steps of the algorithm.

6.5 Experiments

In this section, we apply the Reachability Roadmap Method (RRM) to create roadmaps in five environments. These are depicted in Figure 6.4. We compare the RRM with the PRM and Visibility PRM with regard to the size of the roadmaps as well as the required computation time.

6.5.1 Experimental setup

For the experiments we use our SAMPLE system. We perform experiments on five different environments. They have the following properties:

Grid This is a simple 2D environment that contains 49 squares. The optimal solution is a roadmap containing seven guards, six connectors and twelve edges. We want to know whether the RRM can be competitive compared to the optimal solution.

Rotated grid We rotated the 49 squares which resulted in many narrow passages. We use this environment to study the effect of the reduced local visibility on the size of the roadmaps.

Corridor This environment consists of a 3D winding corridor with four hairpins. While we expect that the RRM creates a small roadmap, we expect that the PRM will be faster than the RRM because this is an easy problem with no narrow passages.

Manipulator This 3D environment features a robot arm with three rotational degrees of freedom that can move through a long passage. We selected this environment to show that our algorithm can handle complex reachability regions in $\mathcal{C}$-space. Note that this environment is much more complex than the environment from Figure 3.3(f).

Toy village This large 3D environment contains seven buildings (>10,000 objects). There are many scale differences in this environment, i.e. the environment has ample free spaces (outside) and small rooms (inside). We want to know whether our algorithm can handle such large environments.

For each environment, we create a roadmap with the Reachability Roadmap Method. See Table 6.1 for their level of discretization. The reachability regions are computed by the WEDGEFILL algorithm for 2D-problems and by FLOODFILL algorithm for the 3D-problems.

We compare this technique with the PRM (as this is a widely used motion planning method) and the Visibility PRM (as this method creates small roadmaps). We refer the reader to Chapter 2 for details on these techniques. They have some parameters that have to be set. We use the Halton sampling strategy with a random seed. The step size of the straight-line local planner corresponds to the size of a cell in the grid. We use the Forest neighbor selection strategy. For both techniques, we perform 100 independent runs. Such a run has been solved if both the coverage and maximal connectivity criteria have been satisfied. When we report the construction time, we do not include the time needed to check these two criteria.

In the first batch of experiments, we focus on the size of the roadmaps. We set the maximum connection distance and maximum number of connections to infinity. This will minimize the number of nodes and edges needed to fulfill the coverage and maximal connectivity criteria. These choices have a negative impact on the running times for the PRM and Visibility PRM (see Chapter 2). Nonetheless, these choices minimize the size of the roadmap.

In the second batch of experiments, we focus on the running times of the three methods. To provide a fair comparison, we use the optimal connection parameters for the PRM and Visibility PRM. These are stated in Table 6.2. Using

	number of cells
Grid	80×80
Rotated grid	200×200
Corridor	$40 \times 8 \times 40$
Manipulator	$60 \times 60 \times 60$
Toy village	$180 \times 22 \times 160$

Table 6.1 Level of discretization of the environments.

	Neighbor selection parameters	
	max. connection distance	**max. number of connections**
Grid$^{(*)}$	10	75
Rotated grid	5	75
Corridor	15	75
Manipulator	2	75
Toy village	50	75

Table 6.2 The optimal values used in the neighbor selection strategy. $^{(*)}$We did not constrain the maximum connection distance for the Visibility PRM in the Grid environment.

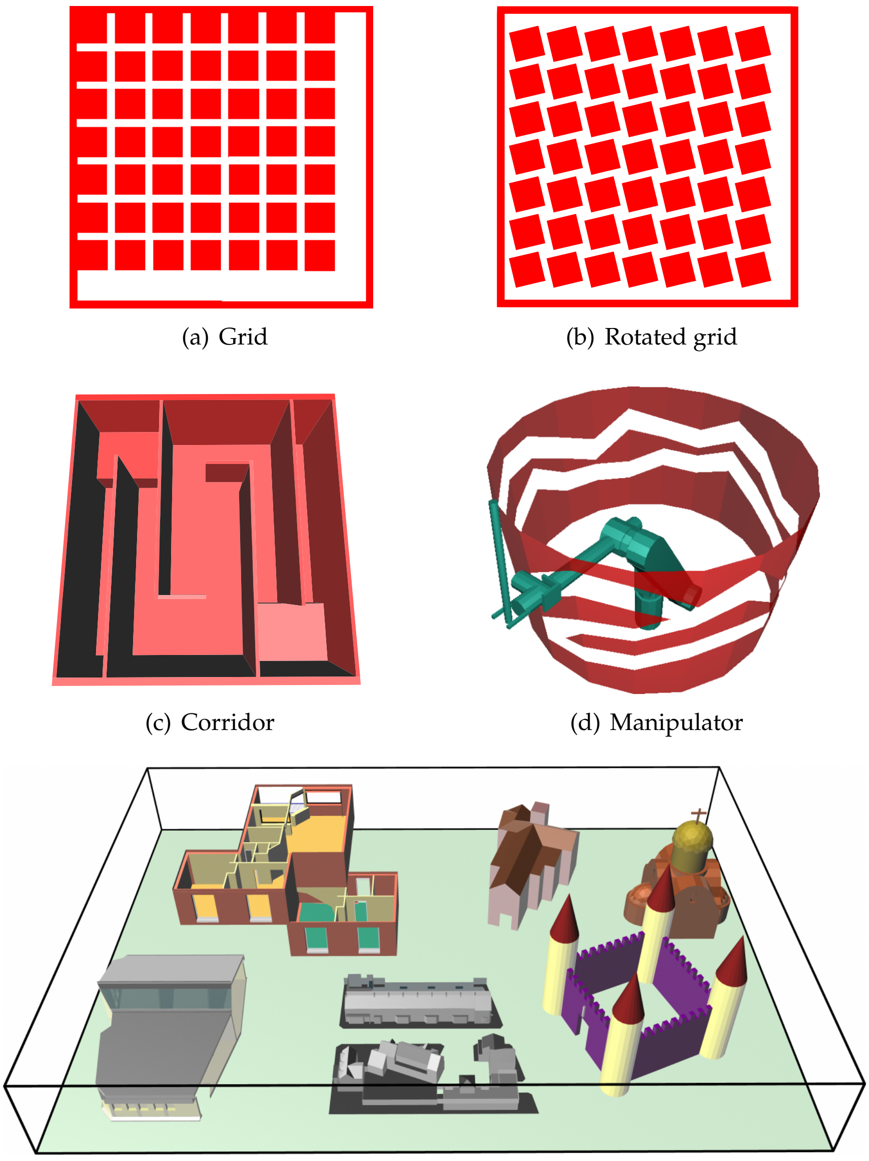
optimal values for the parameters will cause the PRM and Visibility PRM to terminate faster, but larger graphs will be produced.

We record the following statistical data: the construction time of the roadmap, the number of nodes $|V|$ and the number of edges $|E|$ in the roadmap. Furthermore, for each environment and technique we measure the length of the roadmap $|G|$ which we calculate as the sum of the lengths of the edges in the roadmap.

6.5.2 Experimental results

When we conducted the experiments, we experienced that the PRM and the Visibility PRM in particular were not always able to find a solution within the maximum running time that had been set to two hours. In those cases we conducted only 10 experiments.[3] The roadmaps created by the RRM are visualized in Figure 6.5. The results are stated in Table 6.3. Figure 6.6 shows the box plots of the running times.

[3]The PRM could not find any solution for the Manipulator and Toy village environments. The Visibility PRM only found solutions for the Grid and Corridor environments.



(a) Grid

(b) Rotated grid

(c) Corridor

(d) Manipulator

(e) Toy village

Figure 6.4 The five test environments.

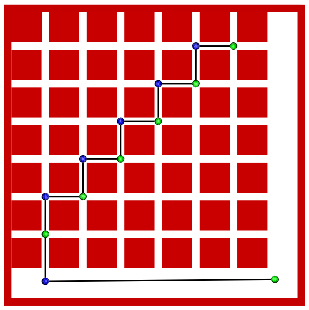
Grid The RRM computed a roadmap that has the minimum number of nodes. The roadmap length is close to optimal. Hence, our algorithm is competitive compared to the optimal solution. While the PRM created much larger roadmaps, the Visibility PRM produced reasonably small ones but this took on average more time than the other two techniques. The box plots of the Visibility PRM in Figure 6.6 show that this high average running time is caused by a few outliers. As this technique sometimes creates artificial narrow passages [121], it may be difficult to solve the problem. When the optimal connection parameters were used for the PRM, its running times were comparable to the running times of the RRM, but the PRM produced considerably larger roadmaps.

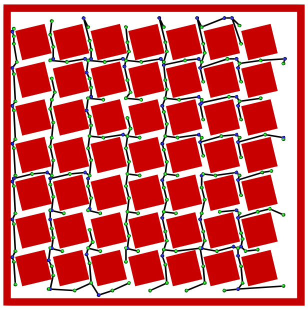
Rotated grid As the reachability regions are much smaller in this environment than the regions in the Grid environment, more nodes are required to get the free space covered. The RRM needed far less nodes and edges than the PRM, because the RRM tries to minimize the amount of double coverage. In addition, the resulting roadmap length was much smaller. The Visibility PRM was unable to solve the problem within two hours.

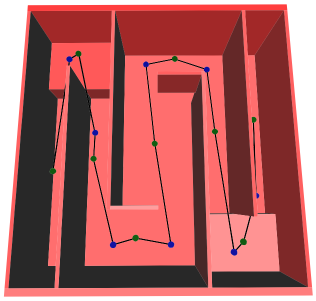
Corridor As expected, the RRM created a small roadmap, consisting of 17 nodes and 16 edges. By lack of narrow passages, the PRM was considerably faster than the RRM. However, the PRM created relatively large roadmaps. The Visibility PRM creates small roadmaps but its variance in running time was very large.

Manipulator The RRM required 313 nodes and 306 edges to cover and connect the three-dimensional configuration space. The PRM and Visibility PRM were unable to solve the problem within two hours. This is because the free $\mathcal{C}$-space contains small regions (which complicates getting $\mathcal{C}_{\text{free}}$ covered) and narrow passages (which complicates getting the nodes connected).

Toy village As this environment has a rather non-uniform distribution of the obstacles, the PRM had difficulties in covering and connecting the small areas. The Visibility PRM was unable to solve the problem within two hours as well. In contrast, the RRM solved the problem within six minutes. The resulting roadmap concentrated the nodes in regions where much detail was present in the environment, while nodes in ample free space were relatively sparse. This resulted in a surprising small roadmap.


(a) Grid


(b) Rotated grid


(c) Corridor

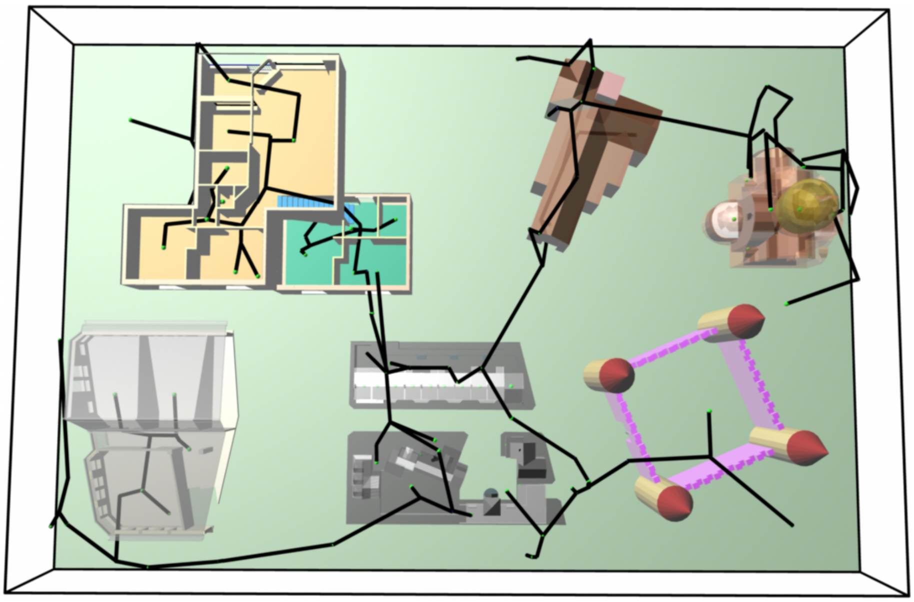

(d) Toy village

Figure 6.5 Resulting roadmaps projected onto the environments.

	Statistics – Grid (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	0.25	**13**	**12**	**88**
PRM	0.72	127	125	629
Vis. PRM	0.86	26	25	331

	Statistics – Grid (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**0.25**	13	12	88
PRM	0.27	192	193	582
Vis. PRM$^{(*)}$	0.86	26	25	331

	Statistics – Rotated grid (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	2.49	**252**	**251**	**388**
PRM	19.50	2,594	2,593	1,614
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Statistics – Rotated grid (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**2.49**	252	251	388
PRM	3.40	2,551	2,550	1,493
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Statistics – Corridor (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	0.22	**17**	**16**	**156**
PRM	0.09	46	45	324
Vis. PRM	0.74	18	17	204

	Statistics – Corridor (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	0.22	17	16	156
PRM	**0.05**	56	55	3270
Vis. PRM	0.28	24	23	207

	Statistics – Manipulator (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	9.94	**313**	**306**	**861**
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Statistics – Manipulator (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**9.94**	313	306	861
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Statistics – Toy village (size)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	351.64	**177**	**143**	**1,278**
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

	Statistics – Toy village (time)			
	time (s)	$\|V\|$	$\|E\|$	$\|G\|$
RRM	**351.64**	177	143	1,278
PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.
Vis. PRM$^{(+)}$	>7,200.00	n.a.	n.a.	n.a.

Table 6.3 Statistics for the five environments. The left column shows the statistics of the experiments that were focused on the *size* of roadmaps, i.e. we put no limits on the connection parameters for the PRM and Visibility PRM in the experiments. The right column shows the statistics of the experiments that were focused on the *construction time* of roadmaps, i.e. we used the optimal connection parameters for the PRM and Visibility PRM. $^{(*)}$The optimal maximum connection distance for the Visibility PRM was ∞. Hence, the results are the same as in the corresponding statistics in the left column. $^{(+)}$Ten out of ten runs were not solved within two hours.

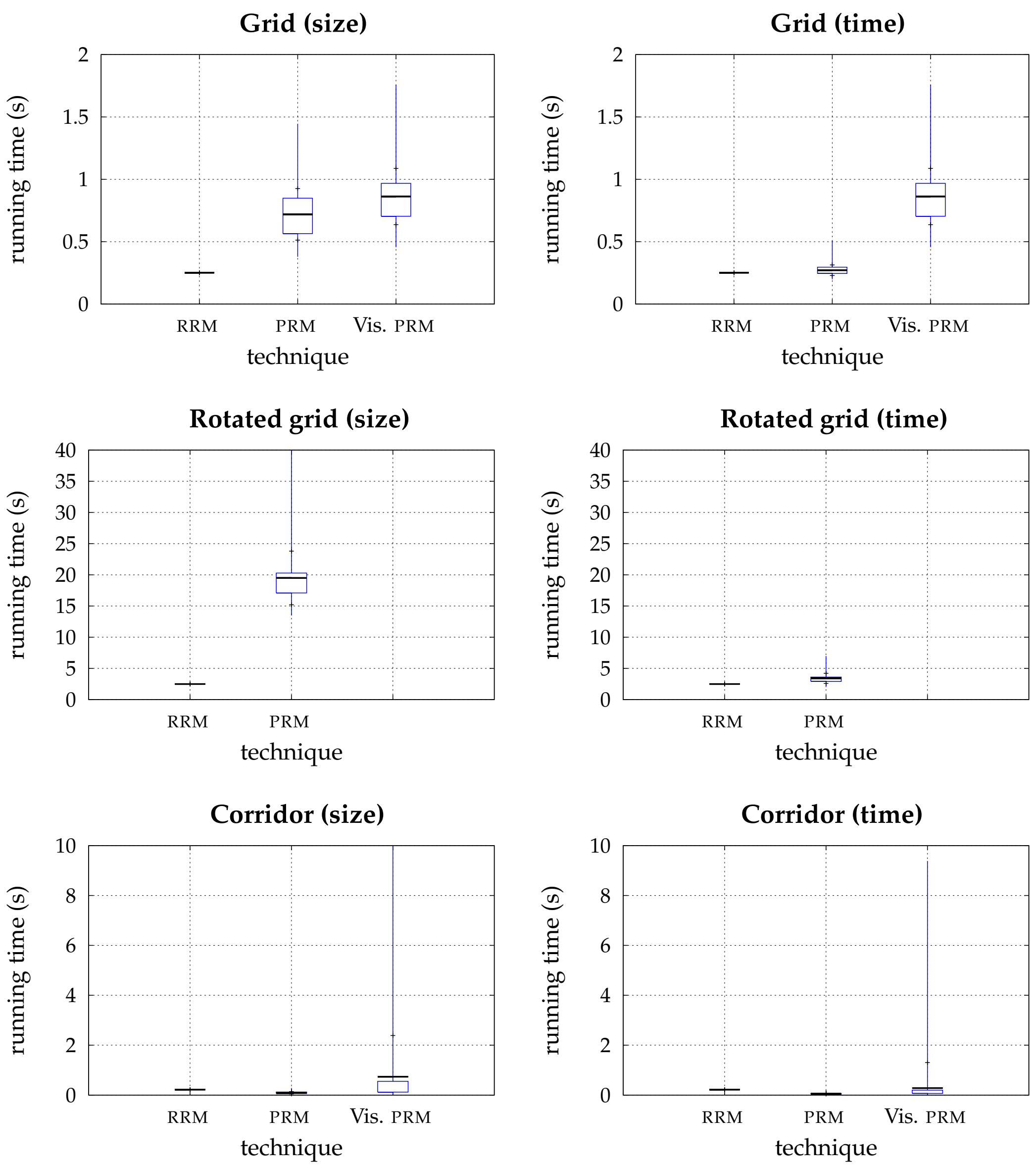


Figure 6.6 Box plots of the running times for the Grid, Rotated grid and Corridor environments. As the Visibility PRM could not solve the Rotated grid environment within two hours, no corresponding box plot is displayed.

6.6 Discussion

We presented a new efficient algorithm, the *Reachability Roadmap Method* (RRM), which creates small roadmaps for two- and three-dimensional motion planning problems. The RRM ensures that every valid query can be connected to the roadmap, as is required to solve the problem. If there exists a path in the (discretized) free space that connects the query, the algorithm ensures that a path can be found in the roadmap.

We compared the RRM with the PRM and the Visibility PRM. The RRM created roadmaps with the fewest number of nodes and edges in all environments. While the PRM produced large roadmaps, the Visibility PRM created relatively small roadmaps but had great difficulties in getting the free space covered and connected. It was surprising to see how few nodes are actually required for roadmaps satisfying the two reachability criteria.

For difficult environments containing narrow passages (e.g. the Grid, Rotated grid, Manipulator and Toy village environments), the RRM was faster than the other two techniques. In these environments, the PRM and Visibility PRM consumed much time before obtaining a path through the narrow passages. For the easy Corridor environment, the RRM was outperformed by the PRM as all samples could reach a large portion of the free space. When the number of dimensions increases and the problem does not contain very narrow passages, the RRM will be outperformed (in running time) by the PRM, as the PRM is less sensitive to the dimensionality of the problem. In addition, full coverage of the $\mathcal{C}$-space is not required in many motion planning problems as queries will be 'reasonable'. In such situations the preprocessing times for the PRM and the Visibility PRM will be much lower. But the roadmaps they produce will still be large, leading to increased query times.

An interesting topic for future research is how to efficiently extend the RRM to higher dimensions. A roadmap that is built for a lower dimensional subspace could be used to guide the motions for an object operating in a high-dimensional $\mathcal{C}$-space (see e.g. [15, 25, 47]). We believe that the Reachability Roadmap Method, and its future modifications, will enhance the quality of motion planners, in particular when query time is the major concern.

CHAPTER
SEVEN

PUTTING IT ALL TOGETHER

In this chapter we unify the techniques from previous chapters to create roadmaps that are particularly suited for motion planning in virtual environments. We use the Reachability Roadmap Method to compute an initial, resolution complete roadmap. This roadmap is small which keeps query times and memory consumption low. For use in virtual environments, there are additional criteria that must be satisfied. In particular, we require that the roadmap contains useful cycles. These provide short paths and alternative routes which allow for variation in the routes entities can take. We will show how to incorporate such cycles. In addition, we provide high-clearance paths by retracting the edges of the roadmap to the medial axis. Since all operations are performed in a preprocessing phase, high-quality paths can be extracted in real-time as is required in interactive applications.

7.1 Introduction

In many virtual environments, paths have to be planned for entities to traverse from a start to a goal position in the virtual world. A common way to plan the path is to use an A* algorithm on a (low-resolution) grid. Such a path is displayed in Figure 7.1. This search algorithm is popular because it always finds a shortest path in the roadmap if one exists. However, as contemporary virtual worlds can be very large, storing the grid and running the algorithm may consume a huge amount of memory which is not always available, in particular on systems with constrained memory such as console systems. In addition, the algorithm may consume too much processor time, especially when many paths (for different entities) have to be planned simultaneously.[1] This will lead to stalls in interactive applications. Paths resulting from A* algorithms tend to have little clearance and can be jaggy and aesthetically unpleasant, so care must be taken to smooth them.

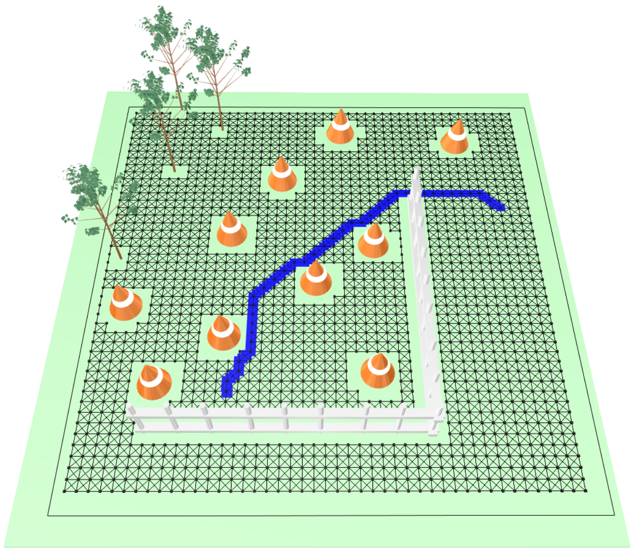


Figure 7.1 A shortest path that was found by an A* algorithm in the displayed roadmap. This roadmap was constructed by placing a node on each corner of each free grid cell. Edges were added for each boundary and each diagonal of a free cell. This resulted in a roadmap with 1,793 nodes and 6,553 edges.

[1]An A* algorithm is most inefficient in determining that there is no path between the start and goal positions. It will then examine every possible cell in the grid accessible from the start [43].

Another popular motion planning technique is the Probabilistic Roadmap Method (PRM), which we discussed in Chapter 2. A drawback of the PRM is that a resulting roadmap often contains many redundant nodes and edges, in particular when the environment contains one or more narrow passages. In addition, the roadmap may contain many short edges which complicates the smoothing phase that often follows a query phase [117].

Nieuwenhuisen *et al.* [117] improve a roadmap generated by the PRM such that it can be used for path planning in games. Their method guarantees that the paths are short, have enough clearance from the obstacles, and are C^1 continuous, leading to natural looking motions. Such a path can be retrieved almost instantaneously. Their method does not guarantee that a path can always be found (if one exists in the free space $\mathcal{C}_{\text{free}}$). In addition, the method is limited to two-dimensional problems.

Sometimes, roadmaps are created manually from which paths can be extracted. However, this may take many hours of precious time.[2]

Our goal is to automatically create a roadmap for 2D and 3D (possibly large and complex) environments that can be used to guide the motions for one or more entities in a virtual environment. By a careful integration of existing and new techniques, we aim at generating roadmaps with the following four properties:

1. The roadmap is *resolution complete*. This means that a valid query (which consists of a start and a goal configuration) can always be connected to the roadmap. If the start and goal belong to the same connected component of the free space, then a corresponding path can always be found (at a given resolution).

2. The roadmap is *small*. A small roadmap assures low query times and low memory consumption. A path that is extracted from a small roadmap will have reasonably long edges. These are easier to optimize. For example, Nieuwenhuisen *et al.* [117] add circular arcs to the roadmap to make the paths C^1 continuous, resulting in natural looking motions. In addition, when a roadmap must obey other criteria, a small roadmap eases manual tuning.

3. The roadmap contains *useful cycles*. These cycles provide short paths and alternative routes which allow for variation in the routes that entities take. Van den Berg *et al.* [14] exploit cycles in dynamic environments

[2]It is mentioned in [1] that a designer spends roughly 8 hours setting up and debugging the roadmap for an indoor level and up to 20 hours for a large outdoor level.

where additional obstacles might appear, and to avoid deadlock situations when multiple robots move in the same environment.

4. The roadmap provides *high-clearance paths*. By retracting the roadmap to the medial axis, paths with much clearance can be extracted in real-time. High-clearance paths work well with entities that have large widths, such as a wide formation of characters. In addition, they are perfectly suitable for guiding the motions of a group of entities or for creating a useful backbone path for the animation of walking characters, see the work of Kamphuis *et al.* [77–79].

In Chapter 6, we have introduced the Reachability Roadmap Method (RRM) which already satisfies the first and second property. In Section 7.2, we propose an algorithm that adds useful cycles to the roadmap. We meet the fourth property in Section 7.3 which shows how to retract a roadmap to the medial axis. We perform experiments with 2D and 3D virtual environments in Section 7.4 and conclude in Section 7.5 that our algorithm successfully creates roadmaps satisfying these four properties.

7.2 Adding useful cycles

In Section 5.1, we discussed three methods for decreasing the path length. Although these methods can decrease the path length considerably, they usually do not remove the detours around obstacles. These detours can be avoided by adding cycles to the roadmap. Besides obtaining shorter paths, cycles provide alternative routes for an entity.

In the following subsections, we will show how to add useful cycles to the roadmap. Our strategy is partly based on the work of Nieuwenhuisen and Overmars [119] which adds *useful edges* to the roadmap. An edge is useful if it introduces a cycle that improves the roadmap according to some criterion. As we are working with small roadmaps, unfavorably placed queries can still lead to long paths. We will show that these can be avoided by adding *useful nodes* and their corresponding edges to the initial roadmap. The final roadmap will then be composed of all nodes from the initial roadmap, as well as the added useful nodes and useful edges between those nodes.

7.2.1 Useful edges

Nieuwenhuisen and Overmars [119] propose a technique that adds useful cycles to the roadmap. The goal is to add only those edges that have a high probability of introducing a path that cannot be continuously deformed into an existing path.[3] A *useful edge* is defined as follows:

Definition 7.1 (Useful edge). *Let v be the node that corresponds to configuration c which has been added to the graph and V^n its set of neighbors. Let v' be a node in V^n and $d(v, v')$ be the distance between v and v'. The graph distance between v and v' is $G(v, v')$ which is the length of the shortest path in the graph from v to v'. If there is no path from v to v', $G(v, v')$ is ∞. Then edge $\epsilon(v, v')$ is K-useful if*

$$K * d(v, v') < G(v, v').$$

This definition only adds an edge to the graph (roadmap) between v and v' if their graph distance improves by a factor K. A small value of K adds more edges than a large value of K. Figure 7.2 shows that no cycles are added if K is set to ∞. If $K \leq 1$, then all collision-free edges (i.e. local paths) are allowed. The authors use a pruned version of Dijkstra's shortest path algorithm to efficiently determine whether a particular edge is useful. They also show, when time goes to infinity, that their approach will find a path with a length converging to $K * |\Pi|$, where $|\Pi|$ denotes the length of shortest possible path Π.[4] Hence, the larger the number of nodes in the roadmap (and the smaller the value of K), the shorter the expected length of a path. As one of our criteria is obtaining a small roadmap, these two conflicting criteria (short path length and small roadmap) need to be balanced.

7.2.2 Adding useful nodes

Figure 7.3 displays the same roadmaps as Figure 7.2, but unfavorably placed start and goal positions were added and connected to the roadmap. As few nodes were placed in the middle of the environment, such a small roadmap can yield long paths, making detours around the obstacles. We handle this problem by adding *useful nodes* (and useful edges) to the roadmap, while attempting to keep it small. Accordingly, we define a useful node as follows:

[3]If a path cannot be continuously deformed into an other particular path, these paths are said to be in different homotopic classes. Schmitzberger *et al.* [135] conducted research on identifying all homotopic classes. This may only work well in 2D, since, in higher dimensions, solutions are often in the same homotopic class but are hard to continuously distort into each other. Hence, identifying these solutions seems to be of little use for motion planning purposes.

[4]In practice, however, we do not have an infinite amount of time or memory, and hence, we have to decide how many nodes the roadmap can contain. In our case, this number of nodes is determined by the output of the Reachability Roadmap Method.

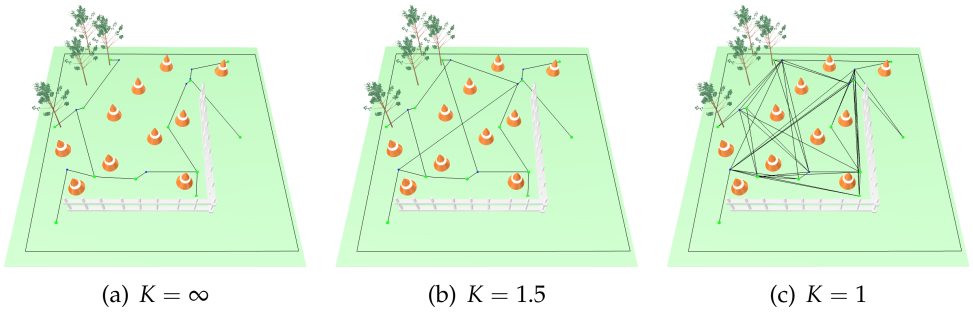

(a) $K = \infty$ (b) $K = 1.5$ (c) $K = 1$

Figure 7.2 The influence of parameter K on the number of cycles. Picture (a) shows a roadmap created by the Reachability Roadmap Method for a 2D rigid, translating square. As K was set to infinity, the roadmap remains a tree. In (b), three additional edges were introduced when K was set to 1.5. Picture (c) displays the roadmap containing all collision-free connections between each pair of nodes, as obtained with $K = 1$.

Definition 7.2 (Useful node). *Let $c \in \mathcal{C}_{\text{free}}$ be a configuration and v be its representing node. Let $\{v', v''\}$ be its two closest neighbors in the graph to which a collision-free connection exists, i.e. edge $\epsilon(v, v') \in \mathcal{C}_{\text{free}}$ and edge $\epsilon(v, v'') \in \mathcal{C}_{\text{free}}$. Let $d(v, v')$ and $d(v, v'')$ be the distances to these neighbors. Furthermore, let $G(v', v'')$ be the graph distance between v' and v'' and Π be the corresponding path in G. If there are no two closest neighbors to which a connection can be made, then $G(v', v'')$ is zero. Then node v is L-useful if*

$$L * (d(v, v') + d(v, v'')) < G(v', v'') \wedge \exists v_i \in \Pi : \epsilon(v, v_i) \in \mathcal{C}_{\text{forb}}.$$

As one of our goals is to obtain high-clearance paths, we only select candidate useful nodes that lie on the medial axis. Definition 7.2 says that a node v is L-useful if it satisfies two conditions, see Figure 7.4 and Algorithm 7.1. First, if v can be connected to two neighbors, then the length of these two connections times the factor L must be shorter than the length of the shortest path in G between those neighbors. Second, the new cycle must guide the entity around an obstacle, i.e. there must be at least one connection from node v to the nodes describing the shortest path Π which causes the moving object to collide with the obstacles. We limit ourselves to checking connections between nodes because checking all connections from node v to each configuration on path Π will consume too much time.

As a clarification, we apply Algorithm 7.1 on our running example. Figure 7.5 shows the resulting roadmaps for three different values of L. A smaller value of L yields more nodes and edges. If L equals zero, then only the second

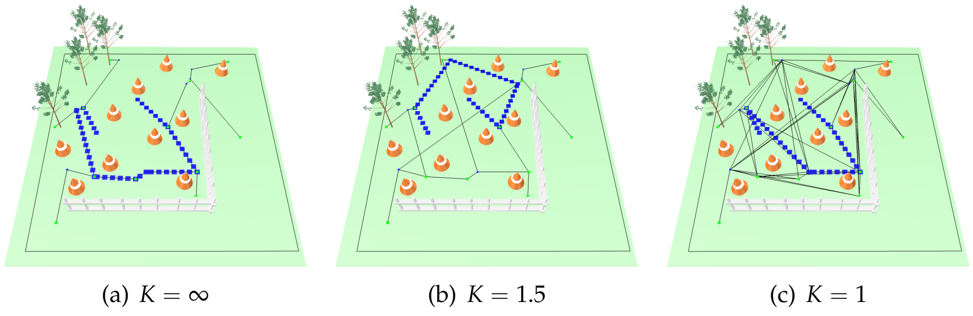


Figure 7.3 A roadmap that contains few nodes can lead to long paths for unfavorably placed queries. Even when parameter K is set to one, i.e. when all collision-free connections are added as edges to the roadmap, the extracted path can be long compared to the optimal path.

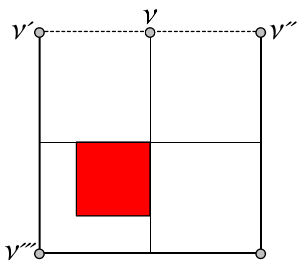
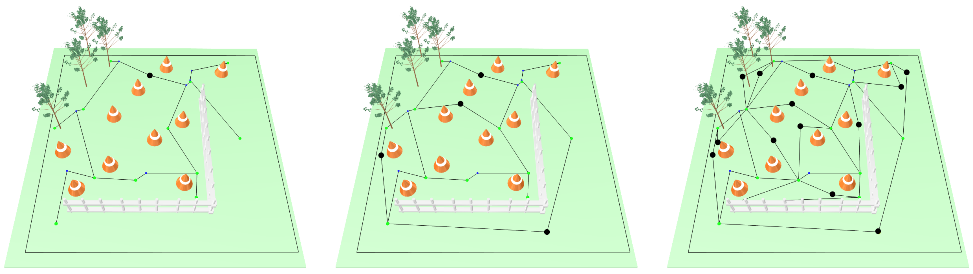


Figure 7.4 Node v is L-useful for $L < 3$ because $L * (1 + 1) < 6$ and edge $\epsilon(v, v''') \in \mathcal{C}_{\text{forb}}$. If the value for L is less than three, then node v and edges $\epsilon(v, v')$ and $\epsilon(v, v'')$ are added to the graph, introducing a useful cycle.

condition of Definition 7.2 has to hold, i.e. a node v is useful if there is at least one edge from v to a node in the cycle that causes the robot to collide.

Figure 7.5 shows that the useful nodes have a tendency to spread. This can be explained as follows. Suppose that a candidate node v is very close to a previously added useful node v' which has already been added and connected to the roadmap. Then it is likely that v has the same two neighbors as v' to which free connections exist. (Node v will not be connected to v' as useful nodes are only connected to nodes from the initial roadmap to keep the roadmap small.) As their edges will lie close together, it is unlikely that v is part of a connection that collides with an obstacle. As a result, node v is not labeled according to Definition 7.2.



(a) $L = 3$ (b) $L = 1.5$ (c) $L = 0$

Figure 7.5 Adding useful nodes. The black discs represent the added useful nodes. Each of these nodes has been connected to the two nearest neighbors from the input graph to which a collision-free connection exists.

7.2.3 Reconnecting the edges

The roadmap quality can be further improved by rearranging its edges. Our approach creates a graph G' that consists of all nodes from graph G. Then, for each pair of nodes, we check if the connection between them is collision-free. Such a connection is added as an edge to G' if the edge is K-useful. The approach is outlined in Algorithm 7.2. First, we create a priority queue (sorted on increasing edge length) and fill it with all collision-free connections between pairs of nodes from graph G. The improved graph G' will have the same nodes as graph G. The edges of G' consist of edges extracted from the queue if the two nodes do not belong to the same connected component[5] or if the edge introduces a K-useful cycle. Due to the reconnection of the edges, it can occur that a (useful) node is no longer part of a cycle. If such a node has degree one, than this node is removed as our goal is to keep the roadmap small. Figure 7.6 shows the effect on the roadmaps of using different values for L.

Figure 7.7 gives an indication of changes in path length of the unfavorably placed query in several phases of the running example. In each picture, the same initial roadmap was used (which was produced by the Reachability Roadmap Method). Figure 7.7(a) shows the path for the roadmap to which useful cycles were added. This path is rather long. A shorter path was found in (b), which shows the roadmap after adding useful nodes and their corresponding connections. Yet a shorter path was found after reconnecting the edges, which is shown in (c).

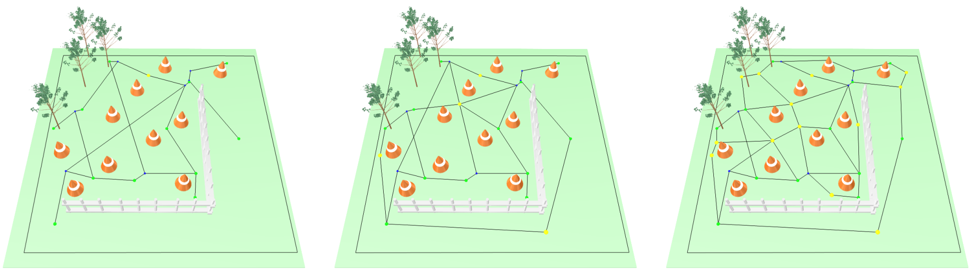
[5]Note that Definition 7.1 already guarantees this criterion, i.e. if there is no path between the two nodes, then the graph distance is set to infinity which validates the condition.

Algorithm 7.1 ADDUSEFULNODESANDCYCLES(graph $G(V,E)$, factor L)

```
1: $MA \leftarrow$ list of configurations, sampled on the medial axis
2: for all $c \in MA$ do
3:     $v \leftarrow$ node that represents configuration $c$
4:     $v', v'' \leftarrow$ the two closest neighbors of the input graph to $v$ for which
       $\epsilon(v,v'), \epsilon(v,v'') \in \mathcal{C}_{\text{free}}$
5:     $\Pi \leftarrow$ shortest node path in $G$ from $v'$ to $v''$
6:     if $L * (d(v,v') + d(v,v'')) < |\Pi|$ then
7:         $addCycle \leftarrow$ false
8:         for all $v_i \in \Pi$ do
9:             if $\epsilon(v,v_i) \in \mathcal{C}_{\text{forb}}$ then
10:                $addCycle \leftarrow$ true
11:                break
12:        if $addCycle$ = true then
13:            $V \leftarrow V \cup v$
14:            $E \leftarrow E \cup \epsilon(v,v')$
15:            $E \leftarrow E \cup \epsilon(v,v'')$
```

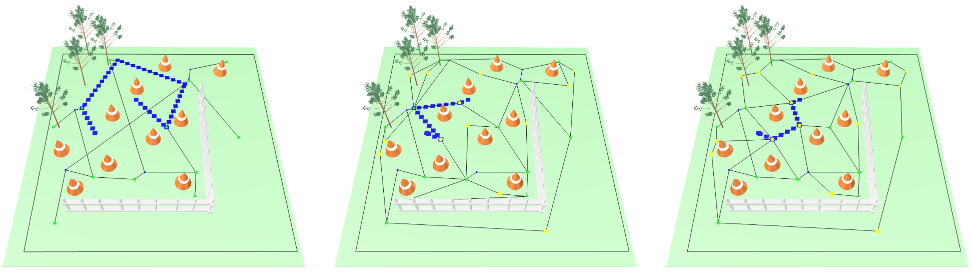
Algorithm 7.2 RECONNECTEDGES(graph $G(V,E)$, factor K)

```
Output: graph $G'(V',E')$
1: PriorityQueue $Q$ {sorted on increasing edge length}
2: for all pair of nodes $v', v'' \in V : v' \neq v''$ do
3:     if edge $\epsilon(v',v'') \in \mathcal{C}_{\text{free}}$ then $Q$.push($\epsilon(v',v'')$)
4: $V' \leftarrow V$
5: $E' \leftarrow \emptyset$
6: while not $Q$.empty() do
7:     edge $\epsilon(v',v'') \leftarrow Q$.top()
8:     $Q$.pop()
9:     if $K * d(v',v'') < G(v',v'')$ then $E' \leftarrow E' \cup \epsilon$
10: Remove all useful nodes from $V'$ with degree 1
```



(a) $K = 1.5$ and $L = 3$ (b) $K = 1.5$ and $L = 1.5$ (c) $K = 1.5$ and $L = 0$

Figure 7.6 Reconnection of the edges with $K = 1.5$.



(a) Useful cycles: length query = 63.8 (b) Useful nodes: length query = 25.9 (c) Useful nodes, useful cycles and edges reconnected: length query = 18.3

Figure 7.7 Path lengths of an unfavorably placed query. The factor K for adding useful cycles is 1.5.

7.3 Retracting a roadmap to the medial axis

The fourth criterion which a roadmap must obey is that high-clearance paths can be obtained in real-time. A path has a high clearance if it follows the medial axis of the free configuration space. We provided two algorithms that retract *paths* to the medial axis in Chapter 4. Recall that Algorithm 4.2 is relatively fast, but it can only be applied to rigid, translating bodies with two or three DOFs. In contrast, Algorithm 4.4 is relatively slow, but it can be applied to a broader range of robots and generally provides a larger clearance for robots with more than two DOFs.

Rather than retracting paths, our goal now is to retract the entire roadmap to the medial axis. Algorithm 7.3 outlines our approach. To ensure that the

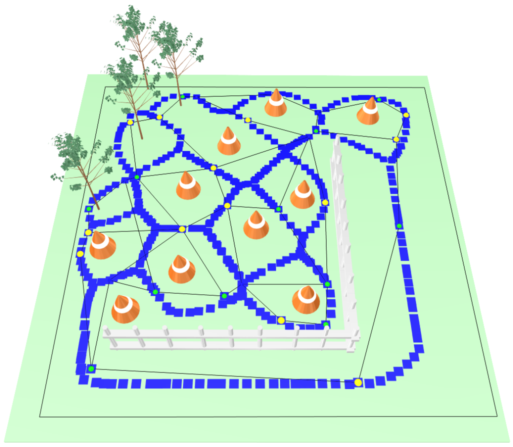


Figure 7.8 Retraction of the roadmap. For each edge in the graph, the local path that corresponds to the edge lies on the medial axis.

complete roadmap will be retracted to the medial axis, we require that its nodes initially lie on the medial axis. Note that the Reachability Roadmap Method already satisfies this requirement. For each edge ϵ, let Π be the local path that corresponds to the edge. If the number of (translational) DOFs of the entity equals two, then we retract this local path by applying Algorithm 4.2. In other cases, we apply Algorithm 4.4.

Algorithm 7.3 RETRACTROADMAP(graph $G(V, E)$)

```
Require: $\forall v \in V$: each configuration that corresponds to node $v$ lies on the
    medial axis
1: for all edges $\epsilon(v', v'') \in E$ do
2:     $\Pi \leftarrow$ LP$[v', v'']$
3:     if number of DOFs equals 2 then $\Pi \leftarrow \mathcal{W}$-RETRACTION($\Pi$) {see Algo-
       rithm 4.2}
4:     else $\Pi \leftarrow \mathcal{C}$-RETRACTION($\Pi$) {see Algorithm 4.4}
```

As an example, we apply Algorithm 7.3 on our running example. The input graph of Figure 7.8 was created in three steps. A small covering roadmap was produced by the Reachability Roadmap Method. Then useful nodes and edges were added. These edges were then retracted by the appropriate algorithm. Note that some retracted edges have overlapping parts. We do not attempt to merge them as this will increase the number of nodes in the roadmap.

7.4 Experiments

In this section, we test our approach on four virtual environments. In the first part of the experiments, we compare the roadmaps produced by the following three algorithms. The first algorithm, which we refer to as the *Grid Roadmap Method* (GRM), creates a grid. This algorithm is often used in the game community, see Figure 7.1 for its construction. The second algorithm is the Reachability Roadmap Method (RRM) from Chapter 6. The third algorithm (RRM*) uses the RRM as input and adds useful nodes and edges. Its edges are then rearranged.

In the second part of the experiments, we retract the edges of the roadmaps produced by the RRM* to the medial axis of the free space to obtain high-clearance paths. We refer to this combination of methods as the Retract RRM* (RRRM).

7.4.1 Experimental setup

We conduct experiments with the four environments depicted in Figure 7.9. Information on the environments and robots is listed in Table 7.1. The environments have the following properties:

Field This small 2D environment contains ten cones, two fences and four trees. These obstacles are cluttered in a large part of the environment. The other part is rather empty. There are many alternative routes. We will test whether our algorithm can capture most of them.

Office This large 3D environment with more than 80 pieces of furniture (79,000 geometrical objects) has a rather non-uniform distribution. There are large open spaces and many narrow passages, requiring a large grid to capture the connectivity of $\mathcal{C}_{\text{free}}$. Also this environment contains many alternative routes. The results will show whether our algorithm can capture them. We will also investigate how well our algorithm deals with many obstacles and large environments.

House This 3D environment has twelve rooms. There are few different routes from one room to another room. Hence, we expect that few cycles will be added to the reachability roadmap. Each edge in the roadmap will be retracted to the medial axis of the environment. We will investigate how much the clearance improves along the roadmap.

	Bounding boxes			
	environment	**robot**	**grid resolution**	**# objects**
Field	47×47	1×1	94×94	16,000
Office	80×80	$1 \times 1 \times 4$	160×160	79,000
House	$57 \times 20 \times 40$	$3 \times 3 \times 3$	$57 \times 20 \times 40$	1,000
Quake	$130 \times 25 \times 80$	$1 \times 1 \times 1$	$130 \times 25 \times 80$	4,000

Table 7.1 Information on the environments and robots used in the experiments.

Quake This 3D environment has been converted from a level from the game Quake.[6] There are many alternative routes. We will investigate how much the average path length decreases when we add useful cycles. Furthermore, we will test whether clearance can be added successfully to roadmaps of problems involving three DOFs.

When we add cycles to a roadmap, we need a way to measure the improvement. For this purpose we define the *Shortest path factor* (SPF):

Definition 7.3 (Shortest path factor). *Let $G' = (V', E')$ be the graph in which node $v' \in V'$ is placed on each corner of each free cell in a grid representation of the environment and let E' be the set of edges placed on each border and diagonal of each free cell. Let $G = (V, E)$ be a graph for which $V \subseteq V'$. Furthermore, let Ψ be the set of shortest paths in G between each pair of nodes in V, n be the number of paths in Ψ and Ψ' be the set of shortest paths in G' between each pair of nodes in V. Finally, let $|\cdot|$ denote the length of a path. Then the* Shortest path factor *is defined as follows:*

$$\text{SPF} = \sum_{i=1}^{n} |\Psi_i| / \sum_{i=1}^{n} |\Psi'_i|.$$

This definition provides a factor that describes how much longer the expected length of a path (between two nodes of graph G) is compared to the optimal path length in the grid. If the factor equals one, than each extracted path will be the shortest one in the grid. The larger this factor, the larger the detour made by the moving entity. For all environments and techniques, we define 100 random queries and report how many seconds it takes to solve them. Regarding the Grid Roadmap Method, finding the closest free neighbor can be done in $O(1)$ time. Hence, we only report the time needed for running Dijkstra's

[6]See www.quake.com.

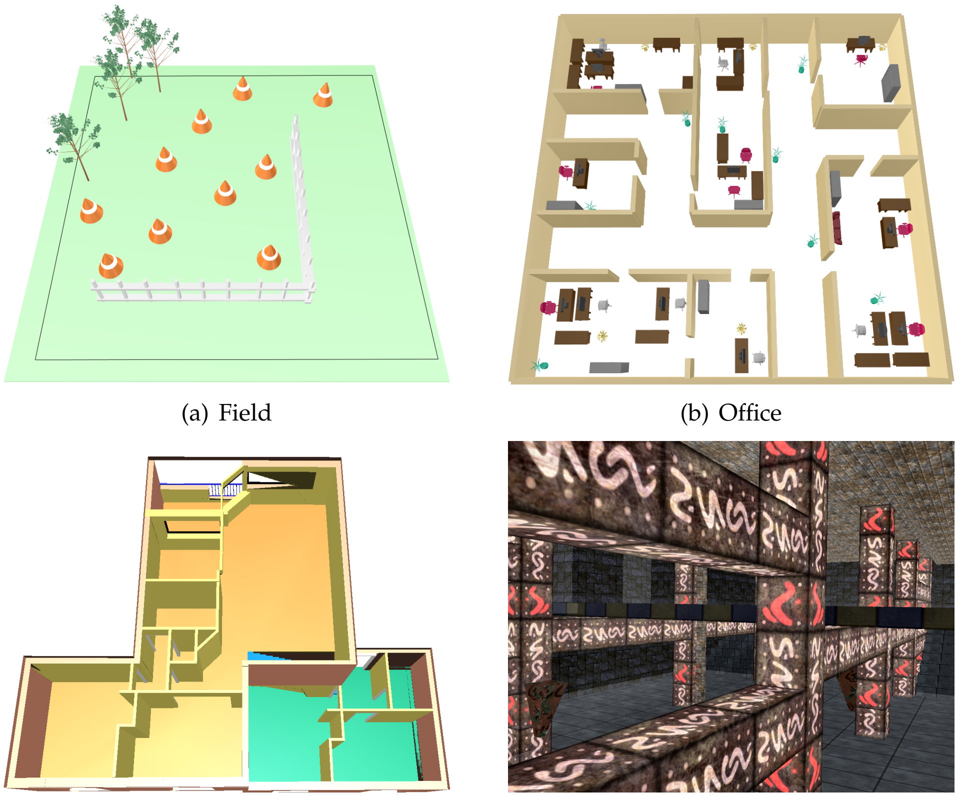

(a) Field (b) Office (c) House (d) Quake

Figure 7.9 The four test environments.

shortest path algorithm. In contrast, the other two methods require finding the closest free neighbors *and* running Dijkstra's algorithm.

We also keep track of the sizes of the roadmaps, the construction times, the query times and clearance information (i.e. minimum, average and maximum clearance of configurations in the roadmap).

7.4.2 Experimental results

In the following paragraphs, we will describe the results of adding useful cycles and nodes, and adding clearance to the roadmap. See Figure 7.10 for visualizations of these results.

Adding useful cycles and nodes

For each environment, we created a roadmap by applying the Grid Roadmap Method (GRM), the Reachability Roadmap Method (RRM) and the modified RRM (RRM*) method. We set parameter K to 1.5 and L to 0. The results are stated in Table 7.2 and are discussed below:

Field Even for this relatively small environment that was discretized with 94 by 94 cells, the GRM produced a huge roadmap. As a result, computing 100 random queries took 1.36 seconds. These running times may be acceptable for real-time behavior. An advantage of the method is that a shortest path in the grid will always be found, but as indicated above, the path lacks clearance as it runs very close to obstacles. The RRM created a small roadmap consisting of 29 nodes and 18 edges. As the graph was small, connecting a query to the roadmap and running Dijkstra's algorithm on average took 4.3 ms. Hence, this roadmap can be used in real-time situations. However, due to the small size of the roadmap, the off-center placement of the nodes in the environment, and the fact that the roadmap does not have cycles, the shortest path factor is rather high (1.57). This means that an extracted path between two nodes in the roadmap is on average 57% larger than the corresponding shortest path in the grid of the GRM roadmap. The RRM* added 14 useful nodes and 29 useful edges to the roadmap. As a result, the SPF decreased to 1.137 which means that an extracted path will be much shorter than an extracted path in the RRM roadmap. This did not have a negative impact on the extraction times of the queries.

Office Again, the GRM created a huge roadmap containing 16,917 nodes and 62,917 edges. Finding a shortest path between existing nodes in this roadmap took 34 ms on average. The RRM created a small roadmap. The RRM* added 15 nodes and 33 edges to this roadmap, improving the SPF with 53 percent points. Hence, an extracted path will make less detours. In addition, a close inspection of this roadmap in Figure 7.10 shows that many alternative routes have been introduced. This did not result in longer query times. Note that these three methods only needed a few seconds to discretize the $\mathcal{C}$-space (35,600 collision checks) and to create the roadmaps. Hence, the RRM and RRM* methods function well in environments with a lot of detail and many obstacles.

House When we go from two-dimensional to three-dimensional problems, a roadmap produced by the GRM will inevitably become huge (i.e. 40,088 nodes and 454,250 edges), even for relatively small environments such as this environment. Running a query on average took 135 ms, which is far too long in real-time situations. It was surprising to observe that the RRM only needed 34 nodes and 33 edges to get the free configuration space covered and connected. Apparently, the nodes were properly placed in each room, as the RRM* did not add any useful node to the roadmap. Only one edge was added. Running a query took on average 8 ms. An extracted path will be short as it will be only 23% larger than the corresponding optimal path in the grid.

Quake This environment was discretized with 260,000 cells. The corresponding roadmap created by the GRM became huge ($>$ 1.5 million edges). As a result, extracting a query took 0.64 seconds on average. The RRM created a small roadmap (71 nodes and 65 edges). This sparse roadmap lead to a high SPF. The expected path length between two nodes in the roadmap was two times the length of the shortest path in the grid. By adding 61 nodes and 151 edges, the RRM* reduced this factor substantially to 1.194. Extracting a query took on average 42 ms, which may be too high in a real-time situation. Connecting the query to the graph consumed relatively much time as many connections had to be checked for collisions.

In conclusion, roadmaps created by the GRM rapidly become huge, especially in 3D environments, resulting in query times that are too high for real-time performance. In practice, much smaller grids are used but this can be problematic when there are narrow passages. In contrast, the RRM creates small roadmaps that completely cover the free space and have low query times. However, the extracted paths can be long. The RRM* combines the advantages of GRM and RRM, i.e. reasonably short paths are produced while the extraction times are kept relatively low. The resulting paths were on average 14 to 23 percents larger than the optimal paths in a corresponding grid. Since the RRM* placed the nodes on the medial axis, a large clearance caused the nodes to lie far away from the obstacles, increasing the path length.

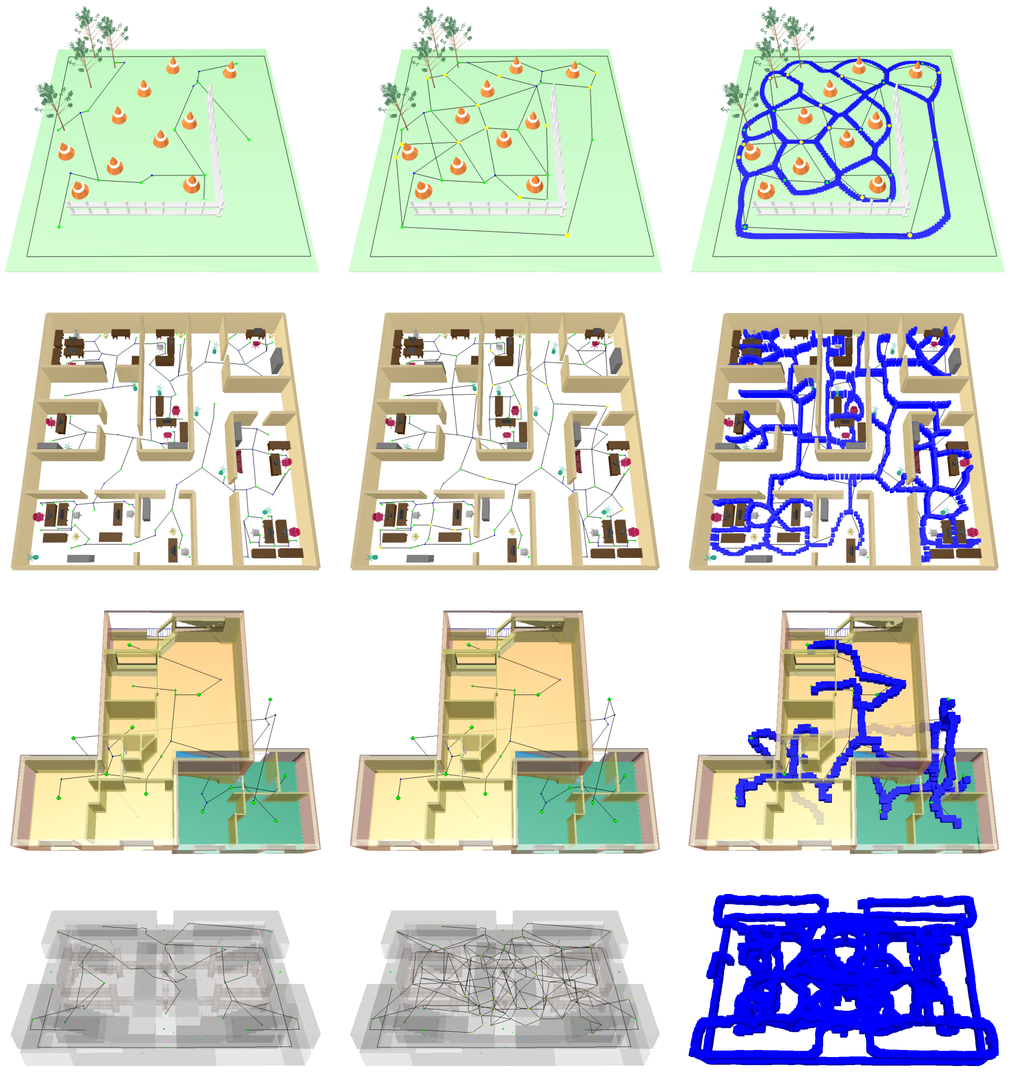


Figure 7.10 The results for the Field, Office, House and Quake environments. The left column shows the initial roadmaps created by the Reachability Roadmap Method. The middle column shows the roadmaps to which useful nodes and cycles were added (RRM*). The right column shows the roadmaps to which clearance was added (RRRM).

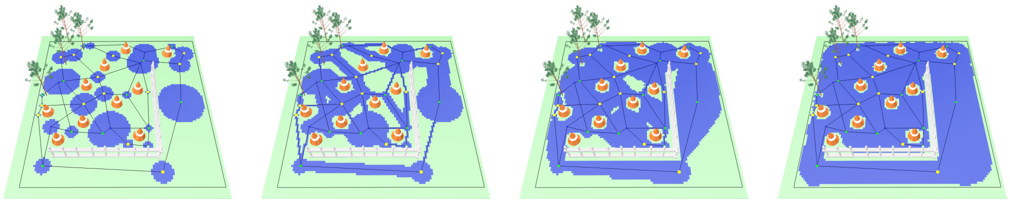
Field	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**100 queries (s)**
GRM	0.88	7,350	27,741	1.000	1.36
RRM	0.75	29	18	1.570	0.43
RRM*	0.91	43	47	1.137	0.23

Office	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**100 queries (s)**
GRM	0.89	16,917	62,297	1.000	3.42
RRM	1.10	154	147	1.812	0.38
RRM*	2.70	167	180	1.181	0.36

House	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**100 queries (s)**
GRM	12.91	40,088	454,250	1.000	13.48
RRM	11.67	34	33	1.225	0.82
RRM*	18.68	34	34	1.224	0.82

Quake	Graph statistics			Path statistics	
	time (s)	$\|V\|$	$\|E\|$	**SPF**	**100 queries (s)**
GRM	210.16	134,492	1,511,241	1.000	63.54
RRM	306.44	71	65	2.068	2.71
RRM*	384.90	132	216	1.194	4.15

Table 7.2 Roadmap statistics for the four environments. Three methods were compared. We collected the following statistical data: the construction time of the roadmap (in seconds), its number of nodes $|V|$ and edges $|E|$. Then we mention the shortest path factor (SPF) and the summed running time of 100 random queries (in seconds).



(a) Nodes: 38.5% (b) Nodes and edges: 57.7% (c) Each configuration: 80.1%. (d) Each retracted configuration: 98.5%.

Figure 7.11 Using clearance information of the roadmap to obtain collision-free motion planning. The number denotes the percentage covered of $\mathcal{C}_{\text{free}}$.

Improving query performance

The largest portion of the query times for RRM and RRM* is occupied by checks for collision-free connections from the query position to the roadmap. We could remove the need for collision checks by using the clearance information of the roadmap. For example, let c be a configuration that has to be added as a node to the roadmap and c' be a configuration located on the roadmap. If distance $d(c, c') <$ Clearance(c'), then we know that c can be connected to c'. Since distance calculations between configurations are much faster than collision checks, using clearance information, gathered in the preprocessing of roadmaps, decreases query time significantly.

As an example, we apply this idea to the Field environment, see Figure 7.11. First, we only use the clearance information of the nodes in the roadmap. Figure 7.11(a) shows a collection of discs centered at the nodes. The radius of a disc equals the distance of (the configuration that corresponds to) the node to the nearest obstacle. A query does not collide with the obstacles when it lies in these discs. The total coverage of the discs is 38.5% of the free configuration space. If we also use the minimum amount of clearance along the paths that correspond to the edges and place a disc along each configuration on the paths, the percentage improves to 57.7 which is shown in Figure 7.11(b). The amount of coverage can be further improved by considering all configurations on all local paths of the roadmap. Figure 7.11(c) shows that this improves the coverage to 80.1%. The highest amount of coverage can be obtained by using each configuration in a *retracted* roadmap. This leads to a coverage of 98.5% of the free space. In this case, the retracted roadmap consists of 870 configurations. Hence, at most 870 distance calculations are needed to check if a configuration can be connected to the roadmap. Checking whether one configuration can be connected takes 0.073 ms which makes the approach suitable for real-time planning. Even when the roadmap is larger (which is the case in the other en-

Field	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.03	2.71	6.44	
RRRM	0.34	3.08	6.46	24

Office	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.00	1.60	6.82	
RRRM	0.01	1.77	7.53	320

House	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.00	2.17	5.64	
RRRM	0.13	3.33	10.41	49

Quake	Clearance			Time
	min	**avg**	**max**	**s**
RRM*	0.00	2.90	9.45	
RRRM	0.05	3.28	9.75	343

Table 7.3 Clearance and time statistics for RRM* and retracted RRM* (RRRM) roadmaps. Statistics corresponding to the RRRM are the averages over 100 independent runs. In the Field and Office environment, the RRRM used the $\mathcal{W}$-RETRACTION algorithm to the retract the roadmap. The $\mathcal{C}$-RETRACTION algorithm was used in the House and Quake environments.

vironments), connecting a query by using clearance information turned out to be efficient, i.e. the times were below 1 ms. Since running Dijkstra's shortest path algorithm was far below 1 ms for the roadmaps produced by the RRM and RRM*, we conclude that this technique enables real-time extraction of queries.

An advantage of using clearance information is that no expensive collision-checking operations are needed for motion planning. In particular, no geometrical data has to be stored for motion planning purposes which saves memory. As we created small roadmaps, storing clearance information is feasible.

Adding clearance

As indicated above, high-clearance paths are often preferred. Such paths can be obtained by retracting each of the four roadmaps created by the RRM* to the medial axis. We refer to this method as the Retracted RRM* (RRRM). We compare the clearance of roadmaps produced by the RRM* and RRRM. Table 7.3 shows the corresponding statistics. In all retracted roadmaps, the (minimum, maximum and average) clearance was improved. The times needed for the retraction were reasonably fast for the Field and House environments. The roadmaps for Office and Quake environments were retracted within six minutes. The main reason for this difference is that the latter two roadmaps are considerably larger than the other two. As we have already indicated in Section 4.5.2, we expect that the running times of the retraction algorithms can be dramatically decreased by incorporating learning techniques.

7.5 Discussion

A common way to plan a path is to use an A* algorithm on a grid. The grid may have many cells because an environment can be large or the environment contains a narrow passage by which a high-resolution grid is needed. We showed that for such grids, the query times are far too high for real-time motion planning.

We presented a method that automatically computes a roadmap for 2D and 3D virtual environments. We used the Reachability Roadmap Method to generate small roadmaps. This ensures low query times and low memory consumption. In addition, the roadmaps are resolution complete which means that a valid query can always be extracted from the roadmap.

As paths, extracted from this roadmap, can make long detours around obstacles, we provided a method that adds useful cycles to the roadmap. Experiments showed that reasonably short paths can now be extracted from the enhanced roadmap. This roadmap also provides alternative routes which allow for variation in the routes that entities take. The query times on this roadmap are low which enables real-time extraction of paths.

Another criterion the roadmap should satisfy is that high-clearance paths can be extracted without extra computation time in the query phase. This criterion was met by retracting the edges of the roadmap to the medial axis of the free space. Experiments showed that this criterion can be satisfied within six minutes in our test environments. However, we believe that the running times of the retraction algorithms can be improved dramatically by incorporating learning techniques. In addition, since closest pair calculations are expensive, much could be saved by using the discretized free space instead of referring these calculations to the collision checker.

In future work, the method could be extended to incorporate extra constraints that level designers put on the roadmaps. For example, one could incorporate non-holonomic constraints, walkable surfaces, take special care of staircases and incorporate tactical information in the roadmap.

CHAPTER
EIGHT

Conclusion

In this thesis, we studied a central problem in robotics: planning a collision-free path for a moving robot in a static and known environment. We restricted ourselves to motion planning for rigid bodies and articulated robots. We compared and analyzed multiple-shot sampling-based motion planning techniques, in particular variants of the Probabilistic Roadmap Method (PRM).

Sampling-based planners can successfully handle a large diversity of problems. The success of these planners in solving problems with possibly many degrees of freedom and many obstacles can be explained by the fact that no explicit representation of the free configuration space is required. The main operation of these planners is checking placements of the robot for collisions with obstacles in the environment, which can be efficiently performed by the current generation of collision checkers. In contrast, exact methods always have to take each obstacle into account, even when the solution path is simple. The second reason for their success is that problems which are not pathological have favorable reachability properties. That is, the free configuration space of a reasonable problem can often be captured by few nodes where each node can reach a large portion of the free space using a local planner. Therefore, a PRM usually finds a solution quickly, even if the geometric complexity is high.

If the length of the allowed connections between nodes in the roadmap is considerably limited, the PRM starts looking like grid-based techniques in which nodes are only connected to their adjacent neighbors in the grid. Experiments showed that this had a dramatic negative impact on the running times. The PRM also tends to perform poorly when crucial configurations lie in and around very narrow regions of the configuration space, which has been identified as the narrow passage problem. The probability of randomly guessing such a configuration can be very small, especially when the rest of the free space is large compared to these regions. Moreover, creating a set of configurations that covers a path going through the passage is not necessarily sufficient to solve the problem. The problem is only solved when all configurations in the set belong to the same connected component. Experiments showed that this last criterion, which we call maximal connectivity, is much more difficult to satisfy than the coverage criterion, especially when we have to deal with a narrow passage.

The narrow passage problem can be tackled by incorporating a hybrid or adaptive sampling strategy that concentrates samples in difficult areas on the one hand, and generates some samples in large open areas on the other hand. Hence, using a uniform sampling strategy is not a good choice for environments involving narrow passages. Another tactic is to employ a more powerful local planner. We presented a potential field local planner that creates larger reachability regions which eases making connections. This planner is also better able to find the entry of a narrow passage, decreasing the number of samples needed to obey the maximal connectivity criterion.

When a passage is extremely narrow, the PRM may not always find a solution within the allowed amount of time as this method is (only) probabilistically complete. In this case, or when the robot needs to maintain contact with the obstacles, using a PRM is a poor choice. In such cases, a problem may only be solved in practice by a careful analysis before it is fed to a complete planner.

In conclusion, when we have to deal with a problem with many degrees of freedom (DOFs) and the passages are not pathologically narrow, a PRM seems to be a logical choice as sampling-based planners can successfully handle the curse of dimensionality. These planners have been used successfully for many applications, including CAD/CAM, computer simulation, computer animation, biology, medical applications, and virtual environments. Our analysis and experiments have provided further insight in the effects of the different choices which should be of use in improving these applications.

In most applications, a path should preferably be short because redundant motions will take longer to execute. Although reasonably short paths can be obtained from a roadmap with cycles, yet even shorter paths can be obtained by creating Partial shortcuts. Experiments showed that this new technique is successfully able to remove redundant (rotational) motions of the robot, improving on existing algorithms. Besides having a short path, the robot often has to keep some minimum amount of clearance to the obstacles because it can be difficult to measure and control the precise position of a robot. Traveling along a path with a certain amount of minimum clearance reduces the chances of collisions due to these uncertainties. We proposed a new technique that increases the clearance along a path without using complex data structures and algorithms. Both techniques can be applied to a wide range of robots which may reside in high-dimensional configuration spaces.

Interactive applications are getting more and more attention from robotics. Research includes planning the motions for a group of entities, the generation of camera motion to track a moving guide and the animation of the entities. The entity that moves in the environment is often represented by a translating cylinder or box. As such an entity only has two or three DOFs, we can often employ a more efficient and more specific algorithm than the PRM. We introduced the Reachability Roadmap Method which can be used efficiently to create small roadmaps for these environments. Such a small roadmap ensures low query times and low memory consumption and is easy to adjust to fit the user's wishes. If there exists a path in the (discretized) free space that connects the start and goal of the query, the algorithm ensures that a path can be found in the roadmap. Hence, the algorithm is resolution complete. Another criterion these roadmaps have to satisfy is that alternative routes and short paths can be extracted. This was met by adding useful nodes and useful cycles. Besides the possibility to extract short paths, paths are often required to have much clearance, as this leads to natural looking motions. For example, high-clearance paths work well with entities that have large width, such as a wide formation of characters. We met this goal by retracting a roadmap to the medial axis which allowed extraction of such paths in real-time.

While the paths we computed may be perfectly suitable for robots operating in virtual environments, they cannot be used directly for controlling real robots due to inexact control and dynamic constraints. The first problem is caused by robots not following a precomputed path exactly. Sensor information may be incorporated into the motion planning algorithm to adjust the robot's actions and to reduce the errors in its position. The second problem exists because we do not take into account constraints such as velocities and acceleration. This problem can be handled by adding extra information in the nodes of the roadmap which can be used by a suitable local planner. Another approach would be transforming the geometric path to a path that can be executed by searching within the neighborhood of the geometric path for a real solution [140].

The PRM was first described some years ago. Originally, there was much doubt about its usefulness, but soon, people realized its power in many different applications. We have come a long way since then. By a combination of faster computers and improvements in the technique, we are now able to solve complex motion planning problems efficiently.

A next step is to create a library of motion planning techniques that can be used as a 'black-box'. That is, users should not have to think about parameter choices that usually have no meaning to them. In this thesis, we tried to automate these choices as much as possible. We also provided insight in the effect these parameters have on the PRM. While our SAMPLE system, algorithms and results provide a foundation for such a library, more research will be needed to enable the black-box to automatically choose parameters, such as an appropriate metric, a step size of the local planner, the maximum connection distance, a hybrid sampling strategy, and a termination criterion.

Creating such a library may prove essential to the development of autonomous robots.

BIBLIOGRAPHY

[1] G. Alt. The suffering: A game AI case study. In *Challenges in Game AI workshop, Nineteenth national conference on Artificial Intelligence*, pages 134–138, 2004.

[2] N.M. Amato, O. Bayazit, L. Dagle, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *IEEE International Conference on Robotics and Automation*, pages 630–637, 1998.

[3] N.M. Amato, O. Bayazit, L. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 155–168, 1998.

[4] N.M. Amato and Y. Wu. A randomized roadmap method for path and manipulation planning. In *IEEE International Conference on Robotics and Automation*, pages 113–120, 1996.

[5] B. Baginski. Efficient motion planning in high dimensional spaces: The parallelized Z^3-method. In *International Workshop on Robotics in the Alpe-Adria-Danube Region*, pages 247–252, 1997.

[6] J. Barraquand, L.E. Kavraki, J.-C. Latombe, T.-Y. Li, R. Motwani, and P. Raghavan. A random sampling scheme for path planning. *International Journal of Robotics Research*, 16:759–744, 1997.

[7] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22:224–241, 1992.

[8] J. Barraquand and J.-C. Latombe. A Monte-Carlo algorithm for path planning with many degrees of freedom. In *IEEE International Conference on Robotics and Automation*, pages 1712–1717, 1990.

[9] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *International Journal of Robotics Research*, 10:628–649, 1991.

[10] O.B. Bayazit, J.-M. Lien, and N.M. Amato. Better group behaviors using rule-based roadmaps. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 95–111, 2002.

[11] O.B. Bayazit, G. Song, and N.M. Amato. Ligand binding with OBPRM and haptic user input. In *IEEE International Conference on Robotics and Automation*, pages 954–959, 2001.

[12] M. Ben-Or, D. Kozen, and J. Reif. The complexity of elementary algebra and geometry. *Computer and Systems Sciences*, 32:251–264, 1986.

[13] M. Bennewitz, W. Burgard, and S. Thrun. Priority schemes for decoupled path planning techniques for teams of mobile robots. *Robotics and Autonomous System*, 41:89–99, 2002.

[14] J.P. van den Berg, D. Nieuwenhuisen, L. Jaillet, and M.H. Overmars. Creating robust roadmaps for motion planning in changing environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2515–2521, 2005.

[15] J.P. van den Berg and M.H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, pages 453–460, 2004.

[16] J.P. van den Berg and M.H. Overmars. Prioritized motion planning for multiple robots. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2217–2222, 2005.

[17] J.P. van den Berg and M.H. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics and Automation*, 21:885–897, 2005.

[18] G. van den Bergen. *Collision Detection in Interactive 3D Environments*. Morgan Kaufmann, 2003.

[19] P. Bessiere, J.M.Ahuactzin, E.-G. Talbi, and E. Mazer. The 'ariadne's clew' algorithm: Global planning with local methods. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1373–1380, 1993.

[20] R. Bohlin. *Motion Planning for Industrial Robots*. PhD thesis, Göteborg University, 1999.

[21] R. Bohlin and L.E. Kavraki. Path planning using lazy PRM. In *IEEE International Conference on Robotics and Automation*, pages 521–528, 2000.

[22] V. Boor, M.H. Overmars, and A.F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, pages 1018–1023, 1999.

[23] M. Branicky, S.M. LaValle, K. Olson, and L. Yang. Quasi-randomized path planning. In *IEEE International Conference on Robotics and Automation*, pages 1481–1487, 2001.

[24] J.E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4:25–30, 1965.

[25] O. Brock and L.E. Kavraki. Decomposition-based motion planning: A framework for real-time motion planning in high-dimensional configuration spaces. In *IEEE International Conference on Robotics and Automation*, pages 1469–1475, 2001.

[26] O. Brock and O. Khatib. Elastic strips: A framework for motion generation in human environments. *International Journal of Robotics Research*, 21:1031–1052, 2002.

[27] R.A. Brooks and T. Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, 15:224–233, 1985.

[28] J. Canny. *The Complexity of Robot Motion Planning*. MIT Press, 1988.

[29] H. Chang and T.Y. Li. Assembly maintainability study with motion planning. In *IEEE International Conference on Robotics and Automation*, pages 1012–1019, 1995.

[30] B. Chazelle. *The discrepancy method*. Cambridge University Press, Cambridge, 2000.

[31] P.C. Chen and Y.K. Hwang. SANDROS: A dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation*, 14:390–403, 1998.

[32] P. Cheng, E. Frazzoli, and S.M. LaValle. Exploiting group symmetries to improve precision in kinodynamic and nonholonomic planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 631–636, 2003.

[33] P. Cheng, Z. Shen, and S.M. LaValle. Using randomization to find and optimize feasible trajectory for nonlinear systems. In *Annual Allerton Conference on Communications, Control, Computing*, pages 926–935, 2000.

[34] J. Chestnutt, M. Lau, G. Cheung, J.J. Kuffner, J.K. Hodgins, and T. Kanade. Footstep planning for the honda asimo humanoid. In *IEEE International Conference on Robotics and Automation*, pages 1909–1915, 2005.

[35] H. Choset and J. Burdick. Sensor-based exploration: The hierarchical generalized Voronoi graph. *International Journal of Robotics Research*, 19:96–125, 2000.

[36] H. Choset, K.M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L.E. Kavraki, and S. Thrun. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, first edition, 2005.

[37] C.M. Clark, T. Bretl, and S. Rock. Applying kinodynamic randomized motion planning with a dynamic priority system to multi-robot space systems. In *IEEE Aerospace Conference*, pages 3621–3631, 2002.

[38] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press/ McGraw-Hill Book Company, second edition, 2001.

[39] J. Cortés and T Siméon. Sampling-based motion planning under kinematic loop-closure constraints. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 59–74, 2004.

[40] J. Cortés, T. Siméon, and J.-P. Laumond. A random loop generator for planning the motions of closed kinematic chains using PRM methods. In *IEEE International Conference on Robotics and Automation*, pages 2141–2146, 2002.

[41] L. Dale. *Optimization techniques for probabilistic roadmaps*. PhD thesis, Texas A&M University, 2000.

[42] L. Dale and N.M. Amato. Probabilistic roadmaps – putting it all together. In *IEEE International Conference on Robotics and Automation*, pages 1940–1947, 2001.

[43] M. DeLoura. *Game programming gems*. Charles River Media, Inc, 2000.

[44] M. Erdmann and T. Lozano-Pérez. On multiple moving objects. *Algorithmica*, 2:477–521, 1987.

[45] C. Esteves, G. Arechavaleta, and J.-P. Laumond. Motion planning for human-robot interaction in manipulation tasks. In *IEEE International Conference on Mechatronics and Automation*, pages 1766–1771, 2005.

[46] B. Faverjon. Object level programming of industrial robots. In *IEEE International Conference on Robotics and Automation*, pages 1406–1412, 1986.

[47] M. Foskey, M. Garber, M. Lin, and D. Manocha. A Voronoi-based hybrid motion planner. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 55–60, 2001.

[48] Th. Fraichard. Trajectory planning in a dynamic workspace: A 'state-time' approach. *Advanced Robotics*, 13:75–94, 1999.

[49] R. Gayle, P. Segars, M.C. Lin, and D. Manocha. Path planning for deformable robots in complex environments. In *Robotics: Science and Systems*, 2005.

[50] C. Geem, T. Siméon, J.-P. Laumond, J.-L. Bouchet, and J.-F. Rit. Mobility analysis for feasibility studies in CAD models of industrial environments. In *IEEE International Conference on Robotics and Automation*, pages 1770–1775, 1999.

[51] R. Geraerts and M.H. Overmars. A comparative study of probabilistic roadmap planners. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 43–57, 2002.

[52] R. Geraerts and M.H. Overmars. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2386–2392, 2004.

[53] R. Geraerts and M.H. Overmars. On improving the path quality for motion planning. In *Conference of the Advanced School for Computing and Imaging*, pages 211–217, 2004.

[54] R. Geraerts and M.H. Overmars. Sampling techniques for probabilistic roadmap planners. In *Conference on Intelligent Autonomous Systems*, pages 600–609, 2004.

[55] R. Geraerts and M.H. Overmars. Creating small roadmaps for solving motion planning problems. In *IEEE International Conference on Methods and Models in Automation and Robotics*, pages 531–536, 2005.

[56] R. Geraerts and M.H. Overmars. On improving the clearance for robots in high-dimensional configuration spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4074–4079, 2005.

[57] R. Geraerts and M.H. Overmars. On the analysis and success of sampling based motion planning. In *Conference of the Advanced School for Computing and Imaging*, pages 313–319, 2005.

[58] R. Geraerts and M.H. Overmars. Reachability analysis of sampling based planners. In *IEEE International Conference on Robotics and Automation*, pages 406–412, 2005.

[59] R. Geraerts and M.H. Overmars. Creating high-quality roadmaps for motion planning in virtual environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4355–4361, 2006.

[60] R. Geraerts and M.H. Overmars. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous System*, 54:165–173, 2006.

[61] R. Ghrist, J.M. O'Kane, and S.M. LaValle. Pareto optimal coordination on roadmaps. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 171–186, 2004.

[62] O. Goemans and M.H. Overmars. Automatic generation of camera motion to track a moving guide. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 187–202, 2004.

[63] K. Gupta. Motion planning for flexible shapes (systems with many degrees of freedom): A survey. *The Visual Computer*, 14:288–302, 1998.

[64] D. Halperin, J.-C. Latombe, and R.H. Wilson. A general framework for assembly planning: The motion space approach. *Algorithmica*, 26:577–601, 2000.

[65] D. Halperin, M.H. Overmars, and M. Sharir. Efficient motion planning for an L-shaped object. *SIAM Journal on Computing*, 21:1–23, 1992.

[66] L. Han and N.M. Amato. *Algorithmic and Computational Robotics: New Directions. The Fourth Workshop on the Algorithmic Foundations of Robotics*, chapter A kinematics-based probabilistic roadmap method for closed chain systems, pages 233–245. 2000.

[67] K. Hoff, T. Culver, J. Keyser, M. Lin, and D. Manocha. Interactive motion planning using hardware-accelerated computation of generalized Voronoi diagrams. In *IEEE International Conference on Robotics and Automation*, pages 2931–2937, 2000.

[68] C. Holleman, L.E. Kavraki, and J. Warren. Planning paths for a flexible surface patch. In *IEEE International Conference on Robotics and Automation*, pages 21–26, 1998.

[69] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *IEEE International Conference on Robotics and Automation*, pages 4420–4426, 2003.

[70] D. Hsu, L.E. Kavraki, J.-C. Latombe, R. Motwani, and S. Sorkin. *Robotics: The Algorithmic Perspective. The Third Workshop on the Algorithmic Foundations of Robotics*, chapter On Finding Narrow Passages with Probabilistic Roadmap Planners, pages 141–154. 1998.

[71] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock. Randomized kinodynamic motion planning with moving obstacles. *International Journal of Robotics Research*, 21:233–255, 2002.

[72] D. Hsu, J.-C. Latombe, and S. Sorkin. Placing a robot manipulator amid obstacles for optimized execution. In *IEEE International Symposium on Assembly and Task*, pages 280–285, 1999.

[73] D. Hsu, G. Sánchez-Ante, and Z. Sun. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *IEEE International Conference on Robotics and Automation*, pages 3885–3891, 2005.

[74] F.K. Hwang, D.S. Richards, and P. Winter. *The Steiner Tree Problem*. North-Holland, 1992.

[75] P. Isto. Constructing probabilistic roadmaps with powerful local planning and path optimization. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2323–2328, 2002.

[76] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann. Planning collision-free reaching motions for interactive object manipulation and grasping. In *Eurographics*, pages 313–322, 2003.

[77] A. Kamphuis and M.H. Overmars. Finding paths for coherent groups using clearance. In *Eurographics/ ACM SIGGRAPH Symposium on Computer Animation*, pages 19–28, 2004.

[78] A. Kamphuis and M.H. Overmars. Motion planning for coherent groups of entities. In *IEEE International Conference on Robotics and Automation*, pages 3815–3822, 2004.

[79] A. Kamphuis, J. Pettre, M.H. Overmars, and J.-P. Laumond. Path finding for the animation of walking characters. In *Poster proceedings of Eurographics/ACM SIGGRAPH Symposium on Computer Animation*, pages 8–9, 2005.

[80] L.E. Kavraki. *Random networks in configuration space for fast path planning*. PhD thesis, Stanford University, 1995.

[81] L.E. Kavraki, M. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *IEEE International Conference Robotics and Automation*, pages 3020–3025, 1996.

[82] L.E. Kavraki, F. Lamiraux, and C. Holleman. A general framework for planning paths for elastic objects. In *International Workshop on the Algorithmic Foundations of Robotics*, pages 313–325, 1998.

[83] L.E. Kavraki and J.-C. Latombe. Randomized preprocessing of configuration space for fast path planning. In *IEEE International Conference on Robotics and Automation*, pages 2138–2145, 1994.

[84] L.E. Kavraki and J.-C. Latombe. Probabilistic roadmaps for robot path planning. In K. Gupta and A. del Pobil, editors, *Practical Motion Planning in Robotics: Current Approaches and Future Directions*, pages 33–53. John Wiley, 1998.

[85] L.E. Kavraki, P. Švestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12:566–580, 1996.

[86] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, 5:90–98, 1986.

[87] P. Khosla and R. Volpe. Superquadratic artificial potentials for obstacle avoidance and approach. In *IEEE International Conference on Robotics and Automation*, pages 1778–1784, 1988.

[88] J. Kim, R. Pearce, and N.M. Amato. Extracting optimal paths from roadmaps for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2424–2429, 2003.

[89] D.E. Koditschek. Exact robot navigation by means of potential functions: Some topological considerations. In *IEEE International Conference on Robotics and Automation*, pages 1–6, 1987.

[90] Y. Koga, K. Kondo, J.J. Kuffner, and J.-C. Latombe. Planning motions with intentions. In *ACM Special Interest Group on Computer Graphics (SIGGRAPH)*, pages 395–408, 1995.

[91] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *IEEE International Conference on Robotics and Automation*, pages 1398–1404, 1991.

[92] J.J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *IEEE International Conference on Robotics and Automation*, pages 3993–3998, 2004.

[93] J.J. Kuffner and J.-C. Latombe. Interactive manipulation planning for animated characters. In *IEEE International Conference on Robotics and Automation*, pages 417–418, 2000.

[94] J.J. Kuffner and S.M. LaValle. RRT-connect: An efficient approach to single-query path planning. In *IEEE International Conference on Robotics and Automation*, pages 995–1001, 2000.

[95] J.J. Kuffner, K. Nishiwaki, S. Kagami, M. Inaba, and H. Inoue. Motion planning for humanoid robots under obstacle and dynamic balance constraints. In *IEEE International Conference on Robotics and Automation*, pages 692–698, 2001.

[96] F. Lamiraux, D. Bonnafous, and C. V. Geem. Path optimization for nonholonomic systems: Application to reactive obstacle avoidance and path planning. In *Workshop Control Problems in Robotics and Automation*, pages 1–18, 2002.

[97] F. Lamiraux and L.E. Kavraki. Planning paths for elastic objects under manipulation constraints. *International Journal of Robotics Research*, 20:188–208, 2001.

[98] F. Lamiraux and J.-P. Laumond. Smooth motion planning for car-like vehicles. *IEEE Transactions on Robotics and Automation*, 17:188–208, 2001.

[99] J.-C. Latombe. *Robot Motion Planning*. Kluwer, 1991.

[100] S.M. LaValle. *Planning Algorithms*. http://msl.cs.uiuc.edu/planning, 2005.

[101] S.M. LaValle and S.A. Hutchinson. Optimal motion planning for multiple robots having independent goals. *Transaction on Robotics and Automation*, 14:912–925, 1998.

[102] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20:378–400, 2001.

[103] A. Lee, I. Streinu, and O. Brock. A methodology for efficiently sampling the conformation space of molecular structures. *Physical Biology*, 2:108–115, 2005.

[104] Y.-H. Lee, T.-W. Kao, and S.-S. Lee. Optimal parallel algorithms for computing the chessboard distance transform and the medial axis transform on RAP. In *IEEE International Symposium on Parallel Architectures, Algorithms and Networks*, pages 22–28, 1996.

[105] D. Leven and M. Sharir. Planning a purely translational motion for a convex object in two-dimensional space using generalized Voronoi diagrams. *Discrete Computational Geometry*, 2:9–31, 1987.

[106] T.-Y. Li and H.-C. Chou. Motion planning for a crowd of robots. In *IEEE International Conference on Robotics and Automation*, pages 4215–4221, 2003.

[107] J.-M. Lien, S. Thomas, and N.M. Amato. A general framework for sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation*, pages 4439–4444, 2003.

[108] S.R. Lindemann and S.M. LaValle. Smoothly blending vector fields for global robot navigation. In *submitted to IEEE Conference on Decision and Control*, 2005.

[109] G.F. Liu and J.C. Trinkle. Complete path planning for planar closed chains among point obstacles. In *Robotics: Science and Systems*, 2005.

[110] J. Lo, G. Huang, and D. Metaxas. Human motion planning based on recursive dynamics and optimal control techniques. *Multibody System Dynamics*, 8:433–58, 2002.

[111] E. Masehianand, M.R. Admin-Naseri, and S.E. Khadem. Online motion planning using incremental construction of medial axis. In *IEEE International Conference on Robotics and Automation*, pages 2928–2933, 2003.

[112] M.T. Mason. *Mechanics of Robotic Manipulation*. MIT Press, 2001.

[113] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudorandom number generator. *Transactions on Modeling and Computer Simulation*, 8:3–30, 1998.

[114] D. Nain, S. Haker, R. Kikinis, and E. Grimson. An interactive virtual endoscopy tool. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 55–60, 2001.

[115] C.L. Nielsen and L.E. Kavraki. A two level fuzzy prm for manipulation planning. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1716–1721, 2000.

[116] D. Nieuwenhuisen. Callisto. http://www.cs.uu.nl/∼dennis/callisto/callisto.html, 2006.

[117] D. Nieuwenhuisen, A. Kamphuis, M. Mooijekind, and M.H. Overmars. Automatic construction of roadmaps for path planning in games. In *International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 285–292, 2004.

[118] D. Nieuwenhuisen and M.H. Overmars. Motion planning for camera movements. Technical Report 2003-004, Utrecht University, 2003.

[119] D. Nieuwenhuisen and M.H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE International Conference on Robotics and Automation*, pages 446–452, 2004.

[120] N.J. Nilsson. A mobile automation: An application of artificial intelligence techniques. In *International Joint Conference on Artificial Intelligence*, pages 509–520, 1969.

[121] C. Nissoux, T. Siméon, and J.-P. Laumond. Visibility based probabilistic roadmaps. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1316–1321, 1999.

[122] C. Ó'Dúnlaing, M. Sharir, and C.K. Yap. Retraction: A new approach to motion planning. In *ACM Symposium on Theory of Computing*, pages 207–220, 1983.

[123] J. O'Rourke. *Art Gallery Theorems and Algorithms*. New York: Oxford University Press, 1987.

[124] M.H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-32, Utrecht University, 1992.

[125] W. Park, D.B. Chaffin, and B.J. Martin. Toward memory-based human motion simulation: Development and validation of a motion modification algorithm. *IEEE Transactions on Systems, Man, and Cybernetics*, 34:376–386, 2004.

[126] J. Peng and S. Akella. Coordinating multiple robots with kinodynamic constraints along specified paths. *International Journal of Robotics Research*, 24:295–310, 2005.

[127] J.H. Reif. Complexity of the mover's problem and generalizations. In *IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[128] E. Rimon and D.E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8:501–518, 1992.

[129] B. Salomon, M. Garber, M.C. Lin, and D. Manocha. Interactive navigation in complex environments using path planning. In *Symposium on Interactive 3D graphics*, pages 41–50, 2003.

[130] G. Sánchez and J.-C. Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *International Symposium of Robotics Research*, pages 403–418, 2001.

[131] G. Sánchez and J.-C. Latombe. On delaying collision checking in PRM planning – Application to multi-robot coordination. *International Journal of Robotics Research*, 21:5–26, 2002.

[132] G. Sánchez and J.-C. Latombe. Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In *IEEE International Conference Robotics and Automation*, 2002.

[133] G. Sánchez and J.-C. Latombe. Using a PRM planner to compare centralized and decoupled planning for multi-robot systems. In *IEEE International Conference on Robotics and Automation*, pages 2112–2119, 2002.

[134] J. Savage, E. Marquez, J. Pettersson, N. Trygg, A. Petersson, and M. Wahde. Optimization of waypoint-guided potential field navigation using evolutionary algorithms. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3463–3468, 2004.

[135] E. Schmitzberger, J.-L. Bouchet, M. Dufaut, W. Didier, and R. Husson. Capture of homotopy classes with probabilistic road map. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2002.

[136] J.T. Schwartz and M. Sharir. On the piano movers' problem: I. The case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on Pure and Applied Mathematics*, 36:345–398, 1983.

[137] J.T. Schwartz and M. Sharir. On the piano movers' problem: II. General techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4:298–351, 1983.

[138] J.T. Schwartz and M. Sharir. On the piano movers' problem: III. Coordinating the motion of several independent bodies: The special case of circular bodies moving amidst polygonal obstacles. *International Journal of Robotics Research*, 2:46–75, 1983.

[139] J.T. Schwartz and M. Sharir. On the piano movers' problem: V. The case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Communications on Pure and Applied Mathematics*, 37:815–848, 1984.

[140] S. Sekhavat, P. Švestka, J.-P. Laumond, and M.H. Overmars. Multilevel path planning for nonholonomic robots using semiholonomic subsystems. *International Journal of Robotics Research*, 17:840–857, 1998.

[141] S. Shimoda, Y. Kuroda, and K. Iagnemma. Random motion to escape local minima potential field navigation of high speed unmanned ground vehicles on uneven terrain. In *IEEE International Conference on Robotics and Automation*, pages 2839–2844, 2005.

[142] K. Shoemake. *Graphics Gems III*, chapter Uniform random rotations, pages 124–132. Academic Press, 1992.

[143] T. Siméon, R. Chatila, and J.-P. Laumond. Computer aided motion for logistics in nuclear plants. In *International symposium on artificial intelligence, robotics and human centered technology for nuclear applications*, pages 46–53, 2002.

[144] T. Siméon, J. Cortés, A. Sahbani, and J.-P. Laumond. A manipulation planner for pick and place operations under continuous grasps and placements. In *IEEE International Conference on Robotics and Automation*, pages 2022–2027, 2002.

[145] T. Siméon, S. Leroy, and J.-P. Laumond. Path coordination for multiple mobile robots: A resolution complete algorithm. *IEEE Transactions on Robotics and Automation*, 18:42–49, 2002.

[146] A.P. Singh, J.-C. Latombe, and D.L. Brutlag. A motion planning approach to flexible ligand binding. In *International Conference on Intelligent Systems for Molecular Biology*, pages 252–261, 1999.

[147] A.F. van der Stappen, M.H. Overmars, M. de Berg, and J. Vleugels. Motion planning in environments with low obstacle density. *Discrete & Computational Geometry*, 20:561–587, 1998.

[148] S. Thomas, G. Song, and N.M Amato. Protein folding by motion planning. *Physical biology*, 2:148–155, 2005.

[149] G. Varadhan and D. Manocha. Star-shaped roadmaps – A deterministic sampling approach for complete motion planning. In *Robotics: Science and Systems*, 2005.

[150] J. Vleugels and M.H. Overmars. Approximating Voronoi diagrams of convex sites in any dimension. *International Journal of Computational Geometry & Applications*, 8:201–221, 1998.

[151] P. Švestka. *Robot Motion Planning Using Probabilistic Road Maps*. PhD thesis, Utrecht University, 1997.

[152] P. Švestka and M.H. Overmars. Motion planning for car-like robots, a probabilistic learning approach. *International Journal of Robotics Research*, 16:119–143, 1997.

[153] P. Švestka and M.H. Overmars. Coordinated path planning for multiple robots. *Robotics and Autonomous System*, 23:125–152, 1998.

[154] X. Wang and F.J. Hickernell. Randomized Halton sequences. *Mathematics and Computer Modeling*, 32:887–899, 2000.

[155] R. Wein, J.P. van den Berg, and D. Halperin. The Visibility-Voronoi complex and its applications. In *Annual Symposium on Computational Geometry*, pages 63–72, 2005.

[156] S.A. Wilmarth, N.M. Amato, and P.F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *IEEE International Conference on Robotics and Automation*, pages 1024–1031, 1999.

[157] J. Yakey, S.M. LaValle, and L.E. Kavraki. Randomized path planning for linkages with closed kinematic chains. *IEEE Transactions on Robotics and Automation*, 17:951–958, 2001.

[158] A. Yershova and S.M. Lavalle. Efficient nearest neighbor searching for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 632–637, 2002.

[159] A. Yershova and S.M. Lavalle. Deterministic sampling methods for spheres and $SO(3)$. In *IEEE International Conference on Robotics and Automation*, pages 3974–3980, 2004.

[160] D. Zhu and J.-C. Latombe. New heuristic algorithms for efficient hierarchical path planning. *IEEE Transactions on Robotics and Automation*, 7:9–20, 1991.

Publications

The chapters of this thesis are based on the following papers:

Chapter 2

R. Geraerts and M.H. Overmars. A comparative study of probabilistic roadmap planners. In *Workshop on the Algorithmic Foundations of Robotics*, pages 43–57, 2002.

R. Geraerts and M.H. Overmars. Sampling techniques for probabilistic roadmap planners. In *Conference on Intelligent Autonomous Systems*, pages 600–609, 2004.

R. Geraerts and M.H. Overmars. Sampling and node adding in probabilistic roadmap planners. *Journal of Robotics and Autonomous Systems*, 54:165–173, 2006.

Chapter 3

R. Geraerts and M.H. Overmars. Reachability analysis of sampling based planners. In *IEEE International Conference on Robotics and Automation*, pages 406–412, 2005.

R. Geraerts and M.H. Overmars. On the analysis and success of sampling based motion planning. In *Conference of the Advanced School for Computing and Imaging*, pages 313–319, 2005.

Chapter 4

R. Geraerts and M.H. Overmars. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2386–2392, 2004.

R. Geraerts and M.H. Overmars. On improving the path quality for motion planning. In *Conference of the Advanced School for Computing and Imaging*, pages 211–217, 2004.

R. Geraerts and M.H. Overmars. On improving the clearance for robots in high-dimensional configuration spaces. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4074–4079, 2005.

Chapter 5

R. Geraerts and M.H. Overmars. Clearance based path optimization for motion planning. In *IEEE International Conference on Robotics and Automation*, pages 2386–2392, 2004.

R. Geraerts and M.H. Overmars. On improving the path quality for motion planning. In *Conference of the Advanced School for Computing and Imaging*, pages 211–217, 2004.

Chapter 6

R. Geraerts and M.H. Overmars. Creating small roadmaps for solving motion planning problems. In *IEEE International Conference on Methods and Models in Automation and Robotics*, pages 531–536, 2005.

Chapter 7

R. Geraerts and M.H. Overmars. Creating high-quality roadmaps for motion planning in virtual environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4355–4361, 2006.

SAMENVATTING

In de toekomst —misschien al over 20 jaar— zullen robots niet meer uit ons dagelijkse bestaan weg te denken zijn. Zo zullen ze tal van huishoudelijke taken, overheidsdiensten en industriële operaties gaan uitvoeren. Ook kunnen ze gevaarlijke taken tijdens reddingsoperaties uit handen nemen.

De robots van nu zijn nog lang niet in staat om dit soort complexe taken autonoom uit te voeren. Veel taken lijken eenvoudig voor mensen, zoals het serveren van koffie, omdat we in ons leven intensief getraind zijn om interactie met de omgeving te hebben. Echter, robots moeten nog vele uitdagingen overwinnen: ze moeten een taak begrijpen en in deeltaken opdelen, de omgeving waarin ze werkzaam zijn interpreteren en in kaart brengen, de obstakels ontwijken terwijl ze navigeren, objecten verplaatsen en manipuleren, en ze dienen sociale interacties met hun omgeving te kunnen hebben.

Hedendaagse robots voeren taken uit die te monotoon, smerig of gevaarlijk voor ons zijn. Denk maar eens aan het verwijderen van giftig afval of het uitvoeren van onderhoudswerkzaamheden in de ruimte. Ook in de industrie zijn robots met succes ingezet voor het doen van repeterende en gevaarlijke taken. Naast de industrie heeft ook de consument de robot ontdekt. Erg populair is de speelgoed-robothond, *Aibo*, die door Sony op de markt is gebracht. Hij is in staat om zich voort te bewegen, de omgeving te 'zien' en op gesproken commando's te reageren. Andere veelverkochte huishoudelijke robots zijn stofzuigers en grasmaaiers die op eigen houtje hun taak volbrengen.

Eén van de fundamentele taken voor robots is het plannen van hun bewegingen terwijl ze botsingen met obstakels in hun omgeving voorkomen. Deze taak vormt het centrale thema van dit proefschrift. Hierin wordt met name bewegingsplanning voor rigide en aaneengeschakelde robots in statische en tevens bekende virtuele omgevingen besproken.

Het proefschrift is opgesplitst in twee delen. In het eerste deel worden sampling-gebaseerde bewegingsplanningtechnieken vergeleken en geanalyseerd. Er zal dieper worden ingegaan op de *probabilistische wegenkaartmethode* ('Probabilistic Roadmap Method'). Het doel van de methode is het maken van een wegenkaart van de omgeving die de robot kan gebruiken om een pad van A naar B te plannen. Omdat deze omgeving bekend wordt geacht, kan voor elke mogelijke positie van de robot getest worden of hij al dan niet een obstakel raakt. De specificatie van een dergelijke positie wordt ook wel een *sample* genoemd. De methode genereert een willekeurige verzameling van botsingsvrije samples. Zij vormen de knooppunten van de wegenkaart. Voor bepaalde zorgvuldig gekozen paren uit deze verzameling probeert de methode de twee betreffende samples met elkaar te verbinden met een lokaal pad. Zo'n pad is meestal een eenvoudige rechtlijnige verbinding tussen de samples. Als het pad botsingsvrij is, dan wordt het als een verbinding in de wegenkaart opgenomen. Een pad tussen een bepaalde start- en eindpositie kan de robot vervolgens in twee stappen vinden: eerst moeten deze posities als samples toegevoegd worden aan en verbonden worden met de wegenkaart. Dan kan de robot simpelweg de wegenkaart gebruiken om tussen deze posities te navigeren.

De methode is succesvol toegepast in een grote verscheidenheid aan applicaties waaronder CAD/CAD applicaties, computersimulatie, computeranimatie, biologie, en medische en interactieve applicaties. Het succes kan toegekend worden aan het feit dat de methode geen expliciete representatie van de vrije bewegingsruimte hoeft uit te rekenen. De belangrijkste operatie is namelijk het testen of een plaatsing van de robot botst met een obstakel. Deze operatie kan tegenwoordig efficiënt uitgevoerd worden. De tweede reden van het succes is dat problemen die 'redelijk' zijn positieve bereikbaarheidseigenschappen hebben. Zo kan de vrije bewegingsruimte van een redelijk probleem vaak gerepresenteerd worden door een kleine verzameling samples waarbij elke sample verbonden kan worden met andere samples die 'ver weg' liggen. Hierdoor vindt de methode vaak snel een pad, zelfs als er veel obstakels zijn of als de robot veel verschillende typen bewegingen kan maken.

De afgelopen vijftien jaar hebben veel onderzoekers aan deze methode gewerkt. Dit leidde tot vele varianten van de methode met elk zijn eigen verdiensten. Het is lastig om deze varianten te vergelijken aangezien ze door verschillende mensen getest werden op verschillende typen virtuele omgevingen met verschillende computers en programma's.

In het eerste deel van dit proefschrift zal een vergelijkende studie worden verricht van gangbare varianten, geïmplementeerd op één systeem en getest op dezelfde problemen en dezelfde computer. In het bijzonder worden de verschillende manieren die bepalen of een lokaal pad botsingsvrij is, en technieken die bepalen hoe de samples gekozen en met elkaar verbonden worden bestudeerd. De resultaten zijn verrassend in de zin dat de technieken vaak anders presteren dan wordt geclaimd door de ontwerpers.

Naast het vergelijken op basis van het vinden van één vooraf gespecificeerde *query* (specificatie van het begin- en eindpunt van een pad), zal er een analyse worden gemaakt op basis van het kunnen oplossen van elk mogelijke query. De experimenten laten zien —in tegenstelling tot de algemene overtuiging— dat het verbonden krijgen van de samples in het algemeen veel lastiger is dan het met de samples overdekt krijgen van de botsingsvrije ruimte. Het verschil wordt groter naarmate er nauwere passages in de omgeving aanwezig zijn. Deze kennis kan men gebruiken om problemen adequater op te lossen. Zo wordt er een nieuwe krachtigere methode gecreëerd die de samples verbindt. Ook een hybride samplingmethode en een beter begrip van de parameters leiden tot een effectieve aanpak van het bewegingsplanningprobleem.

Het tweede deel van het proefschrift gaat over de kwaliteitsaspecten van paden en wegenkaarten. Omdat het soms lastig kan zijn om een pad te creëren, richten algoritmen zich slechts op het vinden van een enkele oplossing. Echter, voor de meeste toepassingen is het van belang dat er genoeg speling is tussen het pad en de obstakels, want het is vaak lastig om de precieze positie van de robot te meten en te controleren. Het bewegen langs een pad dat een bepaalde minimale hoeveelheid speling heeft, reduceert de kans op botsingen die veroorzaakt kunnen worden door deze onzekerheden. Ook dient het pad geen overbodige bewegingen te bevatten, want deze zullen langer duren om uit te voeren. Ten slotte dient een dergelijk pad snel berekend te kunnen worden.

Er worden twee algoritmen voorgesteld die de speling tussen het pad en de obstakels vergroten. De eerste is snel, maar de methode is beperkt tot rigide, schuivende robots. De tweede is langzamer, maar de methode kan omgaan met een breed perspectief aan robots zoals vrij-bewegende en aaneengeschakelde robots. Een groot voordeel van deze algoritmen is dat de speling langs paden nu efficiënt vergroot kan worden zonder dat complexe datastructuren en algoritmen nodig zijn.

Verder worden er algoritmen bestudeerd die de lengte van een pad kunnen verkleinen. Na observatie blijkt dat bestaande algoritmen lang niet altijd overvloedige (rotationele) bewegingen van een robot kunnen wegnemen. Een nieuw algoritme wordt voorgesteld dat deze succesvol verwijdert.

Daarna zal de *Reachability Roadmap Method* (RRM) worden geïntroduceerd welke kleine wegenkaarten creëert voor twee- en driedimensionale problemen. Zo'n kleine wegenkaart verzekert het vinden van een pad binnen een kort tijdsbestek en een minimaal gebruik van de hoeveelheid geheugen voor het opslaan van de wegenkaart. Ten slotte garandeert de methode dat een pad altijd gevonden wordt (indien hij bestaat) bij een gegeven opdeling in cellen van de vrije bewegingsruimte.

De genoemde technieken zullen worden verenigd met enkele nieuwe technieken om wegenkaarten te maken die in het bijzonder geschikt zijn voor interactieve virtuele omgevingen. Hierbij wordt de RRM als uitgangspunt gebruikt. Om te beschikken over alternatieve routes en korte paden worden nuttige ketens aan de wegenkaart toegevoegd. Vervolgens wordt speling aan de wegenkaart toegevoegd, wat men in staat stelt om zonder vertraging paden met hoge kwaliteit te verkrijgen.

Ten slotte wordt aangegeven dat het nuttig is om in de nabije toekomst een bibliotheek van bewegingsplanningtechnieken te bouwen die men als 'black box' kan gebruiken. Zodoende hoeven gebruikers niet na te denken over parameterkeuzes die meestal weinig zinnig voor ze zijn. In dit proefschrift wordt geprobeerd om deze keuzes zo veel mogelijk te automatiseren. Ook wordt inzicht verschaft in het effect van alle parameters op de besproken methoden. Hoewel de software, algoritmen en resultaten een basis voor de bibliotheek verschaffen, zal verder onderzoek nodig zijn de black box mogelijk te maken.

Het creëren van een dergelijke bibliotheek zal essentieel zijn voor de ontwikkeling van autonome robots.

ACKNOWLEDGEMENTS

Many people contributed to this thesis in one way or another, and I wish to express my gratitude toward them.

First of all, I would like to thank my promotor, Mark Overmars. During our regular meetings, Marks' bright observations always helped me to see 'the big picture'. His many suggestions and improvements have helped shape this thesis.

Next, I would like to thank the members of the reading committee, Nancy Amato, Frans Groen, Dan Halperin, Jean-Paul Laumond and Peter Werkhoven for reading my thesis and for their helpful suggestions and discussions. I would also like to thank Marcel and Rodrigo for proof reading the manuscript. Marcel, I especially liked our nighttime tea and coffee breaks before the great day had dawned.

A word of thanks goes to all my colleagues at the Department of Information and Computing Sciences of Utrecht University for having a good time during coffee breaks, lunches and conferences. Due to many hours of playing darts with Guido, Han, and Twan *et al.*, and playing table tennis with Arno, Dennis, Little O and Big O, I did not get RSI. Geert-Jan, thanks for your help on C++.

I want to thank Remco Veltkamp and Dirk Thierens for supervising me during the writing of my M.Sc. thesis. It was during that time that I became convinced that I wanted to do a Ph.D.

I would like to thank the Netherlands' Organisation for Scientific Research (NWO) for funding my position and the Advanced School for Computing and Imaging (ASCI) for providing four nice holidays and partly funding my thesis.

Lastly, I would like to thank Suus (a.k.a. mevrouw Jeurnink), my parents and my friends, for their endless support. Thank you all.

CURRICULUM VITAE

Roland Jan Geraerts was born in Maaseik, Belgium in 1978. From 1990 to 1997, he received his preparatory education at the Scholengemeenschap Sint Ursula in Horn, the Netherlands. From 1997 to 2001, he studied computer science at Utrecht University, the Netherlands. His Masters thesis, entitled 'Pose Estimation with Evolution Strategies', was completed under the supervision of dr. Dirk Thierens and dr. Remco Veltkamp. In 2002, he started as a Ph.D. student under the supervision of Prof. dr. Mark H. Overmars at the same university. In 2006, he completed his thesis there.

Colofon

This thesis was typeset by the author in $\LaTeX\,2_\varepsilon$. The main body of the text was set using 11 points Palatino font. The charts were rendered using Gnuplot 4.0.
The cover was designed by Studio Hedris.
No computers were harmed during the production of this thesis.

Advanced School for Computing and Imaging

This work was carried out in the ASCI graduate school.
ASCI dissertation series number 121.

ISBN-10: 90-393-4229-6
ISBN-13: 978-90-393-4229-9

© 2006 by Roland Geraerts. All rights reserved.