MIKA RANTANEN

Improving Probabilistic Roadmap Methods for Fast Motion Planning

■

ACADEMIC DISSERTATION
To be presented, with the permission of
the Board of the School of Information Sciences of the University of Tampere,
for public discussion in the Auditorium Pinni B 1096,
Kanslerinrinne 1, Tampere, on August 28th, 2014, at 12 o'clock.

UNIVERSITY OF TAMPERE

MIKA RANTANEN

Improving Probabilistic Roadmap Methods for Fast Motion Planning

Acta Universitatis Tamperensis 1958
Tampere University Press
Tampere 2014

UNIVERSITY
OF TAMPERE

ACADEMIC DISSERTATION
University of Tampere
School of Information Sciences
Finland

The originality of this thesis has been checked using the Turnitin OriginalityCheck service in accordance with the quality management system of the University of Tampere.

Cover design by
Mikko Reinikka

Distributor:
kirjamyynti@juvenes.fi
http://granum.uta.fi

Acta Universitatis Tamperensis 1958
ISBN 978-951-44-9528-1 (print)
ISSN-L 1455-1616
ISSN 1455-1616

Acta Electronica Universitatis Tamperensis 1443
ISBN 978-951-44-9529-8 (pdf)
ISSN 1456-954X
http://tampub.uta.fi

Suomen Yliopistopaino Oy – Juvenes Print
Tampere 2014

Abstract

The goal of motion planning is to find a collision-free path for a robot that moves among obstacles. The problem originates from robotics where it is a fundamental part of autonomous robots. However, motion planning can be used in many other application areas as well, for example, in computer games and molecular simulations.

Motion planning is a difficult problem and exact algorithms are rarely useful in practice. Therefore, many approximate algorithms have been proposed to solve it. One promising approach is to use probabilistic roadmap planners that work by constructing a roadmap for a robot. This roadmap uses information from static obstacles and it contains simple, collision-free path segments that are combined together. The robot can then use it as a guide while moving. Probabilistic roadmap planners can work remarkably well even in complex environments with robots that have many degrees of freedom but still there are cases where their performance can be poor.

This thesis investigates these planners and tries to find ways to enhance their performance. Five different methods to improve them are proposed and tested experimentally in simulated environments. Three of them concentrate on speeding up the construction of roadmaps in various ways. One proposed method tries to build a roadmap in such a way that it works well in changing environments where all obstacles are not known in advance. The last method can be used to reduce the size of roadmaps. Results from the experiments show that the suggested methods work well and can indeed lead to better planners.

Acknowledgments

I wish to express my sincere gratitude to my supervisor, Professor Martti Juhola, for the guidance and encouraging attitude that I received throughout the preparation of this doctoral thesis. I would also like to offer my gratitude to the preliminary examiners of my thesis manuscript, Professor Matti Nykänen and Professor Tapio Pahikkala, for their valuable feedback. Their comments were useful and helped me to finalize this thesis. I also want to thank Professor Juha Röning who has agreed to be my opponent in the public defense. Additionally, I also want to thank all anonymous peer reviewers who gave insightful feedback from my article manuscripts.

I acknowledge the financial support I received from the Tampere Doctoral Programme in Information Science and Engineering (TISE). The funding made it possible for me to work with this thesis in full time for two years. I want to thank the School of Information Sciences for providing me with a pleasant working environment. I want to thank all the personnel and especially my colleagues from the Data Analysis Research Group.

I also want to offer my thanks to my parents, Heli and Matti, and my brother, Rami, for their love and encouragement. Finally, I want to give my deepest thanks to my beloved Ville whose presence and support during this whole research process has made everything so much easier.

Tampere, May 2014

Mika Rantanen

Contents

1 Introduction **1**

2 Motion Planning **5**
2.1 Problem Formulation 5
2.2 Solving the Motion Planning Problem 6

3 Probabilistic Roadmap Planners **9**
3.1 Basic Algorithm 9
3.2 Problems with Probabilistic Roadmap Planners 15

4 Improving Probabilistic Roadmap Planners **17**
4.1 Sampling Methods 18
4.2 Neighbor Selection 23
4.3 Connecting Configurations 28
4.4 Improving Paths 29
4.5 Dynamic Environments 30

5 Overview of Publications and Results **33**
5.1 Publication I 33
5.2 Publication II 34
5.3 Publication III 35
5.4 Publication IV 36
5.5 Publication V 37

6 Conclusion **39**

Bibliography **41**

List of Original Publications

The thesis is based on the following publications. In the text, they are referred to by their Roman numerals.

I Mika T. Rantanen and Martti Juhola. A configuration deactivation algorithm for boosting probabilistic roadmap planning of robots. *International Journal of Automation and Computing*, 9(2):155–164, 2012.

II Mika T. Rantanen. A connectivity-based method for enhancing sampling in probabilistic roadmap planners. *Journal of Intelligent & Robotic Systems*, 64(2):161–178, 2011.

III Mika T. Rantanen and Martti Juhola. Using probabilistic roadmaps in changing environments. *Computer Animation and Virtual Worlds*, 25(1):17–31, 2014.

IV Mika T. Rantanen and Martti Juhola. Speeding up probabilistic roadmap planners with locality-sensitive hashing. *Robotica*, forthcoming.

V Mika T. Rantanen and Martti Juhola. How to construct small probabilistic roadmaps with a good coverage? Manuscript submitted for publication.

Personal Contributions

The thesis consists of five publications. In the following, the contribution of the author of this thesis is described for each publication.

I The idea came from the author who also implemented and conducted all the experiments. The publication was mainly written by Martti Juhola.

II The idea came from the author who also implemented and conducted all the experiments. The publication was written by the author.

III–V The idea came from the author who also implemented and conducted all the experiments. The publication was written by the author. Martti Juhola supervised the research.

The author also implemented a software framework that was used to test different probabilistic roadmap planners in simulated environments. This software was used in all experiments.

CHAPTER 1

Introduction

The task in motion planning is to generate a path for a robot between two locations. In a basic case, a rigid-body robot moves in a fully known static environment that has obstacles which do not move or change shape. The robot must not collide with the obstacles while moving. The basic motion planning problem is also known as a piano movers' problem [77]. One can think that there is a free-flying piano in a room and the task is to find a way to move it to another location without colliding with furniture. The motion planning problem has been studied extensively in the last decades (see e.g. books [20, 51, 53] for a good summary).

Solving the motion planning problem is especially important in robotics. A capability to navigate in an environment without guidance is an essential task that autonomous robots must handle. There are many kinds of robots that can benefit from motion planning, for example industrial robots and mobile robots.

Industrial robots can be used in a variety of tasks, like in welding and assembling [67]. Traditionally, industrial robots have not been very autonomous and they have just been programmed to repeat some series of precise actions. Nowadays industrial robots may have more independence and they can make some decisions themselves. For example, robots may use machine vision to help them observe an environment and react accordingly.

The movement of industrial robots is typically quite limited and they are often attached to a fixed base. On the other hand, mobile robots are designed to move more freely in different environments. Mobile robots benefit greatly from the motion planning because it allows them to move without human guidance. Therefore, there has been a lot of active research done in the area and many kinds of mobile robots have been studied. These include, among others, unmanned aerial vehicles [21] and floor cleaning robots [19, 32].

Besides robotics, the motion planning can also be used in other application areas. For example, many computer games have moving characters that must navigate and find a path between two locations in a virtual environment [16].

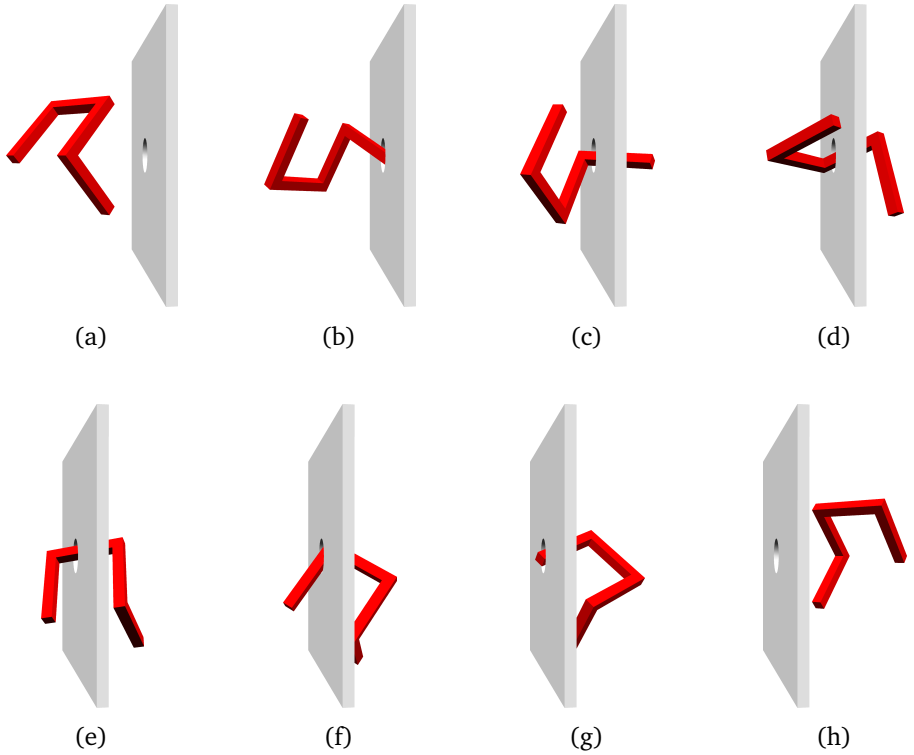


Figure 1.1: An example of motion planning where a hook-shaped robot is going from one side of wall to the other through a small hole. A start location is shown in (a) and a goal location in (h). Six intermediate locations are shown in (b)–(g).

In early games, the movement of characters was often very limited and at least somewhat predetermined. Nowadays, games are usually more realistic than before and therefore it is preferable that also the movement of characters would be as realistic as possible. Characters should move along smooth paths and intelligently react to the changes in an environment. Many motion planning methods that are studied in the field of robotics can also be useful in games.

Motion planning has an important part also in computer-aided design where it can be used especially in virtual prototyping [3]. A good example is a method in [70] where motion planning was used to determine whether there exists an accessible path of travel for a wheeled mobility device through a building. The method is implemented directly to a CAD environment.

One interesting application area for motion planning is computational biology [2, 85]. With some modifications, many motion planning algorithms can be applied to study molecular simulations. Typically, these are difficult problems but lately good results have been achieved with some motion planning methods. Motion planning has, for example, been used to simulate protein folding successfully [87].

Figure 1.1 shows an example of motion planning. In the example, an environ-

ment is three-dimensional and there is one robot and one obstacle in it. The robot is a hook-shaped rigid body that can move and rotate freely, and the obstacle is a wall that has a small circular hole in it. The example shows how the robot can move from one side of the wall to the other through the hole. The example is not trivial because the robot is too large to go directly through the hole. Instead, the robot must rotate itself extensively to be able to go to the other side.

All these applications require a fast and reliable solution to the motion planning problem. Unfortunately, exact solutions are typically too slow to be useful in practice. To overcome this, many approaches have been studied and one prominent solution is to use sampling-based motion planning methods [24]. For example, the problem in Figure 1.1 was solved with a probabilistic roadmap planner which is a sampling-based method. These planners work by first constructing a roadmap that consists of simple free paths connected together. After that, the robot can just use the roadmap to move freely on the environment.

Probabilistic roadmap planners are also the main theme in this thesis. The research consists of this summary and five original publications. All publications concentrate on different aspects of construction of a roadmap. Three main goals in this thesis are

1. to find different ways to speed up the construction of roadmaps,

2. to investigate if it is possible to construct a roadmap in such a way that it works well in changing environments, and

3. to find a way to construct roadmaps that are small but still useful.

All the methods and ideas presented in the publications were experimentally tested in simulated environments and compared with some other methods that have been proposed in literature. A more detailed description of these publications and their contributions is in Chapter 5.

In the experiments of this thesis, a robot is either a rigid body or a set of multiple independent rigid bodies. In any case, it is assumed that the robot can freely move in all directions in an environment. It should be noted that the methods presented in the publications are not directly applicable to nonholonomic robots such as cars which can move only forward and backward but not sideways. Furthermore, probabilistic roadmap planners have been studied mainly in simulated environments so using them with real robots can still be challenging [88].

The rest of this summary is organized as follows. Chapter 2 reviews the basic concepts of motion planning as well as some methods that can be used to solve it. Chapter 3 shows a basic algorithm that is a base for probabilistic roadmap planners and therefore also a base for methods presented in the publications. In Chapter 4, many existing techniques to improve probabilistic roadmap planners are discussed. Chapter 5 overviews the publications and results of this thesis. A conclusion of the thesis is in Chapter 6.

CHAPTER 2

Motion Planning

In this chapter, some basic concepts of motion planning are shortly described and the problem is formulated. A useful concept is a configuration space that is nowadays typically used in motion planners. Over the years, many algorithms have been proposed to solve the motion planning problem and some of those solution methods are also reviewed in this chapter.

2.1 Problem Formulation

In motion planning, the robot moves in a workspace $\mathcal{W}$ which is in practical cases either $\mathbb{R}^2$ or $\mathbb{R}^3$. Usually, the location of a robot is represented as a configuration [60, 61]. The configuration of a robot is a minimal set of parameters that are needed to describe exactly one location of the robot.

For example, if the robot is a rigid body that moves in two-dimensional space but cannot rotate, the exact position can be expressed as two coordinates which means that the configuration has two parameters. In a case of a rigid-body robot that moves in three-dimensional workspace and can both translate and rotate, an exact location can be expressed as a configuration that has six parameters. Three are needed for the position and three for the orientation.

A set of all possible configurations for a robot is called a configuration space. If configurations have d parameters, the configuration space is d-dimensional and, therefore, the configuration space is often higher dimensional than the workspace. It is useful to describe the robot as a configuration because it allows us to treat the robot as a point in a d-dimensional space. Instead of finding a path for a robot directly in a workspace, we can find a path for a point in the configuration space which is usually much more convenient because it allows to handle different robots in the same manner.

In each configuration, the robot occupies some set of points in a workspace $\mathcal{W}$. This set of occupied points in configuration q is denoted as $R(q)$. Typically, $\mathcal{W}$

contains obstacles in addition to the robot. Let us assume that in $\mathcal{W}$, there are n obstacles denoted by $\mathcal{O}_i, 1 \leq i \leq n$. For each $\mathcal{O}_i$ there is a counterpart $\mathcal{CO}_i$ in the configuration space $\mathcal{C}$. This configuration space obstacle can be defined as

$$\mathcal{CO}_i = \{ q \in \mathcal{C} \mid R(q) \cap \mathcal{O}_i \neq \emptyset \},$$

which means that $\mathcal{CO}_i$ is a set of all configurations where the robot R would collide with the obstacle $\mathcal{O}_i$.

In case of a rigid-body robots, the free configuration space can be defined as

$$\mathcal{C}_{\text{free}} = \mathcal{C} - \bigcup_{i=1}^{n} \mathcal{CO}_i,$$

and it is a set of configurations where the robot does not collide with any of the obstacles. However, this is insufficient in case of robots that consists of multiple bodies. In that cases, the collision with obstacles must be checked separately for each body and in addition, collisions between different bodies must also be taken into consideration.

The task in motion planning can be defined as finding a free path from configuration q_{start} to configuration q_{goal}. The path is a continuous function

$$\tau : [0, 1] \rightarrow \mathcal{C}_{\text{free}},$$

where $\tau(0) = q_{\text{start}}$ and $\tau(1) = q_{\text{goal}}$. Because the path lies totally in a free configuration space, it is guaranteed that the robot does not collide with any obstacles on the workspace while moving along it.

It is very difficult to solve this motion planning problem exactly especially if the configuration space is complex and high-dimensional. In fact, it has been shown that motion planning is PSPACE-complete [15, 74]. Therefore, the problem is usually solved with approximate methods in practice.

2.2 Solving the Motion Planning Problem

One common approach to find a path for a robot is to use a grid-based search [98]. The idea is to divide the configuration space into a grid. The cells of the grid are typically uniformly shaped and in the two-dimensional case they can be, for example, squares or hexagons. Each cell can be either free or blocked. Free cells are those that do not have obstacles and blocked cells are those ones that have. A robot occupies one cell at a time and can move to the adjacent cells. It is possible to think that the free cells form a graph and hence, the path between two cells can be found with some graph search algorithm.

This grid-based approach is used especially in computer games where the game world is traditionally divided into a two-dimensional grid of cells [82, 98].

For many games, this works well enough and it is still widely used and studied. For example, in [7], a grid-based path finding method for human characters was proposed. One problem with grid-based methods is that it is possible to move from one cell only to the adjacent cells which causes the final path be unnecessary long and unnatural looking. Any-angle methods, such as [22, 25], try to overcome this issue by allowing the robot to move more freely. However, these methods can still be quite restricted and it might be difficult and slow to use them in complex and realistic environments. For this reason, more sophisticated motion planning methods have been proposed for games (see e.g. [39, 57, 59, 64]).

Cell decomposition methods [18, 51] resemble a basic grid search in some ways. Instead of uniformly shaped cells, $\mathcal{C}_{\text{free}}$ is divided into cells that can all be differently shaped and sized. Just like in basic grid search, two cells are adjacent to each other if they have a common boundary. Because the cells are not uniformly shaped, the information about the adjacency of the cells is explicitly stored in a graph. The robot can move freely inside a cell and based on the information of the adjacency graph, the robot can move also to other cells. It is important that it is very easy to find a path for a robot between two configurations inside one cell. Therefore, the cells should preferably be convex.

There are different ways to exactly decompose $\mathcal{C}_{\text{free}}$ into cells [1, 18, 51]. Another possibility is to use approximate methods where the cells usually have some predefined shape but their size may differ [51]. The problem with cell decomposition methods, whether they are exact or approximate, is their need for a large number of cells in complex environments. These methods typically quickly become impractical as the dimension of the configuration space increases.

In [45], an artificial potential field was used to guide the robot. The idea is that obstacles cause repulsive forces to the robot while the goal configuration causes an attractive force. Together these forces determine where the robot should move from its current configuration. One major problem with these potential field methods is that the robot may end up in a local minimum and never reach the goal. Several techniques to solve this issue have been proposed, for example in [8, 17, 75].

In addition to presenting an improvement for potential field planning, the method in [8] is also one of the earliest sampling-based methods. The method, called Randomized Path Planner (RPP), uses potential fields to find a path and if it gets stuck in a local minimum, it tries to use random motions to escape it. The downside of the method was that it required many parameters that had to be set.

The Ariadne's clew algorithm [11] was another early sampling-based method and it tries to grow a tree from the start configuration to the goal. While the RPP method just tried to find a path to the goal and escape local minima when needed, the Ariadne's clew algorithm does also some exploration of the free space.

Many other sampling-based methods have also been proposed. For example, rapidly-exploring random tree (RRT) and its variations have become a popular way to solve difficult motion planning problems [52, 55, 56]. As Ariadne's clew algorithm, also an RRT method tries to grow a tree starting from the start configu-

ration. The tree is built incrementally by randomly selecting a configuration from the configuration space and then extending the tree toward that configuration. RRT methods tend to expand the tree towards unknown areas in configuration space which means that they can quickly explore $\mathcal{C}_{\text{free}}$. It should be noted that RRT methods are very useful also with car-like robots [55].

Many methods mentioned in this section build some kind of roadmap. For example, the graph formed by cell decomposition methods in order to encode the adjacency of the cells can be seen as a roadmap as well as trees built with the sampling methods. A roadmap is an approximation of $\mathcal{C}_{\text{free}}$ and building it instead of trying to get an exact representation can be very useful in higher-dimensional problems. Probabilistic roadmap planners try to do just that and they are investigated in more detail in the next chapter.

CHAPTER 3

Probabilistic Roadmap Planners

Probabilistic roadmap (PRM) planners [44] are nowadays a popular method to solve difficult and high-dimensional motion planning problems. They are not exact methods but they work remarkably well and quickly in practice. It is also easy to implement PRM planners and they can be used with many kinds of robots.

The probabilistic roadmap planners were introduced in [44] but the work was based on a previous research that was conducted by two independent groups [43, 68, 69]. In [68], a random approach to motion planning was suggested. The suggested method tried to connect two configurations together by building a random network of short paths in the free configuration space $\mathcal{C}_{\text{free}}$. This network is a graph that can also be called a roadmap. PRM planners use roadmaps to represent $\mathcal{C}_{\text{free}}$ in a simple form. The nodes of the roadmap correspond to free configurations and an edge between two nodes means that there exists a simple collision-free path between the corresponding configurations.

The method in [68] was further improved in [69] by dividing it into two phases. In the learning phase, the network of paths was constructed, and in the query phase, this network was used to solve motion planning queries. A similar two-phase approach was independently introduced in [43] and the same approach is still used in current state-of-the-art PRM planners.

In this chapter, a basic PRM algorithm is first presented and then some problematic issues that arise are discussed. The presented algorithm is also used as a base for all methods and experiments in the publications.

3.1 Basic Algorithm

Probabilistic roadmaps are sampling-based methods which do not try to construct an exact representation of $\mathcal{C}_{\text{free}}$. Instead, they utilize the fact that it is quite easy to check whether some configuration is collision-free by using some collision detection algorithms [40, 47, 73]. With these algorithms it is also possible to check collisions

for a local path which is some simple path segment between two configurations. Using these methods, PRM planners can build a roadmap that lies entirely in C_{free}.

Even though the original PRM planner presented in [44] works very well in many environments, it is possible to enhance it in many ways. As a result, several different PRM planner variations have been proposed over the years. However, usually these variations modify only some part of the original PRM planner while other parts are kept intact. A typical PRM planner contains the following parts:

- a sampling method to generate new configurations,
- a method to select neighbor configurations from the roadmap,
- a local planner to connect configurations together with a local path, and
- an ending condition which is used to decide when the roadmap is ready.

By modifying these common parts, it is possible to customize the planner to take into the account possible particularities of different environments. Because different PRM planners share many common features it is possible to build a simple framework for PRM planners. The basic algorithm discussed in this section is one such framework.

Just like other PRM planners, also the basic algorithm works in two phases. In the first phase, the roadmap is constructed and in the second phase it is used to answer queries. The learning phase is typically the most time-consuming part of PRM planners but after the roadmap has been built, it is very fast to solve the motion planning queries. This is very useful especially when multiple queries must be solved. This also distinguishes the PRM planners from many other motion planning algorithms which often can solve only one query at a time.

3.1.1 Constructing a Roadmap

Algorithm 3.1 shows how the basic PRM method constructs a roadmap. The algorithm is simplified and many important details are missing. These details are discussed in later sections more thoroughly.

The algorithm starts with an empty roadmap. The main loop is in lines 3–12. In line 4, at the beginning of one iteration, a free configuration q is generated and in line 5 this new configuration is added to the roadmap as a node. The free configuration is typically generated randomly using some sampling method. The simplest way is to generate configurations uniformly at random until the configuration is collision-free. Different sampling methods are discussed more detailed in Section 4.1.

In line 6, a set of neighbor configurations are chosen for q from the roadmap. Usually, this means that some predetermined number of the nearest configurations

Algorithm 3.1: A basic probabilistic method to construct a roadmap.

Output:
A roadmap $G = (V, E)$.

```
1:  V ← ∅
2:  E ← ∅
3:  repeat
4:     q ← a randomly chosen configuration from C_free
5:     V ← V ∪ {q}
6:     N_q ← all nearest neighbor configurations of q chosen from V
7:     for all q' in N_q do
8:        if the local planner Δ can find a free path between q and q' then
9:           E ← E ∪ {(q, q')}
10:       end if
11:    end for
12: until there are enough configurations in V
13: return G
```

are selected. Several different ways to effectively find the nearest neighbors and calculate a distance between two configurations are described in Section 4.2.

Then, in lines 7–11, the algorithm goes through all these neighbors. For each neighbor q', a local planner is used in line 8 to check whether there is a simple and free path between q and q'. If the local planner finds a free path, an edge (q, q') is added to the roadmap in line 9. The simplest local planner just tries to connect the configurations by a straight line in the configuration space. Typically, it discretizes the line and checks each step for collisions. Other ideas to implement the local planner have been proposed and some of those are described in Section 4.3.

The nodes are added to the roadmap until some ending condition has been met. The roadmap should have a good coverage and connectivity at the end but in practice it can be quite difficult to know when these requirements are met. One possibility is to continue building the roadmap until some predetermined motion planning queries can be answered with it. Another common ending condition is to decide how many nodes the roadmap should have beforehand and stop the building when this has been achieved.

The basic PRM planner assumes that the used local planner Δ is deterministic which means that every time Δ is called for two configuration a and b, the returned path is the same. This is useful property since it allows us to store only the edge (a, b) to the roadmap to mark that there is a free path between a and b. It is not necessary to store the local path $\Delta(a, b)$ itself since it can always be calculated later. At that time, the calculation can be done quickly without collision checking because it is already known that the local path is collision-free.

In the basic case, the roadmap is an undirected graph which means that edges

(a, b) and (b, a) can be considered identical. Therefore, if Δ finds a path from configuration a to configuration b, it can also find a path from b to a. Usually local planners are also symmetric, i.e., the path $\Delta(a, b)$ is the same as $\Delta(b, a)$, just in the opposite direction.

It is important that a roadmap has a good coverage and connectivity. When a roadmap has full coverage, a local planner is able to connect any configuration in $\mathcal{C}_{\text{free}}$ to at least one configuration that already exists in the roadmap. However, it is difficult to achieve a full coverage in practice.

The connectivity measures how roadmap nodes are connected with each other. A roadmap has as good connectivity as possible when two arbitrary configurations q_1 and q_2 in the roadmap belong to the same connected component if and only if there is a free path between q_1 and q_2 in $\mathcal{C}_{\text{free}}$. If the connectivity is not good, it is possible that a PRM planner fails to find a path even though the path actually exists in a workspace.

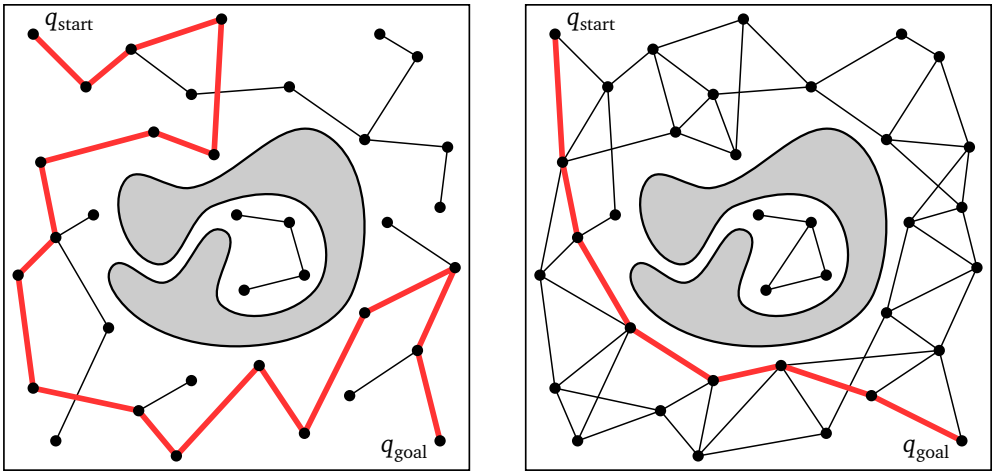
It should be noted that cycles in a roadmap do not have an effect on its coverage or connectivity. Therefore, it is not always necessary to add them. It is much faster to construct a roadmap that is just a set of trees, because every time the planner tries to add an additional edge to the roadmap, it must use a local planner to check if the edge is collision-free and that tends to be a rather slow operation. Additional edges also increase the memory required to store the roadmap.

Cycles may, however, be desired because without them the retrieved paths can easily be unnecessarily long and complex. A roadmap with cycles can also have many alternative routes for a robot between two configurations. This can be useful especially in dynamic environments where obstacles can be moving. If some path becomes blocked by an obstacle, it is possible that there is some other free path in the roadmap that can be used instead. Without cycles, this would be impossible.

Figure 3.1 illustrates roadmaps in the two-dimensional configuration space. In both figures, the obstacles are identical but in Figure 3.1(a), the roadmap has been built without cycles while in Figure 3.1(b) some additional edges have been added. Also, the nodes are in exactly the same places in both figures which means that the coverage of both roadmaps is the same. In both figures, the roadmap has the same connected components, so there are no differences in the connectivity either. However, the path length between nodes q_{start} and q_{goal} differs. The path is much longer in Figure 3.1(a) than it is in Figure 3.1(b) that has cycles.

3.1.2 Solving Queries

Algorithm 3.2 shows how a roadmap can be used to solve one motion planning query. As input parameters, the algorithm needs a previously constructed roadmap $G = (V, E)$, a start configuration q_{start}, and a goal configuration q_{goal}. The algorithm tries to connect q_{start} and q_{goal} to the roadmap and then find and return a free path between those configurations.



(a) A roadmap without cycles (b) A roadmap with cycles

Figure 3.1: An example of roadmaps in the two-dimensional configuration space. Gray areas are obstacles and small dots are roadmap nodes. The obstacles and nodes are the same in both figures but in (a) there are not cycles in the roadmap while in (b) there are. The shortest path between nodes q_{start} and q_{goal} is shown in both figures.

In lines 1 and 2, the algorithm chooses a set of the nearest neighbor configurations for both q_{start} and q_{goal}. Then both configurations are added to the roadmap in lines 3 and 4. Ideally, all configurations from the roadmap should be selected as neighbors to maximize the probability that q_{start} and q_{goal} can be connected to the roadmap. In practice, it is usually enough to select only some of the nearest nodes.

In lines 5–9, there is a loop that goes through all neighbors for q_{start}. For each neighbor q', the algorithm checks whether a local planner can find a free path from q_{start} to q'. If such path exists, an edge (q_{start}, q') is added to the roadmap. In lines 10–14, the same is done for q_{goal}.

In line 15, the algorithm uses some graph search algorithm to find a path between q_{start} and q_{goal} from the roadmap graph. If the path is found, it is returned. Otherwise the algorithm returns `NotFound` which informs that the path could not be found. The path that the algorithm returns is a sequence

$$q_{\text{start}} = v_0, e_1, v_1, e_2, v_2, \ldots, e_n, v_n = q_{\text{goal}},$$

where $v_i \in V$, $0 \leq i \leq n$ and $e_j \in E$, $1 \leq j \leq n$. To retrieve the actual path in $\mathcal{C}$, a local planner Δ must be used to calculate local paths for each edge. The corresponding local path for edge e_j is $\Delta(v_{j-1}, v_j)$ and the actual path from q_{start} to q_{goal} can be composed by concatenating all these local paths together.

It should be noted that the algorithm sometimes fails to find a path even when there would actually be a free path in the configuration space for a robot. There are two reasons why this could happen. It is possible that the algorithm is not able to connect q_{start} or q_{goal} to the roadmap with a local planner. This can happen if the

Algorithm 3.2: Solves a motion planning query by using a roadmap.

Input:
$G = (V, E)$: the roadmap
q_{start}: the start configuration
q_{goal}: the goal configuration

Output:
A shortest path in G from q_{start} to q_{goal} or `NotFound` if a path cannot be found.

```
1:  N_start ← all nearest neighbor configurations of q_start chosen from V
2:  N_goal ← all nearest neighbor configurations of q_goal chosen from V
3:  V ← V ∪ { q_start }
4:  V ← V ∪ { q_goal }
5:  for all q' in N_start do
6:     if the local planner can find a free path between q_start and q' then
7:        E ← E ∪ { (q_start, q') }
8:     end if
9:  end for
10: for all q' in N_goal do
11:    if the local planner can find a free path between q_goal and q' then
12:       E ← E ∪ { (q_goal, q') }
13:    end if
14: end for
15: P ← a shortest path from q_start to q_goal
16: if P is empty then
17:    return NotFound
18: else
19:    return P
20: end if
```

roadmap does not cover $\mathcal{C}_{\text{free}}$ well or if too few neighbors are selected in lines 1 and 2. Another possibility is that the connectivity of the roadmap is not good enough. In that case, q_{start} and q_{goal} will be connected to different components of the roadmap and the graph search fails to find a path between them.

In the shown algorithm, q_{start} to q_{goal} are added to the roadmap permanently. Another possibility is that the nodes are added only temporarily to the roadmap and that they are removed after the path has been found. In that case, also the edges are stored only temporarily, so it might be enough to try to connect q_{start} and q_{goal} to the roadmap with as few edges as possible. Ideally, both of these configurations should be connected to the same connected component of the roadmap with just one edge.

3.2 Problems with Probabilistic Roadmap Planners

It is important that PRM planners build a roadmap in such a way that it covers $\mathcal{C}_{\text{free}}$ well and captures the connectivity. Usually, this can be done easily with a small roadmap in terms of both nodes and edges. However, to reach this goal in difficult configuration spaces, the basic PRM planner sometimes tends to produce quite large roadmaps. This is a problem because a large roadmap requires huge amounts of memory and also slows down the usage of the roadmap.

Narrow passages are the most common reason why the roadmap grows too large. They are regions in configuration space where the obstacles are very near each other so that it is difficult for a robot to move without colliding with them. In [28], it was noted that typically PRM planners are able to cover $\mathcal{C}_{\text{free}}$ quite well with only a few nodes but the problem comes from the connectivity. If there are narrow passages in configuration space, it can be difficult for a planner to find a way through them with a local planner. The result is that the roadmap contains several components that are not connected together even when they should be. This problem was recognized already in the early research [44] and there has been much research done since to solve it.

An example of a narrow passage can be seen in Figure 3.1. The configuration space is divided into two regions by an obstacle. The regions are connected together with a narrow passage. The roadmap has been built in $\mathcal{C}_{\text{free}}$ but it contains two components. A PRM planner has not been able to connect the empty regions together through the narrow passage.

The basic PRM planner tries to solve the narrow passage problem simply by generating a great number of configurations randomly. Eventually, there would be enough configurations to go through narrow passages and connect the separate components of the roadmap together. As said, this would cause the roadmap to be unnecessarily large and it would also take quite a long time to build the roadmap. Another possibility is to sample configurations by biasing the sampling towards narrow passages. Generating configurations in narrow passages is slower than sampling them uniformly at random but usually a lot smaller number of configurations is enough to cover $\mathcal{C}_{\text{free}}$ and capture its connectivity. Therefore biasing the sampling usually reduces both the roadmap size and the time required to build it. It is also possible to use some advanced local planner that can find a way through the narrow passages. However, it is quite difficult to make such a planner that would work with different kinds of narrow passages.

One problem with PRM planners is that they are designed to work in static environments. However, in many practical applications, the environment is likely to be dynamic which means that there can be changes that are unknown in advance. This is true especially for mobile robots which may move outdoors. For industrial robots, it might be easier to assume that the environment is static.

Luckily, it is possible to improve PRM planners in such a way that they work also in dynamic environments. These kinds of planners can be very useful in situations

where the static obstacles are still known in advance. The planner can find a way through the static obstacles and for example through the narrow passages in a learning phase. The dynamic obstacles are taken into account during the query phase and at that time the robot can ignore the static obstacles because they are already handled.

The query phase brings yet another problem. It is possible that it may not always return a high-quality path. To solve this issue, it is possible to add an additional phase to PRM planners that is executed after the query phase. In that phase, the retrieved path is improved before it is actually used to guide the robot. The path can, for example, be shortened if the path returned by the query phase is unnecessarily long.

CHAPTER 4

Improving Probabilistic Roadmap Planners

Probabilistic roadmap planners work well in practice. In many environments, a small number of configurations is enough to cover the free configuration space and capture its connectivity and the roadmap can therefore be built quickly. However, the basic PRM planner has its limitations and in some situations the performance can be poor. Furthermore, the basic planner works only in static environments which limits its usability.

There are different ways to modify the basic PRM algorithm and often it is possible to enhance the performance of PRM planners considerably by using some heuristics. In this chapter, several useful heuristics are discussed which have been proposed over the years. These include different sampling methods in Section 4.1 and neighbor selection methods in Section 4.2 as well as methods that are used to connect configurations together in Section 4.3. These are also the most important parts of the PRM planners. Additionally, methods that can be used to improve retrieved paths are discussed in Section 4.4 and in Section 4.5 some PRM enhancements that can be used in dynamic environments are shortly described.

Many of these issues are also touched in publications. In Publication I, a new method is presented that can speed up the construction of a roadmap. In Publication II, a new region-based sampling method is presented. Publication III presents a method that can be used to decide which edges are important especially when there can be changes in the environment after the roadmap has been built. Publication IV experimentally compares some data structures that are used in neighbor selection. Publication V presents a method that can be used to decrease the number of nodes in the roadmap considerably.

It should be noted that different applications may have different requirements for roadmaps. For example, some applications may need to build new roadmaps often and in those cases it is important that the roadmaps can be built quickly. On the other hand, sometimes it is enough to build just one roadmap and then use it

for all queries. In those applications, the time required to build a roadmap may not be an important factor.

Another example is the size of a roadmap. Typically, it is good to use small roadmaps because they do not consume much memory. Unfortunately, paths retrieved from small roadmaps are often long and they may look unnatural. This can be a problem in some applications and therefore they may prefer to build larger roadmaps.

There are many areas in PRM planners that are not discussed in this thesis. These include, among others, the cases where multiple robots or the large crowds of robots must be controlled or the case where the robot is nonholonomic like a car. However, it is possible to handle these cases also with PRM planners.

4.1 Sampling Methods

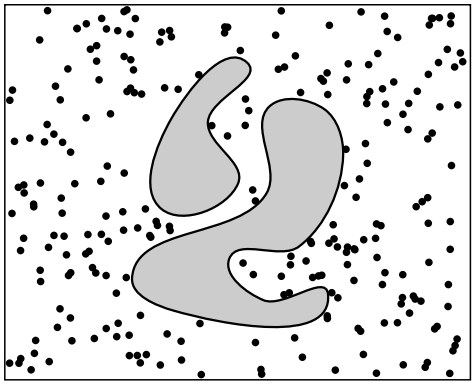
Sampling methods are one of the most extensively studied areas in PRM planners and many different samplers has been proposed. Typically, the narrow passages in the environment cause the PRM planners to work poorly. Therefore, many of the proposed sampling methods try to concentrate their sampling into narrow passages. The difficulty stems from the fact that it is very difficult to know where the narrow passages are.

There are many kinds of sampling methods. Some try to generate samples near the obstacles while some try to do just the opposite. Some try to divide the configuration space into regions and sample each region in different fashion. Because of this, the speed may vary much between the samplers as well as the distribution of the sampled configurations. It is also important to note that all sampling methods have some advantages and weaknesses. A method may work very well in some environments but not so well in the others. Therefore, none of the methods is clearly better than the others. Different sampling methods have been compared, for example, in [26].

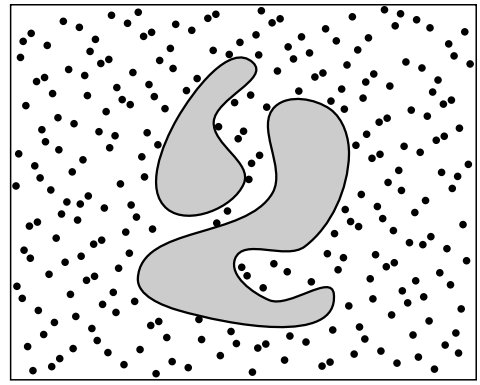
4.1.1 Quasi-Random Sampling

In the basic random sampling, configurations are generated randomly using a uniform distribution. The problem with random sampling is that often the configurations are not actually distributed very evenly [14]. An example of this is shown in Figure 4.1(a) where 250 configurations are generated using the random sampling. As can be seen, some nodes are very near each other and at the same time there are quite large areas where there are not nodes at all. This happens because each configuration is generated independently and they do not “know” where the previously generated configurations are.

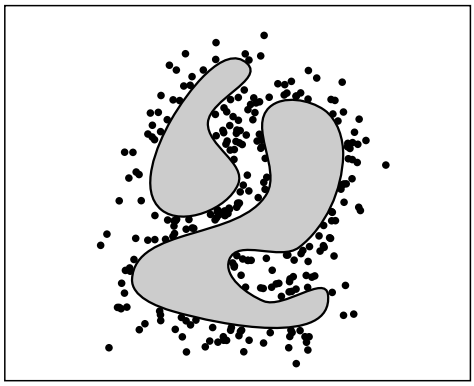
To cope with this problem, it is possible to use quasi-random methods which are deterministic and designed to produce more evenly distributed sequences



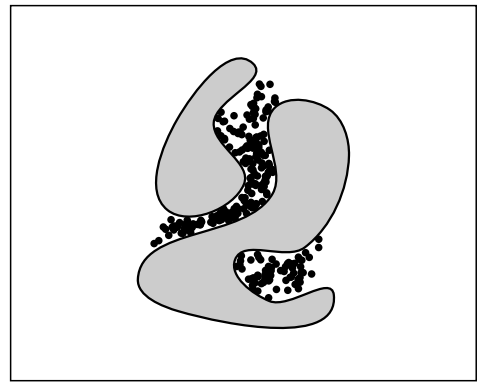
(a) The uniform sampling



(b) The Halton sampling



(c) The Gaussian sampling



(d) The bridge test sampling

Figure 4.1: An example of four common sampling methods in two-dimensional configuration space. Gray areas are obstacles and small dots are sampled configurations. There are 250 dots in each figure.

than the normal random method. These sequences are investigated from the motion planning perspective in [14, 54, 58]. The conducted experiments show that quasi-random sampling has typically better performance than the normal random sampling.

The Halton sampling is one common quasi-random sampling technique and it utilizes the Halton sequence. An algorithm for calculating the Halton sequence is described in [31] and it is investigated in more detail along with some other similar sequences in [46]. A difference between the normal uniform sampling and the Halton sampling can be seen by comparing Figures 4.1(a) and 4.1(b). The Halton sampling is able to distribute configurations in a more evenly manner than the sampler that generates configurations uniformly at random.

4.1.2 Sampling Near Obstacles

Many sampling methods that try to sample nodes near the obstacles have been proposed. The idea is that the difficult configuration space areas are near the obstacles and therefore the sampling should be concentrated there. In areas without obstacles, it is enough to generate only a few samples and still achieve good coverage.

The Gaussian sampler [13] is a popular method to sample near the obstacles. It starts by generating a random configuration q_1 using a uniform distribution. Then it chooses a length d using a normal distribution and generates a configuration q_2 that is at distance d from q_1. The sampler then checks whether q_1 and q_2 are in collision or not. If one configuration is collision-free and the other is in collision, the collision-free configuration is added to the roadmap. The configurations are discarded if both are in collision or both are collision-free.

An example of Gaussian sampling is shown in Figure 4.1(c). The nodes are generated around the obstacles and further away from them there are large areas where there are no nodes. This is because the sampler generates two nodes and requires that one is in collision while the other is not. It is possible to control how near obstacles the generated nodes are by modifying the standard deviation of the normal distribution used by the sampler.

One of the first sampling methods proposed to enhance the original uniform sampling was the one used in OBPRM [4]. This method also tries to sample configurations near the obstacles. It starts by generating a random configuration q_{coll} that is in collision and by selecting a random direction d. Then it finds some free configuration q_{free} in a direction d starting from q_{coll}. At the end, it uses a binary search to find a free configuration q between q_{coll} and q_{free} that is near enough to the surface of the obstacle. The configuration q is then added to the roadmap.

Another method is UOBPRM [99]. It has many similarities with OBPRM but it tries to produce more uniformly distributed samples near the obstacles. It works by generating a random line segment with a predetermined length to the configuration space. Then the method finds intersections between the segment and configuration space obstacles. For each intersection, the method selects one free configuration near that intersection point and adds it to the roadmap.

4.1.3 Sampling in Narrow Passages

Some methods concentrate sampling especially inside narrow passages. One such example is the bridge test sampler [34]. It starts by generating a configuration q_1 using a uniform sampler. Then it selects a length d and generates a configuration q_2 in such a way that the distance between q_1 and q_2 is d. Now q_1 and q_2 are the end points of a “bridge”. The sampler checks whether q_1 and q_2 are both in

collision. If they are, a midpoint configuration q of the line segment between q_1 and q_2 is checked for collisions. If q is collision-free, it is added to the roadmap. In all other cases, none of the configurations is added and this procedure is started all over again.

The length d and the orientation of the bridge are typically selected using normal distribution just like in the Gaussian sampler. The standard deviation of the normal distribution determines what the typical bridge length is. The standard deviation must be selected carefully to ensure that the bridge lengths are suitable for the environment.

The bridge test sampler is capable of sampling nodes only in configuration space regions where it can build bridges. This is a major disadvantage of bridge test sampling because it easily leads to roadmaps that do not have a good coverage and connectivity. An example of this can be seen in Figure 4.1(d). All nodes are sampled in two distinct areas which cannot be connected together when a straight-line local planner is used. Furthermore, there are areas where the local planner cannot reach any of the sampled nodes.

Luckily, it is easy to circumvent this problem by using hybrid sampling [34, 84] which uses the uniform sampler alongside with the bridge test sampler. The hybrid sampler generates most of the configurations by using the bridge test but occasionally some configurations are generated with a normal uniform sampling. Therefore, the hybrid sampler can sample narrow passages effectively while still being able to achieve better coverage and connectivity than the plain bridge test sampler.

Another idea is to sample configurations from a medial axis of the free space. That would allow the configurations to be in narrow passages and at the same time forcing them to lie as far from the obstacles as possible. Unfortunately, it is very difficult to calculate the medial axis exactly especially in complex environments. However, several methods have been proposed that can be used to generate configurations in a medial axis. These include, among others, the methods presented in [33, 95, 97].

4.1.4 Region-Based Sampling

Many PRM planners try to identify the difficult regions of the configuration space and then bias the sampling there. It is not easy to find narrow passages especially in complex environments but, nevertheless, many different techniques to do so has been developed. One is proposed in Publication II.

The method in [91] works by doing an approximate cell decomposition of the free workspace. The cells are then grouped into larger regions and the regions with narrow passages are identified with a method based on watershed segmentation [92]. Cell decomposition is used also in method that is described in [102]. That method divides the configuration space into cells, builds a localized roadmap

within cells that contain obstacles and based on that information constructs a final roadmap.

A machine learning approach to identify regions is taken in method proposed in [63]. The method works by classifying regions into different categories depending on the features extracted from it. If the region was not properly classified, the region is divided recursively into smaller regions which are then classified again. Based on this classification, each region is sampled in different manner. RESAMPL [76] is a method that divides the configuration space into slightly overlapping regions at the start and then classifies them by taking a few sample configurations from the region and inspecting whether those are free or not. The sampling is then biased towards the regions that likely contain narrow passages.

4.1.5 Other Methods

One possible way to enhance sampling in PRM planners is to combine several different sampling techniques together. A good example is the basic bridge test sampler [84] which is usually combined with a normal random sampling to achieve the better distribution of configurations. One sampler can easily generate configurations in narrow passages while the other handles the easier regions of the configuration space.

A more advanced method to combine samplers is presented in [35]. It measures the performance of each sampler it uses and tries to learn which ones are the most effective. Each sampler is associated with a probability that tells how often it is used to sample configurations. The probability is increased for those samplers that perform well and decreased for the others. Another method that combines several samplers together is described in [50]. That method also uses information of the workspace geometry to guide the sampling.

An interesting method, that chains different samplers together, is presented in [86]. The idea is that one sampler generates a configuration that is then used as an input value for the next sampler. This works for samplers that can somehow bias the sampling with the input value. The chaining of samplers makes it possible to take an advantage of the strengths of each sampler.

In Publication I, a configuration deactivation method was proposed. It aims to speed up the planning by detecting the configurations that are not useful. Those configurations are then deactivated which means that while they stay in the roadmap they are not selected as neighbors any more. The deactivation method itself is not a sampler but it can be used together with many existing sampling methods and enhance their performance.

4.2 Neighbor Selection

The basic PRM planner selects a set of neighbor configurations for each new configuration that is added to the roadmap. However, it is not trivial to decide how these neighbors should be selected and how many configurations should belong to the neighborhood. Additionally, the planner should be able to retrieve these neighbor configurations quickly from the roadmap and that may require some specific data structures.

4.2.1 Calculating Distances

The basic PRM planner tries to connect newly added configuration q to all its neighbor configurations with a local planner. The neighbors should be selected in such a way that the local planner can successfully connect them to q with a high probability and therefore, a typical PRM planner selects a set of nearest configurations as neighbors for q. This is because it is more likely that the local planner success to find a free path between configurations that are near each other than between the configurations that are very far apart.

Calculating a distance between two configurations is not very straightforward. One possibility would be to measure the volume of the workspace that is swept by the robot when it moves between two configurations. When the robot sweeps only a small volume, it is likely that the robot does not collide with the obstacles and the probability of collisions increase when the volume grows. However, if the robot is complex, it can be difficult and slow to calculate this swept-volume. Therefore, some other means to calculate the distance between configurations is usually used.

For a rigid-body robot that moves freely in a three-dimensional workspace, a commonly used method to calculate an approximate distance between two configurations is to split the configurations into two components [48]. One component represents the translation and the other rotation. For translational distance a normal Euclidean distance can be used. To calculate the rotational part, one possibility is to use the angle between the rotations as a distance.

This approximation can be calculated as follows. A distance between two points $t_1 = (x_1, y_1, z_1)$ and $t_2 = (x_2, y_2, z_2)$ can simple be defined as

$$\mathrm{dist_t}(t_1, t_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2},$$

and the distance between rotations represented as two unit quaternions $h_1 = (a_1, b_1, c_1, d_1)$ and $h_2 = (a_2, b_2, c_2, d_2)$ can be defined as

$$\mathrm{dist_r}(h_1, h_2) = \arccos(|\, a_1 a_2 + b_1 b_2 + c_1 c_2 + d_1 d_2 \,|).$$

The approximate distance between two configurations $q_1 = (t_1, h_1)$ and $q_2 = (t_2, h_2)$ can now be defined as

$$\mathrm{dist}(q_1, q_2) = w_\mathrm{t}\mathrm{dist_t}(t_1, t_2) + w_\mathrm{r}\mathrm{dist_r}(h_1, h_2),$$

where w_t and w_r are weight constants. The problem that remains is to select good values for w_t and w_r. The best values vary between different problems but in the typical case, the translational distance is more important than the rotational distance [5, 48, 100].

In case of a rigid-body robot that moves in two-dimensional configuration space, a similar approach can be used as in three-dimensional workspace. The distance can be calculated separately for translational and rotational parts and then combined together.

4.2.2 Neighborhoods

The selection of a set of neighbors is a critical part of many PRM planners. It is important that the neighborhood is large enough to ensure that the connectivity of the final roadmap is good. In that sense, the ideal case would be that all possible edges are checked with a local planner and added to the roadmap when they are free. In practice that would not be feasible because there would be $n(n-1)/2$ edges in a roadmap that has n configurations in it. To check them all for collision would dramatically slow down the construction of the roadmap. In addition, the final roadmap would require huge amounts of memory to store all the edges if n grows large. The large roadmap would also slow down many graph algorithms that may be used with the roadmap. Therefore, the neighborhood size should be limited somehow.

A common way to reduce the number of edges is to limit a neighborhood size to some predefined constant k. The difficulty is to find a good value for k because it varies between different motion planning problems. A too small value makes it difficult to get the roadmap well connected while a too large value slows down the construction phase unnecessarily. In [28], it is investigated how different values for k affect the roadmap quality. This issue is also shortly discussed in Publication V.

Another way for reducing the edges is to limit the neighborhood size by the distance. This method selects all those configurations as neighbors for a configuration q that are at most distance r away from q. The distance r is some predefined constant.

Sometimes it can be useful to reduce the neighborhood size even further. This can be done by filtering out some of the nodes from the neighborhood. One of the simplest ways to filter out nodes is to require that there should not be cycles in the roadmap. For configuration q, this is achieved by going through all the configurations in the neighborhood and calling a local planner for a neighbor only if it does not already belong to the same component as q. This can very effectively reduce the number of edges while still keeping the connectivity good because as a result, the roadmap will be a set of trees.

Another simple method to filter out nodes from the neighborhood is to do it randomly. This means that the planner tries to connect the configuration to each of

its neighbor configuration with some predefined probability. If the probability is 1, all free edges are added to the roadmap. By lowering the probability, it is possible to decrease the number of edges in the roadmap. This simple technique allows us to select a large neighborhood and add cycles to the roadmap while still keeping the number of edges in the final roadmap under control. This kind of randomized method was investigated in [62].

In [65], a more sophisticated method to filter out some unnecessary neighbors is presented. The goal is to construct such a roadmap that the paths returned by the planner are short even though the number of edges in the roadmap is kept small. The method tries to create "shortcuts" to the roadmap by identifying which edges would be useful and adding only them.

The method starts by selecting a set of the nearest neighbors for a configuration q and then goes through all those neighbors. For each neighbor q', the method tries to find the shortest path in a roadmap between q and q' and calculate its length d. If the path does not exist, d is set to ∞. The method calls a local planner for q and q' and tries to add an edge between them only if $K \cdot \text{dist}(q, q') < d$, where K is a predefined constant. Otherwise q' is filtered out. This means that an edge (q, q') is added to the roadmap only if it is short enough when compared with the existing path. The number of edges in the final roadmap can be modified by changing the value K.

These different methods to handle neighbor configurations are investigated more detailed in Publication III. In that publication, also a new method to filter out nodes from the neighborhood is presented. The goal of that method is to produce a roadmap that works well in changing environments.

4.2.3 Searching Neighbors

Because neighbor search is a frequently used operation in PRM planners, it should work quickly. Over the years, many data structures have been proposed that can enhance the neighbor search. Usually, these are not designed specifically for motion planning but nevertheless, they can still be useful.

The methods used to find the nearest neighbors can be categorized into two groups: exact methods and approximate methods. Many applications require an exact solution and also in PRM planners exact methods are often used. However, since the PRM planners are not exact algorithms themselves it is possible to use approximate neighbor search with them.

The speed is often the main reason for the use of the approximate methods instead of exact ones. The approximate methods can often substantially reduce the computational time required to retrieve neighbors. The downside is that the result is not necessarily exact. There are several efficient exact algorithms to find the nearest neighbors but they work quickly only in low-dimensional search spaces. The approximate methods are especially useful in high-dimensional spaces where

they work reasonably well [6]. In Publication IV, three different methods for the nearest neighbor search are investigated. Two tested methods are exact and one is an approximate method.

The simplest method to find k nearest neighbors for a configuration q from the roadmap is to use brute-force search. It goes through all existing configurations and calculates the distance between them and q. A list of k nearest neighbors are maintained during the search and when all configurations are handled, the list contains exact solution.

The brute-force method is very easy to be implemented but unfortunately, it is is rather slow especially when the number of nodes in the roadmap is high. However, it does not need any additional data structures to work and its efficiency does not depend on the dimension of the configuration space. Therefore, it can even outperform other exact methods in high-dimensional spaces [93].

The *kd*-tree method [10, 23] is a popular way to speed up the exact nearest neighbor search. There are different variations of the method but it is always based on the binary tree. In a typical *kd*-tree implementation, the points, which in PRM planners are the configurations from the roadmap, are stored to the nodes of the binary tree. Each node stores one point and each node also divides the space into two partitions by a plane that goes through the stored point. The plane is used to divide the remaining points into the subtrees of the node. The nearest neighbors can now be searched for quickly by using this structure because it allows to eliminate the large regions of the search space during the search.

In [100], the *kd*-tree method was investigated more thoroughly from the motion planning perspective. The results show that the *kd*-tree method can drastically increase the performance of the PRM planners when the dimension of the configuration space is low. Unfortunately, when the dimension grows, the *kd*-tree method does not perform so well. This was observed in the experiments of Publication IV, where a simple brute-force method in some cases performs better than the *kd*-tree method.

Locality-sensitive hashing [6, 29, 36, 81] is an approximate method for the nearest neighbor search. The method works quickly and it can return a good approximation with a high probability even with large datasets and with high dimensional data. It has been used in many application areas and in Publication IV it was successfully tested also with PRM planners.

As the name suggests, a hash table is an important part of methods based on locality-sensitive hashing. The table consists of several buckets which are identified with a unique hash value. In the case of PRM planners, all configurations from the roadmap are stored into the buckets. The method uses a hash function to calculate a hash value for each configuration and this value is used to determine the bucket to which each configuration belongs. The number of buckets should be considerably smaller than the number of stored configurations which means that each bucket should contain several configurations. To be useful in the nearest neighbor search, configurations that are near each other should be stored into the same bucket with

a high probability, i.e. the hash function should return the same hash value for nearby configurations.

Searching for a set of neighbors for some configuration can be accomplished by calculating a hash value for the configuration and then selecting all configurations from the bucket indicated by the hash value. If the retrieved set contains too many configurations, it is possible to reduce the set further, for example, with a brute-force search. It is also possible to use multiple hash tables to increase the probability that the nearest configurations are actually found. Each hash table must use a different hash function and configurations must be stored to all hash tables. When neighbors are searched for, configurations are retrieved from all hash tables and united together.

The difficulty is to choose a good hash function and to decide how many hash tables and buckets are needed. These issues are investigated more in Publication IV. A comparison of several hash functions can be found in [71].

4.2.4 Visibility-Based Method

A different approach for neighbor selection is used in visibility-based PRM planners [80]. In that method, there is no need to calculate distances between configurations because the neighborhood always consists of those configurations that are marked as guards in the roadmap. This means that the neighborhood is the same for all configurations. It is, however, possible that the neighborhood grows as new nodes are added to the roadmap.

The visibility-based method is based on the concept of visibility domains. For a configuration q, a visibility domain is defined as

$$V_q = \{ q' \in \mathcal{C}_{\text{free}} \mid \Delta(q, q') \subseteq \mathcal{C}_{\text{free}} \},$$

where Δ is a local planner. This means that the visibility domain is a set of all those configurations that are reachable from q with Δ. We can also say that the configuration q sees all configurations that are in V_q.

The method works by sampling configurations randomly just like the basic PRM planner. Every time a configuration q is sampled, existing guards from the roadmap are selected as neighbors and the method tries to connect q to each of these guards. After that, three things can happen:

1. configuration q is added to the roadmap as a guard if it was not connected to any of the guards, i.e., q cannot see any guards,

2. configuration q is added to the roadmap as a normal node if it was connected to at least two guards from different components, i.e., q can see multiple guards from at least two components, or

3. configuration q is discarded and not added to the roadmap if it was connected to only guards from a single component, i.e., q can see only one guard or several guards from the same component.

The last case causes the visibility method to reduce the number of nodes in the roadmap dramatically which is also the original goal behind the method.

In Publication V, a new method is presented to construct small roadmaps in terms of nodes. It is based on the ideas of the visibility-based method but it does not use guards. Instead, the neighbors are selected just like in the basic PRM planner. Experiments show that it can produce much smaller roadmaps than the visibility-based method.

4.3 Connecting Configurations

After a set of neighbor nodes is selected for a newly sampled configuration q, a local planner is used to check whether there is a free path between q and its neighbors. A typical local planner is a straight-line planner that can easily be used with various robots. It works by interpolating a path between two configurations. With rigid-body robots, this can be done by separately interpolating the translational and rotational components [48, 79]. In this thesis, all experiments were conducted by using this simple local planner.

To check whether a local path is collision-free, it is possible to discretize the local path between two configurations with some resolution and then check all these intermediate configurations for collisions. If some configuration is in collision, it means that the path is not collision-free. It should be noted that this method is not exact. If the resolution is not fine enough, it is possible that some collisions are not found. In [78], an exact approach to check the collisions of straight-line paths is presented.

More powerful local planners than the straight-line planner have also been proposed, for example, in [28, 37]. The advantage of using them is their ability to find free paths between configurations that cannot be connected with simpler planners. This is useful especially in narrow passages. Unfortunately, these powerful local planners are usually much slower than the simpler ones. Performances of different local planners are compared in [5].

Usually, the actual local paths are not stored to the roadmap. Instead, only information that two nodes can be connected with a local planner is saved and during the query phase, a local planner is used again to reconstruct these local paths. When the local planner is simple, this can be done very quickly but with more complex local planners it might take too much time especially if the query should be answered in a real-time. This problem can be circumvented by storing local paths to the roadmaps in some format.

A lazy PRM planner [12] gives a different approach for connecting nodes together. The method tries to minimize the collision checks and it builds the

roadmap assuming that all nodes and edges are collision-free. The collisions are checked at the query phase and nodes and edges that are in collision are removed from the roadmap when they are found. The lazy PRM planner moves much of the work from the learning phase to the query phase but in practice, it can be quite efficient because it calls a local planner only when needed.

4.4 Improving Paths

The basic algorithm finds the path in a query phase and returns it as is. However, the quality of the path may not always be as good as wanted. To improve the quality of paths, it is, for example, possible to apply an optional post-processing operation to the retrieved path. Another possibility is to increase the quality directly in a query phase.

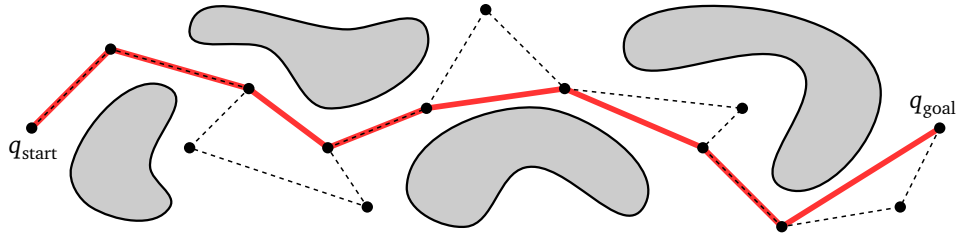
It is ambiguous to define what is a high-quality path. In some cases, the quality can be measured as a path length and usually the short paths are more desirable than long paths. However, sometimes it might be more important that the robot stays as far from obstacles as possible even if it means that the path is long. In addition, it is often important that the path is smooth and natural-looking.

One simple way to shorten a path is to select two configurations q_1 and q_2 from the path and check whether a local planner can find a free path between them. If a free path is found, it is possible to connect q_1 to q_2 with this path and discard a previous path segment that connected them. More clever methods to decrease the path length are proposed, for example, in [27, 30]. In [27], it is also investigated how the path can be modified in such a way that it lies far away from the obstacles. In Figure 4.2(a) is a simple example of how a path can be shortened.

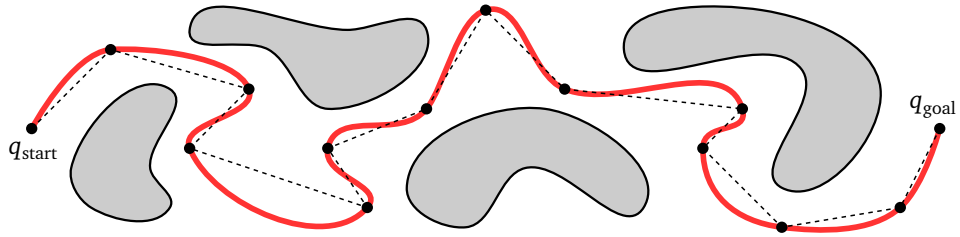
The paths can also be smoothed in such a way that the robot does not make sharp turns while moving. Different smoothing techniques is discussed, for example, in [94, 96, 103]. One example of path smoothening is shown in Figure 4.2(b). In Figure 4.2(c) there is an example of a path that has been first shortened and then smoothed.

Another method to increase the quality of paths is presented in [72]. The idea is based on the fact that even low-quality paths may contain high-quality subpaths. The method starts by generating a set of different paths, for example, by using multiple roadmaps. Then it extracts high-quality subpaths from the generated paths and finally merges these subpaths together. The quality of the result is likely to be better than the quality of any of the original paths.

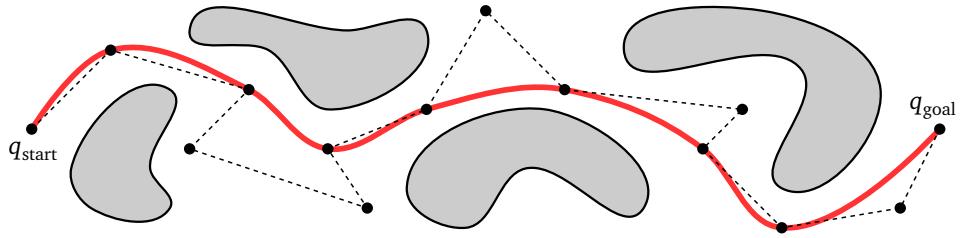
Typically, PRM planners always return exactly the same path if the same query is performed multiple times. In some applications, this is preferred but, for example, in computer games it will not look natural if all characters always move exactly the same way. Several methods have been proposed to add small variation to paths [42, 49]. This allows different characters to move in slightly different manner while still using the same roadmap.



(a) A shortened path



(b) A smoothed path



(c) A shortened and smoothed path

Figure 4.2: An example of post-processing. The dashed line shows an original path from q_{start} to q_{goal} in two-dimensional configuration space. Gray areas are obstacles. In (a), the path has been shortened and in (b), the path has been smoothened. In (c), the path has first been shortened and then smoothed.

Unfortunately, the post-processing is usually quite heavy operation. It can be a significant problem especially in real-time applications where the paths should be retrieved very quickly from the roadmap. Especially in these cases, the roadmap should be constructed in such a way that the need for post-processing is small. That would likely increase the time required to build the roadmap but on the other hand decrease the time used in post-processing.

4.5 Dynamic Environments

The basic PRM algorithm is designed for static environments where the obstacles are known in advance. The assumption is that the obstacles do not move and

therefore the roadmap can be safely constructed in the learning phase. In dynamic environments, however, it is possible that the environment changes after the roadmap has been built. As a solution for this issue, several different techniques have been proposed for PRM planners.

Typically, the roadmap must have cycles if it is used in dynamic environments. With cycles, the roadmap has many alternative paths that the robot can use. If one path becomes blocked by a moving obstacle, the robot can find some other path by using the roadmap. Several ideas to select good edges to form the cycles have been described [62, 65, 89]. One method was proposed in Publication III and in that publication it was also experimentally tested how roadmaps with cycles work in environments where obstacles can appear after the roadmap has been built.

The methods in [66, 89] assume that there can be only limited changes in the environment which means that the roadmap can be built in such a way that there is always a free path that circumvents moving obstacles. The method in [90] assumes that it is known in advance where and when the obstacles move. However, these kinds of assumptions cannot be made always and it may not be enough to just build a roadmap that has cycles. Usually, it is important to maintain information about which parts of the roadmap are free and sometimes it may also be necessary to rebuild or extend parts of the roadmap so that the robot can find a free path. These modifications must often be made in real-time so it is not an easy task.

One example of a method that enhances PRM planners to work in truly dynamic environments is proposed in [38]. It builds the roadmap based on the static obstacles like the basic PRM planner but uses ideas from a lazy PRM planner [12] to avoid checking unnecessary edges for possible collisions in a query phase. If the found path contains blocked edges, the method tries to reconnect these parts of the path with a RRT-based [52, 56] technique. If that does not help, the method tries to create totally new configurations to the roadmap. There are also other similar methods, for example [9, 101], that try to react to moving obstacles when the robot is moving on the path. Some methods, like the ones described in [41, 83], try to constantly adapt to the changes in the environment and modify the roadmap accordingly.

CHAPTER 5

Overview of Publications and Results

This chapter contains an overview of all five publications. Contributions of each publication are described as well as a summary of the results from the experiments that were conducted.

All publications contain experimental tests on simulated environments. The software that was used in these experiments was implemented in C++ by the author of the thesis. All tested methods were implemented into the same software framework to ensure that the results are comparable with each other. In the experiments, the robots and obstacles were rigid bodies that were represented as triangle meshes.

5.1 Publication I: A configuration deactivation algorithm for boosting probabilistic roadmap planning of robots

In this publication, a method to speed up the construction of the roadmap is proposed. The method extends the basic PRM planner. When a new configuration is sampled and a set of neighbor configurations are selected for it, the method starts to count how many times the new node is tried to be connected with a node that is already in the same component with it. The method requires a parameter that defines a maximum allowed number of these connections. If the allowed connections are exceeded, the new configuration is deactivated. It means that the node is no longer selected as a neighbor for any new configuration.

It should be noted that the proposed deactivation method can be used with many existing sampling methods to enhance their performance. In experiments, the method was tested with the normal random sampling, the Halton sampling, the Gaussian sampling and the bridge test sampling. These same sampling methods were also tested with the basic PRM planner and the visibility-based method. The

experiments were conducted in three different environments with a rigid-body robot. In each test, the roadmap was built until a predefined query could be solved. A required time to build the roadmap was measured as well as the number of nodes in the final roadmap.

The results show that the deactivation method can effectively decrease the construction time and that the visibility method takes the longest time in all test cases. The deactivation method works best when combined with the Gaussian sampler or the bridge test sampler. When the roadmap sizes are compared, it can be seen that the visibility method produces the smallest roadmaps. However, the deactivation method can also reduce the roadmap sizes considerably when compared with the basic PRM method.

5.2 Publiction II: A connectivity-based method for enhancing sampling in probabilistic roadmap planners

This publication presents a novel connectivity-based PRM planner that tries to identify narrow passages and other difficult regions in order to enhance the planning. The goal is to recognize easy regions from the difficult regions during the roadmap construction. After some region has been determined to be easy, the planner starts to ignore it. This allows the planner to concentrate on the sampling towards the difficult regions.

The connectivity-based method works by dividing the workspace into regions at the beginning. This can be done, for example, by using the Halton sequence to generate configurations and selecting them as centroids for the regions. By defining a radius for each centroid, it is possible to think regions as spheres. The radiuses should be selected in such a way that the regions slightly overlap each other.

During the roadmap construction, the regions are gone through one by one repeatedly. At each iteration, one configuration is generated for the region. This can be done by utilizing the information of the centroid and the radius. If the generated configuration is free, it is added to the roadmap. If the number of free configurations in the region is larger than a predefined parameter, the method checks whether the configurations in the region belong to the same component in the roadmap. If they do, the region is identified to be an easy region and the method will not generate any samples for that region any more.

In experiments, the new method was compared with the Gaussian sampling and the bridge test sampling. The methods were tested in three different three-dimensional environments where the robot was a rigid body. For each environment, there was a predetermined query and the start and goal configurations of the query were added to the roadmap at the beginning. The roadmap was then constructed until the roadmap was able to connect these configurations together. All tests were

run 100 times to minimize the effect of the random nature of the PRM planners. With the connectivity-based method, also regions were randomly generated at each test run. The results show that the connectivity-based method is clearly faster than the Gaussian sampling or the bridge test sampling. The method can also reduce the size of roadmaps.

5.3 Publication III: Using probabilistic roadmaps in changing environments

The goal of the publication was to explore the usability of PRM planners in changing environments where new obstacles can appear after the roadmap has been built and block some of previously free edges or nodes. The publication concentrated on the cycles which are an essential part of roadmaps in changing environments because they can provide several alternative paths for a robot to move. With a well-built roadmap, the robot should find a free path even if a few edges have become blocked. This kind of roadmap can then be used as a base with methods that can also handle truly dynamic environments where the robot must react in real-time to the changes.

A new distance-based method to choose which edges to add to the roadmap is also presented in the publication. The method tries to ensure that all nodes in the roadmap have multiple edges and that these edges are distributed evenly in different directions. The rationale behind this is that it is easy to block one edge but if there are many edges going in different directions it is much harder to block all of them.

In the publication, the distance-based method was experimentally tested in several simulated environments. The method was compared with a random method [62] and with a useful edges method [65] that were discussed in Section 4.2.2. In addition, a basic PRM planner that constructed a roadmap without cycles was also tested in the same environments.

In experiments, the roadmap was built in such a way that only static obstacles were present in an environment. After that, the environment was changed by adding new obstacles in five steps. At each step, it was checked whether the roadmap could still be used to solve a predetermined query. This was repeated 1000 times for each method in each environment with several different predefined roadmap sizes. A success rate for solving the query was measured along with the length of the obtained path and the time required to build the roadmap.

The test results show that the roadmap built without cycles is not suitable for changing environments at all. The random method was able to achieve little better success rates but it was not a good solution either. The best success rates were achieved by using the useful edges method and the distance-based method which also produced the shortest paths. The random method was the fastest to construct

the roadmap and the distance-based method was almost equally fast. The useful edges method was the slowest.

5.4 Publication IV: Speeding up probabilistic roadmap planners with locality-sensitive hashing

This publication compares three different methods to find the nearest neighbors from the roadmap. The tested methods were an exact brute-force method, an exact *kd*-tree method [10] and an approximate method based on locality-sensitive hashing (LSH) [29, 36]. The goal was to investigate how well and quickly the LSH approach would work with PRM planners. Additionally, the comparison would also show whether a quality of the roadmap decreases when an approximate method is used instead of an exact method. It should also be noted that, to the author's knowledge, the locality-sensitive hashing has not been used with PRM planners previously in the literature.

In the experiments, three environments were used and in each environment the methods were tested with three robots. The robots were composed of a different number of rigid bodies which means that for each robot the configuration space had a different dimension. The experiments were also conducted with different neighborhood sizes and the method based on locality-sensitive hashing was tested with several parameter combinations.

The results show that the approximate LSH works very well with PRM methods. It can speed up the construction of the roadmap considerably when compared with exact methods. The *kd*-tree method can work slightly quicker in low-dimensional configuration spaces but the results also show that the performance of a *kd*-tree method decreases rapidly as the dimension grows. At some point, it will actually start to perform worse than the brute-force method.

Besides the construction time, also the quality of the final roadmap was measured. This was done by predefining a query that the roadmap was supposed to solve. Every time a new node was added to the roadmap, it was checked whether the query could be solved. When the query became solvable, a roadmap size, a number of components and the length of the found path was measured. Each test was repeated 1000 times and average values of the measurements were calculated. The results obtained with different methods were then compared with each other. According to these comparisons, the LSH method was able to achieve the same quality as the exact methods.

The problem with a locality-sensitive hashing is its dependence on parameters. The tested variation required two parameters and choosing the best values can sometimes be tricky. However, the results show that the LSH method works quite well even when the parameters are not the best ones.

5.5 Publication V: How to construct small probabilistic roadmaps with a good coverage?

In this publication, the goal was to investigate how the roadmap could be built in such a way that the number of nodes is kept small while still keeping the coverage and the connectivity at good level. A novel method is proposed in the publication and it tries to reduce the size of the roadmap. This neighborhood-based method combines ideas from the basic PRM planner and from the visibility-based PRM planner [80].

As was discussed in Section 4.2.4, the visibility-based planner keeps a list of guard configurations that are used as neighbors. When a new node is added to the roadmap, the planner tries to connect it to all guards. The problem with this idea is that it ignores all other configurations besides the guards. Another problem is that sometimes the guards may be in such places that it is difficult to connect them together.

The new neighborhood-based method proposed in the publication tries to remove unnecessary nodes in a similar manner as the visibility-based method. To overcome problems that the visibility-based method has, the new method treats all nodes in the roadmap in an equal way and selects the neighbors just like the basic PRM planner. This allows the method to connect newly added configurations to all nearby configurations and not just to the guards. As the experiments show, this can reduce the size of the roadmap considerably.

The experiments conducted in the publication compared the basic PRM planner, the visibility-based planner and the neighborhood-based method with each other. All these planners were tested with the uniform sampling and with the bridge test sampling. The experiments show that the visibility-based method certainly decreases the size of the roadmap when compared with the basic PRM planner but the neighborhood-based method can construct even smaller roadmaps than the visibility-based method. The neighborhood-based method also worked very quickly when compared with the visibility-based method.

It should also be noted that the proposed method does not need any additional parameters when compared with the basic PRM planner. The neighborhood size is the only critical parameter and as was shown in the publication, choosing a good value for it is quite easy and does not need any exhausting fine-tuning. The method also works well with a simple uniform sampling method which does not need any parameters either.

CHAPTER 6

Conclusion

In motion planning, a goal is to find a path for a robot between two locations among obstacles. The robot must not collide with obstacles while moving. The motion planning problem originates from robotics where it is a crucial part of autonomous robots. However, there are many other application areas as well. A motion planning can be used for example in computer games and other virtual environments to guide characters. It can also be used, for example, in molecular simulations.

It is difficult to solve the motion planning problem. There are exact methods but generally they work only in low-dimensional configuration spaces quickly enough to be practical. Different kinds of approximate algorithms have been proposed over the years to tackle higher-dimensional problems. These include probabilistic roadmap planners which usually have a good performance.

PRM planners do not try to build an exact representation of the free configuration space but instead they construct a graph called a roadmap that the robot can use to move. The roadmap is built by taking samples of the free configuration space and adding them as nodes to the roadmap. The nodes can be connected together with edges if there exists a free path between them.

The basic PRM planner can handle quite well the difficult environments but sometimes it is possible to enhance the basic planner to work even better. For example, the environment may contain narrow passages where it is difficult for a robot to move freely. If a PRM planner generates all samples uniformly at random, it can take a great number of nodes to construct the roadmap through the narrow passages. With more advanced sampling methods, it is possible to bias sampling to the narrow passages.

This thesis concentrated on studying PRM planners and trying to increase their performance in various ways. The thesis consists of five publications that each enhanced some parts of the basic PRM planner. Publication I proposed a method that can be used to speed up many existing configuration sampling methods. Publication

II concentrated on configuration sampling and a new region-based method was proposed for sampling. Publication III studied how PRM planners work in changing environments. Publication IV compared different data structures that can be used to quickly retrieve the nearest neighbors for a configuration. Publication V proposed a simple method that can decrease the size of the roadmap considerably.

The methods presented in the publications were experimentally tested in simulated environments. The results were good and suggested that the proposed methods can indeed be useful in practice. Currently, one of the most prominent areas to use them are computer games and different simulated applications. One challenge, and an idea for future research, is to investigate how to get PRM planners work reliably with real robots.

In future, it would also be interesting to investigate how the ideas presented in the publications would work together. For example, in Publication III it was studied how a roadmap can be built in such a way that it works well in changing environments and in Publication V, a method to reduce the number of nodes in roadmaps was presented. By combining these ideas, it could be possible to build roadmaps that have a small number of nodes but which still would work pretty well in changing environments.

Another idea would be to combine ideas from Publication II and Publication IV together. The method in Publication II enhanced sampling and the method in Publication IV speeded up the neighbor selection with a locality-sensitive hashing. Both methods divided the configurations of the roadmap into different groups and this similarity could probably be exploited.

Bibliography

[1] E. U. Acar, H. Choset, A. A. Rizzi, P. N. Atkar, and D. Hull. Morse decompositions for coverage tasks. *The International Journal of Robotics Research*, 21(4):331–344, 2002.

[2] I. Al-Bluwi, T. Siméon, and J. Cortés. Motion planning algorithms for molecular simulations: A survey. *Computer Science Review*, 6(4):125–143, 2012.

[3] N. M. Amato. Equipping CAD/CAM systems with geometric intelligence. *ACM Computing Surveys*, 28(4es), 1996.

[4] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In P. K. Agarwal, L. K. Kavraki, and M. T. Mason, editors, *Robotics: The Algorithmic Perspective*, pages 156–168. A. K. Peters, Natick, MA, 1998.

[5] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Transactions on Robotics and Automation*, 16(4):442–447, 2000.

[6] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[7] S. Bandi and D. Thalmann. Path finding for human motion in virtual environments. *Computational Geometry*, 15(1–3):103–127, 2000.

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

[9] K. Belghith, F. Kabanza, L. Hartman, and R. Nkambou. Anytime dynamic path-planning with flexible probabilistic roadmaps. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2372–2377, 2006.

[10] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[11] P. Bessiere, E. Mazer, and J.-M. Ahuactzin. Planning in continuous space with forbidden regions: The Ariadne's clew algorithm. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 39–47. A. K. Peters, Wellesley, MA, 1995.

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

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

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

[15] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[16] M. Cavazza. AI in computer games: Survey and perspectives. *Virtual Reality*, 5(4):223–235, 2000.

[17] H. Chang. A new technique to handle local minimum for imperfect potential field based motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 108–112, 1996.

[18] B. Chazelle. Approximation and decomposition of shapes. In J. T. Schwartz and C.-K. Yap, editors, *Advances in Robotics: Algorithmic and Geometric Aspects of Robotics*, pages 145–185. Lawrence Erlbaum Associates, Hillsdale, NJ, 1987.

[19] H. Choset. Coverage for robotics – a survey of recent results. *Annals of Mathematics and Artificial Intelligence*, 31(1–4):113–126, 2001.

[20] 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, Cambridge, MA, 2005.

[21] N. Dadkhah and B. Mettler. Survey of motion planning literature in the presence of uncertainty: Considerations for UAV guidance. *Journal of Intelligent & Robotic Systems*, 65(1–4):233–246, 2012.

[22] K. Daniel, A. Nash, S. Koenig, and A. Felner. Theta*: Any-angle path planning on grids. *Journal of Artificial Intelligence Research*, 39:533–579, 2010.

[23] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Ed.* Springer-Verlag, Berlin, 2000.

[24] M. Elbanhawi. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[25] D. Ferguson and A. Stentz. Using interpolation to improve path planning: The field D* algorithm. *Journal of Field Robotics*, 23(2):79–101, 2006.

[26] R. Geraerts and M. H. Overmars. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems*, 54(2):165–173, 2006.

[27] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26(8):845–863, 2007.

[28] R. Geraerts and M. H. Overmars. Reachability-based analysis for probabilistic roadmap planners. *Robotics and Autonomous Systems*, 55(11):824–836, 2007.

[29] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of International Conference on Very Large Data Bases*, pages 518–529, 1999.

[30] R. Guernane and N. Achour. Generating optimized paths for motion planning. *Robotics and Autonomous Systems*, 59(10):789–800, 2011.

[31] J. H. Halton and G. B. Smith. Algorithm 247: Radical-inverse quasi-random point sequence. *Communications of the ACM*, 7(12):701–702, 1964.

[32] C. Hofner and G. Schmidt. Path planning and guidance techniques for an autonomous mobile cleaning robot. *Robotics and Autonomous Systems*, 14(2–3):199–212, 1995.

[33] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1408–1413, 2000.

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

[35] D. Hsu, G. Sánchez-Ante, and Z. Sun. Hybrid PRM sampling with a cost-sensitive adaptive strategy. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3874–3880, 2005.

[36] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of ACM Symposium of Theory of Computing*, pages 604–613, 1998.

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

[38] L. Jaillet and T. Siméon. A PRM-based motion planner for dynamically changing environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1606–1611, 2004.

[39] N. Jaklin, A. Cook IV, and R. Geraerts. Real-time path planning in heterogeneous environments. *Computer Animation and Virtual Worlds*, 24(3–4):285–295, 2013.

[40] P. Jiménez, F. Thomas, and C. Torras. Collision detection algorithms for motion planning. In J.-P. Laumond, editor, *Robot Motion Planning and Control*, pages 305–343. Springer, Berlin, 1998.

[41] M. Kallmann and M. Mataric. Motion planning using dynamic roadmaps. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4399–4404, 2004.

[42] I. Karamouzas and M. H. Overmars. Adding variation to path planning. *Computer Animation and Virtual Worlds*, 19(3–4):283–293, 2008.

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

[44] 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(4):566–580, 1996.

[45] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[46] L. Kocis and W. J. Whiten. Computational investigations of low-discrepancy sequences. *ACM Transactions on Mathematical Software*, 23(2):266–294, 1997.

[47] S. Kockara, T. Halic, K. Iqbal, C. Bayrak, and R. Rowe. Collision detection: A survey. In *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, pages 4046–4051, 2007.

[48] J. J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 3993–3998, 2004.

[49] A. Kumar and A. Ojha. Natural path planning using wavelet noise in static environment. *Computer Animation and Virtual Worlds*, 24(1):17–24, 2013.

[50] H. Kurniawati and D. Hsu. Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning. In S. Akella, N. M. Amato, W. H. Huang, and B. Mishra, editors, *Algorithmic Foundations of Robotics VII*, pages 35–51. Springer, Berlin, 2008.

[51] J.-C. Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, Norwell, MA, 1991.

[52] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report TR 98-11, Computer Science Department, Iowa State University, 1998.

[53] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[54] S. M. LaValle and M. S. Branicky. On the relationship between classical grid search and probabilistic roadmaps. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. Hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 59–76. Springer, Berlin, 2004.

[55] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001.

[56] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A. K. Peters, Natick, MA, 2001.

[57] R. Lawrence and V. Bulitko. Database-driven real-time heuristic search in video-game pathfinding. *IEEE Transactions on Computational Intelligence and AI in Games*, 5(3):227–241, 2013.

[58] S. R. Lindemann and S. M. LaValle. Incremental low-discrepancy lattice methods for motion planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 2920–2927, 2003.

[59] T. Lopez, F. Lamarche, and T.-Y. Li. Space-time planning in changing environments: using dynamic objects for accessibility. *Computer Animation and Virtual Worlds*, 23(2):87–99, 2012.

[60] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computers*, C-32(2):108–120, 1983.

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

[62] T. McMahon, S. Jacobs, B. Boyd, L. Tapia, and N. M. Amato. Local randomization in neighbor selection improves PRM roadmap quality. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4441–4448, 2012.

[63] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In M. Erdmann, M. Overmars, D. Hsu, and F. van der Stappen, editors, *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, Berlin, 2005.

[64] D. Nieuwenhuisen, A. Kamphuis, and M. H. Overmars. High quality navigation in computer games. *Science of Computer Programming*, 67(1):91–104, 2007.

[65] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 446–452, 2004.

[66] D. Nieuwenhuisen, J. van den Berg, and M. Overmars. Efficient path planning in changing environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3295–3301, 2007.

[67] S. B. Niku. *Introduction to Robotics: Analysis, Control, Applications, 2nd Ed.* John Wiley & Sons, 2011.

[68] M. H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-93, Department of Computer Science, Utrecht University, 1992.

[69] M. H. Overmars and P. Švestka. A probabilistic learning approach to motion planning. In K. Goldberg, D. Halperin, J.-C. Latombe, and R. Wilson, editors, *Algorithmic Foundations of Robotics*, pages 19–37. A. K. Peters, Wellesley, MA, 1995.

[70] X. Pan, C. S. Han, and K. H. Law. Using motion planning to determine the existence of an accessible route in a CAD environment. *Assistive Technology*, 22(1):32–45, 2010.

[71] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: A comparison of hash funtion types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010.

[72] B. Raveh, A. Enosh, and D. Halperin. A little more, a lot better: Improving path quality by a path-merging algorithm. *IEEE Transactions on Robotics*, 27(2):365–371, 2011.

[73] M. Reggiani, M. Mazzoli, and S. Caselli. An experimental evaluation of collision detection packages for robot motion planning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2329–2334, 2002.

[74] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[75] E. Rimon and D. E. Koditschek. Exact robot navigation using artificial potential fields. *IEEE Transactions on Robotics and Automation*, 8(5):501–518, 1992.

[76] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato. RESAMPL: A region-sensitive adaptive motion planner. In S. Akella, N. M. Amato, W. H. Huang, and B. Mishra, editors, *Algorithmic Foundations of Robotics VII*, pages 285–300. Springer, Berlin, 2008.

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

[78] F. Schwarzer, M. Saha, and J.-C. Latombe. Exact collision checking of robot paths. In J.-D. Boissonnat, J. Burdick, K. Goldberg, and S. hutchinson, editors, *Algorithmic Foundations of Robotics V*, pages 25–42. Springer, Berlin, 2004.

[79] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3):245–254, 1985.

[80] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

[81] M. Slaney and M. Casey. Locality-sensitive hashing for finding nearest neighbors. *IEEE Signal Processing Magazine*, 25(2):128–131, 2008.

[82] J. Smed and H. Hakonen. *Algorithms and Networking for Computer Games*. John Wiley & Sons, Chichester, U.K., 2006.

[83] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha. Real-time navigation of independent agents using adaptive roadmaps. In *Proceedings of ACM Symposium on Virtual Reality Software and Technology*, pages 99–106, 2007.

[84] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. H. Reif. Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, 21(6):1105–1115, 2005.

[85] L. Tapia, S. Thomas, and N. M. Amato. A motion planning approach to studying molecular motions. *Communications in Information and Systems*, 10(1):53–68, 2010.

[86] S. Thomas, M. Morales, X. Tang, and N. M. Amato. Biasing samplers to improve motion planning performance. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1625–1630, 2007.

[87] S. Thomas, G. Song, and N. M. Amato. Protein folding by motion planning. *Physical Biology*, 2(4):S148–S155, 2005.

[88] K. I. Tsianos, I. A. Sucan, and L. E. Kavraki. Sampling-based robot motion planning: Towards realistic applications. *Computer Science Review*, 1(1):2–11, 2007.

[89] J. P. van den Berg, D. Nieuwenhuisen, L. Jaillet, and M. H. Overmars. Creating robust roadmaps for motion planning in changing environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2415–2421, 2005.

[90] J. P. van den Berg and M. H. Overmars. Roadmap-based motion planning in dynamic environments. *IEEE Transactions on Robotics*, 21(5):885–897, 2005.

[91] J. P. van den Berg and M. H. Overmars. Using workspace information as a guide to non-uniform sampling in probabilistic roadmap planners. *The International Journal of Robotics Research*, 24(12):1055–1071, 2005.

[92] L. Vincent and P. Soille. Watersheds in digital spaces: An efficient algorithm based on immersion simulations. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 13(6):583–598, 1991.

[93] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of International Conference on Very Large Data Bases*, pages 194–205, 1998.

[94] R. Wein, J. van den Berg, and D. Halperin. Planning high-quality paths and corridors amidst obstacles. *The International Journal of Robotics Research*, 27(11–12):1213–1231, 2008.

[95] 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 *Proceedings of IEEE International Conference on Robotics and Automation*, pages 1024–1031, 1999.

[96] K. Yang and S. Sukkarieh. An analytical continuous-curvature path-smoothing algorithm. *IEEE Transactions on Robotics*, 26(3):561–568, 2010.

[97] Y. Yang and O. Brock. Adapting the sampling distribution in PRM planners based on an approximated medial axis. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 4405–4410, 2004.

[98] P. Yap. Grid-based path-finding. In R. Cohen and B. Spencer, editors, *Advances in Artificial Intelligence*, pages 44–55. Springer, Berlin, 2002.

[99] H.-Y. Yeh, S. Thomas, D. Eppstein, and N. M. Amato. UOBPRM: A uniformly distributed obstacle-based PRM. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2655–2662, 2012.

[100] A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.

[101] E. Yoshida and F. Kanehiro. Reactive robot motion using path replanning and deformation. In *Proceedings of IEEE International Conference on Robotics and Automation*, pages 5456–5462, 2011.

[102] L. Zhang, Y. J. Kim, and D. Manocha. A hybrid approach for complete motion planning. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 7–14, 2007.

[103] F. Zhou, B. Song, and G. Tian. Bézier curve based smooth path planning for mobile robot. *Journal of Information and Computational Science*, 8(12):2441–2450, 2011.

Publication I

A configuration deactivation algorithm for boosting probabilistic roadmap planning of robots

Mika T. Rantanen and Martti Juhola

International Journal of Automation and Computing 9(2), April 2012, 155-164
DOI: 10.1007/s11633-012-0628-2

A Configuration Deactivation Algorithm for Boosting Probabilistic Roadmap Planning of Robots

Mika T. Rantanen Martti Juhola
Computer Sciences, School of Information Sciences, University of Tampere, Finland

Abstract: We present a method to improve the execution time used to build the roadmap in probabilistic roadmap planners. Our method intelligently deactivates some of the configurations during the learning phase and allows the planner to concentrate on those configurations that are most likely going to be useful when building the roadmap. The method can be used with many of the existing sampling algorithms. We ran tests with four simulated robot problems typical in robotics literature. The sampling methods applied were purely random, using Halton numbers, Gaussian distribution, and bridge test technique. In our tests, the deactivation method clearly improved the execution times. Compared with pure random selections, the deactivation method also significantly decreased the size of the roadmap, which is a useful property to simplify roadmap planning tasks.

Keywords: Probabilistic roadmaps, motion planning, collision avoidance, sampling algorithms, robotics.

1 Introduction

Probabilistic roadmap algorithms are seen as an important means in robotics and related areas, such as computer animation, games and virtual reality to orientate and move a robot or an autonomous unit in a two- or three-dimensional space and to avoid collision with obstacles[1, 2]. Probabilistic roadmap algorithms investigated since the 1990s[3, 4] are useful and important because complicated routes can quickly be solved. Solving a roadmap planning problem in an exact way is computationally very difficult. Thus, Reif[5] showed it to be generally PSPACE hard, and Canny[6] showed it to be PSPACE complete.

Let us assume that a robot moves in a work space W equal to $\mathbf{R}^2$ or $\mathbf{R}^3$, where there is also a set B of obstacles. A configuration q of a robot is a set of d parameters used to define a place and orientation[7] which belongs to a configuration space C of all possible configurations. Thus, a configuration is seen as a point in d-dimensional space. Quaternions can be applied to rotate a robot while moving this[8] as usual.

Let B_i be an obstacle in work space W. The corresponding obstacle CB_i in the configuration space includes the set of configurations where a robot A would collide B_i in W, i.e., some configurations from the set of all configurations $A(q)$ and B_i would overlap:

$$CB_i = \{\, q \in C \mid A(q) \cap B_i \neq \varnothing \,\}. \tag{1}$$

The free configuration space is now:

$$C_{\text{free}} = C - \bigcup_i CB_i. \tag{2}$$

A path τ defined in C is a route or roadmap between some configurations q_1 and q_2. The task of a roadmap planning problem is to find a continuous function

$$\tau : [0, 1] \rightarrow C_{\text{free}} \tag{3}$$

where $\tau(0) = q_1$ and $\tau(1) = q_2$.

Manuscript received April 5, 2011; revised July 14, 2011

There are several methods to solve a path planning problem[9]. For example, these can be divided into potential field[10], cell decomposition[7, 11, 12] and roadmap[3] methods. In addition, they can be either exact or approximate, but we were interested in developing the latter here. A complete roadmap covers the whole configuration space. To compute such a roadmap is very difficult and slow, especially if the configuration is complicated and of a high dimension. Therefore, incomplete roadmaps are computed in practice by sampling configurations. This way, probabilistic roadmap algorithms are an essential area in robotics.

2 Baseline form of probabilistic roadmap methods

2.1 Premises of baseline algorithm

We present a baseline for probabilistic algorithms. It is a generalization, based on the algorithms of [1, 3] and includes all parts needed virtually by all probabilistic roadmap algorithms. At first, a random roadmap is constructed. This is called learning phase. Thereafter, it is used to solve individual roadmap planning problems which is called query phase.

A roadmap to be constructed is given as a graph $G = (V, E)$ with nodes V of free configurations and edges E. Configurations are randomly chosen from the free configuration space C_{free}. Every edge maps a possible free path between two different configurations along with (3). To compute a path, a local planner Δ is employed. It is a function of q and q' and returns a free path which unites these configurations. If such a path exists, $\Delta(q, q')$ connects q and q'. The Algorithm 1 consists of the roadmap calculation. Nevertheless, some of its details can be implemented in various ways as described later. Fig. 1 shows the flowchart of the Algorithm 1.

Algorithm 1. Baseline algorithm for constructing the roadmap.
Output: Roadmap $G = (V, E)$.

1: $V \leftarrow \varnothing$
2: $E \leftarrow \varnothing$
3: **repeat**
4: **repeat**
5: $q \leftarrow$ random configuration from C
6: **until** q is from C_{free}
7: $V \leftarrow V \cup \{q\}$
8: **until** there are enough configurations in V
9: **for all** q in V **do**
10: $N_q \leftarrow$ neighboring configurations of configuration q from V
11: **for all** q' in N_q **do**
12: **if** q and q' are not in the same component of the roadmap **then**
13: **if** local planner Δ finds a path between q and q' **then**
14: $E \leftarrow E \cup \{(q, q')\}$
15: **end if**
16: **end if**
17: **end for**
18: **end for**
19: **return** G

At rows 1–8 of Algorithm 1, the roadmap is initialized and free configurations are added to it. The main loop is at rows 9–18 and in that loop the algorithm tries to connect the roadmap configurations with free paths. The loop goes over each configuration q choosing a set of neighbor configurations N_q. Local planner Δ searches for a path from q to neighbor configurations q'. If a path exists, edge (q, q') is added. In row 5 a random configuration is chosen. Usually, some heuristic technique is applied here[13−16]. Typically, these favor to choose more configurations from complicated areas of C, e.g. from narrow passages between obstacles. Row 6 explores whether a configuration is in C_{free}. Here a separate collision detection algorithm has to be utilized. There are several such algorithms[17] which depend on the type of a robot. Row 8 checks whether there are enough configurations to cover C_{free}. This number is ordinarily determined at the beginning. Row 10 selects neighbor configurations on the basis of distance. Commonly, the nearest neighbors are favored because it is easy to find a path between near configurations. Local planner Δ in row 13 is important, because it often takes most of execution time[13].

It should be noted that the configuration sampling at rows 3–8 can be integrated into the main loop. This can be done, for example, by sampling one free configuration and adding it to the roadmap in each iteration of the main loop. The execution of the main loop could then stop when there are enough configurations in the roadmap.

Since the query phase is simple and fairly similar to all probabilistic roadmap algorithms, we do not consider it in detail. In the beginning, start and end configurations q_{start} and q_{end} are connected to a roadmap if they are not yet within it. After that, some path is searched for between them. Since a roadmap is presented as a graph, a path is found by using some suitable shortest path method such as Dijkstra′s algorithm which requires the execution time of $O(|V|^2)$ applying a linear list or $O((|E| + |V|) \log |V|)$ with a priority queue to store distances between nodes of a graph[18]. In general, the efficiency of query phase depends on a roadmap formed in learning phase. If there are lots of configurations and edges between these, roadmap searching

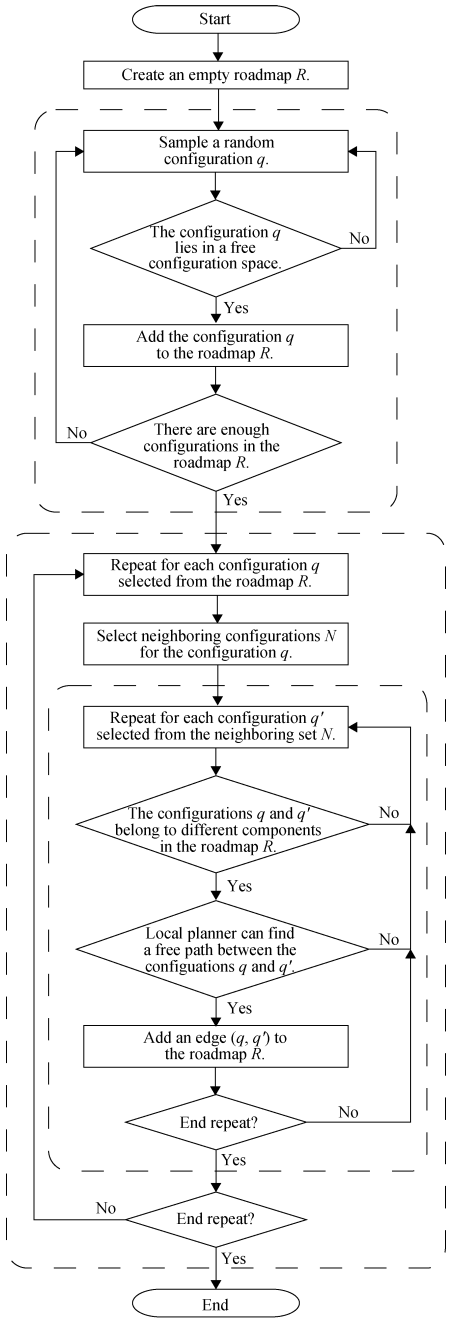


Fig. 1 The flowchart of the baseline algorithm

can be difficult and slow. Searching in a small roadmap can be fast.

2.2 Implementation of baseline algorithm

The baseline algorithm is used in a three-dimensional work space, where a location is defined with three coordinates. The minima and maxima of these coordinates are defined. The orientation of a robot is defined with a quarternion, i.e., with three independent random numbers[19, 20]. After forming a random configuration, it is checked whether it is in free configuration space C_{free}. This is performed with a collision detection algorithm given as program components, for example, in libraries PQP[21], SWIFT++[22], SOLID[23, 24], V-CLIP[25] and V-COLLIDE[26]. A collision detection algorithm only expresses whether a robot collides an object while being in some configuration. Thus, it can be used as a stand-alone program component. Random sampling of configurations is simple and effective in creating roadmaps for straightforward roadmap planning problems[27, 28].

To select neighbor configurations, a distance is computed for configurations $q_1 = (x_1, y_1, z_1, h_1)$ and $q_2 = (x_2, y_2, z_2, h_2)$, ($x_i$, y_i, z_i are reals, and h_i, $i = 1, 2$, quarternions) according to the function:

$$\begin{aligned}&\operatorname{dist}(q_1, q_2) = \\ &\quad w_{\text{t}}\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2} + \\ &\quad w_{\text{r}} \min(\arccos(h_1 \cdot h_2), \arccos(h_1 \cdot -h_2))\end{aligned} \tag{4}$$

where weights w_{t} and w_{r} depend on an algorithm, but the former is recommended to be larger[29].

For a new configuration, k neighbor configurations where number k is constant and set by default are searched for. If there are fewer than k such configurations, all these are taken. After selection the new configuration is connected with the local planner. Nevertheless, some selected neighbor configurations may already be within the graph component of a roadmap. Thus, these configurations are not connected anew.

Union-find data structure is a useful tool to clarify whether certain configurations are located in the same graph component. Its precise definition is given in [18]. Its basic idea is that every configuration added to a roadmap belongs to some set. When the local planner finds a path between two configurations p and q of a roadmap, the sets including p and q are united. Thus, the configurations are in the same set if there is a path between them in a roadmap, i.e., they belong to the same component.

For the baseline algorithm, our local planner is deterministic and symmetric, i.e., for the latter $\Delta(q_1, q_2) = \Delta(q_2, q_1)$. The local planner associates two configurations linearly in the configuration space and examines if a robot collided in their in-between area. Two simplest ways to implement this are either to move iteratively from q_1 via intermediate locations to q_2[3] or to traverse intermediate locations by binary search. The latter is faster with its logarithmic search time if the number of intermediate locations is not small.

3 Improvement techniques for baseline algorithm

In this section we first describe briefly some useful improvements that are used to improve the efficiency of the baseline algorithm. The objective was to use these as either alternative or joint parts with a novel deactivation method which we ultimately present.

Gaussian sampling[14] is based on Gaussian blurring exploited in image processing in order to sample appropriate configurations. It tries to sample free configurations close to the borders of obstacles. The dimension of normal distribution applied is equal to d of the configuration space. Its standard deviation is given as a parameter value whereas its expectation is the origin of the configuration space. We followed the algorithm of [14]. A normal distribution could then be elaborated with various ways as investigated in [30, 31].

Bridge test sampling attempts to particularly sample in narrow passages or in the most difficult parts of the configuration space[32]. When two configurations q_1 and q_2 are encountered so that both are inside obstacles, the area between them is explored. If there is configuration q_3 between them and being also in a free space, q_3 is added to a roadmap. This technique makes use of the density function of some probability distribution to assign the length and orientation of a "bridge". When normal distribution is again exploited, this is reminiscent of Gaussian sampling.

Quasi random sampling differs from the former heuristics which were pseudo random, in other words, attempts to simulate genuine randomness by applying a mathematical algorithm. Quasi randomness does not even aim at producing random numbers. Quasi random number generators are more regular than pseudo random numbers. They are distributed more uniformly than (pseudo) random numbers. This is a very useful feature for applications in which a uniform distribution is more crucial than randomness. This is often true for roadmap algorithms[33]. However, most probabilistic roadmap algorithms still apply pseudo randomness even if this is without any reason. A van der Corput number sequence is used to form quasi random numbers. Some integer is expressed as a sum of series of a given base. The coefficients are written in the reversed order and the decimal point is added to the beginning of the new sequence. For instance, integer 13 is in its binary form (base 2) 1101_2 and its reversed sequence is 0.1011_2 corresponding to decimal 0.6875. Occurrence of van der Corput sequences are uniformly distributed in interval $[0, 1)$. Generalizing one-dimensional van der Corput numbers to d-dimensional Halton numbers[34] we obtain d tuples, whose elements correspond to each dimension, the bases of which are different indivisible integers, usually first d primes.

Using visibility domains can improve probabilistic roadmap algorithms. The method[35] aims at keeping the number of configurations selected to a roadmap as small as possible, but, at the same time, to sufficiently cover the free configuration space. The visibility domain of configuration q is defined as follows:

$$V_q = \{\, q' \in C_{\text{free}} \mid \Delta(q, q') \subset C_{\text{free}} \,\}. \tag{5}$$

Configuration q is then called a guard of visibility do-

main which includes all such free configurations to which the local planner finds a path from q. When we use several guards, the intersection of their visibility domains is used to determine this configuration to associate two guards. The disadvantage of the method is that guards cannot move[35]. This may be a difficulty provided that guards appear in unfavorable locations when the association of guards can be problematic.

4 Deactivation method

We now present a novel deactivation method that aims to accelerate the sampling and reduce the size (number of configurations) of a roadmap. It does so by deactivating some of the configurations of the roadmap which means that those configurations cannot be used as a neighbor configuration anymore during the learning phase. The new method tries to cut the time required to build the roadmap by deactivating those configurations of the roadmap that would not be needed later during the learning phase.

Let us first examine how new configurations are added to the roadmap. After a new configuration q is sampled, a set of neighbors N is selected for it. The probabilistic roadmap (PRM) planners usually try to connect the configuration q to each q_n in N with a local planner. The configurations in N may belong to many different components in the roadmap graph G. This lets us define three distinct cases based on the way the configuration q is connected to the roadmap:

1) The new configuration q cannot be connected to any configuration q_n in N. See Fig. 2 (a).

2) The new configuration q can be connected to multiple configurations in N that are in different components. See Fig. 2 (b).

3) The new configuration q can be connected only to such configurations in N that belong to the same component. See Fig. 2 (c).

In Case 1, the number of the component count increases by one and because the new configuration q cannot be connected to any neighbor configuration with a local planner, it is unlikely that q could be connected to any configuration that is already in the roadmap. So the Case 1 likely increases the coverage of the roadmap. In Case 2, the number of the components of the roadmap decreases and the connectivity of the roadmap increases. In Case 3, the component count does not change. The new configuration may increase the coverage of the roadmap but because all its neighbor configurations already belong to the same component, it is possible that the coverage does not change at all.

In [36], it was argued that the main challenge in sampling based planners is to get all the nodes of the roadmap connected. This suggests that the Case 2 is the most important case. The Case 1 increases the coverage so it is also important. It seems that it might be advantageous to favor configurations that belong to either one of these cases. However, the Case 3 is more problematic. Even though the configuration that falls into this case may sometimes increase the coverage, it is possible that it will be completely useless. The problem is to decide which configurations are useless and what to do with them. Our new algorithm tries to tackle this problem.

The learning phase of the new method is presented in Algorithm 2, but its query phase is standard-like and, thus, excluded here.

Algorithm 2. The deactivation method for boosting the sampling.

Output: Roadmap $G = (V, E)$.

$V \leftarrow \varnothing$
$E \leftarrow \varnothing$
repeat
  $q \leftarrow$ a sampled configuration from configuration space C_{free}
  Write configuration q active.
  $V \leftarrow V \cup \{q\}$
  $N_q \leftarrow$ active neigbor configurations of configuration q from set V
  $c \leftarrow 0$
  for all q' in N_q **do**
    if configurations q and q' do not belong to the same component **then**
      if a path exists between configurations q and q' **then**
        $E \leftarrow E \cup \{(q, q')\}$
      end if
    else
      $c \leftarrow c + 1$
      if $c > c_{\max}$ **then**

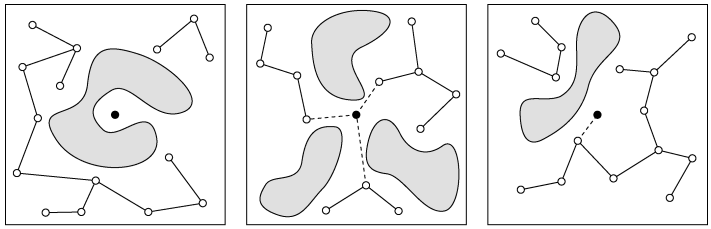


Fig. 2 An example of how a new configuration can be added to the roadmap in three different cases: (a) it cannot be connected to any roadmap component, (b) it can be connected to multiple roadmap components or (c) it can be connected only to one component. The black dot is the new configuration and the white dots are configurations that are already in the roadmap. The gray area represents obstacles.

17: Deactivate configuration q.
18: **break** for loop. Go to Row 22.
19: **end if**
20: **end if**
21: **end for**
22: **until** roadmap G is sufficient
23: **return** G

In Algorithm 2, a new configuration is selected and marked active at rows 4–6. At row 7 neighbor configurations are selected subject to a distance function, such as (4) from the set of active configurations. At row 8, the counter c is set at zero. By means of the union-find data structure, the loop at rows 9–21 checks whether the new configuration and a neighbor configuration selected are situated in the same component. If they are separate, the local planner searches for a path between them. If a path is found, an edge between these configurations is added to the roadmap, in other words, the new configuration is associated to the same component.

If the new configuration and neighbor configuration are in the same component, no path is searched for. Instead, counter c is increased by one. Now the algorithm deactivates the new configuration if the counter c is bigger than the predefined maximum number $c_{\max}$. Deactivation manifests that a configuration is no longer used in the learning phase of a roadmap. This discards neighbor configurations from the set of those active to be selected at row 7.

The counter c is important because it is used to decide whether the new configuration q is useful or not. When the configuration q is connected to some neighbor configuration q_n the first time, it is associated to the same component with configuration q_n. After that, the counter c increases by one every time when some remaining neighbor configuration belongs to the same component. We can now use this information to determine when the configuration q should be deactivated. The configuration is deactivated when the counter c is bigger than the predefined maximum number $c_{\max}$.

Let us consider about the three cases we defined earlier. In the Case 1, the counter c will always be zero and the configuration would be added to the roadmap as active. In Cases 2 and 3, the counter c may be bigger than zero, but it should be noted that it will not increase, if the new configuration q is connected to a new component. It will increase only when there are many neighbors that belong to the same component as q. This means that the configurations that are the most likely to be deactivated belong to the Case 3.

In Fig. 2 (c), for example, there are 15 white dots that represent a roadmap. Those dots are connected with edges and as can be seen, there are two distinct components in the roadmap. The new configuration q is shown as a black dot. Let us consider that all other configurations are its neighbors. First the configuration q is connected to the nearest neighbor configuration and at the same time, it will be associated with one of the components. After that, there are ten configurations left that belong to the same component as the configuration q and four configurations that belong to the other component. The algorithm will then try to connect the configuration q to all other configurations one by one, but it cannot be connected to any other component. At the end, the counter c will have a value of ten so if the constant $c_{\max}$ is smaller than ten, the new configuration will be deactivated.

When configurations are deactivated, they are not deleted from a roadmap. Nor are paths connected to deactivated configurations deleted. Deactivation only affects in the learning phase and in the query phase all configurations are again seen active. Still, it might be reasonable to delete such deactivated configurations that include only one edge to other configurations of a roadmap. Such configurations belong to the Case 3 and are often rather unnecessary. At least by removing them, the roadmap will be even smaller than without removing them.

5 Experimental research

Two programs were implemented for tests, one for roadmap design and the other for their visualization. The programs were written with Visual C++ in Windows, but they were tested with Debian GNU/Linux in a computer of AMD Athlon 64 3500+ processor with 1 GB RAM. During tests neither other programs were executed nor was virtual memory used.

A robot and obstacles were presented as triangular meshes, as usual. They were stored as text files which included the numbers of the vertices and edges of objects and the lists of vertices as three-dimensional coordinates. Collisions were detected using PQP library[21].

Four sampling techniques were programmed. First, the pseudo random sampling was the simplest technique alongside with the baseline sampling. Random numbers were uniformly generated with Mersenne twister generator[37]. Second, Halton sampling was made to produce quasi random numbers. Third, Gaussian sampling was accomplished applying Wallace method[31]. Fourth, the bridge test sampling was implemented.

Two different ways to boost the efficacy of roadmap design were programmed to compare their results. They are the visibility domain technique and deactivation method.

The function of the roadmap algorithms were experimented with four test problems: rooms, pyramids, wall, and circle. In each problem, the robot and the obstacles are different. Each problem also has one predefined roadmap query that the algorithms try to solve. These problems follow such ideas as typically used in literature, e.g., in [38, 39].

In the room, the work space is divided into eight rooms of equal volume. The rooms are perpendicular to each other (see Fig. 3), and there are holes between them. A robot of cubic form has to move via holes from one room to another so that it visits every room and does not bang walls. The robot contains 12 triangles and the obstacles have 216 triangles.

For the problem of pyramids (see Fig. 4), 500 triangular pyramids of various volumes are randomly located in the work space. A small torus robot has to move from one side of the work space to the opposite side without crashing any pyramid. There are 1200 triangles in the robot and 2000 triangles in the obstacles.

The work space of the wall problem (see Fig. 5) is divided into two parts by means of a wall which includes a hole. A robot has to slide through the hole to the opposite side, but

is not allowed to bump the wall. The robot was made of five rectangular prisms connected as an angle tube. Because the hole is rather small compared to the robot, the robot has to be rotated appropriately to go through the hole. This is a complicated problem for sampling, because the robot has to find the "important" area close to the hole even if it can freely move in both half spaces. The robot contains 72 triangles and the obstacles consist of 48 triangles.

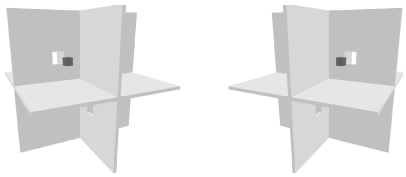


Fig. 3 The rooms problem: (a) A cubic robot at the start position and (b) at the end position

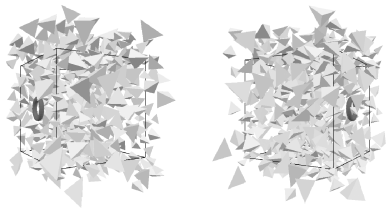


Fig. 4 The pyramids problem: (a) A torus robot at the start position and (b) at the end position

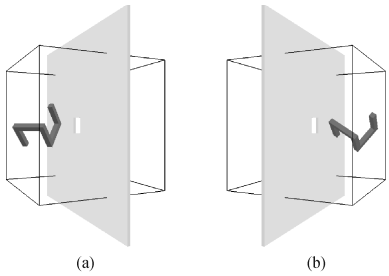


Fig. 5 The wall problem: (a) A robot (five rectangular prisms connected as an angle tube) at the start position and (b) at the end position

The circle problem (see Fig. 6) incorporates five bent tubes located so that their ends are attached to a wall. A robot is of torus form around one of the circle tube. The aim is to move the torus from one end of the bent tube to the other end avoiding any collision with the other tubes. This is a complicated problem, since the torus cannot move freely in the work space. There are 600 triangles in the robot and 10024 in the obstacles.

There is no unequivocal measure to determine the validity of probabilistic roadmap algorithms. Therefore, each test problem has one predefined query that all tested algorithms try to solve. For each algorithm, the time needed to construct a roadmap capable of solving the given query is measured as well as the size of the final roadmap. This means that the time measuring starts when the algorithm begins to construct the roadmap and that the construction and timing stops immediately when the predefined goal and start configurations become connected. By looking at the time needed to construct the roadmap and the size of the roadmap, it is possible to compare different sampling algorithms and see what kind of roadmaps they produce to solve one query. This approach is common in robotics literature.

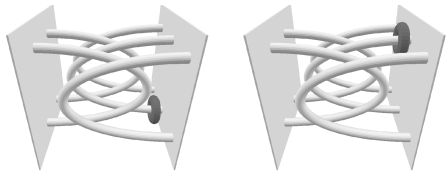


Fig. 6 The circle (bent tubes) problem: (a) A torus robot at the start position and (b) at the end position

It should also be noted that the predefined query is used only in the ending condition of the algorithms. The algorithms themselves do not know anything about the query during the planning and they do not try to bias their sampling to solve the specific query.

6 Test results

All four problems were solved with four sampling techniques: randomization of baseline algorithm, (deterministic) Halton, Gaussian, and bridge test. Since the technique of Halton was deterministic, single runs were only performed. For all other, runs were repeated 100 times to average varying results originated from random inputs. Figs. 7–10 show execution times of all tests. These boxplot results include the lower quartile, median and upper quartile within the box, and the minimum and maximum as bars. Table 1 shows the final sizes of the roadmaps of all tests.

Fig. 7 contains the boxplot results of the rooms problem. Deactivation considerably improved the execution times for the baseline and Halton sampling. It was better than the visibility domain technique for all four sampling methods. Nevertheless, the basic situation was close to it for the Gaussian and bridge test sampling methods. The average execution times using deactivation were 7, 2, 6, and 5 s for the randomization of baseline, Halton, Gaussian, and bridge test methods. In the same order, the means of configurations in a roadmap decreased from approximately 12 000 to 900 for the basic situation, from 1 400 to 600 for deactivation and from 32 to 48 for visibility domain. In the rooms problem there are large areas resulting in several free areas in the configuration space. Since the randomization technique collects great amounts of configurations while uniformly sampling the configuration space, unnecessary configurations are abundantly included. After all, visibility domain is clearly the slowest method.

Fig. 8 depicts the boxplot results of the pyramids problem. This differs from the rooms problem in the sense that there are no large empty areas in the work space. Despite this, both can be characterized as easy in the light of execution times. The results in Fig. 8 indicate how there are no great differences between four sampling methods. Further, the results tend to resemble those in Fig. 7, but the scale of the vertical axis is greater, since the execution times were longer than in Fig. 7. Deactivation was the best of three alternatives, but the basic situation was close to it. Visibility domain technique remained far away from them. The means of the execution times were around 20 s for the basic situation and deactivation, and from 70 to 190 s to the visibility domain technique. The means of the configurations were roughly from 4 500 down to 1 700 for the randomization of the baseline, from 2 500 to 1 300 for deactivation and from 1 100 to 800 for visibility domain. Mostly, the differences of the execution times between four sampling

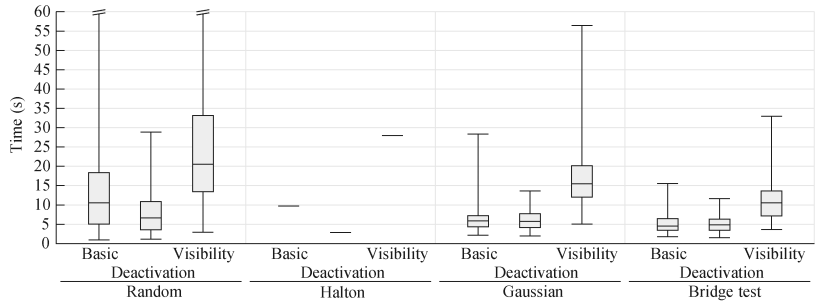


Fig. 7 Boxplots for the rooms problem: minimum, lower quartile, median, upper quartile, and maximum. On the left, times above 60 s are not shown

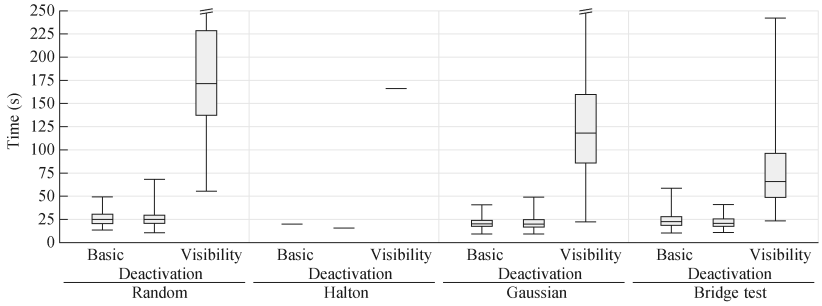


Fig. 8 Boxplots for the pyramids problem: minimum, lower quartile, median, upper quartile, and maximum. Times above 250 s are not shown

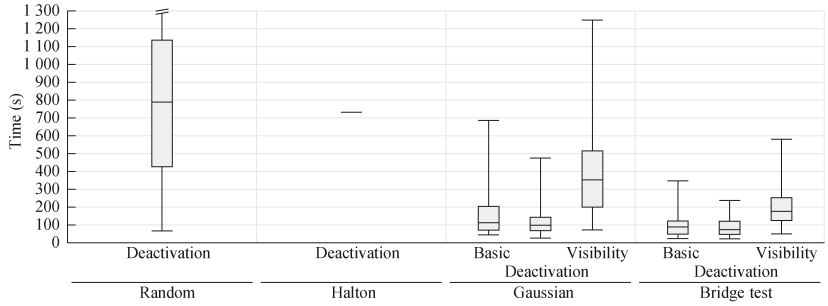


Fig. 9 Boxplots for the circle problem: minimum, lower quartile, median, upper quartile, and maximum

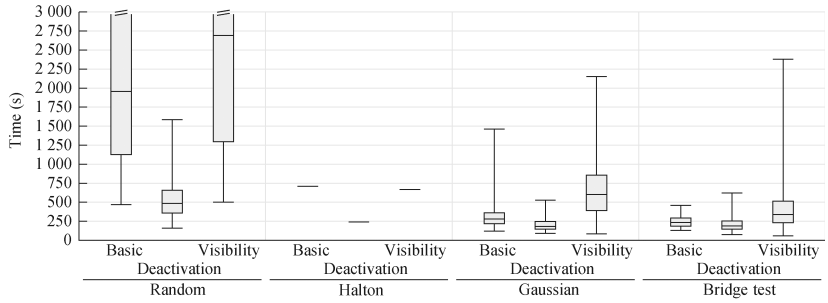


Fig. 10 Boxplots for the wall problem: minimum, lower quartile, median, upper quartile and maximum. Times above 1300 s are not shown

Table 1 Average roadmap sizes (number of configurations) for the rooms, pyramids, circle, and wall problems

	Rooms problem	Pyramids problem	Circle problem	Wall problem
Random, basic	11665.2	4534.1	569778.5	–
Random, deactivation	1422.9	2574.0	1730.2	1599.5
Random, visibility	31.7	1163.1	945.8	–
Halton, basic	7506.0	3239.0	282506.0	1665516.0
Halton, deactivation	786.0	1577.0	1143.0	1575.0
Halton, visibility	29.0	1115.0	612.0	260.0
Gaussian, basic	3527.3	2966.3	67490.0	60294.3
Gaussian, deactivation	1563.1	2080.7	1897.7	1345.3
Gaussian, visibility	68.2	1039.5	1031.5	250.2
Bridge test, basic	894.9	1716.6	26356.3	22928.7
Bridge test, deactivation	573.6	1323.3	1924.1	699.3
Bridge test, visibility	48.4	811.4	660.0	143.1

methods were minor, because in the pyramids problem there are plenty of obstacles uniformly distributed in work space. Deactivation again attained the fastest execution times. However, the execution times of the basic situation were close. Deactivation also reduces remarkably the numbers of configurations although these were greater than those of visibility domain.

Fig. 9 shows the results of the circle problem. Deactivation was faster than two other techniques. Notice how the execution times are essentially greater than in Figs. 7 and 8 because of the more difficult problem. The means of the execution times were from 2 330 down to 230 s for the basic situation, from 500 s to 190 s for deactivation and from 3 340 s to 400 s for visibility domain. Respectively, the means of configurations were from 570 000 down to 26 000, from 1 900 to 1 100 and from 1 000 to 600.

Fig. 10 presents the boxplot results of the wall problem. For the randomization of the baseline algorithm and Halton method deactivation was the only reasonable technique to compute, since individual runs of two other (not given in Fig. 10 or in Table 1) took 1.5 h or even far more hours. Looking at only the Gaussian and bridge test samplings, deactivation was the fastest method with the means of execution times as 109 s and 76 s. The means of configurations were 60 290 and 22 930 for the basic situation, 1 340 and 700 for deactivation and 250 and 140 for visibility domain. Concerning the randomization of the baseline algorithm and Halton method, deactivation gave about 1 600 configurations. The basic situation could consist of even millions of configurations.

7 Conclusions and future work

The execution times obtained imply Gaussian and bridge test to be the fastest sampling methods. The randomization of the baseline algorithm was the slowest of all. Particularly, the circle and wall problems underpin the conclusion. The differences between the sampling methods were at their smallest with the pyramids problem, which posed the most uniformly distributed obstacles in the work space. This was unwieldy for bridge test, whereas randomization and Halton were at their best. On the other hand, Halton sampling with deactivation yielded the fastest execution times.

Domain visibility generated the smallest roadmaps (fewest configurations), but it was also slow, especially compared to deactivation. Throughout all sampling methods, deactivation always reduced execution times.

Deactivation is effective to eliminate such neighbor configuration candidates that are already in the present component. This is frequent since in the sampling of the learning phase close configurations cumulate to a cluster residing in the present component. Then sampling a new configuration

within the same component is useless, because the main objective is to unite separate components. Reducing this useless computation can yield a considerable improvement in execution time. To summarize, deactivation accelerates roadmap processing by pruning unnecessary computation. It does not minimize the number of configurations chosen for a roadmap like in the visibility domain method. Nevertheless, this property could be associated to the deactivation method by eliminating deactivated configurations from a roadmap provided that such elimination would not affect the number of components. This is possible when a deactivated configuration is only connected to one configuration.

Configuration deactivation technique presented is worth studying further with new test problems. The deactivation parameter ($c_{\max}$) was a constant. It deserves additional work to choose it automatically or to regulate it during execution.

The present roadmap algorithms do not make use of collision detection. It would be affordable to collect all information from all single calls of a collision detection algorithm, which explores the configuration space. On the other hand, such co-operation of a collision detection algorithm and a probabilistic roadmap algorithm could diminish generalization; an independent roadmap algorithm can function well, for example, for robots of rigid body and complicated ones of several moving parts.

References

[1] H. Choset. *Principles of Robot Motion: Theory, Algorithms, and Implementations*, USA: MIT Press, 2005.

[2] S. M. LaValle. *Planning Algorithms*, Cambridge, UK: Cambridge University Press, 2006.

[3] L. E. Kavraki, P. Švestka, J. C. Latombe, M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, vol. 12, no. 4, pp. 566–580, 1996.

[4] M. H. Overmars. A Random Approach to Motion Planning. Technical Report RUU-CS-92-93, Department of Computer Science, Utrecht University, the Netherlands, 1992.

[5] J. H. Reif. Complexity of the mover′s problem and generalizations. In *Proceedings of the 20th Annual Symposium on Foundations of Computer Science*, IEEE, San Juan, Puerto Rico, pp. 421–427, 1979.

[6] J. F. Canny. *The Complexity of Robot Motion Planning*, Cambridge, MA, USA: MIT Press, 1988.

[7] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computing*, vol. C-32, no. 2, pp. 108–120, 1983.

[8] J. B. Kuipers. *Quaternions and Rotation Sequences: A Primer with Applications to Orbits, Aerospace and Virtual Reality*, Princeton, USA: Princeton University Press, 2002.

[9] J. C. Latombe. *Robot Motion Planning*, 2nd ed., Boston, USA: Springer, 1991.

[10] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *International Journal of Robotics Research*, vol. 5, no. 1, pp. 90–98, 1986.

[11] Q. J. Peng, X. M. Kang, T. T. Zhao. Effective virtual reality based building navigation using dynamic loading and path optimization. *International Journal of Automation and Computing*, vol. 6, no. 4, pp. 335–343, 2009.

[12] T. K. Wang, Q. Dang, P. Y. Pan. Path planning approach in unknown environment. *International Journal of Automation and Computing*, vol. 7, no. 3, pp. 310–316, 2010.

[13] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, D. Vallejo. OBPRM: An obstacle-based PRM for 3D workspaces. In *Proceedings of the 3rd Workshop on Algorithmic Foundations of Robotics*, ACM, Natick, USA, pp. 155–168, 1998.

[14] V. Boor, M. H. Overmars, A. F. van der Stappen. The Gaussian sampling strategy for probabilistic roadmap planners. In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, Detroit, USA, vol. 2, pp. 1018–1023, 1999.

[15] D. Hsu, J. C. Latombe, R. Motwani. Path planning in expansive configuration spaces. In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, Albuquerque, USA, vol. 3, pp. 2719–2726, 1997.

[16] S. A. Wilmarth, N. M. Amato, P. F. Stiller. MAPRM: A probabilistic roadmap planner with sampling on the medial axis of the free space. In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, Detroit, USA, vol. 2, pp. 1024–1031, 1999.

[17] P. Jiménez, F. Thomas, C. Torras. Collision detection algorithms for motion planning. *Robot Motion Planning and Control*, J. P. Laumond, Ed., Berlin, Germany: Springer-Verlag, pp. 1–53, 1998.

[18] T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein. *Introduction to Algorithms*, 2nd ed., Cambridge, MA, USA: MIT Press, 2001.

[19] J. J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, vol. 4, pp. 3993–3998, 2004.

[20] K. Shoemake. Uniform random rotations. *Graphics Gems III*, D. Kirk, Ed., San Diego, CA, USA: Academic Press, pp. 124–132, 1992.

[21] E. Larsen, S. Gottschalk, M. C. Lin, D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, vol. 4, pp. 3719–3726, 2000.

[22] S. A. Ehmann, M. C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. *Computer Graphics Forum*, vol. 20, no. 3, pp. 500–511, 2001.

[23] G. van den Bergen. Efficient collision detection of complex deformable models using AABB trees. *Journal of Graphics Tools*, vol. 2, no. 4, pp. 1–13, 1997.

[24] G. van den Bergen. A fast and robust GJK implementation for collision detection of convex objects. *Journal of Graphics Tools*, vol. 4, no. 2, pp. 7–25, 1999.

[25] B. Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, vol. 17, no. 3, pp. 177–208, 1998.

[26] T. C. Hudson, M. C. Lin, J. Cohen, S. Gottschalk, D. Manocha. V-COLLIDE: Accelerated collision detection for VRML. In *Proceedings of the 2nd Symposium on Virtual Reality Modeling Language*, ACM, Monterey, USA, pp. 117–124, 1997.

[27] D. Hsu, J. C. Latombe, H. Kurniawati. On the probabilistic foundations of probabilistic roadmap planning. *International Journal of Robotics Research*, vol. 25, no. 7, pp. 627–643, 2006.

[28] L. E. Kavraki. Random Networks in Configuration Space for Fast Path Planning, Ph. D. dissertation, Stanford University, USA, 1995.

[29] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, vol. 1, pp. 630–637, 1998.

[30] D. B. Thomas, W. Luk, P. H. W. Leong, J. D. Villasenor. Gaussian random number generators. *ACM Computing Surveys*, vol. 39, no. 4, Article No. 11, 2007.

[31] C. S. Wallace. Fast pseudorandom generators for normal and exponential variates. *ACM Transactions on Mathematical Software*, vol. 22, no. 1, pp. 119–127, 1996.

[32] D. Hsu, T. Jiang, J. Reif, Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of IEEE International Conference on Robotics and Automation*, IEEE, vol. 3, pp. 4420–4426, 2003.

[33] S. M. LaValle, M. S. Branicky, S. R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *International Journal of Robotics Research*, vol. 23, no. 7–8, pp. 673–692, 2004.

[34] J. H. Halton. On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals. *Numerische Mathematik*, vol. 2, no. 1, pp. 84–90, 1960.

[35] T. Siméon, J. P. Laumond, C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, vol. 14, no. 6, pp. 477–493, 2000.

[36] R. Geraerts, M. H. Overmars. Reachability-based analysis for probabilistic roadmap planners. *Robotics and Autonomous Systems*, vol. 55, no. 11, pp. 824–836, 2007.

[37] M. Matsumoto, T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, vol. 8, no. 1, pp. 3–30, 1998.

[38] R. Geraerts, M. H. Overmars. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems*, vol. 54, no. 2, pp. 165–173, 2006.

[39] R. Geraerts, M. H. Overmars. Creating high-quality paths for motion planning. *International Journal of Robotics Research*, vol. 26, no. 8, pp. 845–863, 2007.

Mika T. Rantanen received the M. Sc. degree at the Department of Computer Sciences, University of Tampere, Finland in 2009, and now continues his research on robotics toward a Ph. D. degree.

His research interest includes probabilistic roadmap algorithms.

E-mail: mika.t.rantanen@uta.fi (Corresponding author)

Martti Juhola received the M. Sc. and Ph. D. degrees in computer science from the University of Turku, Finland in the 1980 s, where he was an academic assistant, lecturer, and researcher, later becoming a professor at the University of Kuopio, Finland. Since 1997, he is a professor at the University of Tampere, Finland.

His research interests include medical informatics, signal analysis, pattern recognition, and information retrieval.

E-mail: martti.juhola@cs.uta.fi

Publication II

A connectivity-based method for enhancing sampling in probabilistic roadmap planners

Mika T. Rantanen

J Intell Robot Syst (2011) 64:161–178
DOI 10.1007/s10846-010-9534-4

A Connectivity-Based Method for Enhancing Sampling in Probabilistic Roadmap Planners

Mika T. Rantanen

Received: 18 September 2010 / Accepted: 23 December 2010 / Published online: 11 January 2011

Abstract The motion planning is a difficult problem but nevertheless, a crucial part of robotics. The probabilistic roadmap planners have shown to be an efficient way to solve these planning problems. In this paper, we present a new algorithm that is based on the principles of the probabilistic roadmap planners. Our algorithm enhances the sampling by intelligently detecting which areas of the configuration space are easy and which parts are not. The algorithm then biases the sampling only to the difficult areas that may contain narrow passages. Our algorithm works by dividing the configuration space into regions at the beginning and then sampling configurations inside each region. Based on the connectivity of the roadmap inside each region, our algorithm aims to detect whether the region is easy or difficult. We tested our algorithm with three different simulated environments and compared it with two other planners. Our experiments showed that with our method it is possible to achieve significantly better results than with other tested planners. Our algorithm was also able to reduce the size of roadmaps.

Keywords Motion planning · Probabilistic roadmaps · Narrow passage problem

1 Introduction

The task in a motion planning is to find a path for a robot from an initial position to a final position. The robot moves in a fixed environment that has obstacles and the collision between the robot and the obstacles must be avoided [19]. The motion planning is an important part of robotics as it can be used to create a robot that can move inside its environment independently. Furthermore, motion planning has many

M. T. Rantanen (✉)
Department of Computer Sciences, University of Tampere, 33014 Tampere, Finland
e-mail: mika.t.rantanen@uta.fi

other applications in addition to robotics. It can be utilised for example in a computer animation and computer games [7, 9] or in a computational biology [6, 24].

In the motion planning, the robot moves in a *work space* that contains obstacles. The robot itself can be a rigid body or for example an articulated robot. The motion planning problem is usually formulated by using the *configuration space* C that is the space of all possible configurations of the robot. One configuration describes exactly one position of the robot. The set of configurations where the robot does not collide with obstacles is called a *free space* C_{free} and the complement set $C \setminus C_{\text{free}}$ is called an *obstacle space*. We are interested in a collision-free path from the initial configuration q_{init} to the final configuration q_{final}.

The motion planning is a difficult problem and the general motion planning problem is PSPACE-complete [5, 21]. That is why exact motion planning algorithms are usually useful only in easy problems that have a low-dimensional configuration space. In practical applications, they are too slow to be used. However, since the motion planning problem has practical significance, there have been many different approaches to create feasible heuristics to solve high-dimensional motion planning problems quickly. Most of these methods can be categorized into three groups: cell decomposition methods, potential field methods and sampling based methods [19]. Especially sampling based methods have turned out to have an excellent performance in high-dimensional problems.

Probabilistic roadmap (PRM) planners [17] are popular and efficient way to solve high-dimensional motion planning problems. The PRM planners are one form of the sampling based methods and they build only a simplified representation of C_{free} as opposed to an exact representation that would be too costly to compute. The representation that the PRM planner uses is a *probabilistic roadmap*. It is built by sampling the configuration space and is represented as a graph that is usually undirected and unweighted. The nodes represent different free configurations in C and the edges between the nodes represent a collision-free path between the corresponding configurations. This means that the whole roadmap lies in the C_{free}.

PRM planners work in two phases. The first phase is the preprocessing phase where the roadmap is constructed. The second phase is the query phase where the roadmap is used to solve motion planning queries. Because the roadmap is done before the query phase, the PRM planners are usually multi-query planners and the same roadmap can be used to solve many queries. On the opposite, the single-query planners can answer only one query, but can usually give the answer faster than multi-query planners.

Once the roadmap has been built, it can be used to solve different queries very quickly and reliable. During the query phase, the path in C_{free} is found between two arbitrary configurations q_{init} and q_{final}. First, both configurations must be connected to the roadmap by finding a simple free path from both configurations to some configurations that are already in the roadmap. After that it is easy to search for the shortest path between q_{init} and q_{final} in the roadmap graph. The query phase fails if connecting configurations q_{init} and q_{final} to the roadmap fails or if the configurations do not belong to the same connected component in the roadmap.

The actual work is done during the preprocessing phase when the roadmap is constructed and, thus, it is also the most time-consuming part of the PRM planners.

Even though the exact representation of the configuration space is difficult to compute, there are fast methods to check whether some configuration or a path between two configurations is collision-free [16, 18]. PRM planners rely on these fast methods during the preprocessing phase to efficiently test if some configuration or path lies in C_{free}.

At the beginning of the preprocessing phase, the roadmap is empty. Configurations are then sampled from C by using some sampling method and if the sampled configurations are free they may be added to the roadmap. For each added configuration q a set of neighbor configurations N is also chosen from the roadmap. Sampled configuration q is then tried to be connected to each neighbor configuration $n \in N$ and if there is a free path between the configurations q and n, an edge is added to the roadmap between the corresponding nodes. This continues until some predefined end condition is met. The most of the PRM planners work only in a fixed environment where the obstacles do not move. However, there have been attempts to use PRM planners also in dynamic environments [2, 9].

The roadmap should have a good *coverage* and *connectivity*. When the roadmap has a good coverage, for any $q \in C_{\text{free}}$ there should be a simple collision-free path that can connect q to some configuration in the roadmap. The good connectivity means that if configurations q_1 and q_2 are in the roadmap and there exists a path between them in the C_{free}, both q_1 and q_2 should also belong to the same connected component in the roadmap graph.

Most of the PRM planner variations work very well in easy environments [1, 3, 12, 27]. However, their performance usually degrades when the environment contains difficult areas like narrow passages between the obstacles. This happens mainly because they try to sample configurations from the entire configuration space and are unable to detect which areas are actually difficult and need sampling. Some planners try to identify difficult areas by using local information about the configuration space [22, 28]. While this approach can be helpful, it still ignores information that has been gathered from the whole configuration space during the planning.

In this paper, we propose a method to recognize possible difficult areas in the work space and a new multi-query planner that uses this information to solve high-dimensional motion planning problems. Our method enhances the basic version of the probabilistic roadmap planner by learning where the most difficult areas of the work space are and concentrating on planning in these areas. At the beginning, it divides work space into different regions and decides during the planning which regions are useful. The main novelty of this method is the way it uses the connectivity of the whole roadmap to make those decisions. This allows our method to guide sampling intelligently by using both local and global information available. Additionally, our method also restricts the roadmap size by removing samples that likely do not improve the connectivity or the coverage.

Our experiments show that with our method it is possible to achieve significantly better results than with other probabilistic planners tested. Our method works faster than other tested methods and it also produces small roadmaps.

In the next section, we describe what kind of heuristics have already been developed to make PRM planners work better.

2 Related Work

Many different approaches have been proposed to enhance the PRM planners. In this section, we give an overview of those proposed methods that are the most relevant to our work.

2.1 Sampling Methods

The nodes in the roadmap must be chosen by some method. In the original PRM planner [17], the configurations were sampled from the C_{free} randomly using a uniform distribution. Even that simple method has been shown to work quite efficiently in many tested environments [8]. However, there has been a lot of interest to develop heuristics that can work even better in difficult environments. The most common reason for an environment to be difficult is that it contains narrow passages.

Obstacle-based PRM [1] was one of the first methods that were developed to enhance the original PRM planner. It tries to sample configurations near the boundary of the obstacles in order to sample those areas of the configuration space that are actually interesting. The Gaussian sampler [3] is somewhat similar since it also tries to sample configurations near the obstacles. First, it generates one configuration randomly using a uniform distribution. Then it uses a normal distribution to select a distance d and then generates another configuration that is at the distance d from the first configuration. If one of the configurations is collision-free and the other is not, the collision-free configuration is added to the roadmap. Otherwise, neither of the configurations is added.

The bridge test sampler [12] tries to sample free configurations that lie between the obstacles and this way concentrate its efforts on narrow passages. At first the sampler selects two configurations q_1 and q_2. These can be selected, for example, in the same way as the Gaussian sampler selects its two configurations. If both of the configurations q_1 and q_2 are in collision, the sampler calculates the midpoint of the line segment between the q_1 and q_2 and checks whether this midpoint configuration q_3 is collision-free. If it is, the configuration q_3 is added to the roadmap. In all other cases, all configurations are discarded.

Some of the proposed methods try to sample configurations from the medial axis of the free space [11, 20, 27]. This way it would be possible to sample the narrow passages, but at the same time to keep the samples as far from the obstacles as possible. It is difficult and slow to compute the exact medial axis especially in high-dimensional spaces, but luckily it is possible to obtain samples from the medial axis without actually computing it [27].

Many planners require a fine-tuning of different parameters for each environment separately. This can be time-consuming and quite difficult. Therefore there has recently been interest to develop methods that are adaptive and require as little user intervention as possible [15, 25].

It is also possible to combine different sampling methods together to achieve better distributions of samples. The bridge test sampler, for example, is often used together with the uniform sampler just as is suggested in [12]. Another way to combine different samplers is presented in [15]. They use a set of different samplers

with a cost associated with them and one of the samplers is selected in each iteration. If the sampler performs well, its cost will decrease and the probability that it will be selected again increases. If the sampler performs poorly, its cost will increase. In [26] another way is presented to combine different sampling methods together. They chain different samplers together so that one sampler outputs its samples to be used as input values for the next sampler.

Even though all these presented methods can be useful, there is one problem: they still use the same sampling strategy to sample the whole configuration space. In each iteration, some sampler or sampler combination is used to generate a configuration from the whole configuration space. It would be much better if the sampler could detect which part of the configuration space actually needs sampling and sample only in those areas.

2.2 Region-Based Samplers

Some algorithms are, at a broad level, similar to our algorithm. One notable example combines an approximate cell decomposition method with the PRM [28]. That method divides the configuration space into cells and in each cell a localized probabilistic roadmap is generated. This way it is possible to decrease the number of cells because the probabilistic roadmap can capture the connectivity of large cells after which there is no need to divide that cell any more. On the other hand, roadmap methods do not perform well in narrow passages and in these areas the cells are divided into smaller cells until the roadmap can be constructed through the narrow passage. Their method is a resolution-complete planner in contrast to normal PRM planners that are probabilistically complete. It means, that there is a predefined minimum size that the cell can be. If the solution is not found with that size, the planner decides that there is no path.

However, their approach has some limitations. First of all, it is a single-query planner unlike most PRM planners. Thus, it is designed to answer only to one motion planning query. Additionally, they mention in [28] that their method has an exponential complexity with the number of degrees of freedom in the worst case. Also, the graph searching becomes a major problem in higher-dimensional environments since the size of the roadmap increases dramatically because their method does not reduce the number of the roadmap nodes efficiently enough.

RESAMPL [22] is another sampler that resembles our method. It also takes into consideration the empty space and tries to bias the sampling towards the difficult areas of the configuration space by dividing the space into different regions. RESAMPL first generates an initial set of samples that can be free or collision configurations. Based on these samples, the set of regions is generated so that each region contains one representative sample and its neighboring samples. One sample can be included in multiple regions. These regions are then classified into four classes: free, blocked, surface and narrow. If necessary, additional samples can be added to the regions during the classification process.

The classification is based on the entropy of the region [22]. Regions that contain mostly free or collision configurations have low entropy and are classified into either a free or a blocked class. Regions that contain both free and collision configurations have high entropy. These regions are classified into a surface or a narrow class.

Surface regions lie near the obstacles and narrow regions contain narrow passages. The tricky part is to decide whether the region should be classified as a surface or a narrow class. In [22], this is done simply by dividing the region into two sub-regions. If both sub-regions have low entropy, the region is classified as a surface. Otherwise the class of the region might be narrow. After the classification, the class information can be used to guide the sampling.

The major problem with RESAMPL is that there is a great risk that it misclassifies regions because narrow passages can be very complex and hence difficult to detect. When RESAMPL classifies a region, it uses only the local information based on the entropy of that region. It does not use the connectivity information of the whole roadmap or any other global means to guide the classification.

3 Connectivity of the Roadmap

It has been shown by experiments and with more formal analyses that the PRM planners usually work remarkably well even in complex environments [8, 13, 14]. In many environments, the PRM planner is able to capture the connectivity of the configuration space with a small number of configurations and the probability that the planner solves the query increases rapidly as the size of the roadmap grows [14]. There are, however, situations when the performance of the PRM planners is poor.

3.1 Narrow Passages

The narrow passages were recognized to be a problem when the PRM planner was introduced [17] and in [14] narrow passages were analysed more thoroughly. The notion of the *expansiveness* was introduced in [14]. In short, the connectivity of the expansive configuration space is easy to be captured by a PRM planner and, on the other hand, it is difficult otherwise. Narrow passages can cause the configuration space to be poorly expansive.

An important observation is that the whole configuration space is not poorly expansive and hence difficult if there are narrow passages in it. The narrow passages may only affect a small subset of the configuration space and while that subset has poor expansiveness, most of the configuration space components may be much easier to solve. This is something we can exploit while trying to increase the effectiveness of the PRM planners. Our algorithm, for example, tries to capture the connectivity of the easy parts of C_{free} quickly and then detect what parts are difficult and focus only on those areas.

It is also important to notice, that especially in real-world motion planning problems, the environment where the robot moves is rarely arbitrarily random. This makes it feasible to develop heuristics for a motion planning since if the environment is random, we can not make any assumptions whether some configuration lies in a free configuration space even if we know that there are many free configurations near it. In random environment it would not be beneficial to use any heuristics to sample the configuration space because the simple uniform sampling would work just as well. Luckily, there are some properties that are common in many real-world environments.

In [13] it is argued that narrow passages are not stable geometric features and that even small perturbations of the workspace geometry either can eliminate the passage altogether or make it wider. This suggests that in real-world environments it is unlikely that very difficult narrow passages would occur by accident. It is also shown experimentally that widening the narrow passages can dramatically decrease the time needed to solve motion planning queries [23]. Additionally, whether we are working with industrial robots or, for example, with some digital character in computer games, there are usually broad areas in the work space that do not contain any obstacles. This is advantageous since clearly the motion planning is easier in areas that do not contain obstacles.

3.2 Detecting Narrow Passages

Many algorithms that have been developed to make probabilistic roadmap planners to work faster concentrate on the obstacles. Many of them sample a random configurations from C_{free} and then, by some heuristic, try to detect whether they lie near the obstacle boundaries and then only accept those. This usually works quite well since the narrow passages are situated near the obstacles being the most difficult areas to be sampled. However, these methods still ignore much of the information about the empty space that has been gained during the planning.

One of the problems with most of the probabilistic roadmap planners is that they sample entire configuration space. The bridge test sampling [12] and the Gaussian sampling [3] can concentrate on the areas near the obstacles, but they still need to sample the entire space randomly until they find an obstacle. In other words, they sample unnecessarily and because each sample needs to be checked for a possible collision, it will cause a lot of computation. While these methods work well on some environments, they can be problematic if there are many obstacles and narrow passages of a different kind in the same environment.

Another problem with some of these samplers is that they depend on parameters used throughout the configuration space. The bridge test sampler, for example, requires a parameter that defines the length of the constructed "bridge". This same length is then used all the time no matter in which part of the configuration space the sample is taken. This is problematical since there may be many narrow passages with different shapes and sizes that would each need a bridge with different length.

In [10], a reachability-based analysis was done for PRM planners and one conclusion was that the main challenge is to get the nodes in the roadmap connected especially in narrow passages. They also noted that the coverage was rarely a bottleneck. That is why it is important to try to connect the nodes in the narrow passage together and concentrate the efforts on there instead of sampling unnecessary nodes in other regions.

Our algorithm tries to tackle exactly these problems. Our method divides the configuration space into different regions at the beginning of the planning. Then it tries to decide during the planning which regions actually need sampling and which already have enough samples to capture the connectivity of the region. When our method decides that some region already has enough samples, it stops sampling there. This way our method can very quickly start to ignore those regions of the configuration space that have a good expansiveness and focus on those that are actually difficult.

In the next section, we describe our method exhaustively. Later we present empirical results obtained when we tested our algorithm.

4 New Method

In this section, we describe a new algorithm that can be used to solve difficult motion planning problems. In the next section, we test this algorithm experimentally and notice that it is constantly faster than some other state-of-the-art planners. The algorithm is based on the observation that many environments in motion planning contain large empty areas in their work space and that it is very easy to capture the connectivity of these easy areas with a roadmap. It is much harder in the difficult areas that contain narrow passages.

The main problem is to recognize efficiently the empty areas and the areas that may contain narrow passages. Since the probabilistic roadmap planners do not have an exact information about the work space, we must rely on the sampling. Basically, we divide the work space into multiple regions and then start to sample each one of them. Based on these samples, we can later decide whether the region is easy or difficult. When the region is determined to be easy, we can decide that it is not necessary to sample inside that region any more and we can classify that region as *solved*. The regions that have a good expansiveness and do not contain many obstacles can be classified as solved quite quickly. The more difficult the region is, the longer time it takes on average to make that decision.

At the beginning, our algorithm divides the work space into the regions that are approximately same-sized. The regions can be generated for example by using Hammersley sequence that creates a point set of a low discrepancy which means that the generated points are not distributed randomly, but instead more uniformly [4]. For each region, we first calculate the centroid of the region. Then for each centroid we calculate its distance to the nearest centroid. We use this distance as a radius that defines the size of the region. The regions are understood as hyperspheres that are described by the center point of the sphere and the radius. It should also be noted that the spheres are not distinct, but instead overlap each other.

After the regions are created we can start to sample configurations. We go through all regions one by one and sample one configuration from each of them. After the last region is sampled, we start again from the first region. If the sampled configuration is free, we may add it to the roadmap. After enough free configurations are sampled on the region, we check whether all of the configurations in the region belong to the same connected component in the roadmap. If they do, we decide that the roadmap has captured the connectivity in the region well enough and we can stop sampling in that region.

4.1 Detailed Description

Algorithm 1 describes our method in a more detailed manner. In line 1, we generate the regions and initialize all parameters associated with each region. In line 2, we start a loop that runs until some predefined end condition is met. We can, for example, stop the loop based on the roadmap size.

Algorithm 1 Generates a roadmap

1: Generate a set of regions. Each region R is associated with three variables. R_{cent} is the centroid point of the region. R_{dist} is the distance from the centroid to the centroid of the nearest region. R_{free} is used to count the free configurations sampled in the region.
2: **loop**
3: $R \leftarrow$ next region
4: $q \leftarrow$ randomly generated configuration based on R_{cent} and R_{dist}
5: **if** $q \in C_{\text{free}}$ **then**
6: $R_{\text{free}} \leftarrow R_{\text{free}} + 1$
7: $N \leftarrow$ all neighbor configurations that are within the distance $2 \times R_{\text{dist}}$ from q
8: **if** $|N| < 2$ **or** all neighbors in N are not in the same component **then**
9: Add q to the roadmap.
10: Add q to the region R.
11: Try to connect q to all neighbors $n \in N$ that are in the different component than q with a local planner. If the connection is successful, add corresponding edge to the roadmap.
12: **if** there are more than one region left **then**
13: **if** $R_{\text{free}} \geq$ `freelimit` **then**
14: **if** all configurations on the region R are in the same component **then**
15: Remove region R.

In line 3, we select a region R and in line 4 we generate a random configuration in that region. The configuration is generated by utilizing the information given by the region: the centroid and the distance from the nearest region. In our experiments, we used the normal distribution to generate a random configuration. The method is shown in Algorithm 2.

Algorithm 2 Generates a configuration in the region

1: $q_1 \leftarrow$ a centroid point of the current region R
2: $d \leftarrow$ a distance chosen from a normal distribution with a standard deviation of R_{dist}
3: $q_2 \leftarrow$ a random configuration at a distance d from q_1
4: **return** q_2

When the normal distribution is used this way, most of the sampled configurations are located near the center of the region. However, some of the configurations will overlap with the configurations in the neighbor regions and there is a small probability that some configurations will be sampled even further. This is a desired outcome because it will help connecting configurations from different regions together.

In line 5, we check if the generated configuration q lies in the free configuration space. If it is not free, we can ignore it and continue to the next region and go back to

the line 3. Otherwise we increase the R_{free} counter associated with the region by one in line 6. This counter keeps track of how many free configurations we have sampled in the region R.

In line 7, we get the neighbor configurations N of the configuration q. In this case, the neighbor configurations mean such configurations that are already in the roadmap and are within the distance $2 \times R_{\text{free}}$ from the configuration q. If there are more than one neighbor, we test whether all neighbor configurations $n \in N$ belong to the same component in the roadmap (line 8). If they are in different components or if there are less than two neighbors, we add configuration q to the roadmap and to the current region R (lines 9 and 10). In line 11, we try to connect the configuration q to all neighbor configurations $n \in N$ that are not already in the same component as configuration q.

If there are more than one neighbors and all of them belong to the same component, we will not add configuration q to the roadmap or to the region. The reason is that the configuration q would probably not enhance the connectivity or the coverage of the roadmap and hence the whole configuration would be quite useless. By not adding it to the roadmap, we can keep the roadmap smaller without losing the connectivity of the roadmap and that is usually a great advantage. Notice that we increase the counter R_{free} in line 6 even though we do not necessarily add the free configuration to the roadmap.

In several occasions, our algorithm must know whether two configurations belong to the same connected component in the roadmap. Fortunately, this can be done very efficiently by utilizing a disjoint-set data structure to keep track of the configurations.

In the last part of our algorithm, we test whether we can classify the whole region as solved. If there are more than one region active, we test whether the R_{free} is equal or greater than a predefined constant `freelimit` (lines 12 and 13). If it is, we check whether all configurations in the region R belong to the same component (line 14). If they all are in the same component, we can conclude that the region is solved and there is no need to sample on it any more and so in line 15 we can remove the region. When the region is removed the configurations of the region are not removed from the roadmap. However, it could be possible to reduce the size of the roadmap even further if we could remove the unnecessary configuration of the region. These could be, for example, the configurations that have only edge.

The constant `freelimit` is an important parameter since it defines how much the regions are sampled before those can be classified as solved. Each region has a counter for how many free configurations have been sampled in that region. After the counter has reached the `freelimit` constant, the region can be removed if necessary. In easy regions that do not have many obstacles the counter will reach the `freelimit` quickly. On the other hand, it will take a longer time in the difficult areas that contain many obstacles. It is possible that a region center is deep inside the obstacle and that all configurations near the center are also inside the obstacle. These kinds of regions are very difficult to be removed. However, this is just the result that the algorithm tries to achieve. Areas that have only a few free configurations may actually lie near the narrow passage and sampling in those areas is important.

Figure 1 shows three examples of the regions in the two-dimensional configuration space. The grey areas are the obstacles. Dots and lines form a roadmap. The black dots belong to the region R under investigation and the white dots belong to the

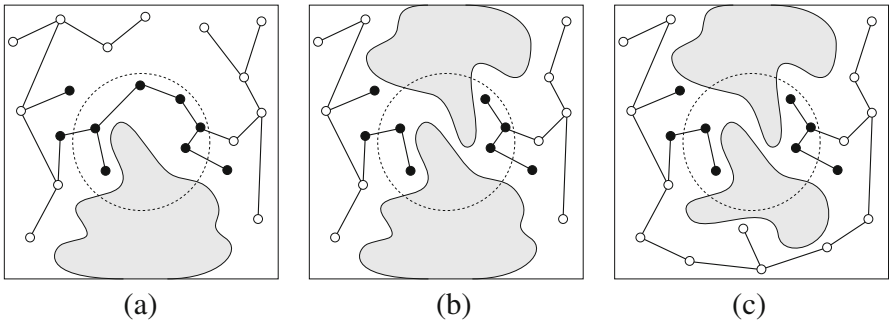


Fig. 1 Three examples of the regions in the configuration space. *Dots* and *lines* form a roadmap and all *black dots* belong to the same region. In (**a**), the *black dots* belong to the same connected component. In (**b**), there is a narrow passage in the region and the *black dots* are no longer in the same component. In (**c**), there is still a narrow passage but this time the *black dots* are in the same component

other regions. The center of the circle is R_{cent} and the radius of the circle is R_{dist}. This hypersphere depicts the area where most of the configurations of the region are generated. Let us assume that in each example the counter R_{free} has reached the `freelimit` parameter.

In Fig. 1a, the configuration space has one obstacle. The roadmap has been generated around the obstacle and all configurations of the region belong to the same component. The region can be classified as solved. In Fig. 1b the configuration space has two obstacles and a narrow passage. The roadmap consists of two connected components: there is no path through the narrow passage. In this case, the region is considered difficult and it can not be classified as solved. This region would be sampled until a path that connects the components is found through the narrow passage.

The example in Fig. 1c is rather similar to the one in Fig. 1b. There are two obstacles, a narrow passage and no path through it. However, in this example the roadmap has only one connected component since there is another path that connects the roadmap. In this case all configurations of the region R also belong to the same component and so the region can be classified as solved. This example shows that the connectivity can be a powerful way in deciding whether the region should be sampled

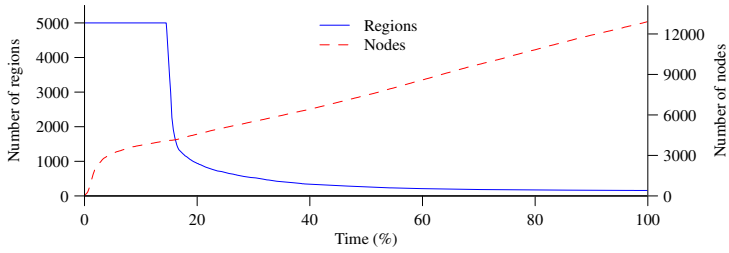


Fig. 2 This shows how the region count and the roadmap size change during the planning

or not. Our planner does not try to find a way through the narrow passage since it detects it would be unnecessary in this case.

Figure 2 shows how region count and roadmap size vary while the algorithm solves a typical motion planning problem. It can be seen from the figure that first the node count increases rapidly. After a while, the algorithm starts to ignore some of the sampled configurations and the node count starts to increase more steadily. The figure also shows how the region count changes during the planning. First the region count does not change at all. After enough sampling is done, the algorithm is able to classify some of the regions as solved and remove them. This leads to a rapid decrease of the region count. Then the algorithm continues with the regions that are left and removes them when the algorithm can conclude that the region is solved.

4.2 Further Improvements

There are only two parameters that must be set with our algorithm. One is the initial count of the regions at the beginning of the planning. The other parameter is the number of free sampled configurations after which the region can be classified as solved if all configurations in the region are in the same connected component. These parameters must, in the current implementation, be set manually, but it might be possible to select these adaptively during the planning.

Our algorithm has two main parts that enhance the speed of the planning compared with the original probabilistic roadmap method. First, it restricts the node count by removing such that are not probably going to increase the connectivity of the roadmap. Secondly, it tries to bias the sampling to those areas that are the most difficult. It intelligently recognizes easy areas from the work space and samples only a few configurations from those areas. However, the node count could further be decreased if we remove unnecessary nodes at the same time we remove the region. Unnecessary nodes can be those that do not increase the connectivity of the roadmap.

In the current implementation, we use the normal distribution to sample the configurations near the center of the region. However, it might be better if more sophisticated sampling methods are used. We could, for example, use bridge test sampling or some other sampling method in addition to the current method. This could be done, for example, by using the chaining method that was proposed in [26]. It would mean that the configuration our method generates is used as an input configuration to some other sampler.

One important part of our algorithm is to generate the regions in the beginning. We used Hammersley quasirandom sequence [4] to generate region centers that are distributed with a low discrepancy. In our tests, this seems to work quite well. One advantage of using Hammersley sequence is that it is easy to implement and it works the same way with all environments. However, there could be more intelligent ways to select the regions even better. It could also be possible to add regions to the difficult areas during the planning.

5 Experiments

We compared our method with two other motion planning methods: the bridge test sampling and the Gaussian sampling. We created three versatile environments where

we tested the planners. Each environment consists of obstacles and a robot that must move from the given initial configuration to the given final configuration. Each planner was run until the roadmap was large enough to solve the predefined query. All tests were run 100 times and the results were averaged to minimize the effect of the outliers caused by the probabilistic nature of the tested algorithms. We generated the regions in our method by using the Hammersley sequence. However, to eliminate the possible bias, we randomly choose the starting index of the sequence. This way the regions were generated in different positions in every test.

All used planners have some parameters that must be set manually before the planning. To be able to select the best parameters, we made test runs with different parameters and selected those parameters that gave the best results. These parameters were then used in the actual experiments. The parameters were different for each environment. We also tested neighbor selection strategies of two different kinds with the Gaussian and the bridge test planners. First we tried to select all neighbors that are at some predefined distance from the configuration and then we tried to select only k nearest neighbors. We noticed that the performance was the best when we selected only k neighbors and so we used that strategy in our tests.

We implemented all of the planners to the same motion planning framework created. This way the test results are comparable. The framework was programmed with C++ and compiled with gcc. All experiments were run on a computer that run Linux and had 1.6 GHz Atom processor with 2 GB memory. The collision detection was done by a PQP library [18].

5.1 Environments

We used three environments to test the planning algorithms. All of these are different and have their own characteristic features even though all environments have a rigid-body robot that has six degrees of freedom.

Spring environment (Fig. 3) consists of two walls and a spring between them. The robot is a small torus and the spring goes through the hole of the torus. The robot has been made of 1,200 triangles and the obstacles have 2,020 triangles total. The work space contains large open free spaces but the robot is unable to move there because the spring greatly limits the movements of the robot.

Walls environment (Fig. 4) has three walls that divide the environment into four regions. There is a small hole in each one of the walls and a hook-shaped robot

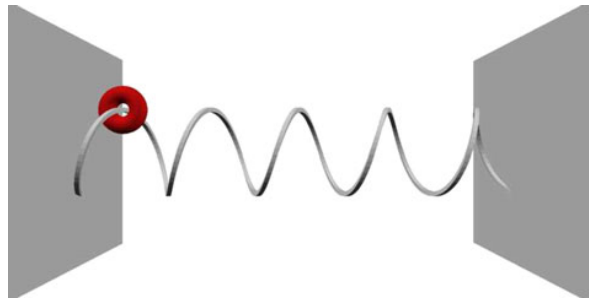


Fig. 3 The spring environment

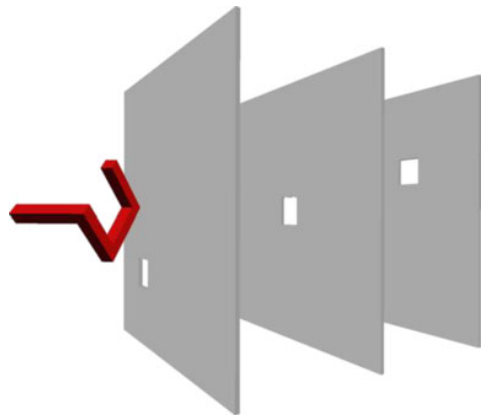


Fig. 4 The walls environment

must go through all of these holes. The robot includes 48 triangles and the obstacles include 144 triangles total. This environment also has large open areas but unlike the spring environment, the robot is able to move freely in this environment. The difficulty comes from the small holes that the robot must pass. The robot must use rotations extensively to be able to go through the holes.

Asteroids environment (Fig. 5) has 750 "asteroids" and a "spaceship" robot that must find a way through these asteroids without touching them. The asteroids are all in different shapes and sizes and there are many possible ways for the robot to move. The robot includes 120 triangles and the obstacles include 60,000 triangles total. This

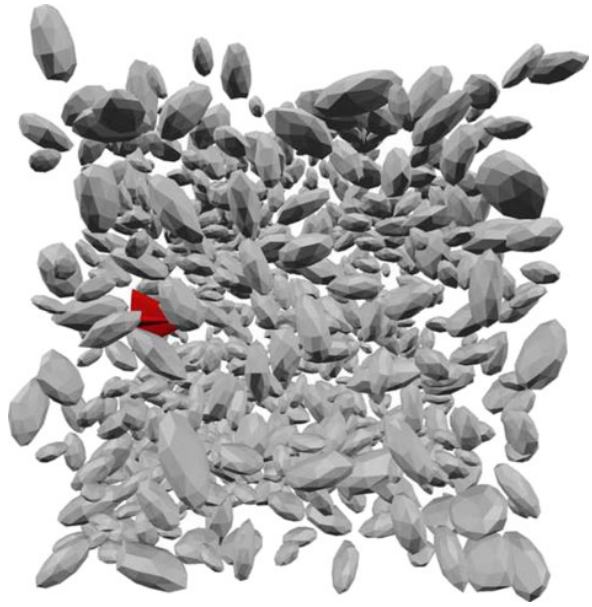


Fig. 5 The asteroids environment

environment does not contain large open areas at all. Instead, it is cluttered with obstacles of different sizes.

5.2 Results

We tested our algorithm as well as the Gaussian sampling and the bridge test sampling with all three environments. Box plots in Figs. 6, 7 and 8 show the running times of the planners in all three environments. The left side of the box shows the lower and the right side the upper quartile. The line in the box shows the median. The whiskers represent the minimum and the maximum values that are within 1.5 times the interquartile range from the box. The values that are not included between the whiskers are outliers and they are plotted as small dots.

As can be seen from Figs. 6–8, our method outperforms both the bridge test sampling and the Gaussian sampling in all three tested environments. The difference was the greatest in the walls environment where our method was about ten times faster than the other planners. In the asteroids environment, the difference was the smallest but still noticeable.

The walls environment contains large empty areas between the walls. Our algorithm quickly detects those areas and stops sampling configurations from there. This is why our algorithm is able to find a path through the holes much faster than the

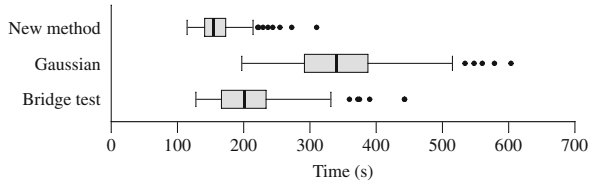


Fig. 6 The *boxplots* for the spring environment

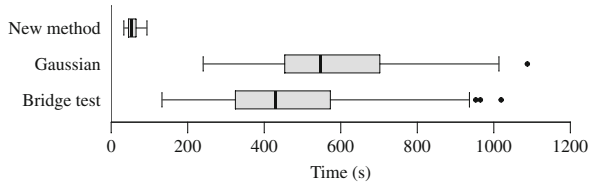


Fig. 7 The *boxplots* for the walls environment

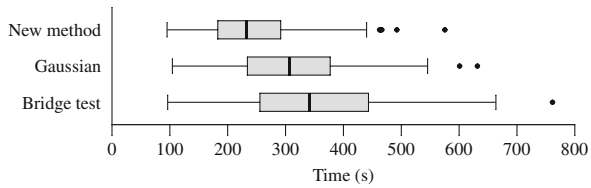


Fig. 8 The *boxplots* for the asteroids environment

other two algorithms. Both the bridge test sampling and the Gaussian sampling are unable to efficiently bias sampling towards the holes. They would continue to sample configurations from the open space even though both samplers would ignore most of the sampled configurations.

The spring environment is more challenging for our method since the empty area is smaller than in the walls environment. The environment is also tough because the robot must move at the narrow passage all the time. Nevertheless, our method is still much faster than other samplers. The Gaussian sampler is significantly slower than our method or the bridge test sampler.

The asteroids environment was generated randomly. There are many obstacles of different sizes cluttered throughout the work space. There are no large empty areas anywhere but there is still enough room for the robot to move from the start configuration to the goal. Yet, our method was the fastest method even though the difference between our method and the other methods were smaller than with the two other tested environments. One thing that should be noticed is that the Gaussian sampler is faster than the bridge test sampler in this environment. One reason may be that the bridge test sampler tries to build one-sized bridges even though there is a huge difference between the narrow passages in this environment.

Tables 1, 2 and 3 show the average running times and the average node counts of the roadmap at the end of the planning of each planner in each environment. Tables 1–3 also show the standard deviations for the times and the node counts. According to the results, it seems that our method can constantly beat the Gaussian and the bridge test samplers.

The size of the roadmap seems to have impact on the planning time since, for example, the time required to get the nearest neighbors increases as the roadmap grows. According to our results, it seems that especially the Gaussian sampler generates much larger roadmaps than our method or the bridge test sampler. This

Table 1 Average running times and node counts for the spring environment

	Time (s)		Node count	
	Mean	Std	Mean	Std
New method	161.4	33.6	15,156.3	2,584.8
Gaussian	350.6	83.5	59,393.9	14,579.3
Bridge test	212.6	62.4	6,882.9	2,062.8

Each test was run 100 times

Table 2 Average running times and node counts for the walls environment

	Time (s)		Node count	
	Mean	Std	Mean	Std
New method	55.6	13.2	5,564.3	1,150.8
Gaussian	580.7	189.7	85,475.7	28,657.6
Bridge test	470.9	188.2	19,954.5	8,039.0

Each test was run 100 times

Table 3 Average running times and node counts for the asteroids environment

	Time (s)		Node count	
	Mean	Std	Mean	Std
New method	248.1	93.0	3,338.8	1,105.1
Gaussian	312.9	109.8	7,015.5	2,709.5
Bridge test	355.0	135.0	3,762.0	1,586.1

Each test was run 100 times

is also probably one major reason why the performance of the Gaussian sampler is quite poor in the spring and walls environments as can be seen in Tables 1 and 2.

The test results show that our method was the fastest in all three environments. The Gaussian sampler was the slowest in two environments and the bridge test sampler was the slowest in one environment. Our method also generated smaller roadmaps than other planners except in spring environment where the bridge test sampler generated the smallest roadmap.

6 Conclusions and Future Work

In this paper, we have presented a novel algorithm that enhances the basic probabilistic roadmap method significantly. Our algorithm tries to recognize easy and difficult areas of the configuration space during the planning by checking the connectivity of the regions. By ignoring the easy areas, our algorithm can sample only the difficult areas that most likely contain narrow passages. This way our algorithm is capable to efficiently solve motion planning problems in difficult environments.

We tested our algorithm with two state-of-the-art planners and noticed that our algorithm works significantly faster than the others. Our algorithm is also capable to limit the size of the roadmap without losing the coverage or connectivity of the roadmap which is useful since it will help to increase the performance of the planner greatly. The test results confirmed that our method indeed constructed small roadmaps.

Despite our algorithm works well in our tests, there may still be possible ways to boost our algorithm further. For example, our algorithm samples the configurations near the center of the regions quite randomly. It might be feasible to use more sophisticated sampling methods, like bridge test, inside each region. We could also try to add regions to the difficult areas during the planning instead of just removing them. This could increase the efficiency in narrow passages.

In our tests we only used rigid-body robots that had six degrees of freedom. It would be interesting to test our algorithm with more complex robots. One example would be articulated robots that move in high-dimensional configuration spaces.

References

1. Amato, N.M., Bayazit, O.B., Dale, L.K., Jones, C., Vallejo, D.: OBPRM: an obstacle-based PRM for 3D workspaces. In: Proc. Int. Wkshp. on Alg. Found. of Rob. (WAFR), pp. 155–168 (1998)
2. van den Berg, J.P., Overmars, M.H.: Roadmap-based motion planning in dynamic environments. IEEE Trans. Robot. **21**(5), 885–897 (2005)
3. Boor, V., Overmars, M.H., van der Stappen, A.F.: The Gaussian sampling strategy for probabilistic roadmap planners. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 1018–1023 (1999)
4. Branicky, M., Lavalle, S., Olson, K., Yang, L.: Quasi-randomized path planning. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 1481–1487 (2001)
5. Canny, J.F.: The Complexity of Robot Motion Planning. MIT Press, Cambridge, MA (1988)
6. Cortés, J., Siméon, T., Ruiz De Angulo, V., Guieysse, D., Remaud-Siméon, M., Tran, V.: A path planning approach for computing large-amplitude motions of flexible molecules. Bioinf. **21**(1), 116–125 (2005)
7. Esteves, C., Arechavaleta, G., Pettré, J., Laumond, J.-P.: Animation planning for virtual characters cooperation. ACM Trans. Graph. **25**(2), 319–339 (2006)

8. Geraerts, R., Overmars, M.H.: Sampling and node adding in probabilistic roadmap planners. Robot. Auton. Syst. **54**, 165–173 (2006)
9. Geraerts, R., Overmars, M.H.: The corridor map method: a general framework for real-time high-quality path planning. Comput. Anim. Virtual Worlds **18**, 107–119 (2007)
10. Geraerts, R., Overmars, M.H.: Reachability-based analysis for probabilistic roadmap planners. Robot. Auton. Syst. **55**(11), 824–836 (2007)
11. Holleman, C., Kavraki, L.E.: A framework for using the workspace medial axis in PRM planners. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 1408–1413 (2000)
12. Hsu, D., Jiang, T., Reif, J., Sun, Z.: The bridge test for sampling narrow passages with probabilistic roadmap planners. In: Proc. IEEE Int. Conf. Rob. Autom. (2003)
13. Hsu, D., Latombe, J.-C., Kurniawati, H.: On the probabilistic foundations of probabilistic roadmap planning. Int. J. Rob. Res. **25**(7), 627–643 (2006)
14. Hsu, D., Latombe, J.-C., Motwani, R.: Path planning in expansive configuration spaces. Int. J. Comput. Geom. Appl. **9**(4 & 5), 495–512 (1999)
15. Hsu, D., Sánchez-Ante, G., Sun, Z.: Hybrid PRM sampling with a cost-sensitive adaptive strategy. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 3885–3891 (2005)
16. Jiménez, P., Thomas, F., Torras, C.: Collision detection algorithms for motion planning. In: Laumond, J.-P. (ed.) Robot Motion Planning and Control, pp. 1–53. Springer, Berlin (1998)
17. Kavraki, L.E., Svestka, P., Latombe, J.-C., Overmars, M.H.: Probabilistic roadmaps for path planning in high-dimensional configuration spaces. IEEE Trans. Rob. Autom. **12**(4), 566–580 (1996)
18. Larsen, E., Gottschalk, S., Lin, M.C., Manocha, D.: Fast proximity queries with swept sphere volumes. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 3719–3726 (2000)
19. Latombe, J.-C.: Robot Motion Planning. Kluwer Academic Publishers, Boston, MA (1991)
20. Lien, J.-M., Thomas, S.L., Amato, N.M.: A general framework for sampling on the medial axis of the free space. In: Proc. IEEE Int. Conf. Rob. Autom. (2003)
21. Reif, J.H.: Complexity of the mover's problem and generalizations. In: Proc. IEEE Symp. Found. Comput. Sci., pp. 421–427 (1979)
22. Rodríguez, S., Thomas, S., Pearce, R., Amato, N.M.: Resampl: a region-sensitive adaptive motion planner. In: Proc. Int. Wkshp. on Alg. Found. of Rob. (WAFR), pp. 285–300 (2006)
23. Saha, M., Latombe, J.-C., Chang, Y.-C., Prinz, F.: Finding narrow passages with probabilistic roadmaps: the small-step retraction method. Auton. Robots **19**(3), 301–319 (2005)
24. Tapia, L., Thomas, S., Amato, N.M.: A motion planning approach to studying molecular motions. Commun. Inf. Syst. **10**(1), 53–68 (2010)
25. Tapia, L., Thomas, S., Boyd, B., Amato, N.M.: An unsupervised adaptive strategy for constructing probabilistic roadmaps. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 2300–2307 (2009)
26. Thomas, S., Morales, M., Tang, X., Amato, N.M.: Biasing samplers to improve motion planning performance. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 1625–1630 (2007)
27. Wilmarth, S.A., Amato, N.M., Stiller, P.F.: MAPRM: a probabilistic roadmap planner with sampling on the medial axis of the free space. In: Proc. IEEE Int. Conf. Rob. Autom., pp. 1024–1031 (1999)
28. Zhang, L., Kim, Y.J., Manocha, D.: A hybrid approach for complete motion planning. In: Proc. IEEE/RSJ Int. Conf. Intell. Rob. Syst. (IROS), pp. 7–14 (2007)

Publication III

Using probabilistic roadmaps in changing environments

Mika T. Rantanen and Martti Juhola

COMPUTER ANIMATION AND VIRTUAL WORLDS
Comp. Anim. Virtual Worlds 2014; **25**:17–31
Published online 18 June 2013 in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/cav.1528

RESEARCH ARTICLE

Using probabilistic roadmaps in changing environments

Mika T. Rantanen* and Martti Juhola

Computer Science, School of Information Sciences, University of Tampere, FI-33014, Finland

ABSTRACT

In this paper, we examine how a path planning problem can be solved in changing environments using probabilistic roadmap planners. A probabilistic roadmap is built in static environment where all obstacles are known in advance, but we show that a roadmap can be built in such a way that it works well even when new obstacles are added to the workspace. However, our experiments show that the roadmap graph must be built carefully. We compare three different methods that are used to decide which edges are added to the roadmap graph to connect the nodes. One of these is a distance-based method, which we present in this paper. In the tests, we built a roadmap by using only the static obstacles. Then, we added additional obstacles to the environment and tested how well the roadmap still worked. The tests showed that our distance-based method worked quickly and that it produced roadmaps, which could be used to find a path amid additional obstacles with a high success rate. Copyright © 2013 John Wiley & Sons, Ltd.

KEYWORDS

path planning; probabilistic roadmaps; changing environments; obstacle avoidance

***Correspondence**

M.T. Rantanen, Computer Science, School of Information Sciences, University of Tampere, FI-33014, Finland.
E-mail: mika.t.rantanen@uta.fi

1. INTRODUCTION

The path planning is an important problem in robotics, and it has been studied in the past decades extensively (e.g., [1–3]). Besides robotics, it has nowadays important applications also in other areas. For example, computer games can use the path planning to guide digital characters to move through the virtual world (e.g., [4]).

The piano mover's problem is the classic path planning problem [5] where the goal is to find a path for a free-flying piano from one location to another. The piano moves in a room, and it has to avoid obstacles while moving. In the basic path planning, the environment where a character moves is static. It means that obstacles do not move or change. It is also assumed that their shape and locations are fully known in advance.

In the path planning, the location of a moving character, for example, the piano, is usually represented as a *configuration* [6]. The *configuration space* $\mathcal{C}$ is a space of all possible configurations that the character can have. In the case of a free-flying piano, one configuration has six parameters: three to represent the position and three to represent the orientation. Therefore the configuration space is also six-dimensional, and the piano has six *degrees of freedom*. A moving object can be more complex than a piano. It can, for example, be a robot arm that has a set of joint angles. In that case, the configuration space would be different, and its dimensionality could be much higher than six.

A configuration is said to be *free* if a moving character does not collide with the obstacles or itself in that configuration. The set of all free configurations is a *free configuration space* $\mathcal{C}_{\text{free}}$. The objective of the path planning is to find a continuous *free path* that lies in $\mathcal{C}_{\text{free}}$ from the initial configuration to a goal configuration.

The general path planning problem has been shown to be PSPACE-complete [7,8], which means that algorithms that can achieve the exact solutions are rarely useful. That is why many heuristic methods have been proposed over the years. In this paper, we are interested in *probabilistic roadmap* (PRM) planners [9–11], which have been shown to work well even in difficult environments.

The PRM planners can solve difficult problems, but still they are fast and quite easy to implement, for example, in games. They build a simplified representation of $\mathcal{C}_{\text{free}}$ called a *roadmap*. It is a graph, and its nodes are randomly selected configurations from $\mathcal{C}_{\text{free}}$. Edges between the nodes indicate that there is a simple and free path between the corresponding configurations. Most of the time-consuming work is carried out during the construction of the roadmap. After that, the roadmap can be used to solve many path planning queries very quickly.

There are many different applications that use roadmaps to guide characters. PRM planners are often designed to work only in static environments where all obstacles are known in advance. However, some roadmap-based applications can handle dynamic obstacles, for example, by repairing or expanding the roadmap when needed. In all cases, an initial roadmap must be built at the beginning. Therefore, it is important that this roadmap is built as good as possible and that the connectivity of the roadmap does not break easily when there are changes in the environment. There has been a lot of research on the probabilistic roadmaps but not so much on how the planners should actually select which nodes to connect together during the construction. In this paper, we investigate that issue. Our findings can be applied to many existing roadmap-based applications because all of them must build the roadmap at the first place.

We compare different node selection strategies by examining how well probabilistic roadmaps work in *changing environments* where some obstacles may appear to the workspace after the roadmap has been built. By *static obstacles*, we mean those that are on the workspace always and which are used to construct the roadmap. By *additional obstacles*, we mean those that can appear to the workspace after the roadmap has been constructed. We show that a roadmap can be built in such a way that it can handle these kinds of changes in an environment and that the most important is to add cycles to a roadmap. We show that additional edges that create the cycles must be selected carefully because adding them randomly does not yield good results. We test empirically three different strategies to select nodes between which edges are added. One of these is a *distance-based method* to be presented in this paper. To our knowledge, this is the first paper that compares how different node selection and edge adding strategies work in changing environments.

2. RELATED WORK

One way to solve the path planning problem is to divide an environment into a grid. Some cells of the grid will be blocked with an obstacle, whereas the others are free. A path can then be found from the grid using some graph search algorithm like A*. This method is very popular in computer games where it is usually used in two-dimensional surfaces [4]. Unfortunately, this method is computationally expensive in large and complex environments and especially when it is used in high-dimensional environments.

There are alternatives for the basic grid method, and many of these are at least partly based on probabilistic roadmaps or similar methods. PRM planners have been studied extensively, but the research has mostly concerned only path planning in static environments. There have been papers about how to enhance sampling (e.g., [12–15]) or how to reduce the size of the roadmap (e.g., [16,17]). Some papers have presented ideas to combine several samplers together (e.g., [18,19]). Some proposed enhancements for probabilistic roadmaps have focused on the virtual worlds such as computer games (e.g., [20]). Usually, these methods have made quite strict restrictions on environments and obstacles. For example, the methods may require that characters move only in two-dimensional planes.

Only a few papers have studied how roadmap nodes should be connected together and how possible cycles should be added to the roadmap. Cycles in a roadmap would allow characters to have many alternative paths, which would be helpful in dynamic environments, for example. In [21], one method for adding useful cycles to roadmap graph was presented. The method adds cycles to the roadmap while still keeping the roadmap building time reasonably low. The main goal was to improve the roadmaps in such a way that the paths found were shorter than with normal roadmaps. However, the method was tested only in static environments. We use this method in our experiments with changing environments.

The method presented in [22,23] uses probabilistic roadmaps in changing environments. It adds cycles to a roadmap but only when necessary. Unfortunately, this method requires the knowledge of the moving obstacles during the preprocessing phase. The algorithm needs to know in advance all moving obstacles and also all possible locations for them. This limits the usefulness of the algorithm. However, the algorithm can guarantee that the path is found if this exists even in the presence of moving obstacles.

The method presented in [24] is also based on the PRM planners. First, it constructs a normal roadmap without cycles using only static obstacles. During the query phase, it finds a path and checks whether moving obstacles block some of the edges of the path. The method uses lazy evaluation [25] to check which edges are blocked and which are not. This helps the method to work faster because it needs to check only relevant edges. If edges are blocked, it tries to reconnect the nodes of the blocked edges by using a method based on the rapidly exploring random tree techniques [26,27]. If that fails, new nodes will be inserted into the roadmap. By adding edges to the roadmap, the method also adds cycles. The method does this during the query phase even though it would be better to do most of the time-consuming calculations during the preprocessing phase.

Another roadmap-based method for dynamic environments is presented in [28]. The method constructs an initial roadmap and then uses it to handle multiple moving characters. When some edge becomes blocked, the method tries to reconnect that edge after a while. The method can also restore the connectivity of the roadmap by sampling new nodes and adding them to the roadmap like the method presented in [24]. Just like other roadmap-based methods, also this method depends on the roadmap that was built in the beginning. Therefore, it is essential that the initial roadmap has been constructed well.

Other approaches to path planning have been suggested as well. For example, in [29], a method that combines

DOI: 10.1002/cav

navigation meshes [30] with probabilistic roadmaps is presented. In their method, characters are supposed to move mainly on two-dimensional surfaces. For each surface, they build a roadmap that can be used to guide a character on that surface. The surfaces are connected together, and a path can also be searched for between them. The problem in their method is that the dimension of the configuration space becomes quickly too large because in this method, all moving obstacles increase the dimensionality. That is why they improve the performance by simplifying the moving characters by using bounding volumes and by constraining their movements mainly to the floor.

3. CHANGING ENVIRONMENTS AND PROBABILISTIC ROADMAPS

In this section, we look how probabilistic roadmaps can be used with changing environments. A basic PRM planner is shown in Algorithm 1. The algorithm tries to build a roadmap in such a way that it *covers* $\mathcal{C}_{\text{free}}$ as well as possible. The roadmap should also have the good *connectivity*, which means that two arbitrary nodes should belong to the same component in the roadmap graph if there is a free path in $\mathcal{C}$ between the corresponding configurations.

Algorithm 1 Constructs the roadmap $G = (V, E)$.

```
1:  V ← ∅
2:  E ← ∅
3:  repeat
4:      q ← randomly generated configuration from C_free
5:      V ← V ∪ {q}
6:      N_q ← all neighbor configurations of q chosen from
        V
7:      for all q′ in N_q do
8:          if q and q′ are not in the same component of the
            roadmap then
9:              if local planner finds a path between q and q′
                then
10:                 E ← E ∪ {(q, q′)}
11:         else
12:             Decide whether an additional edge (q, q′)
                should be added
13: until there are enough configurations in V
14: return G
```

In line 4, a new configuration q is sampled from the free configuration space $\mathcal{C}_{\text{free}}$ randomly. In line 6, neighboring configurations are selected for the sampled configuration. Then, the algorithm goes through all the neighbors. In line 8, it is checked whether the sampled configuration and the neighbor configuration belong to the same component in the roadmap. If they do not belong, it is checked in line 9 if there is a simple straight-line free path between them. If there is, an edge is added to the roadmap between the corresponding configurations in line 10.

The neighboring configurations that are selected in line 6 are usually the k nearest configurations that have at most a distance r from the configuration q. The problem is to decide which values are the best for the parameters k and r. That issue is investigated in [31,32].

Line 12 makes it possible to add additional cycles to the roadmap. If the line is ignored, the algorithm will produce a roadmap that contains one or more trees. For performance reasons, probabilistic roadmaps are often built so that there are no cycles in a roadmap graph. It means that the roadmap can have only one path between two nodes. This is a problem in changing environments, but it also causes the paths to be unnecessary long. It is possible to try to smooth the path afterwards, but it requires more computation time.

3.1. Cycles in Probabilistic Roadmaps

The reason that most PRM planners do not add cycles is purely because of the performance. When the algorithm tries to add a new edge, it must check if it is collision free. Detecting collisions is quite heavy operation [33], so it is time-consuming to add edges. Too many edges also use memory unnecessarily. The additional edges do not increase the coverage or the connectivity of the roadmap, so it is possible to add only a minimum number of edges if the goal is just to find some path between the nodes amid static obstacles.

The cycles in the roadmap are however very useful when the environment can change after the roadmap has been built. The character can then use alternative paths to circumvent the additional obstacles and still use the same roadmap without a need to rebuild or modify it.

We say that an edge in the roadmap is *blocked* if there is an additional obstacle that would cause a collision if the character moved along that edge. In the same way, a node in the roadmap is blocked if the character collides with an additional obstacle in the corresponding configuration.

Figure 1 shows an example of a roadmap in a changing environment. In Figure 1(a), the roadmap is built without any cycles. The roadmap is therefore a tree, and there exists only one path between the nodes in the upper left and bottom right corners. In Figure 1(b), a new obstacle is added to the environment, and it disconnects the roadmap graph so that it is not possible to find a path between those nodes anymore.

In Figure 1(c), the roadmap is built with additional edges. The added obstacle does not disconnect the graph because there are multiple paths between the nodes and the character can choose any one of them. It should also be noted that usually the additional edges make it possible to find a shorter path for the character. This is the case also in this example: The found path is shorter in Figure 1(c) than it is in Figure 1(a).

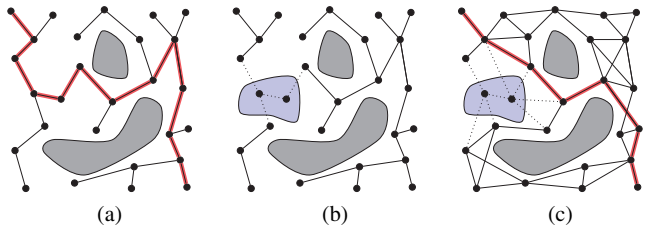


Figure 1. An example of a roadmap on a two-dimensional configuration space. In (a), the roadmap has been built without cycles, and a path between two nodes has been found. In (b), a new obstacle has been added to the environment. The free path cannot be found anymore. In (c), the roadmap has been built with additional edges. Now, the path can be found even though the added obstacle still blocks some of the edges and nodes. The path is also shorter than in (a).

3.2. Advantages

Many proposed path planning algorithms make specific requirements for characters and obstacles. Some algorithms require that characters move in two-dimensional planes (e.g., [29]), and some make restrictions on where dynamic obstacles can appear (e.g., [22,23]). Some algorithms may require that obstacles and the moving character are simplified with a bounding volume because the performance would otherwise be too poor. These restrictions make many proposed algorithms unusable in many situations even though they can work very well if the restrictions are accepted.

The probabilistic roadmaps, however, do not have such restrictions. All obstacles and moving characters can have any size and shape, and they can be located anywhere. It is not necessary to know beforehand what kinds of additional obstacles there will be. A roadmap is built only on the basis of static obstacles. Additional cycles can be added to the roadmap, which will help the roadmap to keep the connectivity even when additional obstacles appear to the environment.

Many path planning algorithms work well in easy environments, but the strength of probabilistic roadmaps is that they can work quite well even in complex environments. There are also a vast number of different techniques that are developed to overcome the difficulty that narrow passages present for path planning. Furthermore, the objects do not need to be simplified, and they do not need to be rigid bodies. In this paper, our experiments are made with characters that move freely in three-dimensional space; that is, our characters have six degrees of freedom. Probabilistic roadmaps, however, work nicely also with smaller and higher degrees of freedom [11]. The moving character could be, for example, an articulated robot arm.

Our approach will not handle all possible additional obstacles that can be added. It is possible that a PRM planner fails to find a path after additional obstacles have been added to the workspace. In that case, a roadmap can be expanded, and there are several ways to do it. For example, it is easy to add single nodes to the roadmap just by running the main loop in Algorithm 1 once. This can be carried out quite quickly if only a few edges are added to connect the node to the roadmap. The techniques presented in [24] could also be used to provide a way to expand the roadmap.

4. NODE SELECTION STRATEGIES

One problem in probabilistic roadmaps is to decide which nodes to connect together. The simplest method would check all possible edges between all roadmap nodes and then add free edges. However, that would not be feasible in practice because there would be $n(n-1)/2$ edges to check if n is the number of nodes. Checking all of them for collision would take too long time, and at the end, there would be too many edges in the roadmap.

To reduce the number of edges, one could try to add edges from each node to only its k nearest neighbors. If k is small enough, the number of edges would also be small. Unfortunately, this does not give good results as we will show in this paper. The impact of different k values was experimentally tested in [32] with roadmaps that did not have cycles. Their experiments showed that too small a value for k will have negative impact on the connectivity of the roadmap. They obtained good results in their experiments with a value of 75 for k. However, when cycles are allowed, too large values will decrease the performance quite quickly because the number of edges would grow and each one of them must be checked for possible collisions. This means that some more sophisticated node selection strategy must be used.

The most important thing that must be taken care of is to try to keep a roadmap connected even when additional obstacles are added to an environment. Even if some of the edges and nodes are blocked, we want that there would still be a path between two arbitrary nodes if there was a path between them before the additional obstacles appeared. Of course, this is not always possible, but if we construct the roadmap correctly, we can try to maximize the possibility that the connectivity does not change.

Figure 2 shows an example of the blocked edges. Both Figure 2(a) and Figure 2(b) show the same environment where there are static obstacles in the upper and bottom

DOI: 10.1002/cav

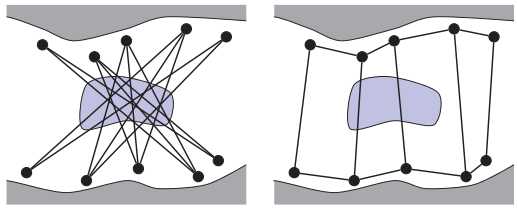


Figure 2. An example of blocked edges. Both (a) and (b) have similar static obstacles at the top and bottom and an additional obstacle in the middle. In both, a roadmap has been constructed. The only difference is how the nodes are connected to each other. In (a), the additional obstacle blocks all of the edges. In (b), it blocks only two edges and does not break the connectivity of the roadmap.

part and an additional obstacle in the middle. There are also same roadmap nodes that are marked with small dots. The only difference between Figure 2(a) and Figure 2(b) is the edges of the roadmap, which connect the nodes together. Both figures have 13 edges, but they are connected in a different way. Still, in both Figure 2(a) and Figure 2(b), it is possible to find a path between any two nodes using the roadmap if the additional obstacle is ignored.

However, if the additional obstacle in the middle is taken into account, there are no free paths between any nodes in Figure 2(a). The additional obstacle blocks all edges. On the other hand, in Figure 2(b), the same additional obstacle only blocks two edges and does not change the connectivity of the roadmap in any way. It is possible to find a path between two arbitrary nodes.

The example in Figure 2 shows that if we want to add cycles to the roadmap, it is important to add edges correctly. The same amount of edges can lead to very different results when the additional obstacles are added to the environment. Node selection strategies have been compared before [31,32], but those comparisons have concentrated on environments with only static obstacles. In this paper, we compare different selection strategies when some additional obstacles might appear to the workspace.

Next, we investigate three different node selection strategies more closely. These methods are used to decide which edges should be added to a roadmap. All methods assume that a set of neighboring nodes have been selected for a certain node. In our tests, we always selected k-nearest nodes at most a radius of r away. Each method will then filter out some of the neighbors. We want to make sure that the connectivity of a roadmap is as good as possible, so we always try to connect two nodes if they belong to the different components in the roadmap.

4.1. Random Method

The simplest way to add additional edges to a roadmap is to add them randomly. It means that we try to add an edge between a sampled configuration and each of its neighbors with some predefined probability. This will allow us to add cycles but still keep the number of the edges low. This kind of method was investigated in [34].

With a probability 0, there will be no additional edges in the roadmap, and with a higher probability, the number of edges increases. When the probability is 1, the algorithm tries to connect the newly sampled configuration to all of its neighboring configurations. The random method is very easy to implement, but in practice, it does not work very well as can be seen in our experimental tests.

4.2. Useful Edges Method

The useful edges method presented in [21] tries to add edges to a roadmap in such a way that the paths returned by the planners are short. It does so by adding useful cycles to the roadmap and thus providing alternative paths between the nodes. However, the method can also be used in changing environments, and as our tests show, it works well. Unfortunately, the method is quite slow especially when it is used with large roadmaps.

The method works as follows. First, nearest neighbors are selected for a newly added configuration q. Then, for each neighbor q', the algorithm calculates the distance d between q and q' as well as the graph distance d_{graph} between the same nodes. The graph distance is the length of the shortest path in the graph. An edge is added between q and q' if $K \cdot d < d_{\text{graph}}$, where K is some predefined constant. By modifying K, it is possible to change the number of the edges. If $K = \infty$, no additional edges will be added to the roadmap. If $K < 1$, the method tries to add an edge to all neighbor configurations.

There is a problem how to calculate d_{graph} efficiently. In [21], they used a modified version of Dijkstra's algorithm to speed up the calculation. The key notion is that it is not necessary to know exact distances. We can stop the shortest path calculation, when we know that d_{graph} must be larger than $K \cdot d$. Modifying Dijkstra's algorithm to support this is easy [21]. We used the same technique in our tests.

4.3. Distance-Based Method

A distance-based method tries to make sure that all nodes are connected to the rest of the roadmap with multiple edges and at the same time ensure that the edges of each node are distributed evenly in different directions. These

properties will decrease the probability that additional obstacles could break the connectivity of the roadmap.

When building a roadmap, the basic PRM planner tries to connect the newly sampled node at least to all neighbor nodes that have degree 0. That is because the planner wants to increase connectivity and nodes with degree 0 will always be in a different component than the rest of the roadmap. In addition, our distance-based method tries to connect the new node q to all those neighbor nodes that have a degree equal to or smaller than a predefined constant DEGLIM. If the roadmap has nodes that have degree 1, there will be only one edge connected to that node. If that edge is blocked by some additional obstacle, the node will not be reachable anymore from other nodes. If the degree is 2, two edges must be blocked to make the node unreachable because the node would still be reachable if one edge were blocked. As the degree grows, the harder it will be to block the node.

When the average degree of the nodes grows, also the density of the graph increases. This is often useful in changing environments because there will be a lot of alternative paths between nodes to circumvent the additional obstacles. However, it is not enough that the degree of the nodes is high as can be seen from Figure 2. Also, the degree cannot be increased too much because the roadmap size would quickly grow too large to be practical. That is why we must limit node connections with the constant DEGLIM.

For those nodes that have a larger degree than DEGLIM, we must use some more sophisticated method. In those cases, the distance-based method will calculate a probability for each neighbor node q_{neigh} that will be used to decide whether the planner will try to add an edge (q, q_{neigh}) to the roadmap.

The probability depends on the edges that the newly added node q already has. We check all the adjacent nodes that q has and calculate their distances from q_{neigh}. Those adjacent nodes that are near q_{neigh} will decrease the probability; on the other hand, those that are far away will increase it. The reason behind this is that it is good if the edges connected to the node are distributed evenly in all directions. That way, it is hard for dynamic obstacles to block all connected edges at the same time.

Figure 3 illustrates how the probability is calculated. There is a node q and its neighbor node q_{neigh}. There are also several adjacent nodes to q and a circle that is drawn around the q_{neigh}. The circle represents the area in which the adjacent nodes will decrease the probability. In other words, the two adjacent nodes that are inside the circle will decrease the probability, and the others will increase it. Algorithm 2 shows in more detail how this works. It can be called from line 12 of Algorithm 1. As input, it requires the roadmap R and a new node q and one of its neighbor configurations q_{neigh}. The algorithm will then decide whether the planner should try to add an edge between those nodes or not.

Algorithm 2 The distance-based method to decide whether the planner should try to add an additional edge between the configurations q and q_{neigh}.

Require: A roadmap $R = (V, E)$
Require: A configuration q and its neighbor configuration q_{neigh}
1: **if** $\deg(q_{\text{neigh}}) >$ DEGLIM **then**
2: $A_q \leftarrow$ all adjacent nodes to q from R
3: **if** $|A_q| > 0$ **then**
4: $p \leftarrow 1$
5: $d_{\text{avg}} \leftarrow$ average length of edges connected to q_{neigh}
6: $d_{\max} \leftarrow$ DISTMULT $\cdot\, d_{\text{avg}}$
7: **for all** $q_{\text{adj}} \in A_q$ **do**
8: $d \leftarrow$ distance between q_{adj} and q_{neigh}
9: $p \leftarrow p \cdot \exp(\frac{d}{d_{\max}}, \text{EXP})$
10: $p \leftarrow \min(p, 1)$
11: $t \leftarrow$ a random number between 0 and 1
12: **if** $p < t$ **then**
13: **return** false
14: **return** true

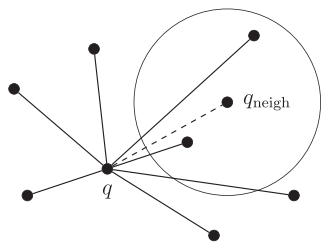


Figure 3. The node q has adjacent nodes, and some of those are located near the neighbor node q_{neigh}. The circle is centered in q_{neigh}, and it represents the area where the nodes decrease the probability that the planner tries to connect q and q_{neigh}. The adjacent nodes outside the circle increase the probability.

In line 1, the degree of q_{neigh} is checked. If it is smaller than or equal to DEGLIM, the algorithm returns true. Otherwise, the algorithm continues to line 2 and retrieves all adjacent nodes to q from R. In line 3, the algorithm makes sure that there are actually some adjacent nodes. If there are not, it means that q is not yet connected to the roadmap. In that case, the algorithm returns true.

In line 4, the probability p is initialized with value 1. In line 5, the average length d_{avg} of the edges connected to q_{neigh} is retrieved. It should be noted that d_{avg} cannot be 0 because of the check we made in line 1.

To decide which nodes are near, we use a predefined constant DISTMULT and the variable d_{avg}. In line 6, the maximum distance $d_{\max} =$ DISTMULT $\cdot\, d_{\text{avg}}$ is calculated. Those adjacent nodes that are nearer than $d_{\max}$ will decrease the probability, whereas the other will increase it. In practice, this means that when DISTMULT grows, there

will be less nodes in the roadmap. In Figure 3, the radius of the circle is $d_{\max}$. We use variable d_{avg} because it allows us to reflect the changes in the roadmap size: As the number of the nodes increases, also the distances between the nodes become smaller. It should be noted that $d_{\max}$ can very well be larger than the distance between q and q_{neigh}.

In lines 7–9, the algorithm goes through all adjacent nodes. In line 8, a distance d between q_{neigh} and one adjacent node is calculated. In line 9, the probability p is updated. In that line, the $\frac{d}{d_{\max}}$ causes the adjacent nodes that are near q_{neigh} to decrease the probability and those that are far away to increase the probability

In line 9, the $\frac{d}{d_{\max}}$ is also raised to the power of `EXP`. By modifying the constant `EXP`, we can control how much each adjacent node actually changes the probability. When the value for `EXP` grows, the adjacent nodes that are near q_{neigh} will decrease the probability more, and similarly, those that are far away will increase the probability more.

In line 10, we make sure that p is between 0 and 1. Lines 11–13 will cause the algorithm to return false with the probability of p. Otherwise, the algorithm returns true. If the algorithm returns true, the planner will try to add an edge between q and q_{neigh} to the roadmap. The edge is added if there is a free path between the nodes. After that, the planner will select the next neighbor, and the algorithm is called again.

5. EXPERIMENTS

We used three different environments in our tests. In each environment, we used three node selection methods, and with each of these, we built roadmaps of six different sizes. The tested node selection methods were the random method, the useful edges method, and the distance-based method. The number of the nodes was fixed, and the edge count was changed by modifying the parameters of each method. Unlike the other methods, the distance-based method has three parameters that can be changed instead of just one. In our tests, we first chose values for parameters `DEGLIM` and `EXP`. Then, we used the parameter `DISTMULT` to tweak the edge count. We tested the distance-based method with several different parameters.

To select neighbors for a configuration q, we selected k nearest nodes from the roadmap that had at most a distance r from q. The values of k and r were predefined constants. To choose good values for these constants, we made some preliminary tests with different values and chose the best ones. We also tested a simple nearest neighbor method, which tries to add an edge to all the nearest neighbors without any additional node selection. For these tests, we selected the k value in such a way that the resulted edge count was comparable with the other methods.

For each environment, we selected two nodes to be used as initial and goal nodes in our path queries. In the tests, we built the roadmaps by using only the static obstacles and then made a predefined query. Then, we added additional obstacles to the environment and checked whether we can still find a path using the previously built roadmap. All additional obstacles were not added to the environment immediately but instead in five incremental steps. For each generated roadmap, we measured how much time it took to build the roadmap and how often the roadmap was able to return a path between the two predetermined locations in the presence of a different number of the additional obstacles. We also measured the lengths of found paths with and without the additional obstacles.

In each environment, all executed tests used the same parameters except those that were directly related to the tested node selection method. In other words, the node sampling, neighbor selection, and local planning worked exactly in the same way in each test. The only difference was the node selection method that was used to add additional edges to the roadmap. For comparison, we also generated roadmaps with the basic PRM planner that does not add any additional edges to roadmaps. In these cases, the built roadmap is therefore a tree or a forest, and the number of edges is smaller than the number of nodes.

In node sampling, we did not use any special method to boost the sampling. Instead, we used a combination of the basic uniform random sampling and the Gaussian sampling [13]. One half of the nodes were generated by using the uniform sampling, and the other half using the Gaussian sampling. This was because we wanted to make use of the static obstacles. The Gaussian sampling provides an easy way to sample near the static obstacles. On the other hand, we wanted to sample also in other areas because we do not know in advance where additional obstacles might appear. That is why we also used uniform sampling. By combining these methods, we were able to obtain good results. However, it could be interesting to see how other sampling techniques would work with additional obstacles. We did not do it in this paper because we were only interested in connection strategies.

5.1. Test Setup

All tested methods were implemented in the same software framework that we had developed to test the robot motion planning. The framework was programmed in C++. Because all of the methods were implemented in the same framework, we were able to compare also the running times of the methods reliably.

The hardware system that we used in our tests was a PC (Dell, Round Rock, Texas, United States) with Intel i7-2600 (3.40 GHz) processor and with 8 GB of memory. The tests were run in Windows 7 (Microsoft, Redmond, Washington, United States). For collision detection, we used PQP library (University of North Carolina at Chapel Hill, Chapel Hill, North Carolina, United States) [35]. Each test was repeated 1000 times to minimize the effect of the randomness.

5.2. Test Environments

The test environments consisted of static and additional obstacles and a moving character. All these objects were

Table 1. Workspace dimensions for each environment and parameters used in the nearest neighbor search.

Environment	Workspace	Parameter k	Parameter r
Asteroids	$200 \times 100 \times 100$	100	50
House	$512 \times 354 \times 220$	50	100
Wall	$200 \times 100 \times 100$	50	50

represented as triangle mesh. Table 1 shows the workspace sizes and the parameters used in the nearest neighbor search for each environment. We used the following three environments:

Wall environment. The moving character is a hook-shaped object that must go through a hole of the wall. The wall is a static obstacle, and the additional obstacles are small blocks on both sides of the wall. The static obstacles consist of 48 triangles, and the additional obstacles 144 triangles in total. The moving character has 48 triangles. The problem is shown in Figure 4.

Asteroids environment. The moving character is a small spaceship that must go through an asteroid field. There are 50 static asteroids and 200 smaller additional asteroids in total. The static obstacles have 16 000 triangles, additional obstacles have 64 000 triangles, and the spaceship has 120 triangles. The problem is shown in Figure 5.

House environment. The house has two floors and stairs between them. The moving character is an upright piano. Its start location is on the first floor, and the goal location above it is on the second floor. The static obstacles consist of walls, floors, and stairs, and they have 500 triangles in total. The additional obstacles contain furniture and doors and a banister, and they have 3898 triangles in total. The piano consists of 519 triangles. The problem is shown in Figure 6.

All environments are different. The wall environment is the simplest because it contains just a wall with a hole and few additional obstacles. However, it is a quite difficult path planning problem even without the additional obstacles. The hole is small, and the character must rotate itself extensively to go through it. This problem could not be solved if the character would have been simplified, for example, with a bounding box.

Unlike the wall environment, the asteroids environment has a lot of additional obstacles. It is easy to find a path without the additional obstacles, but when they are added, it becomes much harder. The environment was generated by adding the asteroids to the environment randomly.

In the house environment, the static obstacles divide the space into the rooms. The planning is quite difficult even without the additional obstacles, and when they are added, it becomes much harder to find a path. This is an important environment to test because similar environments could appear in many actual virtual world applications.

6. RESULTS

Tables 2–4 show results of our experiments. The results are the averaged values of all 1000 test runs, except a success percent. A method column shows the used node selection method. For comparison, we also tested a basic PRM method and a nearest neighbor method, which are shown in the tables. The method column also shows the used parameters for the distance-based method. These are shown in parentheses; D means a parameter `DEGLIM`, and E means a parameter `EXP`.

A node and an edge columns show how large roadmap was built. The number of nodes and edges were predetermined. The number of nodes is a constant, but the number

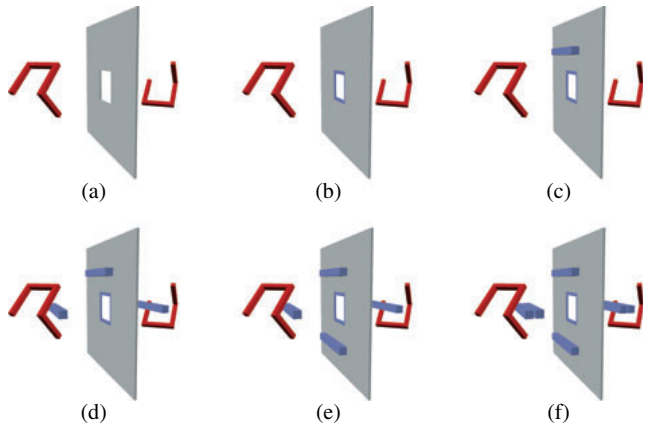


Figure 4. A wall environment. In (a), there are only static obstacles and a moving character in start and goal locations. In (b)–(f), the additional obstacles have been added incrementally.

DOI: 10.1002/cav

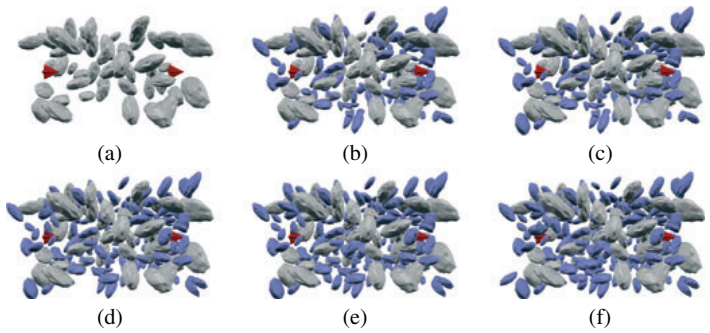


Figure 5. An asteroids environment. In (a), there are only static obstacles and a moving character in start and goal locations. In (b)–(f), the additional obstacles have been added incrementally.

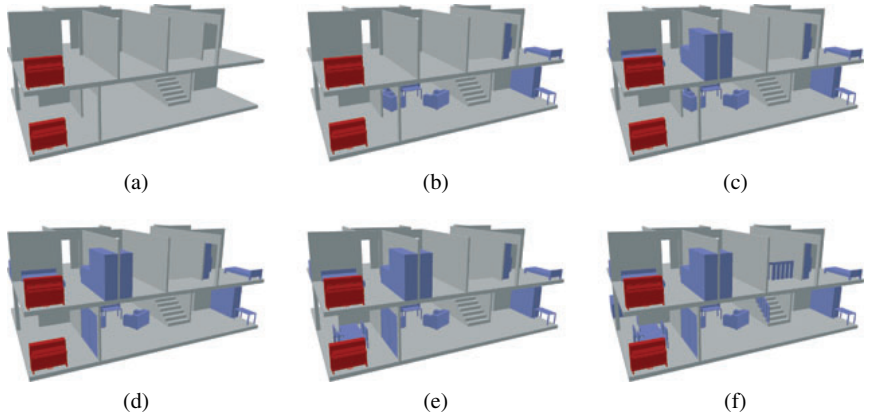


Figure 6. A house environment (shown without the roof and exterior walls). In (a), there are only static obstacles and a moving character in start and goal locations. In (b)–(f), the additional obstacles have been added incrementally.

of edges varies a little between each test run. A time column shows how much time it took to build the roadmap in seconds. There was only very small variation in time between the test runs.

Length columns show how long the shortest free path between two predetermined nodes was. A static path length was measured in an environment that had only static obstacles. The path length with additional obstacles was measured in five steps, and in each step, more obstacles were added to the environment. Corresponding results are shown in columns marked with text Additional 1–5. The success percent column shows how frequently the path was found when there were additional obstacles in the environment. For path lengths, standard deviations are also shown because there was some variation between the test runs.

Table 2 shows results for the wall environment. As can be seen from the results, this is quite a difficult problem especially when all obstacles are added to the environment. Overall, the useful edges method and the distance-based method were able to achieve much better success rates than the random method and the nearest neighbor method. When the speeds were compared, the useful edges method was clearly the slowest. In this environment, the distance-based method had the best results when the parameter `DEGLIM` was as large as possible. This parameter cannot be arbitrarily large because increasing it will also increase the number of edges. We also tested two different values for the parameter `EXP`. Because most of the edges were added because of a large value of `DEGLIM`, the parameter `EXP` did not have much effect on the results.

Table 3 shows results for the asteroids environment. In this environment, we used much smaller and sparser roadmaps than in the wall environment. However, the methods worked quite similarly. If compared by the success rates, the useful edges method and the distance-based method were the best ones. In contrast to the wall environment, the distance-based method had the best results when the parameter `DEGLIM` was small. Now, the

Table 2. Average results of 1000 test runs for wall environment.

Method	Nodes	Edges	Time (s)	Static length	Additional 1		Additional 2		Additional 3		Additional 4		Additional 5	
					Length	SR	Length	SR	Length	SR	Length	SR	Length	SR
Basic PRM	20 000	19 998.6	4.4	359.4 (76.0)	371.5 (114.4)	1	—	0	—	0	—	0	—	0
11 nearest neighbors	20 000	205 995.1	22.5	182.1 (5.7)	204.9 (16.0)	56	207.2 (16.2)	25	263.4 (22.1)	20	272.3 (24.4)	7	—	0
Random	20 000	75 200.4	11.2	205.0 (10.3)	235.9 (22.8)	16	239.1 (19.1)	4	351.4 (60.0)	2	—	0	—	0
Useful edges	20 000	75 017.3	51.8	188.7 (5.3)	218.7 (18.3)	86	226.2 (21.0)	47	327.5 (34.1)	38	353.6 (34.7)	14	386.5 (28.5)	1
Distance-based (D: 5, E: 4)	20 000	75 022.6	14.1	196.7 (6.9)	232.0 (20.3)	90	244.4 (24.9)	57	386.8 (43.6)	41	412.0 (49.1)	12	418.9 (18.6)	1
Distance-based (D: 5, E: 24)	20 000	75 018.2	14.5	196.1 (7.0)	230.2 (19.3)	92	242.1 (24.6)	57	385.3 (44.3)	41	411.5 (43.9)	13	—	0
Random	20 000	150 280.2	20.3	187.4 (6.8)	218.7 (21.5)	46	221.4 (21.4)	17	290.9 (28.2)	12	296.9 (30.2)	4	—	0
Useful edges	20 000	150 168.2	58.6	177.8 (3.8)	199.4 (12.7)	97	207.7 (17.5)	72	263.6 (22.9)	68	272.8 (24.7)	36	301.1 (31.2)	4
Distance-based (D: 5, E: 4)	20 000	150 261.0	24.7	181.6 (4.9)	203.8 (12.6)	98	216.1 (19.5)	78	297.1 (30.0)	70	309.6 (29.0)	32	350.6 (38.5)	4
Distance-based (D: 13, E: 4)	20 000	150 170.5	26.3	183.5 (4.8)	202.1 (9.8)	99	212.0 (15.3)	90	302.4 (27.8)	86	322.6 (30.7)	55	364.4 (43.8)	9
Distance-based (D: 13, E: 24)	20 000	150 052.4	26.5	183.4 (4.9)	202.1 (9.9)	100	213.2 (17.5)	91	303.2 (29.0)	86	320.2 (31.7)	53	350.7 (40.9)	10
Random	20 000	200 230.6	26.4	181.6 (5.2)	209.6 (19.5)	65	214.1 (21.4)	31	273.5 (26.1)	25	282.7 (25.7)	10	300.5 (43.2)	1
Useful edges	20 000	200 174.6	63.9	174.5 (3.4)	193.0 (11.0)	98	201.9 (17.2)	78	248.5 (22.8)	74	255.6 (22.9)	42	283.8 (29.3)	8
Distance-based (D: 5, E: 4)	20 000	200 290.8	31.2	178.5 (4.3)	198.2 (11.1)	99	208.7 (18.2)	85	278.8 (25.9)	79	291.3 (27.6)	46	327.8 (31.6)	7
Distance-based (D: 19, E: 4)	20 000	200 144.0	33.8	180.1 (4.3)	196.5 (8.3)	100	204.8 (13.5)	93	280.0 (25.4)	91	296.5 (28.7)	63	333.4 (36.8)	18
Distance-based (D: 19, E: 24)	20 000	200 201.9	33.9	180.4 (4.6)	196.8 (9.0)	100	205.2 (13.6)	94	280.7 (24.6)	93	295.2 (27.2)	67	331.1 (36.2)	18
Basic PRM	30 000	29 998.4	6.8	361.6 (76.6)	—	0	—	0	—	0	—	0	—	0
8 nearest neighbors	30 000	225 356.2	23.8	187.3 (6.3)	209.5 (15.2)	45	211.9 (15.1)	15	276.9 (21.2)	13	290.2 (25.4)	5	—	0
Random	30 000	75 088.6	12.0	215.9 (11.5)	248.4 (26.6)	12	250.5 (27.8)	4	405.6 (59.2)	1	—	0	—	0
Useful edges	30 000	75 116.1	103.3	196.4 (6.3)	232.4 (19.7)	86	241.3 (22.5)	41	390.2 (42.9)	30	423.5 (49.1)	8	—	0
Distance-based (D: 3, E: 4)	30 000	75 073.8	14.7	211.5 (9.4)	259.7 (26.0)	78	269.5 (31.3)	33	455.2 (61.2)	19	508.3 (59.9)	4	—	0
Distance-based (D: 3, E: 24)	30 000	75 040.5	14.9	209.7 (9.3)	256.1 (24.8)	80	266.0 (28.7)	39	453.7 (52.1)	20	497.0 (59.7)	4	—	0
Random	30 000	150 114.1	20.7	193.4 (7.3)	230.1 (24.9)	42	232.3 (24.1)	16	324.0 (31.9)	10	347.2 (34.6)	3	—	0
Useful edges	30 000	150 107.5	106.3	183.6 (4.3)	205.5 (12.0)	98	215.8 (17.1)	81	287.9 (24.6)	76	305.3 (29.7)	41	342.1 (30.0)	4
Distance-based (D: 5, E: 4)	30 000	150 116.2	25.7	186.5 (5.1)	209.7 (12.4)	100	221.9 (18.3)	87	322.1 (30.5)	80	345.7 (36.7)	41	380.7 (46.0)	4
Distance-based (D: 8, E: 4)	30 000	150 163.2	26.5	190.1 (5.9)	212.5 (10.7)	100	225.7 (17.9)	92	340.0 (33.5)	86	366.3 (38.9)	49	405.8 (43.2)	6
Distance-based (D: 8, E: 24)	30 000	150 142.2	26.7	189.8 (5.6)	212.0 (11.2)	100	224.5 (18.2)	93	340.9 (34.5)	87	365.9 (36.4)	48	407.9 (48.0)	5
Random	30 000	200 259.4	26.6	187.2 (6.0)	220.9 (22.1)	61	224.0 (22.0)	24	299.3 (28.2)	18	309.2 (28.1)	6	—	0
Useful edges	30 000	200 170.7	110.6	179.1 (3.6)	198.6 (9.2)	99	209.1 (16.5)	86	265.6 (21.4)	83	277.9 (25.2)	48	310.8 (28.2)	7
Distance-based (D: 5, E: 4)	30 000	200 142.0	32.1	182.0 (4.8)	201.4 (8.9)	100	212.0 (16.2)	92	294.1 (27.9)	88	311.5 (31.7)	54	350.0 (38.6)	7
Distance-based (D: 12, E: 4)	30 000	200 167.4	34.1	186.5 (5.1)	204.8 (8.2)	100	215.2 (14.0)	98	308.6 (28.0)	96	332.6 (32.2)	71	377.5 (38.4)	13
Distance-based (D: 12, E: 24)	30 000	200 177.8	34.4	185.9 (4.8)	203.9 (7.7)	100	213.7 (13.9)	97	307.0 (27.9)	95	331.4 (30.6)	71	377.6 (37.3)	14

Standard deviations are shown in parentheses.

SR, success rate; D, parameter `DEGLIM`; E, parameter `EXP`; PRM, probabilistic roadmap.

DOI: 10.1002/cav

Table 3. Average results of 1000 test runs for asteroids environment.

Method	Nodes	Edges	Time (s)	Static length	Additional 1		Additional 2		Additional 3		Additional 4		Additional 5	
					Length	SR	Length	SR	Length	SR	Length	SR	Length	SR
Basic PRM	5000	4996.7	5.4	474.1 (83.2)	—	0	—	0	—	0	—	0	—	0
8 nearest neighbors	5000	30 021.9	22.7	267.1 (6.6)	359.4 (39.4)	94	395.7 (54.7)	85	428.1 (76.5)	19	450.6 (85.5)	1	—	0
Random	5000	20 040.0	20.3	275.0 (7.3)	411.2 (54.8)	88	465.8 (68.3)	70	511.9 (105.5)	9	599.8 (128.1)	1	—	0
Useful edges	5000	20 011.4	33.0	265.2 (6.2)	333.5 (20.7)	100	356.8 (35.6)	99	413.5 (51.4)	43	470.2 (90.0)	7	476.0 (94.7)	4
Distance-based (D: 1, E: 2)	5000	20 014.3	24.9	263.8 (5.9)	349.9 (25.9)	100	386.9 (46.3)	99	470.8 (69.4)	39	578.9 (98.8)	5	528.5 (100.6)	1
Distance-based (D: 1, E: 12)	5000	20 113.6	30.0	258.3 (6.1)	347.3 (24.0)	100	387.0 (45.3)	100	485.0 (75.4)	47	594.3 (94.3)	6	589.4 (112.8)	1
Distance-based (D: 4, E: 12)	5000	20 078.3	29.2	261.3 (6.3)	350.2 (23.2)	100	388.0 (41.6)	100	482.6 (73.7)	49	607.5 (107.9)	7	610.5 (107.3)	2
Random	5000	25 143.6	25.3	268.1 (6.0)	377.6 (43.6)	98	426.8 (62.9)	89	505.0 (96.6)	22	583.5 (63.8)	2	—	0
Useful edges	5000	25 065.7	37.5	258.9 (5.5)	316.5 (16.1)	100	331.7 (26.7)	100	384.9 (43.5)	61	450.1 (79.1)	20	451.0 (86.4)	11
Distance-based (D: 1, E: 2)	5000	25 064.7	32.0	257.4 (5.6)	327.1 (18.4)	100	353.8 (33.5)	100	427.9 (64.3)	61	528.0 (84.5)	18	534.4 (91.5)	5
Distance-based (D: 1, E: 12)	5000	25 084.5	37.2	253.9 (5.7)	326.7 (18.4)	100	355.3 (35.6)	100	448.7 (64.1)	68	559.1 (103.2)	22	559.0 (100.3)	8
Distance-based (D: 4, E: 12)	5000	25 086.5	37.2	254.7 (5.9)	327.6 (18.8)	100	353.8 (33.4)	100	442.8 (63.0)	68	553.5 (88.0)	21	564.6 (96.4)	7
Random	5000	30 213.5	30.3	263.8 (5.4)	352.9 (34.3)	98	396.8 (53.2)	95	489.5 (82.7)	32	526.1 (82.5)	5	548.1 (81.1)	1
Useful edges	5000	29 996.3	42.0	255.0 (5.2)	305.7 (13.3)	100	316.7 (21.5)	100	365.1 (40.2)	68	426.2 (73.7)	29	446.8 (84.9)	18
Distance-based (D: 1, E: 2)	5000	29 961.8	39.0	253.1 (5.2)	313.7 (15.2)	100	331.7 (27.1)	100	404.8 (56.1)	74	494.7 (78.1)	32	510.9 (72.8)	16
Distance-based (D: 1, E: 12)	5000	30 045.3	44.0	250.6 (5.1)	316.4 (16.8)	100	338.2 (29.5)	100	425.0 (60.6)	78	532.7 (90.0)	32	560.1 (104.5)	15
Distance-based (D: 4, E: 12)	5000	30 083.1	44.3	250.4 (5.2)	315.1 (16.4)	100	336.5 (28.1)	100	420.8 (57.3)	78	520.2 (82.5)	36	534.4 (97.6)	16
Basic PRM	10 000	9995.3	10.5	468.9 (84.6)	—	0	—	0	—	0	—	0	—	0
4 nearest neighbors	10 000	30 838.7	22.6	292.0 (9.3)	452.5 (66.4)	61	490.6 (75.9)	33	481.6 (70.2)	3	—	0	—	0
Random	10 000	20 128.3	19.5	302.3 (10.1)	520.8 (85.3)	48	573.4 (90.5)	26	586.0 (94.2)	1	—	0	—	0
Useful edges	10 000	20 062.4	61.8	289.0 (9.5)	406.6 (35.5)	100	443.7 (54.6)	74	486.3 (59.4)	13	—	0	—	0
Distance-based (D: 1, E: 2)	10 000	20 013.5	22.0	294.6 (9.3)	470.8 (55.3)	93	536.4 (70.0)	68	578.3 (82.0)	4	—	0	—	0
Distance-based (D: 1, E: 12)	10 000	20 060.2	24.9	285.2 (8.8)	436.5 (41.4)	99	505.2 (68.0)	82	566.3 (68.8)	14	—	0	—	0
Random	10 000	25 067.7	23.8	290.8 (8.3)	462.3 (64.5)	81	521.7 (73.9)	61	524.7 (108.6)	4	—	0	—	0
Useful edges	10 000	25 056.2	64.1	279.8 (8.0)	367.9 (24.4)	100	395.9 (39.1)	93	451.7 (43.6)	34	514.2 (101.8)	2	463.2 (69.4)	1
Distance-based (D: 1, E: 2)	10 000	25 026.8	27.2	281.4 (7.5)	398.8 (36.7)	99	448.0 (55.9)	96	519.1 (71.1)	19	—	0	—	0
Distance-based (D: 1, E: 12)	10 000	25 105.9	32.2	272.6 (7.9)	380.6 (26.2)	100	426.4 (47.4)	98	510.5 (60.0)	40	579.6 (94.7)	2	644.2 (157.4)	1
Distance-based (D: 3, E: 12)	10 000	24 966.2	29.9	288.9 (9.1)	419.7 (35.3)	100	473.3 (57.0)	91	536.0 (58.5)	21	570.4 (120.4)	1	—	0
Random	10 000	30 208.7	28.3	283.0 (7.4)	415.7 (48.9)	95	476.0 (64.3)	85	525.1 (110.7)	11	584.0 (155.5)	1	—	0
Useful edges	10 000	30 064.4	67.0	273.8 (6.8)	346.3 (19.2)	100	368.3 (32.9)	99	424.1 (42.7)	57	473.3 (77.2)	8	490.5 (84.3)	4
Distance-based (D: 1, E: 2)	10 000	30 023.7	32.8	273.4 (7.0)	363.1 (24.0)	100	397.6 (41.3)	100	474.3 (64.8)	39	625.0 (104.0)	2	—	0
Distance-based (D: 1, E: 12)	10 000	30 046.5	38.9	266.1 (6.8)	355.6 (22.1)	100	387.9 (35.7)	100	473.6 (59.6)	60	607.1 (100.5)	9	605.8 (101.1)	4
Distance-based (D: 4, E: 12)	10 000	30 017.6	36.4	279.6 (8.0)	383.4 (24.9)	100	420.6 (41.4)	99	493.9 (56.3)	44	620.7 (117.8)	4	638.8 (123.3)	2

Standard deviations are shown in parentheses.

SR, success rate; D, parameter `DEGLIM`; E, parameter `EXP`; PRM, probabilistic roadmap.

DOI: 10.1002/cav

Table 4. Average results of 1000 test runs for house environment.

					Additional 1		Additional 2		Additional 3		Additional 4		Additional 5	
Method	Nodes	Edges	Time (s)	Static length	Length	SR	Length	SR	Length	SR	Length	SR	Length	SR
Basic PRM	20 000	19 996.6	13.6	2181.2 (235.1)	2114.5 (225.7)	20	2122.1 (236.6)	1	—	0	—	0	—	0
19 nearest neighbors	20 000	303 427.4	119.6	1156.9 (25.5)	1162.8 (25.8)	100	1173.3 (26.7)	97	1875.0 (182.2)	45	1932.1 (182.6)	41	2164.2 (112.9)	28
Random	20 000	100 229.4	48.7	1302.3 (40.2)	1314.1 (40.6)	100	1333.4 (43.0)	82	2178.9 (207.3)	14	2270.7 (197.5)	7	2619.2 (105.9)	2
Useful edges	20 000	100 567.8	89.6	1247.2 (27.7)	1254.9 (28.0)	100	1273.5 (29.7)	96	1997.4 (164.2)	74	2068.8 (166.7)	68	2402.5 (105.7)	47
Distance-based (D: 4, E: 4)	20 000	100 188.3	56.7	1196.2 (29.4)	1205.8 (30.2)	100	1224.1 (33.0)	97	1942.2 (155.6)	76	2025.3 (158.7)	61	2365.6 (102.1)	46
Distance-based (D: 8, E: 4)	20 000	100 219.4	58.1	1231.6 (32.3)	1242.7 (32.8)	100	1263.3 (35.2)	97	1980.3 (149.2)	76	2064.5 (150.5)	72	2433.3 (95.8)	54
Random	20 000	200 091.0	91.5	1197.7 (31.0)	1205.4 (30.8)	100	1219.4 (31.8)	90	1969.8 (189.8)	41	2035.8 (192.0)	32	2289.5 (117.0)	19
Useful edges	20 000	200 431.7	126.0	1157.9 (22.4)	1163.6 (23.0)	100	1176.3 (24.4)	98	1826.1 (144.4)	79	1882.6 (144.7)	77	2149.5 (86.8)	61
Distance-based (D: 4, E: 4)	20 000	200 100.3	102.2	1140.5 (25.4)	1147.1 (26.2)	100	1161.7 (27.6)	97	1825.8 (141.8)	80	1892.5 (142.2)	65	2165.4 (89.1)	53
Distance-based (D: 8, E: 4)	20 000	200 158.3	103.9	1132.3 (23.4)	1138.6 (23.6)	100	1152.2 (25.4)	98	1798.2 (132.7)	81	1862.6 (134.4)	77	2137.7 (82.2)	64
Distance-based (D: 12, E: 4)	20 000	200 257.4	107.3	1131.5 (23.2)	1137.6 (23.7)	100	1150.5 (25.6)	98	1789.2 (127.7)	79	1848.4 (128.9)	78	2121.9 (79.0)	65
Random	20 000	300 548.6	134.7	1148.9 (25.3)	1155.3 (25.2)	100	1167.0 (27.1)	94	1866.5 (171.6)	56	1922.8 (174.2)	49	2152.9 (109.4)	34
Useful edges	20 000	300 558.1	166.2	1123.1 (21.7)	1127.6 (21.8)	100	1138.8 (22.6)	98	1768.4 (139.3)	81	1819.3 (139.2)	79	2058.6 (86.9)	65
Distance-based (D: 4, E: 4)	20 000	300 577.0	144.9	1118.5 (24.4)	1123.9 (24.7)	100	1136.0 (24.9)	98	1770.3 (127.8)	80	1829.1 (129.7)	70	2073.2 (82.4)	59
Distance-based (D: 8, E: 4)	20 000	300 328.4	146.8	1113.7 (22.1)	1118.5 (22.5)	100	1130.1 (24.1)	98	1767.5 (135.2)	82	1824.3 (136.4)	78	2069.7 (86.2)	66
Distance-based (D: 12, E: 4)	20 000	300 335.5	148.8	1110.1 (20.6)	1115.3 (20.9)	100	1126.1 (21.8)	99	1752.4 (130.9)	81	1807.2 (130.4)	80	2053.9 (80.4)	67
Basic PRM	30 000	29 997.2	20.8	2173.4 (234.2)	2122.5 (189.0)	17	—	0	—	0	—	0	—	0
13 nearest neighbors	30 000	322 206.7	118.4	1195.9 (29.3)	1202.8 (29.8)	100	1214.3 (31.5)	98	1905.7 (163.0)	60	1956.3 (161.4)	53	2215.1 (97.9)	40
Random	30 000	100 123.2	49.6	1378.3 (48.1)	1392.0 (48.4)	100	1414.5 (50.2)	81	2327.9 (234.1)	14	2449.5 (233.7)	8	2803.1 (151.3)	2
Useful edges	30 000	100 099.1	151.8	1323.4 (32.2)	1333.0 (32.9)	100	1353.1 (34.8)	98	2072.9 (140.4)	88	2154.7 (139.8)	77	2558.9 (96.2)	56
Distance-based (D: 4, E: 4)	30 000	100 350.0	57.5	1285.2 (35.9)	1299.2 (36.2)	100	1324.2 (39.4)	99	2061.3 (142.8)	87	2160.6 (138.1)	68	2618.6 (101.8)	54
Random	30 000	200 500.7	90.6	1245.0 (34.5)	1253.6 (35.1)	100	1270.1 (36.8)	92	2047.4 (189.8)	47	2125.8 (193.2)	32	2415.9 (116.8)	17
Useful edges	30 000	200 293.5	179.2	1204.8 (24.1)	1211.3 (24.6)	100	1225.5 (25.9)	99	1862.2 (104.7)	94	1922.8 (104.5)	90	2236.6 (73.4)	77
Distance-based (D: 4, E: 4)	30 000	200 182.1	103.4	1165.4 (25.5)	1174.4 (26.4)	100	1190.3 (28.3)	100	1815.0 (92.5)	93	1886.2 (93.4)	76	2204.6 (67.2)	69
Distance-based (D: 8, E: 4)	30 000	200 266.0	106.7	1162.7 (24.8)	1170.6 (25.5)	100	1186.3 (26.8)	100	1809.1 (100.3)	94	1876.2 (102.6)	90	2192.6 (70.1)	85
Distance-based (D: 12, E: 4)	30 000	200 112.6	108.3	1199.4 (27.3)	1207.0 (28.0)	100	1224.8 (30.6)	99	1869.4 (117.5)	92	1935.4 (119.3)	92	2265.4 (75.2)	84
Random	30 000	299 952.9	131.4	1190.6 (29.0)	1197.7 (29.6)	100	1211.1 (31.7)	96	1915.1 (162.2)	67	1972.8 (159.8)	55	2246.3 (100.0)	37
Useful edges	30 000	300 180.8	212.4	1153.4 (20.1)	1158.5 (20.3)	100	1169.7 (21.5)	100	1777.6 (100.4)	95	1831.2 (99.4)	94	2105.5 (64.3)	82
Distance-based (D: 4, E: 4)	30 000	300 109.8	145.5	1138.9 (24.0)	1145.2 (23.9)	100	1158.3 (25.2)	100	1766.8 (96.2)	95	1829.5 (95.0)	78	2111.5 (65.6)	72
Distance-based (D: 8, E: 4)	30 000	300 656.9	148.6	1131.2 (22.2)	1137.3 (22.5)	100	1149.9 (24.0)	100	1748.0 (90.0)	96	1808.2 (89.9)	93	2089.5 (64.1)	87
Distance-based (D: 12, E: 4)	30 000	299 855.5	152.3	1130.7 (22.2)	1136.7 (22.1)	100	1149.4 (24.0)	100	1744.7 (90.0)	95	1799.5 (89.8)	94	2080.6 (59.5)	86

Standard deviations are shown in parentheses.

SR, success rate; D, parameter `DEGLIM`; E, parameter `EXP`; PRM, probabilistic roadmap.

DOI: 10.1002/cav

parameter EXP had much more effect on the results, and as can be seen, too small values did not yield good results. The main difference between these environments is that the wall environment has a lot of empty space whereas the asteroids environment does not.

Table 4 shows results for the house environment. As can be seen, the random method and the nearest neighbor method did not perform well. Like in the other environments, the useful edges method and the distance-based method were clearly better. The distance-based method was tested with three different values for the parameter DEGLIM. The results were not good when the value was 4. The results were much better when the value was increased to 8, but increasing the value to 12 did not have much impact any more. In a sense, the house environment can be seen as a combination of the wall environment and the asteroids environment. It has much more obstacles than the wall environment but it has much more free space than the asteroids environment.

The results show that the useful edges method and the distance-based method were the best in a sense that the roadmaps built using either method were able to handle additional obstacles well. The success percent was high when using either method. The random method did not achieve the same kind of success rate. As expected, the results also show that the success rate of all methods decreases as the number of obstacles increases. However, it is important to note that the useful edges method and the distance-based method were always better than the other methods. The number of obstacles did not have any impact on that.

The test results also show that the random method is the fastest, the distance-based method is almost equally fast, and the useful edges method is the slowest. The differences between the methods were the smallest in the asteroids environment especially in the case where the node count was 5000. The differences were the largest in the wall environment, and they were large also in the house environment. The asteroids environment was the one that was built randomly, and the wall environment was the one that had the simplest obstacles. However, the environments that are used in computer games and in other virtual worlds are rarely fully random, but they are not very simple either. They are designed by humans, and they are usually like the house environment.

In most of the test cases, the useful edges produced the shortest paths, and the random method the longest. That was a quite expected result because the useful edges method was originally developed to specifically produce short paths and the distance-based method was developed to work in changing environments. As can be seen from Figure 2(b), the shortest paths are not always the best in a sense of obstacle avoidance. However, a clear exception was the house environment where the distance-based method usually produced the shortest paths. In the asteroids environment, the distance-based method produced the shortest paths when there were only static obstacles, but it did not perform so well when the additional obstacles were added. However, there were no big differences between the path lengths in any environment.

Tables 2–4 also show results for the basic PRM planner and for the nearest neighbor method. When the basic planner was used, the time required to build the roadmap was short because the number of edges was small. However, the basic planner is not suitable for changing environments at all. In all environments, path lengths in the static environment were very long, and when all additional obstacles were added, the planner was not able to find a path even once. The nearest neighbor method did not perform well either. Its success rate was almost always worse than the success rate of the random method with the same amount of edges.

Overall, our tests showed that the roadmaps do not have to be large to work well in changing environments. Quite sparse roadmaps can work very well, but it is important to choose the right edges to the roadmap. The roadmap must contain cycles, but they can not be added randomly if we want a good roadmap.

7. CONCLUSIONS AND FUTURE WORK

In this paper, we examined how probabilistic roadmaps work in changing environments. We investigated different strategies for connecting nodes together and then compared them experimentally in environments that had additional obstacles. The results suggest that probabilistic roadmaps can be used effectively to solve the path planning problem in changing environments and that the roadmap graph does not have to be large or dense. Even a very sparse roadmap can be built in such a way that additional obstacles do not break the connectivity of the roadmap too much. However, as our experiments showed, it is very important to add edges between the roadmap nodes correctly. A simple nearest neighbor method or a random method do not achieve as a good success rate in changing environments than a useful edges method or a distance-based method.

One of the tested methods was a distance-based method that we presented in this paper. The method was shown to be fast, and it can achieve reliable results even when the produced roadmap is used in a changing environment. Our experiments showed that it produced much better roadmaps than the random method but it still worked almost equally fast. The useful edges method also produced good roadmaps, but it was much slower than the distance-based method.

In future, it would be useful to incorporate our findings to some roadmap-based path planning framework that works with moving obstacles. It would also be interesting to study how probabilistic roadmaps work in changing environments when a moving character is more complex than a free-flying rigid body that was used in this paper. There has also been a lot of research made about probabilistic roadmaps in static environments. It would be good

to see how some of those ideas would work in changing environments. For example, some methods that try to keep a roadmap small could be useful also in changing environments.

ACKNOWLEDGEMENTS

The research of the first author was supported by the Tampere Doctoral Programme in Information Science and Engineering.

REFERENCES

1. Latombe J-C. *Robot Motion Planning*. Kluwer Academic Publishers, Boston, MA, 1991.
2. Choset H, Lynch KM, Hutchinson S, Kantor G, Burgard W, Kavraki LE, Thrun S. *Principles of Robot Motion*. MIT Press, Cambridge, MA, 2005.
3. LaValle S. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.
4. Smed J, Hakonen H. Path finding. In *Algorithms and networking for computer games*. John Wiley & Sons, Chichester, UK, 2006; 97–113.
5. Schwartz JT, Sharir M, Hopcroft J. *Planning, Geometry, and Complexity of Robot Motion*. Ablex Publishing Corporation, Norwood, New Jersey, 1987.
6. Lozano-Pérez T. Spatial planning: a configuration space approach. *IEEE Transactions on Computers* 1983; **32**: 108–120.
7. Reif JH. Complexity of the mover's problem and generalizations, In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, San Juan, Puerto Rico, 1979; 421–427.
8. Canny JF. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.
9. Overmars MH. A Random Approach to Motion Planning. *Technical Report RUU-CS-92-93*, Department of Computer Science, Utrecht University, the Netherlands, 1992.
10. Kavraki L, Latombe J-C. Randomized preprocessing of configuration space for fast path planning, In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Diego, CA, United States, 1994; 2138–2145.
11. Kavraki LE, Švestka P, Latombe J-C, Overmars MH. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics & Automation* 1996; **12**(4): 566–580.
12. Amato NM, Bayazit OB, Dale LK, Jones C, Vallejo D. OBPRM: An obstacle-based PRM for 3D workspaces, In *Proceedings of the International Workshop on Algorithmic Foundations of Robotics*, Houston, TX, United States, 1998; 155–168.
13. Boor V, Overmars MH, van der Stappen AF. The Gaussian sampling strategy for probabilistic roadmap planners, In *Proceedings of the IEEE International Conference on Robotics and Automation*, Detroit, MI, United States, 1999; 1018–1023.
14. Hsu D, Jiang T, Reif J, Sun Z. The bridge test for sampling narrow passages with probabilistic roadmap planners, In *Proceedings IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003; 4420–4426.
15. Lien JM, Thomas SL, Amato NM. A general framework for sampling on the medial axis of the free space, In *Proceedings of the IEEE International Conference on Robotics and Automation*, Taipei, Taiwan, 2003; 4439–4444.
16. Siméon T, Laumond J-P, Nissoux C. Visibility based probabilistic roadmaps for motion planning. *Advanced Robotics* 2000; **14**(6): 477–493.
17. Rantanen MT, Juhola M. A configuration deactivation algorithm for boosting probabilistic roadmap planning of robots. *International Journal of Automation and Computing* 2012; **9**(2): 155–164.
18. Morales M, Tapia L, Pearce R, Rodriguez S, Amato NM. A machine learning approach for feature-sensitive motion planning. *Algorithmic Foundations of Robotics VI* 2005; **17**: 361–376.
19. Lin YT. The Gaussian PRM sampling for dynamic configuration spaces, In *Proceedings of the International Conference on Control, Automation, Robotics and Vision*, Singapore, 2006; 1–5.
20. Geraerts R, Overmars MH. The corridor map method: a general framework for real-time high-quality path planning. *Computer Animation and Virtual Worlds* 2007; **18**: 107–119.
21. Nieuwenhuisen D, Overmars MH. Useful cycles in probabilistic roadmap graphs, In *Proceedings of IEEE International Conference on Robotics and Automation*, New Orleans, LA, United States, 2004; 446–452.
22. van den Berg JP, Nieuwenhuisen D, Jaillet L, Overmars MH. Creating robust roadmaps for motion planning in changing environments, In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Edmonton, Canada, 2005; 1053–1059.
23. Nieuwenhuisen D, van den Berg J, Overmars M. Efficient path planning in changing environments, In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, San Diego, CA, United States, 2007; 3295–3301.
24. Jaillet L, Siméon T. A PRM-based motion planner for dynamically changing environments, In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Sendai, Japan, 2004; 1606–1611.

25. Bohlin R, Kavraki LE. Path planning using lazy PRM, In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, United States, 2000; 521–528.
26. LaValle SM. Rapidly-exploring random trees: A new tool for path planning. *Technical Report 98-11*, Computer Science Dept., Iowa State University, 1998.
27. Kuffner JJ, LaValle SM. RRT-Connect: An efficient approach to single-query path planning, In *Proceedings of IEEE International Conference on Robotics and Automation*, San Francisco, CA, United States, 2000; 995–1001.
28. Sud A, Gayle R, Andersen E, Guy S, Lin M, Manocha D. Real-time navigation of independent agents using adaptive roadmaps, In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, Newport Beach, CA, United States, 2007; 99–106.
29. Lopez T, Lamarche F, Li T-Y. Space-time planning in changing environments: using dynamic objects for accessibility. *Computer Animation and Virtual Worlds* 2012; **23**: 87–99.
30. Snook G. Simplified 3D movement and pathfinding using navigation meshes. In *Game Programming Gems*, DeLoura M (ed.). Charles River Media. 288–304, 2000.
31. Geraerts R, Overmars MH. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems* 2006; **54**: 165–173.
32. Geraerts R, Overmars MH. Reachability-based analysis for probabilistic roadmap planners. *Robotics and Autonomous Systems* 2007; **55**(11): 824–836.
33. Jiménez P, Thomas F, Torras C. Collision detection algorithms for motion planning. In *Robot Motion Planning and Control*, Laumond J-P (ed.). Springer-Verlag, Berlin, 1998.
34. McMahon T, Jacobs S, Boyd B, Tapia L, Amato NM. Local randomization in neighbor selection improves PRM roadmap quality, In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, Vilamoura, Portugal, 2012; 4441–4448.
35. Larsen E, Gottschalk S, Lin MC, Manocha D. Fast proximity queries with swept sphere volumes, In *Proceedings of the IEEE International Conference on Robotics and Automation*, San Francisco, CA, United States, 2000; 3719–3726.

AUTHORS' BIOGRAPHIES

Mika T. Rantanen took his MS degree at the Department of Computer Sciences, University of Tampere, Finland, in 2009, and now continues his research on robotics toward a PhD degree. His research efforts concentrate on probabilistic roadmap algorithms.

Martti Juhola received his MS and PhD degrees in computer science from the University of Turku, Finland, in the 1980s, where he was an academic assistant, lecturer, and researcher, later becoming a professor at the University of Kuopio, Finland. Since 1997, he is a professor at the University of Tampere. His research interests include medical informatics, signal analysis, pattern recognition, and information retrieval.

Publication IV

Speeding up probabilistic roadmap planners with locality-sensitive hashing

Mika T. Rantanen and Martti Juhola

Speeding up probabilistic roadmap planners with locality-sensitive hashing

Mika T. Rantanen Martti Juhola

Computer Science, School of Information Sciences,
FI-33014 University of Tampere, Finland

Abstract

A crucial part of probabilistic roadmap planners is the nearest neighbor search which typically is done by exact methods. Unfortunately, searching the neighbors can become a major bottleneck for the performance. This can occur when the roadmap size grows especially in high-dimensional spaces. In this paper, we investigate how well the approximate nearest neighbor searching works with probabilistic roadmap planners. We propose a method that is based on the locality-sensitive hashing and we show that it can speed up the construction of the roadmap considerably without reducing the quality of the produced roadmap.

1 Introduction

The motion planning is an important part of robotics (see e.g. refs. [1, 2, 3]). The objective is to find a path for a robot from a start location to a goal location. The robot moves in an environment that has obstacles and it must avoid collisions with them. One important application for the motion planning is to guide unmanned vehicles autonomously[4]. Other applications include, among others, computer games where motion planning can be used to move game characters[5], and molecular simulations[6].

The location of the robot is typically represented as a *configuration* and the *configuration space* $\mathcal{C}$ contains all possible configurations.[7] The *free configuration space* $\mathcal{C}_{\text{free}} \subseteq \mathcal{C}$ is a set of all configurations where the robot does not collide with the obstacles or itself. The robot can now be thought to be a single point in $\mathcal{C}$ whereas a path between two locations q_1 and q_2 is a continuous function $\tau : [0, 1] \to \mathcal{C}$, where $\tau(0) = q_1$ and $\tau(1) = q_2$. The path is free if it lies totally in $\mathcal{C}_{\text{free}}$.

The dimension of the configuration space depends on the robot. In case of a rigid body robot that moves in a three-dimensional environment without rotations, three parameters are needed to represent the location. Therefore, the configuration space is three-dimensional and the robot has three *degrees of freedom*. If the robot can also rotate itself freely, three additional parameters are needed (roll, pitch, and yaw). In that

case, the configuration space is six-dimensional. The robot can also be more complex like a robot arm that has many joint angles. It is also possible that the robot consists of multiple rigid bodies and that they form a single multibody robot. In that case the dimensionality of $\mathcal{C}$ is the same as the sum of degrees of freedom of all the bodies.

The general motion planning problem is PSPACE-complete[8,9] which means that exact algorithms are not useful in practice. Luckily, there are many heuristic methods that can be used. Especially *probabilistic roadmap* (PRM) planners[10,11,12] have been shown to work well in difficult environments. These planners construct a graph called a *roadmap*, which is a simplified representation of $\mathcal{C}_{\text{free}}$. Its nodes are randomly sampled configurations from $\mathcal{C}_{\text{free}}$ and an edge between two configurations means that there is a simple and free path between them.

When using probabilistic roadmaps, most of the work is done during the construction of the roadmap and after the roadmap is ready, it can be used to solve different path planning queries quickly. The roadmap is constructed by using information about the static obstacles of an environment. However, PRM planners are versatile and it is possible to use them also in applications that have dynamic environments.[13,14]

While constructing the roadmap, the PRM planner must find a set of the nearest neighbor nodes every time a new node is added to the roadmap. Using exact methods is fast when the roadmap is small and the dimension of the configuration space is low, but it can become a major bottleneck for performance when the roadmap size grows especially in high-dimensional spaces. Therefore, more efficient methods should be used.

The nearest neighbor search can be accelerated by using approximation methods like ANN[15] or FLANN[16]. In this paper, we concentrate on *locality-sensitive hashing* (LSH) which has successfully been applied to a variety of applications lately.[17,18,19] However, as far as we know, LSH has not been used with PRM planners before. In this paper, we propose a simple LSH-based method that can speed up the construction of the roadmaps remarkably. We also investigate how the use of this approximation method affects the quality of the roadmaps.

2 Probabilistic roadmap planners

A typical probabilistic roadmap planner is depicted in Algorithm 1. It starts with an empty roadmap graph $G = (V, E)$, where V is a set of configurations and E is a set of edges. The configurations are added to V one by one. Each time a new configuration q is added, a set of neighboring configurations is retrieved for it from V. Then, the algorithm tries to find a free path from q to each of its neighbors with a *local planner*. If the local planner can find a path, an edge between those configurations is added to E.

A good roadmap captures the *connectivity* of the free configuration space as good as possible. It means that if there is a way for a robot to go from a location q_1 to a location q_2 without colliding obstacles, there should also be a path in G between configurations

Algorithm 1 Constructs the roadmap $G = (V, E)$.

```
1:  V ← ∅
2:  E ← ∅
3:  repeat
4:      q ← sampled configuration from C_free
5:      V ← V ∪ {q}
6:      N_q ← k nearest neighbor configurations of q chosen from V
7:      for all q' in N_q do
8:          if q and q' are not in the same component of the roadmap then
9:              if local planner finds a path between q and q' then
10:                 E ← E ∪ {(q, q')}
11: until there are enough configurations in V
12: return G
```

q_1 and q_2 after those configurations are added to the roadmap with a local planner. If the connectivity is not good, it is possible that q_1 and q_2 belong to different components and there is not a path between them.

In this paper, we are especially interested in searching the neighbors and the following sections will cover that topic. Here we go shortly through other important parts of PRM planners.

Node sampling In line 4 of Algorithm 1, a new configuration is sampled from a free configuration space. The simplest way to generate these new configurations is to sample them randomly using uniform distribution. Another simple method is to use a low-discrepancy sequences to produce the samples.[20,3]

In fully random environments, these sampling methods are adequate. However, the environments where the robots move are often more organized and structured. For example, if the robot moves in a house there are walls and furniture as obstacles and large empty areas where the robot can move freely. In these kinds of environments it is possible to use sampling methods that try to bias sampling towards the difficult areas.

Many different methods have been suggested in order to enhance sampling. Some methods try sample nodes near the obstacles[21,22,23] while some methods try to sample nodes as far of the obstacles as possible[24,25]. There are also methods that divide the configuration space into different regions and then try to detect the best way to sample each region.[26,27,28]

Local planner In line 9, a local planner is used to check whether a path between two configurations is free. One commonly used local planner is a straight-line planner which just interpolates a path between two configurations (see e.g. ref. [12]). The path is then divided into small steps and each step is checked for collisions

separately. Other local planners have been proposed also (see e.g. refs. [29, 30]). These more advanced planners can typically find paths that the straight-line planner can not but, on the other hand, they usually work slower.

The collision checks that the local planner does are one of the most time-consuming parts of PRM planners. Therefore, it is not feasible to check all possible edges for collision and instead, a set of neighbor nodes are selected in line 6 and a local planner is called just for them. Moreover, if the goal is just to generate a roadmap that captures the connectivity of the free configuration space well, it is not necessary to add cycles to the roadmap. That is why the algorithm checks in line 8 whether a neighbor configuration is already on the same connected component of the roadmap.

Distance metric In line 6, a set of the nearest neighbor nodes is selected according to a metric which should be such that the distance between two configurations p and q reflects the difficulty of connecting them together with a local planner. One possibility would be to measure the volume of the workspace that is swept by a robot when it moves between p and q. When this swept-volume is small, also the probability that the robot will collide with obstacles is small. However, it is very difficult and slow to compute the swept-volume exactly. Usually the metrics used in motion planning are approximations.[31] Next we describe one common approach.

If the configuration space is a Cartesian product of n metric spaces and each has been associated with a distance metric $M_1, M_2, \ldots, M_n$, the approximate distance between two configurations $p = (p_1, p_2, \cdots, p_n)$ and $q = (q_1, q_2, \cdots, q_n)$ can be defined as

$$\mathrm{dist}(p, q) = \sqrt{\sum_{i=1}^{n} w_{M_i} \mathrm{dist}^2_{M_i}(p_i, q_i)},$$

where $w_{M_i} \in \mathbb{R}$ are weight constants. The distance for two points $p_1, p_2 \in \mathbb{R}$ can be defined as

$$\mathrm{dist}_{\mathrm{T}}(p_1, p_2) = |p_1 - p_2|.$$

Special care must be taken when handling rotations to ensure that the shortest distance is selected. This can be achieved for three-dimensional rotations that are represented as two quaternions $h_1, h_2 \in \mathbb{H}$ by defining the distance as

$$\mathrm{dist}_{\mathrm{R}}(h_1, h_2) = \arccos(|h_1 \cdot h_2|).$$

One difficulty is to decide the values for the weight constants. Usually the translational distances are more important than the rotational distances and, for example, the study in ref. [29] supports this.

3 Nearest neighbor search

In the nearest neighbor search a set S of points in a metric space M is given and a goal is to find the nearest point in S to a query point $q \in M$. Often one nearest neighbor is not enough but instead k nearest neighbors for q is required. This is the case also with the probabilistic roadmap planners.

The nearest neighbor search is an important part of PRM planners since for each sampled configuration we need to search for its neighbors. Unfortunately, this can become a very time-consuming operation when the roadmap size increases. The neighbor search is one of the bottlenecks of the PRM planners along with the collision checks.

The nearest neighbor search has many important applications also besides the robotics and it has been researched extensively. Many applications require that the nearest neighbor search always returns an exact result. However, there are also many applications where it is not necessary and a good approximation is enough. This is beneficial because while the exact nearest neighbor algorithms can be slow there are several approximation methods that can work much faster.

For example, the probabilistic roadmap methods themselves are not exact algorithms but still exact nearest neighbor search is typically used in these methods. However, this is likely unnecessary and as we show in this paper, the approximation methods are sufficient.

We consider a case, where the roadmap is empty at the beginning and all nodes are added to the roadmap one by one. This way it is possible to construct the roadmap without knowing how large it will be in advance and adding new nodes to the roadmap is easy. Another approach would be that all nodes are generated first and then added to the roadmap at once.

Next we describe nearest neighbor methods that we used in our experiments.

3.1 Brute-force search

The simplest way to find k nearest neighbors is to go through all points in S and for each one calculate the distance from the query point q. While iterating through the points, a list of k nearest neighbors found so far is maintained. At the end, the list has an exact result. The brute-force method is slow since it requires that every point is checked. However, this method does not use any additional data structures which could be difficult and slow to maintain. Therefore, it can outperform other exact methods especially in high-dimensional spaces.[32]

3.2 A *kd*-tree method

There are several space-partitioning methods that can be used in exact nearest neighbor searches. One of the simplest is a *kd*-tree method that uses a binary tree to store the points.[33,34]

In a *kd*-tree each node contains a point from S. Every node divides the space into two partitions by a plane that goes through the associated point. Every node is also associated with one of the dimensions of the space M and the dividing plane is defined to be perpendicular to the axis of that dimension. The left subtree of the node contains all the points that lie on one side of the dividing plane and the right subtree contains the rest of the points.

The results in ref. [35] show that the *kd*-trees can dramatically increase the performance of the probabilistic roadmap planners when compared with brute-force search. However, a *kd*-tree and other space-partitioning methods work well only in low-dimensional spaces and they become inefficient when the dimensionality grows large. In matter of fact, the results in ref. [32] show that a simple brute-force method can outperform space-partitioning methods when the number of dimensions grows larger than around 10.

3.3 Locality-sensitive hashing

The locality-sensitive hashing can be used to accelerate the search of nearest neighbors. It is a popular method and it has been used successfully in many different applications, for example in audio and image retrieval (see e.g. refs. [36, 37]). The method does not guarantee that it will find exactly the nearest neighbors but it will return a very good approximation with a high probability.

The LSH method is based on a hash table which contains several buckets. Each bucket is associated with a unique hash value and a hash function g is used to calculate a hash value for points in metric space M. By using this function, it is possible to store all points in S to the buckets. It should be noted that one bucket should contain several points.

The hashing function g must be selected in such a way that it returns the same value for two points with a high probability if the points are close to each other according to some distance metric. This means that the hash function preserves the locality information and the points that are near each other are likely to be stored to the same bucket in the hash table.

To search for a set of points that are near to a query point $q \in M$, we first calculate a hash value for q. Then we just retrieve all points from the bucket indicated by this hash value. The hash tables can be implemented to work very efficiently which means that in practice, the searching of these points can be done quickly. Unfortunately, it is not certain that the found points contain the actually nearest neighbor for q.

It is possible to increase the probability that the method finds the nearest point

Algorithm 2 Adds a point q to the hash tables

1: **for** $i = 1, 2, \ldots, L$ **do**
2: $h \leftarrow$ hash value for q in i-th hash table
3: $b \leftarrow$ bucket identified by a value h from i-th hash table
4: Add point q to the bucket b

Algorithm 3 Returns k nearest nodes to a query point q

1: $V \leftarrow \emptyset$
2: **for** $i = 1, 2, \ldots, L$ **do**
3: $h \leftarrow$ hash value for q in i-th hash table
4: $b \leftarrow$ bucket identified by a value h from i-th hash table
5: $V' \leftarrow$ all nodes from bucket b
6: $V \leftarrow V \cup V'$
7: **return** V

by using multiple hash tables. It means that instead of one hash table we create independently L tables and for each table we use a different hash function. Now new points must be added to every table as shown in Algorithm 2. To retrieve the nearest points, we must get points from each table and then combine them together as shown in Algorithm 3.

A variety of different hashing functions has been developed (see e.g. refs. [38, 39]). One simple method is to use randomly chosen hyperplanes. Each plane divides the space into two partitions and together they divide the space into several regions. Another method is to use lattices to create regular-shaped regions into the space. In both cases, the hashing function can return the same hash value for each point that lies in the same region.

The problem that remains is to decide how to choose an appropriate hashing function and how to choose the number of buckets and the number of hash tables. The number of buckets is usually linked to the hashing function but in general, the accuracy can be increased by adding the number of hash tables. This, however, makes the method work slower and consume more memory. When choosing a hashing function, it should be remembered that many proposed functions are designed to work only in Euclidean spaces or in some other specific cases which means that they may not be useful in all applications. In the next section, we describe a centroid-based hashing that is easy to be implemented and which can be extended to satisfy the needs of probabilistic roadmap planners.

3.4 Centroid-based hashing

In centroid-based hashing[39], a set of c centroids is generated at the beginning. These centroids belong to the space M and an arbitrary point $q \in M$ can be associated

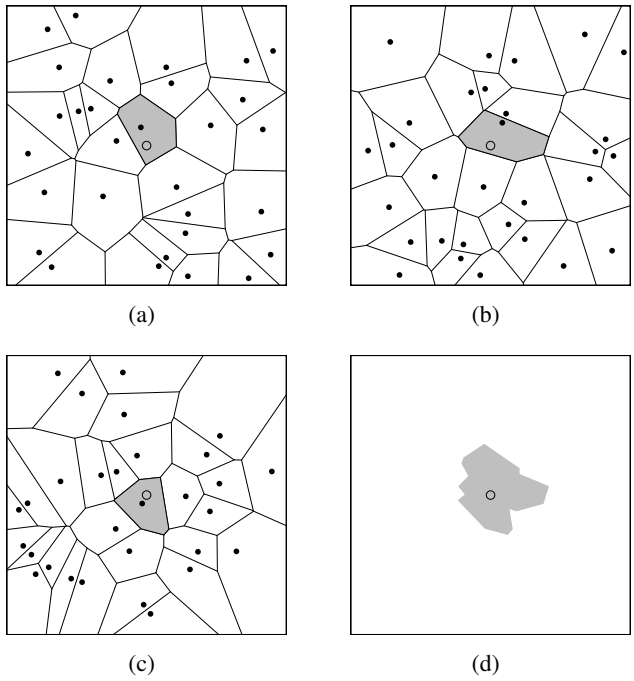


Figure 1: An example of a locality-sensitive hashing. In (a)–(c) the space is divided into several regions. Black dots represent centroids, a small circle represents a query point and the cell where the query point lies is marked with a gray color. In (d), the marked cells from (a)–(c) are combined.

with one of the centroids by calculating the distance from q to each centroid and then selecting the centroid that is the nearest. This essentially divides the space M into c partitions. This division can also be thought to be a Voronoi diagram[40] where each centroid corresponds to one Voronoi cell.

We can now define the hashing function g in such a way that it takes a point, calculates which Voronoi cell it belongs to, and then returns a hash value associated with that cell. By using this hash function, the number of buckets will be the same as the number of centroids.

The centroid-based hashing is illustrated in two-dimensions in Figure 1. In 1(a)–1(c), three different sets of centroids and corresponding Voronoi cells are shown. The centroids are marked with a small dot, a query point q is shown in the middle as a small circle and the cell in which q belongs to is marked with a gray color. To retrieve a set of the nearest points for q it is sufficient to retrieve points from these three marked Voronoi cells. This is shown in 1(d) in which the point q is shown as well as the union of the gray Voronoi cells from 1(a)–1(c).

Next we propose a simple centroid-based method that works well with probabilistic

Algorithm 4 Initializes L hash tables with c centroids

```
1: Create L hash tables
2: for i = 1, 2, ..., L do
3:    P ← ∅
4:    for j = 1, 2, ..., c do
5:       q ← a random free configuration
6:       P = P ∪ {q}
7:    Associate centroids P with i-th hash table
```

roadmaps. We address two issues: how to choose the centroids, and how to handle cases where there are only a few nodes in the roadmap.

The first issue is to choose the centroids. The simplest way to do it would be to generate c random points and use them as centroids. This would work but it would ignore some information that we have from the environment. We propose a method that takes the static obstacles into account. Our method to initialize the centroids is shown in Algorithm 4. We assume that the roadmap is empty at the beginning which means that it is not possible to use the information about the existing roadmap. Therefore, the method chooses c centroids uniformly at random but requires that all of them must lie in the free configuration space. This method yields good results as we show in our experiments. To distribute the centroids in more evenly manner to the space, it is possible to generate them using some low-discrepancy sequence.

The second issue is to handle cases when the roadmap has only a few nodes. For example, if we want to retrieve k nearest neighbors for a query point and the roadmap has only k nodes, all nodes from the roadmap should be returned. However, the method shown in Algorithm 3 would probably return only a small subset of all nodes because it returns nodes only from some of the buckets. An improved method is shown in Algorithm 5. It immediately returns all nodes if the roadmap has less than k nodes and otherwise ensures that k nearest nodes are always returned.

If the constants c and L are both equal to 1, it means that the algorithm works just like the brute-force search. It should be noted in case of probabilistic roadmap methods that if c is larger than 1, also L must be larger than 1. Otherwise, the roadmap would be built separately in each Voronoi cell and they would not be connected together. By increasing L there will be overlapping between the cells which helps the roadmap to get connected.

4 Experiments

In our experiments, we tested how different nearest neighbor methods work with probabilistic roadmap planners. The methods we used were a brute-force search, *kd*-tree method, and centroid-based locality-sensitive hashing. We used three environments in our tests: an asteroids environment, a house environment, and a wall environment

Algorithm 5 Returns k nearest nodes to a query point q from a set of roadmap nodes R

1: **if** $|R| \leq k$ **then**
2: **return** R
3: $V \leftarrow \emptyset$
4: **for** $i = 1, 2, \ldots, L$ **do**
5: $h \leftarrow$ hash value for q in i-th hash table
6: $b \leftarrow$ bucket identified by a value h from i-th hash table
7: $V' \leftarrow$ all nodes from bucket b
8: $V \leftarrow V \cup V'$
9: **if** $|V| < k$ **then**
10: **return** k nearest nodes to q from R
11: **else**
12: **return** k nearest nodes to q from V

(see Figures 2–4). In each environment we tested the neighbor methods with three robots. The robots were composed of a different number of rigid bodies which means that for each robot the configuration space had a different dimension. Each method was also tested with two values for k which determines how many neighbors are retrieved each time the planner wants to find the nearest neighbors (see line 6 in Algorithm 1). To minimize the effect of randomness we made 1000 test runs for each test case.

Our goal was to measure how much different neighbor methods affected the time required to construct a roadmap and to investigate if the LSH method reduces the quality of the produced roadmap. We also tested several different parameters for a LSH method to demonstrate their effects.

The size of the roadmap that we built was 20000 nodes for the asteroids and wall environments, and 30000 nodes for the house environment. We also had a predefined query that we tried to solve. We measured the used time, number of roadmap components, length of the found path and how often the method failed to solve the predefined query. The measurements were taken when the construction of the roadmap was finished, at the point when the predefined query had just been solved, and when the roadmap had 10000 nodes.

4.1 Test setup

In every test, the node sampling method, local planner, and used distance metric were the same. Besides the used environments and robots, the only thing that changed between the tests was the used nearest neighbor method. This makes the results we obtained comparable with each other.

In the node sampling we used a random method where new configurations were sampled from the free configuration space uniformly. The local planner we used was a simple straight-line planner. The distance metric was the same that was described in

Figure 2: An asteroids environment. All robot bodies are shown at a start location (left side) and at a goal location (right side).

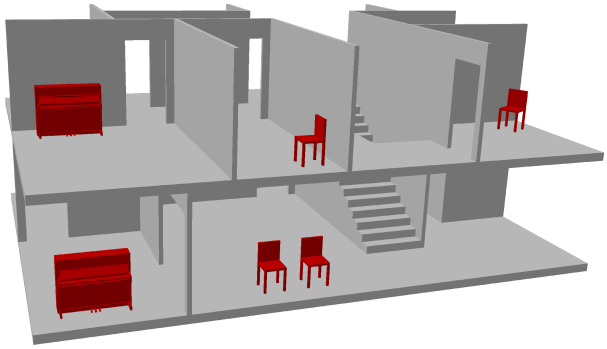


Figure 3: A house environment (shown without the roof and exterior walls). All robot bodies are shown at a start location (downstairs) and at a goal location (upstairs).

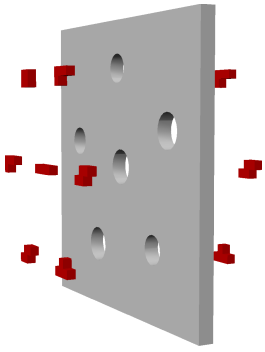


Figure 4: A wall environment. All robot bodies are shown at a start location (left side) and at a goal location (right side).

Section 2. We made these choices because these are probably the simplest methods and they require only a minimal number of additional parameters.

All methods were implemented in C++ to the same software framework. Proximity Query Package (PQP) library[41] was used for collision detection. All tests were run on a PC with Intel i7-2600 (3.40 GHz) processor and 8 GB of memory.

4.2 Environments

Each test environment consisted of static obstacles and a robot that had several independent rigid body parts. The obstacles and the robot were represented as triangle mesh. Next we describe the used environments in more detail.

Asteroids environment Obstacles are small asteroids of different sizes and shapes that are spread throughout the space. The environment was generated randomly. A robot consists of several spaceship-shaped parts. There are three different robots: the first has one part, the second has three parts, and the third has five parts. In a predetermined query, the robot must go through the asteroids from one side to the other. The workspace size is $400 \times 100 \times 100$ units. There are 100 asteroids and together they have 32000 triangles. Each spaceship has 64 triangles. The environment is illustrated in Figure 2.

House environment Obstacles of the house consist of walls, floors, stairs, and a roof. There are three different robots: the first is just a piano, the second consists of a piano and one chair, and the third consists of a piano and two chairs. In a predetermined query, all robot parts are located on the first floor and the goal is to find a way to move them to the second floor. The workspace size is $511 \times 354 \times 220$ units. The house consists of 500 triangles in total. The piano has 519 triangles and each of the chairs has 136 triangles. The environment is illustrated in Figure 3.

Wall environment An obstacle is a wall that has several circular holes in it and a robot consists of several tetromino-shaped parts. There are three different robots: the first has three parts, the second has five, and the third has seven. In a predetermined query, the robot must find a way to go from one side of the wall to the other side. The work space size is $100 \times 100 \times 100$ units. The wall consists of 572 triangles. Each robot part has 20 triangles on average. The environment is illustrated in Figure 4.

All three environments have their own characteristic properties. The asteroids environment is an cluttered environment that is filled with obstacles. There are not large empty areas anywhere which makes it difficult for the local planner to find free paths. On the other hand, the wall environment has large empty areas. The only obstacle is the wall that divides the space into two parts. The difficulty is to find a path through

the holes in the wall. The house environment can be thought to be a combination of these other environments. It has many obstacles but also many empty areas where the robot can move freely.

In addition to the obstacles, the environments also differ by their size and shape. The work space where the robot moves is bounded and the wall environment has the smallest working area in contrast to house environment which has the largest. In the wall environment the shape of the work space is a cube whereas in other environments it is rectangular cuboid. The shape of the work space can have an impact on the performance of the nearest neighbor methods like *kd*-tree which depends on dividing the space by axis-aligned planes.

5 Results

Results of our experiments are shown in Tables 1–3. Except for the success rate, all results are the averaged values of 1000 test runs with standard deviation shown in parentheses. The method column shows the used nearest neighbor method and in the case of a LSH method also used values for parameters L and c are shown. The column k shows how many neighbors were retrieved at maximum in each time neighbors were searched for. The robots column shows how many bodies the robot had. The columns marked with text "Path found" show roadmap properties at the time when a predetermined query was solved. The rest of the columns show properties for certain roadmap sizes. It should be noted that once the predetermined query has been solved, the length of the found path does not change even if the number of roadmap nodes grows. This is because the roadmap does not contain cycles and therefore there can be only one path between two nodes.

Because the brute-force method and the *kd*-tree method are exact, it is expected that they give similar results when applied to the probabilistic roadmap planners. The only difference should be the time required to build the roadmap. By looking at the results in Tables 1–3, we can see that it is the case. The number of roadmap components are approximately the same as well as the success rates and the lengths of the found paths. The numbers are not exactly the same because of the random nature of PRM planners. This effect can be seen also by looking at standard deviations as they can be quite high at some cases. The results also show that the *kd*-tree method slows down when the dimension of the configuration space increases and that it will eventually be slower than the brute-force method.

By comparing the different methods, it can be seen that locality-sensitive hashing speeded up the construction of the roadmap almost in all cases when compared with the exact methods. When looking at the times required to build the whole roadmap, it can be seen that the only exceptions were the cases where there was only one robot body, i.e., the dimension of the configuration space was small. In these cases, the *kd*-tree method was slightly faster. With higher dimensions, the LSH method is considerably faster

than either exact method. When looking at the results it should be remembered that when the roadmap size is very small, our LSH-based method works like the brute-force method. Figures 5–7 illustrate how different methods affect a time used to build the roadmap. In the figures, the used value for parameter k was 100 and the values for LSH parameters L and c were 150 and 20 respectively.

One important aspect that must be taken under consideration when using approximation neighbor methods is the quality of the produced roadmap which should not differ too much between the exact and approximate methods. We measured the quality of each method by calculating the number of roadmap components in certain roadmap sizes and solving a predetermined path query. On average, all methods should solve the query with the same number of nodes and the length of the found path should be the same. Moreover, the success rate and the number of components should also be the same and the success rate should increase as the number of roadmap nodes grows. The results in Tables 1–3 show that locality-sensitive hashing can achieve a similar quality as the exact methods with faster running time. However, the quality of the LSH method depends on its parameters.

For comparison, we ran the tests with three different sets of parameters for a LSH method. From these parameters we obtained the best results with 20 hash tables (parameter L) and 150 centroids (parameter c). These parameters gave quite good results in all environments and with different dimensions for a configuration space. By decreasing the number of hash tables to 5 the planner worked faster but also the quality of the roadmap decreased. Increasing the number of centroids to 300 did not have as large effects but, for example, the success rates in the wall environment decreased as can be seen in Table 3. However, it seems that there is no need for a major fine-tuning of the parameters to get good results with the LSH method.

By comparing the times required to build the whole roadmap with different values for k, it can be seen, that increasing the value also increases the used time. On the other hand, when k is large, the success rates are higher and the predefined query can be solved with a smaller roadmap. This is because k determines the number of configurations to which each new configuration is tried to be connected with a local planner. The planner requires more computation time with large values but at the same time, the connectivity of the roadmap increases. When the connectivity is high, the predetermined query can be solved with a smaller number of nodes and with a better success rate than with lower connectivity.

It can be noticed from the results that the time required to build the roadmap grows also when the number of robot bodies increases no matter what nearest neighbor method is used. There are several reasons for this. The first reason is that the collisions must be checked for each robot body which obviously requires more time as the number of bodies grows. The second reason is that when the robot consists of multiple bodies, the robot can collide with itself in some configurations. Therefore, it is not enough to check collisions with static obstacles because also self-collisions must be checked. The third reason has to do with a local planner which tries to connect a new configuration

Table 1: Average results of 1000 test runs for asteroids environment with standard deviations are shown in parentheses. SR means success rate. L means the number of hash tables, and c means the number of clusters.

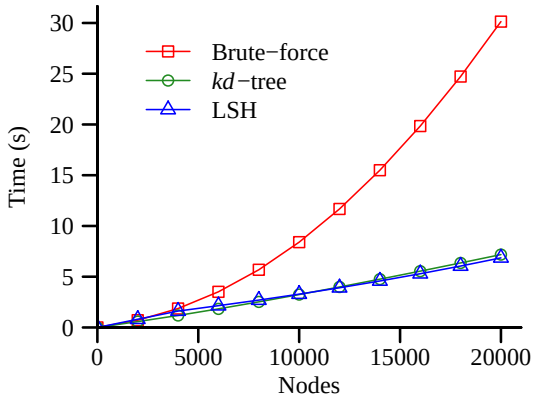
			Path found				10000 nodes			20000 nodes		
Method	k	Robots	Time (s)	Nodes	Components	Length	Time (s)	Components	SR	Time (s)	Components	SR
Brute-force	50	1	<0.05	41.1 (17.0)	2.3 (1.1)	641.1 (104.1)	8.1 (0.1)	4.1 (1.7)	100	29.3 (0.3)	7.0 (2.3)	100
kd-tree	50	1	<0.05	42.0 (16.9)	2.3 (1.2)	643.9 (107.8)	2.7 (0.1)	4.1 (1.7)	100	5.7 (0.2)	7.0 (2.2)	100
LSH (L: 20, c: 150)	50	1	<0.05	41.2 (15.9)	2.4 (1.2)	640.9 (104.9)	2.7 (<0.05)	4.0 (1.7)	100	6.2 (0.1)	6.9 (2.2)	100
LSH (L: 5, c: 150)	50	1	<0.05	42.1 (17.1)	2.3 (1.1)	644.4 (106.8)	2.5 (<0.05)	4.2 (1.7)	100	5.0 (<0.05)	7.0 (2.4)	100
LSH (L: 20, c: 300)	50	1	<0.05	40.5 (16.6)	2.3 (1.2)	641.5 (111.1)	3.3 (<0.05)	4.1 (1.7)	100	6.5 (0.1)	7.0 (2.3)	100
Brute-force	100	1	<0.05	41.6 (16.9)	2.3 (1.1)	639.2 (108.0)	8.4 (0.1)	4.1 (1.7)	100	30.1 (0.2)	6.9 (2.3)	100
kd-tree	100	1	<0.05	41.7 (16.2)	2.3 (1.1)	645.5 (104.7)	3.2 (0.1)	4.0 (1.6)	100	7.2 (0.2)	6.9 (2.3)	100
LSH (L: 20, c: 150)	100	1	<0.05	41.1 (16.2)	2.3 (1.1)	646.3 (106.1)	3.3 (<0.05)	4.0 (1.7)	100	6.9 (0.1)	6.9 (2.3)	100
LSH (L: 5, c: 150)	100	1	<0.05	40.8 (16.4)	2.2 (1.1)	644.3 (112.5)	3.9 (0.1)	4.1 (1.7)	100	6.5 (0.1)	6.9 (2.3)	100
LSH (L: 20, c: 300)	100	1	<0.05	42.1 (16.6)	2.3 (1.1)	644.9 (110.4)	5.2 (0.1)	4.1 (1.8)	100	8.4 (0.1)	7.0 (2.4)	100
Brute-force	50	3	2.2 (0.5)	625.8 (162.7)	147.7 (15.0)	3505.3 (922.1)	35.6 (0.3)	290.9 (17.1)	100	100.5 (0.4)	348.3 (19.3)	100
kd-tree	50	3	2.2 (0.5)	616.3 (162.3)	147.9 (15.6)	3517.1 (959.6)	31.7 (1.0)	291.4 (17.3)	100	79.3 (3.6)	350.5 (18.6)	100
LSH (L: 20, c: 150)	50	3	2.2 (0.5)	621.1 (168.9)	148.3 (16.1)	3555.5 (928.2)	20.4 (0.2)	291.6 (17.0)	100	39.2 (0.4)	350.4 (19.3)	100
LSH (L: 5, c: 150)	50	3	2.2 (0.5)	622.3 (169.9)	148.9 (16.2)	3473.7 (936.3)	19.7 (0.3)	330.8 (18.7)	100	35.5 (0.4)	388.4 (20.6)	100
LSH (L: 20, c: 300)	50	3	2.2 (0.5)	616.0 (162.8)	147.9 (15.8)	3451.2 (911.0)	20.8 (0.2)	290.5 (17.6)	100	38.4 (0.3)	349.1 (19.1)	100
Brute-force	100	3	3.3 (0.7)	592.5 (144.2)	140.9 (13.4)	3474.8 (857.7)	40.0 (0.4)	201.7 (14.5)	100	105.9 (0.5)	227.9 (15.1)	100
kd-tree	100	3	3.4 (0.7)	596.7 (155.2)	141.5 (13.3)	3431.2 (915.5)	36.8 (1.0)	201.8 (14.0)	100	87.9 (3.5)	228.2 (15.3)	100
LSH (L: 20, c: 150)	100	3	3.3 (0.8)	595.1 (158.4)	140.6 (13.4)	3455.7 (932.6)	25.1 (0.3)	202.1 (14.2)	100	44.9 (0.4)	228.2 (14.7)	100
LSH (L: 5, c: 150)	100	3	3.3 (0.8)	598.5 (156.6)	140.8 (13.3)	3471.4 (929.6)	25.8 (0.3)	240.6 (16.5)	100	42.7 (0.4)	265.8 (16.1)	100
LSH (L: 20, c: 300)	100	3	3.4 (0.8)	600.6 (156.8)	141.4 (13.2)	3472.7 (928.4)	26.2 (0.3)	203.4 (14.6)	100	44.7 (0.4)	228.8 (15.3)	100
Brute-force	50	5	132.8 (37.0)	14736.9 (2832.4)	7122.4 (889.5)	10638.1 (3512.0)	74.6 (0.3)	5586.2 (71.7)	4	207.0 (0.4)	8685.1 (92.9)	79
kd-tree	50	5	133.4 (35.5)	14877.3 (2806.8)	7166.0 (885.6)	10687.8 (3570.3)	75.1 (1.3)	5587.3 (68.6)	4	202.6 (4.2)	8685.3 (88.5)	79
LSH (L: 20, c: 150)	50	5	76.9 (17.1)	14772.0 (2983.9)	7134.2 (950.1)	10530.9 (3681.8)	49.8 (0.3)	5590.5 (68.6)	4	107.3 (0.6)	8691.0 (89.4)	80
LSH (L: 5, c: 150)	50	5	73.8 (15.0)	15041.0 (2891.2)	7386.4 (930.1)	10949.9 (3734.6)	47.7 (0.3)	5723.0 (70.7)	4	99.5 (0.4)	8888.4 (93.0)	79
LSH (L: 20, c: 300)	50	5	76.0 (16.0)	14768.7 (2926.6)	7131.4 (927.1)	10858.1 (3707.3)	50.0 (0.4)	5587.6 (69.3)	5	104.6 (0.8)	8692.8 (87.1)	79
Brute-force	100	5	162.7 (46.3)	13887.6 (3033.0)	6279.6 (797.0)	10083.6 (3407.6)	104.9 (0.4)	5242.3 (71.8)	9	262.5 (0.6)	7748.9 (92.9)	89
kd-tree	100	5	166.5 (46.3)	13831.3 (2982.6)	6265.8 (789.7)	10262.5 (3413.3)	108.3 (0.9)	5242.9 (66.1)	10	268.9 (3.2)	7753.2 (82.3)	90
LSH (L: 20, c: 150)	100	5	111.5 (25.0)	13793.8 (3035.7)	6252.3 (806.0)	10172.0 (3399.0)	80.2 (0.3)	5246.5 (69.9)	11	162.4 (0.5)	7753.9 (87.1)	90
LSH (L: 5, c: 150)	100	5	111.6 (22.2)	14103.0 (2898.4)	6583.0 (789.0)	10415.3 (3550.8)	80.0 (0.4)	5432.5 (68.3)	8	156.2 (0.7)	8057.9 (88.3)	89
LSH (L: 20, c: 300)	100	5	113.5 (23.7)	13939.1 (2960.7)	6305.5 (776.0)	10164.7 (3335.4)	81.9 (0.4)	5258.1 (71.2)	10	161.4 (0.6)	7762.4 (90.3)	89

Table 2: Average results of 1000 test runs for house environment with standard deviations are shown in parentheses. SR means success rate. L means the number of hash tables, and c means the number of clusters.

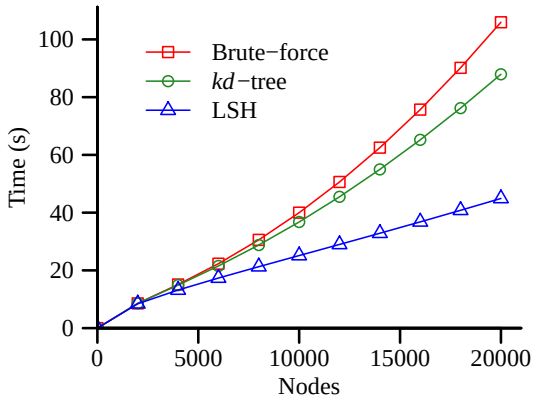
			Path found				10000 nodes			30000 nodes		
Method	k	Robots	Time (s)	Nodes	Components	Length	Time (s)	Components	SR	Time (s)	Components	SR
Brute-force	50	1	10.2 (13.6)	7917.1 (6995.0)	9.7 (3.7)	1407.1 (228.4)	10.0 (0.1)	8.1 (2.2)	63	68.8 (0.4)	3.9 (1.4)	90
kd-tree	50	1	3.7 (3.0)	8080.2 (7320.6)	9.6 (3.8)	1406.6 (237.1)	4.4 (0.1)	8.2 (2.3)	63	12.9 (0.4)	3.9 (1.4)	90
LSH (L: 20, c: 150)	50	1	4.0 (3.4)	8234.5 (7353.1)	9.4 (3.7)	1397.7 (213.9)	4.6 (0.1)	8.3 (2.3)	61	15.3 (0.1)	3.9 (1.3)	89
LSH (L: 5, c: 150)	50	1	3.9 (3.0)	8553.2 (7716.8)	9.7 (3.7)	1405.0 (234.6)	4.4 (0.1)	8.8 (2.4)	60	12.8 (0.1)	4.3 (1.6)	89
LSH (L: 20, c: 300)	50	1	4.5 (3.4)	8507.1 (7441.7)	9.5 (3.7)	1413.3 (244.5)	5.2 (0.1)	8.3 (2.3)	60	14.8 (0.1)	4.0 (1.4)	90
Brute-force	100	1	7.4 (10.0)	5743.0 (5593.0)	9.2 (3.4)	1409.2 (233.2)	11.1 (0.2)	6.2 (1.9)	83	68.8 (0.2)	2.7 (0.8)	98
kd-tree	100	1	3.8 (2.6)	5679.9 (5389.9)	9.3 (3.4)	1392.0 (231.8)	5.8 (0.2)	6.3 (1.9)	83	15.6 (0.3)	2.6 (0.8)	98
LSH (L: 20, c: 150)	100	1	4.2 (2.7)	5617.6 (5251.5)	9.1 (3.3)	1411.9 (237.3)	6.3 (0.2)	6.2 (1.9)	82	17.2 (0.2)	2.7 (0.9)	98
LSH (L: 5, c: 150)	100	1	4.4 (2.9)	5674.2 (5640.4)	9.3 (3.4)	1405.8 (229.4)	6.9 (0.2)	6.4 (1.9)	81	15.5 (0.2)	2.9 (1.0)	98
LSH (L: 20, c: 300)	100	1	5.0 (3.4)	5686.0 (5541.2)	9.2 (3.3)	1387.6 (236.2)	8.1 (0.2)	6.2 (1.9)	82	17.8 (0.2)	2.7 (0.9)	97
Brute-force	50	2	46.4 (35.6)	13486.9 (7586.5)	140.9 (11.2)	4580.3 (1121.5)	26.6 (0.3)	138.1 (9.8)	34	141.5 (0.6)	147.2 (10.5)	88
kd-tree	50	2	26.9 (14.4)	13405.1 (7285.8)	140.9 (10.7)	4571.8 (1100.0)	20.1 (0.4)	137.9 (10.2)	33	60.2 (1.6)	147.2 (10.5)	87
LSH (L: 20, c: 150)	50	2	20.9 (11.0)	13142.5 (7388.6)	140.8 (10.6)	4692.3 (1172.6)	16.2 (0.2)	138.2 (10.4)	36	47.1 (0.4)	146.9 (10.2)	87
LSH (L: 5, c: 150)	50	2	20.5 (9.1)	14092.3 (7349.5)	149.6 (11.0)	4677.7 (1158.2)	15.7 (0.2)	148.6 (10.9)	30	40.3 (0.3)	154.9 (10.6)	87
LSH (L: 20, c: 300)	50	2	20.7 (9.7)	13124.8 (7136.1)	140.5 (10.9)	4667.0 (1140.7)	16.6 (0.2)	137.6 (10.1)	35	43.8 (0.4)	146.5 (10.5)	86
Brute-force	100	2	37.9 (29.7)	10517.4 (6565.2)	128.0 (11.8)	4575.1 (1124.9)	31.8 (0.6)	124.1 (9.2)	53	149.1 (1.1)	129.4 (9.2)	97
kd-tree	100	2	29.1 (17.1)	10547.3 (6490.6)	128.0 (11.6)	4601.9 (1111.0)	27.6 (0.8)	123.7 (9.4)	53	80.7 (2.3)	129.3 (9.2)	97
LSH (L: 20, c: 150)	100	2	22.5 (11.3)	10639.8 (6671.6)	127.8 (11.7)	4643.1 (1121.3)	21.9 (0.6)	123.5 (9.3)	52	54.7 (1.0)	129.4 (10.0)	96
LSH (L: 5, c: 150)	100	2	23.7 (10.4)	11348.8 (6840.5)	135.1 (10.4)	4598.6 (1147.9)	22.6 (0.7)	134.3 (9.9)	47	49.6 (1.1)	138.9 (10.2)	94
LSH (L: 20, c: 300)	100	2	23.6 (10.7)	10755.7 (6667.4)	127.6 (11.7)	4609.7 (1128.5)	23.3 (0.6)	123.4 (9.7)	52	52.3 (1.0)	129.0 (10.0)	96
Brute-force	50	3	188.9 (52.9)	23718.1 (4983.7)	2488.3 (130.8)	9969.3 (2393.7)	57.1 (0.3)	2069.3 (40.0)	0	263.5 (0.6)	2604.7 (45.6)	6
kd-tree	50	3	165.3 (40.2)	23519.5 (4618.2)	2486.3 (116.1)	9968.1 (2462.5)	56.3 (0.6)	2070.6 (38.5)	0	224.9 (3.7)	2607.8 (44.8)	7
LSH (L: 20, c: 150)	50	3	102.9 (19.9)	23839.2 (4426.2)	2510.0 (96.7)	10260.0 (2340.1)	42.5 (0.3)	2072.6 (39.0)	0	130.9 (0.6)	2605.6 (47.6)	6
LSH (L: 5, c: 150)	50	3	94.9 (20.5)	23999.6 (5369.6)	2702.9 (147.8)	11066.4 (3196.6)	41.0 (0.3)	2253.1 (42.0)	0	117.7 (0.6)	2799.6 (50.3)	4
LSH (L: 20, c: 300)	50	3	98.2 (17.7)	23854.8 (4389.1)	2504.8 (113.4)	9951.1 (2608.4)	42.2 (0.3)	2075.6 (40.6)	0	122.8 (0.7)	2610.0 (46.9)	7
Brute-force	100	3	209.7 (57.7)	23398.6 (4985.5)	2238.0 (101.9)	9641.4 (2746.8)	70.2 (0.4)	1902.4 (37.3)	0	292.4 (1.2)	2331.1 (42.9)	16
kd-tree	100	3	201.1 (51.7)	23293.6 (4885.1)	2240.7 (98.9)	9637.0 (2571.8)	71.6 (0.7)	1902.4 (37.6)	0	275.0 (3.7)	2330.0 (43.8)	16
LSH (L: 20, c: 150)	100	3	125.8 (25.9)	23290.2 (4950.2)	2240.5 (121.2)	9908.8 (2670.1)	56.2 (0.5)	1907.6 (37.2)	0	161.2 (1.6)	2332.5 (43.2)	15
LSH (L: 5, c: 150)	100	3	123.0 (22.2)	24118.9 (4804.3)	2468.8 (98.6)	10139.9 (2508.8)	56.1 (0.5)	2091.9 (41.5)	0	149.6 (1.5)	2551.9 (44.5)	9
LSH (L: 20, c: 300)	100	3	121.3 (20.8)	23424.8 (4409.4)	2250.4 (79.3)	9852.7 (2426.8)	56.3 (0.4)	1912.9 (37.9)	0	152.1 (1.3)	2335.7 (43.0)	14

Table 3: Average results of 1000 test runs for wall environment with standard deviations are shown in parentheses. SR means success rate. L means the number of hash tables, and c means the number of clusters.

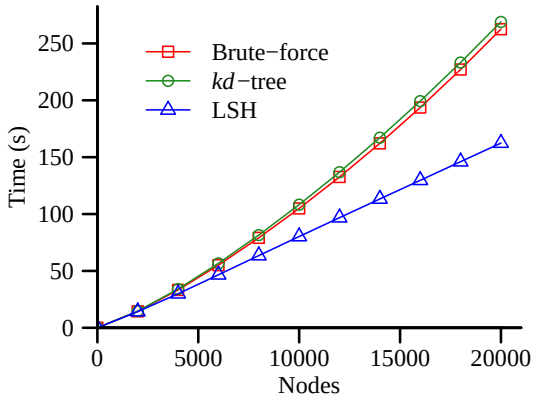
			Path found				10000 nodes			20000 nodes		
Method	k	Robots	Time (s)	Nodes	Components	Length	Time (s)	Components	SR	Time (s)	Components	SR
Brute-force	50	3	0.7 (0.5)	1008.7 (557.6)	2.7 (1.3)	1744.0 (539.2)	18.6 (0.1)	9.3 (3.0)	100	71.4 (0.2)	17.6 (4.1)	100
kd-tree	50	3	0.8 (0.5)	1044.2 (587.3)	2.7 (1.3)	1748.7 (525.8)	19.2 (0.4)	9.3 (2.9)	100	67.3 (1.0)	17.5 (4.1)	100
LSH (L: 20, c: 150)	50	3	0.6 (0.3)	1066.4 (615.6)	2.8 (1.3)	1786.2 (552.0)	4.2 (0.1)	9.4 (2.9)	100	12.5 (0.2)	17.7 (4.2)	100
LSH (L: 5, c: 150)	50	3	0.7 (0.4)	1052.9 (616.9)	2.8 (1.3)	1730.3 (517.6)	2.9 (0.1)	9.3 (2.9)	100	6.6 (0.2)	17.6 (4.1)	100
LSH (L: 20, c: 300)	50	3	0.8 (0.4)	1063.2 (647.4)	2.9 (1.3)	1758.2 (534.0)	4.1 (0.1)	9.4 (2.9)	100	10.0 (0.2)	17.6 (4.1)	100
Brute-force	100	3	0.7 (0.3)	606.3 (287.6)	2.4 (1.2)	1689.8 (502.1)	19.1 (0.2)	9.4 (2.9)	100	72.4 (0.2)	17.7 (4.1)	100
kd-tree	100	3	0.7 (0.3)	596.0 (271.9)	2.4 (1.1)	1676.7 (479.5)	22.3 (0.4)	9.4 (2.8)	100	80.9 (1.1)	17.8 (4.1)	100
LSH (L: 20, c: 150)	100	3	0.7 (0.3)	590.7 (270.6)	2.4 (1.2)	1704.2 (533.5)	4.6 (0.2)	9.4 (2.9)	100	13.0 (0.2)	17.8 (4.1)	100
LSH (L: 5, c: 150)	100	3	0.7 (0.3)	599.9 (273.6)	2.5 (1.2)	1709.0 (541.1)	4.6 (0.2)	9.3 (2.9)	100	8.5 (0.3)	17.6 (4.1)	100
LSH (L: 20, c: 300)	100	3	0.7 (0.3)	599.8 (281.7)	2.4 (1.2)	1681.1 (517.7)	5.2 (0.2)	9.2 (2.9)	100	11.2 (0.2)	17.5 (4.0)	100
Brute-force	50	5	12.8 (7.0)	4901.0 (1715.4)	11.9 (3.7)	5426.9 (2047.6)	35.6 (0.5)	13.4 (3.5)	98	123.2 (0.6)	25.2 (4.9)	100
kd-tree	50	5	15.2 (9.7)	4919.4 (1749.2)	11.8 (3.7)	5656.3 (2068.0)	46.3 (0.8)	13.5 (3.8)	98	175.6 (1.3)	25.2 (5.1)	100
LSH (L: 20, c: 150)	50	5	7.0 (2.4)	4978.8 (1783.5)	11.9 (3.7)	5540.5 (1945.6)	13.1 (0.6)	13.4 (3.5)	98	32.7 (0.8)	25.1 (4.8)	100
LSH (L: 5, c: 150)	50	5	7.0 (1.8)	5453.7 (1967.6)	15.2 (4.4)	5742.0 (2035.8)	10.2 (0.6)	15.4 (3.8)	97	18.8 (0.6)	26.5 (5.1)	100
LSH (L: 20, c: 300)	50	5	7.4 (2.2)	5097.2 (1779.0)	12.0 (3.8)	5634.1 (2149.7)	12.4 (0.6)	13.4 (3.4)	98	26.2 (0.6)	25.2 (4.8)	100
Brute-force	100	5	8.8 (3.4)	3085.3 (1035.4)	9.5 (3.7)	5195.7 (1981.0)	37.3 (0.8)	12.8 (3.3)	100	125.3 (0.8)	24.4 (4.8)	100
kd-tree	100	5	9.9 (4.2)	3160.8 (1059.4)	9.3 (3.6)	5295.4 (1909.1)	48.1 (1.0)	12.9 (3.5)	100	177.8 (1.5)	24.6 (4.8)	100
LSH (L: 20, c: 150)	100	5	7.0 (1.8)	3234.8 (1161.9)	9.5 (3.7)	5315.4 (1944.5)	15.0 (0.8)	12.9 (3.5)	100	34.6 (0.9)	24.6 (4.9)	100
LSH (L: 5, c: 150)	100	5	8.0 (1.9)	3263.7 (1193.4)	10.1 (3.6)	5298.3 (1960.5)	13.2 (0.9)	13.7 (3.5)	100	21.9 (0.9)	25.1 (4.9)	100
LSH (L: 20, c: 300)	100	5	7.9 (1.8)	3297.4 (1136.4)	9.5 (3.7)	5347.8 (1967.1)	14.7 (0.8)	12.8 (3.5)	100	28.4 (0.8)	24.4 (5.0)	100
Brute-force	50	7	158.7 (26.8)	17681.7 (1827.5)	129.4 (12.4)	11655.2 (4808.5)	60.4 (0.2)	165.2 (11.1)	0	195.5 (0.6)	126.4 (12.1)	15
kd-tree	50	7	204.6 (37.4)	17722.0 (1957.7)	128.7 (12.6)	11757.0 (4587.6)	72.7 (0.5)	165.0 (11.4)	0	253.3 (1.0)	126.4 (12.2)	15
LSH (L: 20, c: 150)	50	7	63.4 (8.0)	17684.2 (1694.8)	128.8 (12.4)	11318.4 (4435.1)	30.4 (0.2)	164.9 (11.0)	0	74.7 (0.9)	125.8 (11.9)	14
LSH (L: 5, c: 150)	50	7	47.5 (4.8)	18193.5 (1704.4)	247.9 (24.0)	11782.1 (4904.7)	25.1 (0.1)	310.1 (20.2)	0	52.7 (0.4)	244.9 (19.7)	5
LSH (L: 20, c: 300)	50	7	57.1 (5.8)	18014.0 (1572.0)	127.2 (10.1)	11539.6 (4034.4)	29.1 (0.2)	166.3 (11.3)	0	64.7 (0.7)	125.9 (11.5)	12
Brute-force	100	7	140.0 (35.5)	15157.2 (2552.0)	58.8 (10.6)	13131.6 (5109.7)	73.5 (0.3)	85.3 (9.0)	2	212.1 (2.2)	50.7 (6.9)	72
kd-tree	100	7	174.6 (45.5)	15253.5 (2462.0)	58.6 (10.6)	13055.3 (5029.1)	86.2 (0.4)	85.3 (9.0)	2	270.4 (2.3)	50.6 (7.2)	72
LSH (L: 20, c: 150)	100	7	70.1 (12.3)	15340.6 (2479.1)	59.6 (10.7)	12939.8 (5193.8)	43.7 (0.3)	87.5 (8.9)	1	91.7 (2.3)	51.3 (7.1)	72
LSH (L: 5, c: 150)	100	7	65.1 (7.0)	16876.3 (1975.0)	111.0 (15.8)	13700.1 (5004.6)	39.9 (0.2)	163.6 (13.0)	0	74.9 (2.0)	99.8 (11.8)	51
LSH (L: 20, c: 300)	100	7	65.1 (9.4)	15445.5 (2322.8)	60.8 (10.8)	13231.0 (4862.5)	42.5 (0.4)	89.8 (9.1)	0	81.7 (2.3)	52.1 (7.4)	66



(a) 1 robot body

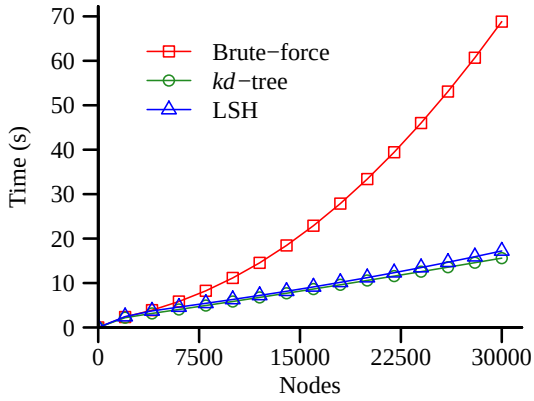


(b) 3 robot bodies

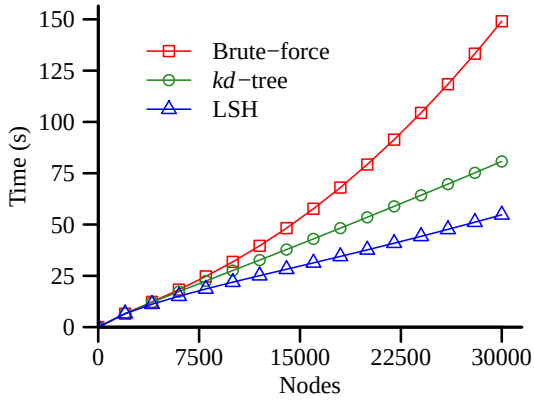


(c) 5 robot bodies

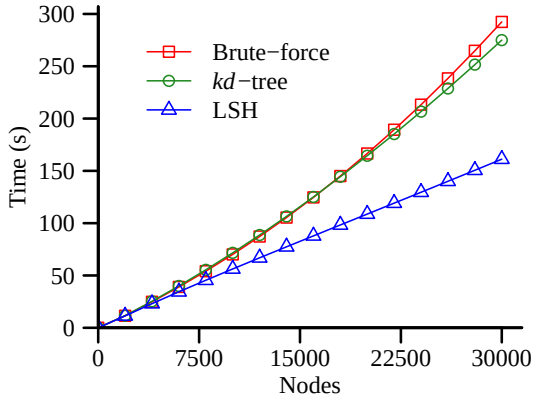
Figure 5: Graphs show how much time was used to build the roadmap in asteroids environment with different methods.



(a) 1 robot body

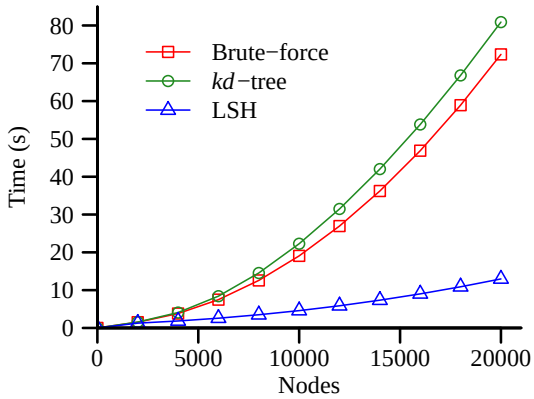


(b) 2 robot bodies

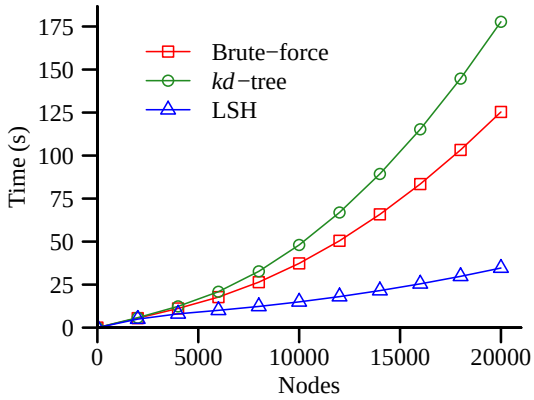


(c) 3 robot bodies

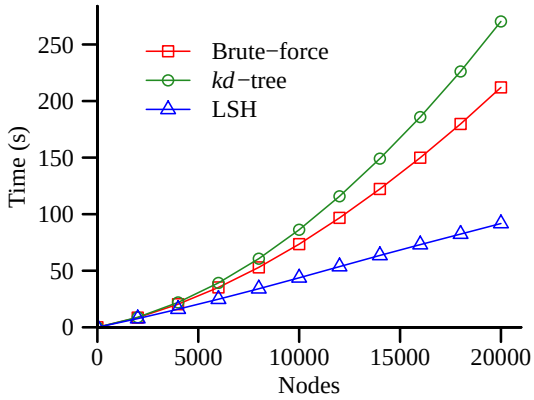
Figure 6: Graphs show how much time was used to build the roadmap in house environment with different methods.



(a) 3 robot bodies



(b) 5 robot bodies



(c) 7 robot bodies

Figure 7: Graphs show how much time was used to build the roadmap in wall environment with different methods.

to only those configurations that do not belong to the same component (see line 8 in Algorithm 1). When the number of robot bodies grows, it will be increasingly difficult for the local planner to find a free path between two configurations. Therefore more and more paths must be checked. This can be seen also from the number of components which grows rapidly when the number of robot bodies increases.

6 Conclusions

Probabilistic roadmap planners must use nearest neighbor methods to retrieve a set of neighboring configurations when a new configuration is added to the roadmap. Neighbors are usually searched for by exact methods which can unfortunately became a major bottleneck for the performance. This can occur when the roadmap size grows and especially when the configuration space is high-dimensional.

In this paper, we investigated how approximate nearest neighbors methods work with PRM planners. We focused on locality-sensitive hashing which is nowadays quite a popular method for approximate neighbor search and which has successfully been used in many applications. We compared the LSH method with two exact methods which were the brute-force search and the *kd*-tree method.

Our experiments showed that it is indeed feasible to use approximate nearest neighbor search when constructing the roadmap. The approximate search can speed up the roadmap construction phase considerably. Our experiments also showed that *kd*-tree is not a good method to be used in high-dimensional spaces as it will eventually became even slower than the brute-force method.

We also compared the quality of the roadmaps produced with different methods. With good parameters for the LSH method, there were no significant differences in quality between the LSH method and the exact methods. The roadmap was able to solve the predetermined query with the same success rate and the length of the found path was approximately the same. We also measured the number of roadmap components in certain roadmap sizes and noticed that there were no big differences between the methods.

The problem with the LSH method is that it requires parameters that must be selected manually to suit each motion planning problem. However, LSH methods generally work faster than exact algorithms and produce reasonably good quality roadmaps even with bad choices for parameter values. Because of this, it seems that there is no need for an extreme fine-tuning of the parameters. Still, it would be worth to investigate whether these parameters could be selected automatically. Another interesting idea for future research is to compare how other approximate nearest neighbor methods works when compared with locality-sensitive hashing.

Acknowledgement

The research of the first author was supported by the Tampere Doctoral Programme in Information Science and Engineering (TISE).

References

[1] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[2] 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, Cambridge, MA, 2005.

[3] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[4] N. Dadkhah and B. Mettler. Survey of motion planning literature in the presence of uncertainty: considerations for UAV guidance. *Journal of Intelligent & Robotic Systems*, 65:233–246, 2012.

[5] J. Smed and H. Hakonen. Path finding. In *Algorithms and Networking for Computer Games*, pages 97–113. John Wiley & Sons, 2006.

[6] I. Al-Bluwi, T. Siméon, and J. Cortés. Motion planning algorithms for molecular simulations: A survey. *Computer Science Review*, 6(4):125–143, 2012.

[7] T. Lozano-Pérez. Spatial planning: A configuration space approach. *IEEE Transactions on Computing*, C-32(2):108–120, 1983.

[8] J. H. Reif. Complexity of the mover's problem and generalizations. In *Proceedings of the IEEE Symposium on Foundations of Computer Science*, pages 421–427, 1979.

[9] J. F. Canny. *The Complexity of Robot Motion Planning*. MIT Press, Cambridge, MA, 1988.

[10] M. H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-93, Department of Computer Science, Utrecht University, the Netherlands, 1992.

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

[12] 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 & Automation*, 12(4):566–580, 1996.

[13] L. Jaillet and T. Siméon. A PRM-based motion planner for dynamically changing environments. In *Proceedings of IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[14] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha. Real-time navigation of independent agents using adaptive roadmaps. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology*, 2007.

[15] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[16] M. Muja and D. G. Lowe. Fast approximate nearest neighbors with automatic algorithm confisguration. In *Proceedings of the International Conference on Computer Vision Theory and Application*, pages 331–340, 2009.

[17] P. Indyk and R. Motwani. Approximate nearest neighbors: towards removing the curse of dimensionality. In *Proceedings of ACM symposium on Theory of computing*, pages 604–613, 1998.

[18] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *Proceedings of International Conference on Very Large Data Bases*, pages 518–529, 1999.

[19] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *Communications of the ACM*, 51(1):117–122, 2008.

[20] L. Kocis and W. J. Whiten. Computational investigations of low-discrepancy sequences. *ACM Transactions on Mathematical Software*, 23(2):266–294, 1997.

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

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

[23] D. Hsu, T. Jiang, J. Reif, and Z. Sun. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of IEEE International Conference on Robotics & Automation*, 2003.

[24] 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 *Proceedings of IEEE International Conference on Robotics & Automation*, pages 1024–1031, 1999.

[25] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *Proceedings of IEEE International Conference on Robotics & Automation*, pages 1408–1413, 2000.

[26] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. In M. Erdmann, M. Overmars, D. Hsu, and F. van der Stappen, editors, *Algorithmic Foundations of Robotics VI*, pages 361–376. Springer, Berlin, 2005.

[27] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato. RESAMPL: A region-sensitive adaptive motion planner. In S. Akella, N. M. Amato, W. H. Huang, and B. Mishra, editors, *Algorithmic Foundation of Robotics VII*, pages 285–300. Springer, Berlin, 2008.

[28] M. T. Rantanen. A connectivity-based method for enhancing sampling in probabilistic roadmap planners. *Journal of Intelligent and Robotic Systems*, 64(2):161–178, 2011.

[29] N. M. Amato, O. B. Bayazit, L. K. Dale, C. Jones, and D. Vallejo. Choosing good distance metrics and local planners for probabilistic roadmap methods. *IEEE Transactions on Robotics & Automation*, 16(4):442–447, 2000.

[30] R. Geraerts and M. H. Overmars. Reachability-based analysis for probabilistic roadmap planners. *Robotics and Autonomous Systems*, 55(11):824–836, 2007.

[31] J. J. Kuffner. Effective sampling and distance metrics for 3D rigid body path planning. In *Proceedings of IEEE International Conference on Robotics & Automation*, 2004.

[32] R. Weber, H.-J. Schek, and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *Proceedings of International Conference on Very Large Databases*, 1998.

[33] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[34] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications, 2nd Ed.* Springer-Verlag, Berlin, 2000.

[35] A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.

[36] M. Ryynänen and A. Klapuri. Query by humming of MIDI and audio using locality sensitive hashing. In *Proceedings of IEEE International Conference on Acoustics, Speech, and Signal Processing*, 2008.

[37] R. Buaba, A. Homaifar, M. Gebril, and E. Kihn. Satellite image retrieval application using locality sensitive hashing in l_2-space. In *Proceedings of IEEE Aerospace Conference*, 2011.

[38] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of Symposium on Computational Geometry*, 2004.

[39] L. Paulevé, H. Jégou, and L. Amsaleg. Locality sensitive hashing: A comparison of hash function types and querying mechanisms. *Pattern Recognition Letters*, 31(11):1348–1358, 2010.

[40] F. Aurenhammer. Voronoi diagrams–a survey of a fundamental geometric data structure. *ACM Computing Surveys*, 23(3):345–405, 1991.

[41] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast proximity queries with swept sphere volumes. Technical report, Technical Report TR99-018, Department of Computer Science, University of North Carolina, 1999.

Publication V

How to construct small probabilistic roadmaps with a good coverage?

Mika T. Rantanen and Martti Juhola

Manuscript submitted for publication.

How to construct small probabilistic roadmaps with a good coverage?

Mika T. Rantanen Martti Juhola

Computer Science, School of Information Sciences,
FI-33014 University of Tampere, Finland

Abstract

We investigate how we can construct small probabilistic roadmaps in a reasonable time while still keeping a good coverage and connectivity. We propose a new neighborhood-based method that can reduce the size of the roadmaps by filtering out unnecessary nodes. We then experimentally test it against a basic probabilistic roadmap planner and a visibility-based planner. We use both a uniform sampling and a bridge test sampling in our tests. The results show that the neighborhood-based method can reduce the number of nodes considerably. The neighborhood-based method is simple to implement, it works well with a uniform sampling and it does not need any additional parameters when compared with the basic planner.

1 Introduction

One of the most important problems in robotics is motion planning [1]. The robot moves in an environment that has obstacles and the goal is to find a collision-free path for a robot between two locations. The methods used to solve a motion planning problem in robotics can be used in other areas as well. For example, they have been successfully used in computer animation and games [2], in molecular simulations [3], and in computer-aided design [4].

A *work space* is a space where the robot moves and where the obstacles reside. The robot's exact position and orientation are described by a *configuration* and the set of all possible configurations is called a *configuration space* $\mathcal{C}$. A configuration where the robot does not collide with obstacles or itself is called a *free configuration*. The set of all free configurations is called a *free configuration space* $\mathcal{C}_{\text{free}} \subseteq \mathcal{C}$. It should be noted that while the work space is typically two- or three-dimensional, the configuration space can have much more dimensions depending on how many degrees of freedom the robot has.

The motion planning problem is to find a collision-free path for the robot from a start configuration to a goal configuration, if such path exists. In a basic case, the obstacles are static which means that they do not move or change their shape. The consequence is that the free configuration space does not change either. Therefore, the basic motion planning problem can be described as follows. Given a free configuration space $\mathcal{C}_{\text{free}}$ and two configurations q_{start} and q_{goal}, find a continuous function $\tau : [0, 1] \to \mathcal{C}_{\text{free}}$, where $\tau(0) = q_{\text{start}}$ and $\tau(1) = q_{\text{goal}}$. This function τ defines a path for a robot from q_{start} to q_{goal}. Because τ lies entirely in the free configuration space, the robot can move along it without colliding with the obstacles.

There are a variety of different approaches to solve the motion planning problem (see e.g. [1, 5, 6]). The early approaches included, among others, cell decomposition methods and potential field methods. The problem with these methods is that they do not scale well to high-dimensional and complex configuration spaces. On the other hand, sampling-based methods like *probabilistic roadmaps* (PRM) [7, 8, 9] and *rapidly-exploring random trees* (RRT) [10, 11] have been experimentally shown to work well even in these difficult spaces (see e.g. [11, 12, 13]).

RRT planners are designed to answer one motion planning query whereas PRM planners can be used to solve many queries quickly after a preprocessing phase. In this paper, we concentrate on PRM planners. They try to build a graph called a *roadmap* that is a representation of the free configuration space. The nodes of the roadmap correspond to the collision-free configurations of the robot and an edge between two nodes means that there is a simple collision-free path between the corresponding configurations.

One problem with PRM planners is that they tend to produce quite large roadmaps (in terms of the number of nodes) especially in difficult environments. Too large roadmaps can lead to drastic memory consumption and also the time required to build and use the roadmap can grow too much to be practical. There have been many different kinds of enhancements proposed for PRM planners to handle these problems but most of them have concentrated on the speed. Moreover, when these different enhancements have been compared, the comparison has also concentrated mainly on the speed and not on the roadmap size.

Our goal is to investigate different methods that can produce small roadmaps in a reasonable time. Additionally, we want that the constructed roadmap has a good coverage and connectivity so that even difficult motion planning queries can be solved with it. We compare the methods experimentally in different environments. We also propose a new neighborhood-based method that can decrease the number of nodes considerably while still being fast and easy to implement.

2 Probabilistic roadmap planners

A typical probabilistic roadmap planner starts with an empty roadmap and then adds new nodes to it one by one. Nodes are added until some ending condition has been met.

The procedure to add one node has the following steps:

1. Sample a new configuration q from $\mathcal{C}_{\text{free}}$ randomly using some *sampling method* and add q to the roadmap R.

2. For q, select a set of *neighboring configurations* N from R.

3. For each neighbor $q' \in N$, try to connect q and q' with a *local planner* $\Delta(q, q')$. If the connection is collision-free, add an edge (q, q') to R.

The roadmap should have a good coverage and connectivity. The coverage means how well the roadmap covers $\mathcal{C}_{\text{free}}$. If the roadmap fully covers $\mathcal{C}_{\text{free}}$, it is possible to connect any configuration in $\mathcal{C}_{\text{free}}$ to the roadmap with a local planner. The connectivity measures how well the nodes in the roadmap are connected together. To have good connectivity, two nodes in the roadmap should be in the same connected component if there were a free path between the corresponding configurations in $\mathcal{C}_{\text{free}}$.

The efficiency of the planners can be modified by changing the sampling method, neighbor selection or the local planner. The simplest sampling method is to select the configurations uniformly at random from the $\mathcal{C}$ and then check whether the robot collides with the obstacles in that configuration. If q is collision-free, it is selected, otherwise a new configuration is sampled. Other node sampling methods are discussed in Section 3.1.

Neighboring configurations for q are usually the nearest configurations from the roadmap measured by some distance metric [14]. For each neighbor q', a local planner is used to check whether there is a simple collision-free path between q and q'. The simplest local planner just tries to connect the configurations with a straight-line. A more powerful local planner can connect nodes that the simpler planner cannot but usually they are slower and they likely work only with certain configuration space types.

The probabilistic roadmap planners can be modified also in other ways. For example, if the number of roadmap nodes has been defined before the actual planning, it is possible to divide the construction of the roadmap into two separate parts. In the first part, all nodes are sampled, and in the second part, a local planner is used to connect them together.

Another example is a case where the roadmap does not need to have cycles. The roadmap can then be built by requiring that edges are added only between nodes that are not already in the same connected component. This can speed up the roadmap building considerably since it is possible to ignore a lot of edges and this reduces the number of collision detections that must be made.

To find an actual path between two configurations by using a roadmap, the configurations must first be connected to the roadmap. This can be done with a local planner. After that, the path between the configurations can be found using some graph search algorithm and the final path τ is a connected sequence of shorter paths that are computed by the local planner Δ. The path is not found if the local planner fails to

connect the configurations to the roadmap or if the configurations are not in the same connected component of the roadmap while doing the graph search.

3 Decreasing the number of nodes

One problem with PRM planners is that they tend to produce large roadmaps. Especially when the configuration space contains narrow passages, PRM planners must produce a great number of samples to find a path through it. This problem was recognized in early PRM papers (see e.g. [9]). Typically, it is beneficial to try to reduce the size of the roadmap because smaller roadmaps use less memory than large roadmaps. Usually, it is also faster to handle small roadmaps.

In some applications it might be enough that the planner can return some path between two configurations. It may not be important how long the path is or whether the robot does any unnecessary moves while moving along the path. In these cases it is enough to produce a roadmap that just covers the free configuration space well and has good connectivity. The roadmap can be as small as possible and there is no need for cycles.

However, it may sometimes be important that there are additional nodes and cycles in the roadmap. They are useful, for example, in applications where the environment is dynamic and the obstacles can move or change their shape [15, 16, 17]. A large roadmap with cycles can provide many alternative paths for a robot to move between two configurations, which in turn helps the robot to avoid the moving obstacles.

Even in these cases where large roadmaps are important, it might be useful to first construct a small roadmap that has a good coverage of the configuration space and that goes through the narrow passages. Once this small roadmap has been built, it can be used as a base which can be expanded to have more nodes and additional cycles in meaningful places (see e.g. [18]).

In this paper, our goal is to investigate methods that can decrease the number of nodes in a roadmap. However, it is also possible to reduce the nodes of a path after it has been retrieved from the roadmap. Overall, the paths can easily be unnecessarily long and jittery and it might be useful to smooth them. Different techniques to process paths have been proposed, for example, in [19, 20]. These techniques can help to produce short paths with a high quality. Sometimes it might be desirable that the planner would not return exactly the same path between two configurations every time. This can be achieved by adding a small variation to the paths like suggested in [21].

Next we discuss different methods that can be used to decrease the size of the roadmaps.

3.1 Node sampling methods

Besides a uniform sampling, many methods to enhance node sampling have been proposed over the years (see e.g. [22, 23, 24, 25, 26, 27]). They typically try to bias

the sampling towards difficult or otherwise interesting parts of the configuration space which usually leads to a better performance in terms of speed. This is because the planner does not need to spend time on nodes in the easy regions of $\mathcal{C}_{\text{free}}$. As a side effect, these techniques usually also decrease the number of nodes of the final roadmap because they can often cover the free configuration space with fewer nodes than uniform sampling.

Some sampling methods try to sample nodes near the obstacle boundaries. Such a method is OBPRM [22] which generates a random node that is in collision and then uses binary search to find free configurations near the surface of the obstacle. These free configurations are then added to the roadmap. UOBPRM [27] resembles OBPRM in many ways. It generates a random line segment in configuration space and finds points where it intersects with the obstacles. The free configurations near these intersection points are then added to the roadmap. The Gaussian sampler [23] generates two random configurations in such a way that the distance between these two configurations is selected according to a normal distribution with some predefined standard deviation. This is repeated until one of the configurations is in collision and the other is collision-free. The collision-free configuration is then added to the roadmap. In contrast, some methods try to sample nodes that are as far from the obstacles as possible (see e.g. [24, 25]).

In the *bridge test sampling* [28], two configurations are sampled. If both are in collision, a third configuration in halfway between them is checked for collision. If it is collision-free, it is added to the roadmap. The distance between the two nodes (i.e. the length of the bridge) can be selected using a normal distribution just like in the Gaussian sampling. Configurations that are sampled this way will likely be located in narrow passages where it is easy to build the bridges. The drawback is that it is possible that other areas of the configuration space will not be covered. One solution is to combine uniform sampling with the bridge test sampling [26]. The majority of the configurations will be generated via bridge test but occasionally a uniform sampler is used.

The bridge test sampling has been shown to have a good performance especially in environments where there are narrow passages [26]. The roadmaps produced with it are usually smaller than the ones produced with a uniform sampling and, in addition, the construction time is often significantly smaller.

It should be noted that there are no single sampling method that is clearly the best in all situations. Different methods have different advantages and weaknesses, which means that they may work very well in some configuration spaces but not in others. One way to try to overcome this problem is to combine different sampling methods together (see e.g. [29, 30, 31]). The planner can, for example, learn during the sampling which methods work well and then start to prioritize those. Another possible solution is to use methods that can divide the configuration space into regions and use local information to guide the sampling in each region (see e.g. [32, 33, 34]). However, usually the more complicated sampling method means that there are also more parameters that must be

set to achieve good results. This can sometimes be quite difficult to do.

In this paper, we use the bridge test sampling in our experiments in addition to uniform sampling to see what effects the sampling methods have on the roadmap size. We use these sampling methods together with a basic PRM planner and with two other planners that can filter unnecessary nodes from the roadmap. Next we describe how these filtering methods work.

3.2 Visibility-based method

The visibility-based method was proposed in [35] and its goal is to produce a roadmap that has a small number of nodes but that still covers the free configuration space well. In a basic PRM planner, some sampling method is used to generate a free configuration that is then added to the roadmap. The visibility-based method differs from this. The configurations are still generated by a sampling method but not all generated configurations are actually added to the roadmap. Some unnecessary configurations are filtered out which in turn reduces the size of the roadmap.

The key concept of the visibility-based method is a *visibility domain* which is a set of configurations that are reachable from some configuration by a local planner. For a configuration q, a visibility region is defined as $V_q = \{ q' \in \mathcal{C}_{\text{free}} \mid \Delta(q, q') \subseteq \mathcal{C}_{\text{free}} \}$, where Δ is a local planner. We can also say that the guard q *sees* all nodes in V_q.

During the roadmap construction, configurations are sampled normally from $\mathcal{C}_{\text{free}}$. The new configuration q will become a *guard* if its visibility domain V_q does not contain any other guards. The configuration q will become a *connection node* if there are at least two guards in V_q that belong to the different components. In both cases, the new configuration q is added to the roadmap. If q is a connection node, also edges which connect it to guards in V_q are added to the roadmap. If there is only one guard in V_q or all guards in it belong to the same component, the configuration q is not added to the roadmap at all. This last case will reduce the size of the roadmap considerably.

The visibility-based method with a straight-line local planner is illustrated in Figure 1 in two-dimensional workspace. Guards are marked with a black circle and other nodes with a white circle. White areas are those configuration space regions that the guards cannot see. In Figure 1(a), there is one guard. In Figure 1(b), a new node has been added. It is a guard because it can not see any other guards. In Figure 1(c), another node has been added. It can see only one guard and therefore it will be removed from the roadmap. In Figure 1(d), a node that can see two guards has been added. It is a connection node and it connects the guards together.

In the basic PRM planner, a set of neighboring configurations are selected for each new configuration q that is added to the roadmap. A local planner is then used to check whether the connection between q and its neighbors is collision-free. In the visibility-based method, there is no need to retrieve neighboring configurations. Instead, a local planner is used to check connections between q and all existing guards.

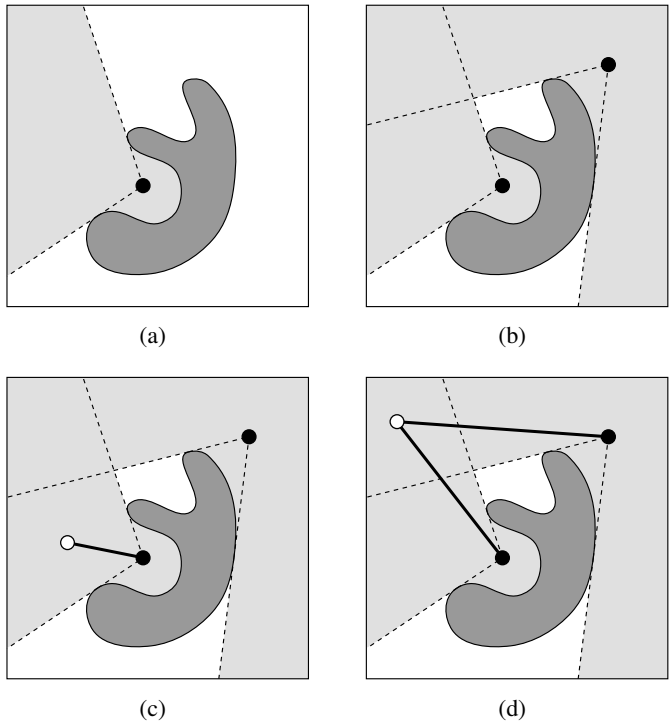


Figure 1: An example of a visibility-based method. A dark gray area represents an obstacle and a light gray area represents area that can be seen by guards. In (a), there is one guard. In (b), another guard has been added. In (c), a node that can see only one guard has been added. It will be removed. In (d), a connection node has connected both guards together.

3.3 Neighborhood-based method

In this section, we propose a neighborhood-based method that is an extension to the basic PRM method and has similarities with the visibility-based method. It tries to combine the strengths of those methods and provide an easy way to reduce the size of roadmaps.

One problem with the visibility-based method is that it completely ignores connection nodes when it attempts to connect a new node to a roadmap with a local planner. It tries to make connections only to guards which can lie very far away from the new node. This is problematic because if nodes are not near each other, it is likely that some obstacles block the path between them. This can easily lead to situations where the visibility-based method must create additional nodes to ensure that the coverage and connectivity of the roadmap is good.

As an example, let us consider the case in Figure 1(d) and especially the upper white area in it. The area represents a region that cannot be seen by any guard. To connect that area to a roadmap with the visibility-based method, a new guard node must

Algorithm 1 A neighborhood method to construct a roadmap $R = (V, E)$.

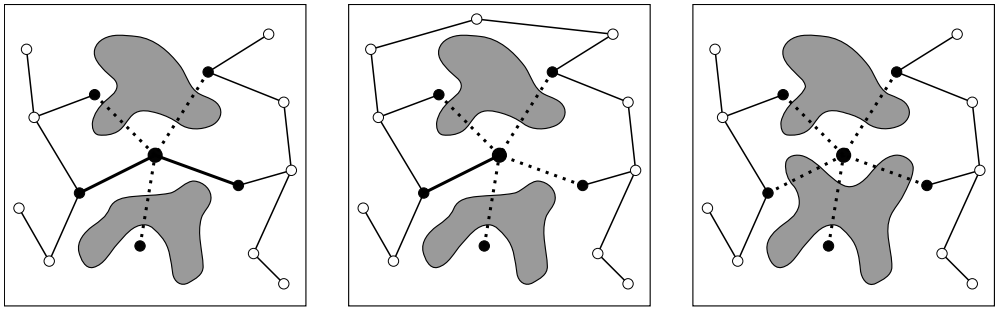
```
1:  V ← ∅
2:  E ← ∅
3:  repeat
4:      q ← randomly chosen configuration from C_free
5:      V ← V ∪ { q }
6:      N_q ← all nearest neighbor configurations of q chosen from V
7:      for all q' in N_q do
8:          if q and q' are not in the same component then
9:              if there is some simple path between q and q' in C_free then
10:                 E ← E ∪ { (q, q') }
11:             end if
12:         end if
13:     end for
14:     if q has only one edge connected to it then
15:         Remove q from the roadmap
16:     end if
17: until there are enough configurations in V
18: return R
```

be sampled inside it. To increase the connectivity, this guard must then be connected to other guards with a new connection node. An important thing to note is that the connection node that already exists in the roadmap can see the whole upper white area. By utilizing this node, the area could be covered without any additional nodes. The neighborhood-based method uses this idea to build small roadmaps with a good coverage and connectivity.

Both the neighborhood-based method and the visibility-based method try to filter out nodes that can be connected to only one component of the roadmap. But unlike the visibility-based method, the neighborhood method does not have guards or connection nodes. All configurations that are in the roadmap are considered equal and the filtering is done based on nearest neighbor configurations. This helps to reduce the size of the roadmap because we can try to connect the new node to all nearby configurations and not just to guards.

The neighborhood method is depicted in Algorithm 1. In lines 4–13, a new configuration q is generated with some sampling method, neighbors are selected for it, and a local planner is used to connect q to its neighbors. In line 8, it is checked whether q and one of its neighbors belong to the same connected component. If they do, the algorithm skips that neighbor. Otherwise, the algorithm attempts to connect them together.

The interesting part happens after the algorithm has gone through all the neighbors. In line 14, it is checked whether the configuration q has only one edge connected to it. If it has, the node is removed from the roadmap along with an edge that is connected to



(a) (b) (c)

Figure 2: An example of different situations in the neighborhood method. A gray area is an obstacle. In (a), a new node in the center of the picture is connected to two other nodes. In (b), the new node is connected to one other node. In (c), the node is not connected to any other nodes.

it. Otherwise, the node can stay in the roadmap. This differentiates the neighborhood method from the basic PRM planner which does not remove any nodes.

There are three different situations that can happen when a new configuration q is sampled and neighbor configurations are selected for it:

1. The configuration q can be connected to neighbors which are at least from two different components.
2. The configuration q can be connected only to neighbors from one component.
3. The configuration q cannot be connected to any neighbor.

In situations 1 and 3, q will be added to the roadmap. In situation 2, q will be rejected. An example in Figure 2 illustrates these situations. In each case, a two-dimensional configuration space is shown as well as a roadmap. Gray areas are obstacles. The circle in the center of the figure is a node that has just been added to the roadmap. In this example, the neighborhood size is five. The neighbors are shown as black circles and other nodes are shown as white circles. Dotted lines show paths from the new node to the neighbors that are not added to the roadmap.

In Figure 2(a), the new node can be connected to two neighbors with a collision-free path. Both nodes belong to different components, so the new node can be accepted. Figure 2(b) is similar but now both nodes belong to the same component. Therefore, the new node can be connected to only one of them. As a result, the new node will have only one edge and it will not be accepted. The new node will be removed from the roadmap. In Figure 2(c), the new node cannot be connected to any neighbor with a collision-free path which means that the node is accepted.

The presented algorithm constructs a roadmap that does not have cycles meaning that it is a set of trees. The cycles would not increase the coverage or connectivity

of the roadmap but they could be useful in some applications. Luckily, it is easy to modify the algorithm in such a way that it allows cycles. The line 8 should be removed and the algorithm should instead count the number of different components the new configuration has been connected. If it were connected to only one component, it should be removed with all its edges. To decide how the additional edges should be added to the roadmap to make cycles, methods investigated in [18] can be used.

It is important to note that the neighborhood method does not require any additional parameters when compared with the basic PRM planner. The only important parameter is the neighborhood size that is also used in the basic planner. Our experiments show, that it is easy to select this parameter in such a way that the resulted roadmap is small. There is no need for any major tweaking of parameters.

4 Experiments

In our experiments, we tested three PRM planners: the basic method, the visibility-based method, and the new neighborhood method. All of them were tested with a uniform sampling and a bridge test sampling. The basic method and the neighborhood method were both tested with two neighborhood sizes. These combinations were tested in four test environments.

In the experiments, we wanted to build roadmaps that were small but that had a good coverage and connectivity. To achieve this, we selected several predefined configurations for each environment. We first added these configurations to the roadmap as normal nodes and then started to construct the rest of the roadmap. New configurations were added to the roadmap one by one. We stopped the construction after all of the predefined nodes belonged to the same component in the roadmap graph, i.e., when the roadmap was large enough to solve all queries between these predefined configurations. At this point, the coverage and connectivity of the roadmap was considered good enough.

After the roadmap was constructed, we measured the time required to build it and the number of nodes. This was repeated 1000 times for each test to minimize the effect of the random nature of PRM planners. The results are shown in Section 5.

We also made an experiment to test how the size of the neighborhood affects the results when using the neighborhood method. We tested this by building a roadmap with different neighborhood sizes and measuring running times and roadmap sizes. All these tests were also executed 1000 times.

4.1 Test environments

In motion planning, workspaces where robots move are almost always two- or three-dimensional. This is especially true for real robots. The difficulty arises from the dimensionality of the configuration space that can be much higher than the dimension of the workspace.

We conducted experiments in four different environments. In three of them the workspace is three-dimensional and the robot is a rigid body that can move and rotate freely in the space. In these environments, the configuration space is therefore six-dimensional.

The fourth test environment is very different from the others. Its workspace is two-dimensional and the robot consists of six parts. Each part is a different sized circle. Each circle can move independently but they may not collide with obstacles or other circles. There are no rotations and each circle has two parameters to determine its location, namely x and y coordinates. Together these circles form one robot which has twelve degrees of freedom. Therefore, the configuration space of this environment is twelve-dimensional.

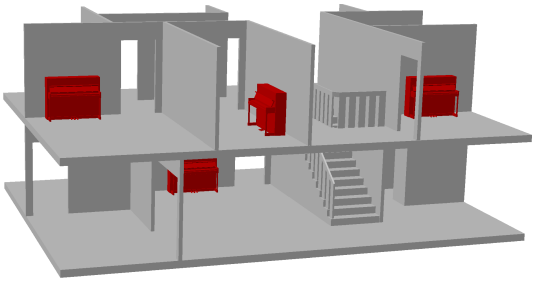
We used the following environments:

House environment A robot is an upright piano which moves in a house where it must avoid collisions with walls, floors, stairs and a roof. The house contains both easy and difficult areas for a robot to move. The goal is to build a roadmap that can be used to guide the robot from room to room. Predetermined configurations that the roadmap had to connect together were selected manually in such a way that they lie on both floors and in different rooms. The environment is shown in Figure 3(a).

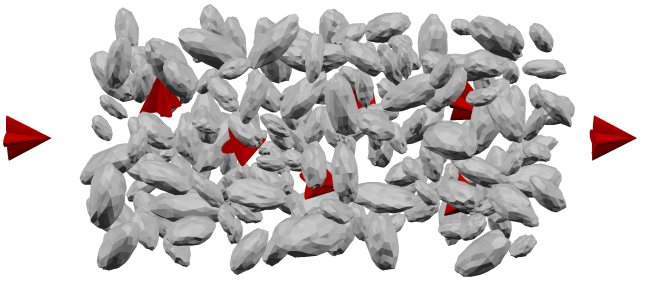
Asteroids environment This is a cluttered environment where small asteroids-shaped obstacles are spread in the space. The robot is a spaceship that must move through the asteroids. This environment contains a lot of narrow passages but not much empty space. There were eight predetermined configurations that had to be connected together with a roadmap. These configurations were generated randomly. The environment is shown in Figure 3(b).

Wall environment This environment has only one obstacle. It is a wall that divides the workspace into two regions. There is a small rectangular hole in the middle of the wall and the robot should find a way through it. The robot itself is a large, hook-shaped object. The wall environment was deliberately made difficult for a uniform sampler. The difficulty comes from the fact that the area around the small hole is practically the only interesting part of the configuration space. Other areas just contain empty space where it is very easy for a local planner to find a path even if only a few configurations have been sampled there. Therefore, a sampler that is able to bias sampling to the area near the hole is likely to perform better. There were two predetermined configurations that had to be connected together. They were on different sides of the wall. The environment is shown in Figure 3(c).

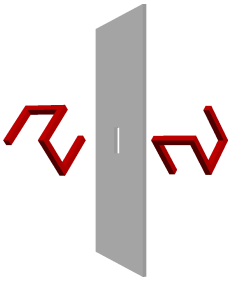
Flat environment In this environment, the workspace is two-dimensional and there are two rectangular and two L-shaped obstacles. The robot consists of six independent parts and in addition to obstacles, the robot must also avoid collision with



(a) A house environment.



(b) An asteroids environment.



(c) A wall environment.

Figure 3: Three test environments which have a three-dimensional workspace. A rigid-body robot is shown in predefined configurations. The house environment is shown without a roof and walls.

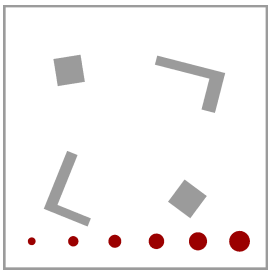


Figure 4: A flat environment that has a two-dimensional workspace. A robot consists of six different sized circles. For clarity, the robot is shown only in one predefined configuration.

itself. Even though the workspace looks simple, this is a difficult environment because the configuration space is very complex. Predetermined configurations that had to be connected together with a roadmap were generated randomly. The environment is shown in Figure 4.

4.2 Implementation details

The experiments were run on a PC with i7-2600 (3.40 GHz) processor and 8 GB memory. All tested methods were implemented in C++ to the same software framework that we developed which helps to ensure that the results are comparable with each other. We used PQP library [36] to perform collision detection.

In [13], it was noted that when using the basic PRM planner the number of selected neighbors k can be set quite high without affecting the running time much. In our experiments, we used the value of 75 for k in the house, asteroids, and wall environments. In the flat environment, we made our tests with the value of 700 for k. In rare tie situations, where there are several nodes at the exactly same distance, we select them all even if it means that the total number of neighbors grows larger than k. We used kd-trees to speed up the nearest neighbor searching. We also ran the tests in such a way that all existing nodes were selected as neighbors.

In this paper, we were not investigating local planners and how they could reduce the size of the roadmap. Therefore, we used a straight-line local planner in all our experiments. It is a popular method and simple to be implemented in different kinds of configuration spaces.

With all methods, we built roadmaps that did not contain cycles. This was done for the performance reasons because adding cycles would increase the running time. We were interested in the number of nodes in the roadmaps and cycles would not have any effect on it. To be able to ignore unnecessary cycles it is important to maintain information about the connected components of the roadmap. This is also needed by the visibility-based method and the neighborhood method. Maintaining can be done efficiently by using a disjoint-set data structure.

5 Results

Tables 1, 2, 3 and 4, show average values of roadmap sizes from all tests as well as average time required to build the roadmap. The standard deviations are also shown. In Tables 2 and 3, the results are not shown for the basic PRM planner when a uniform sampling was used with all nodes in neighborhood. These tests were not executed because constructing such roadmaps would be very slow and it could have taken several hours to build just one roadmap. The reason is the roadmap size that would be very large and taking all of them as neighbors would have a dramatic effect on the speed. The value k is not shown for the visibility method because that method does not use neighbors.

Table 1: Results for the house environment. The results are shown for the new neighborhood method and, for comparison, for the visibility method and for the basic PRM. The average values of 1000 test runs are shown. Standard deviations are shown in parentheses.

Planner	Sampler	k	Nodes	Time (s)
Neighborhood	Bridge	75	218.3 (36.4)	16.4 (7.9)
Neighborhood	Bridge	All	214.2 (32.0)	22.2 (10.7)
Neighborhood	Uniform	75	202.8 (32.5)	14.2 (6.4)
Neighborhood	Uniform	All	197.9 (30.8)	25.0 (13.6)
Visibility	Bridge	-	383.5 (76.5)	23.1 (11.3)
Visibility	Uniform	-	368.8 (73.6)	26.0 (14.2)
Basic PRM	Bridge	75	2202.5 (1072.3)	33.3 (15.8)
Basic PRM	Bridge	All	1424.3 (479.8)	34.4 (16.3)
Basic PRM	Uniform	75	16979.1 (11849.5)	8.7 (5.9)
Basic PRM	Uniform	All	4883.3 (2424.3)	391.7 (784.3)

Table 2: Results for the asteroids environment. The results are shown for the new neighborhood method and, for comparison, for the visibility method and for the basic PRM. The average values of 1000 test runs are shown. Standard deviations are shown in parentheses.

Planner	Sampler	k	Nodes	Time (s)
Neighborhood	Bridge	75	898.6 (172.3)	26.9 (9.1)
Neighborhood	Bridge	All	854.2 (151.3)	76.9 (32.3)
Neighborhood	Uniform	75	816.8 (146.9)	38.7 (9.9)
Neighborhood	Uniform	All	787.4 (126.2)	298.0 (122.9)
Visibility	Bridge	-	1255.4 (253.0)	88.0 (37.1)
Visibility	Uniform	-	1246.2 (208.9)	300.9 (111.9)
Basic PRM	Bridge	75	2553.9 (986.6)	27.0 (9.7)
Basic PRM	Bridge	All	2202.3 (696.2)	87.7 (47.0)
Basic PRM	Uniform	75	18683.5 (9556.8)	15.3 (7.3)
Basic PRM	Uniform	All	-	-

Box plots in Figs. 5, 6, 7 and 8 show a summary of how many nodes different methods generated in each environment. Roadmaps constructed with the basic PRM planner were in all tests much larger than the ones constructed with other methods and therefore, for clarity, results from the basic planner are shown only partially. In the plots, lower and upper quartiles are shown by the left and right sides of the box respectively. A median is shown by the line inside the box. The whiskers show the minimum and maximum values that are within 1.5 times the interquartile range from either side of the box. The outliers are shown as small dots.

By looking at the box plots, it is easy to notice that the neighborhood method clearly produces smaller roadmaps than the visibility method in all tested environments. This was true for both the uniform sampling and the bridge test sampling. Furthermore, it can be seen from the tables that the neighborhood and visibility methods can both produce much smaller roadmaps than the basic planner.

Table 3: Results for the wall environment. The results are shown for the new neighborhood method and, for comparison, for the visibility method and for the basic PRM. The average values of 1000 test runs are shown. Standard deviations are shown in parentheses.

Planner	Sampler	k	Nodes	Time (s)
Neighborhood	Bridge	75	58.7 (10.0)	15.0 (8.9)
Neighborhood	Bridge	All	58.7 (10.2)	15.1 (8.9)
Neighborhood	Uniform	75	58.5 (9.2)	477.9 (358.4)
Neighborhood	Uniform	All	58.6 (9.5)	480.2 (356.9)
Visibility	Bridge	-	76.3 (17.0)	16.5 (8.8)
Visibility	Uniform	-	159.4 (35.1)	605.8 (475.6)
Basic PRM	Bridge	75	181.3 (78.3)	46.6 (20.6)
Basic PRM	Bridge	All	182.2 (77.4)	46.8 (20.5)
Basic PRM	Uniform	75	538062.6 (287780.2)	426.6 (323.5)
Basic PRM	Uniform	All	-	-

Table 4: Results for the flat environment. The results are shown for the new neighborhood method and, for comparison, for the visibility method and for the basic PRM. The average values of 1000 test runs are shown. Standard deviations are shown in parentheses.

Planner	Sampler	k	Nodes	Time (s)
Neighborhood	Bridge	700	2135.4 (803.6)	53.1 (38.5)
Neighborhood	Bridge	All	2035.4 (695.2)	126.9 (153.2)
Neighborhood	Uniform	700	1653.1 (588.3)	37.0 (26.7)
Neighborhood	Uniform	All	1577.3 (526.3)	66.1 (74.8)
Visibility	Bridge	-	2898.8 (1212.2)	83.7 (70.9)
Visibility	Uniform	-	2310.0 (917.8)	52.3 (41.7)
Basic PRM	Bridge	700	5379.5 (5482.1)	81.8 (95.4)
Basic PRM	Bridge	All	4025.9 (2766.1)	358.9 (1015.0)
Basic PRM	Uniform	700	5848.6 (6371.0)	44.0 (68.0)
Basic PRM	Uniform	All	3906.8 (2896.7)	327.4 (1002.0)

When using the basic PRM planner, the bridge test seems to produce smaller roadmaps than the uniform sampling. This is especially clear in the wall environment, where the roadmap had more than half a million nodes on average when using the uniform sampling. By using the bridge test, the roadmap size was only 181 nodes on average. However, the bridge test sampling did not perform much better than the uniform sampling in the flat environment when the basic planner was used.

In contrast to the basic planner, the uniform sampling seems to produce at least as small roadmaps as the bridge test sampler when using the neighborhood method. This is true even in the wall environment where the bridge sampling was supposed to perform much better. With the visibility method, the uniform sampling produces smaller roadmaps in all environments except in the wall environment.

Based on the results, the neighborhood method seems to work quickly. In all environments, it works faster than the visibility method while still producing smaller

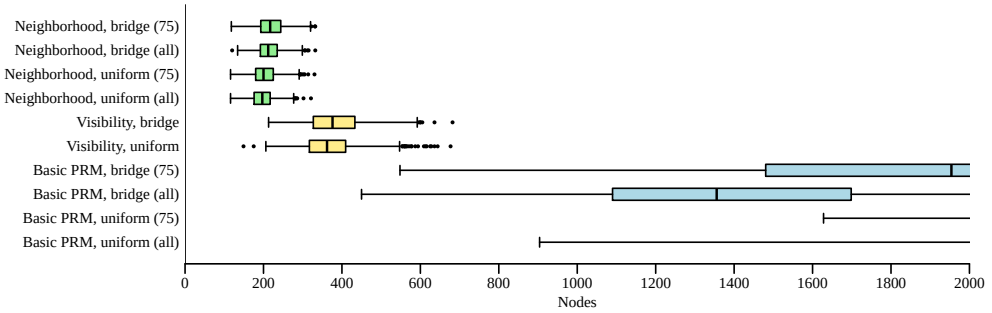


Figure 5: Box plots for the house environment. Note that results for the basic PRM planner are shown only partially.

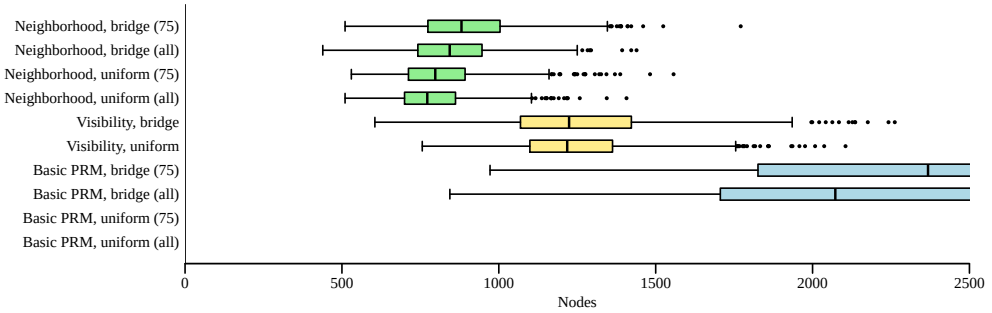


Figure 6: Box plots for the asteroids environment. Note that results for the basic PRM planner are shown only partially.

roadmaps. The speed of the basic planner depends greatly on the used sampling method and the environment. In some cases it is very fast but in some other cases it can take hours to build just one roadmap.

If the uniform sampling and the bridge test sampling are compared, there are differences in the construction times but it is difficult to say which one is better since the fastest sampling method differs from one environment to the other. For example, in the flat environment, the uniform sampling seems to be the fastest with all planners, but in the wall environment, the bridge test sampling is clearly the fastest.

It should be noted that a small roadmap does not mean that building it would take a shorter time than building a larger roadmap even when the same planning and sampling methods are used. Instead, the neighborhood size can affect greatly the time. This can be seen, for example, from Table 1 by looking at average times of the basic planner with the uniform sampling. When the size of the neighborhood was 75, the constructed roadmap had 16979 nodes and time needed to build it was 8.7 seconds. When all nodes were taken as neighbors, the roadmap size shrank to 4883 nodes but the required time increased to 391.7 seconds.

Figure 9 shows an example of how the neighborhood size affects the roadmap size

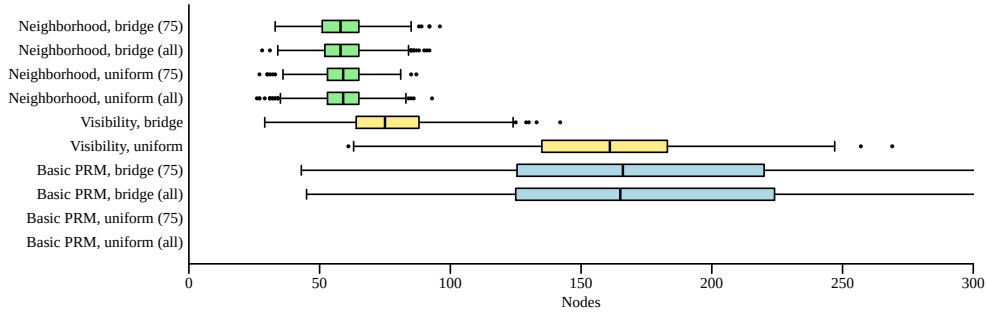


Figure 7: Box plots for the wall environment. Note that results for the basic PRM planner are shown only partially.

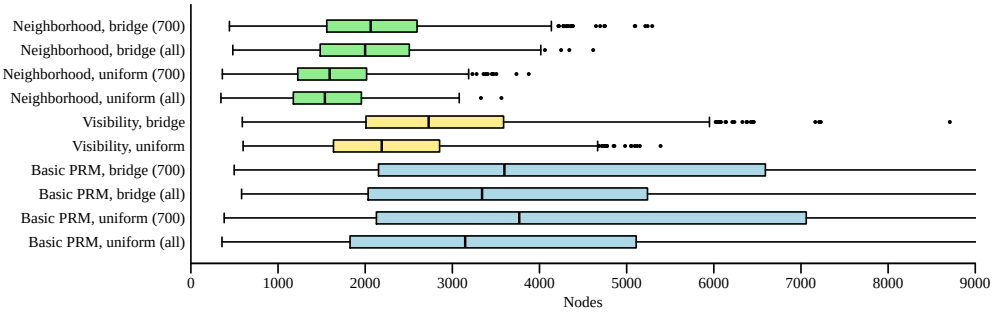


Figure 8: Box plots for the flat environment. Note that results for the basic PRM planner are shown only partially.

and running time when using the neighborhood method with the uniform sampling. In the example, a roadmap was built with different neighborhood sizes and with each size the roadmap was built 1000 times. The line chart shows the mean of both the roadmap size and the running time. We used the house environment in the example but the behavior is also similar in other environments.

From Figure 9 it is easy to see that when the number of neighbors is small, the running time is high and also the roadmap size is very large. However, the running time decreases quickly when the number of neighbors is increased. In the house environment, the algorithm runs fastest when the neighborhood size is about 16. If the size is increased beyond that, the running time will grow. The roadmap size decreases rapidly at the start and then continues with a slower decrease. To get a small roadmap, the neighborhood size should be as large as possible, but in practice it is good to limit the size to reduce the construction time of the roadmap.

The results suggest that the neighborhood method is suitable also when the goal is to decrease the running time of the planning instead of reducing the number of nodes. This can be done by decreasing the size of the neighborhood sufficiently as can be seen in Figure 9.

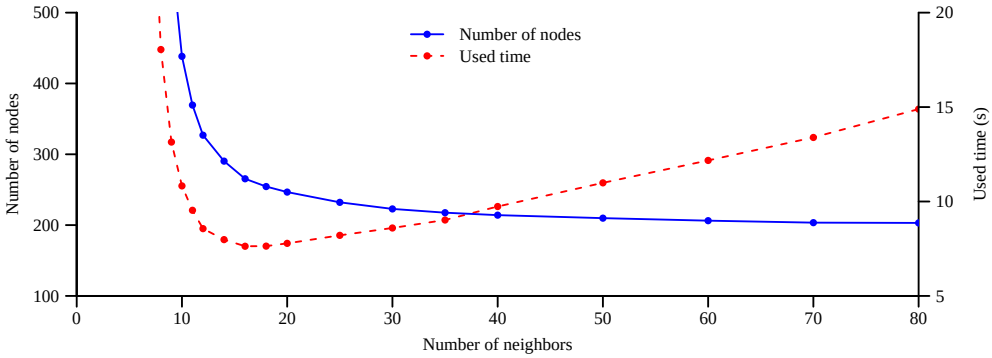


Figure 9: A line chart that shows how the neighborhood size affects the number of nodes and the running time when using a neighborhood method.

One very good aspect is that the neighborhood method works very well even with the basic uniform sampling which can be implemented very easily to different kinds of configuration spaces. Another good feature is that the neighborhood method does not require any additional parameters when compared with the basic planner. The neighborhood size is the only important parameter and it is easy to pick a good number. This is very useful especially when the neighborhood method is used with the uniform sampling that does not require any parameters either.

These features make the neighborhood method a prominent choice for roadmap planners. It can quickly produce a small roadmap that captures the connectivity of the configuration space well. It is also easy to expand the roadmaps by just adding new nodes and edges so if the cycles or denser set of roadmap nodes are required it is possible to add them after the small base for the roadmap has been built.

6 Conclusions

In this paper, we compared experimentally different methods that can be used to reduce the size of the roadmap while still maintaining a good coverage and connectivity. We compared the basic PRM planner with the visibility-based method and with the neighborhood-based method proposed. All these methods were tested with the uniform sampling and with the bridge test sampling.

The results show that the neighborhood method is very good at producing small roadmaps quickly. The visibility method is slower and it produces larger roadmaps. The basic PRM planner produces the largest roadmaps and can be extremely slow in some cases.

The results show that the neighborhood method works quite well with the simple uniform sampling. This is very useful, since neither the uniform sampling nor the neighborhood method require any additional parameters. The only important parameter

is the size of the neighborhood which is also needed in the basic PRM planner. Luckily, it seems that there is no need to do any major fine-tuning of that parameter to get good results.

In future, it might be interesting to investigate how additional cycles and path smoothing techniques could be incorporated to these small roadmaps. It would also be interesting to make experiments with different local planners and see whether those could decrease the size of the roadmaps even more.

Acknowledgements

The research of the first author was supported by the Tampere Doctoral Programme in Information Science and Engineering (TISE).

References

[1] J.-C. Latombe. *Robot Motion Planning*. Kluwer, Boston, MA, 1991.

[2] J. Smed and H. Hakonen. Path finding. In *Algorithms and Networking for Computer Games*, pages 97–113. John Wiley & Sons, 2006.

[3] I. Al-Bluwi, T. Siméon, and J. Cortés. Motion planning algorithms for molecular simulations: A survey. *Computer Science Review*, 6(4):125–143, 2012.

[4] X. Pan, C. S. Han, and K. H. Law. Using motion planning to determine the existence of an accessible route in a CAD environment. *Assistive Technology*, 22(1):32–45, 2010.

[5] 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, Cambridge, MA, 2005.

[6] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006.

[7] M. H. Overmars. A random approach to motion planning. Technical Report RUU-CS-92-93, Department of Computer Science, Utrecht University, the Netherlands, 1992.

[8] L. 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.

[9] 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 & Automation*, 12(4):566–580, 1996.

[10] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning. Technical Report 98-11, Computer Science Deptartment, Iowa State University, 1998.

[11] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308. A K Peters, Wellesley, MA, 2001.

[12] R. Geraerts and M. H. Overmars. Sampling and node adding in probabilistic roadmap planners. *Robotics and Autonomous Systems*, 54(2):165–173, 2006.

[13] R. Geraerts and M. H. Overmars. Reachability-based analysis for probabilistic roadmap planners. *Robotics and Autonomous Systems*, 55(11):824–836, 2007.

[14] A. Yershova and S. M. LaValle. Improving motion-planning algorithms by efficient nearest-neighbor searching. *IEEE Transactions on Robotics*, 23(1):151–157, 2007.

[15] L. Jaillet and T. Siméon. A PRM-based motion planner for dynamically changing environments. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2004.

[16] D. Nieuwenhuisen and M. H. Overmars. Useful cycles in probabilistic roadmap graphs. In *IEEE International Conference on Robotics and Automation*, 2004.

[17] A. Sud, R. Gayle, E. Andersen, S. Guy, M. Lin, and D. Manocha. Real-time navigation of independent agents using adaptive roadmaps. In *ACM Symposium on Virtual Reality Software and Technology*, 2007.

[18] M. T. Rantanen and M. Juhola. Using probabilistic roadmaps in changing environments. *Computer Animation and Virtual Worlds*, 25(1):17–31, 2014.

[19] R. Geraerts and M. H. Overmars. Creating high-quality paths for motion planning. *The International Journal of Robotics Research*, 26(8):845–863, 2007.

[20] R. Guernane and N. Achour. Generating optimized paths for motion planning. *Robotics and Autonomous Systems*, 59(10):789–800, 2011.

[21] I. Karamouzas and M. H. Overmars. Adding variation to path planning. *Computer Animation and Virtual Worlds*, 19(3–4):283–293, 2008.

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

[23] 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.

[24] 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.

[25] C. Holleman and L. E. Kavraki. A framework for using the workspace medial axis in PRM planners. In *IEEE International Conference on Robotics and Automation*, pages 1408–1413, 2000.

[26] Z. Sun, D. Hsu, T. Jiang, H. Kurniawati, and J. Reif. Narrow passage sampling for probabilistic roadmap planning. *IEEE Transactions on Robotics*, 21(6):1105–1115, 2005.

[27] H.-Y. Yeh, S. Thomas, D. Eppstein, and N. M. Amato. UOBPRM: A uniformly distributed obstacle-based PRM. In *IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2655–2662, 2012.

[28] 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*, volume 3, pages 4420–4426, 2003.

[29] 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 3874–3880, 2005.

[30] H. Kurniawati and D. Hsu. Workspace-based connectivity oracle: An adaptive sampling strategy for PRM planning. *Algorithmic Foundation of Robotics VII*, pages 35–51, 2008.

[31] S. Thomas, M. Morales, X. Tang, and N. M. Amato. Biasing samplers to improve motion planning performance. In *IEEE International Conference on Robotics and Automation*, pages 1625–1630, 2007.

[32] M. Morales, L. Tapia, R. Pearce, S. Rodriguez, and N. M. Amato. A machine learning approach for feature-sensitive motion planning. *Algorithmic Foundations of Robotics VI*, pages 361–376, 2005.

[33] S. Rodriguez, S. Thomas, R. Pearce, and N. M. Amato. RESAMPL: A region-sensitive adaptive motion planner. *Algorithmic Foundation of Robotics VII*, pages 285–300, 2008.

[34] M. T. Rantanen. A connectivity-based method for enhancing sampling in probabilistic roadmap planners. *Journal of Intelligent and Robotic Systems*, 64(2):161–178, 2011.

[35] T. Siméon, J.-P. Laumond, and C. Nissoux. Visibility-based probabilistic roadmaps for motion planning. *Advanced Robotics*, 14(6):477–493, 2000.

[36] E. Larsen, S. Gottschalk, M. C. Lin, and D. Manocha. Fast distance queries with rectangular swept sphere volumes. In *IEEE International Conference on Robotics and Automation*, pages 3719–3726, 2000.