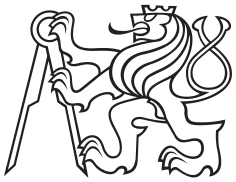
Bachelor Project



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Sampling-based motion planning for 3D rigid objects

Dominik Filyó

Supervisor: Ing. Vojtěch Vonásek, Ph.D.
Field of study: Cybernetics and Robotics
Subfield: Systems and Control
May 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Filyo** Jméno: **Dominik** Osobní číslo: **438666**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

Studijní obor: **Systémy a řízení**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Plánování pohybu pro 3D objekty

Název bakalářské práce anglicky:

Sampling-based motion planning for 3D rigid objects

Pokyny pro vypracování:

1. Seznamte se s úlohou plánování pohybu [3], konkrétně s algoritmy typu Probabilistic Roadmaps (PRM) [5] a Rapidly exploring Random Trees (RRT) [4]. Jednu z těchto metod implementujte pro plánování 3D rigidních objektů v 3D prostředí.
2. Implementujte další plánovací algoritmus založený na RRT nebo PRM (např. [1,2]).
3. Navrhněte alternativní algoritmus pro expanzi stromu konfigurací v algoritmu RRT.
4. Implementované metody experimentálně porovnejte na vybraných problémech z benchmarku [6] a na 3D modelech proteinů. Porovnejte vliv použitých metrik na rychlost a kvalitu plánování.

Seznam doporučené literatury:

[1] Park, Chonhyon, Jia Pan, and Dinesh Manocha. "Parallel Motion Planning Using Poisson-Disk Sampling." IEEE Transactions on Robotics 33, no. 2 (2017): 359-371.
[2] Zhang, Liangjun, and Dinesh Manocha. "An efficient retraction-based RRT planner." In IEEE International Conference on Robotics and Automation, 2008
[3] LaValle, Steven M - Planning algorithms - Cambridge university press, 2006.
[4] LaValle, Steven M., and James J. Kuffner Jr. - Rapidly-exploring random trees: Progress and prospects (2000).
[5] Kavraki, Lydia E., et al. - Probabilistic roadmaps for path planning in high-dimensional configuration spaces -
[6] https://parasol.tamu.edu/dsmft/benchmarks/mp/

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Vojtěch Vonásek, Ph.D., Multirobotické systémy FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **31.01.2018** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Vojtěch Vonásek, Ph.D.	prof. Ing. Michael Šebek, DrSc.	prof. Ing. Pavel Ripka, CSc.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. | Datum převzetí zadání

Podpis studenta

© ČVUT v Praze, Design: ČVUT v Praze, VIC

Acknowledgements

Firstly, I would like to express my sincere gratitude to my supervisor Ing. Vojtěch Vonásek, Ph.D. for his valuable advice and guidance on writing this thesis.

Also, I would like to thank my whole family for their continuous encouragement and supporting me throughout studying the university. Especially my parents and sister, their belief in me has made everything possible.

Special thanks go to my cousin for his help with creating amazing illustrations used in the thesis.

Last, I greatly appreciate access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum provided under the programme "Projects of Large Research, Development, and Innovations Infrastructures" (CESNET LM2015042).

Declaration

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instructions for observing the ethical principles in the preparation of university theses.

Prague, 25. May 2018

Abstract

Sampling-based motion planning is a modern approach for solving a task of the navigation that arises in many scientific disciplines. This work is dedicated to an analysis and application of all relevant motion planning algorithms. Furthermore, we propose a new method for the RRT expansion called RRT-sphere. We show experimental results and comparison of five implemented RRT-based algorithms.

Keywords: Motion, path, planning, RRT, sampling, robotics

Supervisor: Ing. Vojtěch Vonásek, Ph.D.
CTU in Prague,
Faculty of Electrical Engineering,
Department of Cybernetics,
Charles' Square 13,
121 35 Prague 2,
KN:E 130

Abstrakt

Plánování pohybu založené na náhodném vzorkování prostoru je moderní přístup k řešení úlohy navigace, která se vyskytuje v mnoha vědeckých disciplínách. Tato práce je věnovaná analýze a použití všech významných algoritmů pro plánování pohybu. Mimo to představujeme novou metodu pro expanzi stromu v algoritmu RRT, která se jmenuje RRT-sphere. Předloženy jsou experimentální výsledky a srovnání pěti implementovaných algoritmů založených na RRT.

Klíčová slova: Pohyb, cesta, plánování, RRT, vzorkování, robotika

Překlad názvu: Plánování pohybu pro 3D objekty

Contents

Figures

Chapter 1

Introduction

1.1 Motivation

The task of motion planning is to compute collision-free paths for robots. The paths must satisfy constraints for a movement of the robots. In general, environments for the motion planning can be very complex with an arbitrary number of dimensions. The most common environment in the motion planning is *Euclidean space* $\mathbb{R}^3$ since it is the world we are living in. Furthermore, the environment can contain obstacles, the robot cannot interact with.

Since first mobile robots were constructed, it was required to plan their trajectory and control their movement. Recently, with growing research in the area of mobile and intelligent robotics, many new and improved planning algorithms have been proposed. But it is not only the mobile robotics, the motion planning is used for. The motion planning has been successfully applied in a wide spectrum of applications. Namely, robotic arms in the industry [1, 2], CAD systems [3], video game artificial intelligence [4, 5], self-driving cars [6, 7], robotic surgery [8, 9], biochemistry (study of proteins, drug design) [10, 11] and more. An example of usage in biochemistry is outlined in Figure 1.1.

1.2 Thesis Goals

The thesis has following goals designated in the assignment:

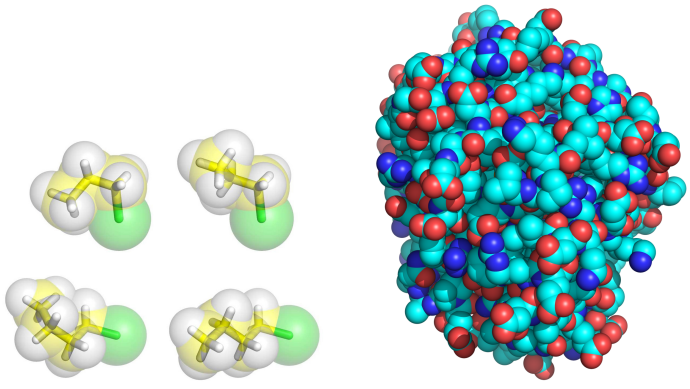
- **To familiarize with the motion planning.** In the thesis, we give an overview of the motion planning algorithms. We further study and describe a group of *sampling-based algorithms*. Specifically *Rapidly exploring Random Trees* (RRT) [12, 13], *Probabilistic Roadmaps* (PRM) [14]

and their derivatives.

- **To implement a planning algorithm based on RRT or PRM.** RRT and four other RRT-based algorithms are implemented in our application coded in C++ programming language.
- **To propose an alternative algorithm for the tree expand in RRT.** We proposed an improvement of RRT. This improvement is based on the adaptive sampling region expanding around the goal.
- **To experimentally examine implemented algorithms.** We test the performance of all methods in terms of the run-time, number of explored nodes and length of the path (according to a metric). The tests are performed in maps from the benchmark [15] and protein models. A metric influence on the planning speed and quality is evaluated as well.

1.3 Thesis Outline

The thesis continues with Chapter 2, where we familiarize a reader with necessary terms for further study of the motion planning. The motion planning itself is discussed in Chapter 3. All significant algorithms, whether newer or older, are described for an understanding of the reader in this chapter. Chapter 4 is devoted to the most relevant algorithm for the thesis — RRT. We study the original RRT and four enhanced RRT-based algorithms in Chapter 4. In Chapter 5, our work is presented. It consists of a motion planning software, introduction of an improved version of RRT and experimenting with various RRT-based algorithms.



(a) : Ligands **(b) :** A protein molecule

Figure 1.1: In biochemistry, the motion planning is used to determine possible exit pathways for a small molecule (ligand) from a large molecule (protein). A simple approach is to compute the pathway only for a single atom, or better, considering the whole ligand. The existence of pathways can help the chemists to decide if a given can react with the protein, which is useful e.g. in drug design. (Illustrations by courtesy of Vojtěch Vonásek)

Chapter 2

Problem Analysis

In this chapter, we describe essential concepts in the motion planning. It gives a reader theoretical background of the subject. All terms in this section are defined for the need of this bachelor thesis and meanings may differ from other literature. The most significant literature about the motion planning is Steven M. LaValle's book Planning Algorithms [16]. This book is the primary source of information for this thesis.

2.1 Representation of the Environment

For the purpose of the motion planning, it is important to define an environment, the planning will be executed at. A point on a plane (Euclidean space $\mathbb{R}^2$), of course, needs fewer variables to describe it's movement than a 12-DOF[1] robotic arm. The concept of determining an object position in the world is the main idea behind the space called *state space*. Further, terms *configuration space* and *metric* are described.

State space The main and the most general space in the field of robotics and cybernetics (including control theory). The state space captures all possible situations which may occur. A state of the robot given by translations, rotations, translational and angular velocities of every independently moving part in all directions (in terms of basis) assigns a point to the N-dimensional state space. For the rigid object in the 3D world with R *holonomic constraints*[2] N is equal to $2 \cdot (6 - R)$.

[1] DOF – Degree Of Freedom.

[2] Holonomic constraint – A geometrical constraint that reduces the number of DOF of the object.

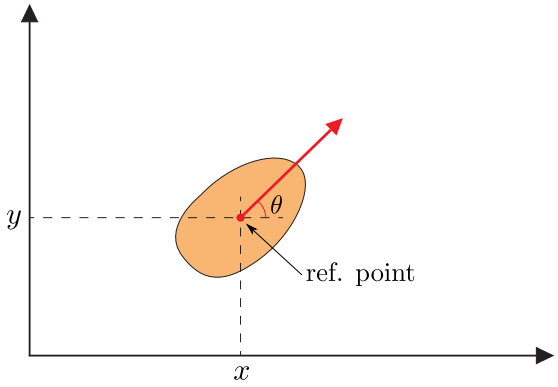


Figure 2.1: The configuration of a 2D rigid object in the form $q = (x, y, \theta)$

Let x denote the current state, $\dot{x}$ derivative of the state, $\mathcal{X}$ the state space, u the vector of control inputs and U the set of possible inputs. The input u is a variable that controls the robot and define the robot's movement over time. This input is determined by a planner or robot controller. A dynamical model of the system (in this case a robot) can be encoded in a function f called *state transition equation*:

$$\dot{x} = f(x, u). \tag{2.1}$$

This model is also called the *forward motion* model. A new state is computed by integrating f over a given time interval Δt. If no analytical solution exists, methods of numerical integration are used. For example *Euler method* or *Runge–Kutta method* [17].

Configuration space Because for some groups of motion planning problems (discussed later in Section 2.3) are some states unimportant, the next space used in the motion planning is the configuration space. The configuration space is a subspace of the state space that contains only translations and rotations. Eventually, it can contain other *generalized coordinates* which clearly identify a position of the robot in the world (distance of two robots). The set of generalized coordinates is called a *configuration* and is denoted q (see Figure 2.1). A state x can be also defined as $x = (q, \dot{q})$.

Let $\mathcal{C}$ denote the configuration space. For a rigid object without moving parts in N-dimensional space, $\mathcal{C}$ can be written as a *Cartesian product*

$$\mathcal{C} = \mathbb{R}^N \times SO(N), \tag{2.2}$$

where $\mathbb{R}^N$ is N-dimensional Euclidean space and $SO(N)$ is N-dimensional *Rotation group* including all rotations about the origin. This applies if no holonomic constrains are present. If so, the number of dimensions is reduced by the number of holonomic constrains.

The configuration space $\mathcal{C}$ can be divided into two subspaces depending on the position of obstacles. The part where the robot can move without

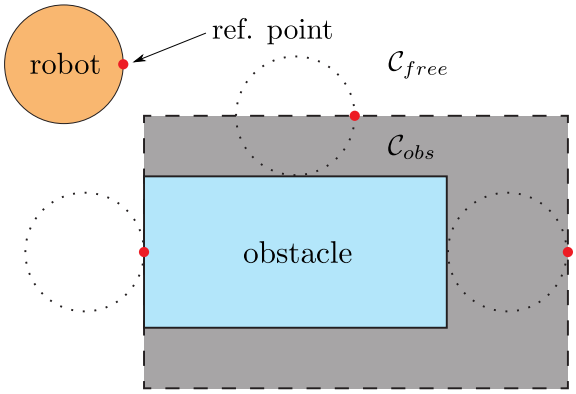


Figure 2.2: The configuration space around an obstacle

collisions is denoted $\mathcal{C}_{free} \subseteq \mathcal{C}$. Thus, the other subspace is a part of $\mathcal{C}$ with obstacles $\mathcal{C}_{obs} = \mathcal{C} \setminus \mathcal{C}_{free}$. The obstacle space $\mathcal{C}_{obs}$ is not just a projection of the obstacles to $\mathcal{C}$ because the robot occupies more space around its reference point as the robot is not just a point (see Figure 2.2).

Just as $\mathcal{C}_{free}$ and $\mathcal{C}_{obs}$ are defined, free state space $\mathcal{X}_{free}$ and obstacles state space $\mathcal{X}_{obs}$ are also defined. The obstacle space $\mathcal{X}_{obs}$ can be constructed by adding geometrical constraints (obstacles) and velocity constraints to the state space $\mathcal{X}$.

Metric For a given set S, metric is a function

$$\rho : S \times S \to \mathbb{R}. \tag{2.3}$$

For all $x, y, z \in S$, the metric has following properties:

$$\rho(x, y) \geq 0 \tag{2.4a}$$

$$\rho(x, y) = 0 \Leftrightarrow x = y \tag{2.4b}$$

$$\rho(x, y) = \rho(y, x) \tag{2.4c}$$

$$\rho(x, z) \leq \rho(x, y) + \rho(y, z) \tag{2.4d}$$

If metric is defined on the set S, this set is called *metric space*. Conditions 2.4 express abstraction of the concept of distance. Examples of metrics are *Euclidean metric*, *taxicab (Manhattan) metric* or *Hamming distance*.

First two mentioned metrics are the part of generalized metric called L_p *metric*, which is defined as

$$L_p(x, y) = \left(\sum_{i=1}^{n} |x_i - y_i|^p \right)^{1/p}. \tag{2.5}$$

For $p = 1$, it yields taxicab metric. For $p = 2$, we get well-known Euclidean metric.

2.2 Collision Detection

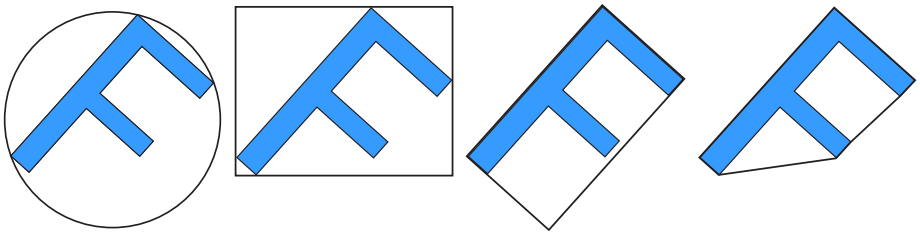
The task of the robot is to avoid obstacles on the way the goal. To determine whether the robot is at a collision configuration is an assignment for the *collision detector* which has to be a part of the motion planner. This problem can be seen as determine whether the robot lies in the free configuration space $\mathcal{C}_{free}$ and therefore, is in the collision-free configuration. The collision detector is a function

$$\phi(\mathcal{C}, q) = \begin{cases} \text{TRUE} & \text{if } q \in \mathcal{C}_{obs} \\ \text{FALSE} & \text{otherwise} \end{cases}, \tag{2.6}$$

where q is the configuration of the robot.

Although it might seem that it is simple to check whether a point (representation of the robot in $\mathcal{C}$) lies in $\mathcal{C}_{obs}$, exact appearance of $\mathcal{C}_{obs}$ is not known for most of the planning problems since it is a complex *non-convex* set. Nevertheless, mapping of obstacles into the configuration space is still possible. One of the approaches is via the fast Fourier transform [18].

Many of collision detection algorithms have been proposed recently [19]. A widespread method used in many collision detectors is called *bounding volume hierarchy* [20]. Suppose two non-convex objects, the robot and obstacle. The objects are decomposed into trees T_r and T_o, where a root of each tree is the whole object and each node represents a bounded subset of the object. Possible bounding regions are shown in Figure 2.3. An intersection is first tested between the root nodes and if no exist, the algorithm reports no collision. If the bounding overlap, the algorithm recursively tests their children until it finds the intersection or reaches leaves of the tree. In that case, each polygon pair of the robot and obstacle is tested.



(a) : Sphere **(b) :** Axis-aligned bounding box **(c) :** Oriented bounding box **(d) :** Convex hull

Figure 2.3: Possible bounding regions for the bounding volume hierarchy method in the collision detection system

2.3 Motion Planning

Generally, the motion planning is defined for a set that contains:

- *world* $\mathcal{W}$, either $\mathcal{W} = \mathbb{R}^2$ or $\mathcal{W} = \mathbb{R}^3$,
- obstacle region $\mathcal{O} \subset \mathcal{W}$,
- semi-algebraic rigid robot $\mathcal{A} \subset \mathcal{W}$ or a collection of m joined segments $\mathcal{A}_1, ..., \mathcal{A}_m \subset \mathcal{W}$,
- state space $\mathcal{X}$,
- set of possible inputs U,
- *initial state* $x_I \in \mathcal{X}$,
- *goal state* x_G or set of goal states $\mathcal{X}_{goal} \subset \mathcal{X}$.

The motion planning executes a probing of the state space. A goal is to find a valid non-colliding path for the robot $\mathcal{A}$ in the world $\mathcal{W}$ from the initial position and velocity encoded by x_1 to the goal region $\mathcal{X}_{goal}$. The robot must obey geometric constraints given by obstacles and *differential (non-holonomic) constrains* given by the robot's design. It was shown that the problem of motion planning is PSPACE-hard [21]. We can classify motion planning problems into three groups by constraints.

Holonomic planning The most basic motion planning is called holonomic planning. The term refers to holonomic constraints. Holonomic constraints are constraints which reduce the number of DOF of the robot. Holonomic constraints appear in the form $h_i(q) = 0$ and thus i generalized coordinates are blocked for the control. See Figure 2.4 for an example.

Non-holonomic planning This planning addresses problems with non-integrable velocity constraints [22]. These constraints are common in systems that involve rotating parts like wheeled-robots (car) or aerial systems (aircraft, UAV[3]). Non-holonomic constraints are the constraints in the form $g_i(q, \dot{q}) = 0$. It prevents to control all DOF separately since a change of the one coordinate might change others coordinates as well. This is a difference with the holonomic planning, where all coordinates can be controlled separately. See Figure 2.5 for an example. Both holonomic and non-holonomic planners perform a search in the configuration space.

Kinodynamic planning The title refers to kinematic and dynamic constraints. Besides position and velocity constraints, acceleration constraints are present in this planning yielding equations $k_i(q, \dot{q}, \ddot{q}) = 0$ [23]. Since acceleration constraints exist, space to be searched is the full state space $\mathcal{X}$.

[3] UAV – Unmanned Aerial Vehicle.

Further in the thesis, we address all problems as probing of the state space regardless of a type even though we defined a state as $x = (q, \dot{q})$. For holonomic and non-holonomic planning, one can assume $x = q$ and hence $\mathcal{X} = \mathcal{C}$.

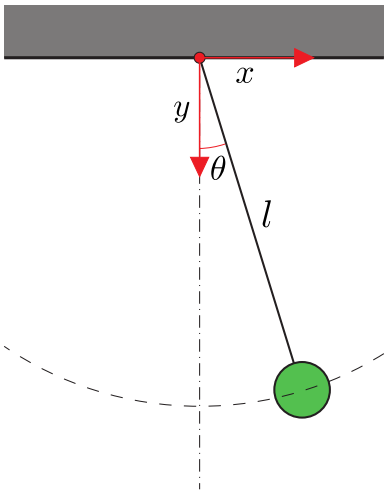


Figure 2.4: A holonomic system, planar pendulum, with constraint in the form $x^2 + y^2 = l^2$ that reduces the number of DOF. This system can be described by one coordinate θ and hence $q = \theta$ and $x = (\theta, \dot{\theta})$.

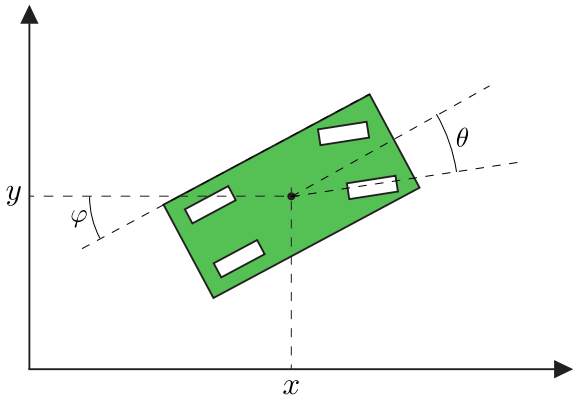


Figure 2.5: A car representing the non-holonomic system. A configuration of the system is $q = (x, y, \theta, \varphi)$.

Chapter 3

State of the Art

This chapter gives a brief overview of motion planning methods. There are three main approaches to motion planning algorithms different in the representation of the state space. The oldest group of algorithms is *combinatorial planning* [24, 25]. The next group are *artificial potential field* methods [26, 27]. At present, intensive research is ongoing in the field of *sampling-based planning* [28].

Since the title of this thesis is Sampling-based motion planning for 3D rigid objects, we study sampling-based algorithms in more detail and Chapter 4 is devoted to the most important algorithm for the work.

3.1 Combinatorial Planning

In the task of probing the state space, the continuous space needs to be discretized. This can be done by several techniques. Combinatorial planning directly represents the free state space as a *roadmap*. The roadmap is an undirected graph $G(V, E)$ in $\mathcal{X}_{free}$, where vertices from the set V are states from $\mathcal{X}$ and edges from the set E are possible transitions between the states. After the roadmap is done, a graph search algorithm (A*,D*) is applied to find the shortest path.

The combinatorial planning requires *polygonal* obstacles $\mathcal{O}$ in order to create the roadmap representation of $\mathcal{W}$. Because the motion planning operates in the state space $\mathcal{X}$ and not in the world $\mathcal{W}$, the obstacles need to be first transformed into $\mathcal{X}$ to make $\mathcal{X}_{obs}$. It is a very demanding task and exact transformation can be done only in simple problems, for example a 3D world with a rigid robot restricted only to the translation movement. In this cases, $\mathcal{X}_{obs}$ can be constructed as $\mathcal{X}_{obs} = \mathcal{O} \oplus -\mathcal{A}(0)$, where $\oplus$ is a special

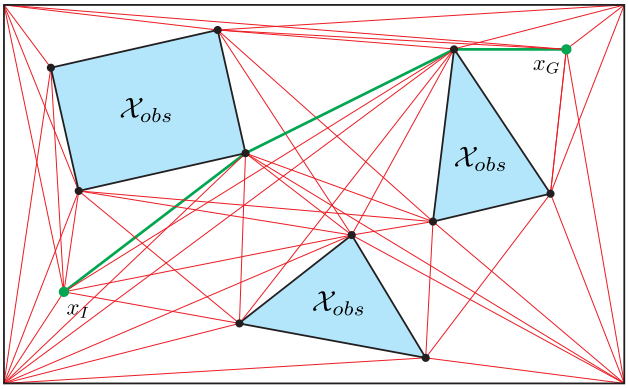


Figure 3.1: The visibility graph

convolution called a *Minkowski sum* defined as

$$A \oplus B = \{\vec{a} + \vec{b} \,|\, \vec{a} \in A, \vec{b} \in B\}. \tag{3.1}$$

Let $\mathcal{A}(q) \subset \mathcal{W}$ define all points of the geometry of the robot $\mathcal{A}$ located in the configuration q.

The combinatorial algorithms are capable of solving easier planning problems very elegant and intuitive. They are unfortunately limited by the low number of DOF and only holonomic constraints. On the other hand, their advantage is that they are *complete*[1]. For example, the 2D world with a robot moving only translationally is a great occasion to employ a combinatorial planner.

Visibility graph The roadmap is constructed by connecting mutually visible vertices of obstacle polygons [29]. First, the initial and goal positions are connected to the all visible vertices of obstacles and world. Each vertex is then connected with others vertices if they are in sight. After this procedure, the roadmap looks like it is shown in Figure 3.1.

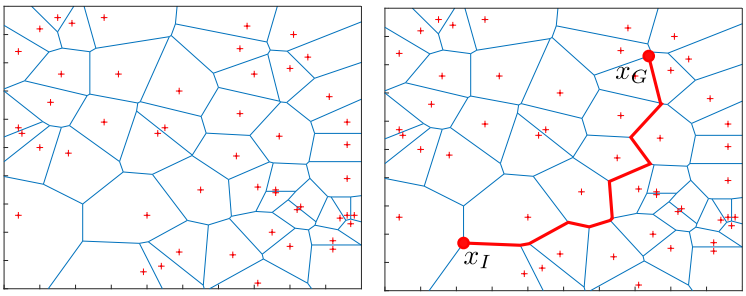
The visibility graph method finds an optimal path every time, but the path is close to obstacles. A naïve algorithm for construction the visibility graph has a *time complexity* $O(n^3)$ and the best algorithm is $O(n^2)$ [30].

Voronoi diagram The Voronoi diagram is a planar structure that partition a plane into regions based on the distance of given points. The Voronoi region V_k of a point x_k is defined as

$$V_k = \{p \in \mathcal{X}_{free} \,|\, \forall i \neq k, \rho(p, x_k) \leq \rho(p, x_i)\}, \tag{3.2}$$

where x_i, x_k are the input points and ρ is a metric. The Voronoi diagram is formed from vertices that have the same distance from two or more points

[1] Complete algorithm – Algorithm either finds a solution or reports that a solution does not exist.



(a) : Voronoi regions **(b) :** The found path

Figure 3.2: The Voronoi diagram with points representing obstacles

(see Figure 3.2). Edges are connections of the closest vertices. Edges of the diagram can be generated between two points, point and edge and between two edges (*segment Voronoi diagram*) which is useful for the motion planning since the obstacle boundary is represented by edges.

Voronoi diagrams have been well studied in the mobile robotics [31]. Construction of the Voronoi diagram is very fast (time complexity $O(n \log(n))$ in simple maps [32]), but it is not suitable for open spaces because the robot is attracted to the middle and sub-optimal paths are created.

Cell decomposition The idea of this algorithm is decomposing $\mathcal{X}_{free}$ into cells with the specified shape. In each cell, which does not include the obstacle, is placed a vertex of the roadmap called *adjacency graph*. Edges of the adjacency graph are lines connecting the adjacent vertices. There are two versions of the cell decomposition algorithm: *exact decomposition* [33] and *approximate decomposition* [34].

In the exact decomposition, space is decomposed into sets of variously large trapezoids or triangles. The cells may be constructed by cutting the space vertically from each polygon vertex as shown in Figure 3.3. Then, the

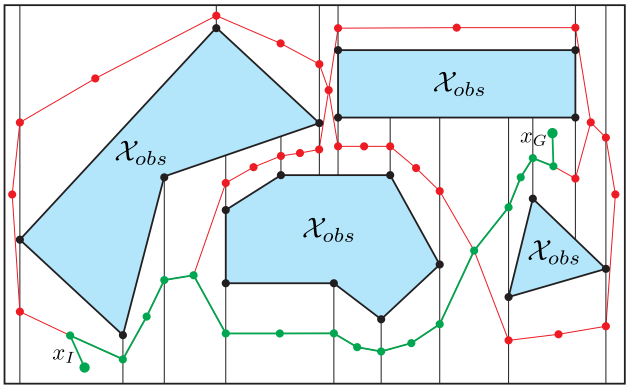


Figure 3.3: The exact cell decomposition

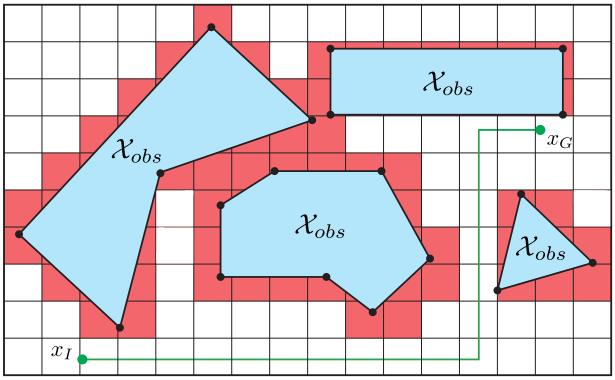


Figure 3.4: The approximate cell decomposition

vertex is placed in each segment border and inside each segment, for example to the center of the segment. The best-known algorithm for the exact cell decomposition has complexity $O(n \log(n))$.

The approximate decomposition is different in a mechanism of constructing the cells. All cells are the same simple shape, most often squares, creating a grid. Cells that intersect with the obstacles are removed. A solution existence depends on density of the grid. The algorithm starts with small density and then refines the density until it finds a solution (which may not exist). Thus, the algorithm is incomplete but easy to implement. See Figure 3.4 for the approximate cell decomposition in a simple environment.

3.2 Potential Fields

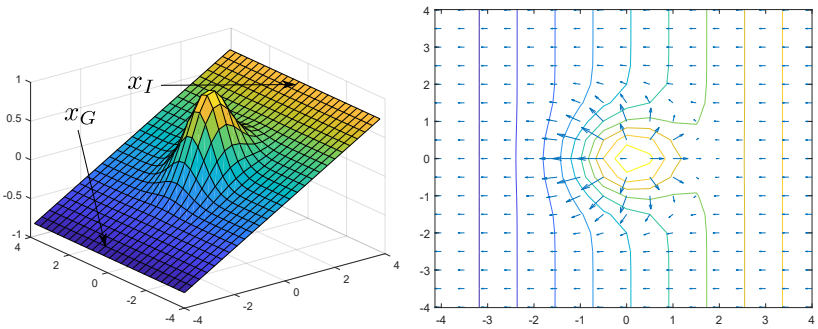
So far, achievable states of the robot were represented by a graph. Another approach for the holonomic or non-holonomic planning is to define an artificial potential field $\mathbf{U}(q)$ similar to the electric potential [26, 27]. The robot acts like a charged particle in the electric field and is attracted by a force

$$\mathbf{F} = -\vec{\nabla}\mathbf{U}(q). \tag{3.3}$$

$\mathbf{U}(q)$ is modelled to has a global minimum at the goal state and a local maximum at the initial state. The goal generates the attractive potential that pulls the robot toward the goal. On the contrary, obstacles generate the repulsive potential that pushes the robot away from them (see Figure 3.5).

The disadvantage of the potential field methods is that the robot can get stuck in a local minimum that is not in at the goal. This can be resolved by using an optimization algorithm such as *simulated annealing*. Potential fields are commonly used for *local motion planning*[2].

[2]Local motion planning – Controls a movement of the robot between two states.



(a) : 3D visualization **(b) :** A vector field

Figure 3.5: A potential field of the space with one obstacle in the middle

Potential fields which represent the environment can be obtained at the start of the planning process or iteratively. The sequential approach computes the potential field on a grid. Before the algorithm starts, the continuous space must be discretized over a mesh with defined proportions.

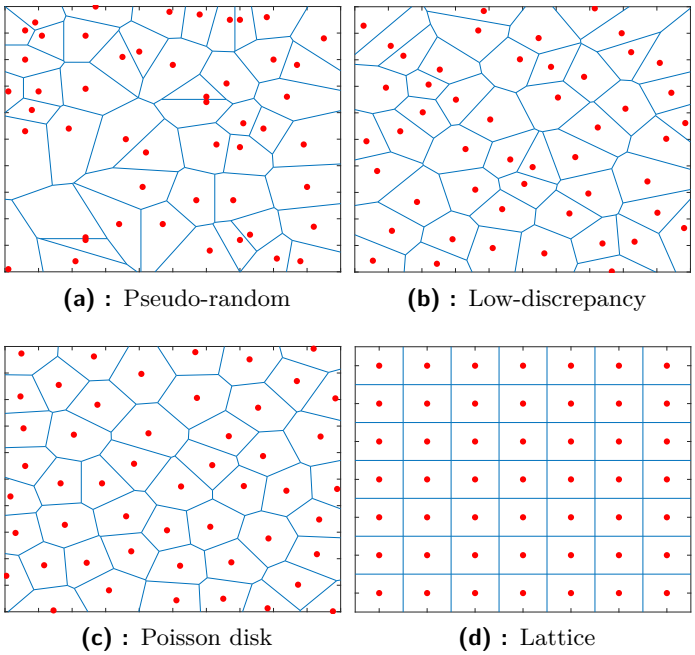
3.3 Sampling-based Planning

A sampling-based approach to the motion planning is based on representing the free space as a roadmap of sampled states. *Sampling scheme* generates random samples in the space. Whenever there exists a collision-free path between two samples, they can be connected by a line. Computation of the line shape is a task for the local planner. Examples of *uniform* sampling schemes are shown in Figure 3.6. Besides the most used uniform sampling, there are others techniques for the space sampling [35, 36].

The main difference with the probabilistic planning is that there is no need to explicit construct $\mathcal{X}_{obs}$ before the algorithm starts. Sampling-based algorithms probe $\mathcal{X}$ without knowing anything about the world. They are independent of the geometrical representation of the world. It is possible due to the collision detection algorithm (see Section 2.2). This makes the collision detector the most critical part of the sampling-based planning.

The major advantage of sampling-based algorithms is the ability to solve all kinds of planning problems with non-holonomic or kinodynamic constraints [37] with a low computational cost. Moreover, they can handle high DOF spaces and a wide variety of geometrical models. There are also modifications capable of planning in dynamic environments [38].

It has been proposed two basic sampling-based algorithms and many improved ones. The first sampling-based algorithm introduced in 1996 by Jean-



(a) : Pseudo-random **(b) :** Low-discrepancy

(c) : Poisson disk **(d) :** Lattice

Figure 3.6: Uniform sampling schemes with the Voronoi diagram representing the scatter of random points

Claude Latombe and his collaborators[3] is called *Probabilistic Roadmaps* [14]. The other algorithm was introduced in 1998 by Steven M. LaValle. This algorithm is called *Rapidly exploring Random Trees* [12].

3.3.1 Probabilistic Roadmaps

Probabilistic Roadmaps (PRM) [14], as the title suggests, is a method for the motion planning similar to the combinatorial planning methods in terms of the algorithm proceeding. The algorithm is separated into two phases: *learning phase* and *query phase*. In the learning phase, a roadmap is constructed by generating random samples (states) in $\mathcal{X}$ (Algorithm 1). The learning phase is followed by the query phase, where the task is to find a collision-free path between initial and goal states by a graph search algorithm.

The roadmap is represented by an undirected graph $G = (V, E)$, where V stands for vertices and E for edges. The vertices represent states and the edges represent transitions between two states given by a local planner. The graph is first composed of several *connected components*. As the algorithm continues, the number of connected components reduces until there is only one connected component left. The construction of PRM is shown in Figure 3.7 and examples of different roadmaps in Figure 3.8.

[3] Lydia E. Kavraki, Petr Švestka, Mark H. Overmars.

Algorithm 1 Basic PRM - learning phase

Require: Initial state x_I, number of iterations K, maximal distance of neighbours d
Ensure: Undirected graph G

```
G.add_vertex(x_I)
for k = 0 to K do
    x_rand ← RANDOM_STATE()
    G.add_vertex(x_rand)
    N ← NEIGHBOURS(x_rand, G, d)
    for each x ∈ N do
        u ← SELECT_INPUT(x_rand, x)
        if x is feasible then
            G.add_edge(x_rand, x, u)
return G
```

Given a state space $\mathcal{X}$, initial state x_I, distance d and empty graph G, the algorithm proceeds as follows:

1. **Learning phase** Add the initial state to the graph G.
2. Generate a random state $x_{rand} \in \mathcal{X}_{free}$ and add it to the graph.
3. Select all neighbour states of x_{rand} within the distance d in terms of the metric ρ.
4. For all the neighbours select a control input $u \in U$ that ensures a collision-free transition and also minimizes a distance from x_{rand} to the neighbour.
5. If such a control input is found, add the neighbour to the graph.
6. If the roadmap is completed, go to step 7. Go to step 2 otherwise.
7. **Query phase** Employ a graph search algorithm to find a path in G from x_I to x_G.

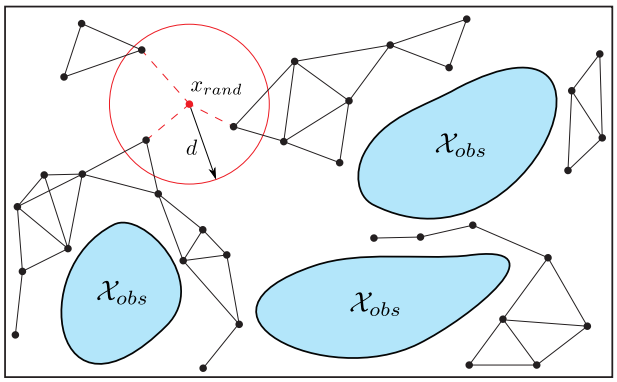
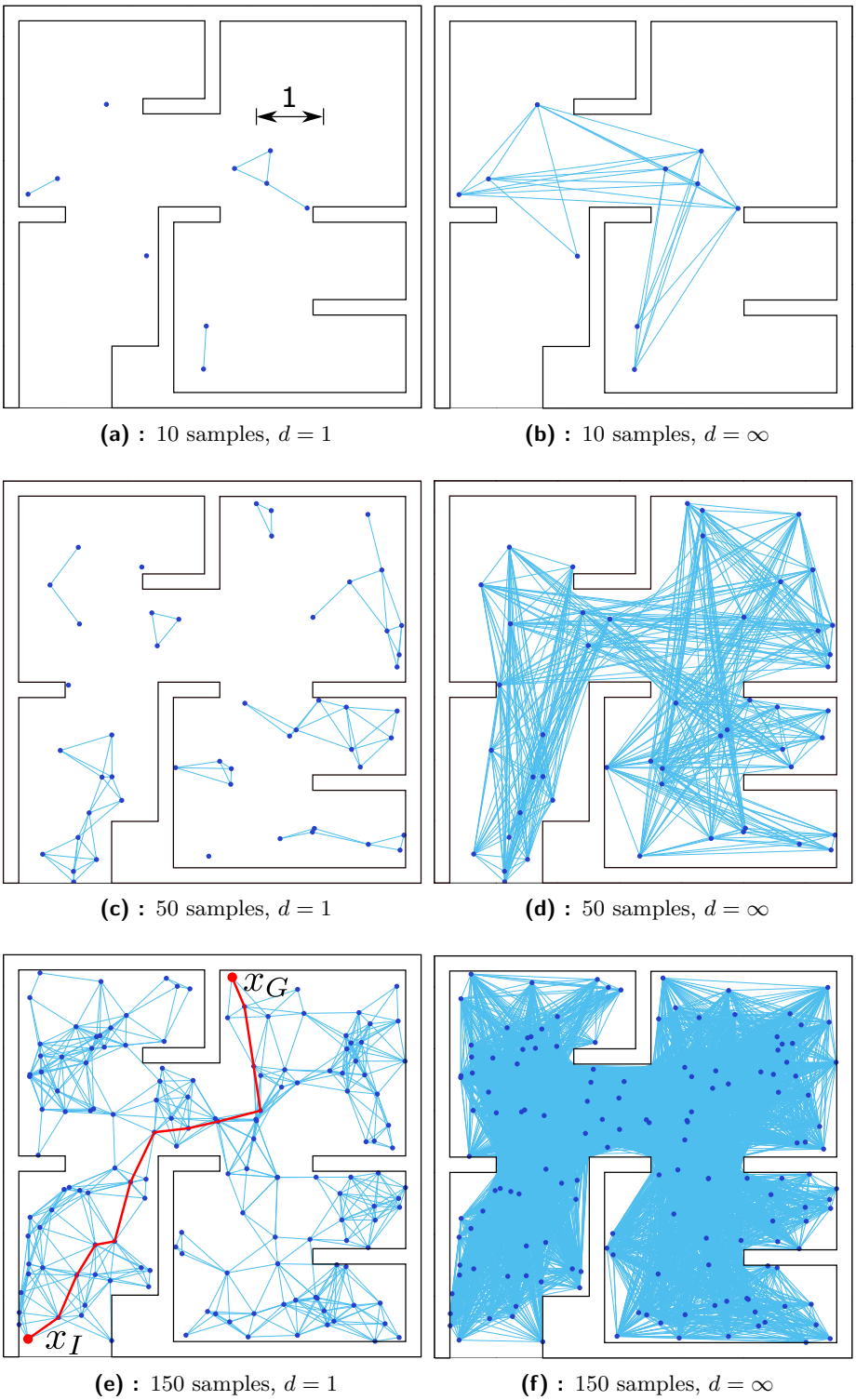


Figure 3.7: The construction of PRM, where x_{rand} is a randomly generated point from $\mathcal{X}_{free}$ and d is the maximal distance to other points needed to make a connection.



(a) : 10 samples, $d = 1$ **(b) :** 10 samples, $d = \infty$

(c) : 50 samples, $d = 1$ **(d) :** 50 samples, $d = \infty$

(e) : 150 samples, $d = 1$ **(f) :** 150 samples, $d = \infty$

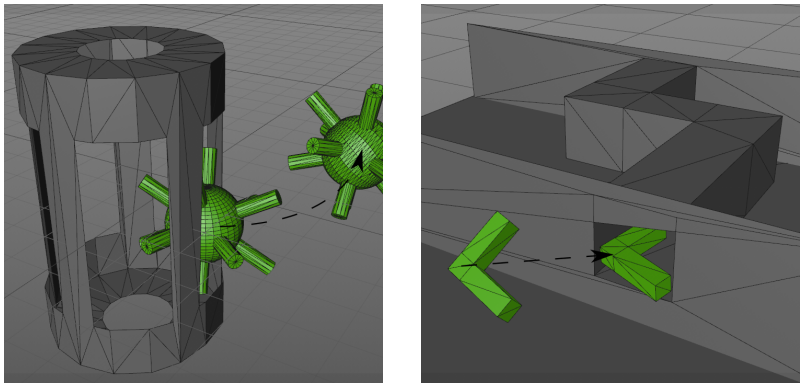
Figure 3.8: PRM with different number of samples and parameter d. With the increasing number of samples, the uniform coverage of the space is achieved. (e) shows a computed path in the query phase.

Chapter 4

Rapidly Exploring Random Trees

Rapidly exploring Random Trees (RRT) [12, 13] is an effective sampling-based planning algorithm designed for handling various motion planning problems. The collision-free samples are saved in a tree $\mathcal{T}$. In contrast with a common graph in case of PRM, the tree is an undirected graph without cycles. The absence of the cycles implies that every vertex in the tree has only one parent and thus no graph search algorithm is needed. It is possible to trace back nodes from the goal node to the initial node in order to find a path.

Although RRT is capable of probing high dimensional spaces with all kinds of constraints, it may be inappropriate for some planning problems. For example, the worlds with obstacles close together decrease the performance and efficiency of the algorithm. The parts with little space for passing the configuration are called *narrow passages* (see Figure 4.1). The basic RRT samples the space by a pseudo-random generator yielding in a lack of uniformity of the distribution. The analysis of weaknesses of the random sampling is shown in [39]. Improved versions of RRT, proposed to solve different types of problems, are also discussed in this chapter.



(a) : Hedgehog in the cage **(b) :** A tunnel

Figure 4.1: Two examples of narrow passages in the 3D world

4.1 Basic RRT

The first version of RRT was proposed in 1998. Introduction of RRT was a breakthrough in the motion planning due to the algorithm rapidity and ability to easy enhance the method. Unlike the combinatorial algorithms and PRM, RRT was designed to handle non-holonomic and kinodynamic planning. The non-holonomic version is described in Algorithm 2.

Given a state space $\mathcal{X}$, initial state x_I, time interval Δt and empty tree $\mathcal{T}$, the RRT algorithm proceeds as follows:

1. Add the initial state to the tree $\mathcal{T}$.
2. Generate a random state $x_{rand} \in \mathcal{X}$.
3. Select the nearest neighbour of x_{rand} laying in $\mathcal{T}$ in terms of metric ρ. Denote it x_{near}.
4. Select a control input $u \in U$ that ensures a collision-free transition and also minimizes a distance from x_{rand} to x_{near}.
5. Apply the input u to x_{near} and compute a new state x_{new}. This state is determined by integration of state equation $\dot{x} = f(x, u)$ over a time interval Δt.
$$x_{new} = x_{near} + \int_0^{\Delta t} f(x, u)\, \mathrm{d}t \tag{4.1}$$
6. If x_{new} is reachable by a collision-free path, add it to the tree. Ignore it otherwise.
7. Go to step 2.

There are two key components in RRT and others sampling-based algorithms, and that is the NEAREST_NEIGHBOUR function and the collision detector. The task of the first one is to find the nearest vertex of another vertex. This is dependent on a choice of the metric. It can be performed by a *KD-tree* approach [40]. See [41] for a library ANN by David M. Mount,

Algorithm 2 Basic RRT

Require: Initial state x_I, number of iterations K, time interval Δt
Ensure: RRT tree $\mathcal{T}$

```
1: T.add_vertex(x_I)
2: for k = 0 to K do
3:     x_rand ← RANDOM_STATE()
4:     x_near ← NEAREST_NEIGHBOUR(x_rand, T)
5:     u ← SELECT_INPUT(x_rand, x_near)
6:     x_new ← NEW_STATE(x_near, u, Δt)
7:     if x_new is feasible then
8:         T.add_vertex(x_new)
9:         T.add_edge(x_new, x_near, u)
10: return T
```

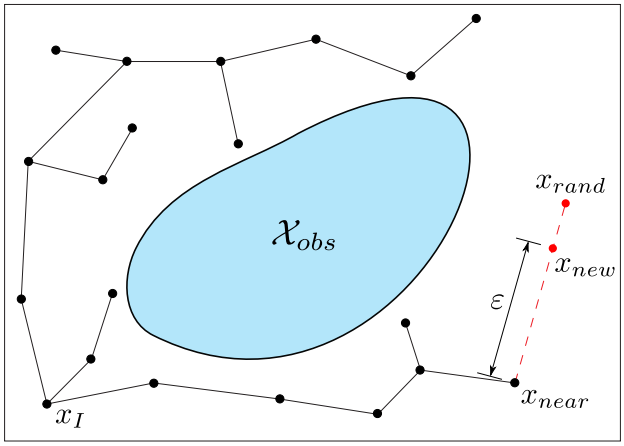


Figure 4.2: The construction of holonomic RRT. ε denotes the maximal distance between two nodes.

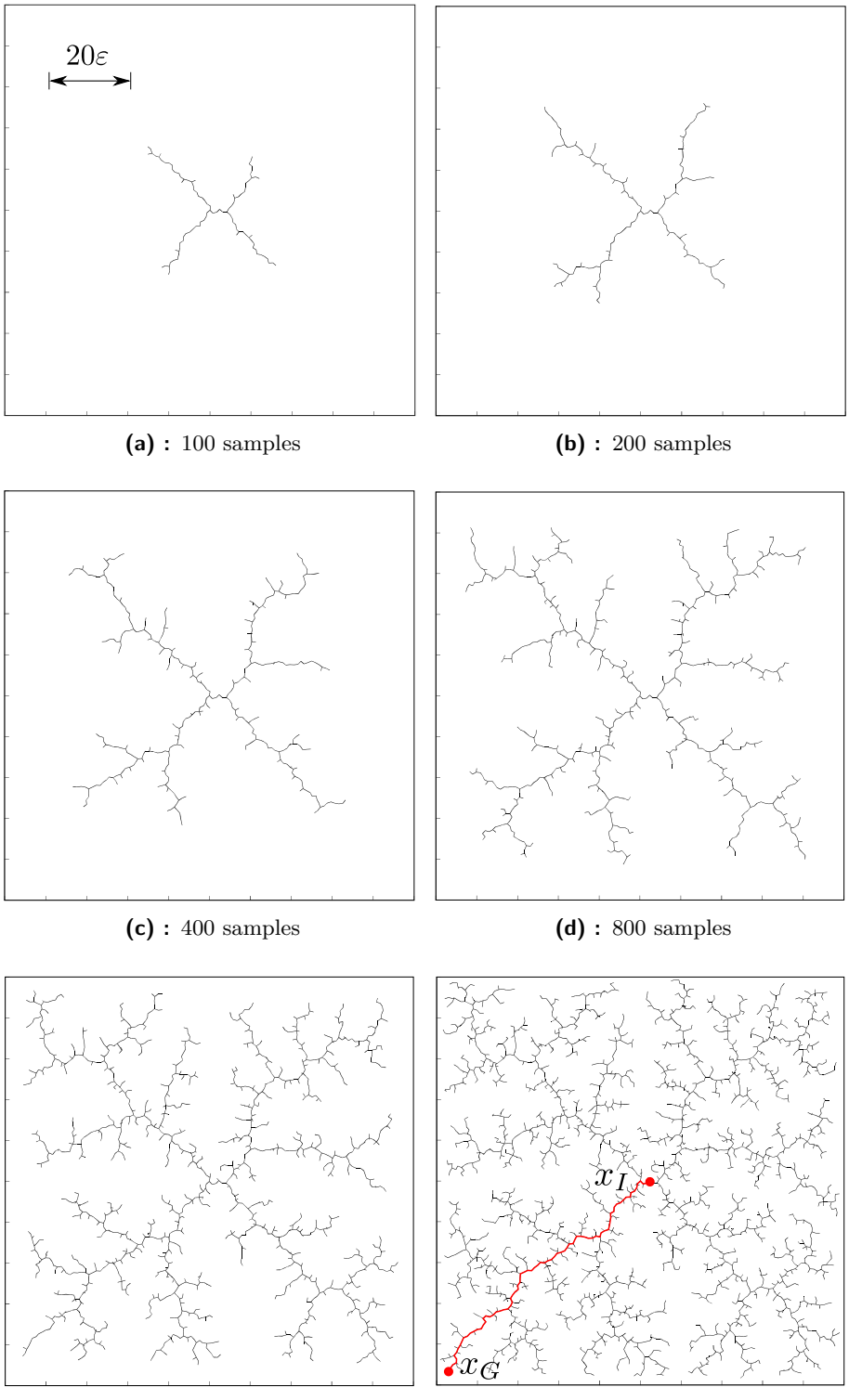
Sunil Arya and [42] for the improved implementation called MPNN by Anna Yershova and Steven M. LaValle. The advantage of MPNN is a possibility to add nodes to the KD-tree while running. An approximate approach of determining the nearest neighbours is possible as well [43].

The collision detection system was briefly introduced in Section 2.2. The collision detector determines whether a new node is in a collision-free configuration. Two libraries frequently used in the motion planning are RAPID [44] and OZCollide [45] for their great usability. Both libraries are based on the method bounding volume hierarchy. An input to the libraries is a *polygon soup*, which is a group of unlinked polygons (most often triangles).

The tree tends to explore yet unexplored portions, so it leads to the uniform coverage of the space, as it can be seen in Figure 4.3. After crossing a threshold, let's say 1600 samples in Figure 4.3, when the tree is spread out all over the space, the expansion stops and the tree is getting denser as the algorithm is running. This fact can be viewed as a ration between *exploration* and *exploitation* [46]. Planning algorithms may be classified according to this ratio. Generally, the state space is infinitely large, but for real-world applications, we can assume the bounded state space by reachable positions and velocities. See figure 4.2 for an illustration of constructing RRT.

In the following picture, it is demonstrated a tree growing from the centre for a holonomic non-steerable system. Since a state in the holonomic planning does not contain any dependence on the previous state, the state transition equation is reduced to $\dot{x} = f(u)$. For a non-steerable system, the input represents a bounded *geodesic*[1] ($|u| < \varepsilon$), where ε denotes the maximal distance between two nodes. Moreover, it is shown a computed path between two selected nodes in Figure 4.3 (f).

[1]Geodesic – The shortest route between two points. For Euclidean geometry it is a straight line.



(a) : 100 samples **(b) :** 200 samples

(c) : 400 samples **(d) :** 800 samples

(e) : 1600 samples **(f) :** 3200 samples

Figure 4.3: The RRT tree growing from the centre for a small holonomic square as the robot. Parameters are: state space width = 100, $\varepsilon = 1$.

4.2 RRT-based Planners

Since the basic RRT planner does not produce optimal results for some planning problems, it has been developed hundreds of enhanced RRT-based planners to employ for specific problems. See [28] for an overview.

The first parameter of RRT that can be altered is the sampling strategy. In the basic RRT, the space is sampled by the uniform pseudo-random number generator. Other possible variants are: the intelligent sampling, biasing sampling, sampling around obstacles, sampling in narrow regions, hybrid sampling and more. Another important parameter is a metric. In general, a metric can represent an arbitrary parameter describing a cost of the transition from one state to another. Different metrics are used among the RRT-based algorithms.

Some algorithms modify the expansion step for better behaviour in highly constrained environments and narrow passages. For the purpose of finding an optimal (shortest, fastest) path, a whole group of optimal RRT-based algorithms exists. Moreover, there are improvements which apply a post-processing to a given path. They can smooth the path or make it shorter.

But first, let's explore the group of algorithms which utilize more that one tree that grows from the beginning.

4.2.1 Multiple Trees

The idea is maintaining multiple trees instead of one tree rooted in the initial state in RRT. There is a basic variant with two trees called *RRT-Connect* [47]. Some of the multiple trees methods even utilize more than two trees, namely three [48] or even more trees as required by the environment [49]. A common characteristic of these algorithms is *heuristic* which connects the trees whether it is possible. In experiments, it was shown that for some applications multiple trees are more efficient than a single tree [47].

RRT-Connect was the first multi-tree algorithm for the holonomic planning proposed just two years after the introduction of RRT in 2000 by Steven M. LaValle and James J. Kuffner. The algorithm was designed for problems that do not contain differential constraints. This approach is sometimes called a *balanced bidirectional search*. It is bidirectional because it involves two trees growing from two directions and balanced in order to keep their rapid exploring property.

Generally, the tree balancing is a part of the multi-tree algorithms, but not all algorithms involve it (RRT-Connect). In the case of the algorithm where the balancing is not present, it can almost lead to the basic RRT in

Algorithm 3 Balanced bidirectional RRT

Require: Initial state x_I, goal state x_G, number of iterations K, distance ε
Ensure: The path from x_I to x_G if is found, failure otherwise

```
1: T_a.add_vertex(x_I)
2: T_b.add_vertex(x_G)
3: for k = 0 to K do
4:     x_rand ← RANDOM_STATE()
5:     x_near ← NEAREST_NEIGHBOUR(x_rand, T_a)
6:     x_new ← NEW_STATE(x_near, ε)
7:     if x_new is feasible from x_near then
8:         T_a.add_vertex(x_new)
9:         T_a.add_edge(x_new, x_near)
10:        x_n ← NEAREST_NEIGHBOUR(x_new, T_b)
11:        if x_new is feasible from x_n then
12:            return path
13:    if |T_b| > |T_a| then
14:        SWAP(T_a, T_b)
15: return failure
```

some environments, because one tree is strongly expanded while the other tree contains few nodes. The balancing can be realized by comparison the number of nodes in the trees or the total length.

Balanced bidirectional RRT starts by adding the initial state x_I to the tree $\mathcal{T}_a$. The goal state x_G is added to the second tree $\mathcal{T}_b$. Then it picks a random state x_{rand} from $\mathcal{X}$, selects the nearest neighbour of x_{rand} in the tree $\mathcal{T}_a$ and determines a new state x_{new} according to the maximal distance between two nodes ε. Then, it is established if x_{new} is reachable from x_{near} by a collision-free path. If so, x_{new} and u are recorded to $\mathcal{T}_a$. The nearest neighbour of x_{new} in the second tree $\mathcal{T}_b$ is selected (x_n). If x_n and x_{new} can

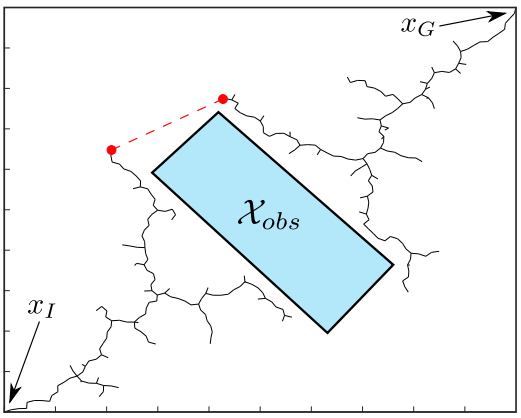


Figure 4.4: Balanced bidirectional RRT rooted in the initial node and goal node. The red line represents a possible connection of the trees.

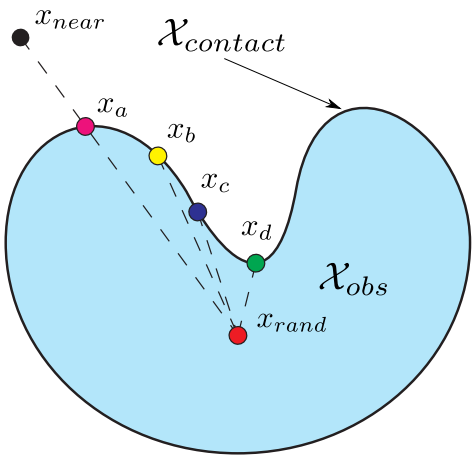


Figure 4.5: The retraction step performs iterative optimization. The tree is expanded toward the nodes $x_a, .., x_d \in \mathcal{X}_{contact}$ with the RRT expansion.

be connected by a collision-free path, the algorithm stops and returns a path (see Figure 4.4). If not, the trees are either swapped or not depending on the balance condition (Algorithm 3 – line 13). Then the loop starts again until a solution is found or the number of iterations expires.

4.2.2 Modification of the Extension Step

The extension step of the basic RRT is as simple as possible: for a given state x_{rand}, determine the nearest neighbour x_{near} and apply the control input to x_{near}. A new node x_{new} is obtained by this procedure. The tree is expanded toward x_{rand}. The described expansion step is good enough for simple problems but struggles in spaces with narrow passages. With high probability, a lot of nodes have to be added to the tree until one of the nodes will be situated in the narrow passage.

Modifications of the extension step improve the usability of RRT in narrow passages. The first algorithm to be discussed is *Retraction-based RRT* [50].

Retraction-based RRT This algorithm proceeds the same as the holonomic RRT until x_{new} does not lie in $\mathcal{X}_{obs}$. In that case, basic RRT ignores the new node and generates a new one. Retraction-based RRT runs a *retraction step* which determines the closest point to the parent such that the point lies on an obstacle boundary. The retraction step is an optimization problem defined as

$$x_m = \underset{x \in \mathcal{X}_{contact}}{\arg\min}\ \rho(x, x_{rand}), \tag{4.2}$$

where x_{rand} is a given colliding state, ρ is the metric and $\mathcal{X}_{contact} \subseteq \mathcal{X}_{obs}$ boundary of obstacles called the *contact space*. See Figure 4.5 for an illustration of the retraction step.

Algorithm 4 Retraction-based RRT

Require: Initial state x_I, number of iterations K, distance ε
Ensure: RRT tree $\mathcal{T}$

```
1:  T.init(x_I)
2:  for k = 0 to K do
3:      x_rand ← RANDOM_STATE()
4:      x_near ← NEAREST_NEIGHBOUR(x_rand, T)
5:      x_new ← NEW_STATE(x_near, ε)
6:      if x_new is feasible then
7:          T.add_vertex(x_new)
8:          T.add_edge(x_new, x_near, u)
9:      else
10:         S ← RETRACTION(x_rand, x_near)
11:         for each x_i ∈ S do
12:             STANDARD_RRT_EXPANSION(T, x_i)
13: return T
```

Given a colliding state x_{rand} and its nearest neighbour x_{near}, the retraction step proceeds as follows:

1. Project x_{near} into $\mathcal{X}_{contact}$ to generate a new sample x_a.
2. Perform a *contact query* by computing new samples around x_a, which have a minimal distance between the robot and obstacle.
3. Search the samples computed by the contact query and determine a new sample x_b in $\mathcal{X}_{contact}$, which locally minimizes the distance to x_{rand}. If the new sample x_b satisfy $\rho(x_b, x_{rand}) < \rho(x_a, x_{rand})$, continue to step 4. End the algorithm otherwise.
4. Assign $x_{near} = x_b$ and go to step 1.

The RETRACTION(x_{rand}, x_{near}) function assigns all samples $x_a, x_b, ...$ given by the retraction step to a set S. The standard expansion from the basic RRT is performed for all nodes from the set S (line 12 in Algorithm 4). Thus, the tree expands around the obstacles and the performance in narrow passages is highly increased.

RRT-blossom Another example of a method with the modified extension step is RRT-Blossom [51]. The algorithm is specific by a local flood-fill behaviour. This mechanism helps RRT-blossom to escape a local minimum, where the robot may get stuck, in highly constrained environments. RRT-blossom is well employable for non-holonomic and kinodynamic problems.

RRT-blossom starts, just like Retraction-based RRT, same as the basic RRT. It generates a random state x_{rand} and finds its nearest neighbour x_{near} in terms of the metric ρ. Then, control inputs u_i are assigned to a set $\mathcal{U}$. The inputs are chosen by the function SELECT_INPUTS(x_{near}) deterministically or stochastically. New states are computed by applying all

Algorithm 5 RRT-blossom

Require: Initial state x_I, number of iterations K
Ensure: RRT tree $\mathcal{T}$

```
T.init(x_I)
for k = 0 to K do
    x_rand ← RANDOM_STATE()
    x_near ← NEAREST_NEIGHBOUR(x_rand, T)
    U ← SELECT_INPUTS(x_near)
    for each u_i ∈ U do
        x_new ← NEW_STATE(x_near, u, Δt)
        x_n ← NEAREST_NEIGHBOUR(x_new, T)
        if x_new is feasible and REGRESSION(T, x_new) then
            T.add_vertex(x_new)
            T.add_edge(x_new, x_near, u)
return T
```

inputs from $\mathcal{U} \in U$ to x_{near}. If x_{new} is feasible and the *regression* function REGRESSION$(\mathcal{T}, x_{new})$ returns true, x_{new} is added to the tree $\mathcal{T}$.

The regression function is an important technique in RRT-blossom. It ensures the rapidly exploring property of the RRT algorithm by eliminating nodes which tend to regress as it is shown in Figure 4.6. Generally, implementing the regression function is a demanding task. For holonomic applications, this task is greatly simplified and we can effectively compute the regression only with knowledge of the distance metric ρ as

$$\text{REGR.}(\mathcal{T}, x_{new}) = \begin{cases} \text{FALSE} & \text{if } \exists x \in \mathcal{T} \mid \rho(x, x_{new}) < \rho(x_{near}, x_{new}) \\ \text{TRUE} & \text{otherwise} \end{cases}. \tag{4.3}$$

The regression function from Equation 4.3 returns TRUE only if no node from $\mathcal{T}$ is closer to x_{new} than the parent and therefore if $x_n = x_{near}$.

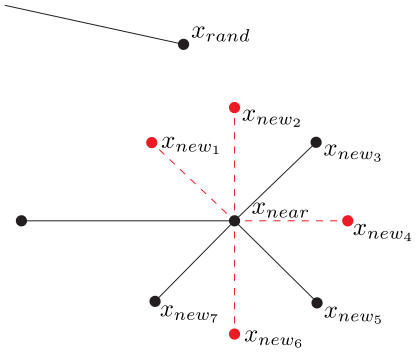
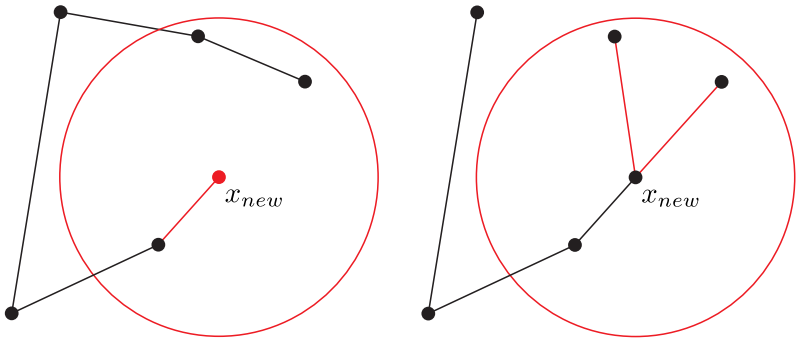


Figure 4.6: RRT-blossom adds nodes $x_{new_3}, x_{new_5}, x_{new_7}$ to the tree. Other nodes are excluded by the regression function because their nearest neighbour is not x_{near}.



(a) : Select the best parent **(b) :** Rewire the tree

Figure 4.7: RRT* performing the optimization of the tree

4.2.3 Optimal Planning

Disadvantage of all basic sampling-based methods is a non-optimal output path. This issue can be solved by utilizing an optimal RRT-based planner called *RRT** [52] introduced by Sertac Karaman and Emilio Frazzoli in 2011. It has been proven that the paths computed by RRT* have the asymptotically optimal property [52] and therefore RRT* always finds the shortest path in the infinite amount of time.

The optimal property of RRT* is redeemed by a longer computation time compared with the basic RRT. The longer computation time is caused by a feature called *rewiring* which improves the path quality as the algorithm runs. The original algorithm is limited to holonomic problems only.

Since the first optimal RRT-based algorithm was introduced, many new relevant RRT*-based algorithms have been proposed. We can name a non-holonomic expansion *Anytime RRT** [53] by the same authors as the original RRT*, memory efficient version *RRT*FN* [54] or a version with the smart sampling *Informed RRT** [55]. A complete survey on RRT*-based methods is presented in [56].

Before we will discuss RRT*, it is necessary to present functions used in Algorithm 6. For two given states $x_a, x_b \in \mathbb{R}^N$, LINE$(x_a, x_b)$ is a function $x_a, x_b \to \mathbb{R}$ that denotes a path length from x_a to x_b. Let COST(x_a) denote a function $x_a \to \mathbb{R}$, that returns a distance of x_a to the beginning when passing by a path. This definition implies that COST$(x_I) = 0$. Let PARENT(x_a) be a function $x_a \to x_b$ that returns a parent node of x_a. With respect to the previous definitions, we can write COST(x_a) = COST(PARENT(x_a)) + LINE$(x_a,$ PARENT$(x_a))$.

Algorithm 6 RRT*

Require: Initial state x_I, number of iterations K, distance ε
Ensure: RRT* tree $\mathcal{T}$

$\mathcal{T}$.init(x_I)
for $k = 0$ **to** K **do**
  $x_{rand} \leftarrow$ RANDOM_STATE()
  $x_{near} \leftarrow$ NEAREST_NEIGHBOUR($x_{rand}, \mathcal{T}$)
  $x_{new} \leftarrow$ NEW_STATE(x_{near}, ε)
  if x_{new} is feasible **then**
    $\mathcal{T}$.add_vertex(x_{new})
    $X_{near} \leftarrow$ NEIGHBOURS($x_{new}, \mathcal{T}, \min\{\gamma(\log|\mathcal{T}| \,/\, |\mathcal{T}|)^{1/N}, \eta\}$)
    $x_{min} \leftarrow x_{near}$
    $c_{min} \leftarrow$ COST(x_{near}) + LINE(x_{near}, x_{new})
    for each $x_n \in X_{near}$ **do**
      if x_n is feasible from x_{new} **and** COST(x_n)+LINE(x_n, x_{new}) < c_{min} **then**
        $x_{min} \leftarrow x_n$
        $c_{min} \leftarrow$ COST(x_n) + LINE(x_n, x_{new})
      $\mathcal{T}$.add_edge(x_{new}, x_n)
    for each $x_n \in X_{near}$ **do**
      if x_n is feasible from x_{new} **and** COST(x_{new}) + LINE(x_n, x_{new}) < COST(x_n) **then**
        $x_{parent} \leftarrow$ PARENT(x_n)
        $\mathcal{T}$.remove_edge(x_n, x_{parent})
        $\mathcal{T}$.add_edge(x_n, x_{new})
return $\mathcal{T}$

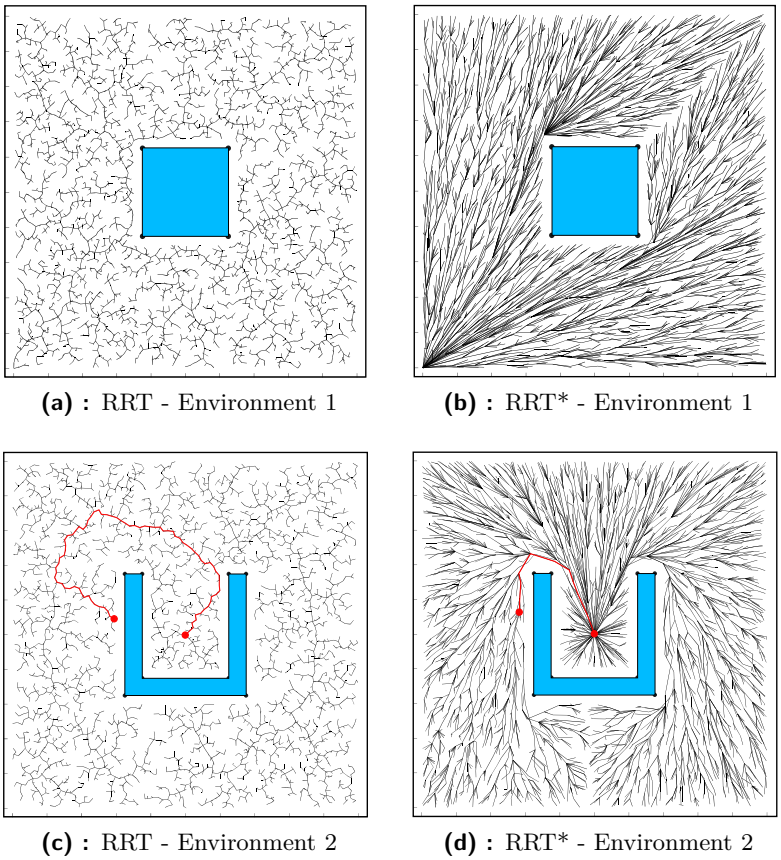
Given an initial state x_I, empty tree $\mathcal{T}$ and state space $\mathcal{X}$, RRT* proceeds as follows:

1. **Holonomic RRT** Add the initial state to the tree $\mathcal{T}$.
2. Generate a random state $x_{rand} \in \mathcal{X}$.
3. Select the nearest neighbour of x_{rand} in terms of metric ρ. Denote it x_{near}.
4. Determine a new state x_{new} by connecting x_{rand} to x_{near} by a straight line. If LINE(x_{rand}, x_{near}) < ε, then $x_{new} \leftarrow x_{rand}$. Compute a state that satisfy LINE(x_{new}, x_{near}) = ε and lie on the line otherwise.
5. If x_{new} is reachable by a collision-free path, add it to the tree. If not, ignore it and go to step 2.
6. **Best parent** Select all neighbours of x_{new} within the distance

$$\min\left\{\gamma\left(\frac{\log|\mathcal{T}|}{|\mathcal{T}|}\right)^{\frac{1}{N}}, \eta\right\} \tag{4.4}$$

where γ and η are parameters dependent on the environment, $|\mathcal{T}|$ is the number of nodes in the tree and N is the number of dimensions.

7. Cycle through all the neighbours and find the parent of x_{new} with a minimum-cost path to the initial node among them.
8. **Rewiring** Cycle again through all the neighbours and if a path to the node through x_{new} is shorter, remove the existing connection to its parent and add the new less costly connection. Assign x_{new} as the new parent. See Figure 4.7 for an illustration of the rewiring procedure.



(a) : RRT - Environment 1 **(b) :** RRT* - Environment 1

(c) : RRT - Environment 2 **(d) :** RRT* - Environment 2

Figure 4.8: Comparison of RRT and RRT* for a holonomic robot. The path generated by RRT* (d) is shorter than the path generated by RRT (c).

Chapter 5

Our Contribution

This chapter is dedicated to our contribution to the motion planning. In the first section, we introduce a new algorithm for the holonomic motion planning based on the basic RRT. In the other section, our new algorithm RRT-sphere and other implemented RRT-based algorithms are tested by means of own 3D models and benchmark [15] for their execution time, number of explored nodes and length of the final path.

5.1 RRT-sphere

RRT-sphere is a novel RRT-based algorithm we propose to improve the performance of RRT in low constrained environments. Every discussed algorithm so far utilized uniform sampling scheme. Proposed RRT-sphere samples the state space by adaptive sampling which attracts the samples toward the goal state whether it is possible. The principle of the non-uniform sampling ensures that no redundant nodes are created and thus it saves system resources like RAM and CPU.

This method is designed to solve holonomic and non-holonomic problems, but we only focus on holonomic systems in the thesis. Testing RRT-sphere under non-holonomic constraints and automatic tuning of parameters is a possible avenue for the future work.

Unlike the RANDOM_STATE() function from previous algorithms, the method for generating new samples is modified in RRT-sphere. RANDOM_STATE (x_G, h) generates new states around the goal state x_G within the radius h. The part of the state space, where random samples x_{rand} are located, looks like N-D *hypersphere* with the centre in x_G, where N is the number of translation dimensions of the state space.

Algorithm 7 RRT-sphere

Require: Initial state x_I, goal state x_G, number of iterations K, initial radius h
Ensure: RRT tree $\mathcal{T}$

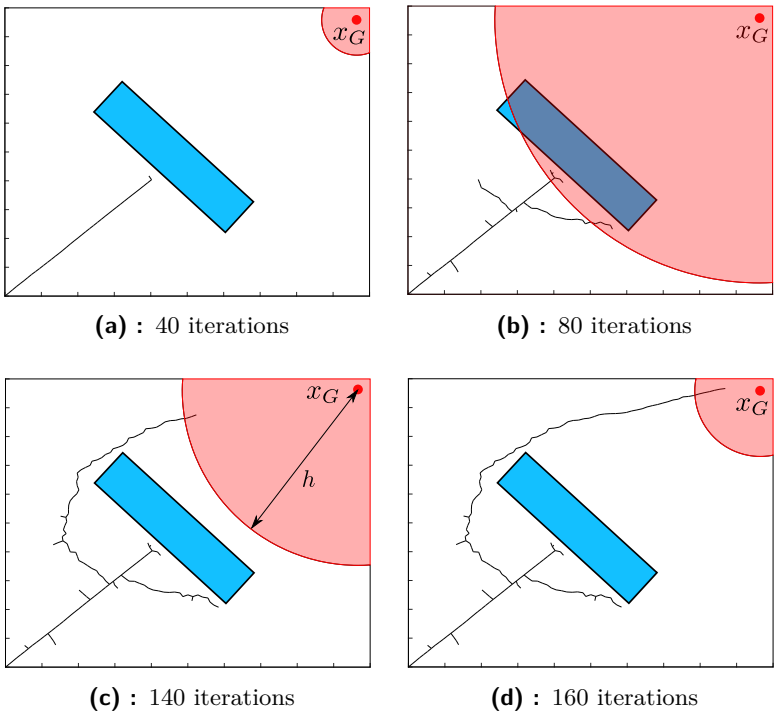
```
1: T.add_vertex(x_I)
2: a, b ← 0
3: for k = 0 to K do
4:     if k mod α = 0 then
5:         a ← 0
6:         b ← 0
7:     h ← SET_RADIUS(a, b)
8:     x_rand ← RANDOM_STATE(x_G, h)
9:     x_near ← NEAREST_NEIGHBOUR(x_rand, T)
10:    u ← SELECT_INPUT(x_rand, x_near)
11:    x_new ← NEW_STATE(x_near, u, Δt)
12:    if x_new is feasible then
13:        T.add_vertex(x_new)
14:        T.add_edge(x_new, x_near, u)
15:        a ← a + 1
16:    else
17:        b ← b + 1
18: return T
```

The radius changes dynamically according to the ratio of non-colliding and colliding nodes in the function SET_RADIUS(a, b), where a is the number of feasible nodes and b is the number of colliding nodes. The counters a and b reset to 0 after the established number of iterations expires. The number of required iterations to reset the counter is denoted α. Let β denote the *stretching factor* which represents the speed of expansion/shrinking of the area around x_G ($\beta \in < 0, 1 >$). Adaptation of the radius is shown in Figure 5.1. The full pseudo-algorithm of RRT-sphere is shown in Algorithm 7.

The function that controls change of the radius h is defined as:

$$\text{SET\_RADIUS}(a,b) = \begin{cases} h \leftarrow h + \beta h & \text{if } k \bmod \alpha = 0 \wedge a < b \\ h \leftarrow h - \frac{\beta}{2} h & \text{if } k \bmod \alpha = 0 \wedge a > b \\ \text{unchanged } h & \text{otherwise} \end{cases} \tag{5.1}$$

Provided that there exists a path without majority of narrow passages from x_I to x_G, RRT-sphere finds the path using a smaller amount of samples in comparison with the original RRT and other RRT-based methods with the uniform probing of the space. This algorithm never produces worse results than the basic RRT because in the worst case, h is large enough to cover the all state space yielding in the uniform coverage.



(a) : 40 iterations **(b) :** 80 iterations

(c) : 140 iterations **(d) :** 160 iterations

Figure 5.1: Changing of the radius h in RRT-sphere according to the number of successfully added nodes. New nodes are generated in the red area.

5.2 Experimental Results

The experimental section of the thesis is divided into three parts. In the first part, we analyse RRT-sphere parameters α, β and their influence on the motion planning performance. Then, we perform experiments on various motion planning problems in 2D worlds with all implemented algorithms — RRT [12], Retraction-based RRT [50], RRT-blossom [51], RRT* [52], RRT-sphere, and in 3D worlds with three selected algorithms. The algorithms are compared in detail and results are provided in the section.

5.2.1 RRT-sphere Parameters

The correct initializing of parameters is the important preliminary phase of all sampling-based motion planning algorithms. Wrongly tuned parameters can lead to substandard quality of the planning. Internal constants of motion planning algorithms affect the run-time, memory requirement and quality of the final path. There exists a specific set of parameters which is optimal for the particular planning task and gives the best results. It is not possible to tune the parameters correctly without knowledge of the environment.

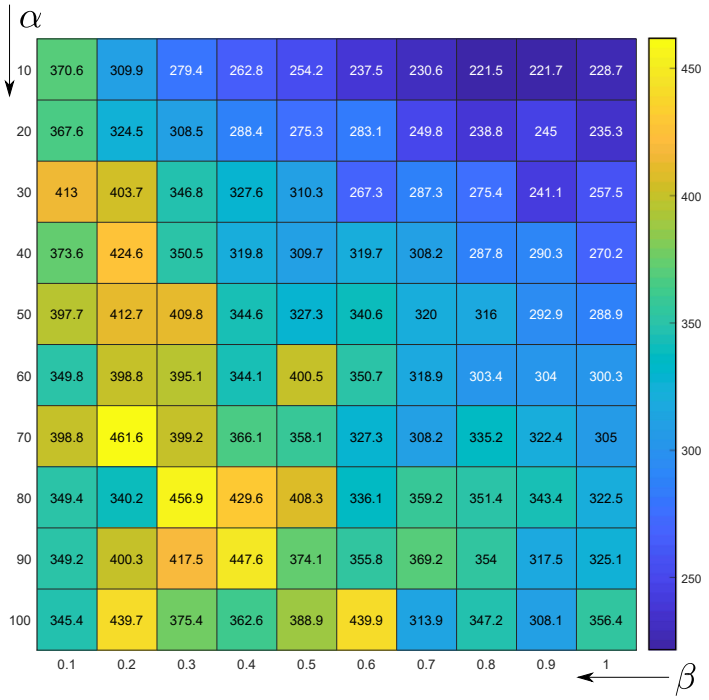


Figure 5.2: The dependency of RRT-sphere parameters α and β on the number of nodes required to find a path in the 2D map

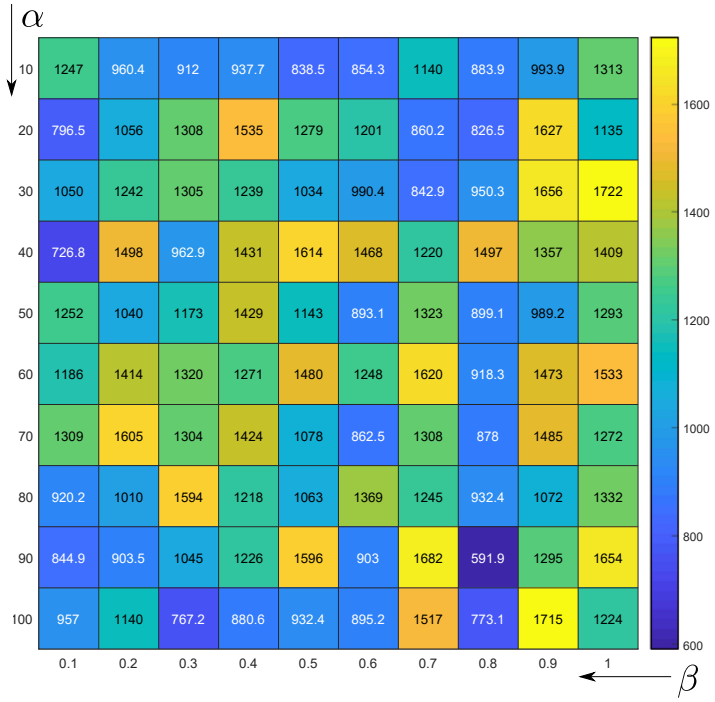
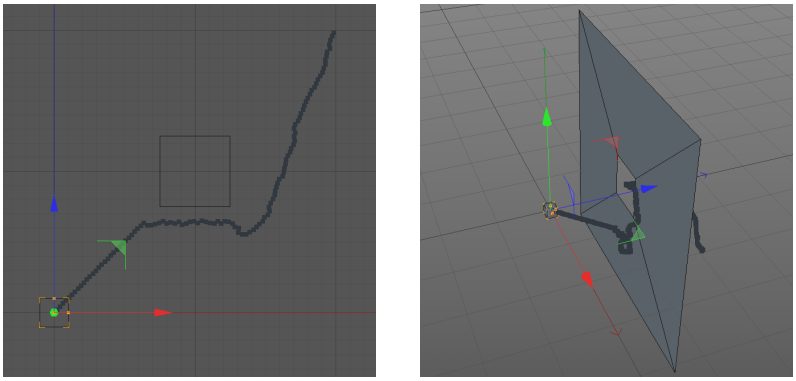


Figure 5.3: The dependency of RRT-sphere parameters α and β on the number of nodes required to find a path in the 3D map



(a) : The 2D map **(b) :** The 3D map

Figure 5.4: Two testing maps utilized for evaluating RRT-sphere parameters with calculated paths

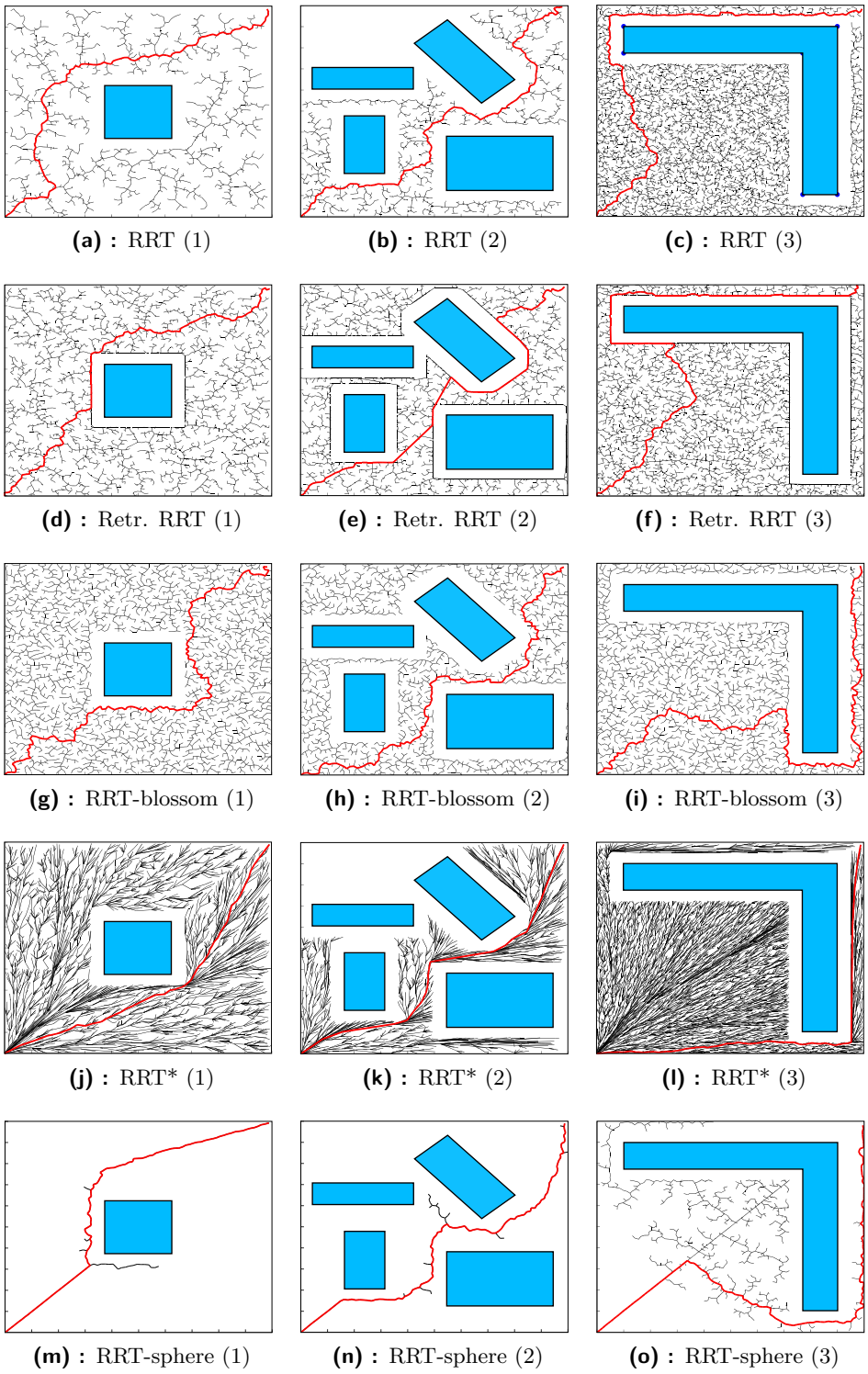
The maps used for testing the dependency of RRT-sphere parameters on the planning quality are shown in Figure 5.4. We determine the number of generated nodes until the algorithm finds a path. The parameter α is examined in the interval $< 10, 100 >$ and the parameter β in the interval $< 0.1, 1 >$. Results are visualised as *heat maps* in Figure 5.2 and Figure 5.3. The value in each box is the mean of ten tests. For comparison, the basic RRT generates 5000 nodes on average in the 2D map.

In the 2D map, there can be observed a strong correlation between the parameters and number of needed nodes to find a path. When the parameters approach closer to $\alpha = 10$ and $\beta = 0.1$, the number of generated nodes reduces. Experiments in the 3D map do not exhibit the correlation. Best results in the 3D map are produced by $\alpha = 90$ and $\beta = 0.8$.

5.2.2 Experiments in 2D Environments

We perform experiments in 2D environments with the robot which is represented by a square. The robot is capable of moving in two DOF and its configuration (state) is $q = (x, y)$, where x and y are translational coordinates. The system is holonomic and thus the robot can displace independently in both directions. The state space has dimensions 200 cm by 200 cm and the robot has dimensions 20 cm by 20 cm.

For the testing, we designed three maps: easy (1), obstacles (2) and L-trap (3) An assignment for the robot is to find a path from $(0, 0)$ to $(200, 200)$. The step size is set to 3 for all algorithms. We utilized Euclidean metric. Examples of computed paths in all testing maps are shown in Figure 5.5.



(a) : RRT (1) **(b) :** RRT (2) **(c) :** RRT (3)

(d) : Retr. RRT (1) **(e) :** Retr. RRT (2) **(f) :** Retr. RRT (3)

(g) : RRT-blossom (1) **(h) :** RRT-blossom (2) **(i) :** RRT-blossom (3)

(j) : RRT* (1) **(k) :** RRT* (2) **(l) :** RRT* (3)

(m) : RRT-sphere (1) **(n) :** RRT-sphere (2) **(o) :** RRT-sphere (3)

Figure 5.5: Five RRT-based algorithms in various 2D maps. The best results are achieved by our proposed algorithm RRT-sphere.

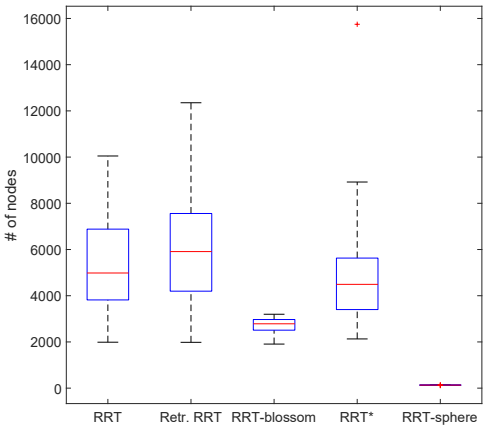


Figure 5.6: The number of nodes – 2D map *easy* (1)

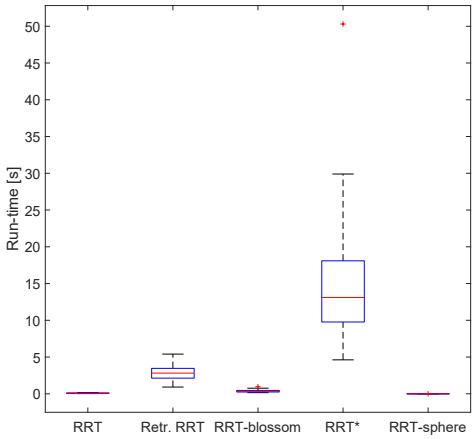


Figure 5.7: The run-time – 2D map *easy* (1)

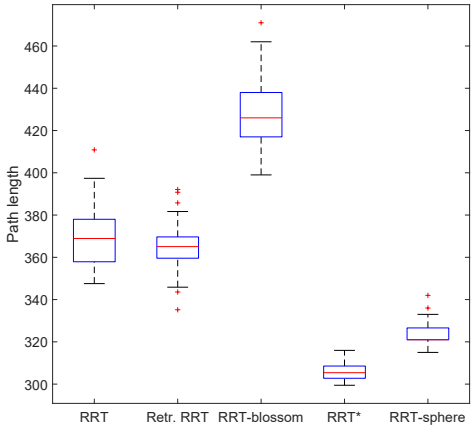


Figure 5.8: The path length – 2D map *easy* (1)

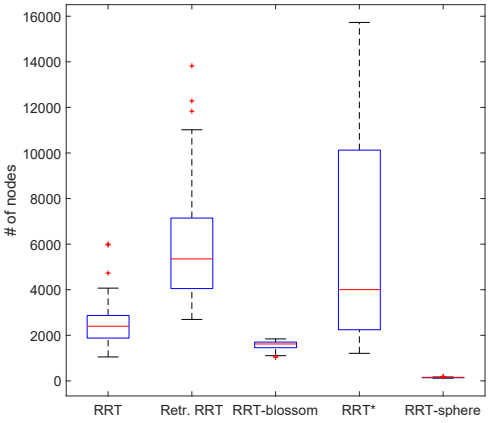


Figure 5.9: The number of nodes – 2D map *obstacles* (2)

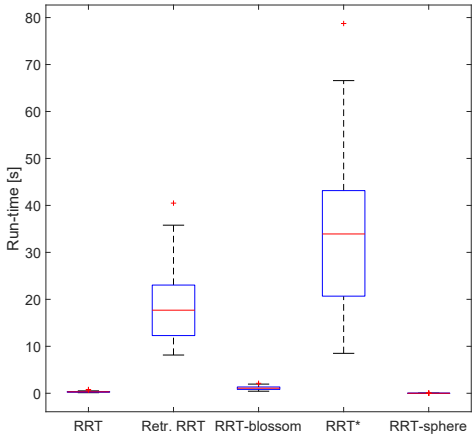


Figure 5.10: The run-time – 2D map *obstacles* (2)

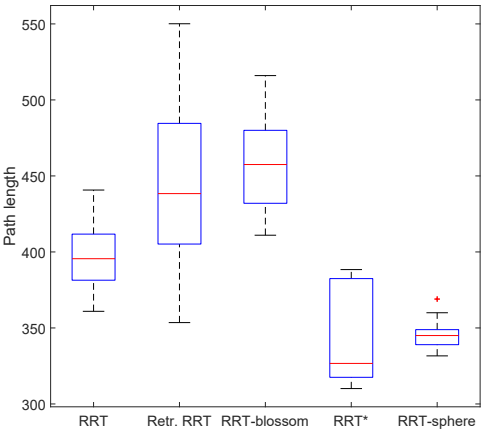


Figure 5.11: The path length – 2D map *obstacles* (2)

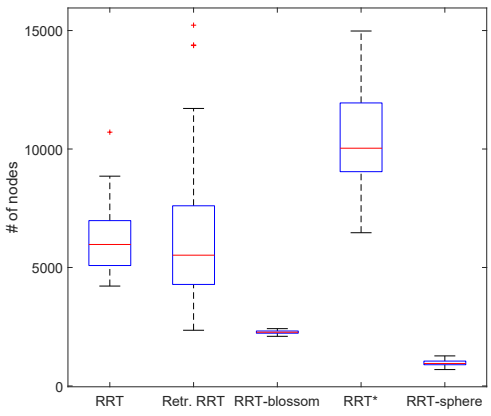


Figure 5.12: The number of nodes – 2D map *L-trap* (3)

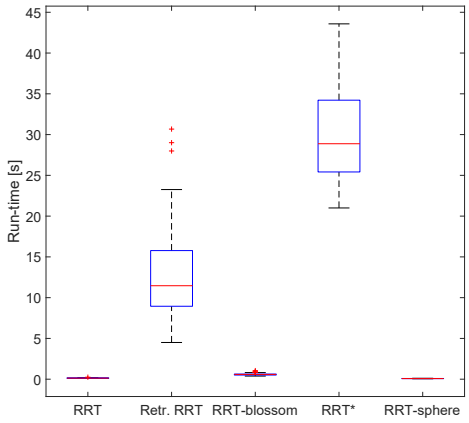


Figure 5.13: The run-time – 2D map *L-trap* (3)

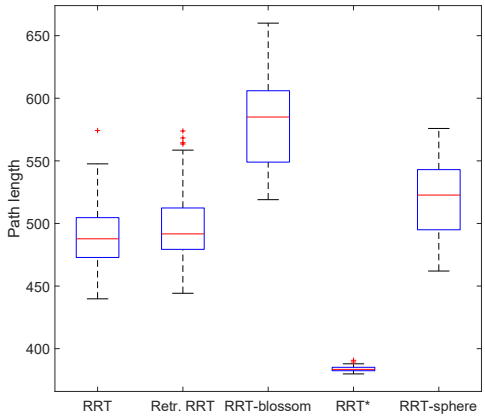


Figure 5.14: The path length – 2D map *L-trap* (3)

In Retraction-based RRT, there are 5 attempts to generate one node in the obstacle space. The maximal number of nodes laying in the obstacle space given by the RETRACTION function is set to 5 and maximal length between two retraction nodes is 3. The γ constant for RRT* is set to 100. The maximal number of expansions in RRT-blossom is 5. In the case of RRT-sphere, $\alpha = 10$ and $\beta = 0.1$ since the best results were achieved with this setting. There was done 50 tests in each map.

Figures 5.6–5.14 show performance of all algorithms as the number of nodes in the tree, path length the run-time. The central mark indicates the median, and the bottom and top edges of the box indicate the 25th and 75th percentiles, respectively. The maximal and minimal values are indicated by marks at the end of the dashed lines. Success rates according to the number of required iterations to find a solution in all maps are shown in Figures 5.15–5.17.

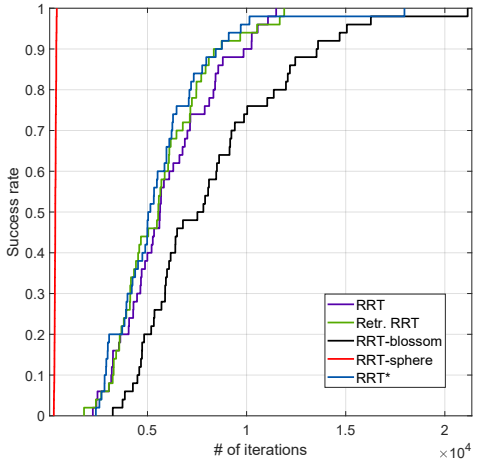


Figure 5.15: The success rate on the *easy* map

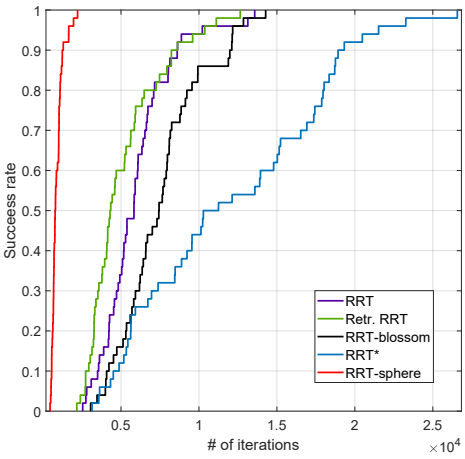


Figure 5.16: The success rate on the *obstacles* map

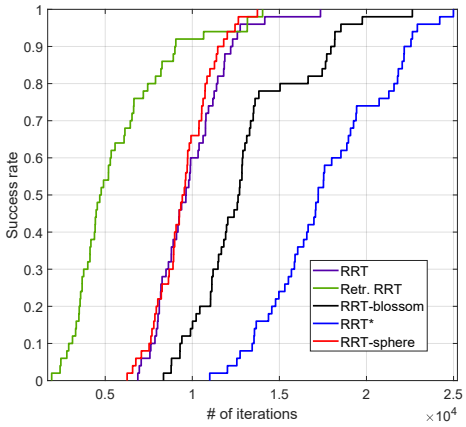
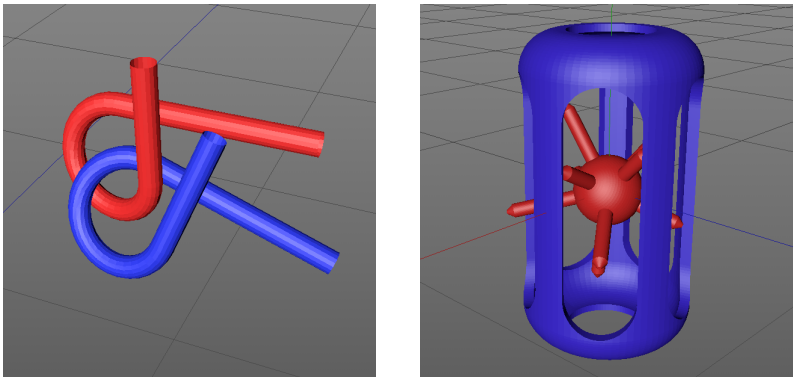


Figure 5.17: The success rate on the *L-trap* map

From the figures, it can be observed that the RRT* run-time is approximately 100 times longer than the RRT run-time. It ensues from the rewiring feature in RRT* where all neighbours of the selected node are reconnected to achieve the optimality of the path. On the other hand, paths given by RRT* are shorter by 20 % in compare with RRT.

RRT-blossom generates the smallest number of nodes to cover the space, but paths computed by RRT-blossom are distinctly sub-optimal. Retraction based-RRT places nodes on the boundary of the obstacle state space in order to improve the performance in narrow passages. In the 2D testing maps, RRT-sphere produces best results in terms of the number of generated nodes and computational time. In contrast with RRT-blossom, RRT-sphere does not cover the space equally from the initial node. Summarizing, all implemented algorithms excel in a specific field, and for each task, a different algorithm is the most suitable for the task.



(a) : The alpha puzzle **(b) :** Hedgehog in the cage

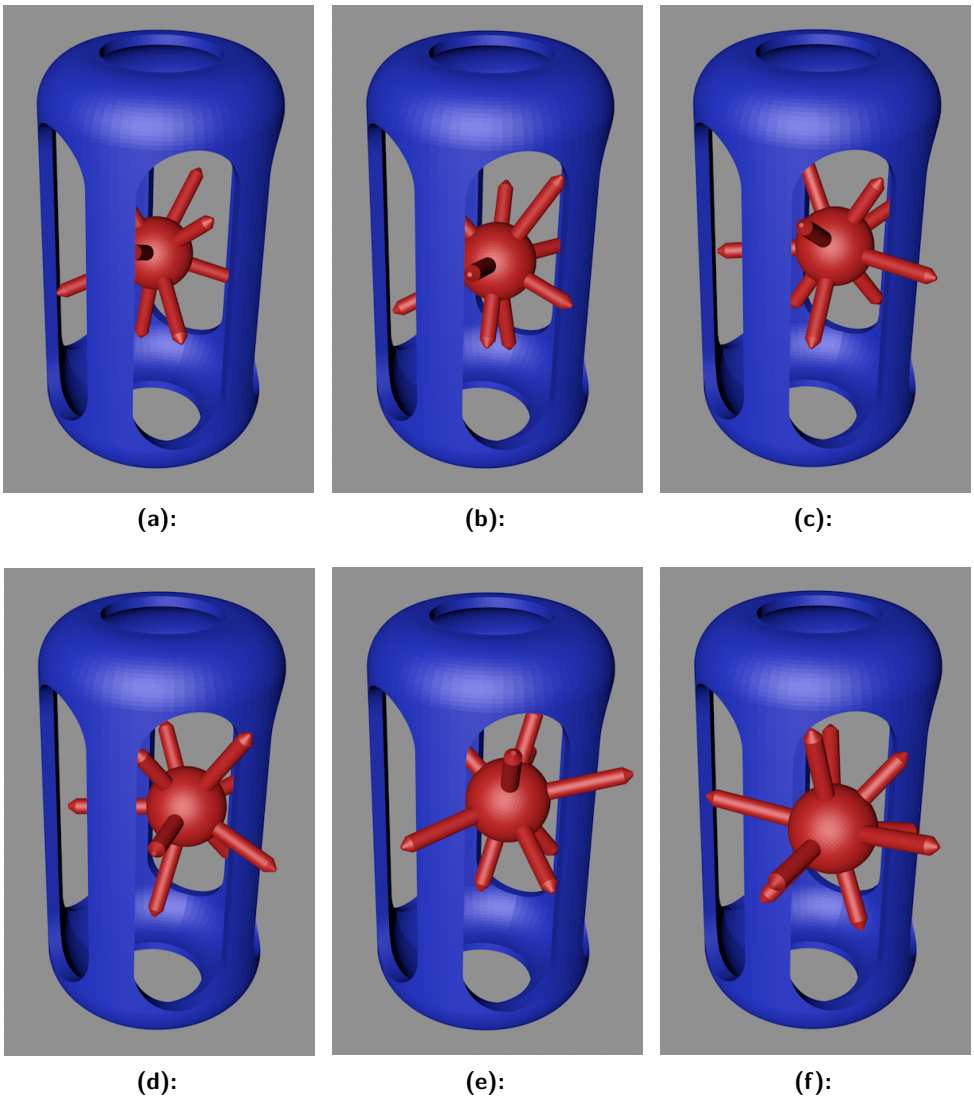
Figure 5.18: Two testing environments utilized for 3D experiments. The red object in both environments represents the robot.

5.2.3 Experiments in 3D Environments

For 3D experiments, we selected three environments – hedgehog in the cage designed by us, alpha puzzle from [15] (both shown in Figure 5.18) and protein models. The robots are capable of moving in 6 DOF (3x translations, 3x rotations). Because of the difficulty of the tasks, we used modified versions of the objects. In the alpha puzzle, the narrow passage between two tubes is expanded. The full alpha puzzle computation was not completed in 48 hours, same as in the hedgehog case. The Hedgehog in the cage modification was done by scaling the hedgehog to 70 %, 80 % and 90 % of its original size.

The first experiment in 3D environments was performed on the alpha puzzle. There were chosen two RRT-based algorithms (RRT-blossom, RRT-sphere) for their results in 2D maps and the original RRT for comparison. Full results for 50 tests are shown in Figures 5.20–5.24. Unlike the 2D experiments, we added an extra graph to show only translational component of the final path length. The shortest run-times are achieved with our proposed algorithm RRT-sphere, as well as the the smallest number of generated nodes. The longest paths are generated by RRT-blossom, but considering only displacements, the differences among path lengths are not substantial (see Figure 5.23).

The second experiment consists of scaled hedgehog as the robot and the cage around. This planning problem involves challenging narrow passages. RRT is able to solve the 70% hedgehog in 3 s on average (Figure 5.26), 80% hedgehog in 3.5 h and 90% hedgehog in 47 h. The computed path for the 80% hedgehog is illustrated in Figure 5.19 and a video showing the path is included on the attached CD.



(a): (b): (c):

(d): (e): (f):

Figure 5.19: The computed trajectory of the hedgehog scaled to 80 % by RRT-sphere (computation time 3.5 h)

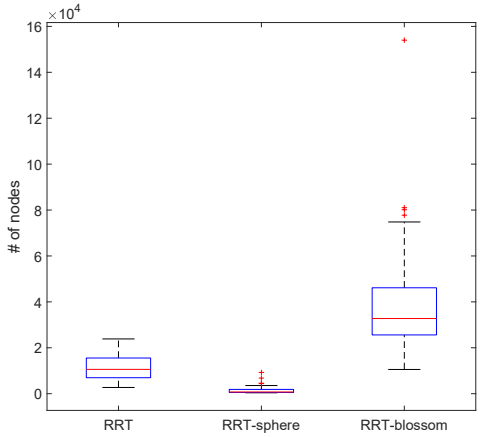


Figure 5.20: The number of nodes in the alpha puzzle map

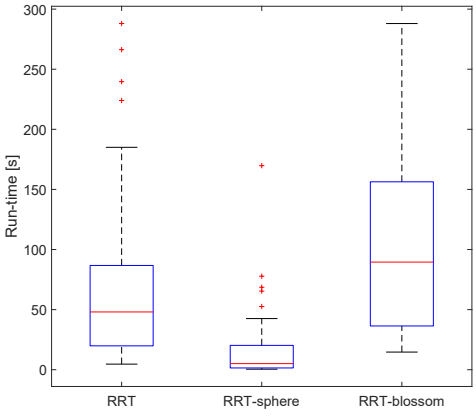


Figure 5.21: The run-time in the alpha puzzle map

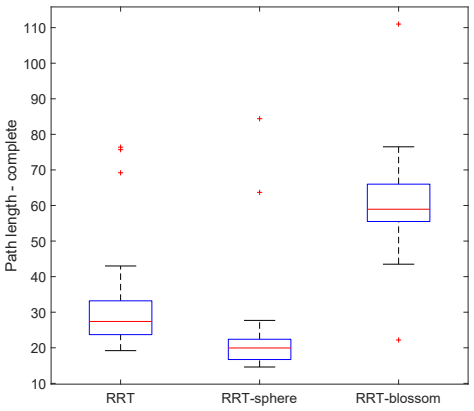


Figure 5.22: The complete path length including rotations in the alpha puzzle map

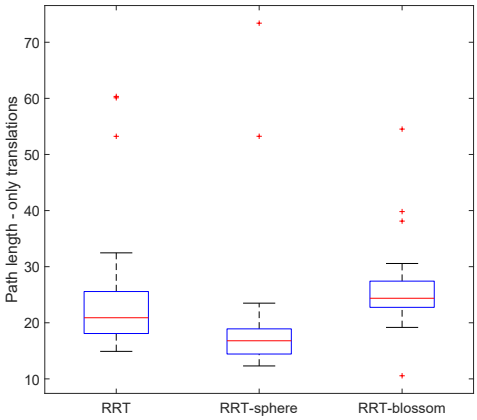


Figure 5.23: A translational component of the path length in the alpha puzzle map

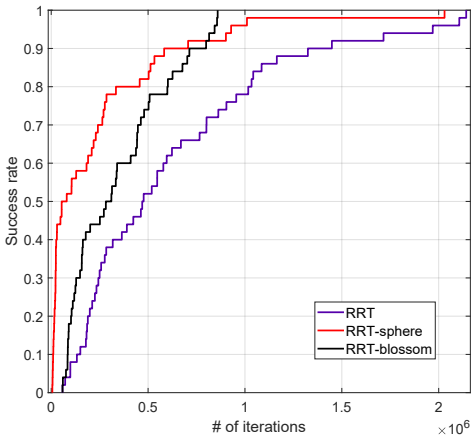


Figure 5.24: The success rate on the the alpha puzzle map

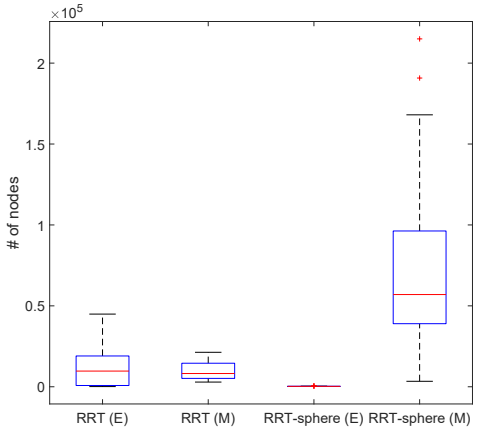


Figure 5.25: Comparison of the Euclidean and Manhattan metric – the number of nodes

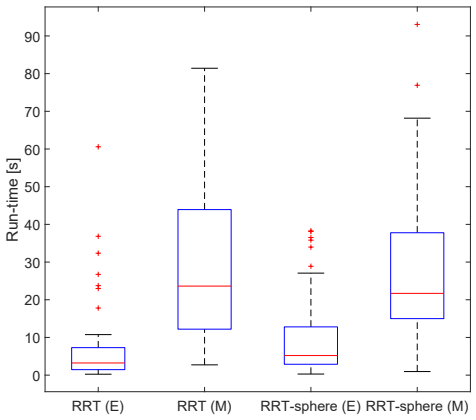


Figure 5.26: Comparison of the Euclidean and Manhattan metric – the run-time

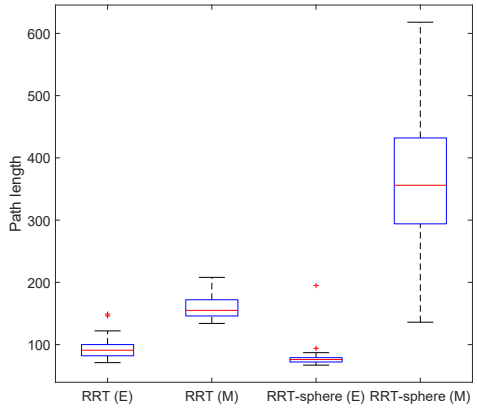


Figure 5.27: Comparison of the Euclidean and Manhattan metric – the path length including rotations

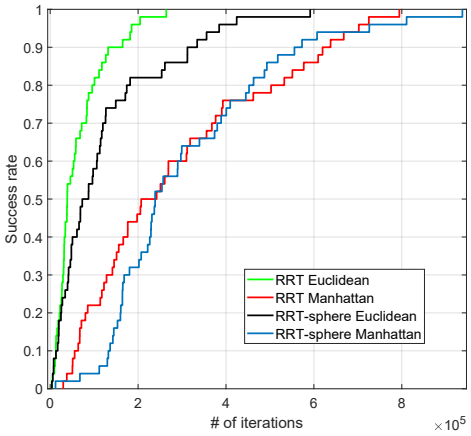


Figure 5.28: Comparison of the Euclidean and Manhattan metric – the success rate

Then, we evaluated the metric influence on the planning performance. As a testing map, we utilized hedgehog in the cage scaled to 70 %. Results for the basic RRT algorithm and RRT-sphere are shown in Figures 5.25–5.28. A letter "E" stands for the Euclidean metric and a letter "M" stands for the Manhattan metric (see Section 2.1 for further information). It has been shown that the Euclidean metric is more suitable for the problem as the run-times, number of nodes and path lengths are lower. The number of required iterations to find a path is lower as well. The only exception is the number of nodes in RRT, where the Manhattan metric gives slightly better results on average than the Euclidean metric.

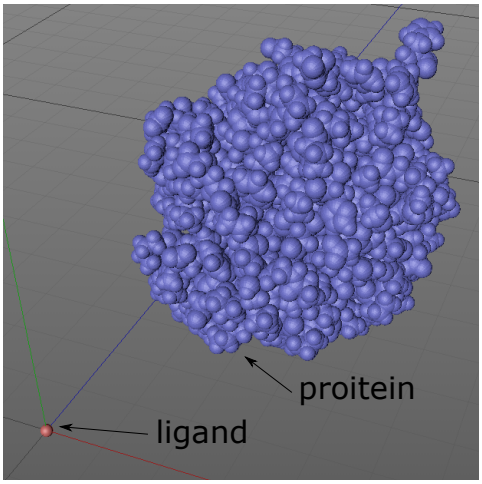


Figure 5.29: Model of a protein and simple ligand

Finally, we performed an experiment with molecule structures. The model consists of a large molecule (protein) and a small molecule (ligand) as shown in Figure 5.29. The task of the ligand is to escape from the center of the protein and get to the position on the edge of the protein. The protein is made of 4652 atoms approximated by triangulated spheres. The ligand is a single atom with radius 0.7 Å[1]. The spheres representing atoms contain 60 triangles. For the accurate planning, more triangles would be needed. However, for purposes of the demonstration in this thesis, the number of triangles is sufficient.

Table 5.1 shows rounded results of the algorithms after one pass. The step size is set to 0.1 Å. The results are heavily dependent on settings of the parameters, and precise tuning of them would lead to better results. A video showing the path of the ligand computed by the RRT-sphere algorithm is included on the attached CD.

	# of nodes	Run-time [s]	Path length	# of iterations
RRT	197 000	46.6	77.7	589 000
Retr. RRT	316 000	588.3	54.2	285 000
RRT-blossom	336 000	238.8	56.2	379 000
RRT*	271 000	608.6	40.5	392 000
RRT-sphere	25 000	7.5	74.7	94 000

Table 5.1: Results of five RRT-based algorithms from the molecules planning

[1] A unit of length equal to 1 nm used extensively in chemistry.

Chapter 6

Conclusions

The motion planning is used in the extensive number of applications every day. There is a growing need for development of new improved motion planning algorithms as new tasks arise. At present, the major field where the motion planning is used, is the mobile and intelligent robotics. We can mention self-driving cars since they are considered as the future of the automotive industry.

This bachelor thesis was focused on the analysis of the group of sampling-based motion planning algorithms. The sampling sampling-based algorithms was a novel and revolutionary approach to the motion planning. They allowed an effective planning in highly constrained environments with many number of DOF which was not possible before. We have successfully implemented five sampling-based algorithms, all based on the Rapidly Exploring Random Trees algorithm, in our planning application.

The application was utilized for testing the algorithms in various 2D and 3D maps. We employed the algorithms to solve three 2D problems. We determined the pros and cons of all methods and illustrated results in figures with computed paths and graphs with all relevant data. Selected algorithms were used to evaluate the performance on two puzzles and protein model because in biochemistry, the motion planning has a significant position. In the experimental part, we determined that the Euclidean metric is a preferable metric in the motion planning.

In the thesis, we proposed a novel RRT-based algorithm called RRT-sphere. This method is based on the basic RRT algorithm and improves its properties such as the number of generated nodes and run-time. The main feature in RRT-sphere is the adaptive sampling radius around the goal state which expands/shrinks according to the ratio of colliding and non-colliding new nodes. RRT-sphere yields the good experimental performance over a wide variety of 2D and 3D examples.

Appendix A

List of Notation and Abbreviations

Symbol	Meaning
$\mathcal{X}$	The state space
$\mathcal{C}$	The configuration space
$\mathcal{O}$	The obstacle region
U	The set of control inputs
$\mathcal{W}$	The world
$\mathcal{A}$	The robot
x	The state
q	The configuration
u	The control input
ρ	The metric
$\mathcal{T}$	The RRT tree

Abbreviation	Meaning
PRM	Probablistic Roadmap
RRT	Rapidly exploring Random Tree
DOF	Degree Of Freedom
RNG	Random Number Generator

Appendix B

Motion Planning Software

The software is designed to handle holonomic planning problems in the 3D world. The world involves a rigid robot and stationary obstacles. The robot is capable of displacement and rotation around the pivot and hence it has 6 DOF (3x translation, 3x rotation) and the state space is 6D. A block diagram of the application is shown in Figure B.1.

The following external libraries are implemented in the software:

- **MPNN** - *Nearest neighbour library for the motion planning by Anna Yershova and Steven M. LaValle, University of Illinois* [40]. MPNN is a key element in our motion planner because the nearest neighbour searching is one of the most challenging tasks in the motion planning. Utilizing a basic linear search would increase the process time dramatically.
- **RAPID** - *Robust and Accurate Polygon Interference Detection (collision detection) by Stefan Gottschalk, Ming C. Lin, Dinesh Manocha, University of North Carolina* [44]. Another key component for sampling-based planning methods is a collision detector. RAPID is an easy to use detector with a set of uncoupled triangle polygons as the input.
- **OBJ LOADER** - *A C++ OBJ Model Loader* [57]. To import objects assembled from polygons to RAPID, a parser that decomposes an object (.obj) file into a triangle mesh is used.
- **TinyXML-2** - *A C++ XML parser by Lee Thomason* [58]. All necessary parameters for the planning such as an object file, type of a planner, start and goal configuration and more are sent to the planner via an XML file. TinyXML-2 carries out the task of transferring a given XML file with parameters to a set of strings and number values.
- **effolkronium random** - *Random for modern C++ with convenient API by Ilya Polishchuk* [59]. To simplify the work with random numbers used for generating random samples in RRT, we use the API for C++11.

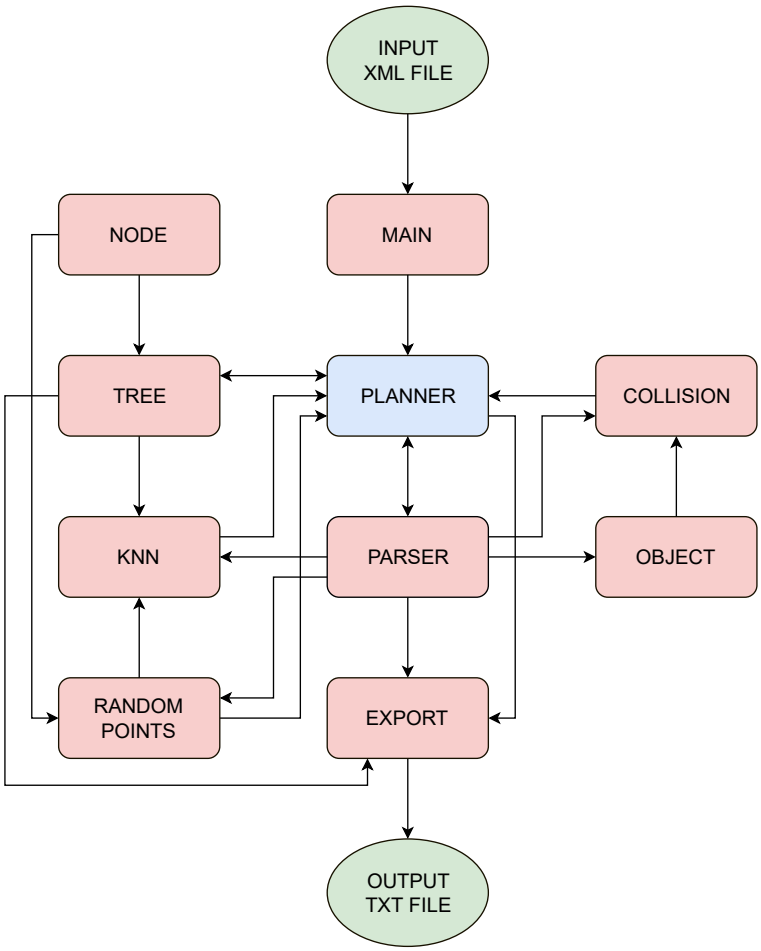


Figure B.1: A block diagram of the application. Arrows illustrate dependences of application classes.

For proper functionality of the application, there needs to be three files in the root directory: an object file with .obj extension, XML file and Makefile. The object file includes just two items, an obstacle and robot in this order. The robot has a reference point in the center of the coordinate system. Any 3D modeling software can be used to generate the file (Blender [60], Cinema 4D [61]). For example, the object file may look like this:

```
# WaveFront *.obj file (generated by CINEMA 4D)
v 0 0 0
...                          #list of vertices of the obstacle
# 8 vertices

vn 0 0 -1
...                          #list of normals of the obstacle
# 6 normals
```

```
o Obstacle
usemtl default
f 1//1 2//1 4//1 3//1
...                          #list of polygons of the obstacle

v 50 50 50
...                          #list of vertices of the robot
# 8 vertices

# 0 normal                   #list of normals of the robot
                             (same normals as the obstacle)
o Robot
usemtl default
f 9//1 10//1 12//1 11//1
...                          #list of polygons of the robot
```

The other file is the input XML file, in which are defined all constants and parameters. Structure of the file is following:

```
<Planning_task>
    <Name>Testing</Name>            <!-- name of the task -->
    <Object>objects.obj</Object>    <!-- the object file -->
    <Planner>RRT</Planner>          <!-- employed planner -->
    <Task_number>1</Task_number>    <!-- task number -->
    <Parameters>
        <iteration_limit>2000</iteration_limit>
        .
        .
        <beta>10</beta>
    </Parameters>
</Planning_task>
```

In the application, five RRT-based algorithms are implemented. A user selects the algorithm by the <Planner> tag. Available options are:

- **Basic RRT** – <Planner>RRT</Planner>
- **Retraction-based RRT** – <Planner>RRTRetraction</Planner>
- **RRT*** – <Planner>RRTStar</Planner>
- **RRT-blossom** – <Planner>RRTBlossom</Planner>
- **RRT-sphere** – <Planner>RRTSphere</Planner> (Our new algorithm discussed in Section 5.2)

In the case of other option, the planner stops and returns an error.

The parameters for a correct functionality of the planner are defined in the <Parameters> group. General parameters, necessary for all planners, are:

- *iteration_limit* – The number of RRT expansions. If set to 0, the planner iterates until it finds a valid path.
- *time_limit* – After this time elapses, the application terminates. The time limit is in seconds.
- *init_pos_x, init_pos_y, init_pos_z* – Translation coordinates of the initial state.
- *init_rot_x, init_rot_y, init_rot_z* – Initial rotation of the initial state.
- *goal_pos_x, goal_pos_y, goal_pos_z* – Translation coordinates of the goal state.
- *goal_pos_radius* – A radius with a centre in the goal state defining the set of goal states.
- *state_space_x, state_space_y, state_space_z* – Sizes of translational components of the state space.
- *step_size* – A maximal length between two nodes ε.
- *interpolation_step* – A step size to determine whether an input path is collision-free.
- *metric* – The used metric. 1 for Euclidean, 2 for Manhattan.
- *scale_trans* – A scale of translation coordinates.
- *scale_rot* – A scale of rotation coordinates.

General parameters are followed by specific parameters for a particular planner:

- *retraction_nodes* – The number of attempts to generate one node in the obstacle space for retraction-based RRT.
- *retraction_iterations* – The maximal number of nodes laying in the obstacle space given by the RETRACTION function.
- *retraction_step* – A maximal length between two neighbouring nodes given by the RETRACTION function.
- *gamma* – The parameter γ for RRT*.
- *inputs* – The number of control inputs of which new states are computed in RRT-blossom $|\mathcal{U}|$.
- *alpha* – The parameter α for RRT-sphere.
- *beta* – The parameter β for RRT-sphere.

Full example of the file is shown in Apendix D.

The state space defined by the *state_space_xxx* parameter determines the samples generated by the *RNG*[1]. If the *state_space_x* is a, x coordinate for samples is generated in the interval $< 0, a >$. Rotational coordinates are generated in the interval $< -\pi, \pi >$.

After making the binary file by the enclosed Makefile (requires gcc 4.9 or later), the planning starts by executing following commands in a linux terminal:

```
chmod 777 bin/a.out
./bin/a.out task.xml
```

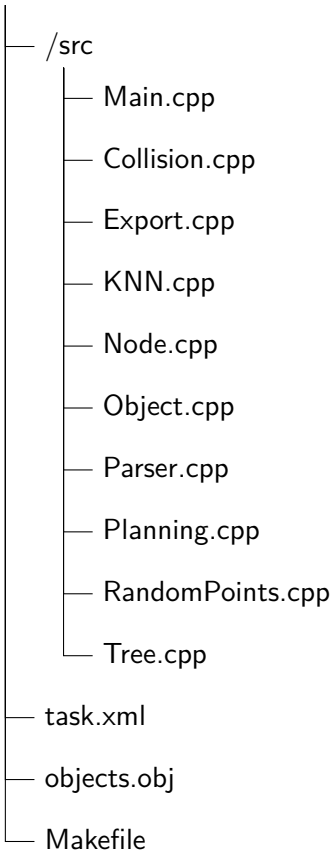
After the planner finishes its task, it prints all relevant information to the terminal and file *info.txt*. This information includes the number of nodes in the path, path length and run-time. List of all connections from the tree is exported to a file called *nodes.txt*. The computed path is listed in the file *path.txt*. The files can be used for visualization.

[1] RNG – Random Number Generator.

Appendix C

Tree Structure of the Application

- /
  - /bin
  - /build
  - /ext
    - /mpnn
    - /OBJ-Loader
    - /random
    - /rapid
    - /tinyxml2
  - /include
    - Collision.h
    - Export.h
    - KNN.h
    - Node.h
    - Object.h
    - Parser.h
    - Planning.h
    - RandomPoints.h
    - Tree.h


/src
Main.cpp
Collision.cpp
Export.cpp
KNN.cpp
Node.cpp
Object.cpp
Parser.cpp
Planning.cpp
RandomPoints.cpp
Tree.cpp
task.xml
objects.obj
Makefile

Appendix D

Example of the Input XML File

```
<Planning_task>
    <Name>Testing</Name>
    <Object>obejcts.obj</Object>
    <Planner>RRTStar</Planner>
    <Task_number>1</Task_number>
    <Parameters>
        <iteration_limit>2000</iteration_limit>
        <time_limit>1</time_limit>
        <init_pos_x>0</init_pos_x>
        <init_pos_y>0</init_pos_y>
        <init_pos_z>0</init_pos_z>
        <init_rot_x>0</init_rot_x>
        <init_rot_y>0</init_rot_y>
        <init_rot_z>0</init_rot_z>
        <goal_pos_x>200</goal_pos_x>
        <goal_pos_y>200</goal_pos_y>
        <goal_pos_z>200</goal_pos_z>
        <goal_pos_radius>3</goal_pos_radius>
        <state_space_x>250</state_space_x>
        <state_space_y>250</state_space_y>
        <state_space_z>250</state_space_z>
        <step_size>2</step_size>
        <interpolation_step>0.1</interpolation_step>
        <metric>1</metric>
        <scale_trans>1</scale_trans>
        <scale_rot>1</scale_rot>
        <retraction_step>0.5</retraction_step>
        <retraction_nodes>10</retraction_nodes>
        <retraction_iterations>10</retraction_iterations>
        <gamma>50</gamma>
        <inputs>50</inputs>
        <alpha>10</alpha>
```

```
        <beta>10</beta>
    </Parameters>
</Planning_task>
```

Appendix E

Content of the Attached Disc

The attached disc contains following directories:

- *text* – A PDF file with the thesis.
- *files* – Source files of the motion planner
- *data* – Output data of experiments
- *maps* – 3D models of utilized maps
- *video* – Videos with motion planning tasks

Appendix F

Bibliography

[1] T. Lozano-Perez, "Automatic planning of manipulator transfer movements," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 11, no. 10, pp. 681–698, 1981.

[2] M. Hosek and H. Elmali, "Trajectory planning and motion control strategies for a planar three-degree-of-freedom robotic arm," Nov. 4 2003. US Patent 6,643,563.

[3] H. Chang and T.-Y. Li, "Assembly maintainability study with motion planning," in *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on*, vol. 1, pp. 1012–1019, IEEE, 1995.

[4] D. Nieuwenhuisen, A. Kamphuis, M. Mooijekind, and M. H. Overmars, "Automatic construction of roadmaps for path planning in games," in *International Conference on Computer Games: Artificial Intelligence, Design and Education*, pp. 285–292, 2004.

[5] J.-C. Latombe, "Motion planning: A journey of robots, molecules, digital actors, and other artifacts," *The International Journal of Robotics Research*, vol. 18, no. 11, pp. 1119–1128, 1999.

[6] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments," *Journal of Field Robotics*, vol. 25, no. 11-12, pp. 939–960, 2008.

[7] J. Levinson, J. Askeland, J. Becker, J. Dolson, D. Held, S. Kammel, J. Z. Kolter, D. Langer, O. Pink, V. Pratt, *et al.*, "Towards fully autonomous driving: Systems and algorithms," in *Intelligent Vehicles Symposium (IV), 2011 IEEE*, pp. 163–168, IEEE, 2011.

[8] A. R. Lanfranco, A. E. Castellanos, J. P. Desai, and W. C. Meyers, "Robotic surgery: a current perspective," *Annals of surgery*, vol. 239, no. 1, p. 14, 2004.

[9] E. Glassman, W. A. Hanson, P. Kazanzides, B. D. Mittelstadt, B. L. Musits, H. A. Paul, and R. H. Taylor, "Image-directed robotic system for precise robotic surgery including redundant consistency checking," Feb. 4 1992. US Patent 5,086,401.

[10] N. M. Amato and G. Song, "Using motion planning to study protein folding pathways," *Journal of Computational Biology*, vol. 9, no. 2, pp. 149–168, 2002.

[11] K. Kazerounian, "From mechanisms and robotics to protein conformation and drug design," *Journal of Mechanical Design*, vol. 126, no. 1, pp. 40–45, 2004.

[12] S. M. LaValle, "Rapidly-exploring random trees: A new tool for path planning," 1998.

[13] S. M. LaValle and J. J. Kuffner Jr, "Rapidly-exploring random trees: Progress and prospects," 2000.

[14] L. E. Kavraki, P. Svestka, J.-C. Latombe, and 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.

[15] Parasol, *Algorithms and Applications Group Benchmarks*. Available at `https://parasol.tamu.edu/groups/amatogroup/benchmarks/`.

[16] S. M. LaValle, *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006. Available at http://planning.cs.uiuc.edu/.

[17] J. C. Butcher, "The numerical analysis of ordinary differential equations: Runge-kutta and general linear methods," 1987.

[18] L. E. Kavraki, "Computation of configuration-space obstacles using the fast fourier transform," *IEEE Transactions on Robotics and Automation*, vol. 11, no. 3, pp. 408–413, 1995.

[19] P. Jiménez, F. Thomas, and C. Torras, "3d collision detection: a survey," *Computers & Graphics*, vol. 25, no. 2, pp. 269–285, 2001.

[20] J. T. Klosowski, M. Held, J. S. Mitchell, H. Sowizral, and K. Zikan, "Efficient collision detection using bounding volume hierarchies of k-dops," *IEEE transactions on Visualization and Computer Graphics*, vol. 4, no. 1, pp. 21–36, 1998.

[21] J. H. Reif, "Complexity of the mover's problem and generalizations," in *Foundations of Computer Science, 1979., 20th Annual Symposium on*, pp. 421–427, IEEE, 1979.

[22] J.-P. Laumond, S. Sekhavat, and F. Lamiraux, "Guidelines in nonholonomic motion planning for mobile robots," in *Robot motion planning and control*, pp. 1–53, Springer, 1998.

[23] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[24] I. Streinu, "A combinatorial approach to planar non-colliding robot arm motion planning," in *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on*, pp. 443–453, IEEE, 2000.

[25] N. M. Amato and Y. Wu, "A randomized roadmap method for path and manipulation planning," in *Robotics and Automation, 1996. Proceedings., 1996 IEEE International Conference on*, vol. 1, pp. 113–120, IEEE, 1996.

[26] Y. K. Hwang and N. Ahuja, "A potential field approach to path planning," *IEEE Transactions on Robotics and Automation*, vol. 8, no. 1, pp. 23–32, 1992.

[27] S. S. Ge and Y. J. Cui, "Dynamic motion planning for mobile robots using potential field method," *Autonomous robots*, vol. 13, no. 3, pp. 207–222, 2002.

[28] M. Elbanhawi and M. Simic, "Sampling-based robot motion planning: A review," *Ieee access*, vol. 2, pp. 56–77, 2014.

[29] H. Kaluđer, M. Brezak, and I. Petrović, "A visibility graph based method for path planning in dynamic environments," in *MIPRO, 2011 Proceedings of the 34th International Convention*, pp. 717–721, IEEE, 2011.

[30] E. Welzl, "Constructing the visibility graph for n-line segments in o (n2) time," *Information Processing Letters*, vol. 20, no. 4, pp. 167–171, 1985.

[31] P. Bhattacharya and M. L. Gavrilova, "Roadmap-based path planning-using the voronoi diagram for a clearance-based shortest path," *IEEE Robotics & Automation Magazine*, vol. 15, no. 2, 2008.

[32] C. K. Yap, "Ano (n logn) algorithm for the voronoi diagram of a set of simple curve segments," *Discrete & Computational Geometry*, vol. 2, no. 4, pp. 365–393, 1987.

[33] N. Sleumer and N. Tschichold-Gürmann, "Exact cell decomposition of arrangements used for path planning in robotics," *Technical report/ETH Zürich, Department of Computer Science*, vol. 329, 1999.

[34] C. Cai and S. Ferrari, "Information-driven sensor path planning by approximate cell decomposition," *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, vol. 39, no. 3, pp. 672–689, 2009.

[35] S. Thomas, M. Morales, X. Tang, and N. M. Amato, "Biasing samplers to improve motion planning performance," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 1625–1630, IEEE, 2007.

[36] D. Hsu and Z. Sun, "Adaptively combining multiple sampling strategies for probabilistic roadmap planning," in *Robotics, Automation and Mechatronics, 2004 IEEE Conference on*, vol. 2, pp. 774–779, IEEE, 2004.

[37] S. Karaman and E. Frazzoli, "Optimal kinodynamic motion planning using incremental sampling-based methods," in *Decision and Control (CDC), 2010 49th IEEE Conference on*, pp. 7681–7687, IEEE, 2010.

[38] D. Hsu, R. Kindel, J.-C. Latombe, and S. Rock, "Randomized kinodynamic motion planning with moving obstacles," *The International Journal of Robotics Research*, vol. 21, no. 3, pp. 233–255, 2002.

[39] S. R. Lindemann and S. M. LaValle, "Current issues in sampling-based motion planning," in *Robotics Research. The Eleventh International Symposium*, pp. 36–54, Springer, 2005.

[40] A. Atramentov and S. M. LaValle, "Efficient nearest neighbor searching for motion planning," in *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, vol. 1, pp. 632–637, IEEE, 2002.

[41] S. A. David M. Mount, *ANN: A Library for Approximate Nearest Neighbor Searching*. Available at `http://www.cs.umd.edu/~mount/ANN/`.

[42] S. M. L. Anna Yershova, *MPNN: Nearest Neighbor Library for Motion Planning*. Available at `http://msl.cs.uiuc.edu/~yershova/software/MPNN/MPNN.htm`.

[43] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," *VISAPP (1)*, vol. 2, no. 331-340, p. 2, 2009.

[44] University of North Carolina, *RAPID - Robust and Accurate Polygon Interference Detection*. Available at `http://gamma.cs.unc.edu/OBB/`.

[45] I. Kravtchenko, *OZCollide - Fast, Complete and Free Collision Detection Library*. Available at `www.tsarevitch.org/ozcollide/` (not accessible).

[46] M. Rickert, O. Brock, and A. Knoll, "Balancing exploration and exploitation in motion planning," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 2812–2817, IEEE, 2008.

[47] J. J. Kuffner and S. M. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 2, pp. 995–1001, IEEE, 2000.

[48] W. Wang, X. Xu, Y. Li, J. Song, and H. He, "Triple rrts: An effective method for path planning in narrow passages," *Advanced Robotics*, vol. 24, no. 7, pp. 943–962, 2010.

[49] M. Strandberg, "Augmenting rrt-planners with local trees," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 4, pp. 3258–3262, IEEE, 2004.

[50] L. Zhang and D. Manocha, "An efficient retraction-based rrt planner," in *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pp. 3743–3750, IEEE, 2008.

[51] M. Kalisiak and M. van de Panne, "Rrt-blossom: Rrt with a local flood-fill behavior," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*, pp. 1237–1242, IEEE, 2006.

[52] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," *The international journal of robotics research*, vol. 30, no. 7, pp. 846–894, 2011.

[53] S. Karaman, M. R. Walter, A. Perez, E. Frazzoli, and S. Teller, "Anytime motion planning using the rrt," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pp. 1478–1483, IEEE, 2011.

[54] O. Adiyatov and H. A. Varol, "Rapidly-exploring random tree based memory efficient motion planning," in *Mechatronics and Automation (ICMA), 2013 IEEE International Conference on*, pp. 354–359, IEEE, 2013.

[55] J. D. Gammell, S. S. Srinivasa, and T. D. Barfoot, "Informed rrt*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic," in *Intelligent Robots and Systems (IROS 2014), 2014 IEEE/RSJ International Conference on*, pp. 2997–3004, IEEE, 2014.

[56] I. Noreen, A. Khan, and Z. Habib, "Optimal path planning using rrt* based approaches: a survey and future directions," *Int. J. Adv. Comput. Sci. Appl*, vol. 7, pp. 97–107, 2016.

[57] Robert S., *A C++ OBJ Model Loader that will parse .obj & .mtl Files into Indices, Vertices, Materials, and Mesh Structures*. Available at `https://github.com/Bly7/OBJ-Loader/`.

[58] L. Thomason, *A simple, small and efficient C++ XML parser*. Available at `http://www.grinninglizard.com/tinyxml2/`.

[59] I. Polishchuk, *Random for modern C++ with convenient API*. Available at `https://github.com/effolkronium/random/`.

[60] Stichting Blender Foundation, *Blender*. Available at `https://www.blender.org/`.

[61] MAXON Computer, *Cinema 4D*. Available at `https://www.maxon.net/en/products/cinema-4d/overview/`.