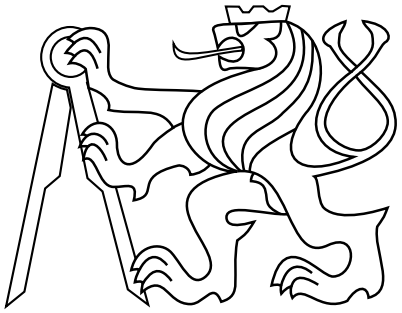
Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Circuit Theory
Multi-robot Systems



Machine learning for fast motion planning

Bachelor's Thesis

Jonáš Kříž

Prague, May 2023

Study programme: Medical Electronics and Bioinformatics

Supervisor: Ing. Vojtěch Vonásek, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Kříž Jonáš** Personal ID number: **499344**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Circuit Theory**

Study program: **Medical Electronics and Bioinformatics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Machine learning for fast motion planning

Bachelor’s thesis title in Czech:

Využití metod strojového učení pro rychlé plánování pohybu

Guidelines:

1. Get familiar with path planning problem [1], focus on path planning for 3D objects. Get familiar with neural networks for regression.
2. Implement a sampling-based method, e.g., RRT, PRM, or one of its variants [2].
3. Implement a method that can sample along a predefined path or an approximate solution [3,4].
4. Design a method to suggest probability of sampling using a neural network (NN). NN will be trained using found solutions, it should provide hints where the searched space should be sampled.
5. Verify the performance of the method from 4) on the dataset (will be provided by the supervisor). Compare the method from 4) against suitable methods from the OMPL benchmark [5].
6. Adapt the method from 4) to protein data in the protein-docking through protein tunnels [6]. The dataset and models of the molecules will be provided by the supervisor. Compare the method 5) against RRT-based search (will be provided by the supervisor).

Bibliography / sources:

1. LaValle, Steven M. Planning algorithms. Cambridge university press, 2006.
2. Elbanhawi, Mohamed, and Milan Simic. 'Sampling-based robot motion planning: A review.' Ieee access 2 (2014): 56-77.
3. J. Denny, R. Sandström, A. Bregger, and N. M. Amato. Dynamic region-biased rapidly-exploring random trees. In Twelfth International Workshop on the Algorithmic Founda-tions of Robotics (WAFR), 2016.
4. V Vonásek, R Pěnička and B Kozlíková. Searching Multiple Approximate Solutions in Configuration Space to Guide Sampling-Based Motion Planning. Journal of Intelligent & Robotic Systems 100:1547-1543, 2020
5. I. A. Sucan, M. Moll and L. E. Kavraki, 'The Open Motion Planning Library,' in IEEE Robotics & Automation Magazine, vol. 19, no. 4, pp. 72-82, Dec. 2012, doi: 10.1109/MRA.2012.2205651.
6. Vavra O, Filipovic J, Plhak J, Bednar D, Marques SM, Brezovsky J, Stourac J, Matyska L, Damborsky J. CaverDock: a molecular docking-based tool to analyse ligand transport through protein tunnels and channels. Bioinformatics. 2019 Dec 1;35(23):4986-4993. doi: 10.1093/bioinformatics/btz386. PMID: 31077297.

Name and workplace of bachelor's thesis supervisor:

Ing. Vojtěch Vonásek, Ph.D. Multi-robot Systems FEE

Name and workplace of second bachelor's thesis supervisor or consultant:

Date of bachelor's thesis assignment: **31.01.2023** Deadline for bachelor thesis submission: **26.05.2023**

Assignment valid until: **22.09.2024**

Ing. Vojtěch Vonásek, Ph.D. Supervisor's signature	doc. Ing. Radoslav Bortel, Ph.D. Head of department's signature	prof. Mgr. Petr Páta, Ph.D. Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgments

I have to express my gratitude to my supervisor Vojtěch Vonásek, for giving me the opportunity to work on this thesis, which I greatly enjoyed. I thank my family for enduring a long period of time when they were not able to see me. Also, I would like to thank Vojtěch Štěpančík for sharing his knowledge about C, which proved to be very useful.

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

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

Abstrakt

Plánování pohybu a strojové učení jsou dvě důležitá témata v dnešním inženýrství a výzkumu. Otázkou, kterou se tato práce pokouší zodpovědět, je, zda kombinace těchto dvou disciplín přináší uspokojivé výsledky. Bylo navrženo šest nových algoritmů plánování pohybu. Tři z nich využívaly samotné strojové učení nebo techniky související se strojovým učením k urychlení procesu plánování pohybu. Motivace ke zrychlení pramení z problémů, se kterými se potýká plánování pohybu při řešení úloh s velkým počtem stupňů volnosti. Například v oblastech, jako je vývoj léků, je dokování proteinů základní disciplínou, která vyžaduje použití plánování pohybu v prostředích s vysokým počtem stupňů volnosti. Proto je výzkum zaměřený na urychlení metod plánování pohybu klíčový.

Klíčová slova plánování pohybu, storojové učení, docking proteinů

Abstract

Motion planning and machine learning are two important topics in today's engineering and research. The question this thesis attempts to answer is whether the combination of these two disciplines yields satisfactory results. Six novel motion planning algorithms were proposed. Three of them utilized machine learning itself or machine learning-related techniques to speed up the motion planning process. The motivation for the speed-up stems from the problems motion planning faces when solving a high number of degrees of freedom tasks. For example in areas like drug design, protein docking is an essential discipline, which requires the use of motion planning in environments with a high number of degrees of freedom. Therefore the research on speeding up motion planning methods is crucial.

Keywords Motion planning, machine learning, protein docking

Abbreviations

1BN7 Haloalkane Dehalogenase from a Rhodococcus species

1MAH Fasciculin2-Mouse Acetylcholinesterase Complex

1TCC The sequence, crystal structure determination and refinement of two crystal forms of lipase B from Candida antarctica

DOF Degrees Of Freedom

NN Neural Network

MD Molecular Dynamics

ML Machine Learning

MP Motion Planning

OMPL Open Motion Planning Library

PSO Particle Swarm Optimization

PWE Parzen windows estimation

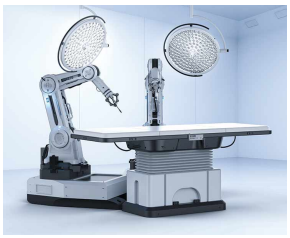
RRT Rapidly-exploring Random Tree

Contents

Chapter 1

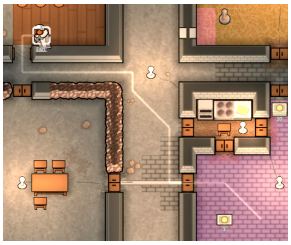
Introduction

The purpose of this chapter is to introduce the reader to the content of this thesis. The thesis will delve into the topic of Motion Planning (MP), which is a very important topic in many fields. MP deals with finding a path for an object in an environment, such that the object does not collide with obstacles. An example of these fields is robotics, chemistry or even the game development industry (Figure 1.1). In robotics, MP is essential for autonomous car driving, or for medical robots performing surgical operations [1]. In computational biochemistry is MP used in the problem of protein docking (introduced in Section 2.3). The game development industry requires MP for the representation of the movement of objects in the environment.



(a) Medical engineering

(b) Biochemistry



(c) Game development

Figure 1.1: An illustration, of fields utilizing Motion planning (surgical robotic hand [2], protein docking, videogame RimWorld [3]).

Together with Motion planning will be also examined the role of the Machine Learning (ML) in the context of MP. Experiments will be done, to shed more light on the question, of whether the ML is actually a useful tool for MP. Specifically, it will be tested, whether it is possible to speed up the MP by utilizing ML-related methods. The ML-related methods, that will be examined are the Particle Swarm Optimization (PSO), the Parzen windows estimation (PWE), and most importantly the Neural Network (NN).

1.1 Motion planning

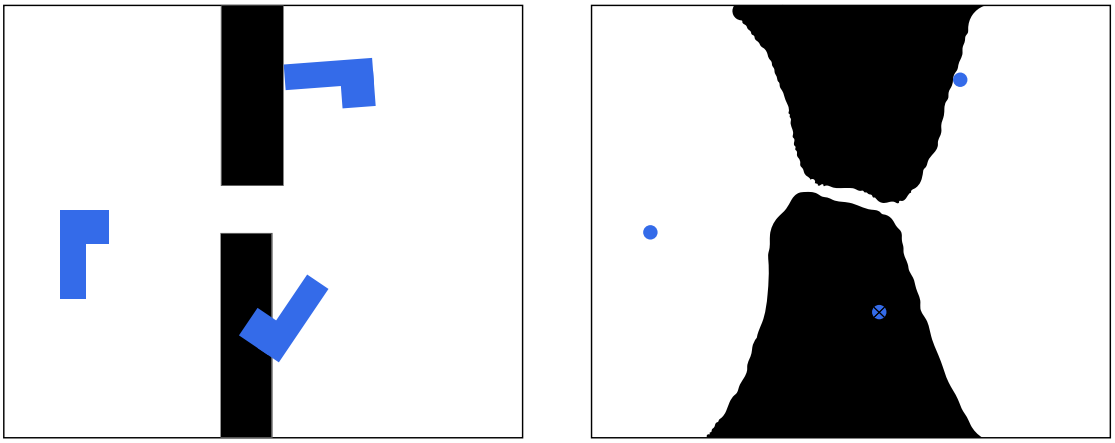
First of all, it is necessary to say, what is Motion planning. Motion planning is the discipline of finding a way to move an object into a desired location while avoiding collisions. With the term location, we do not have to restrict ourselves only to spatial location. A desired location can also be a specific configuration of the object.

The objects can have a various number of Degrees Of Freedom (DOF). For example, moving a table on the kitchen floor has two DOF. One degree of freedom for each possible direction of movement. If we allow the table to be also rotated, we introduce an additional degree of freedom. Another example is picking up a fork from the floor and placing it on the

table. The fork has three DOF, each for movement in one of the three spatial axes. Allowing the fork to rotate around each axis results in the addition of three more DOF. That leads to the conclusion, that to manipulate a solid object freely in space leads to six DOF for object (three for translation, three for rotation).

More degrees of freedom can emerge by adding more characteristics to the object. The object can, for example, represent a molecular structure. It can happen, that molecule can do more than only freely move in space. The molecule may also be able to bend in certain "joints", each joint adding a new DOF. With the increasing number of degrees of freedom, the MP task gets harder to solve.

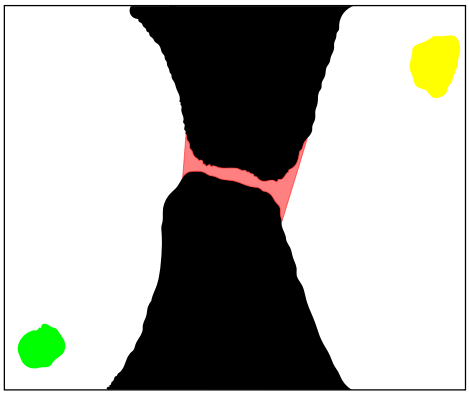
To give us the ability to represent all configurations of the object is constructed configuration space (c-space) (visualized in Figure 1.2). Every axis in this space corresponds to one degree of freedom of the object. Solving the MP task is then equivalent to finding a path for a point between the starting configuration and the desired final configuration in the configuration space.



(a) Objects in workspace **(b)** Configurations in a projection of c-space

Figure 1.2: The Image (a) depicts an object in three different positions (blue) in the two-dimensional space with obstacles (black). The Image (b) shows configurations (blue) in c-space that correspond to the positions of the objects in the previous image. All the configurations representing collision are marked black. On the Image (b) is not in fact the c-space, but its projection, since in this case, the c-space would be three-dimensional.

Currently, the most used family of methods for finding the path are the sampling-based methods [4]. Sampling-based methods utilize the principle of randomly taking points from the c-space and using them as vertices of a graph. Finding a path from the vertex representing the start, to the vertex representing the goal is then a much simpler task than searching the whole c-space. Still, this approach has its limitations. Increasing the number of DOF causes the probability of obtaining random samples from regions of c-space that are crucial for finding the path to decrease. More time is then required, to make sure, that samples from these important regions were obtained. Which leads to slowing down the algorithm. The small region of the c-space that is this important for the success of the task is called a *narrow passage* (illustrated in Figure 1.3).



(a) A Narrow passage

Figure 1.3: A region of the c-space crucial, for finding the path from the start (green) to the goal (yellow). In order to find any path, samples from this crucial region are necessary.

1.2 Motivation

As it was said in the previous subsection, a higher number of DOF causes the MP task to be harder to solve. That causes an increase in the run-time of sampling-based methods. The prolonged timespan of finding the solution negatively affects MP tasks featuring molecular structures. Molecules in MP are represented as strings of many spheres, and each can function as a joint, which leads to many DOF [5].

Several solutions for speeding up the sampling-based methods come up. One of them is to influence the sampling itself. The original sampling-based methods use uniform sampling of the whole c-space. More frequent sampling in the regions of c-space that are expected to contain the solution (final path) should lead to speeding up the search. This is the approach this thesis aims to use.

1.3 Goals

This thesis has two main goals. The first one is, to speed up the sampling-based methods to increase their usefulness for MP tasks with a higher number of degrees of freedom. The motivation for that is the high DOF nature of the motion planning for molecules. Therefore whether this goal was achieved will be verified on multiple MP tasks with molecular objects.

The second goal is to verify how useful ML-related motion planning methods can be in achieving the first goal. That will be verified by benchmarking the methods in three different environments, with varying complexity.

The assumption is, that with the use of a good understanding of the task, the ML-related motion planning methods can be simplified. The simplified versions are expected to outperform the ML-related methods because the ML-related methods are approximating some desired behavior, whereas the simplified methods can have this behavior directly implemented. Because of that, the simplified methods should save a lot of computation time.

1.4 Outline

Chapter 2, Problem Definition aims to provide clarity, on what all further used words and concepts are intended to mean. First of all, it will explain the motion planning problem in more depth, and with the use of more technical terms. Secondly, a machine learning definition relevant to this work will be provided. Lastly, the protein docking problem will be introduced to the reader.

Selection of Related work will be shown in the chapter of the same name, introducing relevant methods for motion planning. For example, Probability Roadmaps (PRM) and most importantly the original Rapidly-exploring Random Trees (RRT) algorithm, on which this work heavily relies on.

The chapter Proposed solutions will introduce three pairs of novel RRT-based algorithms. Each pair will be composed of one machine-learning related solution, and another non-machine-learning solution, derived from the previous one. Firstly, the implementation of RRT used in this work will be specified, together with tools, further used in proposed algorithms.

All results of algorithm benchmarking experiments will be presented in chapter Results. The first section of this chapter will contain the results of benchmarking all six proposed algorithms with another ten algorithms implemented in the Open Motion Planning Library (OMPL). The second section will test proposed algorithms on motion planning for protein structures.

Lastly, in chapter Conclusion, the performance of proposed algorithms will be evaluated, and conclusions will be made about the helpfulness of machine learning in motion planning. It will be elaborated, on which changes to the RRT algorithm caused the most improvement, and what are their advantages and disadvantages.

Chapter 2

Problem definition

2.1 Motion planning

To make explanations in the following chapters clear, it is useful, to define several concepts of motion planning, as well, as concepts introduced and used in this thesis. The notation used in this thesis is based on the well-established notation from the book Planning Algorithms by Steven M. LaValle [4].

Motion planning is finding a sequence of motions, that will allow an *object* to reach a goal configuration from the starting configuration without colliding with obstacles. In our case, there are two kinds of movable objects. One is a three-dimensional rigid body, with the ability to rotate around all three spatial axes. The second kind is a molecule represented by a hard-sphere model, with the ability to use the connection between these spheres as joints.

Each specific placement and rotation (and the joint angles) of the object is defined as a specific $configuration$ of the object. Configurations are denoted with the symbol q. A set of all possible configurations composes the configuration space $\mathcal{C}$. In this thesis, $\mathcal{C}$ is a continuous metric space, where each point represents one possible configuration of our object.

The space $\mathcal{C}$ contains several regions, that are very useful for our problem. First of all the set of all goal configurations is $\mathcal{C}_{goal}$. One element of the set $\mathcal{C}_{goal}$ is q_{goal}. Another important subset $\mathcal{C}_{init}$ is a set of all initial configurations, in our case containing only one element q_{init}. The last two of for us important subsets is $\mathcal{C}_{obs}$ and its complement $\mathcal{C}_{free}$. The set $\mathcal{C}_{obs}$ is an open set of all the configurations of the object, that cause the object to collide with obstacles (or in the case of molecules surpasses certain energetic threshold [6]). The set $\mathcal{C}_{free}$ is a set of all collision-free configurations.

With these concepts defined, we can reformulate, what motion planning task is as: The task of finding sequence of configurations $\mathcal{P} = \{q_1, q_2, ..., q_n\}$, while these three conditions must hold: $q_i \in \mathcal{C}_{free}$, $q_1 = q_{init}$, $q_n \in \mathcal{C}_{goal}$. This task is an NP-hard problem to solve [4]. The sequence $\mathcal{P}$ will be called *path*, and it serves as a sequence of configurations, that can be followed to move the object from the initial configuration to the desired configuration without collision.

A cubic body, that is significantly smaller, than the object will be later referred to, as a *probe*. It will serve to find approximations of the path. Another concept is *impact points*, which are elements of the boundary of the set $\mathcal{C}_{obs}$. Since the set $\mathcal{C}_{obs}$is an open set, the impact points lie in $\mathcal{C}_{free}$.

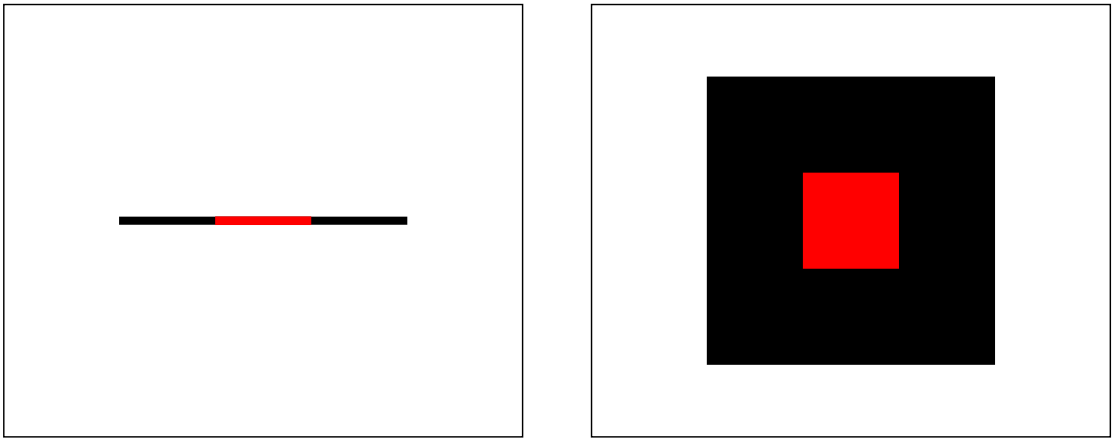
2.1.1 Random sampling methods

One kind of methods designed to solve the problem defined in this section (2.1) are the random sampling methods [5, 7]. The principle of these methods is to take several randomly selected elements $q_{rand} \in \mathcal{C}$ and used them to build a roadmap (graph). By approximating the $\mathcal{C}_{free}$region with the roadmap, the finding of a path in continuous c-space is changed to finding a path in the graph. The sampling is, if not mentioned otherwise a uniform sampling. This approach has a possible disadvantage caused by the randomness of the sampling.

Narrow passages

The disadvantage is the existence of narrow passages (illustrated in the figure 1.3). The narrow passages cause two following closely related problems. One problem is, that the narrow passages must be sampled, in order to find the path. The second problem is, that the probability of obtaining a sample from a narrow passage is low.

That is because the probability to randomly obtain a sample from a specific region of space is proportional to the relative volume of the region. That means, that if the region we wish to sample forms 50% of the whole space, there is a 50% chance to obtain a random sample from that region. Because the narrow passages are generally small (narrow), the probability of obtaining a sample from inside of them is therefore also small. The problem of sampling narrow passages is usually even more challenging with every additional degree of freedom, which is demonstrated in the following Figure 2.1.

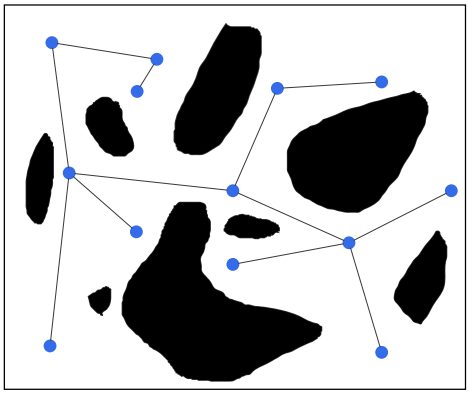


(a) Interval in 1D **(b)** Interval in 2D

Figure 2.1: With growing dimensions, an interval (red) becomes relatively smaller portion of the sampled space (black). In 1D case, the interval made up $33,\overline{33}\%$ of the whole space. Whereas in the 2D case, the interval takes up only $11.\overline{11}\%$ of the sampled space.

Graph construction

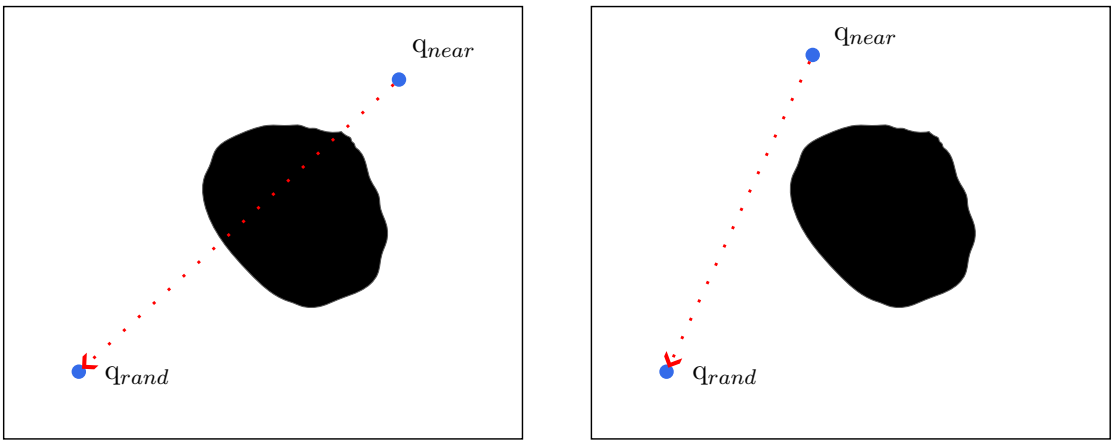
When introducing the random sampling methods, it is also important to explain how the graphs are being constructed from the sampled c-space (an example of a graph in the c-space in Figure 2.2). The usual practice is to use the obtained samples q as the vertices of the graph. The construction of the edges of the graph is more specific for each method. The existence of an edge between two vertices in the graph signifies, that there exists a path between the two configurations these two vertices represent.



(a) Example of graph in the c-space

Figure 2.2: Example of a graph, where vertices represent sampled configurations from the c-space (blue) and edges (gray) represent collision-free paths between the configurations. The obstacles are black.

The existence of such paths is checked by a tool called a *local planner*. The local planner is generally a simple path planner that is very fast to execute. One of the most commonly used local planners is the "straight-line", which creates a line between two points and then checks for a specified number of configurations in the line, whether they are in $\mathcal{C}_{free}$. If all the checked configurations lie in $\mathcal{C}_{free}$ the local planner indicates, that there is a collision-free path between the two points (demonstrated in Figure 2.3).



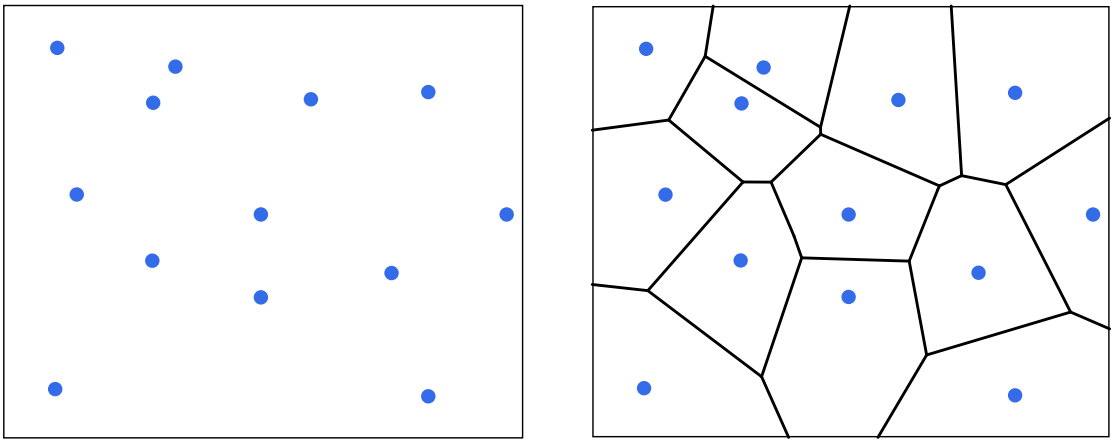
(a) Non-connectable points

(b) Connectable points

Figure 2.3: The local planner checks path between a newly sampled configuration q_{rand} and a nearest already sampled configuration q_{near}. The points colored red get checked, whether any of them lies in $\mathcal{C}_{obs}$ (black). The procedure is done in the direction of the arrow.

Voronoi diagrams

The probability, that a given vertex in the graph will be considered a neighbor for a new random sample can be nicely illustrated with a Voronoi diagram [8, 9]. The Voronoi diagram (or Voronoi tesselation for more than two-dimensional spaces) is a diagram, that splits a given space (in our case the c-space) into a set of convex non-overlapping cells. Each cell contains exactly one of our already sampled points (the vertices of the graph). Every point in a cell is closer to the vertice in that cell than to any other vertice in any other cell (Figure 2.4).



(a) The sampled points **(b)** Voronoi diagram of the sampled points

Figure 2.4: Already sampled points (blue), or also the vertices of the constructed graph, and the Voronoi diagram they generate.

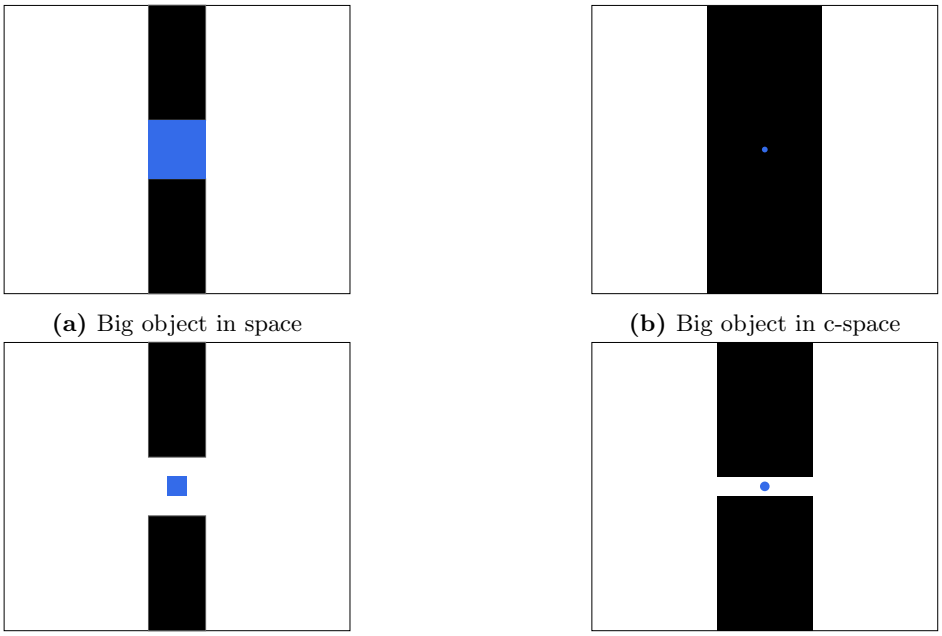
If we normalize the c-space (scale each axis with a scalar, such that the final volume of the space is 1), the volume of each cell equals the probability, with which a new uniform sample will be connected to the vertice present inside that cell (if $\mathcal{C}_{obs} = \emptyset$) [8]. For example, the method Adaptive dynamic-domain RRT [8] (Section 3.4) is based on using this knowledge to its benefit.

Relation between the object and the c-space

The properties of the c-space are not influenced only by the number of DOF of the object. The spatial parameters of the object also heavily influence the c-space, especially the $\mathcal{C}_{obs}$ and $\mathcal{C}_{free}$ regions. The shape of the object heavily influences the dimensions of the c-space related to rotation. The more complex the shape of the object is, the narrower can the narrow passages in these dimensions.

Another spatial property, that heavily influences what point of the c-space lies in $\mathcal{C}_{obs}$ or $\mathcal{C}_{free}$ is the size of the object. The intuition is, that the higher volume the object has, the fewer regions it fits in (depicted in Figure 2.5).

That means that the MP task is easier for smaller and less complicated objects since their $\mathcal{C}_{obs}$ populates smaller portions of the c-space than in the case of big and complicated objects. This property is utilized in most of the algorithms proposed in this thesis (Section 5.1.2).



(a) Big object in space

(b) Big object in c-space

(c) Small object in space

(d) Small object in c-space

Figure 2.5: Demonstration on how the volume of the object (blue) influences the size of $\mathcal{C}_{obs}$ (black). Since the big object fully fills the narrow passage, the narrow passage in the c-space becomes only a line, which has an extremely low probability, to get sampled. For simplicity, the objects are not allowed to rotate in this scenario, so the c-space is only two-dimensional.

2.2 Machine learning

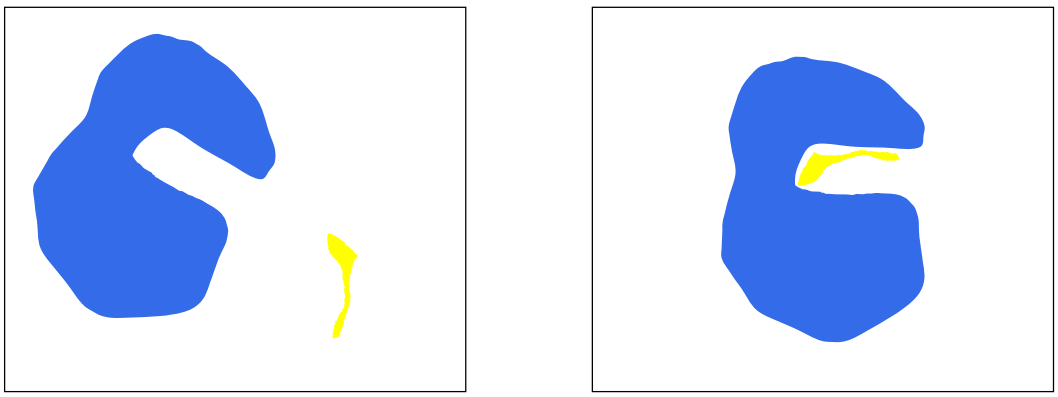
In this thesis will be used the following definition from the Oxford English Dictionary: "(Machine learning is) the use and development of computer systems that are able to learn and adapt without following explicit instructions, by using algorithms and statistical models to analyze and draw inferences from patterns in data." [10]. In this thesis, three attempts at harnessing algorithms satisfying this definition will be presented. Namely using PWE, PSO, and finally, one very relevant in the current time NN.

2.3 Protein docking

Because of the increase in the computational power of computers, it is increasingly viable, to analyze chemical reactions with computers [11]. One of these now available methods is Protein docking [11]. Protein docking is a discipline of finding the "best-fitting" spatial[1] configuration of two molecules (see Figure 2.6). The fitness of the configuration is evaluated by a *scoring function* [6, 12, 13].

One of the molecules (*receptor*) is a protein and is usually bigger than the other one. The other molecule is called a *ligand* and it can be another smaller protein or any other smaller molecular structure. Together these two molecules form a *complex*. The sought for configuration of these two molecules is a position, where they spatially "best fit together" (illustrated in Figure 2.6) [14].

Protein docking is useful for example in designing new drugs [11], where the ligand can be the designed drug, and the receptor a protein, which is supposed to receive the drug. Via the protein docking, it can be checked, whether the reception of the drug is physically possible. Since it is complicated to carry out these experiments in the amounts and quality required in real experiments, computing simulations are the most viable option [15]. One option for said simulations is to thoroughly simulate all the atoms with Molecular Dynamics (MD) [16]. A downside of this approach is the huge computational time required to carry out even a nanosecond of the simulation.

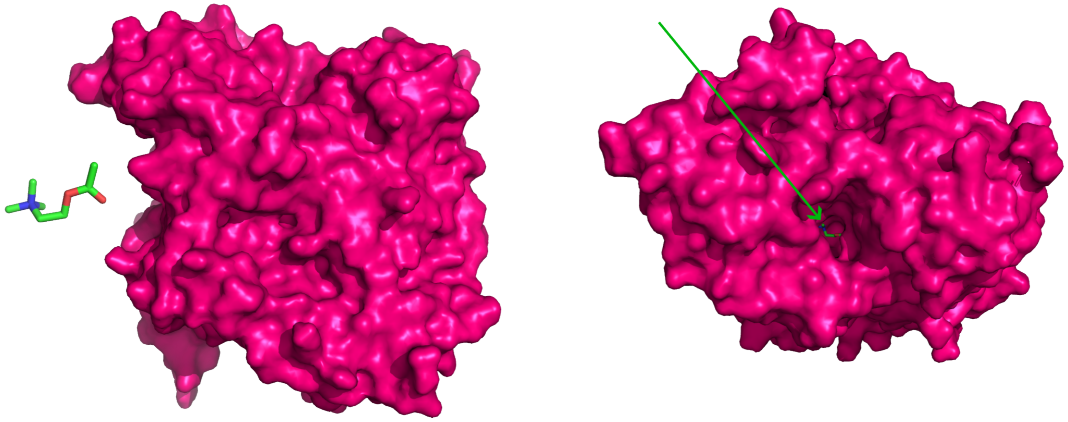


(a) The receptor and the ligand **(b)** The best fitting position

Figure 2.6: Protein docking. The receptor is blue and the ligand is yellow.

[1] The distances in the molecular environment are measured in Ångström (Å). One Ångström equals 1×10^{-10} meters.

There are several approaches to this problem. One of them is through shape complementarity [17]. Usually, these methods represent the molecules with their surfaces in space and use optimization algorithms, to find a configuration, where they geometrically lock (dock) into each other. Another and more recent approach is, to carry out the process of docking the proteins together (Figure 2.7). For this MP algorithms are used [18, 19, 20]. Because on the molecular level, the collision of some configurations does not have to necessarily be given only by a physical overlap, the scoring functions [6] are used to evaluate, whether a configuration is energetically possible, which is another possible collision-inducing factor. Two of the proposed methods (from Section: 5.4) in this thesis will be used, to solve this task.



(a) The ligand outside the receptor **(b)** The ligand docked in the receptor

Figure 2.7: Graphical demonstration of protein docking in PyMol [21]. The protein (pink) is visualized using its solvent surface. The ligand is shown in green color.

The molecules in the MP are represented by a hard-sphere model [22, 23]. The hard-sphere model is a model used to model particles in for example fluid dynamics. The particles are impenetrable spheres, that can not overlap in space. These spheres in our case represent the atoms in the molecules. The connections between the spheres can function as joints. Only the ligands can have joints in this thesis.

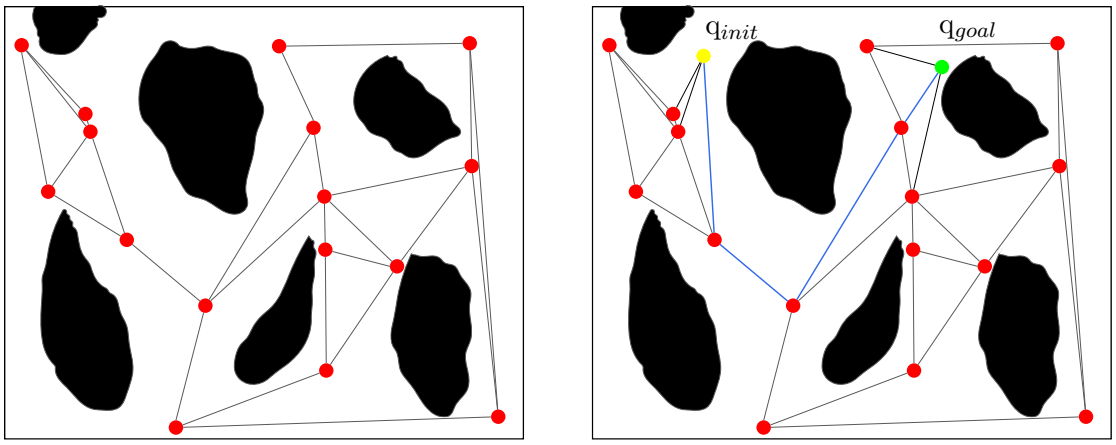
Chapter 3

Related work

Speeding up motion planning methods is an important topic, in which many other works are interested. This chapter presents several methods designed to enhance the performance of MP algorithms. All methods in this chapter are based on the sampling-based principle since that is currently the most successful approach to motion planning of many-DOF objects. The first two introduced methods are the basis of many other MP methods, including the rest of the methods in this chapter and also the methods in the chapter Proposed solutions.

3.1 Probabilistic Roadmaps

The first notable random sampling method is called Probabilistic Roadmaps (PRM) [5]. The PRM algorithm consists of two phases, a *learning phase* (Algorithm 1), and a *query phase* (Algorithm 2). The learning phase is run first and constructs a graph from random configurations in the c-space. The query phase then finds paths between two specified configurations (vertices) in the graph (illustrated in Figure 3.1). One of the advantages of this algorithm is, that for each path query the learning phase does not have to be run again, because the graph it produces is reusable if the environment or the object does not change.



(a) Generated Graph **(b)** Found path

Figure 3.1: An illustration of how the found graph in c-space looks (also called a *roadmap*), and how it is used, to find the path between q_{init} and q_{goal}.

At the beginning of the learning phase, the algorithm finds a specified number of configurations from $\mathcal{C}_{free}$. That is done by the uniform random sampling of the c-space. Then, for each found configuration, the algorithm attempts to connect it, with a set of its neighboring configurations. The set of neighbors can be constructed in multiple ways. One way is to take a specified number of closest configurations, another is to take all the configurations under a certain distance. The connection between configurations is checked by a local planner (introduced in Section 2.1.1). This way, a graph can be constructed. The vertices of the graph represent the sampled configurations. The edges of the graph represent a successful connection between any two neighboring vertices by the local planner.

In the learning phase, two more vertices are added and connected with their neighbors. One stands for the initial configuration q_{init} and the second for the goal configuration q_{goal}. Then any graph-searching algorithm can be employed to find the path between the two nodes. For example the Dijkstra algorithm [5, 24].

Algorithm 1: PRM learning phase

Input: K = maximal number of graph vertices, $V = \emptyset$, $E = \emptyset$
Output: V, E

```
for k ∈ 1:K do
    point ← c ∈ C_free;
    V ← V ∪ c;
for point ∈ V do
    for neighbor ∈ neighbors(point) do
        if connects(neighbor, point) then
            E ← E ∪ {neighbor, point};
```

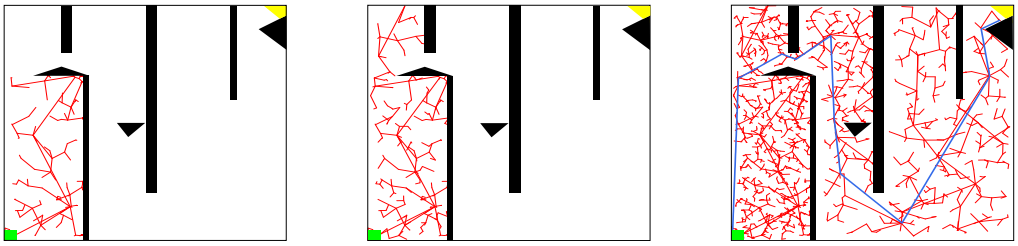
Algorithm 2: PRM query phase

Input: V, E, q_{init} = initial state, q_{goal} = goal state
Output: *path*

```
V ← V ∪ q_init;
V ← V ∪ q_goal;
for neighbor ∈ neighbors(q_init) do
    E ← E ∪ {neighbor, q_init};      // if collision-free connection is possible
for neighbor ∈ neighbors(q_goal) do
    E ← E ∪ {neighbor, q_goal};      // if collision-free connection is possible
G ← graph(V, E);
path ← find_path_in_graph(q_init, q_goal, G);
```

3.2 RRT

One of the most popular path planning methods, especially in robotics is Rapidly-exploring Random Tree (RRT). It was introduced in 1998 by Steven M. LaValle and James J. Kuffner Jr [7]. This method iteratively builds a tree graph in the c-space over a specified number of iterations (demonstrated Figure 3.2). In each iteration, the c-space is uniformly sampled (Algorithm 3). The sample q_{rand} is then connected to the closest node of the tree if the connection is possible. Whether the connection is possible is checked by a local planner (Section: 2.1.1). If we obtain a sample, that lies in $\mathcal{C}_{goal}$ and it gets successfully connected to the tree, the search ends. The resulting tree can be traversed from the last added node to the root, which yields the desired path.



(a) 500 iterations **(b)** 1 000 iterations **(c)** 4 000 iterations

Figure 3.2: An illustration of RRT's growth over iterations.

The function $tree.find\_nearest\_neighbor()$ takes any state from the configuration space and returns a node from the tree. The returned node represents the state, with the smallest distance in configuration space from the input state, out of all nodes in the tree. The nearest-neighbor search is typically realized using KD-tree data structure [25], which yields the $O(log(n))$ complexity, where n is the number of nodes in the KD-tree.

The function $connect()$, indicates, whether a local planner was able to connect two states without colliding. Usually, the local planner is connecting the two states with a straight line segment and checks, if any of the points on the line lies in $\mathcal{C}_{obs}$ (see Figure: 2.3).

Algorithm 3: RRT

Input: K = maximal number of steps, q_{init} = initial position, q_{goal} = goal position
Output: T

```
1 T.root ← q_init;                                   // initialize the search tree
2 for k ∈ 1 : K do
3     q_rand ← c ∈ C;                                // uniform random sample
4     q_near ← T.find_nearest_neighbour(q_rand);
5     if connect(q_near, q_rand) then
6         T.append(q_near, q_rand);
7         if q_rand ∈ C_goal then
8             return T;
```

3.3 Bidirectional RRT

Bidirectional RRT (Algorithm 4) uses two trees to explore the configuration space. One of the trees grows from the q_{init} and the second one has the root in the q_{goal}. This approach was first introduced in the year 2000 under the name RRT-connect [26] but over time the name Bidirectional RRT, or shortly Bi-RRT caught on [27].

Thanks to growing two trees at once, the samples that could not be connected to one tree may be still connected to the other tree. When such a sample is found, that is possible to connect it to both of the trees, in each tree is found a path from its root to that sample. Then the two found paths are connected, resulting in one path leading from the root of one tree into the root of the second one. The resulting path then begins in the q_{init} and ends in the q_{goal} and becomes the returned solution of the MP problem.

Algorithm 4: Bi-RRT

Input: K = maximal number of steps, q_{init} = initial position, q_{goal} = goal position
Output: *path*

```
1  T1.root ← q_init;                                   // initialize a search tree
2  T2.root ← q_goal;                                   // initialize a search tree
3  for k ∈ 1 : K do
4      q_rand ← c ∈ C;
5      q_near1 ← T1.find_nearest_neighbour(q_rand);
6      q_near2 ← T2.find_nearest_neighbour(q_rand);
7      if connect(q_near1, q_rand) then
8          T1.append(q_near1, q_rand);
9          connects1 ← True;
10     if connect(q_near2, q_rand) then
11         T2.append(q_near2, q_rand);
12         connects2 ← True;
13     if connects1 and connects2 then
14         path1 ← path(T1.root, q_rand);
15         path2 ← path(q_rand, T2.root);
16         path ← path1 + path2;
17         return path;
```

3.4 Adaptive DD_RRT

The full name of this algorithm is Adaptive dynamic-domain RRT [8]. The underlying principle is the same as in the original RRT (Section 3.2). However, in DD_RRT the Voronoi diagram [28] does not fully determine, which node will be the new sample q_{rand} connected to. Each node has also a dynamic domain, which the q_{rand} must lie in, in order to be connected to the node (illustrated in Figure 3.3). For this method, the dynamic domain is adaptive, which means, the size of the domain adapts while the algorithm executes. When a node has been successfully connected with q_{rand}, the radius of its domain grows. On the other hand, with each unsuccessful (colliding) connection the domain of the node shrinks (Algorithm 5). This way, nodes that are harder to get connected to have a smaller chance, to be considered

as q_{near} to a new q_{rand}. A smaller number of calling local planner (Subsection 2.1.1) queries should be achieved this way.

(a) RRT Voronoi diagram **(b)** Adaptive DD_RRT Voronoi diagram

Figure 3.3: Influence on sampling by the Adaptive DD_RRT method. The brown area shows which samples will be considered for attempted connection. Images courtesy of [8].

Algorithm 5: Adaptive DD_RRT

Input: K = maximal number of steps, q_{init} = initial position, q_{goal} = goal position, α = numerical parameter chosen from interval [0,1], R = initial radius
Output: T

$T.root \leftarrow q_{init}$; // initialize the search tree
$T.root.radius \leftarrow \infty$;
for $k \in 1 : K$ **do**
  $q_{rand} \leftarrow c \in \mathcal{C}$;
  $q_{near} \leftarrow T.find\_nearest\_neighbour(q_{rand})$;
  if *not* $connect(q_{near}, q_{rand})$ **then**
    if $q_{near}.radius = \infty$ **then**
      $q_{near}.radius \leftarrow R$;
    else
      $q_{near}.radius \leftarrow q_{near}.radius \cdot (1 - \alpha)$; // shrinking of dynamic domain
  else
    $tree.append(q_{near}, q_{rand})$;
    $q_{new}.radius \leftarrow \infty$;
    if $q_{near}.radius \neq \infty$ **then**
      $q_{near}.radius \leftarrow q_{near}.radius \cdot (1 + \alpha)$; // growing of dynamic domain
    if $q_{rand} \in \mathcal{C}_{goal}$ **then**
      return T;

3.5 Guided RRT

A family of RRT algorithms uses already obtained knowledge about certain environment to lead the sampling. Instead of sampling uniformly, the sampling is more dense around some leading path. These leading paths can be obtained in several ways. One way is to scale down the object, find the path (for the small object) and then sample along the path for the scaled-down object [29]. Another possible approach is to make a Voronoi diagram (Subsection 2.1.1)

of obstacles and then use the edges of the cells as guidance (sample with higher density around them) [30]. Some works implement a library of already found paths and then they use the best-fitting path from the library for the guidance [31]. It is also possible for a human to provide some insights and help guide the sampling [32]. Most of the methods proposed in this thesis fall into this category of guided methods.

An example of one guided method can be the RRT-Path [30] (Algorithm 6). This algorithm can use a path obtained in any of the above-mentioned manners. The algorithm samples in proximity of a configuration in the path and when the tree reaches the configuration, the sampling moves along to another configuration further along the path.

Algorithm 6: RRT-Path

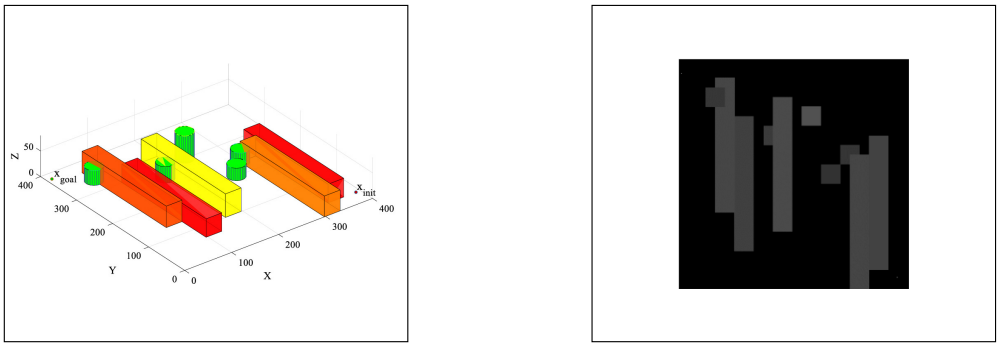
Input: K = maximal number of steps, q_{init} = initial position, q_{goal} = goal position, P = guiding path, D = distances from guiding path points, $goal_{tmp} = P[1]$
Output: T

$T.root \leftarrow q_{init}$; // initialize the search tree
for $k \in 1 : K$ **do**
  if *distance*($goal_{tmp}$, T) $<$ *threshold* **then**
    $goal_{tmp} \leftarrow p \in P$; // closest not yet reached point
  if k *mod* $goal\_bias_{tmp} \neq 0$ **then**
    $q_{rand} \leftarrow goal_{tmp}$; // guided sample
  else
    $q_{rand} \leftarrow c \in \mathcal{C}$; // uniform random sample
  $q_{near} \leftarrow tree.find\_nearest\_neighbour(q_{rand})$;
  if *not connect*(q_{near}, q_{rand}) **then**
    continue;
  else
    tree.append(q_{near}, q_{rand});
    if $q_{rand} \in \mathcal{C}_{goal}$ **then**
      return T;
  $D \leftarrow update(D)$;

3.6 BiLSTM-PSO-GDRRT*

Three intertwined methods (GDRRT*, PSO-GDRRT*, and BiLSTM-PSO-GDRRT*) are presented in the publication [33]. The third method is of interest because it utilizes a neural network to find a guiding path. The first two methods are used, to generate the training dataset for the final algorithm. The first algorithm GDRRT* is used, to find a path in many randomly generated environments. The found paths are then post-processed by the second algorithm PSO-GDRRT*. Then a NN of architecture BiLSTM [34] is trained using the post-processed paths and floor projections of the environment (shown in Figure 3.4). The network is trained so that for a projection of an environment, it is able to return an optimal path from the start to the goal in the projection. The third method BiLSTM-PSO-GDRRT* then utilizes the trained NN to obtain a guiding path before the search begins.

Since one of the proposed methods in this thesis also utilizes a neural network (Subsection 5.4.1) it is appropriate to make a deeper analysis of this work [33], and also a comparison of these two methods (the BiLSTM-PSO-GDRRT* and the proposed NN-RRT (Algorithm 13)).

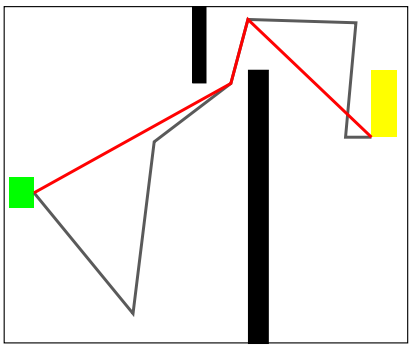


(a) Environment **(b)** Projection

Figure 3.4: One environment used in the [33] and its projection which serves as an input to the NN. Images courtesy of [33].

The first algorithm in [33] is GDRRT* (Goal distance RRT*), an enhancement of RRT*[1]. The method accepts new samples not only if it is possible to connect them to the search tree, but also if their distance from the closest node of the tree is lower than a certain threshold. The threshold is influenced by distance from the goal.

The second algorithm of [33] PSO-GDRRT* is using the algorithm PSO (described in Section 4.1), to optimize the path found by the above-mentioned method. In this context, optimizing the path means shortening the length of the path and omitting any redundant nodes (illustrated in Figure 3.5).



(a) Path optimization

Figure 3.5: A path from the $\mathcal{C}_{init}$ (green) to $\mathcal{C}_{goal}$ (yellow) before optimization (gray) and after optimization (red).

The final implementation of the publication [33] utilizes the trained neural network (Section 4.3) (architecture BiLSTM [34]) to provide a guiding path. The network is trained with the optimized paths obtained by running the PSO-GDRRT*. The authors feed the network with floor projections of the environment as input and train the network to return

[1]RRT* [35, 36] is a modification of RRT, that finds a path that converges towards the optimal path (Figure 3.5).

an optimized path for that environment (Figure 3.4). These paths are then used as a guiding path to find the final solution.

3.6.1 Analysis

The final algorithm BiLSTM-PSO-GDRRT* is fast to find the solution (see Table 3.1) and in addition, it finds the optimal path (RRT-based algorithms in general do not find the optimal path). A downside is that for every new type of environment, the first two (slow) methods have to be run many times to generate a sufficiently big dataset for training NN. Afterwards, the network has to be trained. After all this time the RRT* (a slower version of RRT that finds the optimal path) would most certainly have already found the path many times. Another big downside is the floor projections the network obtains as input. In this projection, a whole dimension of information about the environment is lost. It is not unusual to have a narrow passage in the workspace, which would with this projection completely disappear and the network could never suggest a path passing through that passage (illustrated in Figure 3.6). This could lead even to the impossibility of solving certain tasks, where the solution in fact exists. Also, the act of optimizing the path with the PSO algorithm can in some cases be unnecessary, as for some methods, the guiding path can function better when not optimal (for explanation see Appendix A.3.1).

(a) Environment **(b)** Projection

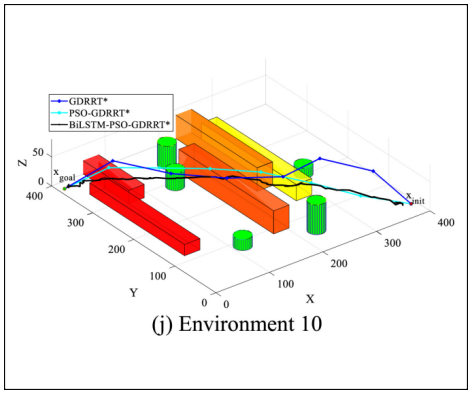
Figure 3.6: The blue path from the start (green) to the goal (yellow), which uses a narrow passage that disappears with top-down projection. The blue path would be impossible to find in the projection. Instead, the red path which is not optimal would have to be used. If the obstacle was even longer, it would be impossible to find a path in the projection

Table 3.1: Performance of the three algorithms (Data from the publication [33])

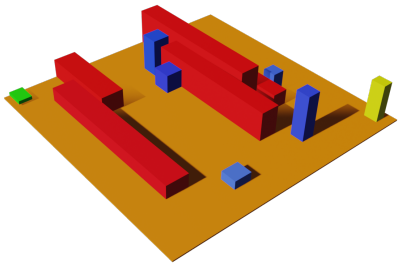
	ref	Average path length	Average time (s)
GDRRT*	[33]	645.84	7.51
PSO-GDRRT*	[33]	524.85	42.85
BiLSTM-PSO-GDRRT*	[33]	669.22	0.019

3.6.2 Comparison with proposed methods

We chose one of the environments the methods from [33] were benchmarked on, and created a similar environment (Figure 3.7). In our environment model, we benchmarked algorithms proposed in this thesis to get a comparison with this related method [33].



(a)



(b) Our environment

Figure 3.7: The original environment (a), and our environment (b) mimicking the original.

The resulting tables can be found in the Appendix (A.4). It is important to point out, that the environment the authors tested their method on is exceptionally simple, which paradoxically put methods proposed in this work at a disadvantage (explained in Section 7.3.3). Because of that, the BiLSTM-PSO-GDRRT* is faster, than the proposed methods.

Chapter 4

Utilized methods

This thesis aims to speed up motion planning by changing the sampling distribution from the uniform distribution to a distribution that has higher density in the regions of c-space that are considered crucial for solving the MP task (as shown in Figure 4.1). We estimate these important regions by finding an approximate path. The approximate path is found using a small cubic object (probe) (explained in Section 5.1.2).

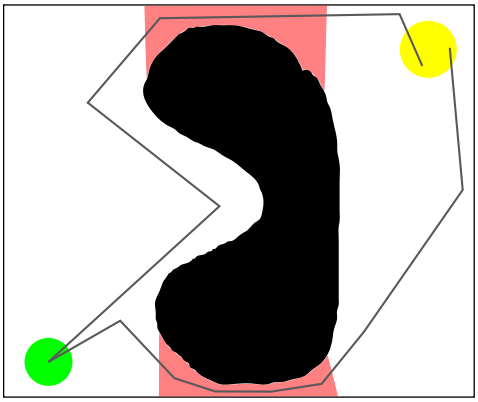


Figure 4.1: Red color represents regions of c-space that have to be sampled in order to obtain path from $\mathcal{C}_{init}$ to $\mathcal{C}_{goal}$.

Three methods were utilized to achieve the goal of speeding up the motion planning. Two of them are closely related to ML, Particle swarm optimization (PSO) [33, 37, 38] and Parzen windows estimation [39]. One of them is an essential ML method, an Artificial neural network (NN) [40]. PSO is an evolutionary-based optimization algorithm that is often used in machine learning to solve optimization problems or for hyperparameter tuning [40]. Parzen windows estimation in a non-parametric density estimation method in machine learning, can be used for estimating the probability density function of the training data. However, in this thesis, the two methods will be used directly in the planning to improve the sampling distribution, and not to tune the NN. Neural networks are used for both classification [40] and regression problems [40]. In this thesis, the NN will be utilized for the regression. These three methods will be now presented to the reader in the following subsections of this chapter.

4.1 Particle swarm optimization

Particle Swarm Optimization (PSO) [37] is an evolutionary method for finding the minimum (or maximum) for a given function. Its big advantage is, that it is able to find local optima of (not only) functions with unknown definitions. The optimized function will be called an *objective function* and denoted $f_o()$.

In PSO (Algorithm 7), a swarm of particles is generated. Each coordinate of each particle represents one variable of the objective function. In every iteration of the algorithm, the particles are evaluated using the objective function. Accordingly to the output of the objective function, the particles in the swarm change their coordinates. The particles of the swarm should over time converge to the local optima of the objective function.

The movement of a particle in the swarm is described by the following equation:

$$\vec{p}_{v\_updated} = w \cdot \vec{p}_v \; + \; c_1 \cdot r_1 \cdot (\vec{p}_b \; - \; \vec{p}_c) + \; c_2 \cdot r_2 \cdot (\vec{g}_b \; - \; \vec{p}_c), \tag{4.1}$$

where constants w, c_1, c_2 are called inertia weight, cognitive coefficient, and social coefficient respectively. The cognitive coefficient represents the bias of every particle in the swarm, toward its own best previously reached position $\vec{p}_b$. The social coefficient stands for the bias of all the particles towards the best position $\vec{g}_b$ reached by any particle of the swarm. Parameters r_1, r_2 are random numbers from interval [0, 1]. The vector $\vec{p}_v$ stands for the velocity assigned to the evaluated particle. The vector $\vec{p}_c$ is the current position of the particle.

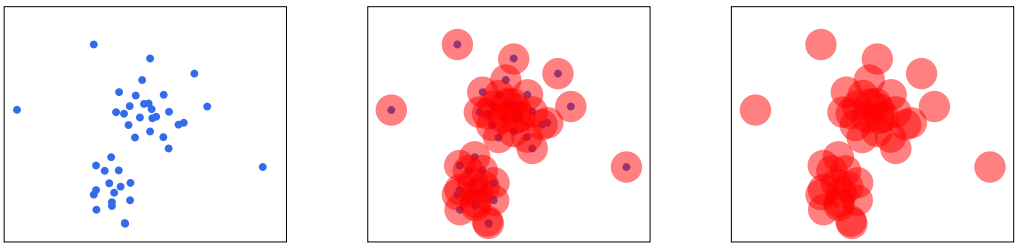
Algorithm 7: PSO (maximizing)

Input: P = number of particles, I = number of iterations, $f_o()$ = objective function
Output: *global_best*

```
1  global_best = ∅;
2  global_best_value = −∞;
3  swarm ← initialize_points(P);
4  for i ∈ 1 : I do
5      for point in swarm do
6          point.velocity ← update(point);        // accordingly to the equation 4.1
7          point.current ← point.current + point.velocity;
8          value ← f_o(point.current);
9          if value > global_best_value then
10             global_best_value ← value;
11             global_best ← point.current;
12         if value > point.best_value then
13             point.best_value ← value;
14             point.best ← point.current;
15 return global_best;
```

4.2 Parzen window estimation

The other name Kernel density estimation is a little more suggestive of the principle of this method [39]. This method is used to estimate a non-parametric probability density function. The estimation is done from a set of points, that were obtained from the estimated distribution (Figure 4.2). The method estimates these points by placing a specified kernel onto each point and then summing the kernels. The sum has to be normalized after this process, so the result we obtain is a probability density function (the integral over the whole domain of the function has to be 1). The estimated distribution can not be directly sampled, because it is non-parametric. But the method can answer for any point, with what probability it was sampled from the estimated distribution. The solution to the problem of the impossibility of direct sampling is explained in Subsection 5.3.1.

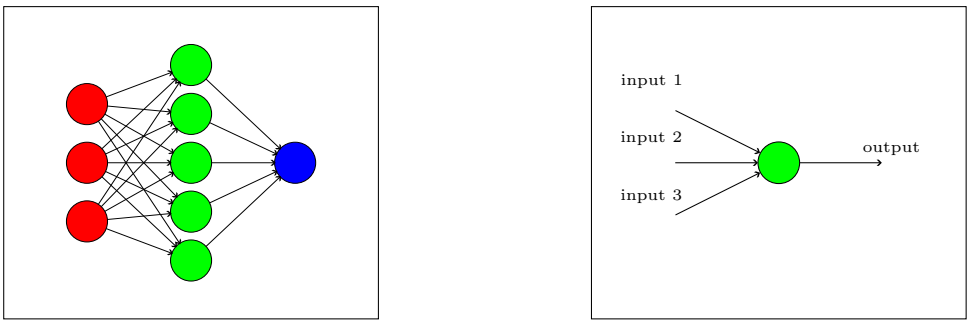


(a) Points from estimated distribution **(b)** Points interlaced with kernels **(c)** The final estimation

Figure 4.2: The procedure of estimation of a non-parametric distribution. As we can see, the estimated distribution probably consists of two normal distributions.

4.3 Neural network

(Artificial) Neural networks are structures that in some sense mimic the behavior of biological brains. The network composes of layers, each containing a given number of nodes. The behavior of the nodes vaguely resembles the behavior of neuron cells in biological brains. The layers, that are not directly responsible for outputting the output of the network, or receiving the input of the network are called hidden layers (Figure 4.3). A specific configuration of the neural network is called a *model*.



(a) An Artificial Neural network **(b)** An Artificial Neuron

Figure 4.3: Network and a node. The red nodes compose an input layer, the green nodes a hidden layer, and the blue node an output layer.

In this thesis, we will assume, the neural network is feed-forward. That means the output of each layer goes only to the following layer without any back-loops (without recursions). The output of the j-th node in the i-th layer of the network can be represented with the following equation:

$$o_{ij} = f(\vec{i'}_{i-1} \cdot \vec{w'}_{ij}). \tag{4.2}$$

Where o_{ij} is a scalar output of the node. The vector $\vec{i'}_{i-1}$ is a vector containing the output of the previous layer (each element is an output of one node in the previous layer), with 1 added as the last element. The vector $\vec{w'}_{ij}$ represents *weights* of the evaluated node except for the last element called *bias*.

The function $f()$ is the *activation function* [40]. The purpose of the activation function is to remove linearity from the neural network, allowing the network to adapt to more complex tasks, than a linear classifier. One of the more well-known activation functions is for example the sigmoid function,

$$\sigma(x) = \frac{1}{1 + e^{-x}},\ x \in \mathbb{R}. \tag{4.3}$$

Another important activation function is the ReLU (Rectified Linear Unit) function, which is used in one of the methods proposed in this thesis

$$ReLU(x) = max(0, x),\ x \in \mathbb{R}. \tag{4.4}$$

The training of the NN refers to the process of finding values of weights and biases of nodes in the network, that will lead to the desired behavior of the NN. The setting of these parameters is done through a process called *backpropagation* [40], which minimizes a *loss function* of the trained model. The loss function is a function dependent on the output of the network and output we are training the model to provide and should in some sense represent the difference between the output the model is returning and the output we want it to return.

One of the frequently used loss functions is for example the *mean square error* function, defined as

$$mse(\vec{y}, \vec{y'}) = \frac{1}{n} \sum_{i=1}^{n} (\vec{y}_i - \vec{y'}_i), \tag{4.5}$$

with $\vec{y}$ standing for desired output, $\vec{y'}$ for the output of the model, and n for the number of elements in the $\vec{y}$ and $\vec{y'}$ vectors.

Training of networks is run in *epochs* which correspond to the number of iterations, in which the weights and biases in the network are tweaked by the backpropagation. The data fed into the network in each epoch is called a *batch*. Usually in each epoch, the network takes batches with higher *batch size*, which is a hyperparameter specifying the number of inputs the network is given in an epoch. In each training epoch, the network is given a batch of training data, then accordingly to the loss function, the weights and biases of the net are updated. How will be changed the weights and biases is determined by *optimizer*. The optimizer used in this thesis is called *ADAM* [40, 41]. The last hyperparameter is *learning rate* which controls how big is the gradient, that will be backpropagated through the network.

Chapter 5

Proposed solutions

This chapter will introduce the proposed methods for motion planning. The proposed methods are designed to speed up solving of high dimensional MP tasks. An increase in the performance of MP methods is desired for example in protein docking (introduced in Section 2.3).

There are six new proposed methods. The order in which they are presented is the chronological order in which they were designed. Each new method's design takes some advantage of the understanding obtained by designing the previous methods. The listed methods are coupled in three pairs. Each pair contains one algorithm that utilizes one of the methods presented in Chapter 4. The other method in the pair is a result of simplification of the ML-related one, in accordance with the assumption stated in Section 1.3 (see Figure 5.1).

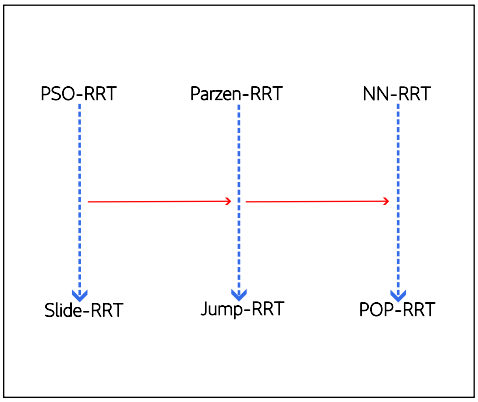


Figure 5.1: A diagram depicting the proposed methods. The blue arrows illustrate the simplification of a method and the red arrows the creation of a new pair of methods based on knowledge gained from designing the previous pair.

The idea the following methods implement is a manipulation of the sampling distribution. A sampling distribution that has a higher density in areas of the c-space that are more helpful to the search should considerably speed up the algorithm. One option how to achieve this is to sample with a higher density around the area where the sought path will be. Since we do not know the position of the sought path before we find it, we have to approximate it. In most of the methods, the approximation will be done with the help of the probe (defined in Section 5.1.2, and Section 2.1).

5.1 Modifications and specifications

This section will introduce and explain all the additional tools used in proposed implementations and modifications to the original RRT (Algorithm 3). A modification of the original RRT is used as a base for the proposed methods. This modification utilizes the concept of impact points introduced in Section 1.1. In the following chapters, this algorithm (8) will be referred to as RRT instead of the original one (Algorithm 3) if not specified otherwise.

Algorithm 8: Impact points RRT

Input: K = maximal number of steps, q_{init} = initial position, q_{goal} = goal position
Output: T

```
1 T.root ← q_init;                                   // initialize the search tree
2 for k ∈ 1 : K do
3     q_rand ← c ∈ C;
4     q_near ← T.find_nearest_neighbour(q_rand);
5     connection, connected = impact_connect(q_near, q_rand);      // Section 5.1.1
6     tree.append(q_near, connection);
7     if connected then
8         if connection ∈ C_goal then
9             return T;
```

The function $tree.find\_nearest\_neighbor()$, is the same as in the original RRT algorithm (Section 3.2). It takes any configuration from the configuration space and returns a node from the search tree. The returned node represents the configuration that is closest to the input configuration out of all nodes in the tree. For this, the KD-tree method was utilized [25], namely, the library MPNN [42].

The function *connect*() from the original RRT algorithm is replaced with the function *impact_connect*(), which plays essentially the same role as the function *connect*(), but it utilizes the idea of impact points. The specifics of this function are explained more in-depth in the following Section 5.1.1.

The local planner (introduced in Section 2.1.1, Figure 2.3) used in this thesis constructs a line between two input configurations q_a and q_b and with a step of predefined size ϵ takes configurations from that line. If any of the configurations belong to $\mathcal{C}_{obs}$ the local planner signals that the two configurations q_a and q_b cannot connect. The collision detection is done using a library RAPID [43, 44].

5.1.1 Impact points

When sampling the configuration space by RRT, it can easily happen to obtain samples that would cause the object's collision with some obstacle. In the original RRT, this would lead to essentially forgetting this iteration and continuing with the next one. Impact points are used to make use of all of the iterations of the algorithm.

The impact point is a point in the configuration space that lies close to the surface of the obstacle. As Figure 5.2 shows, it is the farthest connectible point to our tree node when attempting to connect it with the new random sample. The impact point is then used as a sample instead of the colliding random sample. This way, each iteration is guaranteed to produce a new tree node.

With this concept in mind, an alternative to the function *connect*() (see Section 3.2) was implemented. The function *impact_connect*(). The function indicates if two states are connectible without intersection with $\mathcal{C}_{obs}$ in the same way as *connect*() does. But in addition, when the collision-free connection is not possible appropriate *impact point* is returned. Otherwise, q_{rand} gets returned.

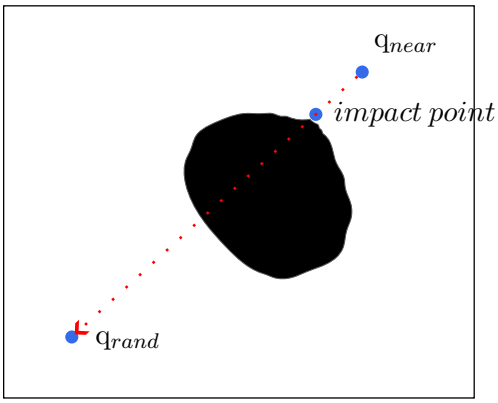


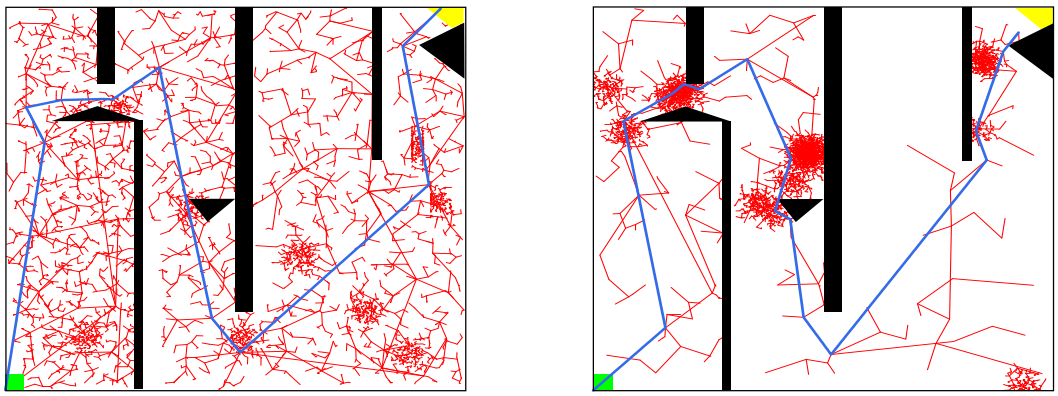
Figure 5.2: Image illustrating, how the impact points are obtained by the local planner.

5.1.2 Probe

The approximation of the path is obtained using standard RRT using a small cubic object (probe). This approach is motivated by the fact that finding a path from the start to the goal for only a small cube is very fast (explained in Section 2.1.1, Figure 2.5). Therefore, several approximate paths can be found at the beginning of the actual search and then later used as an approximation of the final path for our actual search.

5.2 Adaptive sampling distribution methods

The first pair (Figure 5.3) of the proposed methods focus on finding new sampling distributions simultaneously with the search. While the search is running, certain events can arise. For example, a collision. Events like these can serve as a trigger to compute a new sampling distribution. The new distribution is then used for a given number of following iterations. To compute the parameters of the new distributions, the approximate path is used along with data obtained from the event that triggered the computation.



(a) PSO-RRT search tree **(b)** Slide-RRT search tree

Figure 5.3: The search trees of the proposed pair of algorithms.

5.2.1 PSO-RRT

As it was said, algorithms in this section sample from non-uniform distributions when a certain event occurs. In the case of PSO-RRT (Algorithm 9) the event that triggers the computation of alternative distribution is a collision with an obstacle. More specifically if it is a first collision that happened with a specific face of the obstacle. If the object was to collide with a face it has already once collided with, the event would not have been triggered. This was implemented thanks to the ability of the RAPID collision detection library [44], to return IDs of colliding meshes.

On the occurrence of the above-specified collisions, a PSO algorithm (Algorithm 7) is called. The PSO algorithm will return the parameters of a new sampling distribution for a number of following iterations. In the beginning, the PSO algorithm initializes a swarm of 20 points. The space the swarm occupies is a space of parameters (p-space) of possible sampling distributions (illustrated in Figure 5.4). The objective function, the PSO maximizes is defined by the equation

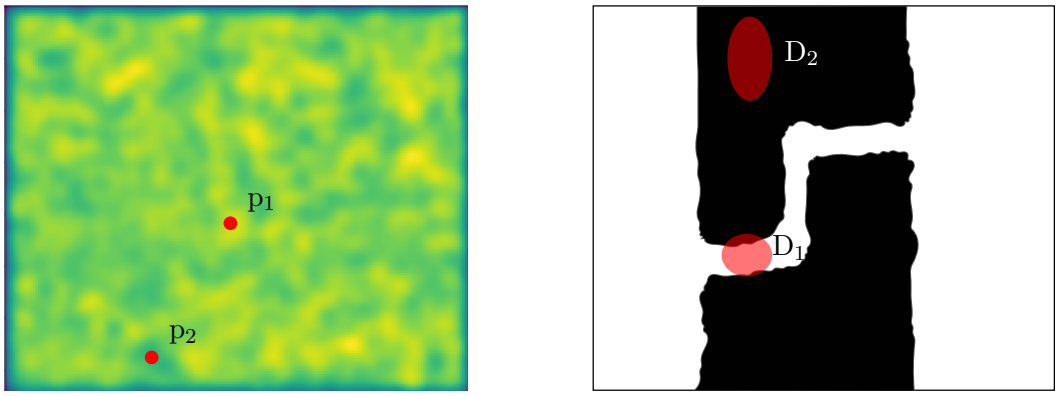
$$f_o(particle) = \frac{closeness \cdot connectability}{distance}. \tag{5.1}$$

The variables of the equation are parameters of the distribution represented by the swarm particle. The variable *closeness* stands for the index of the closest point in the approximate path from the center of the suggested distribution. The closer to the goal configuration, the higher the index. Parameter *connectability* stands for how many points out of a number of points in *distribution* can be without collision connected with the probe to the point of impact. *Distance* represents the distance of the center of the distribution to the closest point in the approximated path.

In this implementation, the sampling distribution is a clipped normal distribution (in the c-space) defined by twelve parameters. The parameters are six spatial coordinates, specifying the position of the center of the distribution in the c-space, and another six parameters specifying a maximal distance from the center in each direction. There are six parameters for both the center and the maximal distance because the task has six degrees of freedom, and the c-space is, therefore, six-dimensional. One point of the p-space is defined as

$$p = \{\mu_x, \mu_y, \mu_z, \mu_\alpha, \mu_\beta, \mu_\gamma, x_r, y_r, z_r, \alpha_r, \beta_r, \gamma_r\}. \tag{5.2}$$

The variance of the sampling distribution is specified as a parameter of the whole PSO-RRT algorithm because it has an overall smaller impact on the performance due to the clipping of the distribution (shown in Appendix A.3, Figure A.4).



(a) Points in p-space **(b)** Corresponding distributions

Figure 5.4: Figure (a) is a heatmap of the optimized function in p-space (the brighter the color, the higher the value of the objective function). Figure (b) shows how that value reflects on the distributions, represented by the points in the p-space. It is shown that the distribution represented by the point in which the optimization function has a higher value is more useful.

The slowest part of the random sampling algorithms is collision detection. Therefore a smaller number of calls of the detection collision is desired. This algorithm (9) does not necessarily do that. Each iteration of an RRT-based algorithm can be counted as one collision detection call. The PSO-RRT, in addition to that, calls the collision when evaluating the $f_o()$ (Equation 5.1). In each of its iterations, the PSO algorithm has to call the collision detection for each of its particles for the number of times the connectability requires. Therefore, whereas the complexity of the base RRT algorithm can be interpreted as $O(K \cdot log(K))$, the complexity of PSO-RRT would be $O(K \cdot A \cdot P \cdot I \cdot B \cdot log(K))$, where A is how many times the PSO algorithm was called, and B is how many points are evaluated when counting the connectability of particle represented distribution. The values of P and I refer to the number of particles in the PSO swarm and the number of PSO iterations, respectively. The increased amount of calls of collision detection is addressed in the next proposed method.

Algorithm 9: PSO-RRT

Input: K = maximal number of steps, N = distribution sample count,
P = number of PSO particles, I = number of PSO iterations,
q_{init} = initial position, q_{goal} = goal position
Output: T

```
1  n ← 0;      // number of samples to be taken from the suggested distribution
2  T.root ← q_init;                                    // initialize the search tree
3  approximate_path ← probe_RRT(q_init, count = 1);       // get approximate path
4  for k ∈ 1 : K do
5      if n > 0 then
6          q_rand ← sample_distribution(dist);
7          n ← n − 1;
8      else
9          q_rand ← uniform_sample();
10     q_near ← tree.find_nearest_neighbour(q_rand);
11     connection, connected = impact_connect(q_near, q_rand);
12     tree.append(q_near, connection);
13     if connected then
14         if connection ∈ C_goal then
15             return T;
16     else
17         dist ← PSO(P, I, f_o()) ;
18         n ← N;
```

5.2.2 Slide-RRT

The problem of the excessive complexity of the previous method had to be dealt with. By observing the outputs the PSO is returning, it occurred that there is a way to model such results with a simpler method requiring fewer collision detection calls. The way to do that is by sliding a fixed-shaped sampling distribution along the approximate path. At the beginning of the search, the distribution will be centered around the q_{init}. An impulse to slide the distribution is a collision. When a collision occurs, the distribution will slide along the approximate path to the furthest location reachable from the point of the impact (Figure 5.5).

The number of collision detection calls is significantly decreased by this modification (Algorithm 10). The complexity of this algorithm can be considered $O(K \cdot L \cdot S \cdot log(K))$, where K is the number of iterations, L is the number of configurations in the approximate path, and S is the number of slidings done in a run. Therefore, for the speed of the algorithm (10), it is beneficial to omit as many unnecessary configurations from the guiding path as possible (illustrated in Figure 3.5). Another aspect, that contributes to the speed of the algorithm is not making any unnecessary slidings. That can be, for example, achieved by applying the concept introduced in the PSO-RRT, that is, by not triggering the slide with every collision, but only if a collision arises with a face of an obstacle that has not yet been collided with.

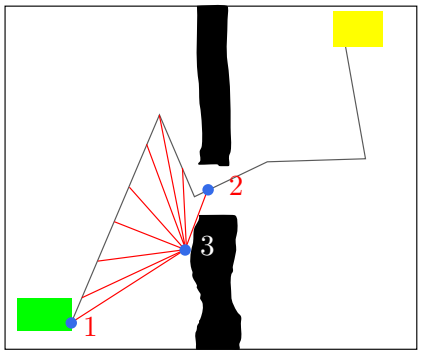


Figure 5.5: Princliple of how the "sliding" is done. The first point is, where the distribution center was before the collision, and the second point is, where the collision was moved to. The third point is an impact point from the collision.

Algorithm 10: Slide-RRT

Input: K = maximal number of steps, N = distribution sample count, q_{init} = initial position, q_{goal} = goal position

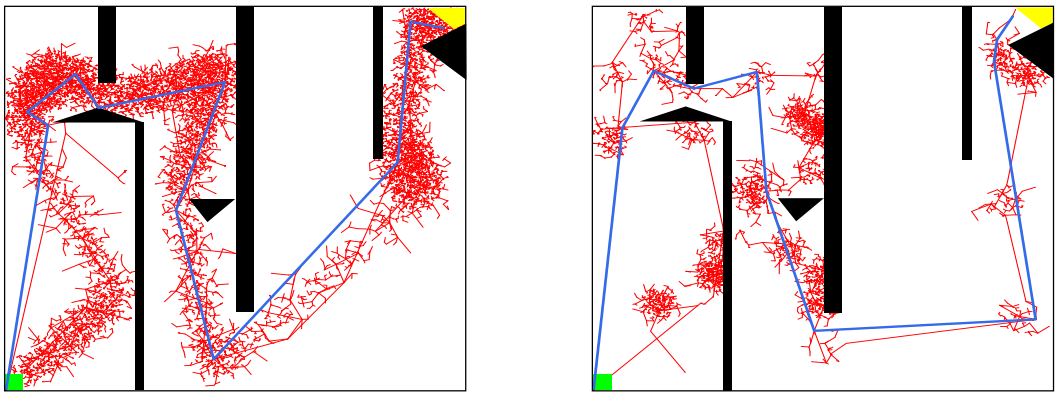
Output: T

$n \leftarrow 0$; `// number of samples to be taken from the suggested distribution`
$slider \leftarrow q_{init}$;
$T.root \leftarrow q_{init}$; `// initialize the search tree`
$approximate\_path \leftarrow probe\_RRT(q\_init, count = 1)$; `// get approximate path`
for $k \in 1 : K$ **do**
    if $n > 0$ **then**
        $q_{rand} \leftarrow sample\_around\_point(slider)$;
        $n \leftarrow n - 1$;
    else
        $q_{rand} \leftarrow uniform\_sample()$;
    $q_{near} \leftarrow tree.find\_nearest\_neighbour(q_{rand})$;
    $connection, connected = impact\_connect(q\_near,\ q\_rand)$;
    $tree.append(q_{near}, connection)$;
    if *connected* **then**
        if $connection \in \mathcal{C}_{goal}$ **then**
            return T;
    else
        $slider \leftarrow slide(approximate\_path, q_{rand})$;
        $n \leftarrow N$;

5.3 Precomputed distribution sampling methods

Even though, the previous pair of algorithms (PSO-RRT and Slide-RRT) modified the sampling in a way, that fewer iterations were needed to find the solution (see benchmarking results in Section 7.3)), the increased number of required collision detection calls was still significant. As an answer to the time increase, the following two methods were designed. The complexity achieved by these methods (Algorithms 11, 12) is the same as the complexity of the original RRT $O(K \cdot log(K))$.

This pair of algorithms (Figure 5.6) presents a modification of the principle the previous pair used. Instead of computing the new sampling distributions on cue, while the search is running, the distributions get computed before the search starts. Both methods then use this precomputed distribution for sampling in more important regions, which speeds up the search.



(a) Parzen-RRT search tree **(b)** Jump-RRT search tree

Figure 5.6: The search trees of the proposed pair of algorithms.

5.3.1 Parzen-RRT

At the start of this algorithm (Algoritm 11), the approximate path is obtained with the help of the probe. Then, points from the obtained path are passed into Parzen windows estimation (introduced in Section 4.2) as points from the distribution we want to estimate. The PWE then estimates what distribution were these points most probably sampled from. The rest of the search is identical to the search of RRT (Algorithm 8), but the random samples are not taken from the uniform distribution. Instead, is being sampled the distribution estimated by PWE. The kernel used in the PWE is a six-dimensional normal distribution.

Since the probability density distribution estimated by PWE is non-parametric, it can not be directly sampled. Instead, an alternative method has to be used. The PWE can tell for any point in its domain, with which probability it would be sampled from the estimated distribution. This was utilized to sample the distribution indirectly. A given number of points from the domain of the distribution function is uniformly sampled. Then PWE returns with how high probability each of them would have been sampled from the estimated distribution. The point, that had the highest probability to be sampled from the estimated distribution is then used as a sample from the estimated distribution.

Algorithm 11: Parzen-RRT

Input: K = maximal number of steps, q_{init} = initial position, q_{goal} = goal position
Output: T

$T.root \leftarrow q_{init}$; // initialize the search tree
$approximate\_path \leftarrow probe\_RRT(q\_init, count = 1)$; // get approximate path
$sampling\_distribution \leftarrow parzen\_windows\_estimate(approximate\_path)$;
for $k \in 1 : K$ **do**
  $q_{rand} \leftarrow sample(sampling\_distribution)$;
  $q_{near} \leftarrow tree.find\_nearest\_neighbour(q_{rand})$;
  $connection, connected \leftarrow impact\_connect(q\_near,\ q\_rand)$;
  $tree.append(q_{near},\ connection)$;
  if *connected* **then**
    if $connection \in \mathcal{C}_{goal}$ **then**
      return T;

5.3.2 Jump-RRT

Densely populated environments posed a challenge for the previously proposed Parzen-RRT (Subsection 5.3.1). That is due to the time invariance of the sampling distribution. The progress of the search has no influence on the sampling, therefore there is a high probability to obtain a sample that is close to the goal, even when the search tree is still closer to the start, than the end and vice versa. In dense environments, samples like such resulted only in excess of impact points and did not much contribute to the growth of the tree. The method proposed in this subsection answers that problem, by introducing an implicit influence of the stage of the search on the sampling.

Similarly to the Parzen-RRT, the search begins with finding the approximate path, with the help of the probe (Algorithm 12). The change is in the sampling process. As in the Slide-RRT (Subsection 5.2.2) moving a normal distribution with fixed variance is being sampled. The normal distribution with fixed variance is initialized in the start configuration q_{init}. The variance is one of the parameters of this method. Two additional parameters are chosen, a *progress threshold* f_t and a *regress threshold* b_t. A big advantage of these two parameters is, that the optimal values seem to be the same in every scenario (shown in Appendix A.3, Figure A.4). In each iteration, the normal distribution is sampled. When a number of successful connections to the search tree surpass the forward threshold, the normal distribution jumps forward on the approximated path (Illustrated in Figure 5.7). If a number of collisions surpass the backward threshold, the normal distribution jumps backward on the approximated path. As a positive side effect, we no longer have to tackle the problem of sampling from non-parametric distribution (as in the Parzen-RRT).

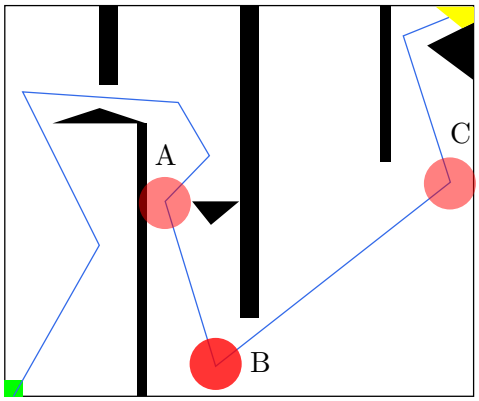


Figure 5.7: While sampling from the normal distribution in the position B, if enough collisions arise, the sampling distribution jumps back to the position A, if enough collision-free connections arise, the sampling distribution jumps forward to the position C.

Algorithm 12: Jump-RRT

Input: K = maximal number of steps, q_{init} = initial position, q_{goal} = goal position, f_t = progress threshold, b_t = regress threshold

Output: T

$f \leftarrow 0$; // number of successful connections
$b \leftarrow 0$; // number of collisions
$path\_idx \leftarrow 0$;
$T.root \leftarrow q_{init}$; // initialize the search tree
$approximate\_path \leftarrow probe\_RRT(q\_init, count = 1)$; // get approximate path
for $k \in 1 : K$ **do**
  $q_{rand} \leftarrow sample\_normal(approximate\_path[path\_idx])$;
  $q_{near} \leftarrow tree.find\_nearest\_neighbour(q_{rand})$;
  $connection, connected \leftarrow impact\_connect(q\_near,\ q\_rand)$;
  $tree.append(q_{near},\ connection)$;
  if *connected* **then**
    if $connection \in \mathcal{C}_{goal}$ **then**
      return T;
    $f \leftarrow f + 1$;
    if $f\ mod\ f_t$ ***is*** 0 **then**
      $path\_idx \leftarrow path\_idx + 1$; // jump forward
  else
    $b \leftarrow b + 1$;
    if $b\ mod\ b_t$ ***is*** 0 **then**
      $path\_idx \leftarrow path\_idx - 1$; // jump back

5.4 Impact point translation methods

The methods in this section determine new sampling distributions with specific translation in c-space away from impact points. That means the location of a new sampling distribution is defined by a vector in c-space directed from the point of the impact into $\mathcal{C}_{free}$.

5.4.1 NN-RRT

The method in a similar fashion to the PSO-RRT (Subsection 5.2.1) computes a new sampling distribution after a collision with a new face of obstacle happens. In NN-RRT (Algorithm 13) the computation, where the following samples should be sampled from, is done by regression with neural network (introduced in Section 4.3). The structure of the training dataset and the training of the network is explained in this subsection.

Algorithm 13: NN-RRT

Input: K = maximal number of steps, N = distribution sample count, q_{init} = initial position, q_{goal} = goal position
Output: T

$f \leftarrow 0$; // number of successful connections
$b \leftarrow 0$; // number of collisions
$n \leftarrow 0$; // number of samples to be taken from the suggested distribution
$path\_idx \leftarrow 0$;
$T.root \leftarrow q_{init}$; // initialize the search tree
$approximate\_path \leftarrow probe\_RRT(q\_init, count = 1)$; // get approximate path
for $k \in 1 : K$ **do**
  if $n > 0$ **then**
    $q_{rand} \leftarrow sample\_normal(q_{near} + translation)$;
    $n \leftarrow n - 1$;
  else
    $q_{rand} \leftarrow sample\_normal(approximate\_path[path\_idx])$;
  $q_{near} \leftarrow tree.find\_nearest\_neighbour(q_{rand})$;
  $connection, connected = impact\_connect(q\_near,\ q\_rand)$;
  $tree.append(q_{near},\ connection)$;
  if *connected* **then**
    if $connection \in \mathcal{C}_{goal}$ **then**
      return T;
    $f \leftarrow f + 1$;
    if $f \bmod f_t$ ***is*** 0 **then**
      $path\_idx \leftarrow path\_idx + 1$; // jump forward
  else
    $translation \leftarrow Network(q_{near},\ q_{rand},\ impact\ point)$;
    $n \leftarrow N$;
    $b \leftarrow b + 1$;
    if $b \bmod b_t$ ***is*** 0 **then**
      $path\_idx \leftarrow path\_idx - 1$; // jump back

Dataset generation

Data acquired from running any other RRT-based method is collected to generate the dataset. While generating the dataset, specific information about collisions gets saved, which is shown in Figure 5.8. Each collision in the dataset generating search produces this data.

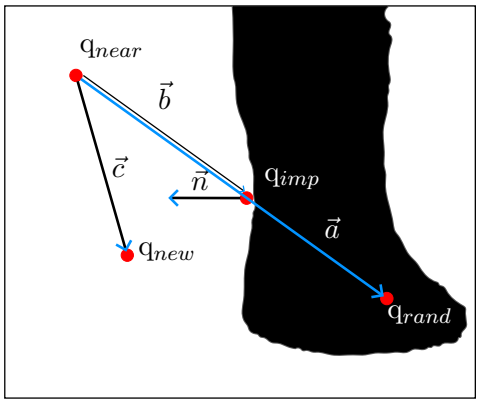


Figure 5.8: The point $\vec{q}_{near}$ represents a configuration that is already connected to the search tree. The point $\vec{q}_{rand}$ represents a new randomly sampled configuration that causes a collision. The point $\vec{q}_{new}$ is a configuration that got connected to $\vec{q}_{near}$ later on during the next iterations. The point $\vec{q}_{imp}$ is an impact point. The vector $\vec{n}$ is a normal of the face, the object collided with. The vectors $\vec{a}$, $\vec{b}$, $\vec{n}$ are given the NN as an input. The vector $\vec{c}$ is the desired output.

Subsequently, a feed-forward NN model of five layers is trained on the dataset, where the input $\vec{x}$ is a vector of size nine

$$\vec{x} = [\vec{a},\ \vec{b},\ \vec{n}], \tag{5.3}$$

and the desired output $\vec{y}$ is given by a vector of size three

$$\vec{y} = [\vec{c}]. \tag{5.4}$$

We obtain vectors $\vec{x}_i$ and $\vec{y}_i$ with each collision and add them to the dataset. The dataset used for the training contained 60 000 pairs of $\vec{x}_i$ and $\vec{y}_i$. And was split into 5 000 samples for validation and 55 000 samples for training.

Training the NN

The training consists of 100 epochs with batches of size 32. As a loss function is used the mean square error function (represented by the equation 4.4). As an optimizer is used ADAM, the learning rate begins at 1e-3, and with every 10 epochs is divided by ten.

The first (input) layer of the trained model has nine nodes (one for each element in the $\vec{x}$). Followed by hidden layers with, 256, 256 and 128 layers with ReLU activation functions (Section 4.3). The last (output) layer has three nodes (one for each element in the $\vec{y}$). The output of the network represents a desired translation of the object in the environment.

The data regarding the final loss after the training and the learning time for both models is in the Appendix (A.5).

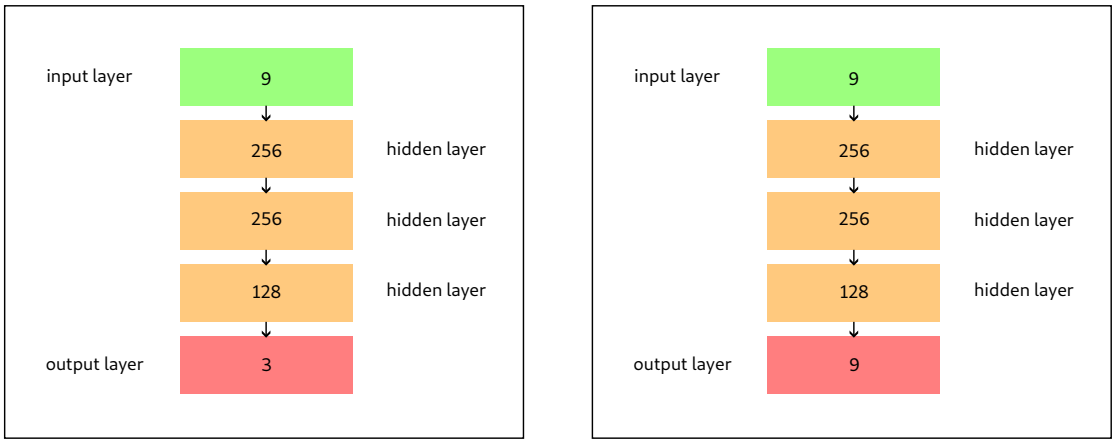
Modifications of the NN for protein docking

Some modifications of the network and the dataset of the network have to be made, in order to use this algorithm in the molecular environment (Figure 5.9). First of all, the collisions in the molecular environment are not given only by physical collisions of two objects, but also by the scoring function (Section 2.3), which checks whether a configuration of molecules is energetically viable. Because of this, it is impossible to obtain the normal of the collision $\vec{n}$.

Also, the number of joints in ligands is not restricted, therefore the number of DOF is not restricted either (each joint adds one degree of freedom). Therefore the size of the configuration vectors $\vec{q}$ can also theoretically grow to any arbitrarily high number.

Since the number of nodes in the input layer of the networks is dependent on the size of the configuration vectors $\vec{q}$, a boundary had to be set on how many degrees of freedom will the network work with. It was decided, that the network will work with maximally nine degrees of freedom. Three DOF for translation, four DOF for rotation of the whole molecule (in the molecular environment, the rotation is implemented in quaternions), and two for two joints of the ligand. With this boundary, the size of the input layer is nine, similar to the previous model. The size of the output is in this case nine accounting for the translation and for the rotation given by quaternion and for the angle in the two first ligand joints (both NN model are illustrated in Figure 5.9).

For implementation reasons imposed by the protein docking environment, the algorithm of this method (Algorithm 13) is combined with the algorithm Bi-RRT (Algorithm 4).



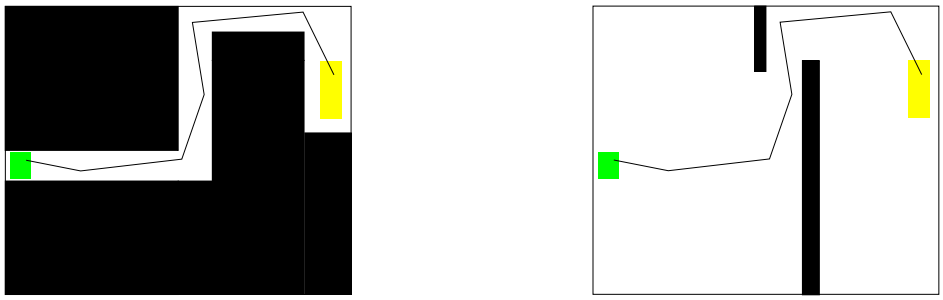
(a) Model for mesh environments **(b)** Model for protein environments

Figure 5.9: The neural network models. The numerical values show the number of nodes in each layer.

5.4.2 Pop-RRT

As is the rule in the presented pairs of algorithms, this algorithm is a simplification of the previous one (Subsection 5.4.1). The simplification is based on the following idea. Since the translation, the network suggests cannot be big (otherwise it would tell the object to constantly collide), we can approximate the translation with a vector of zero size. Therefore not translate at all. This means that on collision, the algorithm will not call the network and instead take a number of following steps from a normal distribution with the center exactly where the collision occurred.

This algorithm is the only one proposed algorithm, that does not use the approximated path computed with the probe. Since the algorithm is designed for environments densely populated with obstacles, the obstacles themselves are used as an approximation of the final path (Illustrated in Figure 5.10).



(a) Dense environment **(b)** Spatious environment

Figure 5.10: In very dense environments, the obstacles themselves can approximate the path.

Algorithm 14: POP-RRT

Input: K = maximal number of steps, N = distribution sample count, q_{init} = initial position, q_{goal} = goal position
Output: T

$n \leftarrow 0$; // number of samples to be taken from the suggested distribution
$T.root \leftarrow q_{init}$; // initialize the search tree
for $k \in 1 : K$ **do**
  if $n > 0$ **then**
    $q_{rand} \leftarrow sample\_normal(connection)$;
    $n \leftarrow n - 1$;
  else
    $q_{rand} \leftarrow c \in \mathcal{C}$; // uniform sample
  $q_{near} \leftarrow tree.find\_nearest\_neighbour(q_{rand})$;
  $connection, connected = impact\_connect(q\_near,\ q\_rand)$;
  $tree.append(q_{near}, connection)$;
  if *connected* **then**
    if $connection \in \mathcal{C}_{goal}$ **then**
      return T;
  else
    $n \leftarrow N$;

There is a possibility to add the approximated path to the Pop-RRT (Algorithm 14) by combining it with the Jump-RRT (Algorithm 12). In denser environments, the use of the approximated path leads to unnecessary actions in each iteration and slows the algorithm down, but in sparsely occupied environments, the path significantly helps the algorithm (because the environment itself is no longer a good approximation of the final path). The resulting algorithm is shown in Appendix (A.1).

5.4.3 Summary

In this section, six new MP methods were proposed. These methods utilize using alternative sampling distributions instead of uniform distribution. That approach should lead to a denser sampling in the more important regions of the c-space and therefore speed up the search for a path from q_{init} to $\mathcal{C}_{goal}$. The methods were divided into three pairs.

The first pair (Adaptive sampling distribution methods) computed the alternative sampling distributions when a specific event occurred. The downside was an increase in computational complexity. The increase was caused by the calls of collision detection when computing the alternative sampling distributions.

The problem of the high complexity was addressed with the second pair of the proposed algorithms (Precomputed distribution sampling methods), which had the same complexity as the original RRT algorithm. The second pair of methods computed the alternative distribution before the search started. That led to a decrease in the time complexity but also in adaptability (if the precomputed distribution is not good enough at the start, the method has no means to improve it while the search is running).

The last pair of the proposed algorithms (Impact point translation methods) combined the positives of both previous pairs and addressed their weaknesses as well. The methods in this pair compute the alternative distributions (as well as the first pair) when a collision arises. This way, the problem of the low adaptability of the second pair is solved. Instead of computing parameters for a new distribution, the parameters of the new distributions are obtained simply by translation in the c-space. That means when a collision occurs, the methods suggest how far we should move from the point of impact and then sample from a predefined distribution in the location we moved to. This reduces the computational complexity problem of the first pair of methods.

Chapter 6

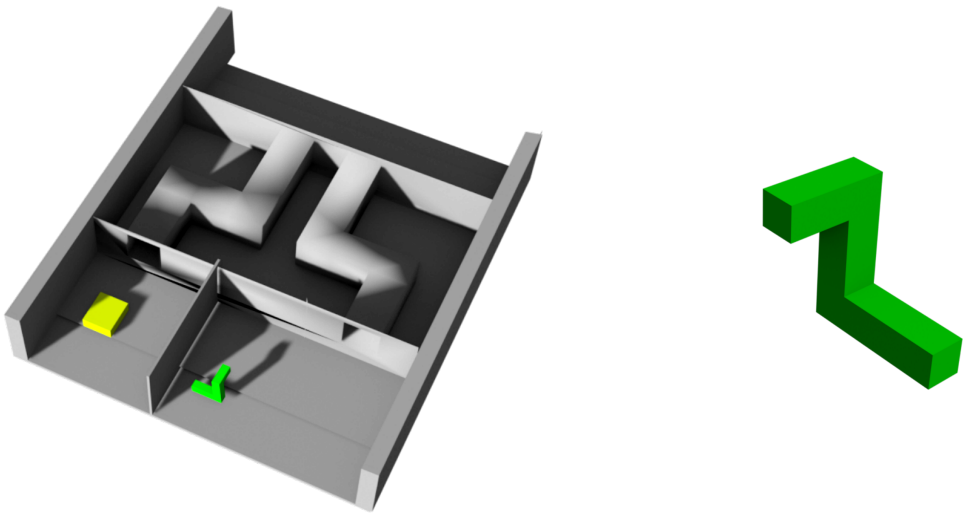
Benchmarking environments

There are four main environments, on which the proposed methods will be benchmarked. All of the environments are three-dimensional in order to test whether the proposed methods perform well in MP tasks with higher dimensions.

6.1 Dense environment

This work is aimed at speeding up MP in order to become viable in protein environments. Hence, the Dense environment is the most important of the modeled environments, because it best resembles the challenges of the protein structures. The protein environments are very densely occupied with the receptor protein molecule and contain only a few (if any) very narrow tunnels for the ligand (Section 2.3). Therefore, the performance of the proposed algorithms in this environment will be important for their evaluation.

The maze box has a width of 35 units, a length of 29 units, and a height of 6 units. The diameter of the tunnel is 3 units. The height width and length of the object are 3 units long (Figure 6.1).



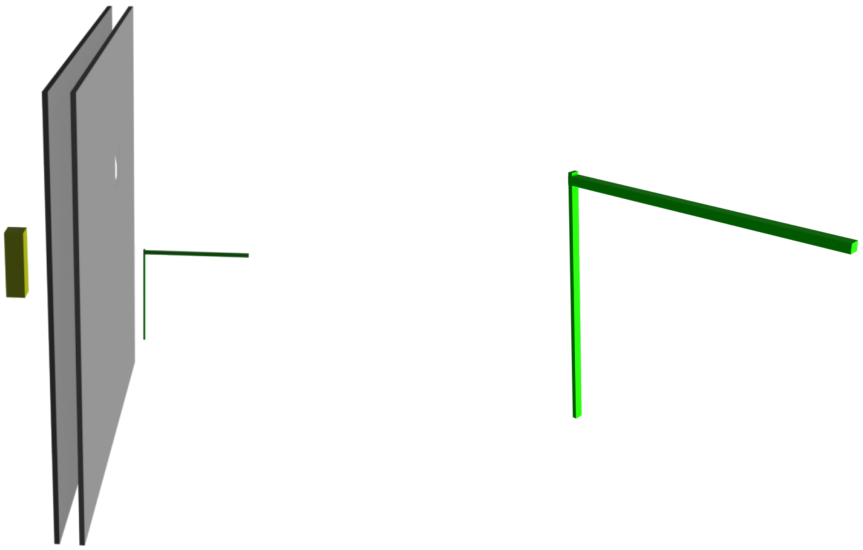
(a) Without a roof **(b)** The Object

Figure 6.1: The dense environment. The obstacles are gray, the object is green, and the $\mathcal{C}_{goal}$ is yellow.

6.2 Complex environment

Even if the proposed methods perform well in the Dense environment, to declare them viable methods they should be tested in another challenging scenario. The challenging element of this environment is caused by one narrow passage. To make achieve this, the object is spatially prominent so precise rotation of the object is necessary for finding its way to the other side of the obstacle. The obstacle composes of two walls, to make for even more strict requirements on the rotation. The space in between the walls is empty, so the object is possible to be manipulated through the walls without any collision.

The height and width of the wall is 97 units, the diameter of the hole is 9 units, and the distance between the walls is 5 units. The height of the object is 22 units and the length is 24 units, width is 1 unit (Figure 6.2).



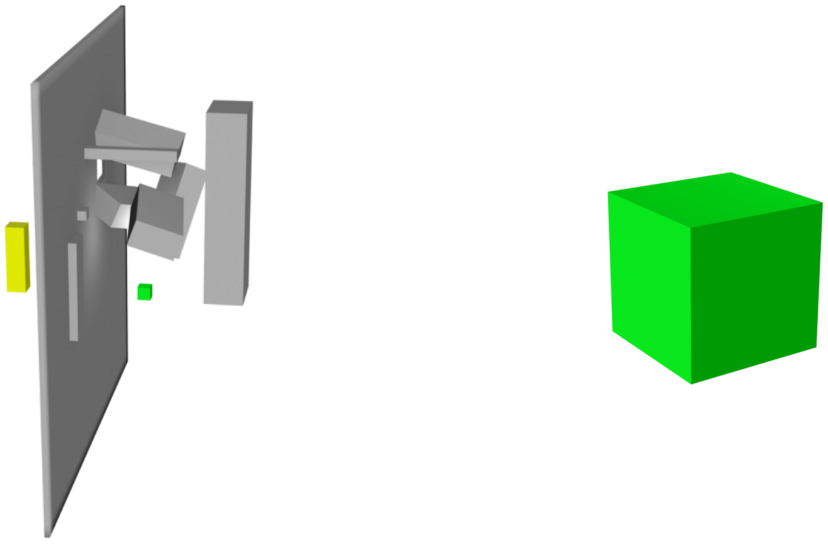
(a) The model **(b)** The Object

Figure 6.2: The complex environment. The obstacles are gray, the object is green, and the $\mathcal{C}_{goal}$ is yellow.

6.3 Simple environment

Benchmarking should be also carried out in a simple environment. This should point out the weaknesses of the proposed models, which were designed with challenging tasks in mind. Only one narrow passage is present in this situation. The object is a cube, which makes the influence of rotations almost negligible (basically transforming the task from 6D to 3D). The environment is the same as the Complex environment, but with only one wall and with random clutter added so the search would not be too fast, so the benchmarked run-times would have noticeable differences.

The environment has the same dimensions, as the Complex environment, but the object is a cube with an edge length of 3 units (Figure 6.3).

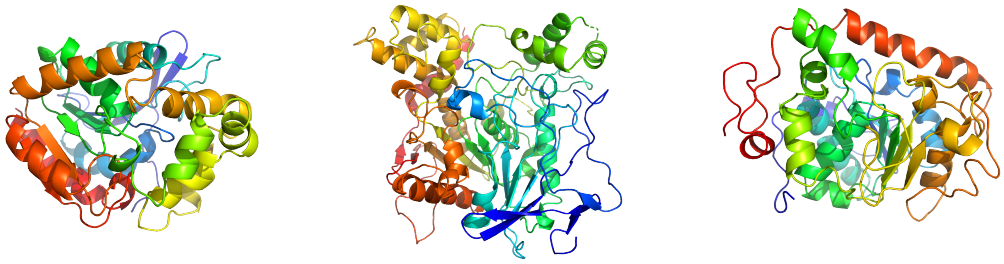


(a) The model **(b)** The Object

Figure 6.3: The simple environment. The obstacles are gray, the object is green, and the $\mathcal{C}_{goal}$ is yellow.

6.4 Protein environment

The protein docking (Section 2.3) will be carried out on the following three molecular complexes (Figure 6.4). These environments will pose the biggest challenge to the selected proposed methods, and the proposed methods will be compared with a related algorithm Bi-RRT.



(a) 1BN7 **(b)** 1MAH **(c)** 1TCC

Figure 6.4: The protein environments.

The first molecule is Haloalkane Dehalogenase from a Rhodococcus species (1BN7) [45]. The second molecule is Fasciculin2-Mouse Acetylcholinesterase Complex (1MAH) [46], which is a complex of a protein (Fasciculin) from snake poison bound to the protein Acetylcholinesterase from a Mouse [46]. The last molecule is The sequence, crystal structure determination and refinement of two crystal forms of lipase B from Candida antarctica (1TCC). The ligand in the first scenario has 2 joints (resulting in 9 DOF). The ligand in the second scenario has 4 joints (11 DOF), and the ligand in the last scenario has 6 joints (13 DOF).

Chapter 7

Benchmarking results

This chapter shows the results of two benchmarking experiments. The first benchmark was done with methods from OMPL [47, 48] (and with each other as well) in environments represented by a 3D triangulated mesh of objects (see results in Section 7.3). The second benchmark compared two of the proposed methods (from Section 5.4) with one state-of-the-art method in protein docking.

For the first benchmark, the expectation was, that the best performance in complex environments will be obtained from proposed non-ML solutions, closely followed by their ML counterparts. The baseline (OMPL) methods are expected to be heavily underperforming. As the complexity of environments decreases, the baseline methods should start to gain an advantage (due to their simplicity) and the proposed solutions should start to decrease in performance (due to their complexity).

The second benchmark compared the proposed ML-related method NN-RRT (proposed in Subsection 5.4.1), its non-ML counterpart Pop-RRT (see Subsection 5.4.2) and the state-of-the-art algorithm Bi-RRT (introduced in Section 3.3). The proposed methods were expected to perform better than Bi-RRT.

7.1 Technical specifications

The proposed methods were implemented in the programming language Julia [49]. For the collision detection and nearest neighbor search were utilized C++ libraries RAPID [43, 44] and MPNN [42] respectively. The choice of the programming language Julia was motivated by its high readability which takes almost no toll on the speed of the executed code. Another important factor was its compatibility with C/C++, which produces generally faster code than Julia. Thanks to the ease with which C++ code can be used by Julia, the parts of the algorithm, where speed was crucial could have been written in C++.

All of the benchmarks were run on one hardware setup with specifications listed in Table **??**. The benchmarking in the OMPL had a time limit of 120 seconds per run. Each planner was executed 100 times per environment. With three OMPL environments, six proposed algorithms, and ten baseline methods, the OMPL benchmarking could take maximally $120 \cdot 10 \cdot 6 \cdot 100 \cdot 3 = 2\ 160\ 000$ seconds (600 hours or 25 days). Because the execution of an algorithm terminated when a solution was found the final time was only approximately 61 hours. The benchmarking of the protein docking had a time limit of 100 seconds per run. In total, the protein docking benchmarking took only 24 hours.

7.2 Used parameters

All of the proposed algorithms have at least one parameter. This section should make it clear to the reader what is their role in the algorithms, and which values were used in the benchmarking (Tables 7.1, 7.2, 7.3)

The most common parameter is a variance of the normal distribution used in the sampling, which will be referred to as *variance* (Algorithms 9, 10, 12, 13, 14). Another parameter is the number of points that will be taken from the suggested distribution, called *density* (Algorithms 9, 10, 12, 13, 14). The next parameters are the regress and progress threshold, in tables named b_t and f_t respectively (Algorithms 12, 13). One parameter is specific to the Parzen-RRT algorithm (11) and it is called *kernel variance* σ_k^2. It is a variance of normal distribution, used in the PWE to estimate the sampling distribution (illustrated in Figure 4.2). The influence of the parameters is illustrated in Appendix (A.3)

Table 7.1: Parameters used in the Dense environment

Planner	Reference	Variance	Density	σ_k^2	f_t	b_t
PSO-RRT	alg.9	2	25	−−	−−	−−
Slide-RRT	alg.10	2	150	−−	−−	−−
Parzen-RRT	alg.11	−−	−−	1.0	−−	−−
Jump-RRT	alg.12	2	−−	−−	5	2
NN-RRT	alg.13	2	100	−−	5	2
Pop-RRT	alg.14	2	25	−−	−−	−−

Table 7.2: Parameters used in the Complex environment

Planner	Reference	Variance	Density	σ_k^2	f_t	b_t
PSO-RRT	alg.9	3	100	−−	−−	−−
Slide-RRT	alg.10	3	500	−−	−−	−−
Parzen-RRT	alg.11	−−	−−	1.0	−−	−−
Jump-RRT	alg.12	3	−−	−−	5	2
NN-RRT	alg.13	3	10	−−	5	2
Pop-RRT	alg.14	3	200	−−	−−	−−

Table 7.3: Parameters used in the Simple environment

Planner	Reference	Variance	Density	σ_k^2	f_t	b_t
PSO-RRT	alg.9	3	100	−−	−−	−−
Slide-RRT	alg.10	3	500	−−	−−	−−
Parzen-RRT	alg.11	−−	−−	1.0	−−	−−
Jump-RRT	alg.12	3	−−	−−	5	2
NN-RRT	alg.13	3	10	−−	5	2
Pop-RRT	alg.14	3	10	−−	−−	−−

The optimal variance of the methods is approximately a square root of the diameter of a narrow passage. Regress and progress thresholds are basically environment independent and their optimal values range between two to ten. The optimal density for each method is different for each method, but it shares a common trend. The denser the environment, the higher the optimal density (elaborated on in Appendix A.3).

7.3 Open Motion Planning Library benchmark

The Open Motion Planning Library (OMPL) is a library, which among other things contains implemented algorithms for solving motion planning problems. Most importantly, it is able to benchmark contained algorithms, as well as any user-provided implementations.

This section contains the results of benchmarking the proposed methods on all three environments, together with some OMPL methods. The methods tested from OMPL are widely used state-of-the-art methods. Namely: BiEST [50], BKPIECE [51], EST [50], KPIECE [51], LazyPRM [52, 53], LazyRRT [53], RRT-connect [26], RRT [7], SBL [54], STRIDE [55].

Each algorithm was run 100 times in each environment. If the algorithm found the path in under 100 seconds it was considered a successful attempt. Otherwise the search was halted and labeled as a failed attempt. The algorithms were evaluated with further-listed criteria. The first already touched criterion is the success rate s_r of the algorithm. Another two criteria are the average run-time t_a of the algorithm and the average number of iterations per run i_a. Criterion t_i is the average time it took to finish one iteration of the algorithm. The last criterion is efficiency e_t, defined as

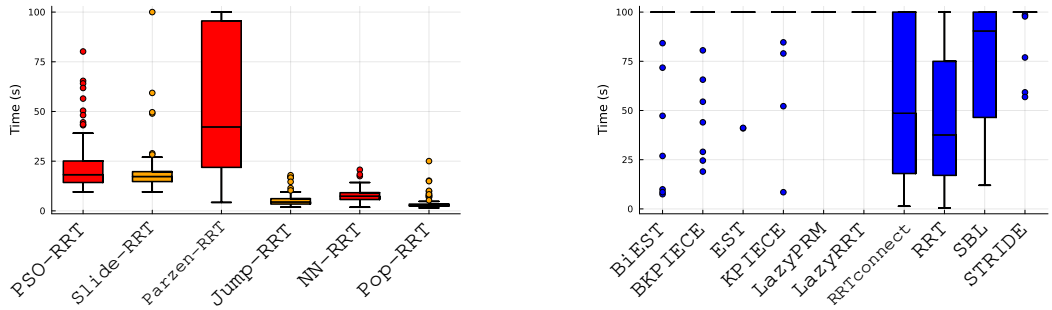
$$e_t = \frac{s_r}{t_a}. \tag{7.1}$$

The efficiency grows with the success rate and with the speed of the algorithm.

In the benchmarking results graphs, the non-ML-related methods will be colored in orange color, and the ML-related ones will be shown in red. The OMPL implementations will be blue.

7.3.1 Dense environment

As expected, in the Dense environment (described in Figure 6.1) the proposed algorithms performed much better than the OMPL algorithms (Figure 7.1). Both in terms of speed, number of iterations required to find the path, and overall success rate (Table 7.4).



(a) The proposed methods **(b)** The state of the art

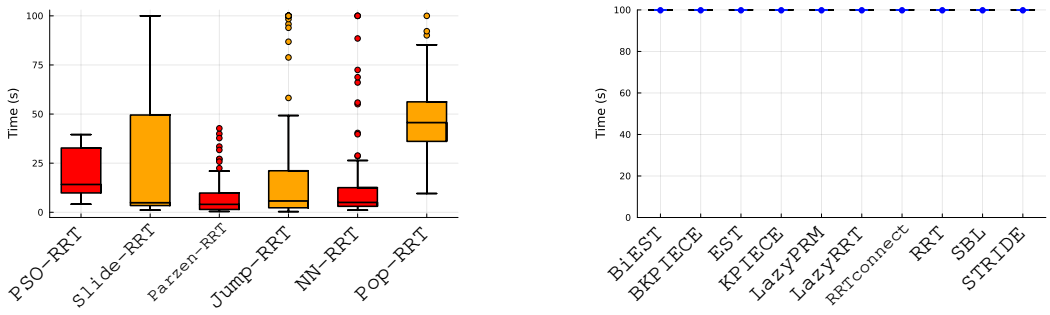
Figure 7.1: The speed of the algorithms in the Dense environment.

Table 7.4: Performance of the proposed algorithms in the Dense environment

Planner	Reference	t_a (s)	i_a ($\times 10^3$)	s_r (%)	e_t (s^{-1})	t_i (ms)
PSO-RRT	alg.9	22.94	87	100	4.36	0.30
Slide-RRT	alg.10	19.57	33	99	5.05	0.64
Parzen-RRT	alg.11	89.60	166	76	0.84	0.44
Jump-RRT	alg.12	5.57	36	100	17.94	0.15
NN-RRT	alg.13	7.87	46	100	12.69	0.17
Pop-RRT	alg.14	2.75	23	100	26.64	0.15
BiEST	[50]	95.61	12	7	0.07	8.10
BKPIECE	[51]	96.22	719	7	0.07	0.13
EST	[50]	98.87	6	2	0.02	15.50
KPIECE	[51]	98.29	480	4	0.04	0.20
LazyPRM	[52]	100	60	0	0	1.66
LazyRRT	[53]	100	12	0	0	7.88
RRTconnect	[26]	54.39	151	72	1.32	0.31
RRT	[7]	46.67	92	83	1.77	0.44
SBL	[54]	72.80	1527	52	0.71	0.04
STRIDE	[55]	98.94	377	5	0.05	0.26

7.3.2 Complex environment

In the Complex environment (shown in Figure 6.2), the OMPL methods were not able to successfully find a single path. The performance of the proposed methods is visibly worse than in the Dense scenario (Figure 7.2), but the methods are still reasonably fast (Table 7.5). For some of the OMPL methods, the number of iterations was not possible to obtain, due to inner OMPL implementation reasons.



(a) The proposed methods **(b)** The state of the art

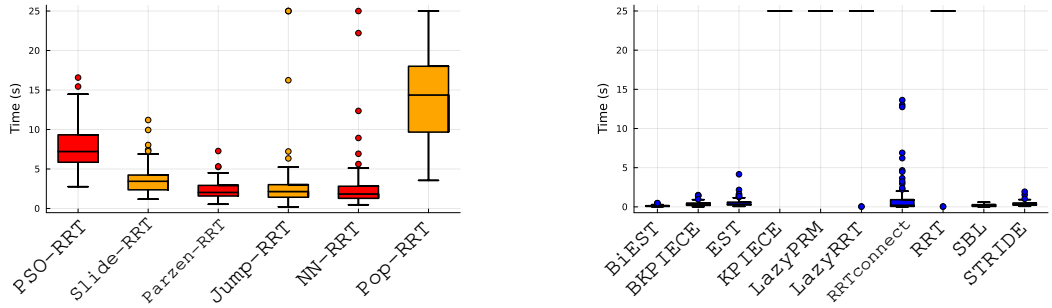
Figure 7.2: The speed of the algorithms in the Complex environment.

Table 7.5: Performance of the proposed algorithms in the Complex environment

Planner	Reference	t_a (s)	i_a ($\times 10^3$)	s_r (%)	e_t (s^{-1})	t_i (ms)
PSO-RRT	alg.9	19.50	43	100	5.13	0.20
Slide-RRT	alg.10	29.53	82	86	2.91	0.94
Parzen-RRT	alg.11	7.69	21	100	13.03	0.56
Jump-RRT	alg.12	27.8	91	88	3.16	0.84
NN-RRT	alg.13	14.08	43	97	6.88	0.47
Pop-RRT	alg.14	47.27	177	99	2.09	0.20
BiEST	[50]	100	−−	0	0	−−
BKPIECE	[51]	100	−−	0	0	−−
EST	[50]	100	10	0	0	9.54
KPIECE	[51]	100	795	0	0	0.12
LazyPRM	[52]	100	−−	0	0	−−
LazyRRT	[53]	100	68	0	0	3.65
RRTconnect	[26]	100	−−	0	0	−−
RRT	[7]	100	395	0	0	0.25
SBL	[54]	100	−−	0	0	−−
STRIDE	[55]	100	−−	0	0	−−

7.3.3 Simple environment

The Simple environment (illustrated in Figure 6.3) proved to be the toughest challenge for the proposed methods, as expected (Table 7.6). The methods from the OMPL were on the other side performing very well (Figure 7.3). They were able to find the path in every single run. Due to the low complexity of the environment, it is possible to say, that the proposed methods had to run the search twice because finding the path for the probe was similarly simple to solving the whole task. Because of that for some planners, it was tested how well they would perform if they did not have to compute the approximate path, and instead, it was computed before the benchmarking and passed directly into them as an argument (Appendix A.2, Figure A.1).



(a) The proposed methods **(b)** The state of the art

Figure 7.3: The speed of the algorithms in the Simple environment.

Table 7.6: Performance of the proposed algorithms in the Simple environment

Planner	Reference	t_a (s)	i_a ($\times 10^3$)	s_r (%)	e_t (s^{-1})	t_i (ms)
PSO-RRT	alg.9	7.65	5.46	100	13.06	1.98
Slide-RRT	alg.10	3.63	6.10	100	27.50	0.63
Parzen-RRT	alg.11	2.27	0.48	100	44.01	6.13
Jump-RRT	alg.12	7.09	14.20	98	13.81	2.70
NN-RRT	alg.13	3.35	6.34	100	29.80	2.36
Pop-RRT	alg.14	14.34	74.11	100	6.97	1.93
BiEST	[50]	0.12	0.60	100	830.80	0.18
BKPIECE	[51]	0.37	5.40	100	268.80	0.07
EST	[50]	0.51	1.66	100	193.90	0.26
KPIECE	[51]	100	−−	0	0	−−
LazyPRM	[52]	1.08	4.18	100	92.18	0.13
LazyRRT	[53]	97.01	48.01	5	0.05	1.99
RRTconnect	[26]	0.18	1.45	100	542.36	0.12
RRT	[7]	3.60	395	100	253.30	0.11
SBL	[54]	0.30	12.43	100	338.70	0.02
STRIDE	[55]	2.14	17.71	100	46.81	0.12

7.4 Protein docking benchmark

The most complex environments the proposed methods were benchmarked on were the Protein docking scenarios (introduced in Section 6.4). The paths in the protein docking environments are harder to find because of two main factors. The first factor is, that the environment usually consists of narrow tunnels. The second factor is, that the ligand usually has at least one joint and each joint adds an additional degree of freedom to the task, rendering it harder to solve.

Three planners were tested on three Protein docking tasks (Section 2.3). The molecular complexes were 1BN7, 1MAH, and 1TCC (Section 6.4). The planners were BiRRT, NN-RRT, and the simplification of NN-RRT, the Pop-RRT, (Algorithms 4, 13, 14). The algorithm evaluation criteria are the same as in the previous section (7.3), but the time limit is increased from 100 seconds to 120 seconds.

7.4.1 Molecule 1BN7

Since the molecular environment is more complex than the previously benchmarked environments, the time it takes, to find the solution is higher (Table 7.7). The newly proposed methods (Algorithm 14, 13) are outperforming the state-of-the-art method Bi-RRT (Algorithm 4).

Table 7.7: Performance of the proposed algorithms in docking ligand into the 1BN7 molecule.

Planner	Reference	t_a (s)	i_a ($\times 10^3$)	s_r (%)	e_t (s^{-1})	t_i (ms)
NN-RRT	alg.13	38.41	26	100	2.60	1.52
Pop-RRT	alg.14	32.15	24	100	3.11	1.38
Bi-RRT	alg.4	50.69	56	100	1.97	1.78

7.4.2 Molecule 1MAH

The gap in performance, between the state-of-the-art method, and the proposed methods is even more obvious in this more complex molecular environment (Table 7.8). The difference is especially significant when it comes to the success rate of the algorithms. Since the two proposed algorithms are similar in their nature (the Pop-RRT is a simplification of the NN-RRT), their performance is also similar.

Table 7.8: Performance of the proposed algorithms in docking ligand into the 1MAH molecule.

Planner	Reference	t_a (s)	i_a ($\times 10^3$)	s_r (%)	e_t (s^{-1})	t_i (ms)
NN-RRT	alg.13	101.76	27	47	0.46	3.91
Pop-RRT	alg.14	99.99	27	45	0.45	3.40
Bi-RRT	alg.4	115.77	35	6	0.05	3.29

7.4.3 Molecule 1TCC

All of the methods are faster than in the previous two scenarios (Table 7.9). The state-of-the-art method Bi-RRT is the fastest, which was not true in any of the previous environments. The proposed methods are still more successful in finding the path than the state-of-the-art method.

Table 7.9: Performance of the proposed algorithms in docking ligand into the 1TCC molecule.

Planner	Reference	t_a (s)	i_a ($\times 10^3$)	s_r (%)	e_t (s^{-1})	t_i (ms)
NN-RRT	alg.13	17.40	3	100	5.75	10.89
Pop-RRT	alg.14	12.28	4	99	5.73	9.50
Bi-RRT	alg.4	9.04	4	96	10.61	2.57

7.5 Summary

In this chapter, all the proposed methods were benchmarked with OMPL methods and in protein docking. The proposed methods outperformed all the baseline methods in all scenarios except the simplest one. That was caused by the fact that the proposed methods use the probe to obtain the approximated path, but in the Simple environment, finding the final path was not much harder than finding the approximated path. Therefore the proposed methods were basically solving the task twice per run.

Where the proposed methods performed the best was in the Dense environment. That is because the methods were designed with dense scenarios in mind because that is usually the case with protein docking environments. The Complex scenario posed a challenge for the proposed methods, but they all were able to solve the task (the OMPL methods were not).

In the protein docking itself, expectations of outperforming Bi-RRT by the proposed algorithms were met. As it was mentioned already, the methods were designed with the challenge of protein docking tasks in mind, which gave them an advantage.

Chapter 8

Conclusion

This work was mainly focused on two goals. Proposing motion planning algorithms, that will perform better than state-of-the-art implementations in environments with a high number of degrees of freedom. And investigating, whether machine learning (and related tools) is a viable option for speeding up motion planning algorithms.

Six novel algorithms were introduced and benchmarked first against algorithms from Open Motion Planning Library in three environments with varying complexity, and later against Bidirectional-RRT in protein docking. The proposed methods outperformed the state-of-the-art methods in all of the environments except the simplest one.

In the more complex scenarios benchmarked against the Open Motion Planning Library methods the difference between the proposed methods was significant. However, in the simple scenario, the state-of-the-art methods stood their ground which is due to the simplicity of the state-of-the-art methods. Finding the path in the simple environment was so fast, the advantages of the proposed methods had no time, to manifest themselves. Quite the contrary, the additional complexity of the proposed methods caused them to compute more than was necessary and slowed them down.

The initial assumption of this thesis was, that machine learning would not prove to be of much help (RRT-based) in motion planning. The assumption came from the thought, that when the task is well-defined and well-understood by the programmer, he should have the ability to directly code algorithms manifesting exactly the behavior he deemed best for solving the task. That would eliminate any benefit of machine learning, which helps in tasks that are hard (or impossible) to define (such as deciding, whether it is a cat or a sunlamp in a picture), by converging toward the desired behavior over time, by obtaining more data and accordingly adjusting its behavior.

As the data from benchmarking show, this assumption was not totally accurate. In the scenarios densely populated with obstacles, the proposed methods utilizing machine learning (or related methods) were outperformed by their hard-coded counterparts. However, as the density of the environments decreased any prediction about whether the machine-learning-related methods or their counterparts would perform better fell apart.

Two of the proposed methods were benchmarked in protein docking. A task very challenging due to its high number of degrees of freedom. One of the methods utilized a neural network and the other was a simplification of that method. The method the proposed algorithms were benchmarked against was a state-of-the-art method called Bidirectional RRT. In all of the benchmarking scenarios, the proposed methods performed better, than the state-of-the-art method.

Chapter A

Appendix

A.1 Pop+Jump-RRT

The combination of the algorithms Pop-RRT and Jump-RRT (Algorithms 14, 12).

Algorithm 15: P+J-RRT

Input: K = maximal number of steps, N = distribution sample count, q_{init} = initial position, q_{goal} = goal position
Output: T

$f \leftarrow 0$; // number of successful connections
$b \leftarrow 0$; // number of collisions
$n \leftarrow 0$; // number of samples to be taken from the suggested distribution
$path\_idx \leftarrow 0$;
$T.root \leftarrow q_{init}$; // initialize the search tree
$approximate\_path \leftarrow probe\_RRT(q\_init, count = 1)$; // get approximate path
for $k \in 1 : K$ **do**
    if $n > 0$ **then**
        $q_{rand} \leftarrow sample\_normal(connection)$;
        $n \leftarrow n - 1$;
    else
        $q_{rand} \leftarrow sample\_normal(approximate\_path[path\_idx])$;
    $q_{near} \leftarrow tree.find\_nearest\_neighbour(q_{rand})$;
    $connection, connected = impact\_connect(q\_near,\ q\_rand)$;
    $tree.append(q_{near},\ connection)$;
    if *connected* **then**
        if $connection \in \mathcal{C}_{goal}$ **then**
            return T;
        $f \leftarrow f + 1$;
        if $f \bmod f_t$ ***is*** 0 **then**
            $path\_idx \leftarrow path\_idx + 1$; // jump forward
    else
        $b \leftarrow b + 1$;
        if $b \bmod b_t$ ***is*** 0 **then**
            $path\_idx \leftarrow path\_idx - 1$; // jump back
        $n \leftarrow N$;

A.2 Performance with precomputed paths

The performance of selected proposed methods in the Simple environment when the computation of the approximate path is computed before their run-time and passed into the as an argument (Figure A.1 and Table A.1). The performance of the OMPL methods is also displayed for the comparison (Figure A.2).

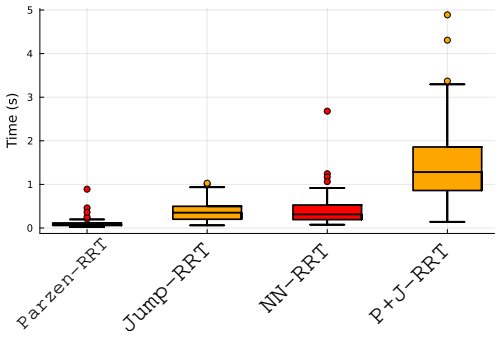


Figure A.1: Performance of selected proposed methods with a precomputed approximate path in the Simple environment.

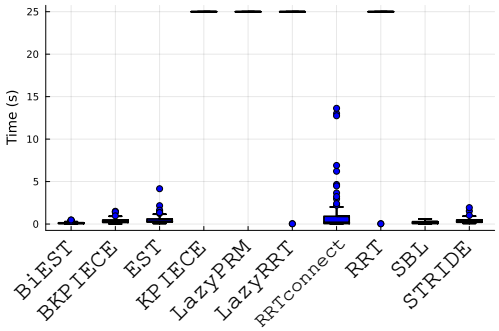


Figure A.2: Performance of OMPL methods in the Simple environment.

Table A.1: Performance of selected proposed methods with a precomputed approximate path in the Simple environment. Best-performing OMPL methods are included for reference.

Planner	Reference	t_a (s)	i_a ($\times 10^3$)	s_r (%)	e_t (s^{-1})	t_i (ms)
Parzen-RRT	alg.11	0.10	0.33	100	960.11	0.35
Jump-RRT	alg.12	0.37	1.27	100	264.54	0.42
NN-RRT	alg.13	0.41	1.09	100	241.78	0.35
P+J-RRT	alg.15	1.49	7.67	100	66.99	0.02
BiEST	[50]	0.12	0.60	100	830.80	0.18
BKPIECE	[51]	0.37	5.40	100	268.80	0.07
EST	[50]	0.51	1.66	100	193.90	0.26
LazyPRM	[52]	1.08	4.18	100	92.18	0.13
RRTconnect	[26]	0.18	1.45	100	542.36	0.12
SBL	[54]	0.30	12.43	100	338.70	0.02

A.3 Parameter influence

Graphs of influence of parameters on some selected algorithms. The graphs are interpreted as what percentage (y axis) of the 100 runs of the algorithm finished under a given time (x axis).

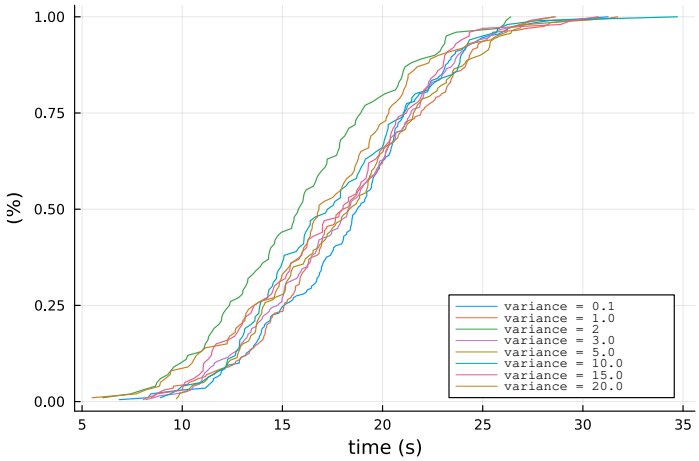


Figure A.3: Influence of the variance parameter on the PSO-RRT algorithm (9) performance, when the parameter density is fixed at value 25.

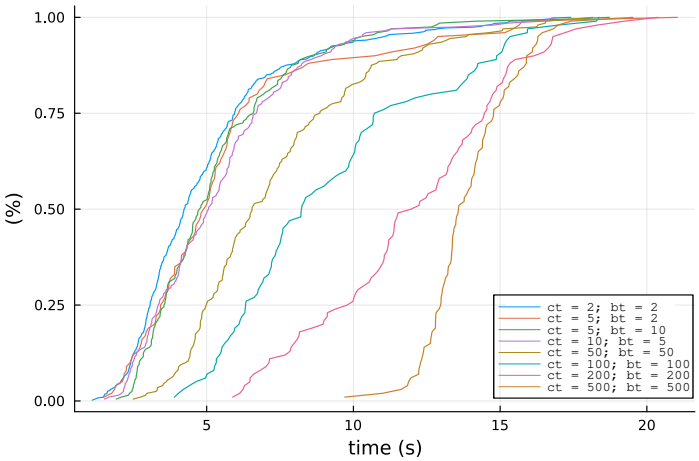


Figure A.4: Influence of the progress threshold (f_t) and regress threshold (b_t) parameters on the Jump-RRT algorithm (12) performance.

Benchmarking data (can be found in attachments B) shows, that the optimal value for a variance of algorithms is around the square root of the diameter of a narrow passage present in the environment. The optimal value of the density parameter decreases with decreasing density of the obstacles. When the density is set to 1 the algorithms will behave like the original RRT. The optimal values move around hundreds for denser environments and around tens in more sparse environments. For each algorithm, the optimal value is different (as opposed to variance, which is the same for all algorithms in the same environment). The optimal progress and regress thresholds lie always in the interval [2,10]. The optimal kernel variance is usually close to one-half of the optimal variance.

A.3.1 Optimizing paths

Results of testing, whether it pays off, to optimize the guiding (approximate) paths. If a path was optimized, it is tagged with "o" in the tables (A.2, A.3) and with "no" when not. If the path was interpolated (more configurations were inserted between the configurations already present in the path) it was tagged with "i" and with "ni" otherwise.

Table A.2: Average run-time (s) in the Dense environment

Planner	Reference	o-i	no-i	o-ni	no-ni
PSO-RRT	alg.9	−−	−−	12.89	17.45
Slide-RRT	alg.10	−−	−−	16.71	14.14
Parzen-RRT	alg.11	−−	−−	20.63	25.33
Jump-RRT	alg.12	5.98	6.28	6.10	6.51
NN-RRT	alg.13	24.22	22.24	4.49	5.09

Table A.3: Success rate (%) in the Dense environment

Planner	Reference	o-i	no-i	o-ni	no-ni
PSO-RRT	alg.9	−−	−−	92	93
Slide-RRT	alg.10	−−	−−	96	96
Parzen-RRT	alg.11	−−	−−	44	78
Jump-RRT	alg.12	97	89	93	90
NN-RRT	alg.13	89	79	100	100

A.4 BiLSTM-PSO-GDRRT* comparison with proposed methods

It is important to note, that this comparison (Tables A.4, A.5) can not lead to conclusions about which algorithms are better because the environments (Figure 3.7) were not the exactly the same, the benchmarking hardware was different, and the methods proposed in this thesis are not designed to find the optimal path.

Table A.4: Performance of the three algorithms in environment from Figure 3.4. (Data are from the publication [33])

	Reference	Average time (s)
GDRRT*	[33]	11.53
PSO-GDRRT*	[33]	48.73
BiLSTM-PSO-GDRRT*	[33]	0.013

Table A.5: Performance of the proposed algorithms in environment from Figure 3.7

	Reference	Average time (s)
RRT	algorithm 8	0.08
PSO-RRT	algorithm 9	0.75
Slide-RRT	algorithm 10	0.27
Parzen-RRT	algorithm 11	0.61
Jump-RRT	algorithm 12	0.76
NN-RRT	algorithm 13	0.63
Pop-RRT	algorithm 14	0.40
P+J-RRT	algorithm 15	0.33

Because the final algorithm in [33] BiLSTM-PSO-GDRRT* is not computing the path while running the search, but using the path provided by the NN, we also tested the performance of some of our methods with the path computed before the benchmark (Table A.6), and then our method NN-RRT (Algorithm 13) performed very similarly.

Table A.6: Performance of the proposed algorithms in environment from Figure 3.7, with precomputed path

	Reference	Average time (s)
NN-RRT	algorithm 13	0.03
Pop-RRT	algorithm 14	0.07
P+J-RRT	algorithm 15	0.05

A.5 Network training

Here are presented the data regarding the time and final loss of training of two used neural network models (Table A.7). The model named "Mesh" was used on environments represented by meshes (used for benchmarking with OMPL). The model named "Protein" was used for the protein docking environments. The training was implemented using Flux.jl library [56, 57].

Table A.7: Performance of the proposed algorithms in environment from Figure 3.7, with pre-computed path

Model	Training-time (a)	Final loss
Mesh	247.45	1.50
Protein	173.51	0.02

Chapter B

Attachments

The thesis comes together with an attached `source_code.zip` file. The file contains all the source code of the algorithms, the meshes of the benchmarked environments, and the benchmarked data.

The folder `algorithms/` contains the implemented algorithms. The folder `analyze/` contains the benchmarked data and Julia [49] scripts for the analysis of the data. The folder `jl_libs/` contains definitions of functions and structures shared by the algorithms. The folder `libs/` contains external libraries. The folder `ompl/` contains files related to the Open Motion Planning Library benchmarking. The folder `training/` contains datasets and NN models. The folder `meshes/` contains blender files of environments and Python scripts for visualizing the planning results. The home directory then contains scripts to run the search and run the training of networks, along with wrappers for the libraries in `libs/`.

Chapter C

References

[1] A. Sagitov et al. "Design of Simple One-Arm Surgical Robot for Minimally Invasive Surgery". In: Oct. 2019, pp. 500–503. DOI: 10.1109/DeSE.2019.00097.

[2] *Surgical Robotic Arm Systems Hand Tools*. Accessed: 2023-05-21. URL: https://www.alliedmotion.com/surgical-robotic-arm-systems/.

[3] *RimWorld*. https://rimworldgame.com/. Montreal: Ludeon Studios, 2018.

[4] Steven M. Lavalle. *Planning Algorithms*. Cambridge University Press, 2006. ISBN: 0521862051.

[5] L.E. Kavraki et al. "Probabilistic roadmaps for path planning in high-dimensional configuration spaces". In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), pp. 566–580. DOI: 10.1109/70.508439.

[6] O. Trott and A. J. Olson. "AutoDock Vina: improving the speed and accuracy of docking with a new scoring function, efficient optimization, and multithreading". en. In: *J Comput Chem* 31.2 (Jan. 2010), pp. 455–461.

[7] Steven M. LaValle. "Rapidly-exploring random trees : a new tool for path planning". In: *The annual research report* (1998).

[8] L. Jaillet et al. "Adaptive tuning of the sampling domain for dynamic-domain RRTs". In: Sept. 2005, pp. 2851 –2856. DOI: 10.1109/IROS.2005.1545607.

[9] P. A. Burrough et al. *8.11 Nearest neighbours: Thiessen (Dirichlet/Voroni) polygons*. Oxford University Press, 2015, pp. 160–163. ISBN: 978-0-19-874284-5.

[10] *Oxford English Dictionary*. Accessed: 2023-04-11. URL: https://languages.oup.com/research/oxford-english-dictionary/.

[11] T. Lengauer and M. Rarey. "Computational methods for biomolecular docking". In: *Current Opinion in Structural Biology* 6.3 (1996), pp. 402–406. ISSN: 0959-440X. DOI: https://doi.org/10.1016/S0959-440X(96)80061-3. URL: https://www.sciencedirect.com/science/article/pii/S0959440X96800613.

[12] J. Li, A. Fu, and L. Zhang. "An Overview of Scoring Functions Used for Protein–Ligand Interactions in Molecular Docking". In: *Interdisciplinary Sciences: Computational Life Sciences* 11.2 (2019), pp. 320–328. ISSN: 1867-1462. DOI: https://doi.org/10.1007/s12539-019-00327-w. URL: https://doi.org/10.1007/s12539-019-00327-w.

[13] A. N. Jain. "Scoring functions for protein-ligand docking". en. In: *Curr Protein Pept Sci* 7.5 (Oct. 2006), pp. 407–420.

[14] D. Devaurs et al. "MoMA-LigPath: a web server to simulate protein–ligand unbinding". en. In: *Nucleic Acids Research* 41.W1 (July 2013). 31 citations (Crossref) [2023-03-30], W297–W302. ISSN: 1362-4962, 0305-1048. DOI: 10.1093/nar/gkt380. URL: http://academic.oup.com/nar/article/41/W1/W297/1097124/MoMALigPath-a-web-server-to-simulate-proteinligand (visited on 03/28/2023).

[15] K. Furmanová et al. "DockVis: Visual Analysis of Molecular Docking Trajectories". In: *Computer Graphics Forum* 39.6 (2020), pp. 452–464. DOI: https://doi.org/10.1111/cgf.14048. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.14048. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cgf.14048.

[16] J. Polanski. “4.14 - Chemoinformatics”. In: *Comprehensive Chemometrics*. Ed. by Steven D. Brown, Romá Tauler, and Beata Walczak. Oxford: Elsevier, 2009, pp. 459–506. ISBN: 978-0-444-52701-1. DOI: https://doi.org/10.1016/B978-044452701-1.00006-5. URL: https://www.sciencedirect.com/science/article/pii/B9780444527011000065.

[17] B. K. Shoichet, I. D. Kuntz, and D. L. Bodian. “Molecular docking using shape descriptors”. In: *Journal of Computational Chemistry* 13 (1992).

[18] M. K. Nguyen, L. Jaillet, and S. Redon. “ART–RRT: As-Rigid-As-Possible search for protein conformational transition paths”. en. In: *Journal of Computer-Aided Molecular Design* 33.8 (Aug. 2019), pp. 705–727. ISSN: 0920-654X, 1573-4951. DOI: 10.1007/s10822-019-00216-w. URL: http://link.springer.com/10.1007/s10822-019-00216-w (visited on 04/13/2023).

[19] M. K. Nguyen. “Efficient exploration of molecular paths from As-Rigid-As-Possible approaches and motion planning methods”. en. In: (2018).

[20] N. Lindow, D. Baum, and H.-C. Hege. “Voronoi-Based Extraction and Visualization of Molecular Paths”. en. In: *IEEE Transactions on Visualization and Computer Graphics* 17.12 (Dec. 2011), pp. 2025–2034. ISSN: 1077-2626. DOI: 10.1109/TVCG.2011.259. URL: http://ieeexplore.ieee.org/document/6064966/ (visited on 03/28/2023).

[21] Schrödinger, LLC. “The PyMOL Molecular Graphics System, Version 1.8”. 2015.

[22] I. W. M. Smith. “Chapter 3 - Molecular collision dynamics”. In: *Kinetics and Dynamics of Elementary Gas Reactions*. Ed. by Ian W.M. Smith. Butterworths Monographs in Chemistry and Chemical Engineering. Butterworth-Heinemann, 1980, pp. 59–109. ISBN: 978-0-408-70790-9. DOI: https://doi.org/10.1016/B978-0-408-70790-9.50008-6. URL: https://www.sciencedirect.com/science/article/pii/B9780408707909500086.

[23] L. Lu and S. Benyahia. “Chapter Two - Advances in Coarse Discrete Particle Methods With Industrial Applications”. In: *Bridging Scales in Modelling and Simulation of Non-Reacting and Reacting Flows. Part II*. Ed. by Alessandro Parente and Juray De Wilde. Vol. 53. Advances in Chemical Engineering. Academic Press, 2018, pp. 53–151. DOI: https://doi.org/10.1016/bs.ache.2017.12.001. URL: https://www.sciencedirect.com/science/article/pii/S0065237717300522.

[24] E. W. Dijkstra. “A Note on Two Problems in Connexion with Graphs”. In: *Numer. Math.* 1.1 (1959), 269–271. ISSN: 0029-599X. DOI: 10.1007/BF01386390. URL: https://doi.org/10.1007/BF01386390.

[25] J. L. Bentley. “Multidimensional Binary Search Trees Used for Associative Searching”. In: *Commun. ACM* 18.9 (1975), 509–517. ISSN: 0001-0782. DOI: 10.1145/361002.361007. URL: https://doi.org/10.1145/361002.361007.

[26] J.J. Kuffner and S.M. LaValle. “RRT-connect: An efficient approach to single-query path planning”. In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 2. 2000, 995–1001 vol.2. DOI: 10.1109/ROBOT.2000.844730.

[27] X. Tang and F. Chen. “Robot Path Planning Algorithm based on Bi-RRT and Potential Field”. In: *2020 IEEE International Conference on Mechatronics and Automation (ICMA)*. 2020, pp. 1251–1256. DOI: 10.1109/ICMA49215.2020.9233539.

[28] P. F. Ash and E. D. Bolker. “Generalized Dirichlet tessellations”. In: *Geometriae Dedicata* 20 (1986), pp. 209–243.

[29] V. Vonásek. “Motion planning of 3D objects using Rapidly Exploring Random Tree guided by approximate solutions”. In: *2018 IEEE 23rd International Conference on Emerging Technologies and Factory Automation (ETFA)*. Vol. 1. 2018, pp. 713–720. DOI: 10.1109/ETFA.2018.8502446.

[30] V. Vonásek et al. “RRT-path — A guided rapidly exploring random tree”. In: *Robot motion and control (RoMoCo)*. London: Springer London, 2009, pp. 307–316. ISBN: 978-1-84882-985-5. DOI: https://doi.org/10.1007/978-1-84882-985-5_28.

[31] M. Minařík. “Improving Sampling-Based Motion Planning Using Library of Trajectories”. Prague, Czech Republic: Czech Technical University in Prague, 2021.

[32] J. Denny et al. “On the theory of user-guided planning”. en. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 7 citations (Crossref) [2023-05-03]. Daejeon, South Korea: IEEE, Oct. 2016, pp. 4794–4801. ISBN: 978-1-5090-3762-9. DOI: 10.1109/IROS.2016.7759704. URL: http://ieeexplore.ieee.org/document/7759704/ (visited on 03/28/2023).

[33] M. F. Aslan, A. D., and K. Sabanci. “Goal distance-based UAV path planning approach, path optimization and learning-based path estimation: GDRRT*, PSO-GDRRT* and BiLSTM-PSO-GDRRT*”. In: *Applied Soft Computing* 137 (2023), p. 110156. ISSN: 1568-4946. DOI: https://doi.org/10.1016/j.asoc.2023.110156. URL: https://www.sciencedirect.com/science/article/pii/S1568494623001746.

[34] A. Graves and J. Schmidhuber. “Framewise phoneme classification with bidirectional LSTM and other neural network architectures”. In: *Neural Networks* 18.5 (2005). IJCNN 2005, pp. 602–610. ISSN: 0893-6080. DOI: https://doi.org/10.1016/j.neunet.2005.06.042. URL: https://www.sciencedirect.com/science/article/pii/S0893608005001206.

[35] S. Karaman and E. Frazzoli. “Sampling-based algorithms for optimal motion planning”. In: *The International Journal of Robotics Research* 30.7 (2011), pp. 846–894. DOI: 10.1177/0278364911406761. eprint: https://doi.org/10.1177/0278364911406761. URL: https://doi.org/10.1177/0278364911406761.

[36] S. Karaman et al. “Anytime Motion Planning using the RRT*”. In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1478–1483. DOI: 10.1109/ICRA.2011.5980479.

[37] M. R. Bonyadi and Z. Michalewicz. “Particle Swarm Optimization for Single Objective Continuous Space Problems: A Review”. en. In: *Evolutionary Computation* 25.1 (Mar. 2017), pp. 1–54. ISSN: 1063-6560, 1530-9304. DOI: 10.1162/EVCO_r_00180. URL: https://direct.mit.edu/evco/article/25/1/1-54/1040 (visited on 05/05/2023).

[38] A. Z. Nasrollahy and H. H. S. Javadi. “Using Particle Swarm Optimization for Robot Path Planning in Dynamic Environments with Moving Obstacles and Target”. In: *2009 Third UKSim European Symposium on Computer Modeling and Simulation*. 2009, pp. 60–65. DOI: 10.1109/EMS.2009.67.

[39] T. F. Iversen and L.-P. Ellekilde. “Kernel density estimation based self-learning sampling strategy for motion planning of repetitive tasks”. In: *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2016, pp. 1380–1387. DOI: 10.1109/IROS.2016.7759226.

[40] I. J. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016.

[41] D. Kingma and J. Ba. “Adam: A Method for Stochastic Optimization”. In: *International Conference on Learning Representations* (Dec. 2014).

[42] A. Yershova and S. M. LaValle. “Improving Motion-Planning Algorithms by Efficient Nearest-Neighbor Searching”. In: *IEEE Transactions on Robotics* 23.1 (2007), pp. 151–157. DOI: 10.1109/TRO.2006.886840.

[43] *Rapid - robust and accurate polygon interference detection system*. Accessed: 2023-05-12. URL: http://gamma.cs.unc.edu/OBB/.

[44] S. Gottschalk, M. C. Lin, and D. Manocha. “OBBTree: a hierarchical structure for rapid interference detection”. en. In: *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH ’96*. New York, NY, USA: ACM Press, 1996, pp. 171–180. ISBN: 978-0-89791-746-9. DOI: 10.1145/237170.237244. URL: http://portal.acm.org/citation.cfm?doid=237170.237244 (visited on 05/05/2023).

[45] J. Newman et al. “Haloalkane dehalogenases: structure of a Rhodococcus enzyme”. en. In: *Biochemistry* 38.49 (Dec. 1999), pp. 16105–16114.

[46] Y. Bourne, P. Taylor, and P. Marchot. “Acetylcholinesterase inhibition by fasciculin: crystal structure of the complex”. en. In: *Cell* 83.3 (Nov. 1995), pp. 503–512.

[47] I. A. Şucan, M. Moll, and L. E. Kavraki. "The Open Motion Planning Library". In: *IEEE Robotics & Automation Magazine* 19.4 (2012). https://ompl.kavrakilab.org, pp. 72–82. DOI: 10.1109/MRA.2012.2205651.

[48] M. Moll, I. A. Şucan, and L. E. Kavraki. "Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization". In: *IEEE Robotics & Automation Magazine* 22.3 (2015), pp. 96–102. DOI: 10.1109/MRA.2015.2448276.

[49] J. Bezanson et al. "Julia: A fresh approach to numerical computing". In: *SIAM Review* 59.1 (2017), pp. 65–98. DOI: 10.1137/141000671. URL: https://epubs.siam.org/doi/10.1137/141000671.

[50] D. Hsu, J.-C. Latombe, and R. Motwani. "Path planning in expansive configuration spaces". In: *Proceedings of International Conference on Robotics and Automation*. Vol. 3. 1997, 2719–2726 vol.3. DOI: 10.1109/ROBOT.1997.619371.

[51] I. A. Şucan and L. E. Kavraki. "Kinodynamic Motion Planning by Interior-Exterior Cell Exploration". In: *Algorithmic Foundation of Robotics VIII: Selected Contributions of the Eight International Workshop on the Algorithmic Foundations of Robotics*. Ed. by G. S. Chirikjian et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 449–464. ISBN: 978-3-642-00312-7. DOI: 10.1007/978-3-642-00312-7_28. URL: https://doi.org/10.1007/978-3-642-00312-7_28.

[52] R. Bohlin and L.E. Kavraki. "Path planning using lazy PRM". In: *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*. Vol. 1. 2000, 521–528 vol.1. DOI: 10.1109/ROBOT.2000.844107.

[53] R. Bohlin and L. E. Kavraki. "A Randomized Algorithm for Robot Path Planning Based on Lazy Evaluation". In: *Handbook on Randomized Computing*. Kluwer Academic Publishers, 2001, pp. 221–249.

[54] G. Sanchez-Ante and J.-C. Latombe. "A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking". In: Jan. 2001, pp. 403–417.

[55] B. Gipson, M. Moll, and L. E. Kavraki. "Resolution Independent Density Estimation for motion planning in high-dimensional spaces". In: *2013 IEEE International Conference on Robotics and Automation*. 2013, pp. 2437–2443. DOI: 10.1109/ICRA.2013.6630908.

[56] M. Innes et al. "Fashionable Modelling with Flux". In: *CoRR* abs/1811.01457 (2018). arXiv: 1811.01457. URL: https://arxiv.org/abs/1811.01457.

[57] M. Innes. "Flux: Elegant Machine Learning with Julia". In: *Journal of Open Source Software* (2018). DOI: 10.21105/joss.00602.