Bachelor Project

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Cybernetics

Improving Path Planning Methods Using Machine Learning

Artyom Tsoy

Supervisor: Ing. Vojtěch Vonásek, Ph.D.
Field of study: Cybernetics and Robotics
May 2024

Acknowledgements

I am sincerely thankful to my supervisor Ing. Vojtěch Vonásek, Ph.D., for his continuous support and invaluable feedback during my Bachelor project. His guidance and support have been instrumental in navigating the challenges and refining the outcomes of this research endeavor. Additionally, I am truly grateful for the opportunity to study at CTU University, where I have gained invaluable knowledge and skills that have greatly contributed to the successful completion of this project.

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. I utilized artificial intelligence for grammar correction.

Prague, May , 2024

Signature

Abstract

Path planning in robotics plays a critical role in enabling autonomous systems to navigate in complex environments efficiently. This project focuses on enhancing traditional sampling-based path planning methods, such as Rapidly-exploring Random Trees (RRT) and RRT*, by integrating machine learning techniques. The objective is to improve the efficiency and adaptability of path planning algorithms through the utilization of learned information about the environment.

To validate the efficacy of the proposed approach, comparisons were conducted against existing methods. Through experimentation and analysis, the performance and adaptability of the developed algorithms were assessed, highlighting their potential to outperform traditional techniques and contribute to the field of path planning.

Keywords: Path planning optimization, Sampling-based methods enhancement, RRT and RRT* algorithms, Environmental adaptation, Learning-based planning, Machine learning

Supervisor: Ing. Vojtěch Vonásek, Ph.D.

Abstrakt

Plánování cest v robotice hraje kritickou roli při umožňování autonomním systémům efektivně navigovat složitými prostředími. Tento projekt se zaměřuje na zlepšení tradičních metod plánování cest založených na vzorkování, jako jsou Rapidly-exploring Random Trees (RRT) a RRT*, prostřednictvím integrování technik strojového učení. Cílem je zlepšit efektivitu a přizpůsobivost algoritmů plánování cest využitím informací o prostředí získaných ze strojového učení.

Pro ověření účinnosti navrženého přístupu byly provedeny srovnání s existujícími metodami. Skrz experimentování a analýzu byly zhodnoceny výkonnost a přizpůsobivost vyvinutých algoritmů, které zdůraznily jejich potenciál překonat tradiční techniky a přispět k oblasti plánování cest.

Klíčová slova: Optimalizace plánování cest, Zlepšení metod založených na vzorkování, Algoritmy RRT a RRT*, Adaptace na prostředí, Plánování založené na učení, Strojové učení

Překlad názvu: Využití strojového učení v úloze plánování pohybu

Contents

Figures

Tables

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Tsoy Artyom** Personal ID number: **504451**

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

Department / Institute: **Department of Cybernetics**

Study program: **Cybernetics and Robotics**

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Improving Path Planning Methods Using Machine Learning

Bachelor’s thesis title in Czech:

Využití strojového učení v úloze plánování pohybu

Guidelines:

1. Study path planning problem [1] and get familiar with sampling-based path planning methods (e.g., RRT and RRT*) [1,2,3]. Implement basic RRT and RRT* in C/C++ or Python. Get familiar with neural networks [6].
2. Implement a machine-learning method for estimating suitable sampling regions for RRT-based planners (use e.g. [4,7]). The method should predict where to draw random samples based on the state of the environment and goal. Implement the method for 2D configuration space.
3. Extend the method from task 2) to 3D and 6D configuration space.
4. Compare all implemented methods from tasks 2) and 3) with related work using the OMPL library [5].

Bibliography / sources:

[1] LaValle, Steven M. Planning Algorithms. 1st ed. Cambridge University Press, 2006. https://doi.org/10.1017/CBO9780511546877.
[2] LaValle, Steven. "Rapidly-exploring random trees: A new tool for path planning." Research Report 9811 (1998).
[3] Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. The international journal of robotics research, 30(7), 846-894.
[4] O. Arslan and P. Tsiotras, "Machine learning guided exploration for sampling-based motion planning algorithms," 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Hamburg, Germany, 2015, pp. 2646-2652, doi: 10.1109/IROS.2015.7353738.
[5] Mark Moll, Ioan A. Şucan, Lydia E. Kavraki, Benchmarking Motion Planning Algorithms: An Extensible Infrastructure for Analysis and Visualization, IEEE Robotics & Automation Magazine, 22(3):96–102, September 2015.
[6] Deep Learning, Ian Goodfellow and Yoshua Bengio and Aaron Courville, MIT Press, 2016.
[7] A. H. Qureshi, Y. Miao, A. Simeonov and M. C. Yip, "Motion Planning Networks: Bridging the Gap Between Learning-Based and Classical Motion Planners," in IEEE Transactions on Robotics, vol. 37, no. 1, pp. 48-66, Feb. 2021, doi: 10.1109/TRO.2020.3006716.

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: **23.01.2024** Deadline for bachelor thesis submission: _____

Assignment valid until: **21.09.2025**

Ing. Vojtěch Vonásek, Ph.D.	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	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

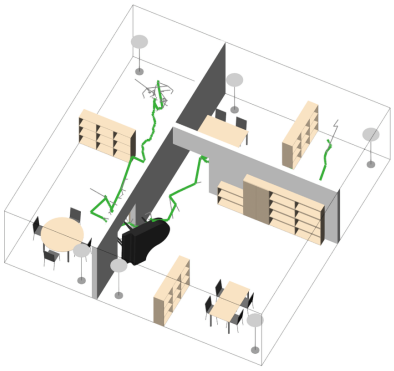
_____ Student's signature

Chapter 1

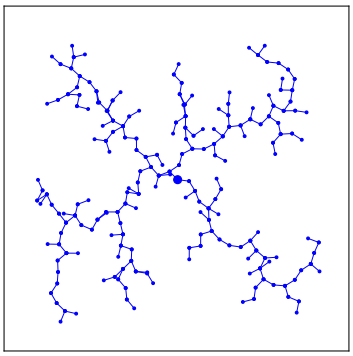
Introduction

Path planning is a fundamental challenge in the field of robotics, essential for enabling autonomous systems to navigate through complex environments efficiently. It allows vehicles or robots to find the shortest, obstacle-free route from their starting point to their destination. This route, often represented as a series of states comprising position and orientation or as waypoints, guides the vehicle or robot towards its goal effectively. One of the famous examples is the Piano Movers Problem. Path planning involves determining the optimal route or path for transporting a piano from one location to another while navigating through various obstacles and constraints. In Figure 1.1a, the illustration of the Piano Movers Problem is depicted.

Traditional path planning methods, such as Rapidly-exploring Random Trees (RRT) [15] and its enhanced variant RRT* [12], have been widely used due to their effectiveness in handling high-dimensional configuration spaces. The configuration space, often denoted as $\mathcal{C}$-space, is a fundamental concept in robotics and motion planning. It represents all possible configurations or states that a robot or system can occupy within its environment.



(a) : The Piano Movers Problem. Image courtesy of [18].



(b) : Rapidly-exploring Random Trees examples.

Figure 1.1: Figure 1.1a illustrates an example of a path planning problem, along with a representation of the Rapidly-exploring Random Tree (RRT) algorithm in Figure 1.1b.

To enhance clarity within the text, it is crucial to briefly explain how RRT works: The algorithm incrementally builds a tree structure from an initial configuration towards randomly sampled configurations in the configuration space. At each iteration, a new configuration, known as a random sample, is generated within the configuration space. The algorithm then extends the existing tree towards this random sample by selecting the nearest node (configuration) in the tree to the sample and moving towards it in small increments. The key idea behind RRT is to rapidly explore the configuration space by growing the tree towards unexplored regions. This exploration strategy ensures that the algorithm efficiently covers the space while avoiding costly computation. RRT continues this process iteratively, gradually expanding the tree towards the goal configuration. As the algorithm progresses, the tree becomes denser in regions that are closer to the goal, eventually converging to a path from the start to the goal configuration. In Figure 1.1b, simple examples of how RRT looks can be seen.

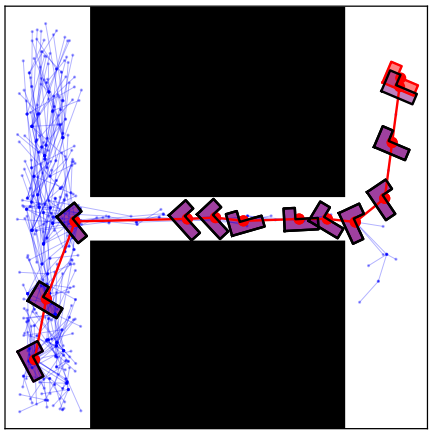


Figure 1.2: Illustration of the narrow passage where the aim is to maneuver a purple object through the limited space between black obstacles to reach the red goal configuration.

However, with the increasing complexity of real-world environments, such as systems with many obstacles or narrow passages, as illustrated in Figure 1.2, there is a growing need to enhance these methods to improve their efficiency and adaptability. Efficiency in path planning denotes the capability of the algorithms to generate feasible paths within a reasonable timeframe, while adaptability refers to its ability to dynamically adjust strategies in response to changes in environmental conditions or task requirements.

1.1 Machine Learning

Understanding the concept of machine learning is pivotal for comprehending its utilization within this thesis. Therefore, this section will offer a brief explanation of machine learning without delving into mathematical definitions and complex explanations. Main idea and inspiration for this section is drawn

from [7].

Machine learning is a field of artificial intelligence that focuses on developing algorithms capable of learning patterns and making predictions from data without being explicitly programmed. It enables computers to learn from past experiences and improve their performance over time.

Most machine learning algorithms can be categorized into supervised learning and unsupervised learning. In supervised learning, the algorithm receives guidance from an instructor or teacher, who provides the target output for each input example. Conversely, unsupervised learning involves no external guidance, requiring the algorithm to derive insights from the data independently.

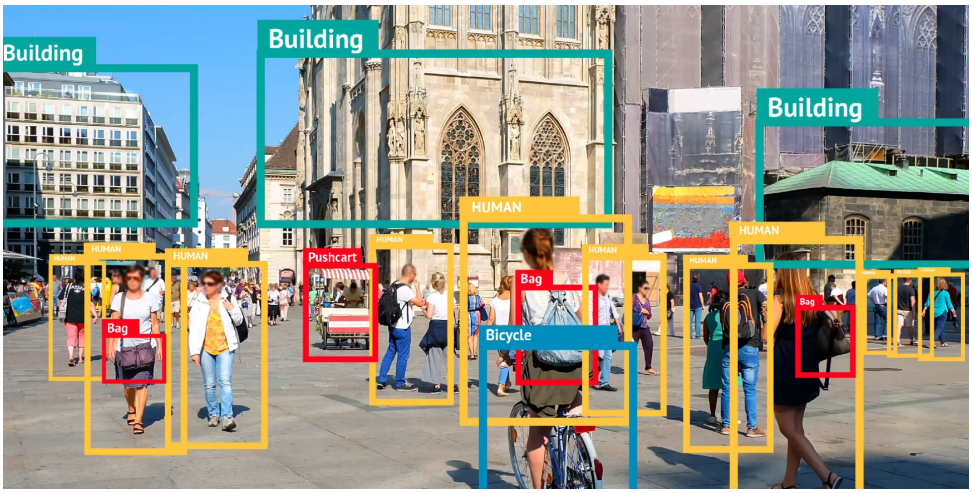


Figure 1.3: An example of an object classification problem, where the program classifies objects in the photo into their respective classes. Image courtesy of [9].

Supervised learning tasks typically include classification, regression, and density estimation or probability mass function estimation. In classification tasks, the algorithm classifies input data into predefined categories or classes. For instance, object recognition can be considered a classification task, where the computer program identifies objects in images and assigns them to specific categories. In Figure 1.3, an example of object recognition is depicted.

In regression tasks, the algorithm predicts a numerical value based on input data. For example, predicting house prices based on features like location, size, and amenities is a regression task.

Density estimation involves learning a function $p_{\text{model}} : \mathbb{R}^n \to \mathbb{R}$, where $p_{\text{model}}(x)$ represents a probability density function for continuous data or a probability mass function for discrete data. The algorithm learns the underlying data structure, identifying regions where examples are densely clustered and areas where they are less likely to occur. This understanding is crucial for effectively modeling the data distribution.

This thesis aims to improve path planning algorithms by using machine learning, including classification and density estimation techniques.

1.2 Thesis structure

Chapter 1 introduces the concepts of path planning and machine learning. It outlines the central objectives of this thesis, focusing on the integration of machine learning techniques to enhance path planning algorithms. Additionally, Chapter 1 provides a concise overview of the thesis structure.

Chapter 2 will delve deeper into the path planning problem, offering an understanding of essential terms and notations. Building on this foundation, the Chapter 2 will formulate the challenge of this thesis.

In Chapter 3, familiarity with related works, including RRT and RRT*, will be gained, which also contributes to better understanding the text.

Chapter 4 will be pivotal, as it will explain the machine learning methods employed in this study. Additionally, it will detail the integration of these methods into the sampling-based algorithm RRT* in 2D $\mathcal{C}$-space and its extension to 3D $\mathcal{C}$-space and 6D $\mathcal{C}$-space.

The effectiveness of the developed methods will be evaluated in Chapter 5 through comparisons with existing algorithms using the Open Motion Planning Library (OMPL) [20].

In Chapter 6, the thesis will be concluded by summarizing all results and findings.

Chapter 2

Task formulation

This chapter lays the foundation for better understanding the research objectives and motivations. Firstly, it will explain the path planning problem and provide a brief introduction to sampling-based path planning, including an overview of its definition, advantages, and disadvantages. Based on this, the main objective of the thesis is formulated. Subsequently, key terms related to path planning are elucidated to ensure clarity and comprehension. Finally, the chapter concludes by formulating the primary challenge addressed in the thesis.

2.1 Path planning problem

The path planning problem, also known as motion planning or the navigation problem, is a fundamental challenge in robotics. It involves finding a path that connects a start configuration to a goal configuration while satisfying specified constraints, such as avoiding collisions.

The path planning problem and relevant terms will be explained using the Piano Movers Problem. An illustration of this problem can be seen in Figure 1.1a. In this scenario, the task is to move a piano from one room to another trying not to collide with any objects.

The piano has six degrees of freedom (DoF), meaning its position in space can be described using six coordinates: three for position (x, y, z) and three for rotation $(yaw, roll, pitch)$. The position most often represents the coordinates of its center of mass or as a reference point of the robot. It can be stated that there a **6D $\mathcal{C}$-space** because to describe the configuration of the piano, we need six coordinates. These coordinates define configurations of the piano.

Our objective is to find a sequence of configurations that safely moves the piano (our **robot**) from its starting position to its destination without colliding with any objects in the room. Objects like tables, closets, walls represent **obstacles**, and the door represents **narrow passage**. These obstacles must be avoided. To avoid these obstacles, a sequence of configurations is sought. This sequence is referred to as a **path**. The problem of path planning can be

stated as finding a collision-free path from the initial point to the goal point.

2.2 Sampling-based path planning

Sampling-based path planning [14] is a computational approach utilized in robotics to generate feasible paths for autonomous agents navigating through complex environments. One of the sampling-based algorithms is the Rapidly-exploring Random Tree (RRT) [15].

Unlike traditional grid-based methods such as A* [10], which discretize the environment into a grid and search for paths within this grid, as illustrated in Figure 2.1a, sampling-based methods operate by randomly sampling points in the configuration space. In the case of the Piano Movers Problem, a random sample represents random values for $(x, y, z, yaw, roll, pitch)$. Subsequently, these sampled points are used to construct a graph. This probabilistic approach allows for efficient exploration of high-dimensional spaces and is particularly well-suited for environments with complex obstacles.

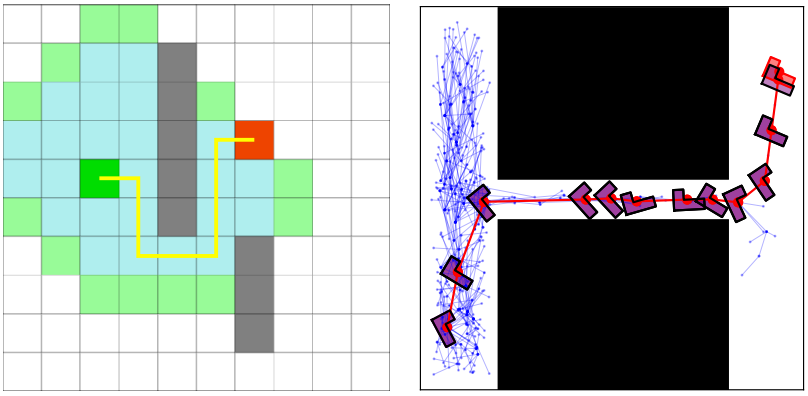
The main goal of sampling-based algorithms in path planning is to efficiently explore the configuration space to find feasible paths from a start configuration to a goal configuration, while avoiding obstacles.

2.3 Research Objectives

Sampling-based algorithms offer several advantages. They are flexible and can handle high-dimensional configuration spaces, making them suitable for complex environments with obstacles and constraints. Additionally, they are computationally efficient, focusing on sampling feasible configurations rather than exhaustive exploration.

However, sampling-based methods also have their disadvantages. Random sampling can create biases in certain areas of the configuration space, resulting in less-than-optimal paths or missing potential solutions. In highly cluttered environments or spaces with narrow passages, these algorithms may struggle to adequately explore the configuration space, resulting in incomplete coverage and suboptimal path generation. It can be seen in Figure 2.1b. The performance of sampling-based algorithms can be sensitive to various parameters, such as sampling density and collision checking threshold. Additionally, while they offer computational efficiency, they may not always generate the highest quality paths compared to other methods, especially in scenarios where path smoothness or optimality is crucial.

The research objectives of this thesis will concentrate on enhancing sampling-based algorithms by addressing their limitations in scenarios characterized by numerous obstacles or narrow passages. The primary goal is to enhance



(a) : A* algorithm. Image courtesy of [21].

(b) : Narrow passage in the 2D workspace with a 3D $\mathcal{C}$-space.

Figure 2.1: llustration of the A* algorithm realizes path-finding on grid map in Figure 2.1a and illustration of the narrow passage in Figure 2.1b, where the blue lines represent RRT* exploration in the workspace. It can be seen in Figure 2.1b that numerous samples were generated on the left side, resulting in incomplete coverage of the space. In summary, a large number of samples were generated in inappropriate locations.

sampling-based algorithms through the integration of machine learning techniques to more effectively generate points. The proposed approach will improve random sampling by selectively sampling points only from obstacle-free spaces, thereby enhancing the efficiency and adaptability of the algorithms.

2.4 Components of the Path Planning Problem

Let us begin by defining the configuration space, denoted as $\mathcal{C}$. It serves as a critical component of motion planning, representing all possible states of system. This space is a subset of $\mathbb{R}^d$, where $d \geq 2$ and $d \in \mathbb{N}$. Within $\mathcal{C}$, there are three primary regions:

- **Obstacle Space ($\mathcal{C}_{\text{obs}}$):** This region encompasses areas occupied by obstacles, constraining the motion of our system.
- **Obstacle-Free Space ($\mathcal{C}_{\text{free}}$):** Defined as the complement of $\mathcal{C}_{\text{obs}}$ within $\mathcal{C}$, this space allows unrestricted movement for our system.
- **Goal Region ($\mathcal{C}_{\text{goal}}$):** Representing the destination our system aims to reach. This region must be within the free space $\mathcal{C}_{\text{goal}} \subseteq \mathcal{C}_{\text{free}}$, as the system cannot reach regions outside of the free space. Within this region, there must exist a goal configuration, denoted as $q_{\text{goal}} \in \mathcal{C}_{\text{goal}}$, which represents the optimal configuration. However, it is notable that in certain cases, the goal region may be represented just by q_{goal}.

Path planning aim to find a feasible path from the initial configuration $q_{\text{start}} \in \mathcal{C}_{\text{free}}$ to a goal configuration $q_{\text{goal}} \in \mathcal{C}_{\text{goal}}$, ensuring avoidance of

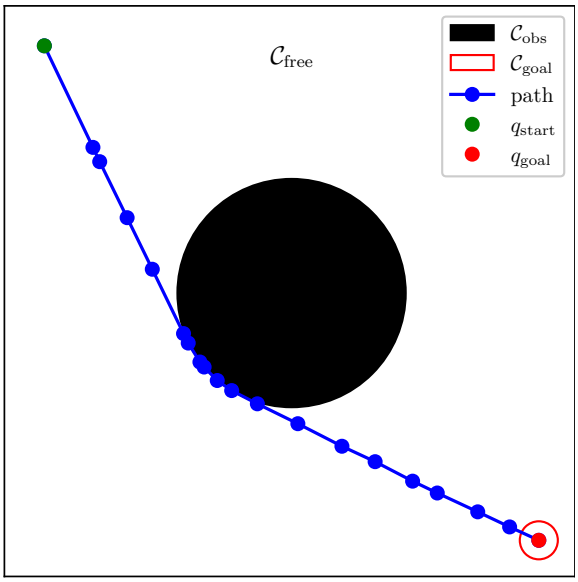


Figure 2.2: 2D workspace with a 2D $\mathcal{C}$-space.

regions occupied by obstacles ($\mathcal{C}_{\text{obs}}$).

To represent the connectivity between configurations within $\mathcal{C}_{\text{free}}$, a graph $G = (V, E)$ is utilized. Here, V denotes the finite set of vertices, each corresponding to a configuration point, and $E \subseteq V \times V$ denotes the set of edges, signifying connections between configurations.

In Figure 2.2, a simple 2D workspace is depicted, wherein the robot is represented by a point. A workspace is the area within which a robot operates, and in this context, the configuration of the robot can be described by two coordinates, such as $q = (x, y)$. Therefore, it has a 2D $\mathcal{C}$-space. The start configuration, denoted by q_{start}, is marked in green, while the goal configuration, denoted by q_{goal}, is marked in red. The black circle at the center of the space represents the $\mathcal{C}_{\text{obs}}$. The red ring represents the boundary of the $\mathcal{C}$-space, with points inside it corresponding to the $\mathcal{C}_{\text{goal}}$. The blue trajectory, comprising vertices and edges, illustrates the path through the configuration space. These vertices, represented as blue circles, illustrates configurations or nodes along the path.

2.5 The Challenge

The challenge of this thesis is to reduce the number of sampling points, or iterations, required to converge on the optimal path. This will be achieved by learning the configuration space and generating points within $\mathcal{C}_{\text{free}}$ and enhance the overall efficiency of the motion planning algorithm. Further information is provided in Chapter 4.

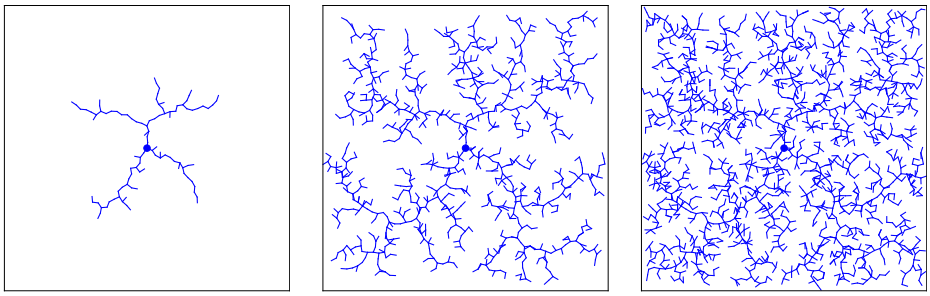
Chapter 3

Related works

In this chapter, an overview of various path planning methods will be presented. This exploration is crucial for gaining a deeper understanding of how sampling algorithms function and how they can be enhanced. Building upon this understanding, the main idea of improvement was implemented, and the foundation for this implementation was laid upon the knowledge of sampling-based algorithms. These methods will be described and analyzed to provide insights into their effectiveness and limitations.

3.1 Rapidly-exploring Random Tree (RRT)

The Rapidly-exploring Random Tree (RRT) [15] algorithm is a versatile and widely used sampling-based motion planning algorithm in robotics. A pseudocode is shown in Algorithm 1. Its appeal lies in its efficiency in exploring high-dimensional configuration spaces, making it suitable for various robotic applications. RRT excels in systems with non-holonomic constraints, which limit a robot's movement beyond just its position and orientation. These constraints often involve restrictions on how the robot can change its velocity or direction. In the Fig. 3.1, the progressive expansion in 2D $\mathcal{C}$-space of the RRT can be seen.



(a) : 100 iterations **(b) :** 1000 iterations **(c) :** 2000 iterations

Figure 3.1: Example of RRT expansion in 2D $\mathcal{C}$-space.

Simplicity and effectiveness of the RRT algorithm have made it a popular choice among researchers and practitioners in the field. This highlights the

importance of explainig its operational phases.

In the initialization phase, the algorithm begins by establishing a tree structure with only the initial state, which represents the root node. This initial state represents the starting configuration of the robot or system.

During the expansion phase, the algorithm iteratively executes the following steps until a predetermined number of iterations or the goal state is reached: generating a random state within the configuration space, identifying the nearest node in the existing tree to the randomly generated state, extending the tree towards the random state by incrementally advancing in its direction and creating a new node, and subsequently verifying whether the new node collides with any obstacles. If no collision occurs, the new node is added to the tree, thereby expanding the coverage of the configuration space.

After completion of each iteration, the algorithm conducts goal checking to determine if the goal state has been reached. Upon reaching the goal state, the algorithm terminates, and the path to the goal is extracted from the tree structure.

To extract a path to the goal, a hierarchical structure within the tree must be established. For every configuration added into the tree, a parent configuration is assigned, typically being the nearest configuration in the tree. This parent configuration acts as the reference point from which the new configuration is generated or "steered" towards. Consequently, each configuration has only one parent but can have many children, thereby forming a hierarchical relationship within the tree.

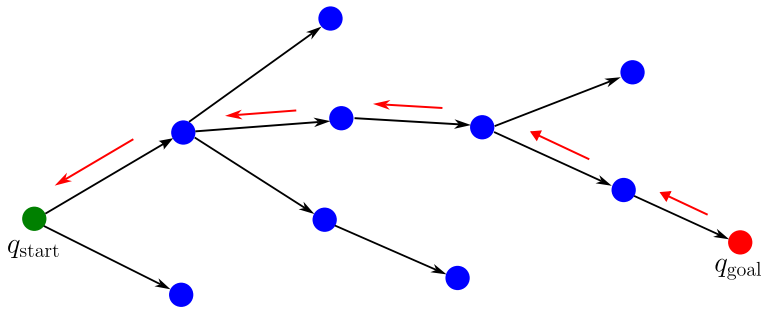


Figure 3.2: An illustration of a hierarchical structure, where black arrows indicate the parent-child relationships, with the origin of the arrow representing a parent node and its destination denoting a child node. Additionally, red arrows illustrate a retracing process, indicating the movement from the goal configuration back to the start configuration by moving through parent nodes.

Upon reaching the goal configuration, the path to the start configuration is retraced by traversing from the goal configuration to its parent and then recursively traversing through each parent until the start configuration is reached. In Figure 3.2, an illustration of this process is presented. This

process capitalizes on the parent-child relationships established during the tree construction phase, enabling systematic navigation from the goal configuration back to the start configuration, and thereby facilitating path extraction.

The RRT algorithm is known for its relatively straightforward implementation, yet it effectively explores the configuration space. This efficiency refers to its ability to rapidly expand the search space, thereby quickly accessing various regions of the configuration space. However, despite these advantages, RRTs are not without their limitations. One notable drawback is their tendency towards suboptimality. Unlike certain other algorithms, basic RRTs do not guarantee the discovery of the optimal path to the goal. Consequently, the path found may not be the shortest or most efficient route. Furthermore, RRTs often necessitate frequent collision checks, particularly in environments characterized by clutter, leading to a notable increase in computational time.

Algorithm 1: Rapidly-exploring Random Tree (RRT)

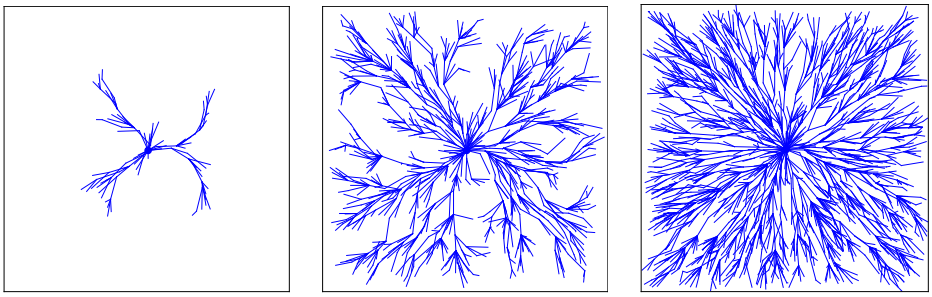
Data: Start configuration q_{start}, Number of iterations K
Result: RRT tree T

```
Initialize tree T with root q_start;
for k ← 1 to K do
    Generate random configuration q_rand ∈ C;
    Find nearest node q_near in T to q_rand;
    Steer towards q_rand to get new node q_new;
    if ObstacleFree(q_near, q_new) then
        Add q_new to T with an edge from q_near;
        if GoalFound(q_new) then
            break;
return T;
```

3.2 Rapidly-exploring Random Tree star (RRT*)

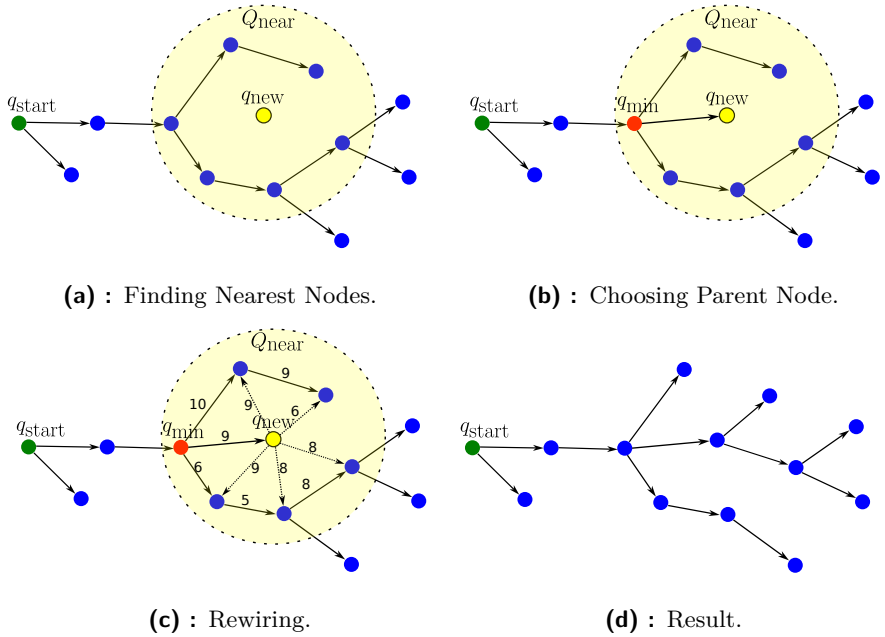
The Rapidly-exploring Random Tree Star (RRT*) [12] also known as Optimal RRT is an extension of the original RRT algorithm designed to enhance the efficiency and optimality of path planning in robotics. A pseudocode is shown in Algorithm 2. RRT* incorporates a rewiring step that allows the algorithm to dynamically adjust the tree structure, thereby promoting the discovery of more optimal paths. In the Fig. 3.3, the progressive expansion in 2D $\mathcal{C}$-space of the RRT* can be seen.

Similar to RRT, RRT* begins with an initialization phase and continues with an expansion phase. However, the key enhancement in RRT* lies in the **Rewiring phase**, aimed at improving the optimality of paths discovered. This phase can be divided into three parts: Finding Nearest Nodes, Choosing Parent Node and Rewiring 3.4.



(a) : 100 iterations **(b) :** 1000 iterations **(c) :** 2000 iterations

Figure 3.3: Example of RRT* expansion in 2D $\mathcal{C}$-space.



(a) : Finding Nearest Nodes. **(b) :** Choosing Parent Node.

(c) : Rewiring. **(d) :** Result.

Figure 3.4: Illustration of the **Rewiring phase**: The green circle denotes the initial configuration (q_{start}), while the yellow circle represents the new node to be added (q_{new}), along with its transparent yellow circular area with certain radius (Q_{near}). Figure 3.4a displays the nodes within this circular area. A parent node for the new node, identified by the red circle (q_{min}), can be seen in Figure 3.4b. Figure 3.4c exemplifies the rewiring step, with numerical values indicating the lengths of respective paths from the new node to nearby nodes. Finally, Figure 3.4d shows the final appearance of the rewired tree.

Finding Nearest Nodes step involves identifying nearby nodes within a certain radius from the new node. After identifying nearby nodes, the algorithm proceeds to the Choosing Parent Node step. Here, it calculates cost for each node relative to the new node and selects the one with the lowest cost. This cost evaluation considers both the path from the root to the current node and the path from the current node to the nearby node. It typically reflects the path length or other user-defined metrics. This step is crucial because, unlike in RRT where the algorithm simply identifies the nearest node, the goal is

to find the most optimal node, which may not always be the nearest one. After adding a new node to our hierarchical tree structure, the Rewiring step initiates. Similar to the Choosing Parent Node step, the algorithm evaluates the costs of all possible paths from our new node to all nearest nodes. If a new path reduces the cost of the node, the algorithm removes the old path to the node and adds the new path. During this process, the rewired node gets a new parent, and the old parent, from which the old path originated, may no longer have a child.

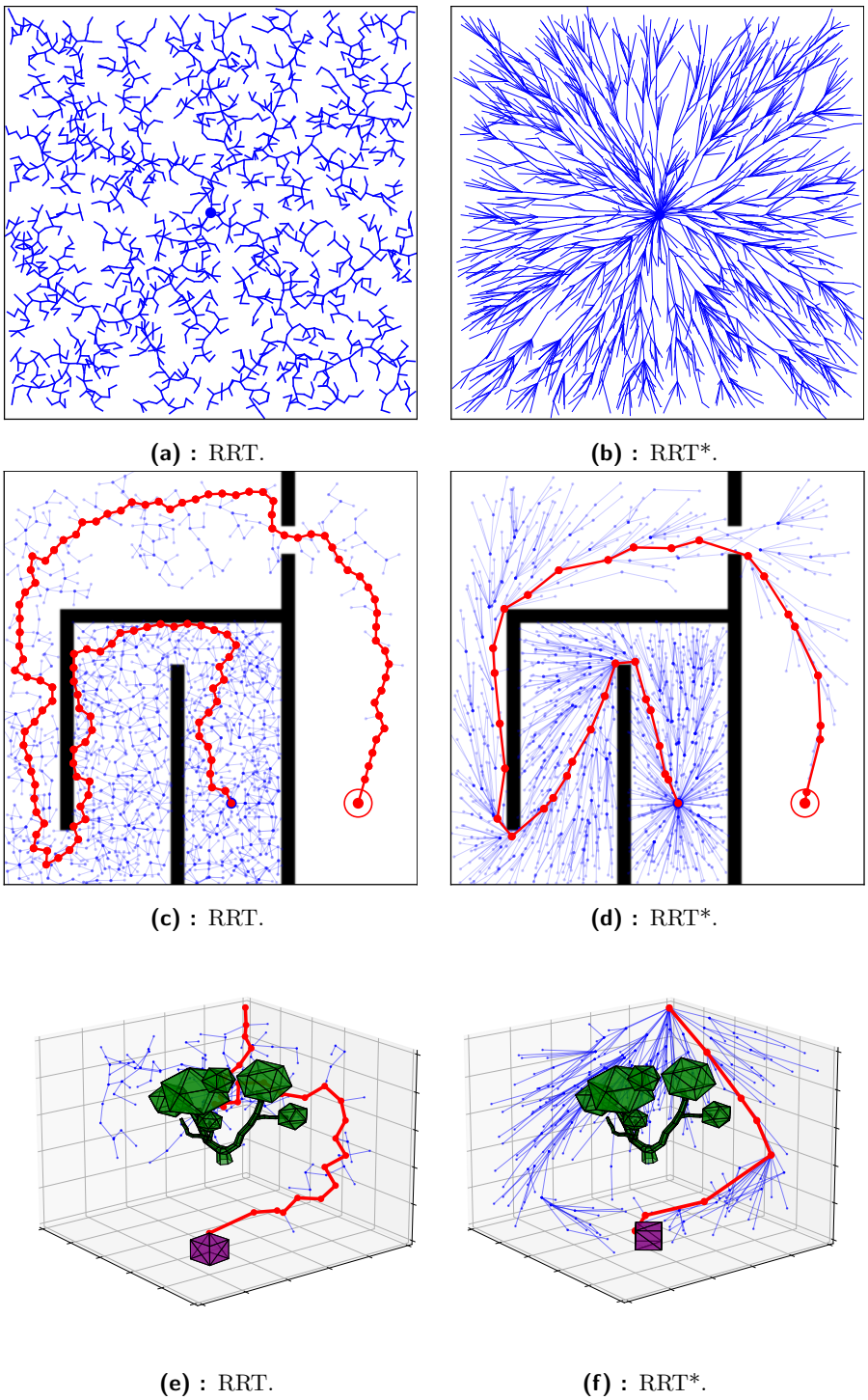
Through these iterative phases, RRT* continuously improves the tree structure, resulting in progressively optimal and efficient paths. By strategically rewiring the tree, RRT* effectively converges towards an optimal solution while efficiently exploring the configuration space.

The RRT* algorithm offers improved optimality over the original RRT, making it particularly useful when precision and efficiency are crucial considerations in robotic path planning. As illustrated in Figure 3.5, RRT* typically generates more direct paths with fewer unnecessary zigzags compared to RRT, thereby enhancing path quality. However, RRT* is more computationally complex, especially compared to RRT, which can lead to increased time consumption.

Algorithm 2: Rapidly-exploring Random Tree Star (RRT*)

Data: Start configuration q_{start}, Number of iterations K, Radius r
Result: RRT* tree T

$T \leftarrow \text{InitializeTree}(q_{\text{start}})$;
for $k \leftarrow 1$ **to** K **do**
  $q_{\text{rand}} \leftarrow \text{RandomConfiguration}()$;
  $q_{\text{near}} \leftarrow \text{NearestNode}(T, q_{\text{rand}})$;
  $q_{\text{new}} \leftarrow \text{Steer}(q_{\text{near}}, q_{\text{rand}})$;
  if *ObstacleFree*(q_{near}, q_{new}) **then**
    $Q_{\text{near}} \leftarrow \text{NearNodes}(T, q_{\text{new}}, r)$;
    $q_{\text{min}} \leftarrow \text{ChooseParent}(Q_{\text{near}}, q_{\text{new}})$;
    $T \leftarrow \text{AddNode}(q_{\text{new}}, q_{\text{min}})$;
    $T \leftarrow \text{Rewire}(T, Q_{\text{near}}, q_{\text{new}})$;
return T;



(a) : RRT. **(b) :** RRT*.

(c) : RRT. **(d) :** RRT*.

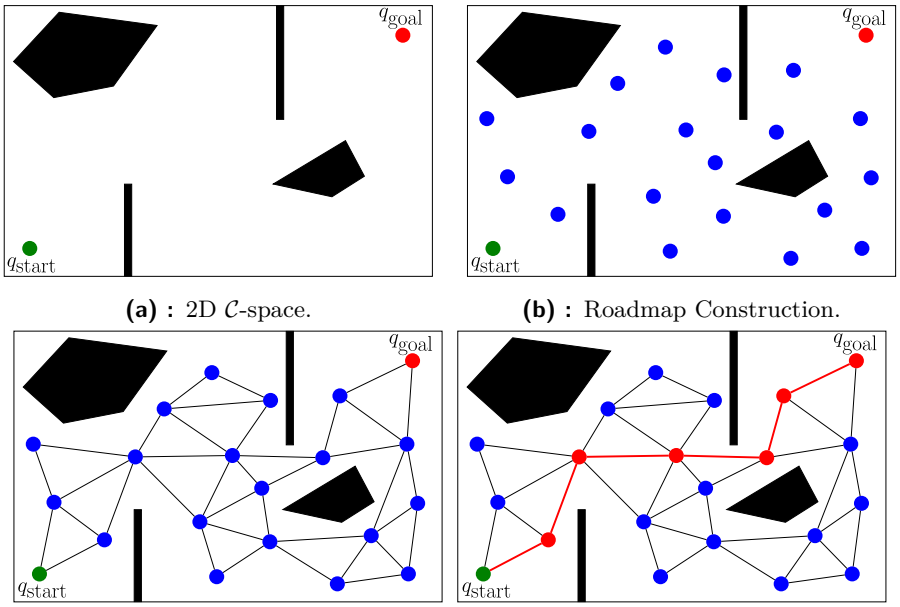
(e) : RRT. **(f) :** RRT*.

Figure 3.5: Comparison in 2D $\mathcal{C}$-space without obstacles in 3.5a and 3.5b with obstacles in 3.5c and 3.5d. Comparison in a 3D workspace with a 6D $\mathcal{C}$-space, with the robot represented as a purple cube and the obstacle represented as a green tree in 3.5e and 3.5f.

3.3 Probabilistic RoadMaps (PRM)

The Probabilistic Roadmap (PRM) [13] algorithm is a popular motion planning technique used in robotics to find feasible paths for robots operating in complex, high-dimensional spaces. Unlike traditional grid-based approaches, PRM operates by constructing a roadmap of the configuration space, which is then used to efficiently search for valid paths. The roadmap generated by PRM can be likened to a street network in a city. Just as roads connect various locations in a city, edges in the PRM roadmap connect different configurations in the configuration space. This analogy helps to conceptualize how the PRM algorithm constructs a network of feasible paths for the robot to navigate through the environment.

The PRM algorithm consists of the following main steps: Roadmap Construction, Neighborhood Search, Edge Connection, and Path Planning. A pseudocode of construction step is shown in Algorithm 3.



(a) : 2D $\mathcal{C}$-space. **(b) :** Roadmap Construction.

(c) : Edge Conecting. **(d) :** Path Planning.

Figure 3.6: In the 2D workspace depicted in Figure 3.6a, with a corresponding 2D $\mathcal{C}$-space, obstacles are represented in black. The initial configuration (q_{start}) is indicated by a green circle, while the goal configuration (q_{goal}) is marked with a red circle. Additionally, randomly sampled valid configurations appear as blue circles 3.6b. The connections between these configurations can be seen in Figure 3.6c. The path discovered between the start and goal configurations is highlighted in red 3.6d.

Roadmap Construction involves randomly sampling configurations from the configuration space and checking each sampled configuration for collision with obstacles. Valid configurations are added to the roadmap. In the Neighborhood Search step, for each sampled configuration, a set of neighboring

configurations is identified. These neighbors serve as potential candidates for connecting edges in the roadmap. In the Edge Connection step, pairs of neighboring configurations are evaluated to determine if an edge should be added between them in the roadmap. This evaluation typically involves checking for collision-free paths between the configurations. Finally, in the Path Planning step, once the roadmap is constructed, path planning becomes a graph search problem. Algorithm such as A* [10] is commonly used to find the shortest path between the start and goal configurations in the roadmap. These steps can be seen in Figure 3.6.

PRM can handle complex, high-dimensional configuration spaces and environments with obstacles of varying shapes and sizes. Its straightforward implementation ensures efficient and fast solution finding. However, the quality of the roadmap depends heavily on the sampling strategy and collision checking accuracy. Poorly constructed roadmaps may lead to suboptimal or infeasible paths. Moreover, PRM encounters challenges when navigating through narrow passages, as the likelihood of sampling configurations capable of traversing tight gaps diminishes with the decreasing gap.

Despite its drawbacks, PRM remains a widely used and effective approach for motion planning in robotics, F particularly in scenarios with complex environments and obstacles.

Algorithm 3: Probabilistic Roadmap (PRM) construction step

Data: Start configuration q_{start}, Number of iterations K
Result: PRM graph G

```
1  Initialize graph G with vertices V and edges E;
2  for i ← 1 to K do
3      q_rand ← a randomly choosen free configuration;
4      Q_near ← a set of candidate neighbors of q_rand chosen from V;
5      V ← V ∪ {q_rand};
6      for q ∈ Q_near do
7          if not SameConnectedComponent(q_rand, q) then
8              E ← E ∪ {(q_rand, q)};
9              update connected components in G;
10 return G;
```

3.4 Informed RRT*

Informed RRT* [6] extends the capabilities of the RRT* algorithm by addressing the challenges inherent in exploring high-dimensional configuration spaces with greater efficiency. It achieves this by integrating heuristic guidance, typically through cost-to-go estimates or other informed sampling strategies, to direct exploration towards regions more likely to yield the optimal solution.

The cost-to-go refers to the estimated cost or distance from a given configuration to reach a goal configuration in a path planning problem. Building upon the fundamental principles of RRT*, Informed RRT* retains its core mechanics of incremental tree growth and probabilistic sampling. However, unlike traditional RRT*, which uniformly samples configurations at random, Informed RRT* employs informed sampling techniques to prioritize sample points in areas with lower estimated costs or higher likelihood of containing the optimal solution.

In Informed RRT*, hyperspheroid are often utilized as a representation of the cost-to-go heuristic. These hyperspheroids encapsulate regions of the configuration space that are likely to contain the optimal solution. By leveraging these hyperspheroids, This utilization of hyperspheroids enables Informed RRT* to efficiently explore high-dimensional spaces and converge towards the optimal solution more effectively than traditional RRT* algorithms.

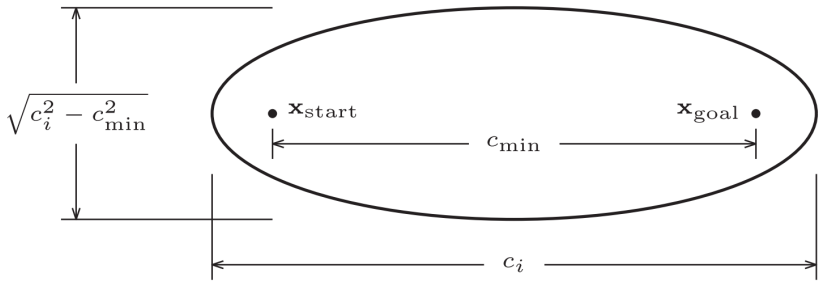


Figure 3.7: An example of a two-dimensional prolate hyperspheroid. Image courtesy of [6]

The prolate hyperspheroid (3.7) is defined by its focal points at start configuration $\mathbf{x}_{\text{start}}$ and goal configuration $\mathbf{x}_{\text{goal}}$, a transverse diameter of current solution cost c_i, and conjugate diameters of $\sqrt{(c_i^2 - c_{\min}^2)}$, where

$$c_{\min} := \|\mathbf{x}_{\text{goal}} - \mathbf{x}_{\text{start}}\|_2$$

is the theoretical minimum cost. Figure 3.8a presents an illustrative example of Informed RRT*.

3.5 RRT sharp (RRT#)

RRT sharp (RRT#) [1] is a sampling-based motion planning algorithm, which is based on Rapidly Exploring Random Graphs (RRG) [11]. The RRT# algorithm utilizes informed sampling techniques to bias the selection of sample points towards relevant regions with lower estimated costs or higher likelihoods of containing the optimal solution. This information enhances the convergence speed of the algorithm and enables more efficient exploration

of the obstacle-free space. Figure 3.8b presents an illustrative example of RRT$^{\#}$.

To find the relevant region, the following formula is used:

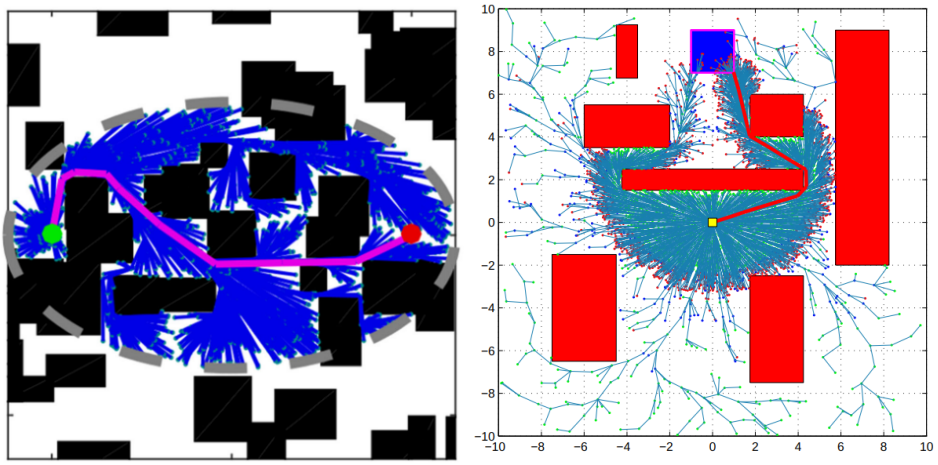
$$X_{\text{rel}} = \{x \in X_{\text{free}} : g^*(x) + h(x) < g^*(x^*_{\text{goal}})\}$$

where x^*_{goal} is the point in the goal region with the lowest optimal cost-to-come value, $g^*(x)$ is the optimal cost-to-come value of point x, and $h(x)$ is the estimate of the optimal cost moving from x to the goal region X_{goal}. If a point x is within the region X_{goal}, $h(x)$ is equal to zero.

3.6 RRT^X static

RRT^X [16] is an algorithm primarily engineered to navigate dynamic environments, where configurations of obstacles are unpredictable, demanding real-time adaptability. In this study, the focus is on delving into the static adaptation of RRT^X, named RRT^X static. Tailored for scenarios where environmental dynamics are stable.

The implementation of the RRT^X static algorithm closely resembles that of RRT$^{\#}$, with the addition of a single parameter, ϵ, representing the minimum threshold for cost improvement required to rewire the tree. In simpler terms, ϵ specifies the minimum amount by which a new path must improve over the existing path for the algorithm to justify rewiring the tree.



(a) : Informed RRT*. **(b) :** RRT$^{\#}$.

Figure 3.8: An example of Informed RRT* 3.8a and RRT$^{\#}$ 3.8b. Images courtesy of [6] and [1] respectively.

3.7 Motion Planning Networks (MPNet)

Motion Planning Networks (MPNet) [17] is an innovative learning-based neural planner. MPNet leverage neural networks to learn optimal path planning strategies, making them highly efficient in navigating complex environments. One of the primary reasons for using MPNets is their ability to significantly reduce computation time compared to traditional methods like RRT*. This speed advantage makes MPNets particularly suitable for real-time applications where rapid decision-making is essential.

According to the documentation [17], MPNet comprises two key components: the Planner Network (Pnet) and the Encoder Network (Enet). Enet processes information about the surrounding environment of the robot, including a raw point cloud. A raw point cloud refers to a collection of data points in a 3D space that represents the surface of an object or scene. The output of Enet is a latent space embedding of this raw point cloud information. Pnet utilizes the encoding of the environment, along with information about the current state and goal state of the robot, to generate samples for path or tree generation.

Let the environment surrounding the robot, referred to as the workspace, be denoted as $\mathcal{X} \subseteq \mathbb{R}^m$, where m is the workspace dimension. This workspace encompasses both obstacle regions $\mathcal{X}_{obs}$ and obstacle-free regions $\mathcal{X}_{free} = \mathcal{X} \setminus \mathcal{X}_{obs}$. MPNet plans feasible, near-optimal paths using raw point-cloud of obstacles $x_{obs} \subset \mathcal{X}_{obs}$. Similar to other planning algorithms, there is an assumed availability of a collision-checker that verifies the feasibility of MPNet-generated paths based on $\mathcal{X}_{obs}$. Precisely, Enet, parameterized by θ^e, processes the raw point cloud information x_{obs} and compresses it into a latent space Z.

$$Z \leftarrow \text{Enet}(x_{obs}; \theta^e)$$

Pnet, parameterized by θ^p, utilizes this latent space encoding Z, as well as the current or initial configuration $c_t \in \mathcal{C}_{\text{free}}$ and the goal configuration $c_{goal} \subset \mathcal{C}_{\text{free}}$ of the robot, to produce a trajectory through incremental generation of states $\hat{c}_{t+1}$.

$$\hat{c}_{t+1} \leftarrow \text{Pnet}(Z, c_t, c_{goal}; \theta^p)$$

Together, these neural networks enable MPNet to plan paths efficiently in complex environments.

One notable feature of MPNet is its integration of classical sample-based planners in a hybrid approach, which allows for worst-case theoretical guarantees while retaining computational efficiency and optimality improvements. Additionally, an active continual learning approach is presented to train MPNet models, enabling learning from streaming data and expert demonstrations as needed, thus reducing training data significantly.

Despite their numerous advantages, MPNets also have some limitations. The

quality of the path generated by MPNets heavily depends on the training data and the accuracy of the learned heuristic. Poorly trained MPNets may produce suboptimal or infeasible paths.

In the context of this thesis, MPNets provide valuable insights into the diversity of path planning methodologies. Moreover, through the integration of neural networks, which is a fundamental machine learning technique, MPNets aim to enhance sampling-based methods. Consequently, they are particularly relevant for understanding the existing methods and their advancements in the area of sampling-based approaches.

3.8 Summarize

In the Related Works chapter, various approaches relevant to this thesis were explored, including Motion Planning Networks (MPNet). MPNet serves as a notable example, indicating the potential for improvement in sampling-based planners. Sampling-based methods such as RRT, RRT*, and PRM are widely used in path planning. However, they all exhibit limitations when navigating through narrow passages and environments dense with obstacles. These limitations stem from their random sampling strategy, which often results in scenarios with narrow passages and dense obstacle spaces, leading to the generation of configurations within obstacle space. Consequently, this leads to unnecessary collision checking for inappropriate configurations. As a result, there is an increase in computational time and a decrease in the probability of sampling configurations capable of traversing tight gaps as the gap size decreases. To address these challenges and achieve improvements in the performance of sampling-based methods, akin to those seen in MPNet, the proposed solution to the problem will be provided in the subsequent Chapter 4. This method will be compared to its performance with RRT*, RRT^X static, RRT$^\#$ and Informed RRT*.

Chapter 4

Improving RRT* algorithm using machine learning method

In this chapter, the proposed solution to the challenge formulated in Chapter 2 will be explained. As previously mentioned, the focus on generating samples within the $\mathcal{C}_{\text{free}}$ ensures that configurations within the $\mathcal{C}_{\text{obs}}$ space are not sampled. This strategic approach minimizes the number of iterations required, as samples within the $\mathcal{C}_{\text{obs}}$ space invariably result in collisions, thus slowing down the process of achieving the optimal solution. To solve this challenge, the chapter will delve into the process of learning the configuration space, elucidating the proposed sample strategy and its integration into the RRT* algorithm. Furthermore, the extension of this approach to the 3D $\mathcal{C}$-space will be discussed.

The main idea and implementation of the approach presented in this chapter are inspired by the work of [2]. Consequently, many of the terminologies and concepts will closely resemble those expounded upon in the referenced work.

4.1 Approach

To provide further clarity, it is imperative to introduce some new terminology. Let $\mathcal{X}$ and $\mathcal{Y}$ denote the spaces of inputs and outputs, respectively. In this context, each $x_i \in \mathcal{X}$ represents a sampled point or configuration within the configuration space, while each $y_i \in \mathcal{Y}$ signifies its label, indicating whether the point is in $\mathcal{C}_{\text{obs}}$ or $\mathcal{C}_{\text{free}}$. In this context, the label $y_i = 1$ corresponds to points within $\mathcal{C}_{\text{obs}}$, while $y_i = 0$ denotes points within $\mathcal{C}_{\text{free}}$. A pair (x_i, y_i) is called a training example. $\mathcal{D} = \{(x_1, y_1), \ldots, (x_m, y_m)\}$ is called a training set consisting of m training examples.

The objective is to solve the classification problem by finding a function $f : \mathcal{X} \mapsto \{0, 1\}$ that provides prediction of the label for a given point. Thus, if a point is generated within the obstacle space, collision checking is unnecessary as the point is within the obstacle and an object representing a robot will definitely collide. Collision checking is only performed for points predicted by the program to be in $\mathcal{C}_{\text{free}}$ space. To accomplish this, learning the configuration

space become important.

4.2 Learning the Configuration Space

Function $f : \mathcal{X} \mapsto \{0, 1\}$ makes predictions based on available data about the configuration space at each program iteration. Due to this, there is a significant need to learn the configuration space, which underscores the importance of understanding how this available data is obtained.

This data, represented as a training set $\mathcal{D} = \{(x_i, y_i) : i = 1, \ldots, m\}$ with points and their labels, initially consists of randomly generated points along with their labels, determined through collision checking. During each program iteration, the function f predicts labels for randomly sampled points based on this available data in the training set.

However, at the beginning, our training set may not be sufficiently large for accurate predictions. Therefore, if the program predicts that a point lies within the $\mathcal{C}_{\text{free}}$ space, it must verify this prediction through collision checking. If the point is indeed within $\mathcal{C}_{\text{free}}$, the program selects this point for sampling and adds it to the trainning set $\mathcal{D}$ with its corresponding label. Conversely, if the point is predicted incorrectly and lies within $\mathcal{C}_{\text{obs}}$, the program continues searching for a point within $\mathcal{C}_{\text{free}}$. Every point identified within $\mathcal{C}_{\text{obs}}$ is also added to the trainning set $\mathcal{D}$. Through this trainning set $\mathcal{D}$, which increases after each iteration, the algorithm learns the configuration space. With each iteration, it improves its predictions and increasingly identifies points within $\mathcal{C}_{\text{free}}$. Consequently, the generation of points within $\mathcal{C}_{\text{obs}}$ decreases, thereby reducing the need for collision checking on unsuitable points.

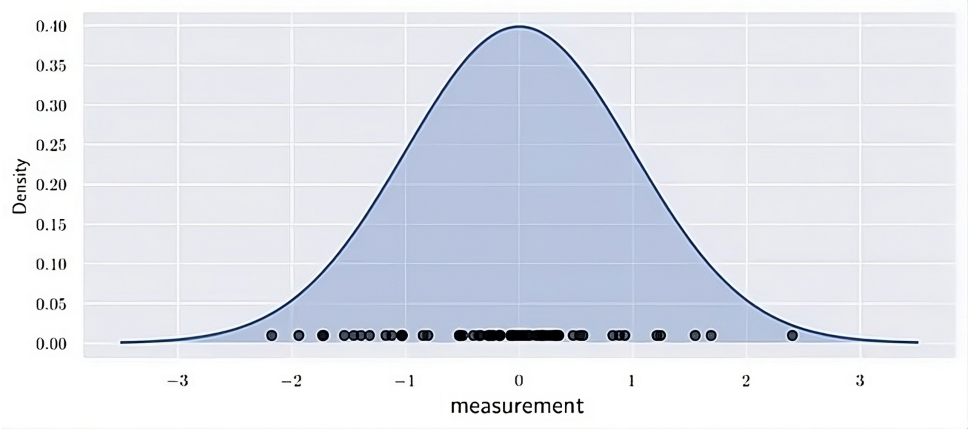
The key question now arises: how does the program make predictions? To address this, a Bayesian classifier [3] is utilized in conjunction with a kernel density estimator to determine the function $f : \mathcal{X} \mapsto \{0, 1\}$.

4.3 Proposed Solution

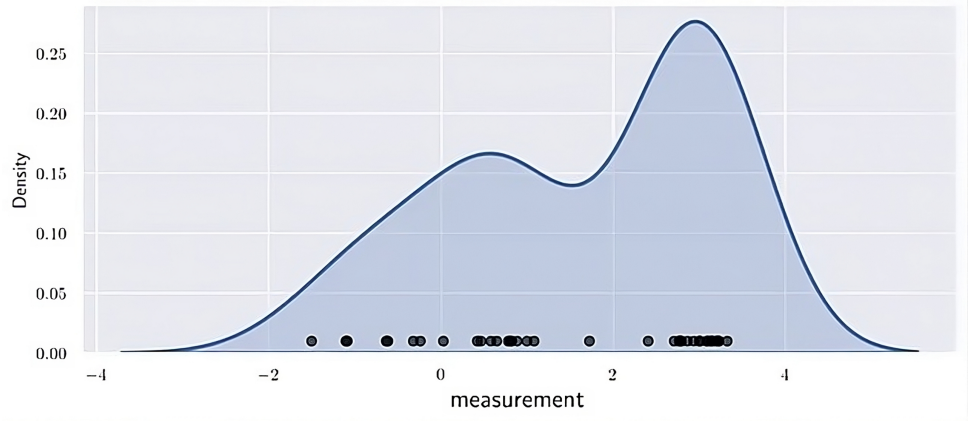
Now that the method for learning the configuration space and the significance of the training set are clear, let us delve deeper. The training set can be divided into two datasets: $\mathcal{X}_{\text{free}}$ comprising points in $\mathcal{C}_{\text{free}}$ and $\mathcal{X}_{\text{obs}}$ containing points within $\mathcal{C}_{\text{obs}}$. Based on these datasets, probability density functions will be computed to approximate the locations of obstacle and obstacle-free spaces. Using a Bayesian classifier, predictions can then determine whether a given point belongs to $\mathcal{C}_{\text{free}}$ or $\mathcal{C}_{\text{obs}}$. For clarity in the text, the probability density function that will be utilized in the proposed solution will be described.

When the data fit well, it means that the data points closely match the expected distribution or pattern 4.1a. In such cases, commonly used probability density functions include the Normal, Poisson, and Geometric distributions,

among others. However, in the case of our problem, the data consists of randomly sampled points, resulting in an irregular data distribution 4.1b. In such cases, Kernel Density Estimator is employed to approximate the



(a) : Regular.



(b) : Irregular.

Figure 4.1: One-dimensional example of regular 4.1a and irregular 4.1b data distribution. Images courtesy of [4].

underlying distribution of the data. The Kernel Density Estimator $\hat{f}_{\mathcal{X}}(x)$ for the estimation of the density value at point x is defined as:

$$\hat{f}_{\mathcal{X}}(x) = \frac{1}{m} \sum_{i=1}^{m} K\left(x - x_i\right),$$

where m is a number of points x_i in respective dataset $\mathcal{X}_{\text{free}}$ or $\mathcal{X}_{\text{obs}}$ and $K(x)$ is the Gaussian Kernel function [5], represented as follows:

$$K(x) = (2\pi)^{-\frac{d}{2}} \cdot \det(\mathbf{H})^{-\frac{1}{2}} \cdot e^{-\frac{1}{2} x^T H^{-1} x},$$

where x represents the input vector, with a dimension of the input space d, $\mathbf{H}$ denotes the bandwidth matrix, $(2\pi)^{-\frac{d}{2}}$ is a normalization constant. Matrix $\mathbf{H}$ serves as a covariance matrix and acts as a user-defined parameter

influencing the kernel function. When $d = 2$, it corresponds to a bivariate case. For instance, in a 2D configuration space, as depicted in Figure 4.2, the input vector x comprises two coordinates of the point: $x = [x_1, x_2]$, and the bandwidth matrix takes the form $H = \begin{bmatrix} h_{11} & h_{12} \\ h_{21} & h_{22} \end{bmatrix}$. Here, h_{11} and h_{22} correspond to the variances of x_1 and x_2 respectively, while $h_{12} = h_{21}$ represents the covariance between x_1 and x_2. Because of this, the matrix **H** is symmetric. An illustrative example of kernel estimation in the configuration space depicted in Figure 4.2 can be observed in Figure 4.3.

Algorithm 4: Sample Density

Data: $\mathcal{X}_{\text{obs}}$, $\mathcal{X}_{\text{free}}$
Result: Predicted sampled point x

$\gamma_{\text{free}} \leftarrow 0$;
$\gamma_{\text{obs}} \leftarrow 1$;
while $\gamma_{free} < \gamma_{obs}$ **do**
  $x_{\text{rand}} \leftarrow$ RandomConfiguration();
  $P_{\text{free}} \leftarrow \frac{|\mathcal{X}_{\text{free}}|}{|\mathcal{X}_{\text{free}}|+|\mathcal{X}_{\text{obs}}|}$;
  $P_{\text{obs}} \leftarrow 1 - P_{\text{free}}$;
  $b_{\text{free}} \leftarrow$ DensityEstimator(x_{rand}, $\mathcal{X}_{\text{free}}$);
  $b_{\text{obs}} \leftarrow$ DensityEstimator(x_{rand}, $\mathcal{X}_{\text{obs}}$);
  $\gamma_{\text{free}} \leftarrow b_{\text{free}} \cdot P_{\text{free}}$;
  $\gamma_{\text{obs}} \leftarrow b_{\text{obs}} \cdot P_{\text{obs}}$;
$x \leftarrow x_{\text{rand}}$;
return x;

Now, leveraging our density estimator, let us proceed to Bayesian classifier. Firstly, it is necessary to introduce Bayes' theorem, which is formulated as follows:

$$P(A|B) = \frac{P(B|A) \cdot P(A)}{P(B)}.$$

In this context, the aim is to determine the probability of a point x belonging to a particular space, denoted by $P(y|x)$. Our Kernel Density Estimator, on the other hand, represents $P(x|y)$, which is the probability distribution of points x given a specific space y. As previously mentioned, the training set was divided into two datasets: $\mathcal{X}_{\text{free}}$ and $\mathcal{X}_{\text{obs}}$. These datasets were utilized to determine $P(y)$ for each respective space. So now our Bayesian theorem looks like this:

$$P(y|x) = \frac{P(x|y) \cdot P(y)}{P(x)},$$

where

$$P(x|y=1) = \hat{f}_{\mathcal{X}_{\text{obs}}}(x) = \frac{1}{|\mathcal{X}_{\text{obs}}|} \sum_{x' \in \mathcal{X}_{\text{obs}}} K\left(x - x'\right);$$

$$P(x|y=0) = \hat{f}_{\mathcal{X}_{\text{free}}}(x) = \frac{1}{|\mathcal{X}_{\text{free}}|} \sum_{x' \in \mathcal{X}_{\text{free}}} K\left(x - x'\right);$$

$$P(y=1) = \frac{|\mathcal{X}_{\text{obs}}|}{|\mathcal{X}_{\text{obs}}| + |\mathcal{X}_{\text{free}}|}; \;\; P(y=0) = \frac{|\mathcal{X}_{\text{free}}|}{|\mathcal{X}_{\text{obs}}| + |\mathcal{X}_{\text{free}}|};$$

$|\mathcal{X}_{\text{obs}}|$ denotes the number of points contained within the $\mathcal{X}_{\text{obs}}$ dataset;

$|\mathcal{X}_{\text{free}}|$ denotes the number of points contained within the $\mathcal{X}_{\text{free}}$ dataset;

Regarding $P(x)$, further discussion will follow.

Now, the Bayesian decision rule can be formulated. This rule will be represented as a function $f : \mathcal{X} \mapsto \{0, 1\}$, which will predict the label of the given point:

$$f(x) = \begin{cases} 0 & \text{if } \gamma_{\text{free}}(x) \geq \gamma_{\text{obs}}(x), \\ 1 & \text{otherwise,} \end{cases}$$

where

$$\gamma_{\text{free}}(x) = P(y=0|x); \;\; \gamma_{\text{obs}}(x) = P(y=1|x).$$

In this decision rule, where $\gamma_{\text{free}}(x) \geq \gamma_{\text{obs}}(x)$, it's evident that the computation of $P(x)$ is not necessarily required.

The proposed solution is outlined in Algorithm 4: Sample Density. Default parameters for γ_{free} and γ_{obs} are defined at Line 1 and Line 2, where γ_{free} should be less than γ_{obs}. The main process occurs within a while loop at Line 3-10. The condition in this loop is inverted compare to our decision rule because the goal is to exit the loop when this condition is met. Inside this loop, a random point is generated at each iteration.

For each random point, the probability of belonging to each space is computed as explained in this section. Subsequently, the point predicted to be in the C_{free} space is returned. Now, let us explore how this method can be integrated with sampled-based methods.

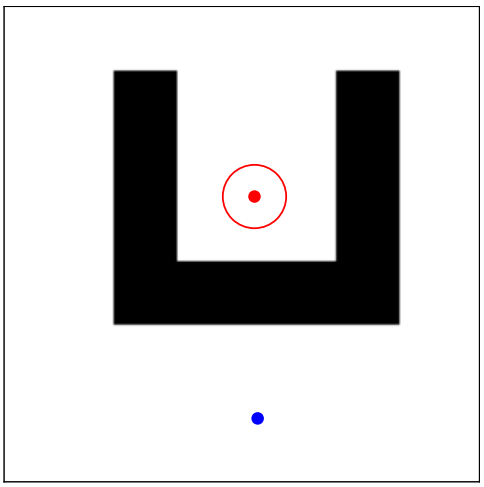
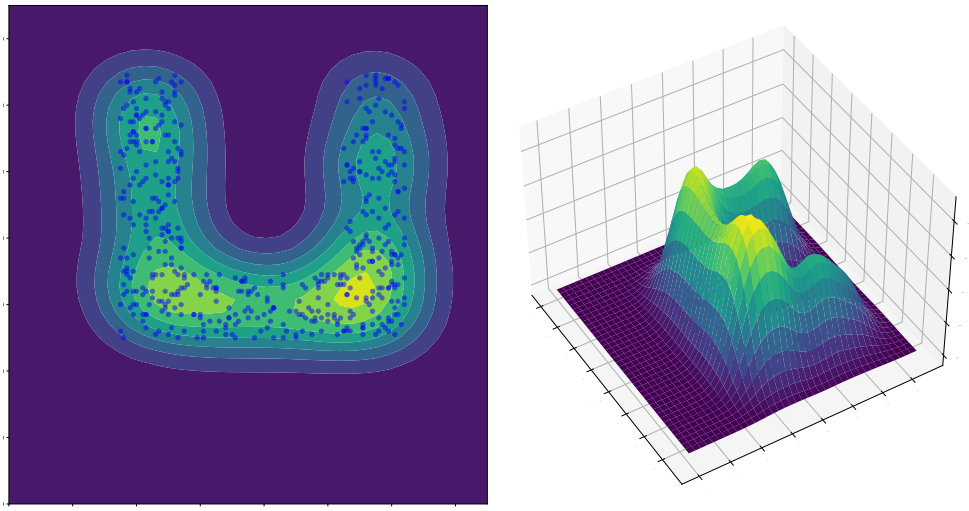
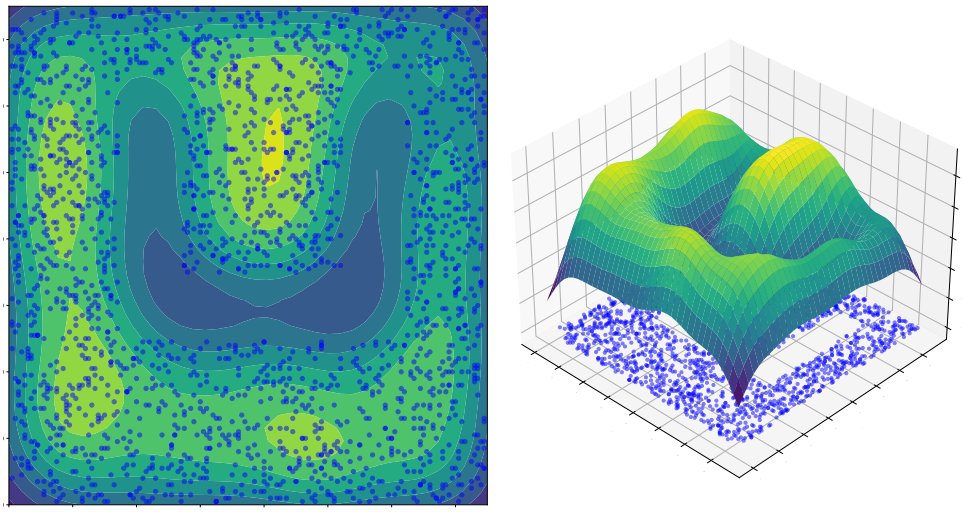


Figure 4.2: An example of the 2D workspace with a 2D $\mathcal{C}$-space.



(a) : Obstacle density.

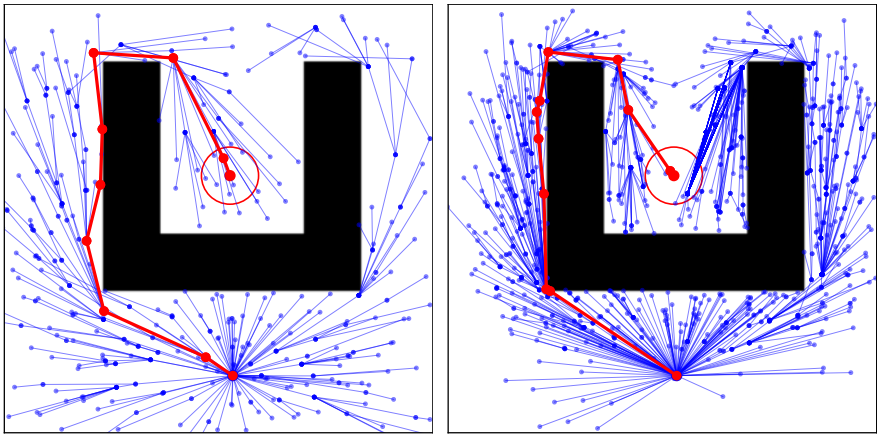


(b) : Obstacle-free density.

Figure 4.3: An example of density estimation with Gaussian Kernel function of the $\mathcal{C}$-space in Figure 4.2. Circles represent randomly generated points in the $\mathcal{C}$-space. In Figure 4.3a, the density estimation of the obstacle space can be seen. Figure 4.3b illustrates the density estimation of the free space. The higher the graph, the higher the density, indicating a higher probability for points to be in respective space.

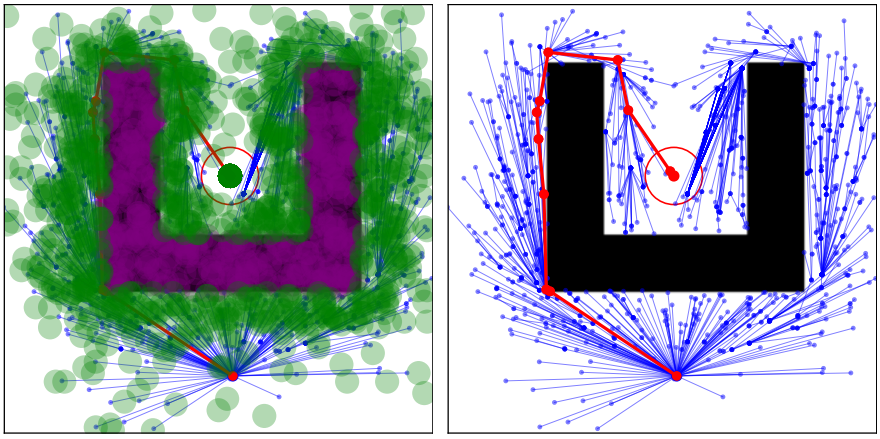
4.4 Integration to the RRT* algorithm

The proposed solution is designed to be compatible with any sampling-based method that employs single configuration generation to explore the configuration space in a single iteration. In this implementation, RRT* was specifically chosen for its ability to converge efficiently towards an optimal solution while effectively exploring the configuration space.



(a) : RRT*. **(b) :** Improved RRT*.

Figure 4.4: Both algorithms, RRT* 4.4a and the improved RRT* using machine learning 4.4b methods, are presented, each consisting of 1000 iterations. It is evident that the improved RRT* efficiently explores the configuration space and eventually finds a path to the goal with the lowest cost of 206 u.d.(units of dimension), compared to RRT*, which finds a path to the goal with a cost of 216 u.d.. Additionally, despite both algorithms having the same number of iterations, the improved RRT* exhibits a higher number of collision-free paths found. This difference arises from the strategy of sampling points in the $\mathcal{C}$-space, which increases the probability of finding a collision-free path after each iteration.



(a) : Learning the $\mathcal{C}$-space. **(b) :** Improved RRT*.

Figure 4.5: In Figure 4.5a, the information about the $\mathcal{C}$-space can be seen. The purple color represents the $\mathcal{X}_{\text{obs}}$ dataset, while the green color represents the $\mathcal{X}_{\text{free}}$ dataset. Utilizing these datasets, the algorithm learns the $\mathcal{C}$-space.

The integration approach relies on a specific function for sampling predicted points, rather than utilizing random sampling. This function is outlined in Algorithm 5. Within the while loop, point predicted by the function f to be in the $\mathcal{C}_{\text{free}}$ are generated. After each iteration, the algorithm verifies if this point indeed lies inside $\mathcal{C}_{\text{free}}$, and upon finding such a point, it is returned. If a point lies within the $\mathcal{C}_{\text{obs}}$ space, it is added to the $\mathcal{X}_{\text{obs}}$ dataset. Otherwise,

it is added to the $\mathcal{X}_{\text{free}}$ dataset. This function effectively replaces the random configuration generation step in Algorithm 2, Line 3, thus demonstrating integration with the RRT* algorithm.

Algorithm 5: Sample

Data: $\mathcal{X}_{\text{obs}}$, $\mathcal{X}_{\text{free}}$
Result: Sampled point x

```
1 x ← Sample Density(X_obs, X_free);          // Algorithm 4;
2 while OnObstacle(x) do
3 |   X_obs ← X_obs ∪ {x};
4 |   x ← Sample Density(X_obs, X_free);      // Algorithm 4;
5 X_free ← X_free ∪ {x};
6 return x;
```

To demonstrate how this enhancement works, in Figure 4.4 can be seen the expansion of RRT* with enhanced RRT* using machine learning, which explores $\mathcal{C}$-space. Figure 4.5 illustrates the learning process of the $\mathcal{C}$-space by our enhanced RRT* algorithm.

After explaining how the proposed solution was integrated into the RRT*, it is crucial to evaluate its effectiveness compared to other planners. Chapter 5 provides further insights into this comparative analysis.

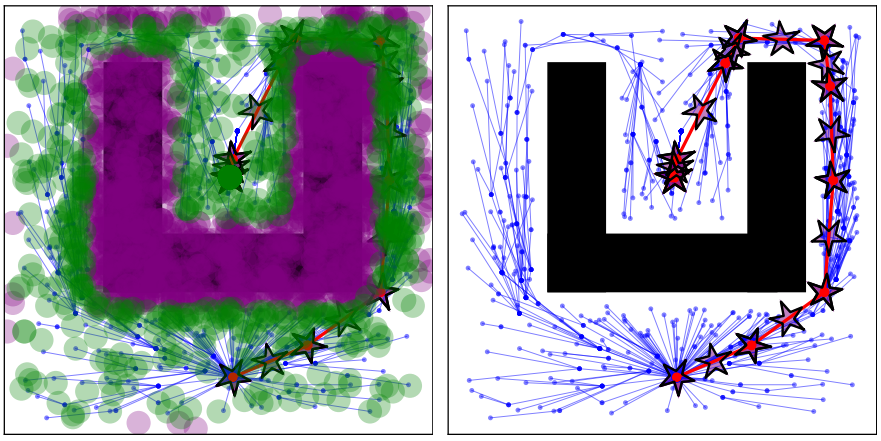
4.5 Extension of the machine learning method to 3D and 6D configuration spaces

In this problem, the robot is represented as a polygon. To describe the position of the robot in space, three coordinates are utilized: two for its position in $\mathcal{C}$-space and a third one for its rotation angle, denoted as $q = (x, y, \alpha)$. The position of the robot is defined as its center of mass. The main difference from the implementation in 2D $\mathcal{C}$-space is that the procedure for checking whether a point is inside the $\mathcal{C}_{\text{obs}}$, as shown in Algorithm 5 Line 2, is replaced by a function that checks if at least one point of robot lies inside the $\mathcal{C}_{\text{obs}}$. Also, verifying whether the path is obstacle-free, it is essential to ensure that the robot does not collide with obstacles while moving along the path. To address this, linear interpolation between two points is utilized to detect potential collisions between the robot and obstacles along the path. The linear interpolation formula

$$q(s) = (1 - s) \cdot q_{\text{near}} + s \cdot q_{\text{new}}, \quad s \in [0, 1],$$

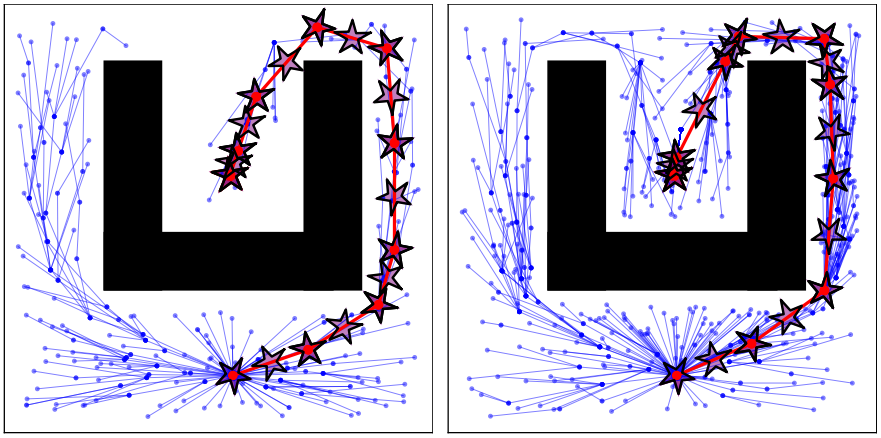
defines the points along the line segment connecting q_{near} and q_{new}. Consequently, the $\mathcal{X}_{\text{obs}}$ space is not always represented just by the area of the obstacle, but also includes points where the robot intersects with the obstacle. An example can be seen in Figure 4.6.

Similarly as in 2D $\mathcal{C}$-space, the improved version of RRT* demonstrates better performance in finding optimal paths 4.7.



(a) : Learning the $\mathcal{C}$-space. **(b) :** Improved RRT*.

Figure 4.6: In Figure 4.6a, it can be seen that the $\mathcal{X}_{\text{obs}}$ dataset contains more data points. This is because the purple area contains not only points inside the $\mathcal{C}_{\text{obs}}$, but also points where robot, represented as a polygon in the shape of a star, collides with obstacles.

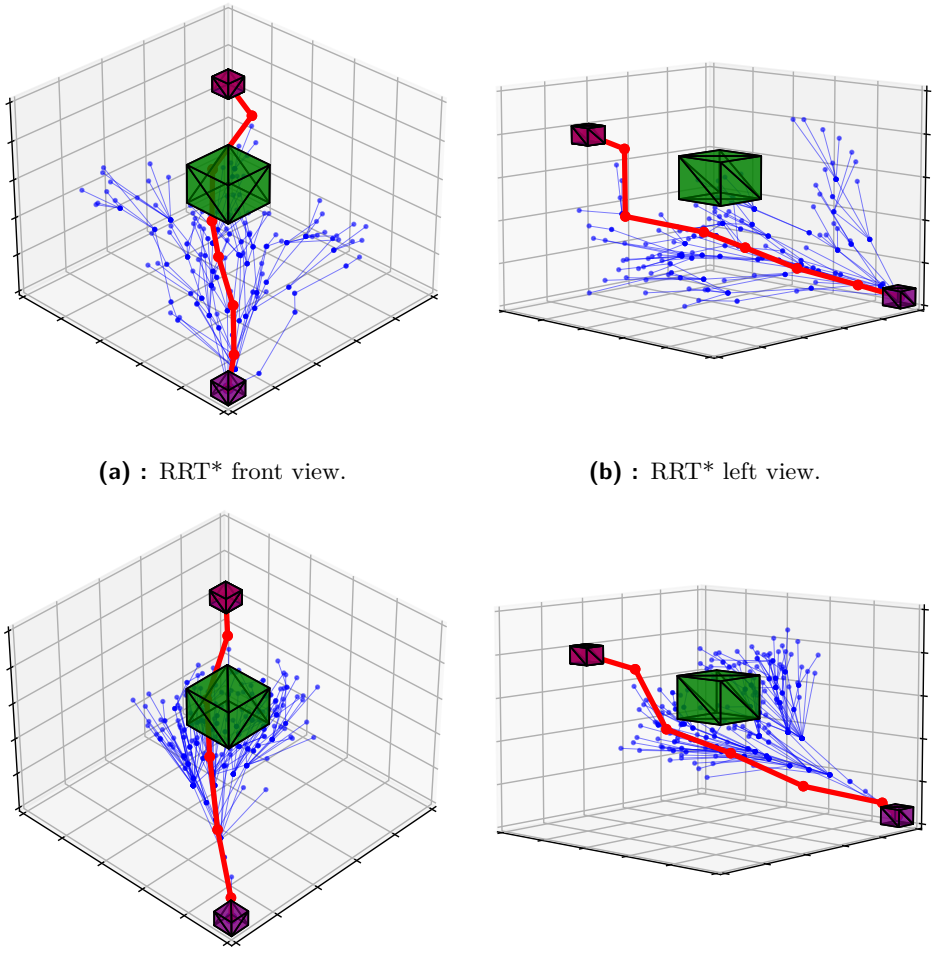


(a) : RRT*. **(b) :** Improved RRT*.

Figure 4.7: Both algorithms, RRT* 4.7a and the improved RRT* using machine learning 4.7b methods, are presented, each consisting of 1000 iterations. The improved RRT* finds a path to the goal with the lowest cost of 237.2 u.d. (units of dimension), compared to RRT*, which finds a path to the goal with a cost of 241.4 u.d..

To extend the improved RRT* algorithm to a 6D $\mathcal{C}$-space, such as in the Piano Movers Problem, similar procedures to those in 3D $\mathcal{C}$-space need to be followed. However, now six coordinates are required to describe the configuration of the robot, denoted as $q = (x, y, z, yaw, roll, pitch)$. Additionally, the robot will be represented by a 3D triangle mesh. With the adoption of a 3D triangle

mesh representation, collision detection becomes more complex. To address this challenge, the RAPID library [8] can be utilized. RAPID (Robust and Accurate Polygon Interference Detection), implemented in C++, is a robust tool for collision detection in 3D environments. It provides a narrow and user-friendly API (Application Programming Interface) for programmers, simplifying the detection of intersections between objects in various applications such as physically based modeling, virtual prototyping, and CAD. In Python 3.7 and later, the Trimesh library can be utilized for collision detection. This library is well-tested for handling triangle mesh objects. Figure 4.8 shows an example of a 3D workspace with a corresponding 6D $\mathcal{C}$-space.



(a) : RRT* front view.

(b) : RRT* left view.

(c) : Improved RRT* front view.

(d) : Improved RRT* left view.

Figure 4.8: An example of path planning in the 3D workspace with the 6D $\mathcal{C}$-space. The green color represents obstacle object. The purple color indicates object representing our robot. The red line shows the found solution path, and the blue lines illustrate the paths explored in the $\mathcal{C}$-space. It can be seen that improved RRT* demonstrates better performance than RRT* by finding a shorter path to the goal from the same number of iterations.

Chapter 5

Results and discussion

Throughout this chapter, the performance of the implemented RRT* with machine learning methods, denoted as RRT*_ML in the rest of the text, will be presented and discussed alongside other planners across various configuration spaces. In this work, the specification of units of measurement is not necessary as the choice of units can vary depending on the preferences or requirements. In addition to testing the runtime of the code, the experiments were conducted on a computer with the specifications detailed in Table 5.1 This information is provided to ensure transparency of the results.

Parameter	Value
OS	Ubuntu 22.04.4 LTS x86_64
CPU	11th Gen Intel i5-11400H
GPU	NVIDIA GeForce RTX 3050 Mobile
RAM	16GB

Table 5.1: Computer specifications.

5.1 2D Configuration Space

To assess the effectiveness of the implemented enhanced RRT* algorithm across various scenarios, planners from the Open Motion Planning Library (OMPL) [20] will serve as benchmarks. OMPL is a widely-used library that provides an extensive collection of motion planning algorithms, making it a valuable resource for comparing and evaluating different planning strategies.

Name	Parameter	Value [-]
Goal bias	p_{goal}	0.05
Epsilon	ϵ	0.01
Rewire Factor	k	1.1
Rewiring radius	r	40
Step size	s	10

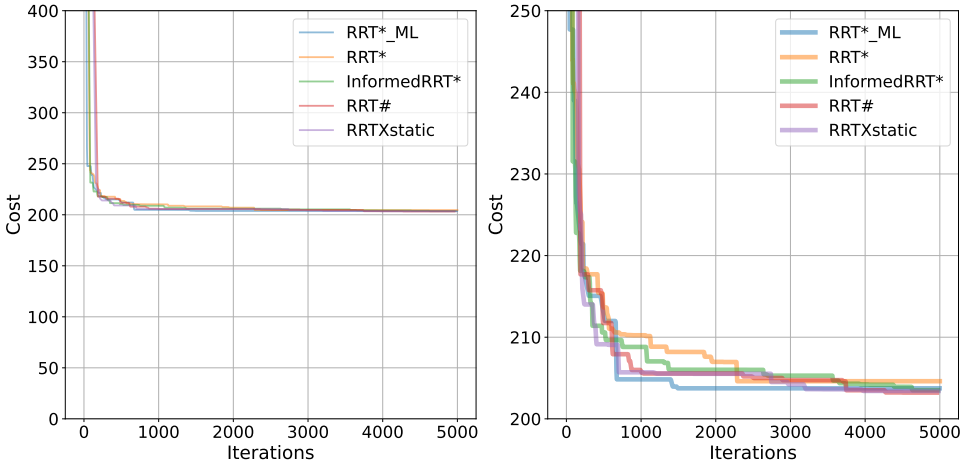
Table 5.2: Planners parameters.

The primary objective of this comparison is to evaluate how integrating

machine learning methods enhances the performance of the RRT* algorithm. Additionally, the implemented method will be compared against similar planners that provide information about the cost of the path, such as RRT$^{\#}$ (sharp) [1], RRT^{X} static [16], and Informed RRT* [6]. This comparative analysis will offer insights into how our approach performs in terms of optimizing path costs compared to these existing algorithms.

5.1.1 Planners parameters

All planners share a common goal bias parameter, $p_{\text{goal}} = 0.05$, indicating that there is a 5% probability that a randomly sampled point will serve as a goal point. While RRT$^{\#}$ and RRT^{X} static share similar implementations, they diverge in their use of the parameter ϵ. In the case of RRT$^{\#}$, ϵ is set to 0, whereas in RRT^{X} static, it is initially set to 0.01. ϵ representing the minimum threshold for cost improvement required to rewire the tree.



(a) : Original unscaled graph.

(b) : Scaled graph.

Figure 5.1: To enhance visualization of the graphs, all will be scaled to depict changes more clearly. Consequently, the cost axes will not necessarily start at zero.

Additionally, all OMPL planners start with a rewiring factor of $k = 1.1$. This factor, applied multiplicatively, determines the size of the neighbor ball or the number of neighbors considered during tree expansion. In the implementation of RRT*_ML, the rewiring radius is set to $r = 40$ and the step size to $s = 10$. The rewiring radius represents the radius within which nearest neighbor searches are conducted, while the step size represents the maximum length of a path segment.

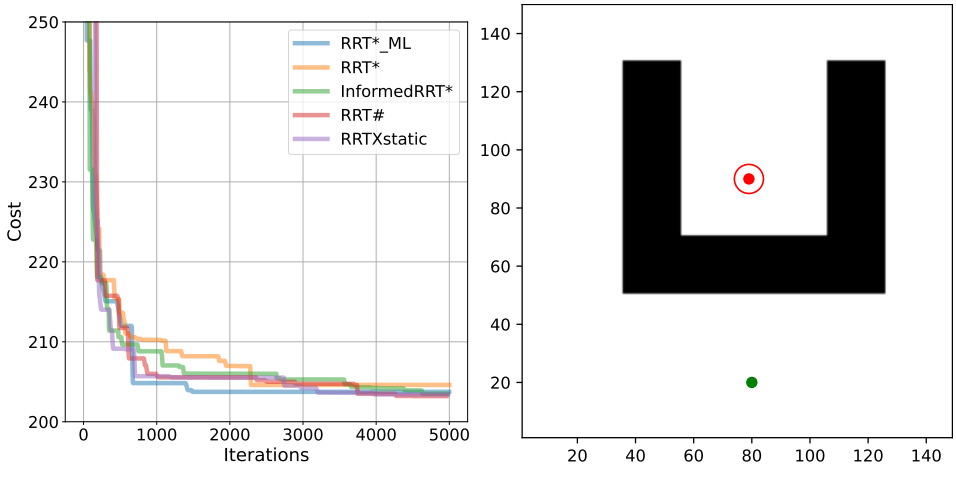
The parameters for RRT*_ML were manually tuned, potentially resulting in suboptimal solutions. In certain scenarios, a larger or smaller rewiring radius or step size might be more beneficial. As a consequence, the OMPL implementation may outperform the custom Python implementation in terms of speed.

To determine the bandwidth matrix **H** for RRT*_ML, Scott's rule was utilized [19]. Thanks to this rule, the bandwidth matrix is not searched manually, and during program execution, Scott's rule dynamically adjusts the bandwidth matrix **H**. In the case of a 2D configuration space, the bandwidth matrix **H** is a symmetric matrix $\begin{bmatrix} h & 0 \\ 0 & h \end{bmatrix}$, where h is calculated using Scott's rule: $h = n^{-\frac{1}{d+4}}$. Here, n represents the number of data points, and d represents the number of dimensions.

All parameters are listed in Table 5.2.

5.1.2 Cost convergence analysis

This subsection will present a comparative analysis of the RRT*_ML method with OMPL planners, with a focus on their convergence during iterations. To enhance visibility and emphasize changes, the graphs will be scaled. Therefore, the Y-axis will be adjusted to start at a value just below the convergence line of the graph, ensuring larger and clearer graph representation (Figure 5.1). In all maps discussed in this subsection, the robot is represented by a point, with its configuration denoted by two coordiniates $q = (x, y)$, indicatind a 2D $\mathcal{C}$-space. A green circle indicates the q_{start}, a red ring signifies the $\mathcal{C}_{\text{goal}}$, and a red point denotes the q_{goal}. Black areas represent the $\mathcal{C}_{\text{obs}}$.



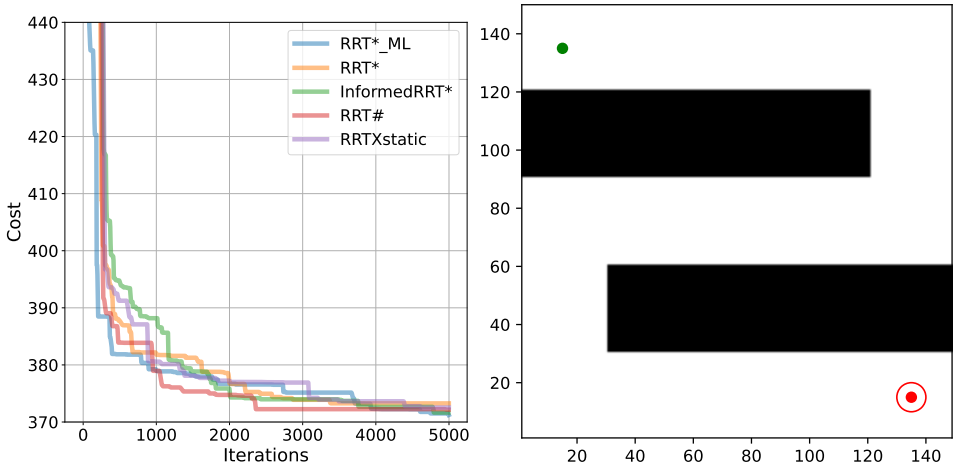
(a) : Graph. **(b) :** Map.

Figure 5.2: Map 5.2b and corresponding convergence graphs 5.2a of selected planners.

Figures 5.2, 5.3, 5.4, 5.5 depict the maps and convergence graphs of each planner. All tests were conducted up to a point where the graphs showed minimal change. Path costs were computed the same way across all planners, as the path length.

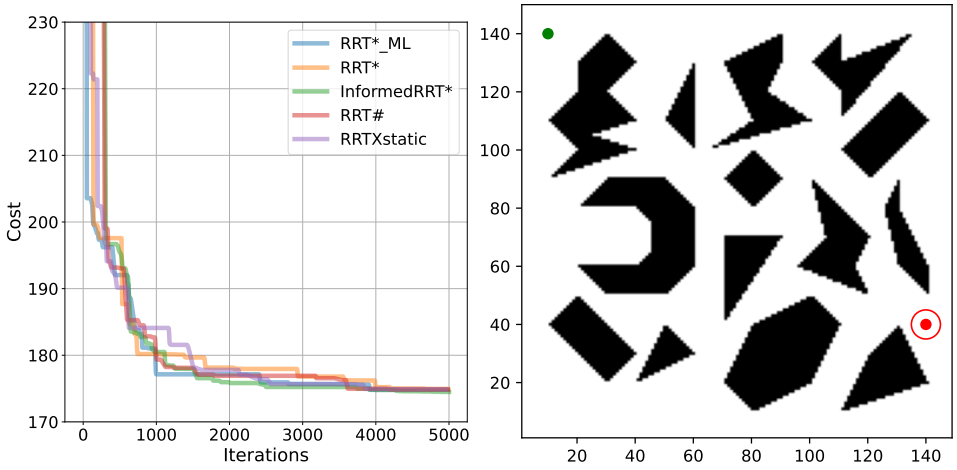
In Figure 5.2, a simple map, previously discussed, is shown. The implemented

RRT*_ML notably converges faster to the optimal solution compared to RRT*. Other planners, while slightly slower, also find solutions. However, the performance of the implemented machine learning method surpasses that of the other planners in this scenario.



(a) : Graph. **(b) :** Map.

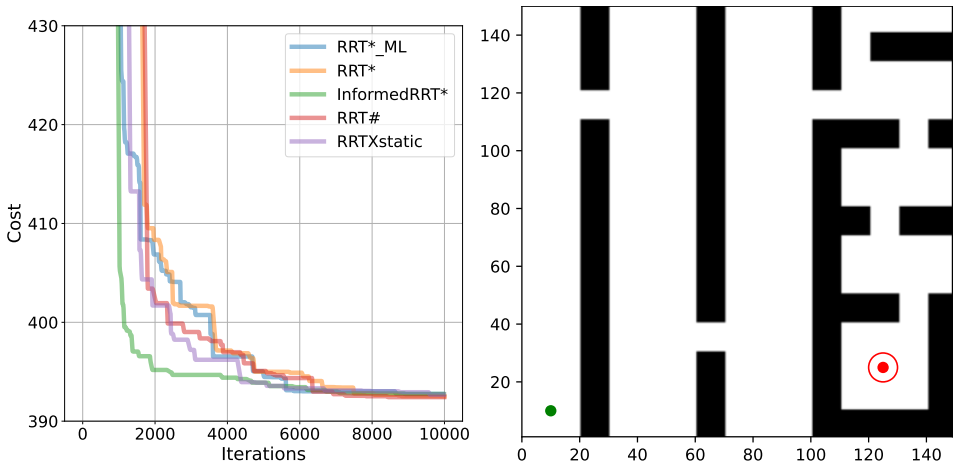
Figure 5.3: Map with narrow passages 5.3b and corresponding convergence graphs 5.3a of selected planners.



(a) : Graph. **(b) :** Map.

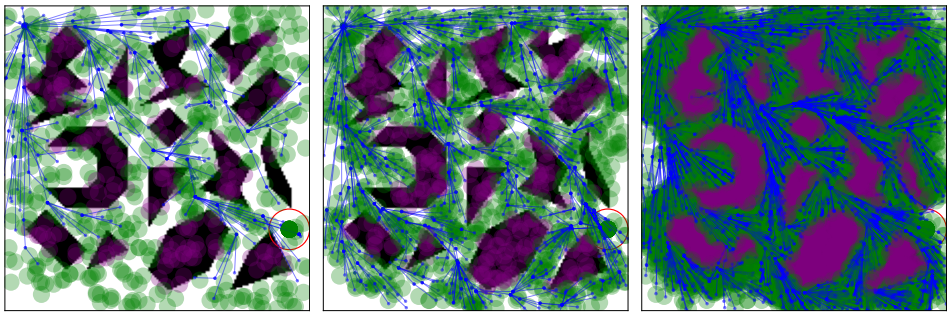
Figure 5.4: Cluttered map 5.4b and corresponding convergence graphs 5.3a of selected planners.

Figure 5.3b illustrates a simple example of a map with narrow passages. The implemented method discovers a solution slightly earlier than other planners, although its performance becomes comparable to RRT* after 2000 iterations. Figure 5.4 presents a cluttered map where all methods exhibit similar behavior. However, Improved RRT* initiates convergence to the first possible solution earlier than the others and demonstrates better convergence to the optimal solution than RRT*. In Figure 5.5, a more challenging scenario with numerous



(a) : Graph. **(b) :** Map.

Figure 5.5: Difficult map with numerouse narrow passages 5.4b and corresponding convergence graphs 5.3a of selected planners.

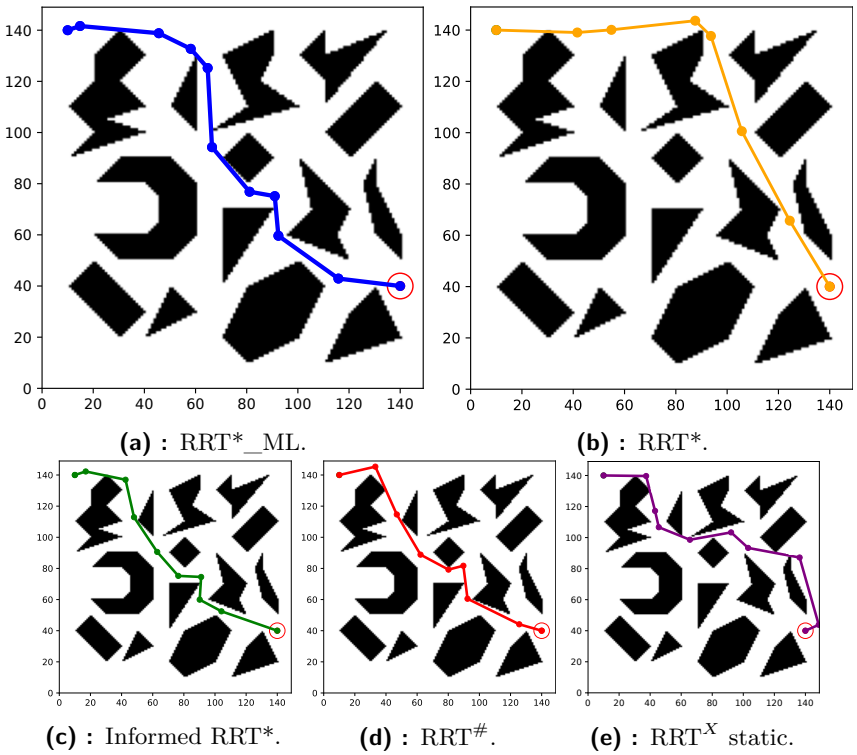


(a) : 500 iterations. **(b) :** 1000 iterations. **(c) :** 5000 iterations.

Figure 5.6: Learning $\mathcal{C}$-space in the cluttered map 5.4b.

narrow passages is demonstrated. In this case, the implemented enhanced RRT* once again discovers the first possible solution earlier than RRT*. However, in terms of convergence, it behaves similarly to RRT*.

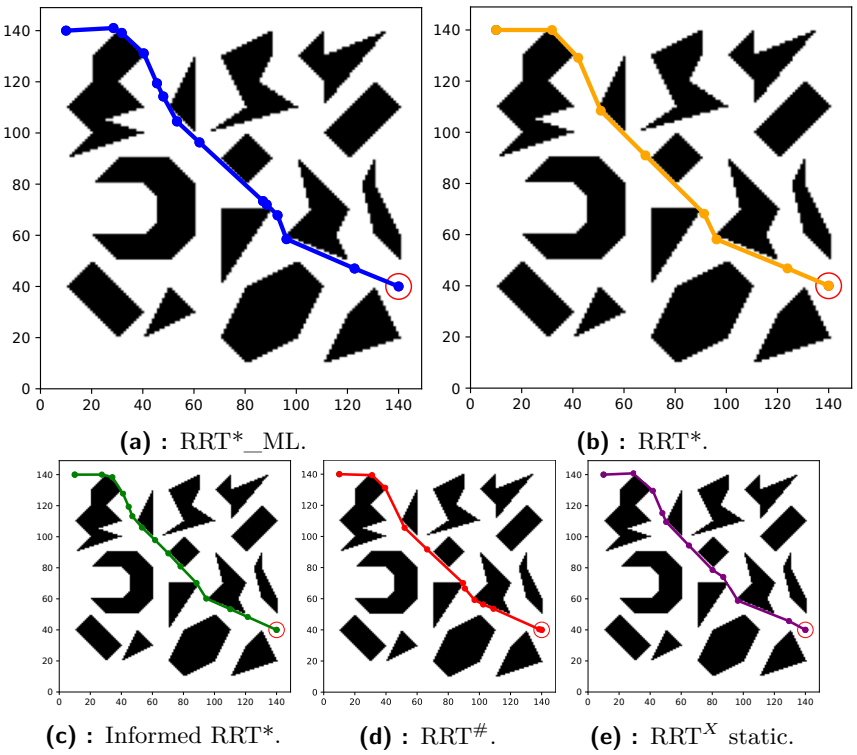
Figure 5.6 demonstrates how the RRT*_ML algorithm learns and adapts to the cluttered map. Using the example of a cluttered map in Figure 5.4b, the initial solutions found by each planner are illustrated in Figure 5.7. Figure 5.8 shows the final solutions provided by each planner. The final solution is found after 5000 iterations, while the initial solution is the first found possible solution. The cluttered map was selected for its potential to offer a greater diversity of paths compared to other maps due to the multitude of possible path variations.



(a) : RRT*_ML. **(b) :** RRT*.

(c) : Informed RRT*. **(d) :** RRT$^{\#}$. **(e) :** RRT^{X} static.

Figure 5.7: An example of the initial solutions found by each planner in the cluttered map.



(a) : RRT*_ML. **(b) :** RRT*.

(c) : Informed RRT*. **(d) :** RRT$^{\#}$. **(e) :** RRT^{X} static.

Figure 5.8: An example of the final solutions found by each planner in the cluttered map.

5.1.3 Runtime

It is important to note that while the RRT*_ML method was programmed in Python, the OMPL planners were implemented in C++, contributing to potential differences in computation time.

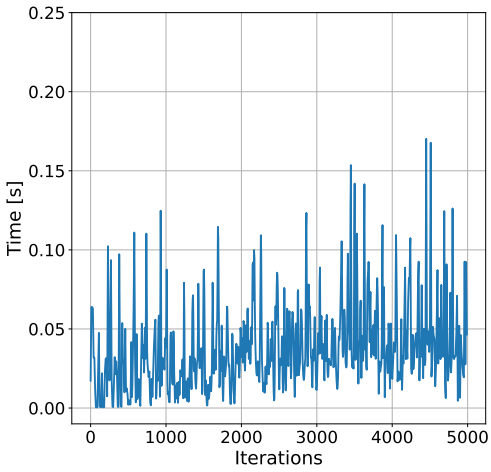


Figure 5.9: Time taken to learn predicting points (Algorithm 5) in the cluttered map 5.4b to be in the C_{free}.

Runtime		
Planner	**Initial solution [s]**	**Final solution [s]**
RRT*_ML	0.278594 ± 0.157754	289.625891 ± 2.611068
RRT*	0.006518 ± 0.001462	0.321272 ± 0.014779
Informed RRT*	0.007330 ± 0.000536	0.259046 ± 0.003154
RRT$^{\#}$	0.011917 ± 0.002803	0.305096 ± 0.010232
RRT^{X} static	0.011775 ± 0.002566	0.305259 ± 0.008140

Table 5.3: Time spent to find the first and final solutions in the cluttered map.

In Table 5.3, the computational times to find the first and final solutions in the cluttered configuration space are presented. All values were computed as the mean of ten measurements, along with their standard deviations. The mean value $\bar{x}$ and the variance σ^2 were calculated using the following formulas:

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i; \quad \sigma^2 = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2,$$

where the standard deviation σ is the square root of the variance:

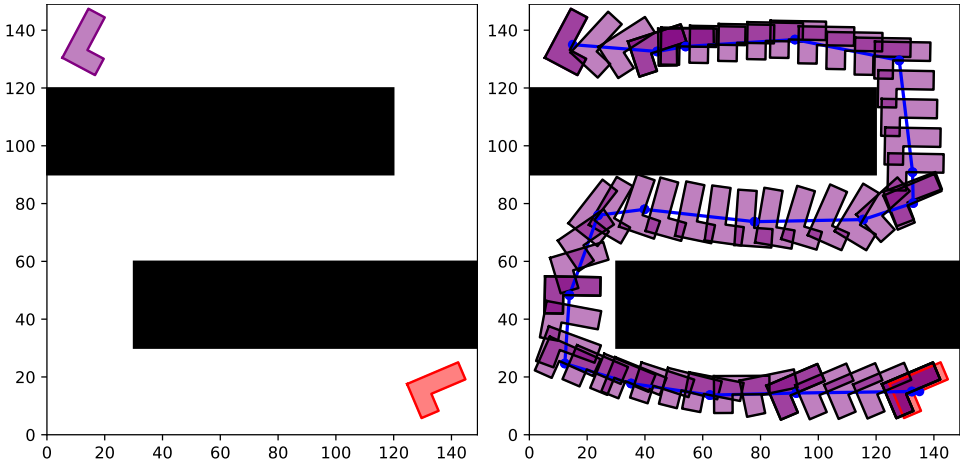
$$\sigma = \sqrt{\sigma^2}$$

The average time spent by RRT*_ML to predict point (Algorithm 5) in the $\mathcal{C}_{\text{fere}}$ is (0.037988 ± 0.030603) seconds (Figure 5.9). It can also be observed that as the number of iterations increases, the computation time tends to rise. This is because both datasets, $\mathcal{X}_{\text{free}}$ and $\mathcal{X}_{\text{obs}}$, grow linearly with each

iteration, gradually accumulating more data points. Consequently, the density estimation process becomes more time-consuming. After 5000 iterations, the $\mathcal{X}_{\text{free}}$ dataset contains 4816 datapoints, and the $\mathcal{X}_{\text{obs}}$ dataset contains 1954 datapoints.

5.2 3D and 6D Configuration Spaces

In comparing the implemented method with OMPL planners in 3D and 6D $\mathcal{C}$-spaces, it is not feasible due to the initial implementation of RRT*_ML in Python. OMPL does not directly provide the capability to represent objects in the same manner as implemented in RRT*_ML. Additionally, OMPL does not directly provide collision checking for 2D and 3D objects. As a result, the RRT*_ML method will be compared with RRT*, which is also implemented in Python and into which RRT*_ML was integrated.



(a) : Workspace and robot. **(b) :** Solution.

Figure 5.10: In Figure 5.10a, 2D workspace and robot are depicted, while Figure 5.10b shows the found solution. The initial configuration of our robot is denoted in purple, and the goal configuration is shown in red.

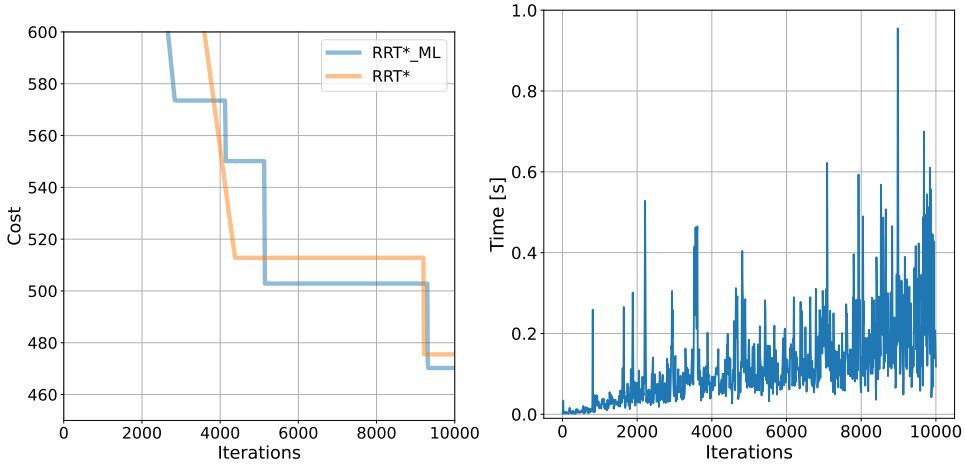
All parameters are the same as in the 2D $\mathcal{C}$-space, but now the robot is represented in 3D $\mathcal{C}$-space as an 'L'-shaped polygon, with its configuration denoted by $q = (x, y, \alpha)$. The cost is computed with respect to the rotation angle of the robot, as follows:

$$\text{cost} = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (\alpha_1 - \alpha_2)^2},$$

where $q_1 = (x_1, y_1, \alpha_1)$, and $q_2 = (x_2, y_2, \alpha_2)$. This allows the algorithm to search for not only the shortest path but also a path where the robot makes fewer rotations.

In Figure 5.10, the selected workspace and the solution found by RRT*_ML are shown. The solutions found from both RRT*_ML and RRT are mostly similar, so it is unnecessary to show the solution found by RRT*.

According to the Figure 5.11a, the implemented RRT*_ML algorithm finds the first solution with fewer iterations. However, as illustrated in Figure 5.11b, the computational time for prediction ((Algorithm 5)) increases with each iteration. As a result, RRT* is significantly faster, completing 5000 iterations in ~30 seconds, while RRT*_ML took ~1041 seconds for the same number of iterations. After 10000 iterations, the $\mathcal{X}_{\text{free}}$ dataset contains 6551 datapoints, and the $\mathcal{X}_{\text{obs}}$ dataset contains 19246 datapoints.



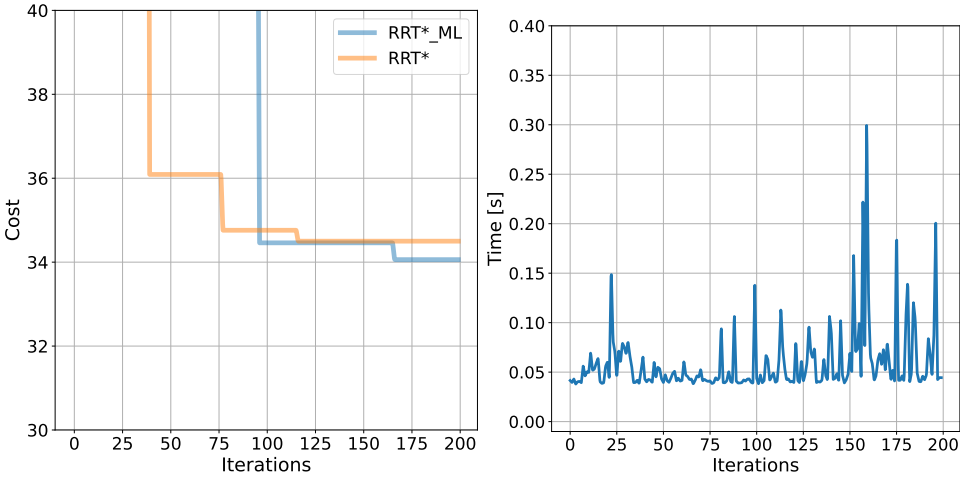
(a) : Convergence graph of RRT*_ML and RRT*.

(b) : Time taken to learn predicting points (Algorithm 5) in the 2D workspace with the 3D $\mathcal{C}$-space (5.10a) to be in the C_{free}.

Figure 5.11: Convergence and time graphs for the 3D $\mathcal{C}$-space.

Computation in 6D $\mathcal{C}$-space is significantly more complex. Therefore, tests were conducted for 200 iterations. In Figure 5.13, the workspace is shown with the robot represented as a purple cube and obstacles represented as green cubes. The configuration of the robot is denoted by $q = (x, y, z, yaw, roll, pitch)$. The cost is computed as the Euclidean distance, with the reference point of the robot located at its center of mass. This map is 25x25x25, so the s was changed to 3 and the r to 12. The p_{goal} remains at 0.05.

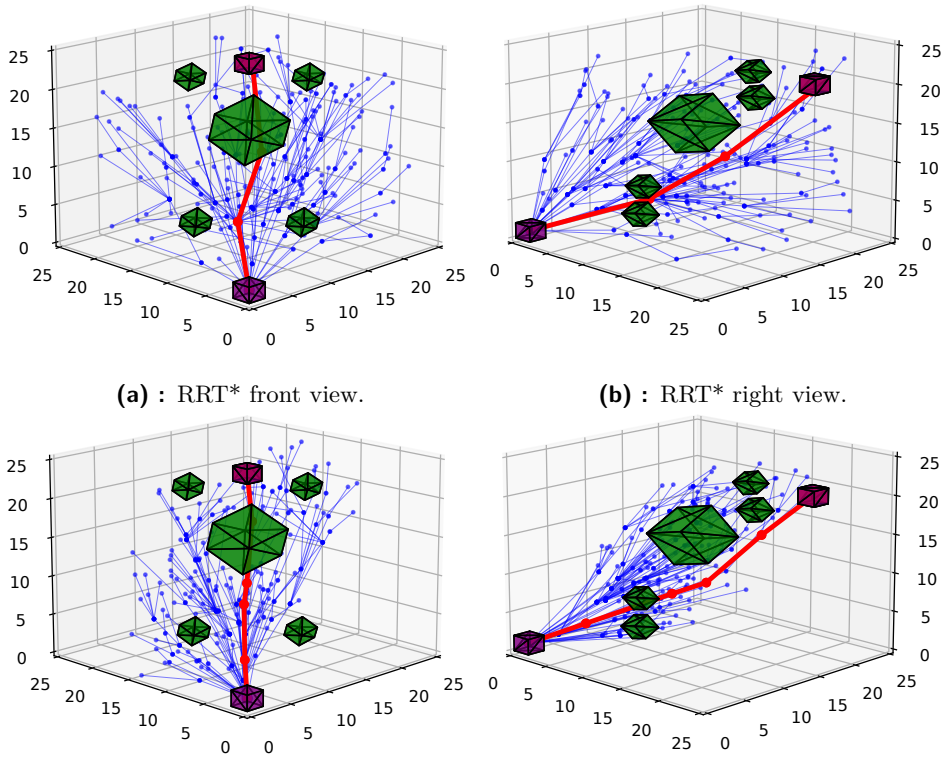
According to the graph illustrated in Figure 5.12a, the implemented RRT*_ML converges to the optimal solution faster. However, in this case, RRT* found the first feasible solution earlier. Similar to previous tests, RRT*_ML is slower than RRT*. The runtime for RRT*_ML over 200 iterations is ~504 seconds, while the runtime for RRT* is ~379 seconds. Figure 5.12b shows the time taken to predict a sample in $\mathcal{C}_{\text{free}}$, and as before, this time increases over iterations. This increase is due to the $\mathcal{X}_{\text{free}}$ and $\mathcal{X}_{\text{obs}}$ datasets accumulating more data points over time, making them larger. After 200 iterations, the $\mathcal{X}_{\text{free}}$ dataset contains 440 datapoints, and the $\mathcal{X}_{\text{obs}}$ dataset contains 46 datapoints.



(a) : Convergence graph of RRT*_ML and RRT*.

(b) : Time taken to learn predicting points (Algorithm 5) in the 3D workspace with the 6D $\mathcal{C}$-space (4.8) to be in the C_{free}.

Figure 5.12: Convergence and time graphs for the 6D $\mathcal{C}$-space.



(a) : RRT* front view.

(b) : RRT* right view.

(c) : RRT*_ML front view.

(d) : RRT*_ML right view.

Figure 5.13: An example of a 3D workspace with a 6D $\mathcal{C}$-space, showing the found solution path. The green color represents obstacle object. The purple color indicates object representing our robot. The red line shows the found solution path, and the blue lines illustrate the paths explored in the $\mathcal{C}$-space.

5.3 Result

As a result, it can be concluded that the implemented RRT*_ML algorithm utilizing machine learning methods performs better in terms of converging to the optimal solution in certain situations. Most of the time, the enhanced version of RRT* discovers the first possible solution earlier than other planners. However, in challenging scenarios, such as in Figure 5.5, it behaves similarly to RRT*. In other instances, it achieves either equivalent or superior convergence to the optimal solution compared to RRT*. However, implementation in Python and computationally intensive function to predicting points (Algorithm 5), causing RRT*_ML to work much slower than other planners.

Chapter 6

Conclusion

In this study, the limitations of sampling-based algorithms were addressed. Specifically, in scenarios characterized by numerous obstacles or narrow passages, these challenges were tackled through the integration of machine learning techniques. It was identified that while sampling-based methods offer flexibility and computational efficiency, exploring cluttered configuration spaces can be challenging, potentially resulting in suboptimal path generation.

To enhance the efficiency and adaptability of sampling-based algorithms, an approach was proposed to selectively sample points only from obstacle-free spaces, leveraging machine learning to learn the configuration space. Through testing and comparison with planners from the Open Motion Planning Library (OMPL) [20], including RRT$^{\#}$ (sharp) [1], RRT^{X} static [16], and Informed RRT* [6], the effectiveness of our approach was evaluated.

Our results demonstrate that the implemented enhanced RRT* algorithm, leveraging machine learning methods, performs better in terms of converging to optimal solutions in certain situations. The enhanced version often discovers the first possible solution earlier and achieves equivalent or superior convergence to the optimal solution compared to traditional RRT* [12].

While our approach exhibits promising results, it is essential to acknowledge its limitations. In challenging scenarios, such as those demonstrated in Figure 5.5, the enhanced RRT* behaves similarly to traditional RRT*. Additionally, the implemented algorithm runs significantly slower due to its implementation in Python and the method used to predict points in $\mathcal{C}_{\text{free}}$ space (Algorithm 5). Although this method is crucial for our implementation, it takes considerable time to predict points, and as the $\mathcal{C}$-space is learned, more data points are needed to store. However, the main objective is to implement and assess the performance of the method in terms of convergence to the optimal solution through iterations and this goal is reached.

Overall, this research contributes to the advancement of sampling-based path planning algorithms by mitigating their limitations through the integration of machine learning techniques. One potential improvement could involve exploring the integration of these techniques into other path planners, such

as RRT$^{\#}$, to further enhance their capabilities in complex environments. Additionally, implementing the algorithm in C++ could help reduce runtime and improve efficiency.

Bibliography

[1] Oktay Arslan and Panagiotis Tsiotras. The role of vertex consistency in sampling-based algorithms for optimal motion planning. 2012.

[2] Oktay Arslan and Panagiotis Tsiotras. Machine learning guided exploration for sampling-based motion planning algorithms. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2646–2652, 2015.

[3] Christopher M. Bishop. Pattern recognition and machine learning (information science and statistics). 2006.

[4] Jaroslaw Drapala. Kernel density estimator explained step by step. 2023. `https://towardsdatascience.com/kernel-density-estimation-explained-step-by-step-7cc5b5bc4517`.

[5] Jaroslaw Drapala. Kernel density estimator for multidimensional data. 2023. `https://towardsdatascience.com/kernel-density-estimator-for-multidimensional-data-3e78c9779ed8`.

[6] Jonathan D. Gammell, Timothy D. Barfoot, and Siddhartha S. Srinivasa. Informed sampling for asymptotically optimal path planning. *IEEE Transactions on Robotics*, 34(4):966–984, 2018.

[7] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] S. Gottschalk, Ming Lin, and Dinesh Manocha. Obbtree: A hierarchical structure for rapid interference detection. *Computer Graphics*, 30, 10 1997.

[9] Markus Hafellner. Object detection. *Bitmovin*, 2019. `https://bitmovin.com/object-detection`.

[10] Peter Hart, Nils Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[11] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. pages 2222–2229, 2009.

[12] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011. https://doi.org/10.1177/0278364911406761.

[13] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

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

[15] Steven M LaValle et al. Rapidly-exploring random trees: A new tool for path planning. *The annual research report*, 1998.

[16] Michael Otte and Emilio Frazzoli. RRT^X*: Real-Time Motion Planning/Replanning for Environments with Unpredictable Obstacles*, pages 461–478. Springer International Publishing, Cham, 2015.

[17] Ahmed Hussain Qureshi, Yinglong Miao, Anthony Simeonov, and Michael C. Yip. Motion planning networks: Bridging the gap between learning-based and classical motion planners. *CoRR*, abs/1907.06013, 2019.

[18] Markus Rickert, Arne Sieverling, and Oliver Brock. Balancing exploration and exploitation in sampling-based motion planning. *IEEE Transactions on Robotics*, 30:1305–1317, 12 2014.

[19] D.W. Scott. Multivariate density estimation: Theory, practice, and visualization. John Wiley & Sons, New York, Chicester, 1992.

[20] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. https://ompl.kavrakilab.org.

[21] Yuhang Xie, Chen Yihong, Ziyi Wang, and Yangjun Ou. Ai intelligent wayfinding based on unreal engine 4 static map. *Journal of Physics: Conference Series*, 2253:012016, 04 2022.

Appendix A

Attachments

Inforamation about attached file, how to run code, what i have, also add latex code to attachments