AN ASYMPTOTICALLY OPTIMAL PATH PLANNING METHOD WITH CUBIC BÉZIER SPLINE

by

Zifan Fei

Submitted in partial fulfillment of the requirements
for the degree of Master of Applied Science

at

Dalhousie University
Halifax, Nova Scotia
June 2023

This thesis is dedicated to my fiancée, Yi.

Table of Contents

List of Tables

List of Figures

Abstract

This dissertation introduces a novel path planning algorithm for robotics, known as Informed SRRT$^{\#}$. Departing from traditional RRT algorithms that utilize Euclidean metrics, our algorithm integrates a local planner from SRRT, accommodating both external and internal constraints. The path to the target area is calculated using parameterized cubic spline adapted from SRRT, an approach that avoids reliance on intensive numerical methods. We introduce two extra lines at the Bézier spline's endpoints, which facilitates the rewiring process. A minimum of three state connections need adjustment during rewiring to meet kinematic constraints. The algorithm guarantees a G^2 continuity of curvature for the path within upper-bound constraints. The effectiveness of the proposed method is demonstrated through various channels: Python-based simulations, Gazebo/Rviz — a robot simulator and visualization tool in Robot Operating System, and real-world scenarios. In real-world experiments, the algorithm successfully maneuvered TurtleBot3 past obstacles in the physical map, leading to a smooth, streamlined and optimal navigation approach. Throughout its operation, the search area progressively decreased and the path maintains smoothness, showcasing the algorithm's ability to enhance and concentrate its path planning with precision. Our results reveal that the new algorithm identifies shorter paths than SRRT while achieving the same number of node sampling iterations. However, these enhancements come with a trade-off, as the computational time of the proposed method is slightly higher compared to traditional methods.

List of Abbreviations and Symbols Used

$C(\nu)$	Coriolis and centripetal matrix
$D(\nu)$	damping matrix
$J(\nu)$	cross-covariance matrix
$J(\nu)$	rotational inertia
M	inertia matrix
β	maximum turning angle
η	position and orientation in world coordinate
∞	infinity
κ	curvature
$\mathbb{N}$	set of non negative integer numbers
$\mathbb{R}^n$	set of $n \times 1$ real vectors
$\mathbf{I}^n$	$n \times n$ identity matrix
$\mathcal{X}$	configuration state space
μ	center point of hyperellipsoid in $\mathbb{R}^n$
ν	linear and angular velocities
$\det(\cdot)$	determinant of a matrix
diag	diagonal matrix
ψ	angular orientation
σ	a continuous path with a sequence of states
τ	control forces and torques
θ	angular orientation
$g(\eta)$	combined forces of gravity
lms	local minimum cost-to-come value
rhs	right-hand side value
AI	artificial intelligence
ARA*	Anytime Repairing A*
AUV	autonomous underwater vehicle
BFS	breadth-first search

FMT	Fast Marching Tree
ICC	Instantaneous Center for Curvature
IMU	Inertial Measurement Unit
Lidar	Light Detection and Ranging
LPA*	Lifelong path planning A*
ML	machine learning
PRM	Probabilistic Roadmap
RMSD	root mean squared deviation
ROS	Robotics Operating System
RRT	Rapidly-exploring Random Tree
SLAM	Simultaneous Localization and Mapping
SMC	sliding mode control
SRRT	Spline RRT
SVD	singular value decomposition
tf	A transform library package which records multiple coordinate frames in a tree structure
USV	unmanned surface vehicle

Acknowledgements

I am deeply humbled and greatly privileged to express my sincere gratitude to those who have contributed to the successful completion of this thesis.

First, I extend my heartfelt appreciation to my supervisor, Professor Ya-Jun Pan, whose meticulous guidance, insightful critiques, and unwavering patience have been fundamental in navigating through this journey. Her mentorship is not merely academic; it has been instrumental in shaping my growth as a junior research student and as a person with a sense of mission.

I am also indebted to the members of the Supervisory Committee, Professor Clifton Johnston and Professor Yuan Ma, whose invaluable feedback and thoughtful suggestions have significantly enriched my work. Their insightful critiques and challenging questions have spurred me to delve deeper and refine my arguments, fostering academic rigor and intellectual growth. Their dedication and time are highly appreciated.

I would like to express my sincere appreciation to Professor Dominic Groulx, Professor Farid Taheri and Professor Robert J. Bauer for their exceptional coordination and support as the graduate student coordinators. The successful completion of my master's thesis would not have been possible without their efficient management of the department's Graduate Studies Committee and their dedication to ensuring a smooth academic journey for all of our graduate students in these two years.

To my beloved parents and my fiancée, I owe an immeasurable debt of gratitude. Her unyielding support, encouragement, and faith in my capabilities have been my pillars of strength during this journey. The courage to strive and the resilience to persevere are lessons I have gleaned from her.

My time at the Advanced Controls and Mechatronics (ACM) Lab has been uniquely enriching. Our bi-weekly presentations in each semester have been a platform for knowledge sharing, fostering a deeper understanding of our respective areas of research. The discussions that followed these presentations were enlightening, sometimes leading to fruitful collaborations. The collective intellect and support of the ACM lab members have been a great source of inspiration, driving me to explore and reach beyond the confines of my comfort zone. For all these and more, I express my heartfelt gratitude.

The experiences and learning I have had as part of the Unmanned Surface Vehicle (USV) team in Marine Thinking Incorporated, which have been integral to my academic journey. The industry environment coupled with a strong team spirit has provided me with invaluable insights and practical experience. I am grateful to my colleagues for their collaboration and willingness to share knowledge and expertise.

Lastly, but by no means least, I acknowledge the tremendous efforts of Ms. Andrea Andriopoulos, Ms. Donna Laffin, Ms. Kate Hide and Ms. Vicki Sullivan from administrative office, whose support and assistance during my time at Dalhousie University have been indispensable. The smooth operation of the department and my academic journey owes much to their organizational skills, attention to detail, and unfailing commitment. Their tireless dedication and professionalism have been greatly enhanced our academic environment. For this, I express my profound gratitude.

In conclusion, I am profoundly grateful for the collective wisdom, guidance, and support of these exceptional individuals. The completion of this thesis is not a solitary accomplishment, but a testament to the power of collaboration and community. I stand on the shoulders of giants and it is their strength and wisdom that have allowed me to reach such heights.

Funding for this work was partially supported by the MITACS Accelerate program.

Chapter 1

Introduction

Motion planning algorithms have become increasingly prevalent in a wide range of robotic industries over the past three decades. There has been significant research on path planning in both mobile and industrial robotics. The problem is typically split into global and local planning by most authors. Numerous navigation methods from traditional robotics have been adapted to address the unique challenges faced by various emerging types of robotics. Numerous motion planning algorithms are now widely used in transportation and logistics industries, information technology industries, marine industries, manufacturing and robotics industries and many others [1].

In the transportation and logistics industries, motion planning algorithms play a critical role in enabling autonomous vehicles to navigate their surroundings safely and efficiently. For instance, self-driving cars rely on motion planning algorithms to chart optimal routes through complex urban environments, avoid obstacles, and adhere to traffic regulations. Motion planning is also considered as decision making modules of the entire autonomous system in self-driving cars [2, 3]. It is further hierarchically divided into four subsystems: route planning, behavioral decision making, local motion planning and feedback control [3]. The route planning was classified in four groups, according to their implementation in automated driving: graph search, sampling, interpolating and numerical optimization. These algorithms have the potential to transform transportation and logistics, making them more efficient and environmentally friendly.

In the information technology industries, motion planning algorithms are used in computer graphics, virtual prototyping, game and animation to generate realistic motion for virtual characters and objects. For example, motion planning algorithms can be used to generate a path inside a virtual scene for video game characters [4].

In manufacturing and industrial robots, motion planning algorithms are used to plan optimal trajectories for robot arms and end-effectors. For example, motion planning algorithms can be used to optimize the movements of a robot arm as it assembles a product on a manufacturing line [5, 6].

For marine robotics industries, motion planning algorithms are used in autonomous underwater vehicles (AUVs) and unmanned surface vehicles (USVs) to plan safe and efficient trajectories while minimizing the risks of collision or damage. For example, motion planning algorithms can be used to help AUVs and USVs navigate complex underwater or surface environments, where they may face various disturbances such as strong winds or currents. This technology has broad applications, including ocean exploration, military operations, and environmental monitoring [7, 8].

1.1 Research Motivation

The main goal of this project is to develop a global planner that can minimize the distance progressively between the initial and final states, maintain smoothness that satisfies curvature continuity, and is particularly suitable for non-holonomic robots with complex dynamic systems. It is quite challenging to strike a balance between a shorter path and a smoother path for the global planner.

The increasing use of non-holonomic robots in various environments emphasizes the significance of path planning to ensure their autonomous functionality. Past literature suggests a variety of techniques for robot optimal path planning, considering various criteria, such as time delays, traveling distances, and planned arrival time. In most applications, time and energy are critical factors. Generating the shortest path can help robots to minimize the time and save more fuel or energy taken to reach the goal point, which can extend the robot's operational time. Furthermore, it can also improve the safety level for robots. The shorter the path, the less time the robot spends navigating in the environment, which reduces the chance of unexpected accidents or collisions with dynamic obstacles.

However, when many existing solutions for optimal global path planning are directly applied to robots, they often result in piece-wise linear or sharp paths. This leads to the robot having to frequently stop, rotate, and restart, resulting in a discontinuous and time-consuming motion. Such movements can excessively drain the robot's power and unnecessarily extend its operational time. A more efficient and continuous path planning technique is needed to avoid such issues. To be more specific, several reasons are listed below:

1. A smooth path can help the robots or conveyances avoid jerky movements. Jerky motion can make it unstable or even cause damage (high-level noise or vibration) to the sensors, electronics and other dynamic systems. This is especially important for systems that operate at high speeds or have high inertial loads.

2. While quantifying the exact energy savings can be challenging, a smooth path is generally believed to reduce the system's energy consumption by minimizing unnecessary accelerations and decelerations. This concept is rooted in the fundamental principles of motion physics, where sudden changes in velocity require more power compared to maintaining a steady speed.

3. A smooth path can lead to enhanced control precision for controllers such as sliding mode control (SMC), thus yielding improved performance as demonstrated in previous research [9].

1.2 Literature Review of Motion Planning for Mobile Robots

1.2.1 Optimal Paths for Holonomic Robots

To quickly and effectively generate a path, there are a multitude of algorithms proposed in the past 50 years as shown in Fig. 1.1.

1. Search Based Algorithms

When planning a route, one can establish a route using graph search based algorithms that explore the various states in a grid map. Dijkstra's algorithm [10] and A* [11] are the two most fundamental algorithms developed in the mid 20th century. Both methods search the state spaces which are represented as occupancy grid. Dijkstra's algorithm is designed to efficiently find the shortest path between two nodes in a graph. The algorithm achieves this by exploring the neighboring nodes of known nodes with the smallest weight, gradually building up a picture of the entire graph. The weight of a node is determined by the sum of all the weights of the edges leading to that node. Once the node with the smallest weight is found, it is added to the set of visited nodes, and the algorithm continues to explore its neighbors. By systematically searching for the node with the smallest weight, Dijkstra's algorithm

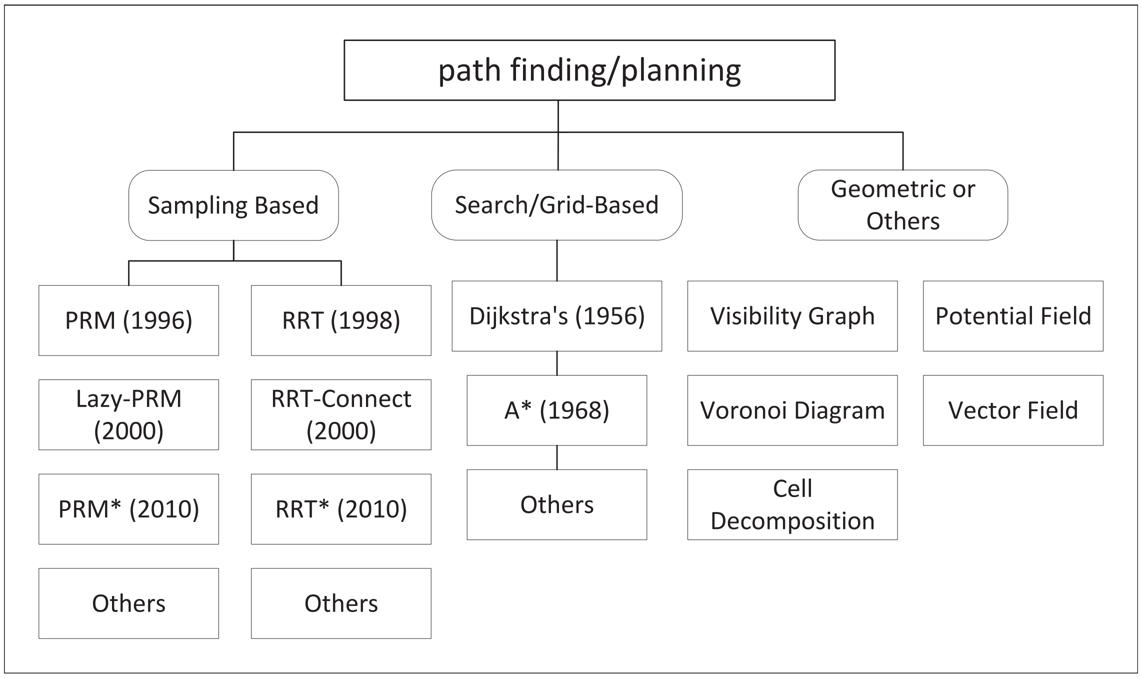


Figure 1.1: Overview of Path Planning Algorithms

is guaranteed to find the shortest path between the source and destination nodes, assuming that there are no negative weight edges in the graph. Based on the Dijkstra's algorithm, A* uses heuristics to prioritize the search in a specific promising direction, which can make it faster and more efficient in some cases.

After that, many of the improved methods based on A* was proposed, some of the popular ones are D* [12], D*-lite [13], ARA*[14], Weighted A* [15], Jump Point Search [16] etc. All these algorithms similar to A* that use various optimistic heuristic functions to guide grid cell expansion.

2. Sampling Based Algorithms

Sampling-based planners such as the Rapidly-exploring Random Tree (RRT) [17] and the Probabilistic Roadmap (PRM) [18] have revolutionized pathfinding in complex environments. These methods overcome the challenges that traditional grid mapping techniques face when dealing with higher dimensional spaces or densely clustered obstacles. Notably, they operate without the necessity for a grid map, thereby providing more efficient alternatives. Some popular sampling based algorithms are listed in Fig. 1.2.

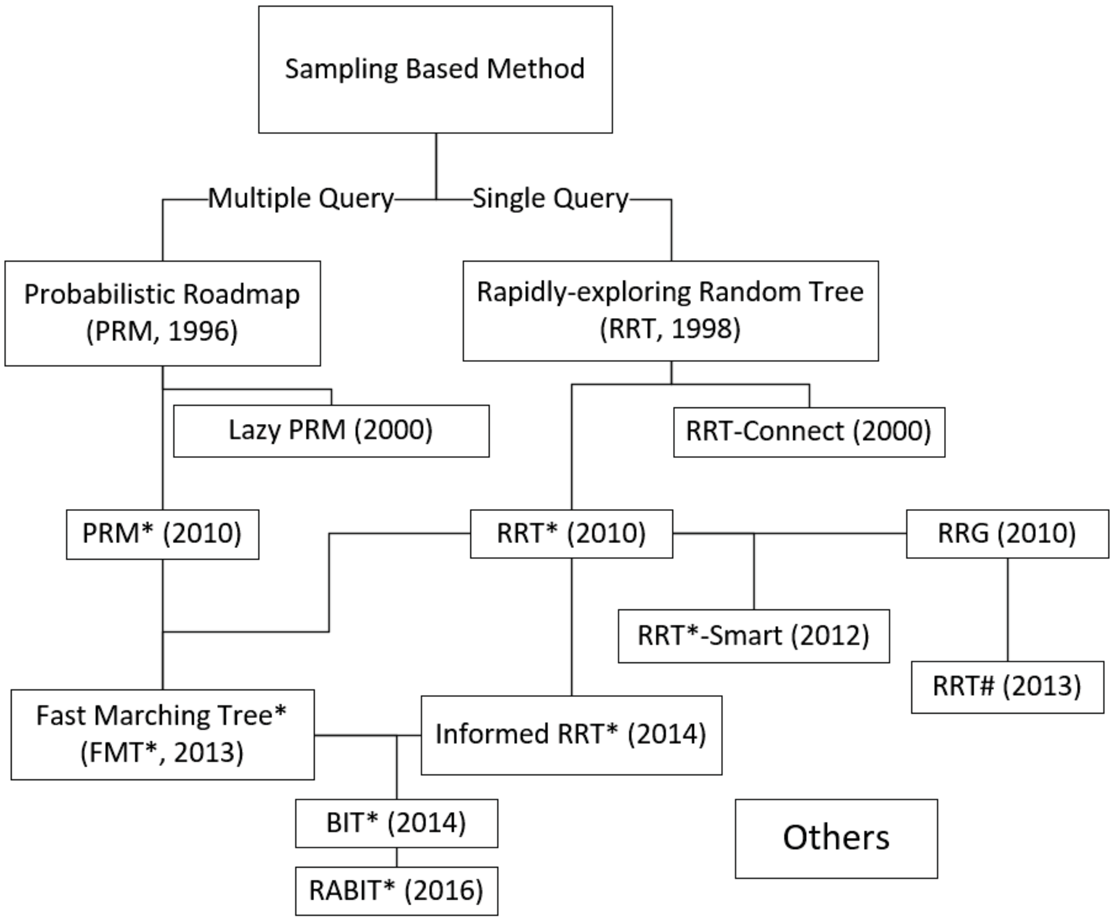


Figure 1.2: Overview of Sampling Based Planning Algorithms

RRT, introduced by LaValle in 1998, and PRM, developed by Kavraki and colleagues in 1996, have distinct differences in their methodologies. The primary variation between them lies in the exploration approach. PRM invests time to explore the entire space, sampling nodes in the process. This approach facilitates a broad overview of the space, mapping out multiple potential paths. Conversely, RRT is a single query method with its primary focus on identifying one viable path from start to end as quickly as possible.

Two significant variants have been developed based on these methods, each bringing unique contributions to the field. Lazy-PRM [19] improves the efficiency of the PRM algorithm by delaying the collision checking until a path is actually needed. This greatly reduces the number of collision checks and prunes the roadmap of useless configurations. RRT-connect [20] enhances the performance of RRT by incrementally building two RRTs, one from the start configuration and one from the goal configuration. The trees are grown until they connect and collision checking is performed to ensure a safe path. This method can be implemented synchronously or asynchronously, and is effective in environments with obstacles and narrow passages.

In 2010, Karaman and Frazzoli proposed famous PRM* and RRT* algorithms [21] to enhance path quality for sampling based planners, which immediately garnered the attention of numerous researchers when they come out. PRM* and RRT* maintain a cost-to-come value for each node in the graph, which represents the cost of the best path found so far that passes through that node. In the meanwhile, they include a rewiring step for every new node found and all its neighbour nodes. Therefore, these two algorithms guarantees almost surely asymptotically optimal. After RRT* and PRM* algorithms, many variants proposed based on one or both of them, such as RRT$^\#$ [22], Fast Marching Tree (FMT) [23], Informed-RRT* [24]. Some of these sampling based algorithms are discussed in detail in Chapter 2.

1.2.2 Smooth Paths for Non-holonomic Robots

The objective of studying path planning for a non-holonomic robot is to find the best possible route from its starting point to its destination. The ideal smooth trajectory should have minimal tracking error, require the shortest possible time and distance to travel. Deviations from the planned path can result in tracking errors, which can lead to collisions with obstacles or even mission failure. Moreover, tracking errors can increase travel time and distance because the robot needs to make additional adjustments to stay on course. This happens when the robot transits from a straight path to a curved one or at a point of inflection, where the robot can easily stray from its intended course due to abrupt changes in direction. Therefore, careful path planning is essential to reduce tracking error and several techniques had been intensely investigated to achieve smoother paths. Merely generating collision-free routes that consist of waypoints connected by straight lines, as done in the subsection 1.2.1, is seldom beneficial in addressing motion planning challenges. Specifically, certain techniques are introduced to incorporate a vehicle's kinematics and differential constraints to effectively determine a smooth and even optimal path.

Line and Circle

1) Dubins Curve:

Based on the theorem proven from Mathematician L.E. Dubins, all shortest paths between two configurations (positions and orientations) are either of the form C, S, C, or C, C (angle > *pi*), C, while C represents an arc that the robot could turn at its minimum turning radius and S represents a straight line. Further, any of the C or S segments could be of length zero [25]. Two general types are shown in Fig. 1.3.

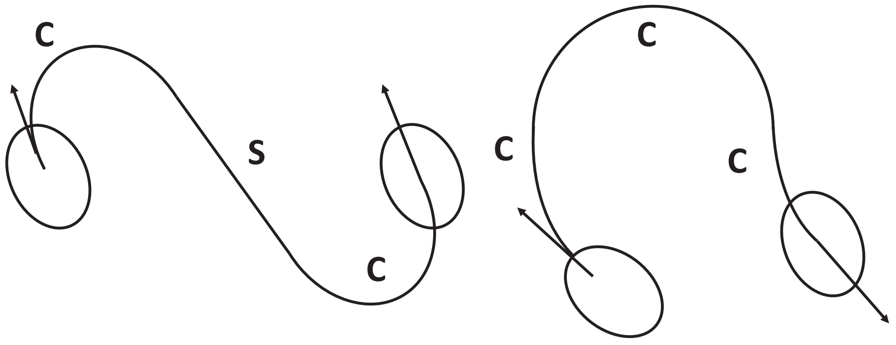


Figure 1.3: Two Types of Dubins Curves

2) Reeds-Shepp Curve:

In 1990, Reeds and Shepp extended Dubins Curve into nine different types of paths, which can be classified as an arc, a straight line segment and a cusp [26]. The ability for a robot to move both forward and backward is showcased in three classical examples depicted in Fig. 1.4:

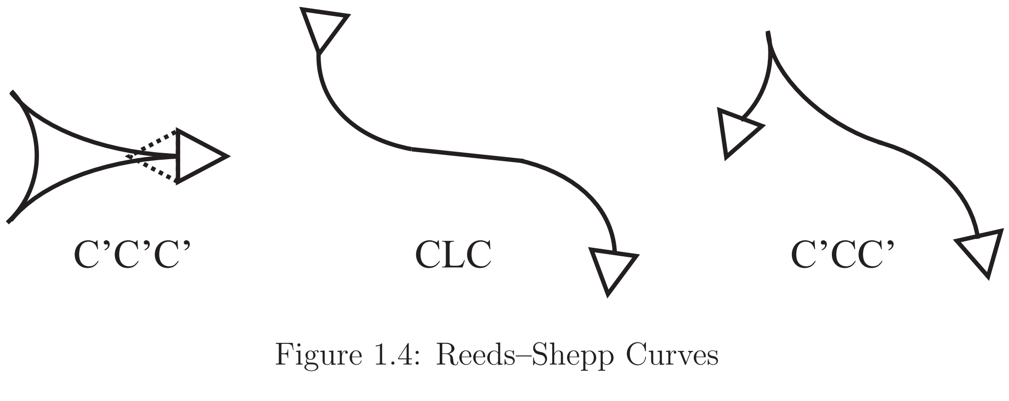


Figure 1.4: Reeds–Shepp Curves

- C'C'C': This refers to a sequence of maneuvers using a triple reverse-curve path. The vehicle (denoted as triangle) makes all its maneuvers in the reverse gear. It starts by turning in one direction in reverse, then makes another turn in the opposite direction while still in reverse, and finally makes another turn in the same direction as the first, still in reverse.

- CLC: This refers to a sequence of maneuvers using a curve-line-curve path. This typically begins with a turn (either left or right), followed by a straight-line segment, and ends with another turn (left or right). In our example, the path could start with a left turn, then proceed in a straight line, and finally end with a right turn.
- C'CC' or CC'C: This refers to a sequence of double reverse-curve (curve) attached to the both ends of the curve (reverse-curve) path. In our example, the vehicle starts by turning in one reverse direction, then makes a turn in the opposite direction, and finally makes another turn while moving in reverse as the first.

Note that "C" refers to a curve (turning maneuver), "L" refers to a straight line segment, and the apostrophe (') indicates the reverse direction.

These methods exist discontinuities at the junction of the line which can cause non-smooth motions and tracking errors.

Spline Curves

Designing a trajectory with a single polynomial curve that is constrained to pass through all control points presents considerable challenges. Such a trajectory can be highly sensitive to the positions of the control points and may fail to consistently produce a smooth shape. For example, even in a relatively simple map with few selected path points - such as 10 - the resulting high-degree polynomial might display undesirable oscillations. These oscillations can be particularly displayed near the curve's ends (Runge's Phenomenon [27]), resulting in a trajectory that is not acceptable. Additionally, in a scenario with 201 points, the polynomial would need to be of degree 200 to fit these points. Such a high-degree polynomial would not only oscillate excessively, but its computation would also be unfeasibly resource-intensive with currently available computing power. Given these challenges, employing spline curves becomes a sensible approach for trajectory planning.

Spline curves, defined by a set of control points, are a series of piece-wise polynomials used to represent complex shapes. These control points, which can either lie on or closely follow the curve, shape its form [28]. The biggest advantage of piece-wise polynomial curve is that a multitude of points can be fit with low-degree polynomials. The point that connect two curves are called knot. Typically, spline curves can be pieced by two or more cubic curves together, as cubic polynomials are the lowest degree polynomials that can support inflection points. The most basic type of spline curve is the natural cubic spline. However, it is not ideal for path planning because the entire curve is defined by a single system of linear equations. If any modification is made to the path, the shape of the entire curve must be reconstructed, which leads to a significant computational burden.

Hence, alternative types of spline curves that provide local control — such as B-splines, Bézier splines and Catmull-Rom splines [29] — are often preferable for trajectory planning. These splines allow for modifications in a localized section of the path without necessitating a recalculation of the entire curve.

Spline curves offer the dual advantages of low computational cost and the ability to maintain some degree of continuity **??**. There are six different types of spline curves will be introduced and discussed.

1) Linear Splines

Linear splines, as the name suggests, are essentially just composed of multiple line segments. It is a simplest interpolating curve, which means it passes through all the control points. The lack of excessive oscillations can be attributed to a trade-off made for curve smoothness. These splines are rudimentary in their nature and only offer C^0 continuity, which signifies a basic degree of position continuity.

2) Clothoid Curves

The Fresnel integrals are used to define a particular kind of curve known as a clothoid [30]. With clothoid curves, it is feasible to create trajectories that have a uniform rate of change in curvature because the curve curvature is proportional to its arc length. This allows for seamless transitions between straight sections and curved sections. However, while clothoid pairs offer smooth transitions without discontinuity, there is no analytical formula for determining the position along the path of the clothoid curve. This means that approximations and look-up tables are necessary.

3) Bézier Splines

The fundamental component of Bézier curves consists of the Bernstein polynomials [31]. These curves offer the advantage of being computationally inexpensive since their behavior is determined by control points. Placing these points correctly enables meeting the tangent and curvature constraints from start to end. Third and fourth-degree Bézier curves are widely utilized in autonomous vehicles to identify the most suitable curve for various situations, including turns, lane changes, obstacle avoidance etc. [32]. In Chapter 2, one method called SRRT incorporates cubic Bézier splines is introduced.

4) Hermite Splines

The defining parameters of a Hermite spline include the positions and their corresponding derivatives at two endpoints, typically realized as cubic Hermite splines. These splines ensure maximum C^1 continuity, implying a consistent changing slope along the curve, thereby guaranteeing G^1 tangent continuity. Nevertheless, a potential downside is the abrupt change in acceleration at each junction point. Hermite splines are a popular choice in the animation industry due to their unique properties [33].

5) Catmul-Rom Splines

Catmull-Rom splines are also C^1 continuous curves. Their principal advantage is that the control points for the spline curve are intrinsically defined by the original set of points. Therefore, only a sequence of control points needs to be specified, with corresponding tangents automatically derived from these points. To compute the value of the spline between two points (denoted as p_1 and p_2), the preceding and succeeding points (denoted as p_0 and p_3) must also be known.

6) Cardinal Splines

Cardinal spline represents a generalized form of Catmull-Rom spline, incorporating an additional shape parameter k in the calculation of tangents. When $k = 0$, the Cardinal spline simplifies to a Catmull-Rom spline.

7) B-splines

Unlike other types, B-spline curves and joints do not necessarily intersect with control points. Instead, they prioritize geometric G^2 continuity and C^2 acceleration continuity. One distinct advantage is the retention of local control points, allowing for adjustments while still preserving the overall continuity. This is achieved by sacrificing the interpolating property in favor of maintaining acceleration continuity.

1.2.3 Model of Differential Driving Mobile Robot

To create a path for a mobile robot that uses differential driving, it is necessary to understand the kinematics model. This analysis helps model the behavior of the robot's movement, which serves as the foundation for planning its trajectory.

A mobile robot that utilizes differential driving is equipped with two wheels on a single axis, and each wheel is controlled separately by its own motor. To describe the robot's movement, we can use the variables v_L and v_R to represent the velocities of the left and right wheels, respectively, and l to represent the distance between the wheels. By using v_L and v_R, Equation (1.2.1) determines the robot's linear and angular velocities, represented by v and w.

$$\begin{cases} v = \frac{v_R+v_L}{2}, \\ w = \frac{2(v_R-v_L)}{l}. \end{cases} \tag{1.2.1}$$

In Fig. 1.5, the kinematics model of a mobile robot that utilizes a differential driving mechanism is depicted.

The mobile robot's location in two-dimensional X-Y Cartesian coordinates can be described as $x(t)$ and $y(t)$, while its direction is indicated by $\theta(t)$. The linear velocities are represented by $\dot{x}(t)$ and $\dot{y}(t)$, while the angular velocity is indicated by $\dot{\theta}(t)$. The kinematics model of the mobile robot is defined as:

$$\begin{bmatrix} \dot{x}(t) \\ \dot{y}(t) \\ \dot{\theta}(t) \end{bmatrix} \begin{bmatrix} cos(\theta) & 0 \\ sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} v \\ w \end{bmatrix}. \tag{1.2.2}$$

The motion states of a differential driving mobile robot are affected by the speeds of its two wheels. If a robot with more than two wheels rotates at one point, that point is called the Instantaneous Center for Curvature (ICC). The turning radius of the robot R in Fig. 1.5 is determined by the velocities of its left and right wheels, which is defined as:

$$R = \frac{v}{\omega} = \frac{l_w}{2} \left| \frac{\omega_r - \omega_l}{\omega_r + \omega_l} \right|. \tag{1.2.3}$$

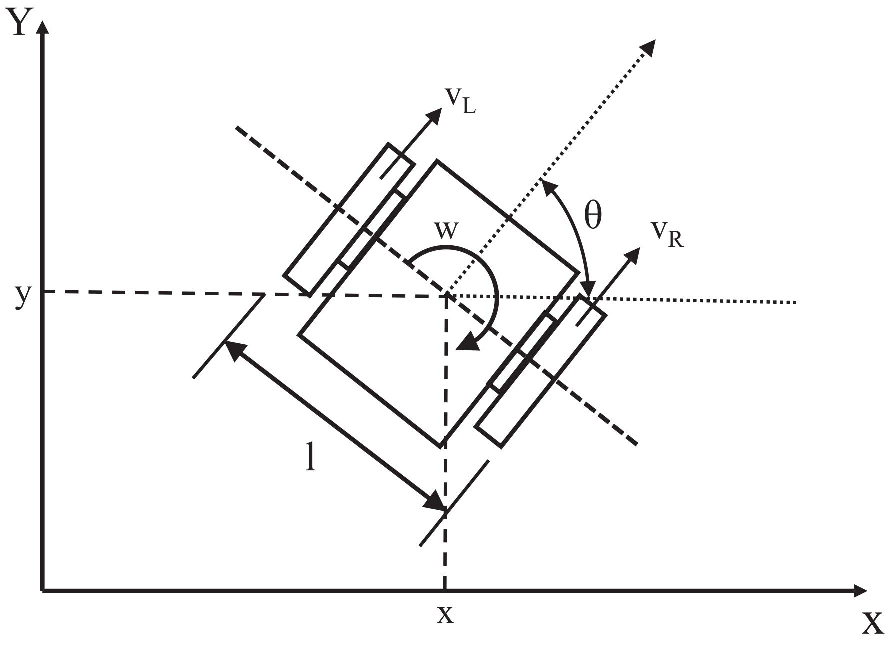


Figure 1.5: Differential Mobile Robot Model

The distance from the wheel to the ICC is proportional to its velocity, which means that changing the rotation radius will also change the velocity and acceleration of the robot.

1.3 Contributions

The main contribution of this thesis is the development of a novel path planning algorithm for non-holonomic robot system called Informed SRRT$^{\#}$. It consistently minimizes path length as well as its curvature can always be G^2 continuous at all points.

1. The algorithm ensures that the curvature's G^2 continuity is within predetermined constraints.

2. If the length, which has a maximum curvature constraint, is shorter than the extended edges between the intermediate points of three consecutive states, the algorithm can be divided into piece-wise segments of Bézier spline. To maintain curvature continuity, the second rewiring process must consider three or four consecutive states.

3. During the initial rewiring process for path optimization, sub-trees may be partially deleted, or these nodes can be reused as "new" sampled nodes if the previous vertices and edges conflict with kinematic constraints.

4. Simulations were carried out using both Python 3 and Robotics Operating System (ROS) with C++ programming, and a detailed analysis was presented to evaluate the feasibility and effectiveness of the proposed path planning algorithms under ground robot conditions. Additionally, a field test was conducted using a TurtleBot3 to further validate the performance of the algorithm in real-world scenarios.

1.4 Thesis Structure

The rest of this work is organized as follows. Chapter 2 describes the background theories used in this work. Chapter 3 gives a detailed description and algorithm implementation for informed SRRT$^{\#}$. Chapter 4 presents a comprehensive analysis of the simulation results, conducted both in Python and the Robotics Operating System (ROS) using C++. This chapter also includes a comparative analysis with conventional methods. The final chapter showcases real-world experiments conducted on the TurtleBot3 platform.

Chapter 2

Review of RRT-Based Path Planning

2.1 RRT

Rapidly-exploring Random Trees (RRT), a single-query planner initially proposed by LaValle [17], has emerged as a powerful tool in computational motion planning due to its distinctive properties. With a balanced trade-off between exploration and exploitation, RRT algorithms have shown remarkable efficiency in navigating complex, high-dimensional search spaces, particularly when the complete specification of the environment is not known in advance.

The RRT methodology revolves around the iterative construction of a search tree by incorporating randomly selected points from the search space. This stochastic, yet guided approach aids in extending the tree towards unexplored regions, thus ensuring a rapid, widespread coverage of the search space. Over time, this tree progressively approximates the connectivity of the underlying space, increasing the likelihood of finding a feasible solution even in cluttered and higher dimensional space. There are several other properties should be mentioned here as well.

- RRT is probabilistically complete.
- The Voronoi diagram property leads to the strong inclination of RRT expansion towards unexplored areas of the state space.
- Despite being well-suited for holonomic systems, this randomized planning technique can also be adapted to fit non-holonomic constraints.
- Integrating a specific state transition equation to link the nearest neighboring state and the new state, while the equation is adhering to specific non-holonomic constraints. This allows to incorporate control inputs that consider kinematics and dynamics seamlessly.

There are several versions of pseudo-code for the RRT algorithm, and Algorithm 1 represents one of them:

Algorithm 1 RRT

```
Require: $V \leftarrow \{x_{\text{Init}}\}; E \leftarrow \emptyset$;
Ensure: $\mathcal{T} = (V, E)$, $X_{sol}$
  for $i = 1, \ldots, N$ do
      $x_{\text{rand}} \leftarrow$ SampleFreeSpace;
      $x_{\text{nearest}} \leftarrow$ NearestNeighbour $(\mathcal{T}, x_{\text{rand}})$;
      $x_{\text{new}} \leftarrow$ Steer $(x_{\text{nearest}}, x_{\text{rand}})$;
      if NoCollision$(x_{\text{nearest}}, x_{\text{new}})$ then
          $V \leftarrow V \cup x_{\text{new}}$; $E \leftarrow E \cup (x_{\text{nearest}}, x_{\text{new}})$;
          if $x_{new} \in X_{goal}$ then
              $X_{Sol} \leftarrow X_{Sol} \cup x_{new}$;
              $X_{Sol} \leftarrow$ backtracking all $x$'s parent nodes from $x_{new}$ until start node.
              (optional) break the loop;
```

we illustrate the entire process of employing the RRT algorithm, specifically using a holonomic robot as an example. To initiate the process, select a random point within the map (Refer to Fig.,2.1). It's crucial to note that every iteration involves sampling a random point. This point is then connected to its nearest counterpart within the tree. This consistent and judicious selection facilitates the effective operation of the algorithm.

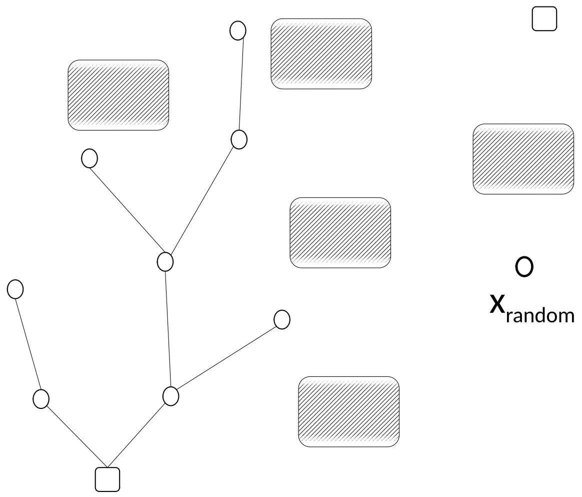


Figure 2.1: Sample a Random Point.

Next step, you need to find the one in the tree with the shortest distance to

x_{random} as shown in Fig. 2.2.

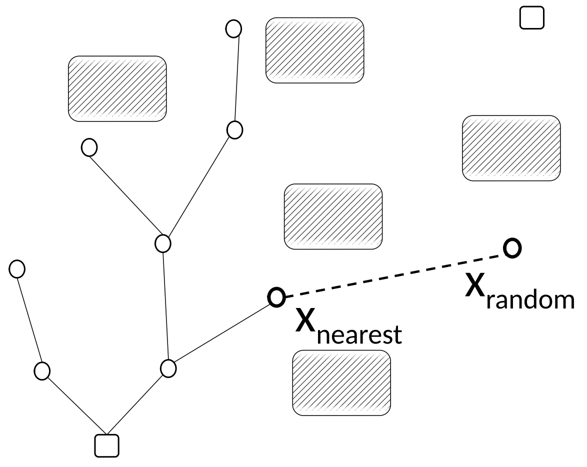


Figure 2.2: Find the Nearest Random Point in RRT.

If the distance is longer than the step size, then a new node grows in the same direction and use step size, see Fig. 2.3.

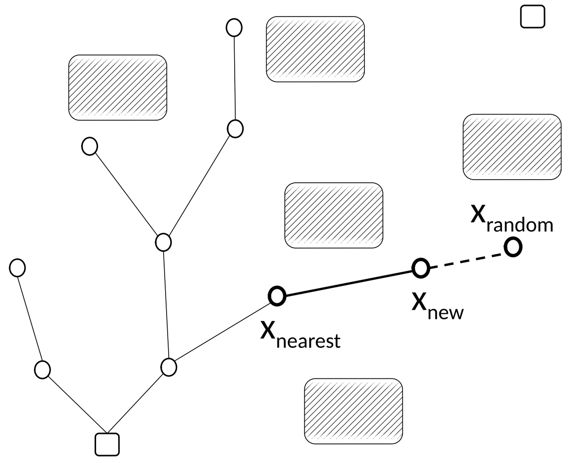


Figure 2.3: The Distance Compared to the Step Size (Case 1).

If the distance is shorter than the step size, a new node grows in the same direction with fixed stepsize or take over the position of x_{random}, as shown in Fig. 2.4. In the second case, the RRT has length variations for different extended edges.

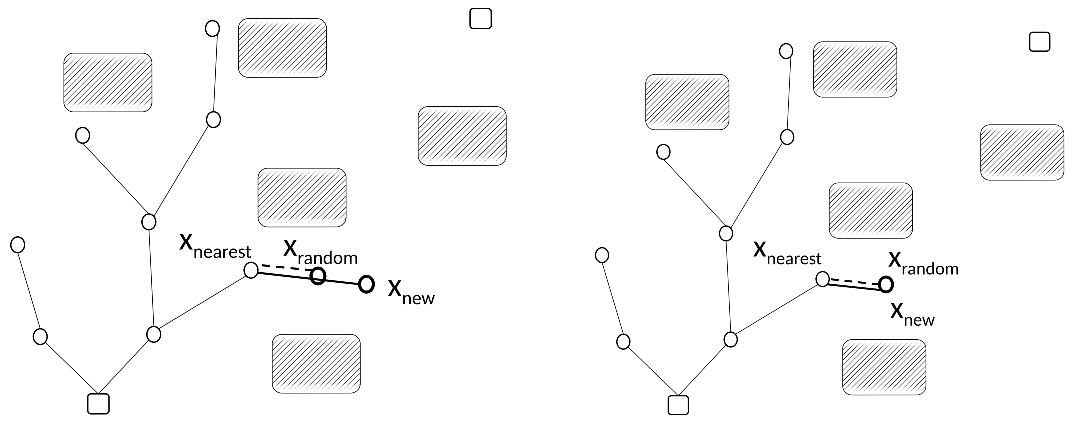


Figure 2.4: The Distance Compared to the Step Size (Case 2).

Collision checking should include both new vertex x_{new} and the edge between $(x_{\text{nearest}}, x_{\text{new}})$, see Fig. 2.5. If collision-free, then following the procedure:

- Add the vertex into the tree.
- Record the parent node of new node, which is the nearest node.
- Record the edge (optional).

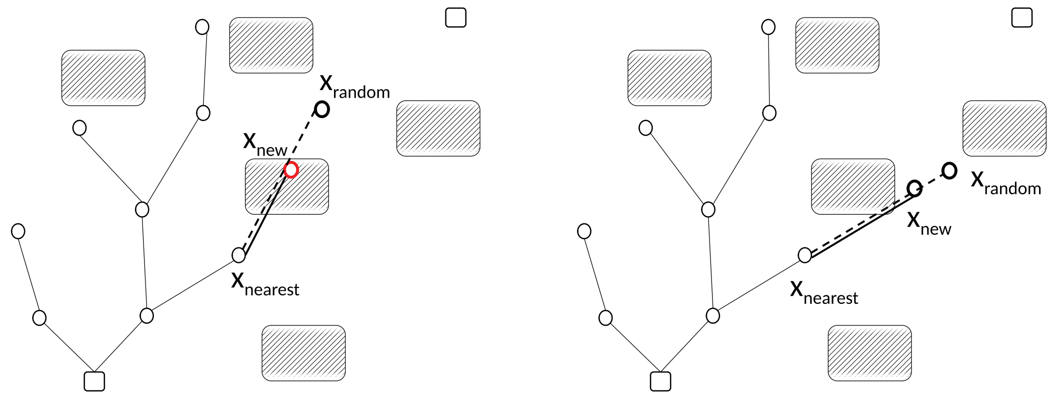


Figure 2.5: Collision Checking for New Edge and Vertex

If x_{new} is within the Range of X_{goal}, add one last edge to the x_{goal}, then you could start backtracking process from the goal node until the start node to generate a path, as shown in Fig. 2.6.

Assume the step size of all edges in RRT is always a fixed value and the robot is holonomic system, the simulation results for the RRT algorithm in Matlab are

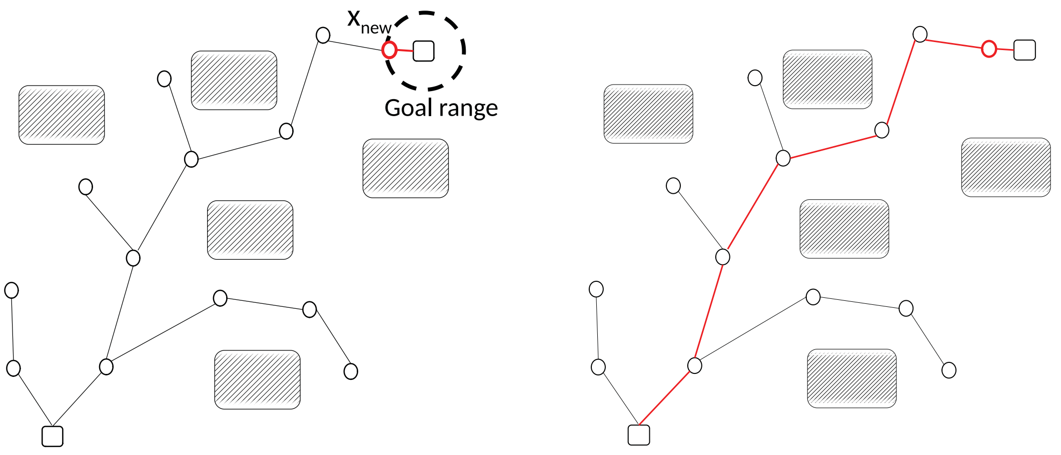


Figure 2.6: RRT Final Path

displayed in Fig. 2.7. Additionally, we conducted a comparison of the results using edges with different lengths. The choice of the metric p like edge length has a significant impact on the performance of RRT, which is also considered as a main problem in this method. If the length is too long, the final path may contain zigzag shapes, making it difficult for controllers to optimize the trajectory. On the other hand, if the edge length is too short, the algorithm may require more iterations and running time, leading to longer path lengths. For non-holonomic systems, using the Euclidean distance metric alone may not be sufficient to connect the new state with the nearest state. Therefore, combining other local planners or using different steering methods, such as applying arc or spline, can lead to better results.

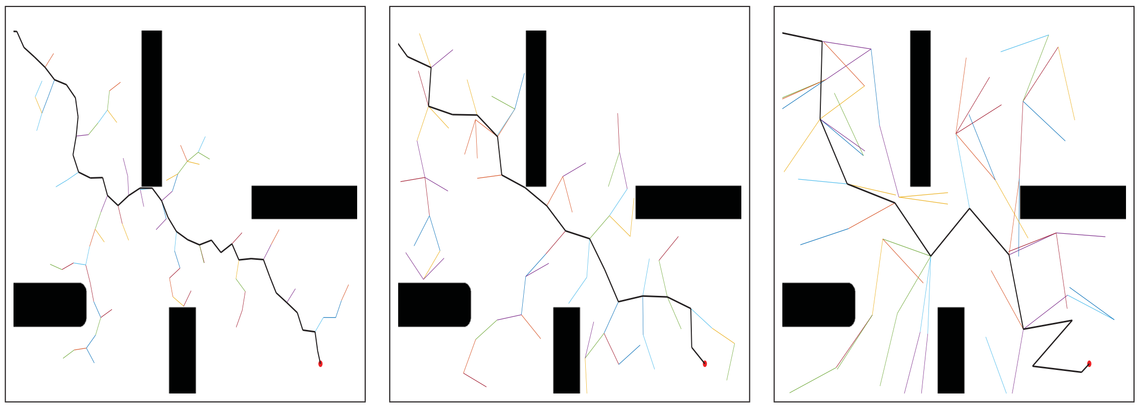


Figure 2.7: RRT Extension with Edge Length $= 5, 10, 20cm$

2.2 RRT*

Another significant drawback of the RRT method is its inability to generate a path that achieves an asymptotically optimal result. In the context of this thesis, optimality is defined as finding a feasible path with the minimal possible cost. Two researchers Karaman and Frazzoli proved RRT almost surely produce a suboptimal path and proposed RRT* to address such issue.

The pseudo-code for RRT* is shown in Algorithm 2.

Algorithm 2 RRT*

```
Require: $V \leftarrow \{x_{\text{inital}}\}; E \leftarrow \emptyset$;
Ensure: $\mathcal{T} = (V, E)$;
  for $i = 1, \ldots, N$ do
      $x_{\text{rand}} \leftarrow$ SampleFreeSpace;
      $x_{\text{nearest}} \leftarrow$ NearestNeighbour$(\mathcal{T}, x_{\text{rand}})$;
      $x_{\text{new}} \leftarrow$ Steer$(x_{\text{nearest}}, x_{\text{rand}})$;
      if NoCollision$(x_{\text{nearest}}, x_{\text{new}})$ then
          $X_{\text{near}} \leftarrow$ Neighbours$(\mathcal{T}, x_{\text{new}}, r_{\text{min}})$;
          $V \leftarrow V \cup x_{\text{new}}$;
          $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
          $c_{\text{min}} \leftarrow$ Cost$(x_{\text{nearest}})$+Dist$(x_{\text{nearest}}, x_{\text{new}})$;
          for $x_{\text{near}} \in X_{\text{near}}$ do
              if NoCollision$(x_{\text{new}}, x_{\text{near}})\wedge$Cost$(x_{\text{near}})$+Dist$(x_{\text{new}}, x_{\text{near}})<c_{\text{min}}$ then
                  $x_{\text{min}} \leftarrow x_{\text{near}}$;
                  $c_{\text{min}} \leftarrow$ Cost$(x_{\text{near}})$+Dist$(x_{\text{near}}, x_{\text{new}})$;
          $E \leftarrow E \cup (x_{\text{min}}, x_{\text{new}})$;
          for $x_{\text{near}} \in X_{\text{near}}$ do
              if NoCollision$(x_{\text{new}}, x_{\text{near}})\wedge$Cost$(x_{\text{new}})$+Dist$(x_{\text{new}}, x_{\text{near}})<$Cost$(x_{\text{near}})$
  then
                  $x_{\text{new}} \leftarrow$ Parent$(x_{\text{near}})$;
                  $E \leftarrow (E \setminus (x_{\text{parent}}, x_{\text{near}})) \cup (x_{\text{new}}, x_{\text{near}})$;
```

The formula of r_{min} is describe as Equation (2.2.1) [34], where Φ is a steering parameter such as the predetermined size of search range. Dist Function is the measured distance with specific metric p between two states. Cost function is the total sum of the distance from the input state towards the initial state.

$$r_{min} = \min\left\{\gamma(\log(\mathrm{card}(V))/\,\mathrm{card}(V))^{1/(d+1)}, \Phi\right\}; \tag{2.2.1}$$

In a two-dimensional context, we can conveniently establish the minimum radius, r_{min}, to be equivalent to Φ. This value of Φ can range from 1.2 to 2.0 of the extended edges. Prior to initiating the first rewiring process, the procedure aligns with the conventional approach used in Rapidly-Exploring Random Trees (RRT) for identifying the nearest node and a new node. Subsequently, this new node probes all neighboring nodes within the radius denoted by Φ in the tree. This process is graphically illustrated in Fig. 2.8.

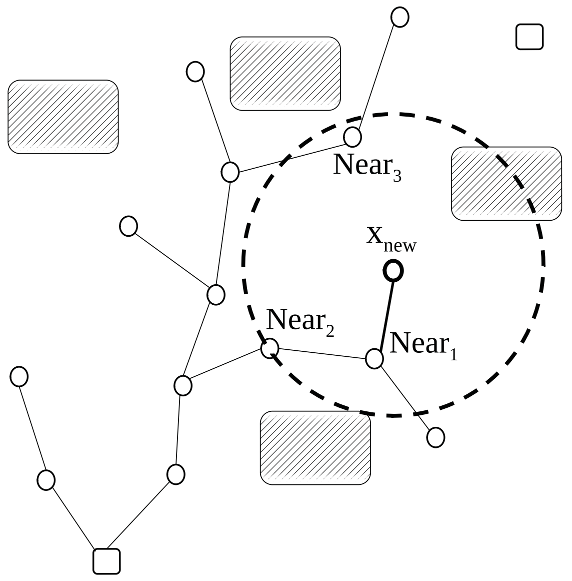
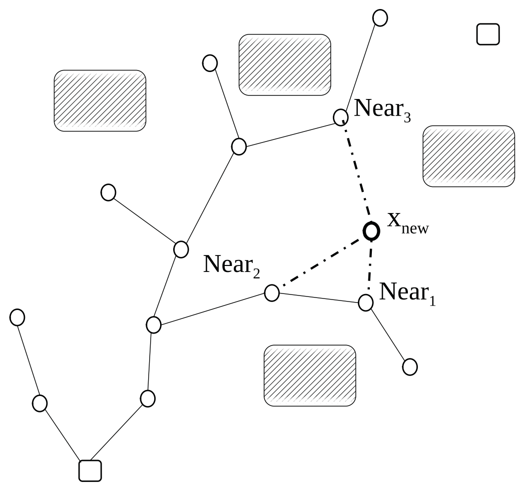
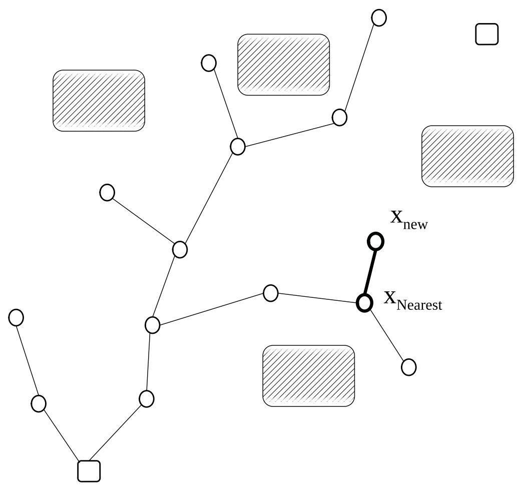


Figure 2.8: All Neighbours within the Radius Φ

The differences between RRT* and RRT is shown in Fig. 2.9. RRT* will choose the best parent node for x_{new} which has the total sum of the lowest cost towards the initial point, while RRT will only select $x_{nearest}$ which has a shortest distance between the $x_{nearest}$ and x_{new}.

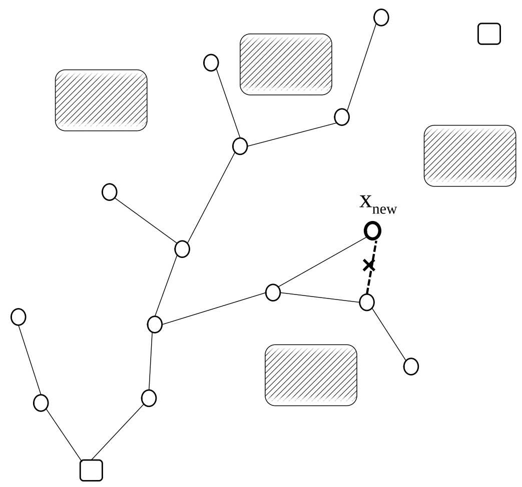


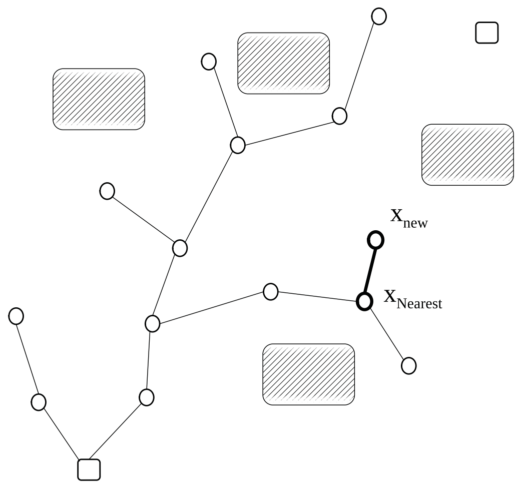


(a) The Selection Process of RRT* (b) The Selection Process of RRT

Figure 2.9: RRT* Explores More Neighbors than RRT.

In this example, the RRT* chooses $Near_2$ instead of $Near_1$ as shown in Fig. 2.10.





(a) The Selection Process of RRT* (b) The Selection Process of RRT

Figure 2.10: RRT* Chooses Superior Neighbor Nodes over RRT.

The subsequent step in the RRT algorithm involves a rewiring process that operates in reverse. Rather than searching for the best parent node for the new node, this process evaluates whether the new node is a potential better parent node for each of its neighboring nodes. If one of the neighboring nodes chooses the new node as its parent and the total cost to the starting point decreases, that neighboring node also gets optimized in the end. From Fig. 2.11, the previous total cost of $Near_3$ is

$(5+5+2)+2+2=16$. However, in the second rewiring step from RRT*, $Near_3$ chooses x_{new} as its new parent node and the total cost reduces to $(4{+}3{+}3){+}2{+}2=14$. Then it will remove the edge between $Near_3$ and its previous parent node in Fig. 2.12 and connect to x_{new} as shown in Fig. 2.13.

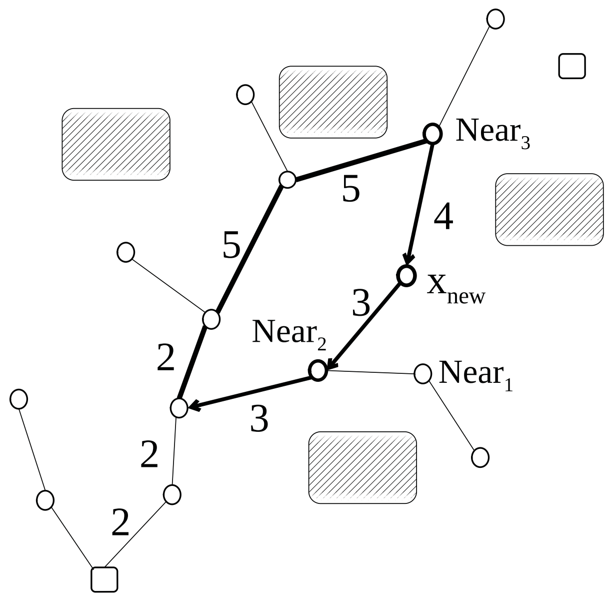


Figure 2.11: Compare the Total Cost Between x_{new} and the Preivous x_{parent}.

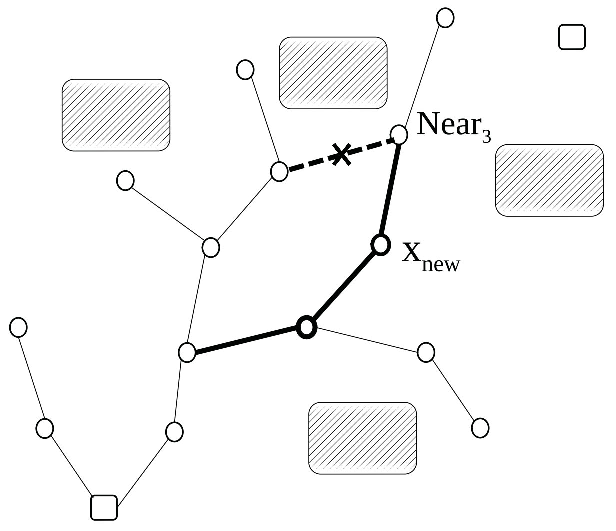


Figure 2.12: Remove the Previous Connection.

The primary limitation of the RRT* method lies in its asymptotic convergence towards the optimal path across all states within the problem domain, as pointed out

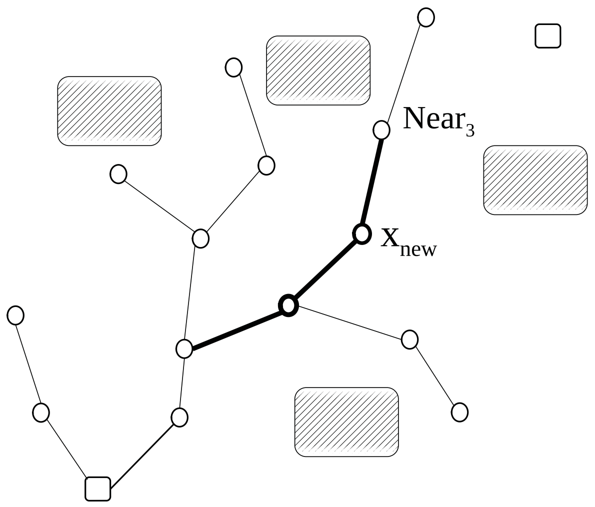


Figure 2.13: RRT* Final Path

in [24]. This characteristic proves inefficient for many real-world applications where the end goal or target area is typically known in advance. Two main areas of concern can be identified. Firstly, as demonstrated in Fig. 2.14, The RRT* method exhibits inefficiency as it unnecessarily rewires certain nodes that offer no promise of improved efficiency. Some of these nodes are even located further from the destination than the initial starting point. Secondly, it neglects to rewire all potentially beneficial nodes capable of shortening the path length. An example of this oversight can be seen with node x in Fig. 2.15.

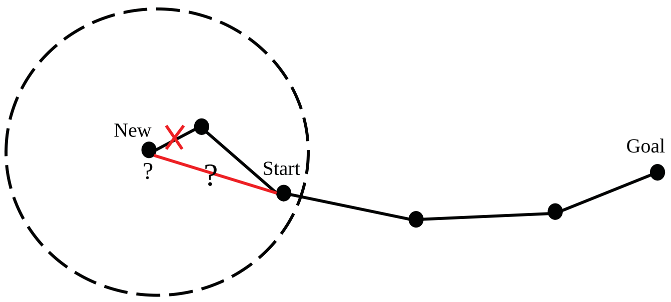


Figure 2.14: Over-exploitation of RRT*

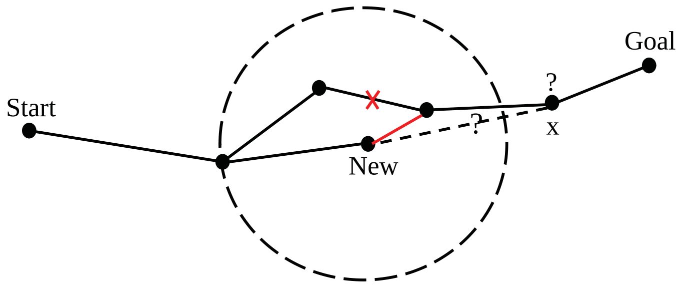


Figure 2.15: Under-exploitation of RRT*

The simulation of RRT* for holonomic robot is shown in Figs. 2.16 to 2.19.

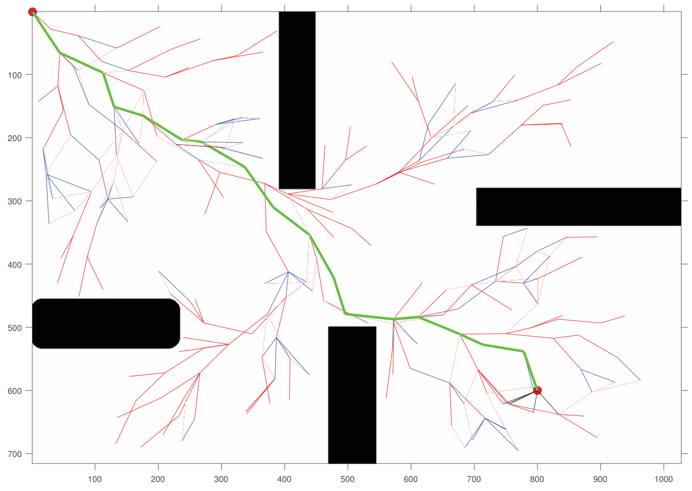


Figure 2.16: RRT*: Iteration 250

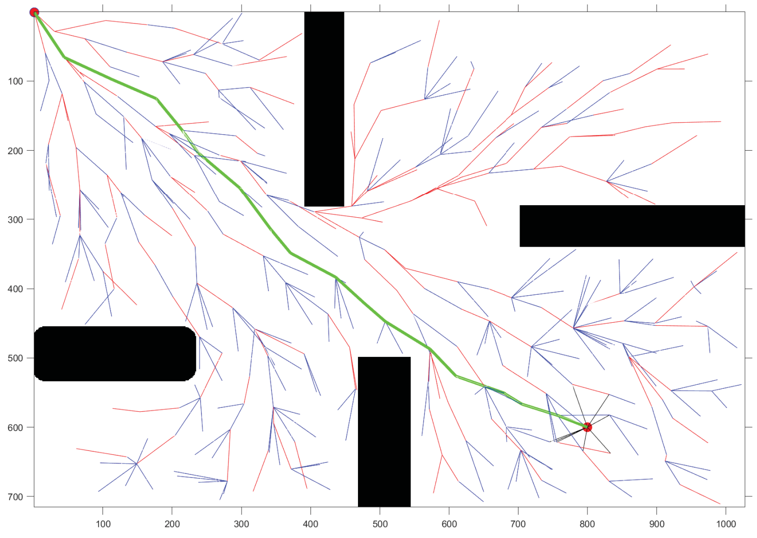


Figure 2.17: RRT*: Iteration 500

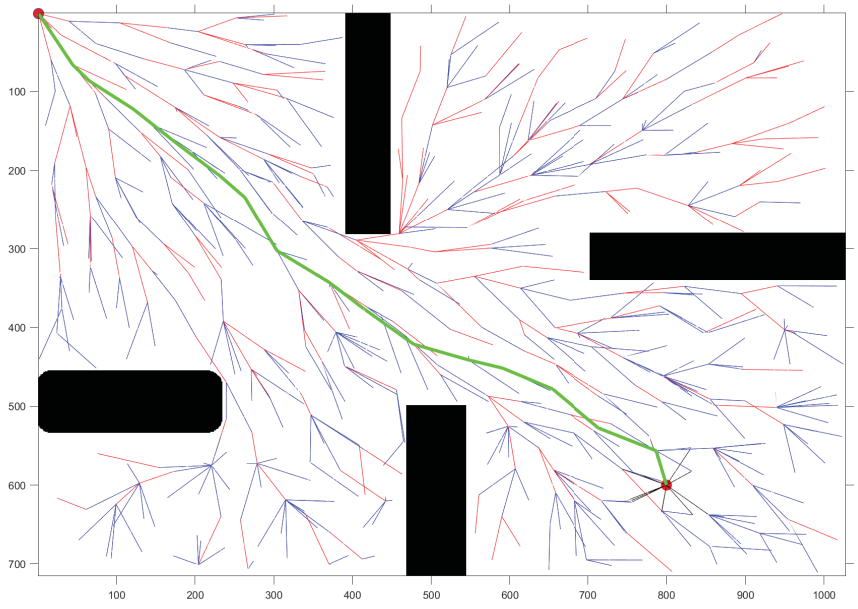


Figure 2.18: RRT*: Iteration 750

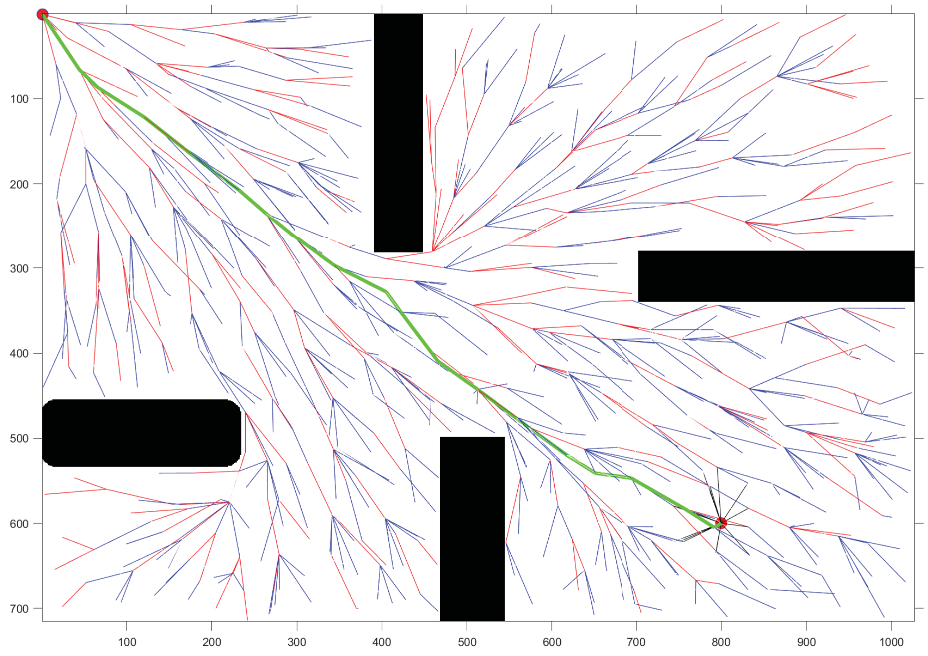


Figure 2.19: RRT*: Iteration 1000

2.3 RRT$^{\#}$ and LPA*

RRT$^{\#}$ is an improved version of RRT* proposed by Arslan and Tsiotras [22]. There are three different variants for RRT$^{\#}$. The main idea of RRT$^{\#}$ is to use a priority queue to maintain a consistent spanning tree at every iteration, where the accumulated cost-to-come of each vertex always contains optimal value. It classifies the nodes of the tree into four different types. This method is inherited from Lifelong path planning A* (LPA*), which is an improved method of A* for path plan reuse method.

Before we get into the details of RRT$^{\#}$, we need to explain the underlining concepts of LPA*. LPA* is a type of continual planning that reuses information from previous searches, combining the strengths of both A* [11] and modified DynamicSWSF-FP [35]. Since modified DynamicSWSF-FP becomes an incremental search version of breadth-first search (BFS), it adopts A* to find an optimal path at the initial stage. Later, if the environment changes or the graph is nondeterministic, LPA* can efficiently find another shortest path with the help of DynamicSWSF-FP. This approach makes LPA* a valuable tool for tasks that require frequent updates to the environment or graph. By quickly examining previously searched paths that remain unchanged, LPA* can efficiently find new shortest paths without starting from scratch. This reduces the computational overhead of continually replanning and further enhances the efficiency of LPA*.

The algorithm of LPA* are shown in Algorithms 3 to 5. Each grid maintains two estimates from the start distance (g-value and rhs-value). It is worth noting that the pseudocode I modified from [36] is just a basic outline of the LPA* algorithm, and the actual implementation can vary depending on the specific problem and requirements. In practice, the details of the algorithm may need to be adapted or optimized based on the particular use case.

In the original paper, the blocked grid D1 in the simple example and the blocked grid C4 in the complex example have actual effect on the initial LPA* search. How about a few but multiple grids change simultaneously and strongly affect the previous search path? If that happens, in Algorithm 3, every vertex that get affected and its all successors (may includes its successors of successors) should potentially update their g-values and rhs-values as well. In common cases, LPA* only considers the immediate

Algorithm 3 LPA* Main Function

```
Require: for all $t \in T \setminus t_{init}, rhs(t) = g(t) = \infty; rhs(t_{init}) = 0; PQ \leftarrow rhs(t_{init})$.
  while 1 do
      CalcShortestPath();
      if Graph changes appear then
          for Every explored vertex $t$ and edge are affected do
              Update the cost of that edge.
              VertexUpdate($t$);
```

Algorithm 4 LPA* CalcShortestPath()

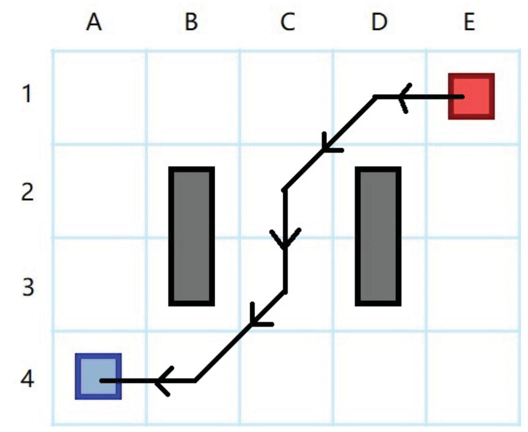
```
  while $PQ.PeekTop() < Key(t_{goal}) \vee rhs(t_{goal}) \neq g(t_{goal})$ do
      $t = PQ.pop()$;
      if $g(t) > rhs(t)$ then
          $g(t) = rhs(t)$;
          for Every immediate successor of $t$ do
              VertexUpdate($t$);
      else
          $g(t) = \infty$;
          for $t$ and every immediate successor of $t$ do
              VertexUpdate($t$);
```

Algorithm 5 LPA* Auxiliary Functions

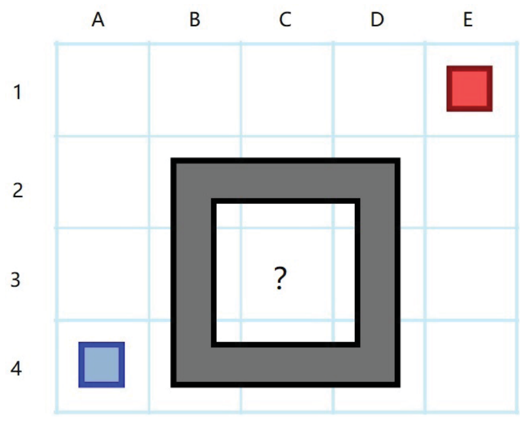
```
  VertexUpdate($t$)
  if $t \neq t_{start}$ then
      $rhs(t) = \min_{t' \in \text{Predecessor(t)}}(g(t') + c(t', t))$;
  if $t \in PQ$ then
      $PQ.delete(t)$;
  if $g(t) \neq rhs(t)$ then
      $PQ.insert(t$, Key$(t))$;


  Key($t$)
  return $[\min(g(s), rhs(s)) + h(s); \min(g(s), rhs(s))]$;
```

successors of a node, not including successors of successors. This is because LPA* is designed to work efficiently with incremental changes to the graph, and updating the successors of successors can be computationally expensive.

If a vertex's immediate predecessors are all unreachable, which means the edge costs become infinity due to the environments change, this vertex should be reinitialized and removed from PQ. Furthermore, its successors may be affected if they have no other predecessors. One special case is shown in 2.20.



(a) The Original Environment



(b) The Changed Environment

Figure 2.20: The $5 * 4$ Grid Map for LPA*

Here is another simple example to demonstrate the process of LPA*. The initial search of LPA* is almost the same as A* as shown in Fig. 2.21. The starting node $E0$ has g = 0, the key value is $[f = 4, g = 0]$. It checks the $D1$ and $E2$ and choose $D1$ as the next node and set g = 1 and key value $[f = 4, g = 1]$, $E2$ is only updated with the key value $[f = 5, g = 1]$ and g-value remains ∞. The following process is followed by this: $C2 \Rightarrow E2 \Rightarrow C1 \Rightarrow C3 \Rightarrow B4 \Rightarrow A4$ while grids $E3$, $B1$, $C4$ and $D4$ are updated with keys only.

Now the environments changed when the grid $D3$ becomes vacant and grids $C3$ and $A3$ are blocked. In this case, $B4$, $C4$ and $D4$ are previously the successors of $C3$ but now they become underconsistent and have no predecessors so they should be removed from the priority queue and set as ∞. The same situation occurs for the grid $A4$, as the successors of the grid $B4$. The grid $D3$ builds its own g-value and key based on a predecessor which has the smallest costs. In the initial graph, the g-value of the grid $D3$ is ∞ but now its rhs-value (aka one-step look-ahead value)

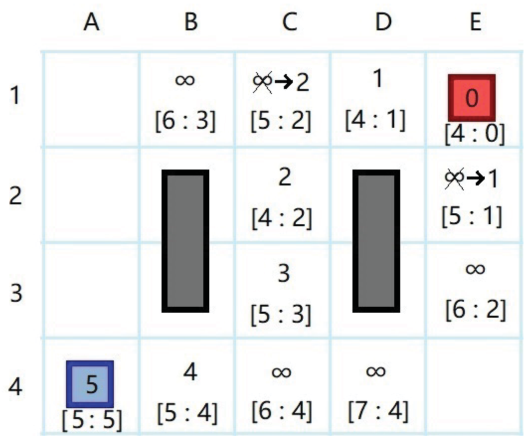


Figure 2.21: Initial Search of LPA*

becomes overconsistent. The whole process is demonstrated in 2.22.

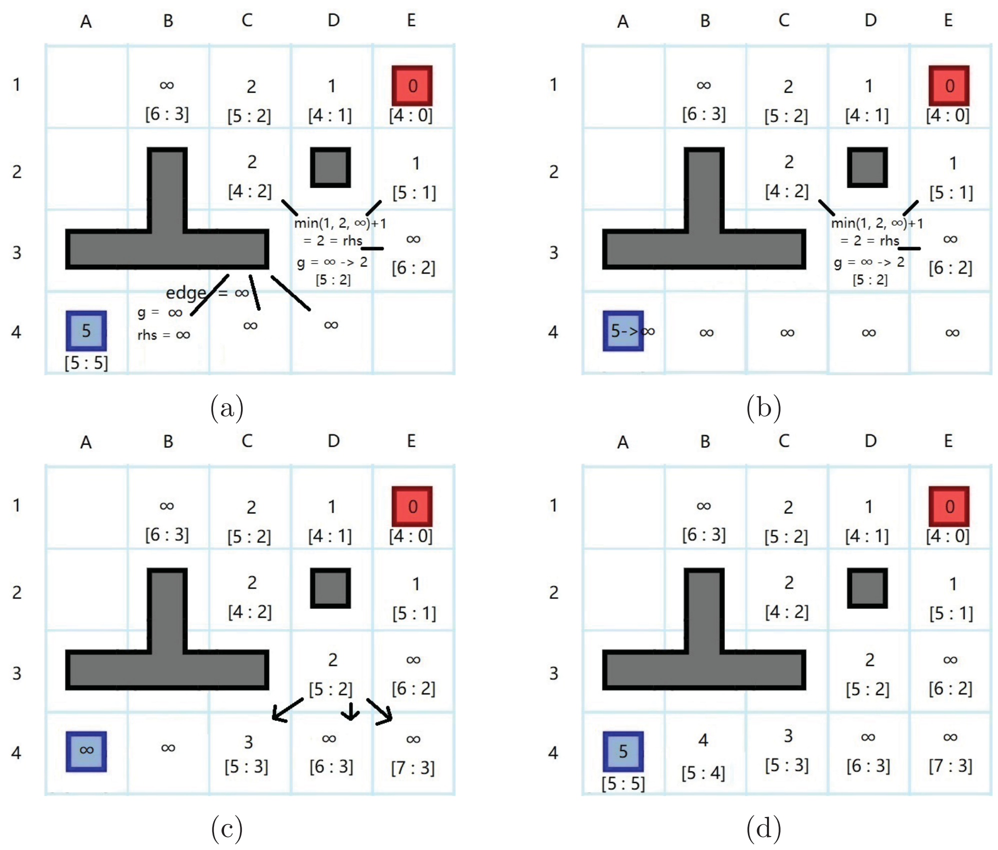


Figure 2.22: The Iterative Process of LPA*

We can easily conclude that this method does not need to perform another complete search and it only requires the checking of a few inconsistent nodes, either it is locally underconsistent (iff $g(t) \leq rhs(t)$) or overconsistent (iff $g(t) > rhs(t)$).

It continuously expands nodes until t_{goal} is locally consistent and the key of every expanded vertex is bigger than or equal to the key of t_{goal}.

The RRT$^{\#}$ algorithm incorporates the concept of overconsistent vertices from LPA* to maintain a tree of consistent vertices. When a new node is added, the algorithm considers only the neighboring nodes in the tree whose g-values are no bigger than their locally minimum cost-to-come (lmc) values (aka rhs-values in LPA*), ensuring that the constructed spanning tree of the geometric graph remains consistent after each iteration. The first part is very similar as RRT*:

- The new node is added into the tree and selects the best neighbour node as its parent node with the minimum cost-to-start.
- The authors assumed that its neighbours rewiring process is done implicitly and some neighbours may select the new node as their new parent node.

Algorithm 6 is the main body of RRT$^{\#}$.

Then the second part is the core function *ReduceInconsistency* and modified *QueueUpdate* from LPA* shown in Algorithm 7. Notice that the condition of the for loop in *ReduceInconsistency* function actually assumed that the neighbour rewiring process is completed. The *Key* function is exactly the same as LPA*.

RRT$^{\#}$ separate the nodes into four different groups and marked them as Green, Blue, Red and Black. The relationship between these four nodes are shown in Fig. 2.23. The dash arrow lines demonstrated the affiliation between two types of nodes, such as the the black node is the successor of the red node. The solid arrow lines demonstrate the transition of two types of nodes, such as the green node could become the blue node and vice versa.

- Green node is consistent and its key is a finite value.
- Blue node is inconsistent ($g \neq lmc$) and its key is a finite value.
- Red node is inconsistent because $g = \infty$ and lmc is finite.
- Black node is consistent but $g = lmc = \infty$.

In a specific iteration, only green nodes have potentials to contribute promising nodes. A subset of green nodes forms the relevant area for generating an optimal

Algorithm 6 RRT$^{\#}$

```
Require: $V \leftarrow \{x_{\text{inital}}\}; E \leftarrow \emptyset$;
Ensure: $\mathcal{T} = (V, E)$;
  for $i = 1, \ldots, N$ do
      $x_{\text{rand}} \leftarrow$ SampleFreeSpace;
      $x_{\text{nearest}} \leftarrow$ NearestNeighbour$(\mathcal{T}, x_{\text{rand}})$;
      $x_{\text{new}} \leftarrow$ Steer$(x_{\text{nearest}}, x_{\text{rand}})$;
      if NoCollision$(x_{\text{nearest}}, x_{\text{new}})$ then
          $X_{\text{near}} \leftarrow$ Neighbours$(\mathcal{T}, x_{\text{new}}, r_{\text{min}})$;
          $V \leftarrow V \cup x_{\text{new}}$;
          $x_{\text{min}} \leftarrow x_{\text{nearest}}$;
          $c_{\text{min}} \leftarrow$ Cost$(x_{\text{nearest}})$+Dist$(x_{\text{nearest}}, x_{\text{new}})$;
          for $x_{\text{near}} \in X_{\text{near}}$ do
              if NoCollision$(x_{\text{new}}, x_{\text{near}}) \wedge$Cost$(x_{\text{near}})$+Dist$(x_{\text{new}}, x_{\text{near}}) < c_{\text{min}}$ then
                  $x_{\text{min}} \leftarrow x_{\text{near}}$;
                  $c_{\text{min}} \leftarrow$ Cost$(x_{\text{near}})$+Dist$(x_{\text{near}}, x_{\text{new}})$;
          $E \leftarrow E \cup (x_{\text{min}}, x_{\text{new}})$;
          $V \leftarrow V \cup x_{\text{new}}$;
          QueueUpdate$(x_{new})$;
          ReduceInconsistency$(\mathcal{T}, X_{goal})$;
  Rebuild connections among the nodes that are affected.
```

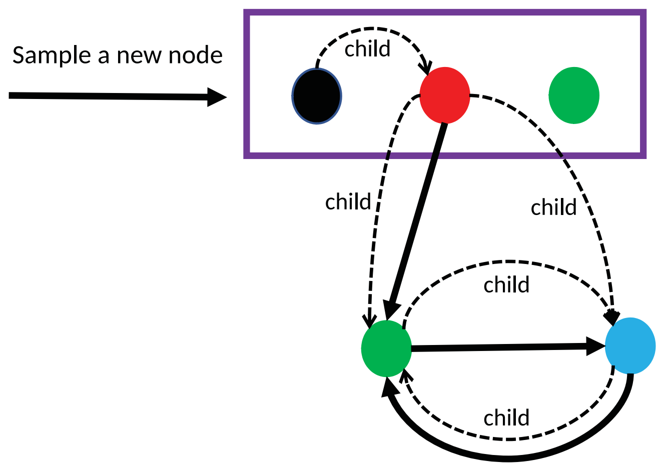


Figure 2.23: The Relationship Between Four Types of Nodes in RRT$^{\#}$

Algorithm 7 Core Functions of RRT$^{\#}$

```
**ReduceInconsistency**
while $PQ.PeekTop() < Key(t_{goal})$ do
    $t = PQ.pop();$
    $g(t) = lmc(t);$
    for every immediate successor $s$ of $t$ do
        if $lmc(s) > g(t) + c(t, s)$ then
            $s_{parent} \leftarrow t;$
            $lmc(s) = g(t) + c(t, s);$
            VertexUpdate($s$);

**QueueUpdate**
if $t \in PQ$ and $g(t) \neq lmc(t)$ then
    $PQ.update(t,$ Key$(t));$
if $g(t) \neq rhs(t)$ and $g(t) \neq lmc(t)$ then
    $PQ.insert(t,$ Key$(t));$
```

path. The other three are all non-promising. In the simulation of RRT$^{\#}$, there are some green nodes' parent nodes are blue nodes. These green nodes can also be outside of the optimal region. The reason that these blue nodes exist due to the sampled new nodes reduced the key value of t_{goal}. Hence, these blue nodes will never satisfy the condition of the while loop and be placed in the priority queue forever. Similar situations apply for the red node as well.

The black node's parent node must be a red or black node. It is useless for finding an optimal path, so it won't be placed in the priority queue and rejected in the second condition of the *QueueUpdate* function.

Every time when RRT$^{\#}$ extends a new node, if its parent node has finite value (green or blue node), it will be temporarily marked as a red node. A red node may become green node immediately in the current iteration and removed from the priority queue. Its successors must be a black node or none.

A red and blue node may become green node if they found themselves have smaller key compared to current best estimate towards the goal. Green node will be removed from queue in the current iteration and it will never become red or black node. Same applys for the blue node.

Finally a green node may become a blue node if the new node added in the tree and produced a shorter path towards the goal. These green nodes must be the successors of the new node and they are affected due to the propagation of *ReduceInconsistency* function.

RRT$^{\#}$ can be further modified into three other variants. The first variant directly ignore the black nodes and they will not be added into the tree. The second variant only allows those nodes to be added into the tree iff their parent nodes are promising nodes. This variant greatly accelerates the speed due to the large reductions of inconsistent red node. The last one is the most selective variant and these nodes itself must be promising otherwise it will not be included in the tree. Hence the red node is completely removed and only blue and green nodes exist, which means every node has a finite g-value. The comparisons among RRT$^{\#}$ and its variants can be seen in the paper [37].

2.4 Informed RRT*

With the exception of sampling from the hyperellipsoid, the informed RRT* algorithm is essentially identical to the basic RRT* algorithm [24]. Thus, apart from this variation, the overall procedure remains the same for both algorithms. The main process of informed RRT* is shown in Algorithm 8. In general, this method samples a new node within a unit circle or a unit ball and transform the shape of the circle or the ball into an ellipse or ellipsoid using Kabsch algorithm [38] and switched to another position, as illustrated in Fig. 2.24.

Algorithm 8 Informed RRT*

```
Require: $V \leftarrow \{x_{\text{inital}}\}\,; E \leftarrow \emptyset$;
Ensure: $\mathcal{T} = (V, E); X_{sol}$;
  while iteration $i \leq N$ do
    if $X_{sol} \neq \emptyset$ then
      $a_{\min} \leftarrow minCost\,(\boldsymbol{x} \mid \boldsymbol{x} \in \boldsymbol{X}_{\text{sol}})\,;$
      $\boldsymbol{x}_{\text{new}} \leftarrow InformedSample\,(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}_{\text{goal}}, a_{\min})\,;$
      $T \leftarrow Rewire\,(T, \boldsymbol{x}_{\text{new}})\,;$
    else
      $\boldsymbol{x}_{\text{new}} \leftarrow RandomSample()$;
      $T \leftarrow Rewire\,(T, \boldsymbol{x}_{\text{new}})\,;$
    if $\boldsymbol{x}_{\text{new}}$ reaches goal then
      $\boldsymbol{x}_{\text{sol}} \leftarrow \boldsymbol{x}_{\text{sol}} \cup \{\text{x}_{\text{new}}\}$
```

The method is adapted from [39]. Assume $\mu \in \mathbb{R}^n$ be the center point. For a fixed constant $d = 1$, points that are within the distance d from μ form a hyperellipsoid in $\mathbb{R}^n$. Equivalently, we can define this hyperellipsoid as

$$\left\{x \in \mathbb{R}^n \mid (x-\mu)^T S^{-1}(x-\mu) < d^2\right\},$$

where S is a $n \times n$ positive definite matrix. Recall that the following formula is called the Mahalanobis distance:

$$d(x, y) = \sqrt{(x-y)^T S^{-1}(x-y)}.$$

Now we can apply Cholesky decomposition for S^{-1}:

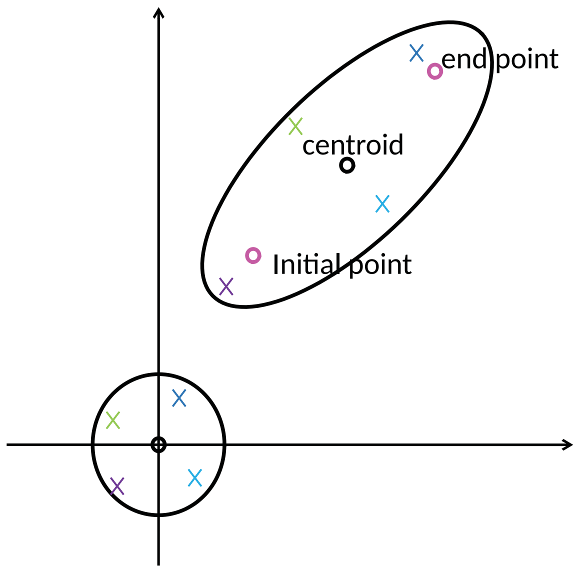


Figure 2.24: The Direct Sampling Method of Informed RRT*

$$(x-\mu)^T S^{-1}(x-\mu) < d^2 \iff (x-\mu)^T \left(LL^T\right)(x-\mu) < d^2$$
$$\iff \left(L^T(x-\mu)\right)^T \left(L^T(x-\mu)\right) < d^2.$$

Then the uniformly sampled points within the hyperellipsoid centered around μ can be generated from uniformly generated points within a unit n-dimensional ball based on the equation:

$$\mathbf{x}_{\text{ellipse}} = \mathbf{L}\mathbf{x}_{\text{ball}} + \mathbf{x}_{\text{centre}}.$$

The next step is to transform the points within the hyperellipsoid body coordinates to the global coordinates. The optimal rotation for these series of points between two different frames can be described as a general Wahba problem [40] formulated as:

Given two sets of k points $\{\boldsymbol{p}_1, \boldsymbol{p}_2, \cdots, \boldsymbol{p}_k\}$ and $\{\boldsymbol{p}'_1, \boldsymbol{p}'_2, \cdots, \boldsymbol{p}'_k\}$, where $k \geq 2$, find the rotation matrix M that yields the optimal least squares coincidence between the first and second sets. The goal is to minimize the following formula:

$$\sum_{i=1}^{k} \left\| \boldsymbol{p}'_i - M\boldsymbol{p}_i \right\|^2.$$

Several solutions have been proposed in the past five decades [41]. One of the solutions such as Kabsch algorithm [42]. The core part of this algorithm uses singular value decomposition (SVD) to minimize the root mean squared deviation (RMSD) between two paired sets of points. The step-by-step procedure is as follows:

- Assume the matrix P and matrix Q represent two sets of points, respectively.
- Subtract each element using the average of the whole column for both Matrices.
- Multiply the transpose of Q by P and get a new matrix H.
- The weighted cross-covariance matrix $H = U\Lambda V^T$.
- The rotational matrix R is calculated by the multiplication of V and U^T.
- Finally, the new points are simply multiplied by the original points with the matrix R.

Note that if determinant$(V, U) < 0$, scale the U^T by an I^n whose last column's entry 1 is swapped to -1, $\Lambda = \mathrm{diag}(1, \ldots, 1, \det(U)\det(V)$, $\det(\cdot)$ is the determinant and both U and V are unitary matrices.

In [43], the body coordinate system is undetermined in the conjugate directions before sampling, so the matrix H can be simplified as $\mathbf{h_1 1_1}^T$, where $\mathbf{h_1}$ is the major axis in the global coordinate, which is $Distance(x_{start}, x_{goal})$ bounded by L^2 norm and $\mathbf{1_1}$ is simply the first column of the identity matrix.

Hence, a state $\mathbf{x}_{\text{sample}}$ which is uniformly sampled in the informed subset can be computed as:

$$\mathbf{x}_{\text{sample}} = HL\mathbf{x}_{\text{ball}} + \mathbf{x}_{\text{centre}}.$$

Algorithm 9 presents a step-by-step procedure for carrying out this process, the parameter a is the current minimum cost of the optimal path.

Fig. 2.25 to Fig. 2.28 below demonstrate a Python simulation of the informed RRT* algorithm. The entire process runs for a maximum of 2000 iterations, starting from the point $(0m, 0m)$ and aiming to reach the goal point $(200m, 220m)$. The sampling area spans $(-270m, 270m)$, with an extended edge of $7m$ and a rewiring radius of $40m$. Additionally, 10% bias towards the goal point is used in the sampling process [44].

Algorithm 9 InformedSample

```
Require: $\boldsymbol{x}_{\text{start}}; \boldsymbol{x}_{\text{goal}}; a$
Ensure: $\boldsymbol{x}_{\text{sample}}$
    $c \leftarrow Distance(\boldsymbol{x}_{\text{start}}, \boldsymbol{x}_{\text{goal}}),\ \boldsymbol{x}_{\text{centre}} \leftarrow \frac{1}{2}(\boldsymbol{x}_{\text{start}} + \boldsymbol{x}_{\text{goal}});$
    $h_1 \leftarrow (\boldsymbol{x}_{\text{goal}} - \boldsymbol{x}_{\text{start}})/c;$
    $\mathbf{U}, \mathbf{V} \leftarrow \text{SVD}(h_1 1_1^T);$
    $\boldsymbol{\Lambda} \leftarrow \text{diag}(1, \ldots, 1, \det(\mathbf{U})\det(\mathbf{V}));$
    $\mathbf{H} \leftarrow \mathbf{U}\boldsymbol{\Lambda}\mathbf{V};$
    $r_1 \leftarrow \frac{1}{2}a;$
    $\{r_j\}_{j=2,\ldots,n} \leftarrow \frac{1}{2}\left(\sqrt{a^2 - c^2}\right);$
    $\mathbf{L} \leftarrow \text{diag}\,(r_1, r_2, \ldots, r_n);$
    $\boldsymbol{x}_{\text{ball}} \leftarrow \text{SampleUnitBall}(n);$
    $\boldsymbol{x}_{\text{sample}} \leftarrow \mathbf{HL}\boldsymbol{x}_{\text{ball}} + \boldsymbol{x}_{\text{centre}};$
```

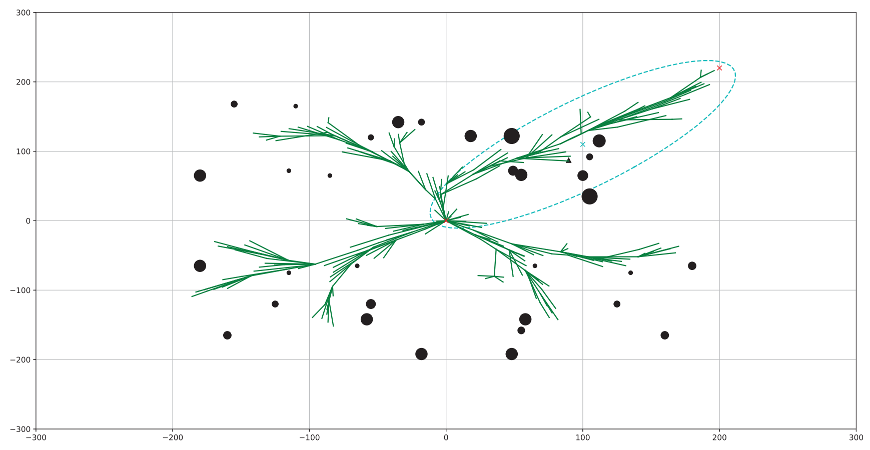


Figure 2.25: Generation of an Initial Path and Form an Ellipse

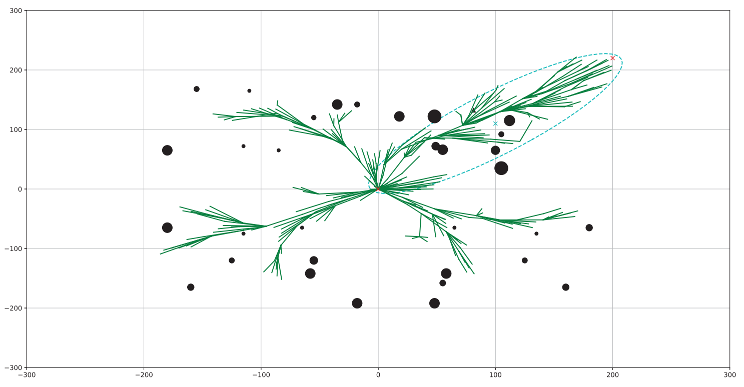


Figure 2.26: Final Path Length Reduced and a Smaller Ellipse

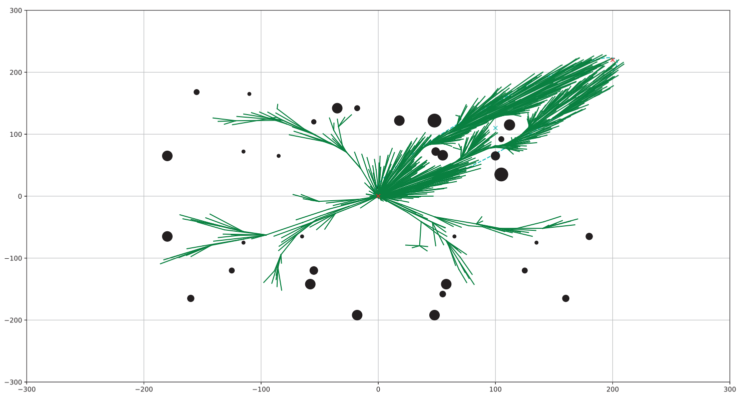


Figure 2.27: Tree Edges Optimization and a Smaller Ellipse

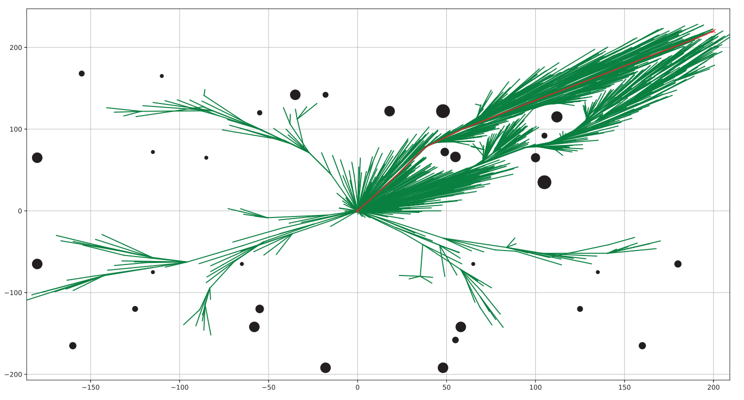


Figure 2.28: The Optimal Path After 2000 Iterations

2.5 Spline RRT and Node Pruning

Former approaches produce geometric routes that are composed solely of straight lines. However, because of the breach of non-holonomic constraints, many mobile robots cannot follow this course with precision. Recent research has proposed several new methods that consider the non-holonomic constraints of the robots and generate paths that are feasible for them to follow accurately. Spline RRT (SRRT) is one of the successful methods [45]. Spline curves are considered superior to Dubins and Reeds-Shepp arcs due to the discontinuities present at the line junctions in the latter. These discontinuities can lead to significant tracking errors and may even cause wheel slippage, which can deteriorate the robot's dead reckoning capability [31].

The SRRT algorithm uses cubic Bézier splines as a local planner to generate smooth and feasible paths that satisfy both external and internal constraints. There are three important kinematic constraints that are considered. First, the feasibility constraint defines an allowable region bordered by the maximum turning angles a_{max} and a_{min}. Second and last, it guarantees G^2 continuity of curvature along the path satisfying a specific upper-bounded curvature constraints ($\kappa_{dir} \leq \kappa_{\max}$). The term "curvature" finds application across various disciplines, and in the context of motion planning, it assumes a mathematical role in quantifying the ratio of angle change of a tangent as it traverses a specific arc. Essentially, curvature serves as a measure of the sharpness of a turn. The formula and the image of the curvature are shown in 2.29, where T is a point in any position of the curve, O is the center of the circle and R is the radius:

$$\kappa_{max}(t) = \frac{1}{R_{min}}, \quad \kappa(t) = \frac{|\dot{g}(t) \times \ddot{g}(t)|}{|\dot{g}(t)|^3}. \tag{2.5.1}$$

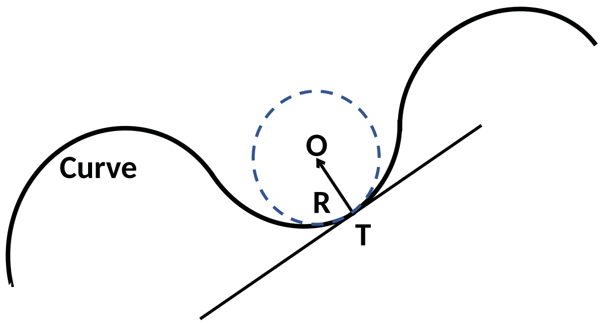


Figure 2.29: Definition of Curvature

Among the three candidate points in Fig. 2.30, only points $P2$ and $P3$ falling within this region can be accepted as feasible points in the SRRT. After checking this precondition, we can then choose $P3$ as it is the nearest neighbour of the P_{start}.

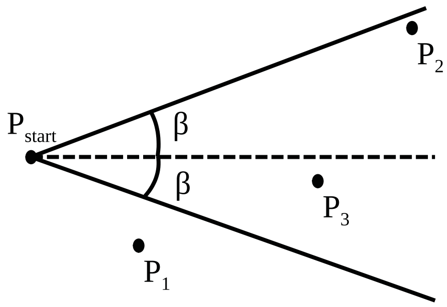


Figure 2.30: Sampled Points Feasibility within Allowable Angle

The paper [46] proposes an analytical algorithm that utilizes parametric cubic Bézier curves to create a smooth path with G^2 continuous curvature, requiring only the specification of a maximum curvature constraint. The cubic Bézier curve is defined as:

$$P(s) = \sum_{i=0}^{3} P_i B_{3,i}(t), \quad B_{3,i}(t) = \begin{pmatrix} 3 \\ i \end{pmatrix} t^i (1-t)^{3-i}. \tag{2.5.2}$$

where t is a variable such that $0 \leq t \leq 1, P_i$ are control points, and $B_{3,i}(t)$ are Bernstein polynomials.

A rigorous proof and definition among G^0, G^1 and G^2 can refer to [47]. Geometric continuity, often denoted by G^k, is a measure of the smoothness between two adjacent curves. G^0 continuity, also known as positional continuity, only requires that two curves share a common point. G^1 continuity, or tangency continuity, goes a step further by requiring that the tangent vectors of the two curves at their common point lie along the same direction. G^2 continuity, or curvature continuity, requires that in addition to the tangent continuity, the two curves also have the same curvature at their common point.

The definition of G^0, G^1 and G^2 are illustrated as follows:

$$\begin{aligned} \mathbf{p}_i(s_i) &= \mathbf{p}_{i+1}(s_{i+1}) \, \forall i \in [1, n-1], \\ \chi_i(s_i) &= \chi_{i+1}(s_{i+1}) \, \forall i \in [1, n-1], \\ \kappa_i(s_i) &= \kappa_{i+1}(s_{i+1}) \, \forall i \in [1, n-1]. \end{aligned} \tag{2.5.3}$$

where $\mathbf{p}$ is the function of position, χ is the tangent angle and κ is the curvature. i denotes the transition points between two connected curves.

In a two-dimensional space, Dubins curve has been proven to be G^1 continuous [48]. This means that the tangent vectors of adjacent curves are aligned, ensuring a continuous change in direction. On the other hand, splines are capable of maintaining G^2 continuity [46], which means that the adjacent curves not only have aligned tangent vectors but also the same curvature at their common point, resulting in a smooth and continuous change in curvature as well.

The maximum curvature κ can be defined based on the kinematic constraints of the vehicle, so the maximum turning angles $\beta = \frac{\gamma}{2}$ is the only variable to be determined. When both variables are known, we can specify the extended edge of the SRRT. The extended edge d and eight control points B_{0-3} and E_{0-3} among P_{1-3} as well as the coefficients c_{1-4} are defined as:

$$c_1 = 7.2364, \quad c_2 = \frac{2}{5}(\sqrt{6}-1), \quad c_3 = \frac{c_2+4}{c_1+6}, \quad c_4 = \frac{(c_2+4)^2}{54c_3}. \tag{2.5.4}$$

$$h_b = h_e = c_3 d, \quad g_b = g_e = c_2 c_3 d, \quad k_b = k_e = \frac{6c_3\cos\beta}{c_2+4} d. \tag{2.5.5}$$

$$d = \frac{c_4 \cdot \sin\beta}{\kappa_{\max} \cdot \cos^2\beta}. \tag{2.5.6}$$

$$\begin{aligned} B_0 &= P_2 + d\cdot u_1, & B_1 &= B_0 - g_b \cdot u_1, \\ B_2 &= B_1 - h_b \cdot u_1, & B_3 &= B_2 + k_b \cdot u_d, \\ E_0 &= P_2 + d\cdot u_2, & E_1 &= E_0 - g_e \cdot u_2, \\ E_2 &= E_1 - h_e \cdot u_2, & E_3 &= E_2 - k_e \cdot u_d. \end{aligned} \tag{2.5.7}$$

where h, g, k can also be determined by some of the coefficients of c_{1-4} and β.

Fig. 2.31 presents an intuitive description of a closed-form algorithm that efficiently computes eight control points and generates a curve based on the equations mentioned earlier.

The research paper [45] provides the pseudo-code for SRRT and a description of their node pruning strategies. Here we demonstrated our implementation for node pruning method in Algorithm 10.

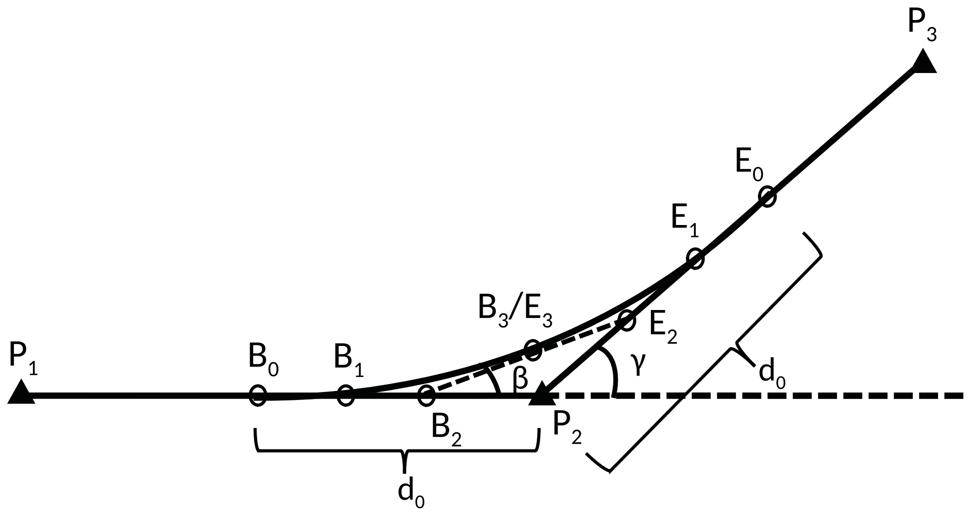
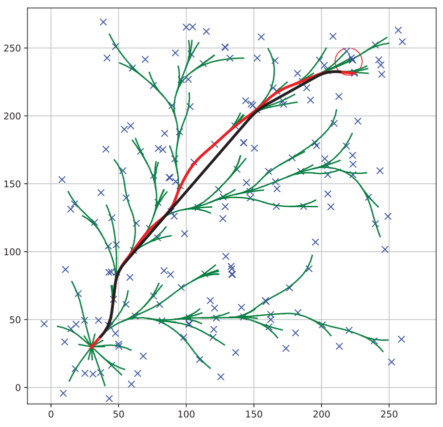


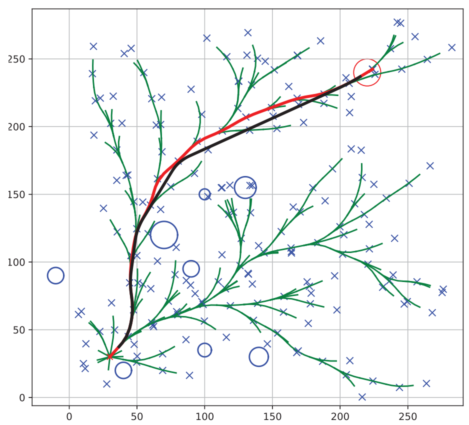
Figure 2.31: Cubic Bézier Curve Defined by Eight Control Points

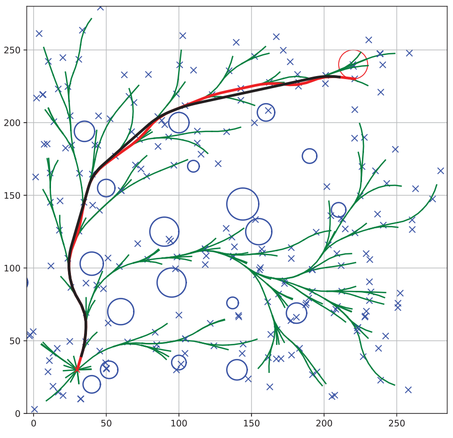
Algorithm 10 Node Pruning of SRRT

```
Require: $\boldsymbol{path}_{\text{old}}; \boldsymbol{X}_{\text{obst}};$
Ensure: $\boldsymbol{path}_{\text{new}}$
  $\boldsymbol{pt}_{\text{frwd}} = \boldsymbol{pt}_{\text{curr}} = \boldsymbol{path}_{\text{old}}(goal - 6);$
  while $\boldsymbol{pt}_{\text{curr}} \geq 0$ do
      $\boldsymbol{angle}_1 \leftarrow \text{CalcAngle}(\boldsymbol{path}_{\text{old}}([frwd+3],[frwd+1],[frwd]);$
      $\boldsymbol{angle}_2 \leftarrow \text{CalcAngle}(\boldsymbol{path}_{\text{old}}([frwd+4],[frwd+3],[frwd+1]);$
      if $\boldsymbol{angle}_{1,2} < \boldsymbol{angle}_{steerMax}$ then
          $\text{CollisionCheckingSpline}(\boldsymbol{path}_{\text{old}}([\text{frwd}],[\text{frwd}+1],[\text{frwd}+3]);$
          $\text{CollisionCheckingSpline}(\boldsymbol{path}_{\text{old}}([\text{frwd}+1],[\text{frwd}+3],[\text{frwd}+4]);$
          $\text{CollisionCheckingLine}(\boldsymbol{path}_{\text{old}}([\text{frwd}+1],[\text{frwd}+3]);$
          if NoCollision then
              Remove old edges and $\boldsymbol{pt}_{\text{frwd+2}}$ and create new edges in $\boldsymbol{path}_{\text{new}}$.
      $\boldsymbol{pt}_{\text{frwd}} - 1;$
      if $\boldsymbol{pt}_{\text{frwd}} < 0$ then
          $\boldsymbol{pt}_{\text{curr}} - 1; \boldsymbol{pt}_{\text{frwd}} = \boldsymbol{pt}_{\text{curr}};$
```

Our implementation of SRRT with path smoothing in Python was visualized in Figs. 2.32a to 2.32d, excluding the use of concentration-based sampling strategies suggested from [45]. To evaluate the algorithm's performance, we tested it in four distinct environments with varying obstacle densities, ranging from sparsely to densely cluttered. The maximum number of iterations was established differently for each test. For tests (a) and (b), it is set to 200. However, for test (c), it is increased to 300, and for test (d), the limit is further raised to 500. The robot's original point is at $(0m, 0m)$, and the destination is located at $(220m, 240m)$ with radius equals to $10m$. The sampling area was defined as $x = [0m, 280m]$, $y = [0m, 280m]$, and we assume the robot is circular shaped and its radius is $5m$.


(a) SRRT: No Obstacles


(b) SRRT: Few Obstacles


(c) SRRT: Moderate Obstacles

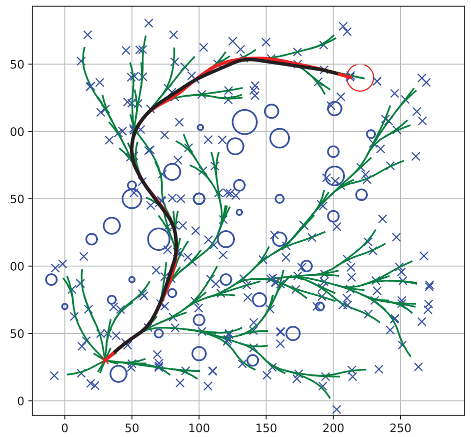

(d) SRRT: Clustered Obstacles

Figure 2.32: The Simulation of SRRT Under Varying Conditions

The above four images are the visual representation of the SRRT algorithm's performance under varying conditions: from environments free of obstacles, through those with few and moderate obstacles, to densely clustered obstacle scenarios. These illustrations demonstrate the limitation of this method in consistently providing optimal paths, especially in dense obstacle scenarios.

Chapter 3

Proposed Optimal Sampling Methods for SRRT

Sampling-based non-holonomic motion planners generally exhibit faster convergence rates compared to alternative techniques. However, many non-holonomic motion planning algorithms commonly face the issue of slow convergence and cannot directly apply optimal method using Euclidean distance metric. In this chapter, we present a novel algorithm called Informed SRRT$^{\#}$ that effectively addresses both external constraints arising from obstacles and internal kinematic constraints of robots. Moreover, it maintains a reasonable convergence rate, allowing it to discover shorter and smoother paths. In contrast to traditional RRT* algorithms that employ Euclidean metrics, our approach enhances the methodology by integrating a local planner derived from SRRT. To determine the path towards the goal region, we employ parameterized cubic curves instead of computationally intensive numerical techniques.

3.1 Problem Formulation

The motion planning problem is described in a similar manner as documented in the works of [22] and [24]. Let $\mathcal{X} \subseteq \mathbb{R}^n$ denote the configuration state space, where $n \subseteq \mathbb{N}$ with $n \geq 2$. The set $\mathcal{X}_{obs}$ belongs to $\mathcal{X}$, it consists of states that collide with obstacles, while $\mathcal{X}_{free}$ represents the non-collision permissible state space, obtained by taking the closure (denoted by $cl(\cdot)$) of the set in $\mathcal{X}$ and excludes $\mathcal{X}_{obs}$. $\mathbf{x}_{start}$ and $\mathbf{X}_{goal}$ are the known initial point and goal region, respectively, both belonging to $\mathcal{X}_{free}$.

Assume $\mathcal{T} = (V, E)$, where V and E are finite sets of vertices and edges, respectively. A continuous path for the nonholonomic robot is represented by the sequence of states $\sigma : [0, 1] \rightarrow \mathcal{X}_{free}$, with $\sigma(0) = \mathbf{x}_{start}$ and $\sigma(1) \in \mathbf{X}_{goal}$. For a vertex x in the graph $\mathcal{T}$, pred$(\mathcal{T}, x)$ or x_{parent} refers to the parent vertex in V that can reach Node x, while succ$(\mathcal{T}, x)$ or x_{succ} denotes a child vertex that can be reached from Node x.

Within the SRRT algorithm proposed by [45], the generation of a new extended segment towards a feasible point involves an analytical algorithm that always adheres to the upper bound curvature constraints. Fig. 2.30 depicts an additional constraint that represents a region bounded by the maximum turning angles of the robot. The middle portion of each extended segment is defined using a Bézier spline. As outlined in [45] and Section 2.5, eight control points (B_{0-3} and E_{0-3}) are employed to generate a continuous curvature among three points (W_{1-3}), as illustrated in Fig. 2.31.

The two key variables, the turning angle β and the length d_0 are set as

$$\beta = \frac{\alpha}{2}, \tag{3.1.1}$$

$$d_0 = \frac{c_4 \cdot sin\beta}{\kappa_{max} \cdot \cos^2 \beta}, \tag{3.1.2}$$

where κ_{max} is the maximum curvature, c_4 will be calculated by c_{1-3}, γ is the angle between the vector $\overrightarrow{P_1P_2}$ and $\overrightarrow{P_2P_3}$, the formula and values for c_{1-3} is in Eqns 2.5.4 in Section 2.5.

In our proposed algorithm for 2D cases, apart from the starting point and goal point, the extension length ε for each segment will be set to at least twice the length d_0. This ensures that the robot's allowable maximum angle β and the upper bound curvature constraints κ_{max} are satisfied. Similar to RRT*, a circle is utilized to encompass neighboring nodes and establish potential new connections. This circle-based approach aims to reduce the overall cost by effectively including nearby nodes and facilitating the creation of connections. The radius of the circle can be easily determined by setting it to a slightly longer length than the minimum extended edge of the tree (L_{min} or $2d_0$). This circle will encompass potential neighbors (x_{nbr}) within its range, excluding the current parent node of the new node (x_{new_p}).

The objective of the proposed path planning algorithm is to efficiently discover a shorter path, denoted as $\mathcal{X}_{sol}$, within a reasonable time frame. This algorithm consistently ensures the feasibility of the differential constraints, including maximum angles (α), curvatures (κ), and their G^2 continuity, for each segment of the path. All sets comprising this path are constrained, continuously connected, and exclusively belong to $\mathcal{X}_{free}$.

3.2 Informed SRRT$^{\#}$

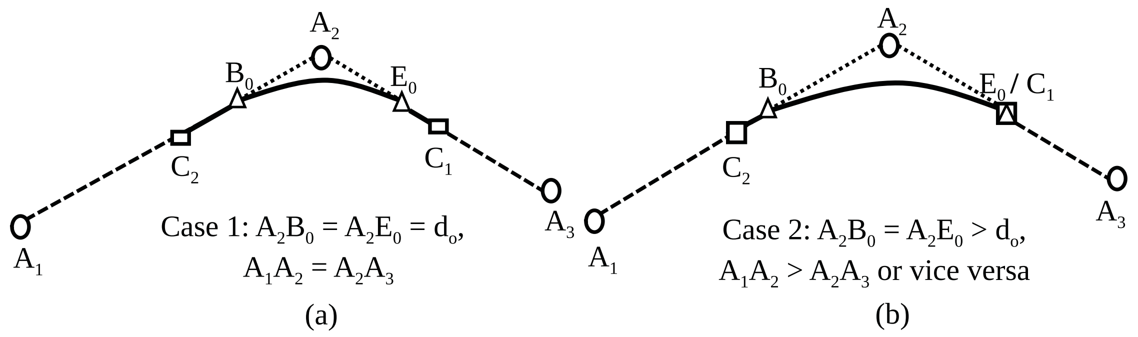
In our proposed algorithm, the first step is to perform a feasibility check on the kinematic constraints for a new node or a rewired neighbor node. To guarantee G^2 continuous curvature, it is essential to ensure that B_0X_2 is equal to E_0X_2. The constraint of the maximum turning angles should be also satisfied. Additionally, we enforce the constraint that the extended edge L must be greater than or equal to L_{min}, which guarantees $\kappa_{dir} \leq \kappa_{max}$. Finally, this method ensures that the edges avoid obstacles, even though the sampled nodes may be located within the obstacles. The last three constraints can be checked within the function FeasiblePath$(x1, x2, x3)$.

Our algorithm incorporates several enhancements to improve its effectiveness. Firstly, we introduce additional kinematic constraints to the optimization functions used in RRT*, ensuring proper connection of points using Bézier curves. Furthermore, we integrate the sampling method employed by Informed RRT* in 2D environment, which directly samples nodes within a circle. During the rewiring process, our algorithm discards or reconnects sub-trees when their nodes fail the kinematic feasibility check. We may keep track of and utilize the previous path with the current lowest cost as the semi-major axis of the ellipse, even some of the paths are deleted. This axis defines an elliptical region where we can directly sample new potentially promising points. The recorded lowest cost is of utmost importance as it continues to accelerate the search speed and enables the discovery of shorter paths. Lastly, we have incorporated the concept from RRT$^{\#}$ of applying heuristic functions to rewire only promising nodes. The function $LowerCost$ has two conditions. Firstly, the total distance of $c_2 + Dist(x_1, x_2) + Est(x_1, x_{goal})$ must be smaller than c_{best}. Secondly, the sum of $c_{new} + Dist(x_1, x_2)$ should not be larger than the cost c_{nbr}.

To achieve G^2 continuity in a piece-wise Bézier curve using Yang's method [46], we divide the connections between nodes A_1, A_2, and A_3 into three segments, as depicted in Fig. 3.1. While this local spline planner differs slightly from [45] in solving the two-point boundary problem, it shares a similar concept with the half distance algorithm described in [31].

The first segment (B_0C_2) and the third segment (C_1E_0) are straight lines, or one or both of them may not exist. To create the Bézier curve for the second segment (B_0E_0), it is crucial to ensure that the lengths of A_2B_0 and A_2E_0, which determine the positions of eight control points, are always equal. This condition guarantees G^2 continuity as well mentioned above. Also, it should be maintained regardless of whether the curve is generated using the steering function or modified using any two of the rewire functions.

Furthermore, the lengths of A_2B_0 and A_2E_0 can be equal to or greater than the minimum length d_0. When they are equal, the curve may have a higher curvature, similar to Case 1 shown in Fig. 3.1. Note that both B_0C_2 and E_0C_1 could be zero in this scenario. Alternatively, if the extended edges slightly exceed d_0, we can utilize the remaining straight line segments, either B_0C_2 or C_1E_0, to create a Bézier curve with lower curvature, as illustrated in Case 2 of Fig. 3.1. In this case, only one of B_0C_2 or C_1E_0 is set to zero.



A_2
B_0
E_0
C_2
C_1
A_1
A_3
Case 1: $A_2B_0 = A_2E_0 = d_o$, $A_1A_2 = A_2A_3$
(a)
A_2
B_0
E_0 / C_1
C_2
A_1
A_3
Case 2: $A_2B_0 = A_2E_0 > d_o$, $A_1A_2 > A_2A_3$ or vice versa
(b)

Figure 3.1: Link a New Node with a Existing Tree Nodes.

Theorem 1 If $A_2B_0 = A_2E_0 \geq d_0$, then there exists a G^2 continuous curvature curve that between A_2C_1 and A_2C_2. The piece-wise connections of all edges of a tree are G^2 continuous.

Proof: The proof for generating a continuous curvature of the Bézier curve refers to [46]. Both B_0C_2 and C_1E_0 are straight lines ($\kappa = 0$) or shrink to one point and they have the same directions with either side from A_2 to the boundary points of the curve. ∎

Our algorithm is based on the original Informed-RRT* and incorporates three fundamental functions, as described in Algorithm 11. The initial function, Algorithm 12 (*Extend*), establishes a connection between the nearest node, $x_{nearest}$, and the new point, x_{new}. The second function, Algorithm 13 (*SelectforNew*), selects a new parent node from the tree for the recently added node. Finally, the third function ($SelectforNeighbours$) determines whether a new node can act as the parent node for its neighboring nodes. We propose three different approaches for this function. Method 1 involves trimming the leaf nodes, which is exclusively employed prior to finding a feasible path towards to goal region. Method 2 cuts the previous subtrees if the new connections break the kinematic constraints to connect these nodes in the subtrees. Method 3 draws inspiration from [49], the nodes in these subtrees may be reused as "new" sampled nodes.

Algorithm 11 Cubic Bézier Spline Informed RRT$^{\#}$

```
Require: $V \leftarrow \{x_{start}\}$, $E \leftarrow \emptyset$, $X_{sol} \leftarrow \emptyset$
Ensure: $\mathcal{T} = (V, E)$, $X_{sol}$
  for Iteration$= 1, 2, \ldots, N$ do
      $c_{best} \leftarrow min_{x_{sol} \in X_{sol}}\{c_{sol}\}$
      if $q$ is not empty then
          $x_{rand} \leftarrow q.pop()$
      else
          $x_{rand} \leftarrow Informed\_Sample(x_{start}, x_{goal}, c_{best})$
      $x_{nearest} \leftarrow (\mathcal{T}, x_{rand})$;
      $x_{new} \leftarrow Extend(\mathcal{T}, x_{nearest}, x_{rand})$
      $X_{nbr} \leftarrow Neighbours(\mathcal{T}, x_{new}, r)$
      $SelectforNew(\mathcal{T}, X_{nbr}, x_{new})$
      if $x_{new} \subsetneq V$ then
          $SelectforNeighbours(\mathcal{T}, X_{nbr}, x_{new}, pq)$
          $ReduceInconsistency(pq)$
          if $x_{new} \in X_{goal}$ then
              $X_{Sol} \leftarrow X_{Sol} \cup x_{new}$
```

Later, we incorporated the optimal strategies from the third variations of RRT$^{\#}$. However, our algorithm structure does not strictly adhere to the same approach as the original paper. Instead, we directly integrated the heuristic distance $Est(x_{new}, x_{goal})$ into either the $SelectforNew$ or the $SelectforNeighbours$ optimization function to identify promising new nodes and connect them with the tree. Additionally, in the

$SelectforNeighbours$ function, we included neighbor nodes that select the new node as their parent node into a priority queue. Finally, the *ReduceInconsistency* function simply requires removing all the promising nodes, establishing new connections, and maintaining a consistent spanning tree.

The initial operation called "Extend" aims to expand the edges of the tree during each iteration. The nodes directly extended from the starting node, denoted as x_{new}, are treated as a special case. In this function, the first step is to extend the edge from the original point x_{start} to the midpoint M_0. This midpoint lies between the starting point x_{start} and the new point x_{new}, and the edge connecting them is a straight line. Alternatively, if we are extending a different node, we use a Bézier curve to connect the parent node of the nearest neighbor, referred to as x_{nbr_p}, the nearest neighbor node x_{nbr}, and a randomly sampled point x_{rand}. We ensure that the angle of change $\beta(x_{nbr_p}, x_{nbr}, x_{new})$ does not exceed a predetermined maximum value β. To achieve this, we calculate the direction from x_{nbr} to x_{rand} and from x_{nbr_p} to x_{nbr} using the function $Dir(x_1, x_2)$. If the resulting Bézier curve does not intersect with any obstacles present in the environment, it is included in the tree. The specific details of the *Extend* function are outlined in Algorithm 12.

Algorithm 12 $Extend(\mathcal{T}, x_{nbr}, x_{rand})$

```
$x_{new} \leftarrow x_{nbr} + Dir(x_{nbr}, x_{rand}) * L_{min}$
**if** $x_{new\_p} \leftarrow x_{start}$ **then**
    **if** $CollisionFreeLine(x_{start}, x_{new})$ **then**
        $E \leftarrow E \cup \{x_{start}, M_0\}$ with line
        $V \leftarrow V \cup \{x_{new}\}$
**else**
    **if** $\beta(x_{new}, x_{nbr}, x_{nbr\_p}) < \beta$ **then**
        **if** $NoCollisionBézier(x_{nbr\_p}, x_{nbr}, x_{new})$ **then**
            $E \leftarrow E \cup \{M_1 M_2\}$ with Bézier curve
            $V \leftarrow V \cup \{x_{new}\}$
```

The first step of the rewiring process, known as the "SelectforNew" function, as presented in Algorithm 13, involves multiple constraints that must be satisfied before a new node can be incorporated into the tree. Firstly, the new node, the neighboring node, and its parent node must adhere to the maximum turning angle restriction, denoted as $\beta(x_{new}, x_{nbr}, x_{nbr_p})$. Secondly, the length of the new edge must surpass the minimum required length of $2d_0$ to avoid violating the maximum curvature limitations, κ_{max}. Once these constraints are fulfilled, the sum of the cost to reach the new node, c_{new}, and the cost of the edge, $Dist(x_{new}, x_{nbr})$, must be smaller than the cost-to-come of the neighboring node, c_{nbr}. Furthermore, the combined cost-to-come and estimate-to-go, $Est(x_{new}, x_{goal})$, must be less than the current minimum cost, c_{best}. If all these conditions are satisfied, the new node, x_{new}, can be added to the tree and connected with either a Bézier curve or a straight line, depending on whether the path is collision-free. Thus, this function follows a similar optimization process as RRT* but includes an additional feasibility check.

Algorithm 13 $SelectForNew(\mathcal{T}, X_{nbr}, x_{new})$

```
if $x_{new\_p}$ is $x_{start}$ then return NULL
for $\forall x_{nbr} \in X_{nbr}$ do
    if $x_{nbr}$ is $x_{start}$ then
        if $CollisionFreeLine(x_{start}, x_{new})$ then
            $E \leftarrow E \cup \{x_{start}, M_0\}$ with line
            $V \leftarrow V \cup \{x_{new}\}$
            Update $c_{new}$ and $c_{best}$, break the loop
    else if $LowerCost(x_{new}, x_{nbr})$ and
            $FeasibleBézier(x_{new}, x_{nbr}, x_{nbr\_p})$ then
            $E \leftarrow E \cup \{M_1 M_2\}$ with Bézier curve
            $V \leftarrow V \cup \{x_{new}\}$
    Update $c_{new}$ and $c_{best}$
```

The third function, called "SelectForNeighbours", encompasses three distinct methods aimed at optimizing the RRT tree. The first method involves rejecting a neighbor node, x_{nbr}, if it possesses one or more child nodes, $x_{nbr_{cld}}$, along with all its successors, x_{succ}. This method provides a rapid way to guide the RRT tree towards achieving a sub-optimal path to the final goal region by partially optimizing the leaf nodes.

The second method partially prunes branches that are not feasible with the new constraints, which effectively helps reduce the overall path length. This approach selectively removes portions of the tree that do not conform to the updated criteria. The third method treats these infeasible nodes as “new” sampled nodes and adds them to a queue, denoted as q, for the subsequent iteration.

If a node is not rejected, the function proceeds to rewire two pairs of edges among three states. The first edge connects the middle points, M_1 and M_2, of the neighbor node x_{nbr}, the new node x_{new}, and its parent node x_{new_p}. The second edge links the middle points, M_1 and M_2, of the new node x_{new}, the neighbor node x_{nbr}, and all of its child nodes $x_{nbr_{cld}}$. A visual representation of this is depicted in Figs. 3.2 to 3.5. The details of the $SelectForNeighbours$ function are presented in Algorithm 14.1 (Method 1) and Algorithm 14.2 (Methods 2 and 3). Algorithm 15 describes the process to maintain a consistent spanning tree using RRT$^{\#}$.

Algorithm 14.1 $SelectForNeighbours(\mathcal{T}, X_{nbr}, x_{new}, pq)$

```
for $\forall x_{nbr} \in X_{nbr}$ do
    if $x_{nbr\_cld}$ exists then
        Skip this iteration
    if $LowerCost(x_{nbr}, x_{new})$ and
      $FeasibleBézier(x_{nbr}, x_{new}, x_{new\_p})$ then
            $E \leftarrow E \cup \{M_1 M_2\}$ with Bézier curve
        Add $x_{nbr}$ into $pq$, update $c_{nbr}$ and $c_{best}$
```

Algorithm 15 $ReduceInconsistency(pq)$

```
while $pq$ is not empty do
    $x = pq.pop()$
    $X_{nbr} \leftarrow Neighbours(\mathcal{T}, x, r, pq)$
    $SelectforNeighbours(\mathcal{T}, X_{nbr}, x, pq)$
```

The collision checking strategy implemented in the algorithm takes into account the piece-wise segments of the Bézier curve, as depicted in Fig. 3.6. When a new node, x_{new}, is collision-free, the algorithm tries to establish a connection with the nearest node. If the initial connection attempt fails, the algorithm proceeds to attempt connections with other neighboring nodes. However, if both the *Extend* and

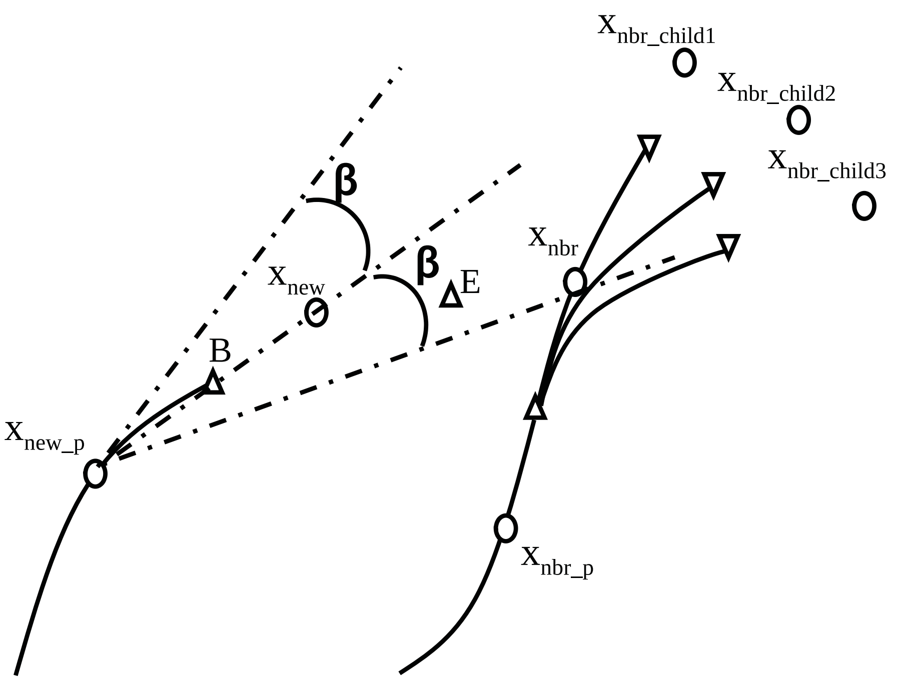


Figure 3.2: Feasibility Check Among $x_{nbr}, x_{new}, x_{new\_p}$

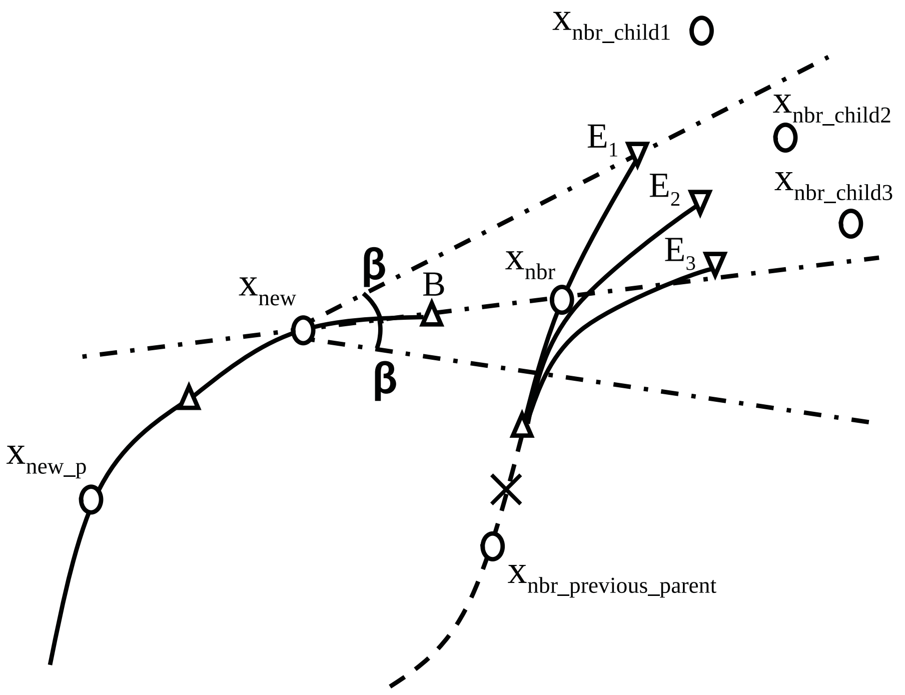


Figure 3.3: Feasibility Check Among $x_{nbr}, x_{new}, x_{nbr\_cld}$

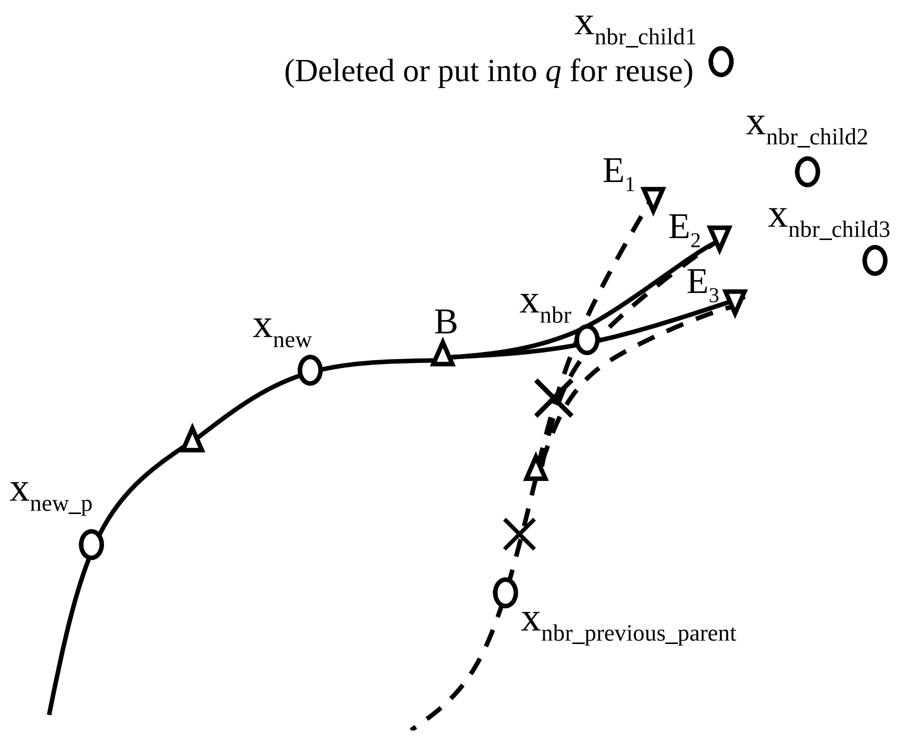


Figure 3.4: Deletion of x_{nbr_child1}

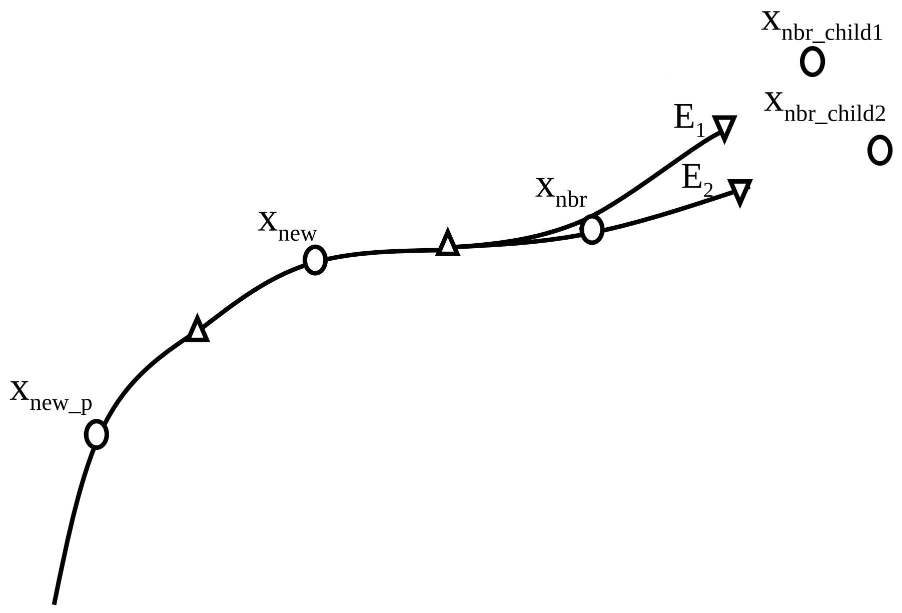


Figure 3.5: New Rewired Connection

Algorithm 14.2 $SelectForNeighbours(\mathcal{T}, X_{nbr}, x_{new}, pq)$

```
for $\forall x_{nbr} \in X_{nbr}$ do
    if $LowerCost(x_{nbr}, x_{new})$ and
     $FeasibleBézier(x_{nbr}, x_{new}, x_{new\_p})$ then
        $E \leftarrow E \cup \{M_1 M_2\}$ with Bézier curve
        Add $x_{nbr}$ into $pq$
        if $x_{nbr\_cld}$ exists then
            if $FeasibleBézier(x_{new}, x_{nbr}, x_{nbr\_cld})$ then
                $E \leftarrow E \cup \{M_1 M_2\}$ with Bézier curve
      else
            Remove $x_{nbr\_cld}$ and all $x_{succ}$ (Method 2)
            Put $x_{nbr\_cld}$ and all $x_{succ}$ into $q$ (Method 3)
        Update $c_{nbr}$, all its $c_{nbr\_cld}$ and $c_{best}$.
```

$SelectForNew$ functions are unable to establish a connection for x_{new}, the sampled node x_{rand} is rejected, and the algorithm starts a new iteration.

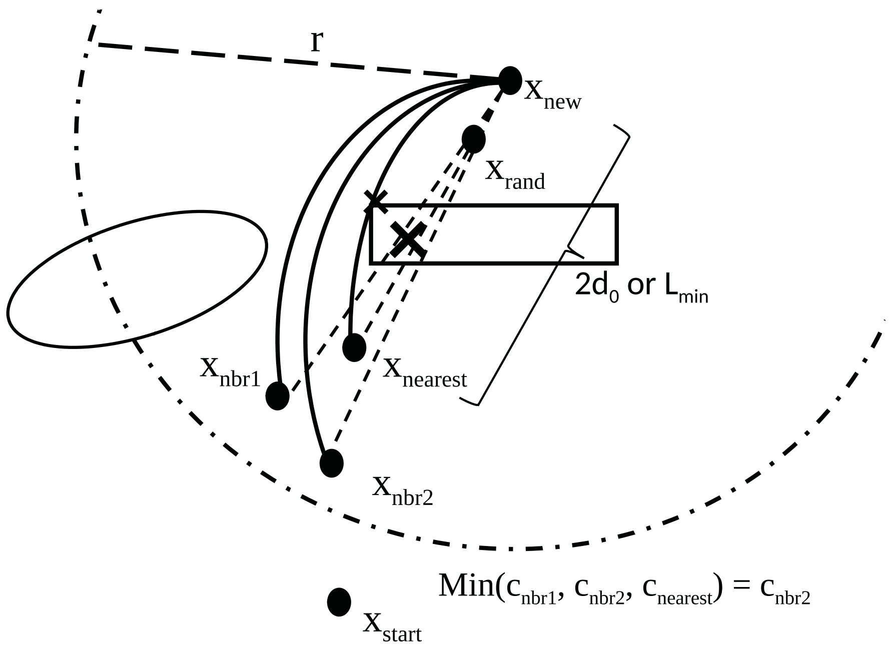


Figure 3.6: Collision Checking for the Initial Connection of x_{new}

There are various collision checking strategies that can be employed in RRT family algorithms. One straightforward approach is to examine the distances between each point that forms an edge and all the obstacles. If none of these points collide with any obstacles, then it is possible to generate a new edge.

However, it is important to note that this collision checking strategy may be computationally expensive since it involves calculating the distances between every point and all the obstacles. The computational complexity escalates with an increase in the number of points and obstacles. To optimize the collision checking process, more efficient algorithms such as spatial partitioning techniques (e.g., bounding volume hierarchies, spatial hashing) or proximity queries (e.g., collision detection algorithms based on bounding boxes or spheres) can be utilized. These techniques help reduce the number of distance calculations required, resulting in improved performance for collision checking in RRT algorithms.

Another crucial aspect of the algorithm is the calculation of the edge length when a new node is added to the tree. One straightforward approach is to utilize straight lines connecting all sampled points that form a Bézier curve as the length for the cost. However, to ensure that the ellipse ranges cover sufficient sampling space, this approach needs to add additional cost. For a more precise calculation of the length of each Bézier curve segment, advanced numerical methods can be employed since there is no closed-form solution for determining the length of cubic or higher-order Bézier curves.

Chapter 4

Simulation

In this chapter, we will demonstrate the simulations for the algorithm proposed in Chapter 3 in both python simulation and Gazebo/Rviz simulation in ROS with USV model.

4.1 Python Simulation

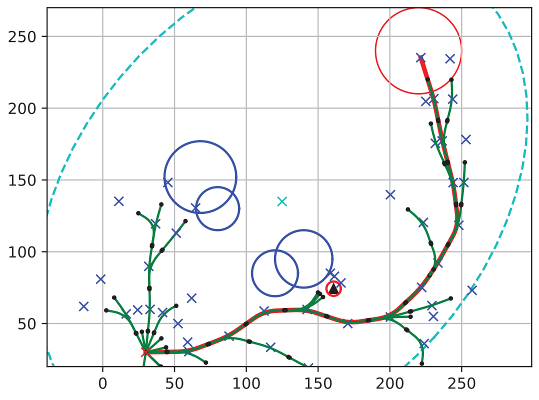
We developed an optimal cubic Bézier spline Informed RRT$^{\#}$ algorithm in Python 3 that considers kinematic constraints. To demonstrate the optimization process of this algorithm, we ran two simulations up to 500 iterations, with an early stop if the result was close to the desired smallest path ($283m$ and $225m$). Both simulations stopped around 400 iterations and final path lengths are $297m$ and $250m$. After applying the path smoothing technique, the path can reduce to $287m$ and $244m$, respectively.

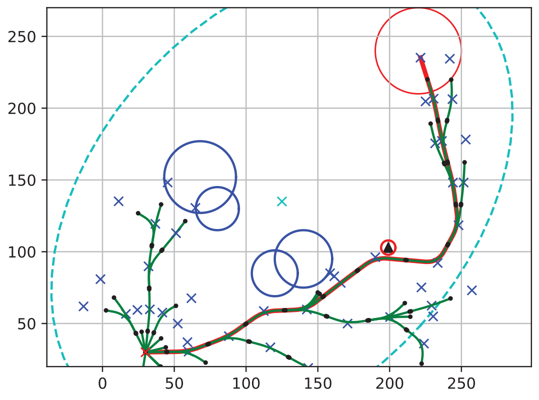
The map measures $280m$ by $280m$ and is limited to the first quadrant ($x, y > 0$). The goal region in simulations is a circle with a radius of $30m$ centered at $(220m, 240m)$. The starting point was set at $(30m, 30m)$. The maximum curvature κ_{max}, angle 2β and the minimum extended length $2d_0$ is set as 0.1, 0.4π and $20.18m$, respectively, as suggested in [45]. For simplicity, the extended length L_{min} is fixed as $30m$ and the radius r in 2D environment is fixed as $45m$. In the diagrams, nodes that have successfully connected to the tree are denoted by blue cross signs, while blue circles signify obstacles. The tree's edges are represented by green curves, and connection points among these edges are indicated by black dots. It's important to note that sampled nodes may or may not intersect with obstacles, as long as the green paths themselves remain free from any collision. Due to the 0.5-unit resolution of the map, some minute gaps between two black dots might appear, and this is acceptable. This is because in the Python simulation, points can only be drawn at intervals of 0.5 units along both the x and y axes. This means points can be placed at locations such as x = 0.5, 1, 1.5, 2.0, and any other multiple of 0.5, with the same rule applying for the y-axis.

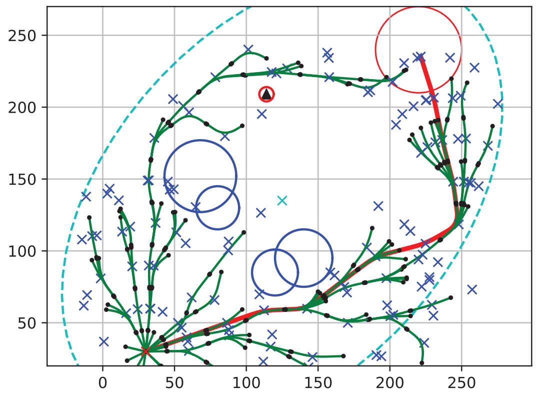
The red path shown in our figures represents the final path, with a cost close to the optimal region $[Est(x_{start}, x_{goal}) \pm D_{goal}]$. We applied the path smoothing technique from Algorithm 10 to generate the black path, as shown in Fig. 4.1f and Fig. 4.2d.

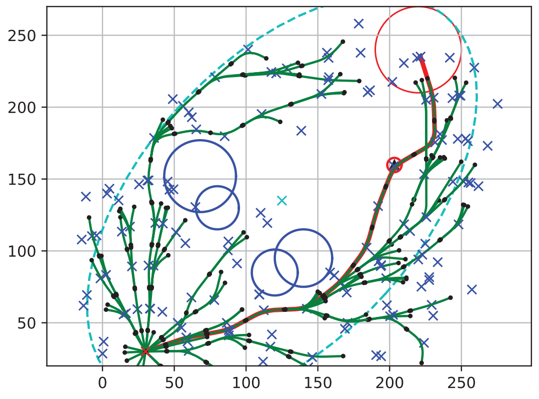
As shown in Figs. 4.1a to 4.1f, Simulation 1 incorporates minor obstacles within a narrow passage. Our method retains the advantages of RRT* — rewiring neighboring nodes in the tree to consistently select a new node as the parent node — albeit with fewer opportunities due to more restricted conditions. In comparison to SRRT, our algorithm leverages numerous pre-existing connections to construct paths with lower costs, thereby significantly reducing the expense of creating feasible paths that adhere to kinematic constraints. Additionally, it introduces several deletion operations. In Fig. 4.1c and Fig. 4.1e, the red curve deviates slightly from the green curve of the tree, indicating the utilization of the previous feasible path and its optimal cost as the long-axis of an ellipse for focused sampling. Lastly, our algorithm can eliminate many unnecessary nodes located outside the designated sampling region, as illustrated from Fig 4.1e to Fig 4.1f.

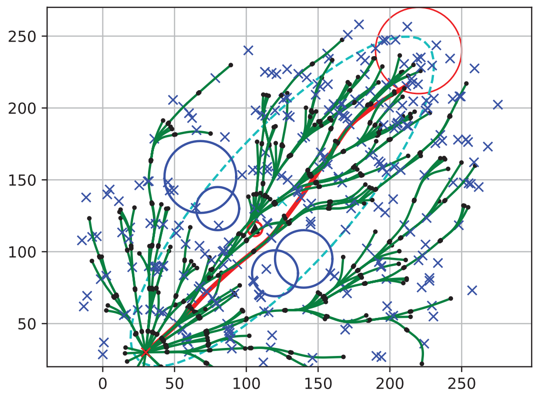
In Simulation 2, the goal region is situated at coordinates $(220m, 150m)$. All other conditions remain unchanged. As depicted in Figs. 4.2a to 4.2d, clustered obstacles are present. The initial path found in Fig. 4.2a took a huge detour, bypassing the majority of obstacles in the central area at a great cost, instead choosing to head towards the destination through peripheral areas with fewer obstacles. However, it later has ability to get across several obstacles and find a shorter path with proper rewiring and deletion process. Although this method can encounter challenges when navigating through narrow spaces in environments populated by clustered obstacles, occasionally causing the extension process to get stuck and consequently prolonging convergence time to decrease the path's cost, it nonetheless offers an improvement in final path length compared to the traditional SRRT algorithm, especially when leveraging the benefits of informed RRT*. This variant of RRT* sustains the value of the optimal path length, equating it to twice the length of the ellipse's major axis. Preserving this value facilitates the continuous refinement of the search space.


(a) The Initial Smooth Path


(b) Few Deleted Nodes and Narrower Search Space


(c) Keep Previous Path to Limit the Search Space.


(d) The Reduced Path Length


(e) The Shorter Path Across Narrow Passage

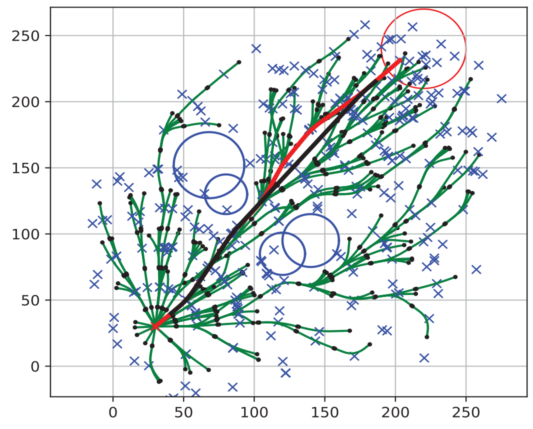
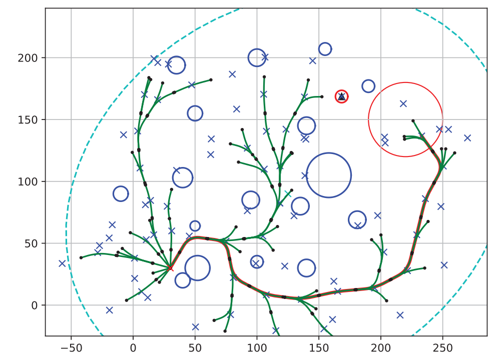
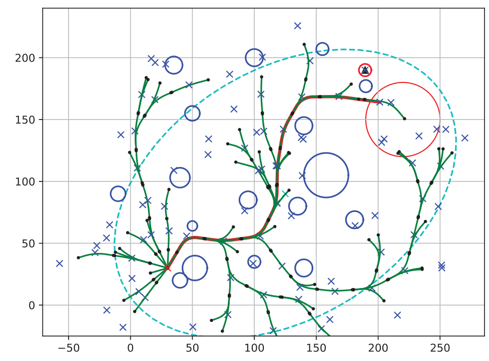
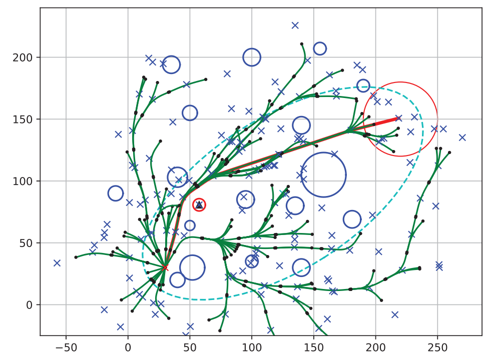

(f) The Final Path After 300 Iterations

Figure 4.1: The Python Simulation 1 of Informed SRRT$^{\#}$


(a) The Initial Smooth Path


(b) Further Optimized Path Length


(c) The Path Length Close to Optimal

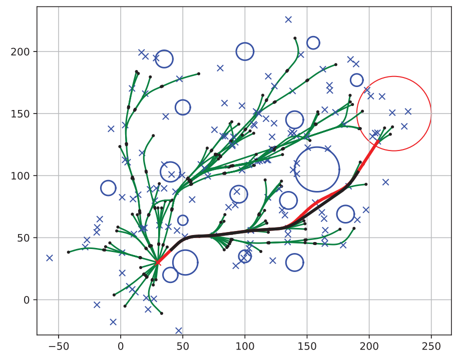
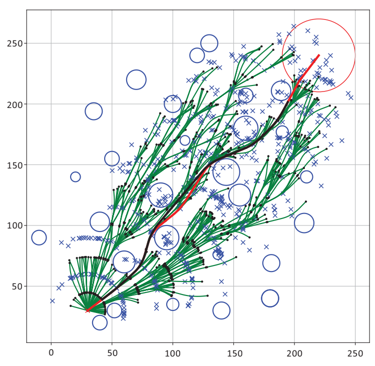
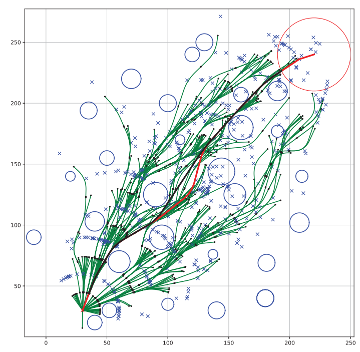

(d) The Final Path After 450 Iterations

Figure 4.2: The Python Simulation 2 of Informed SRRT$^{\#}$

In Simulation 3, we conducted an extensive run of 1000 iterations and compared three results of informed SRRT$^{\#}$, informed RRT$^{\#}$ and SRRT.


(a) Case 1


(b) Case 2

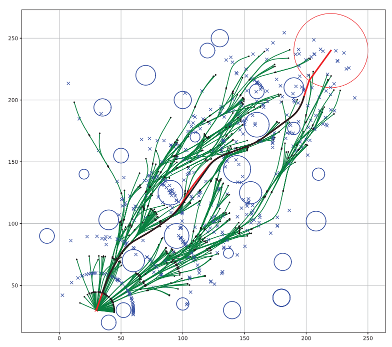
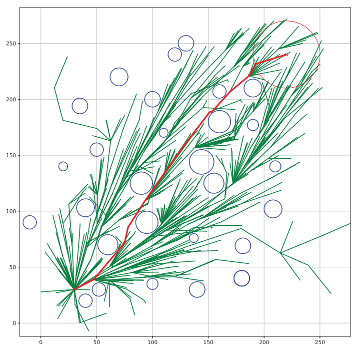
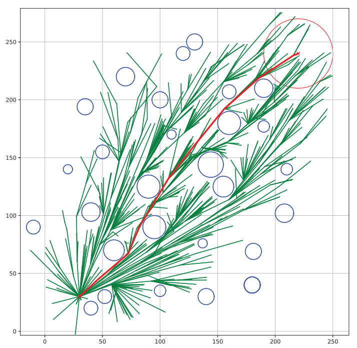

(c) Case 3

Figure 4.3: Final Path of Informed SRRT$^{\#}$ After 1000 Iterations


(a) Case 1


(b) Case 2

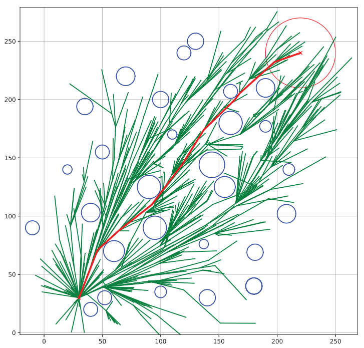
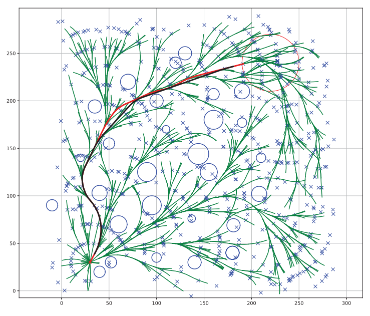
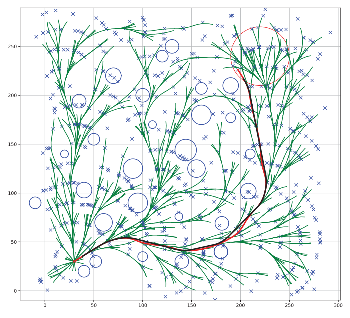

(c) Case 3

Figure 4.4: Final Path of Informed RRT$^{\#}$ After 1000 Iterations


(a) Case 1


(b) Case 2

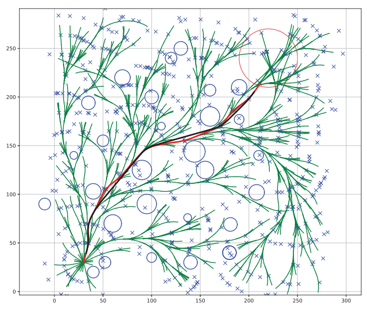
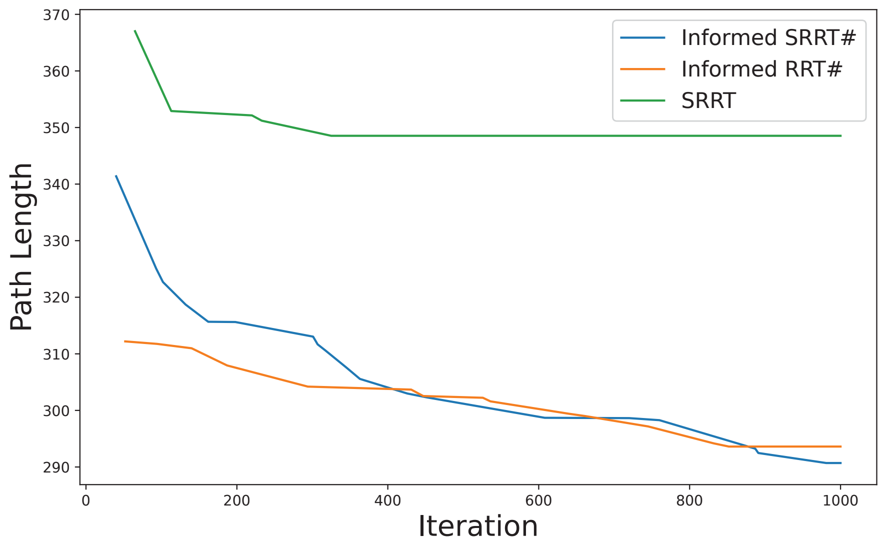
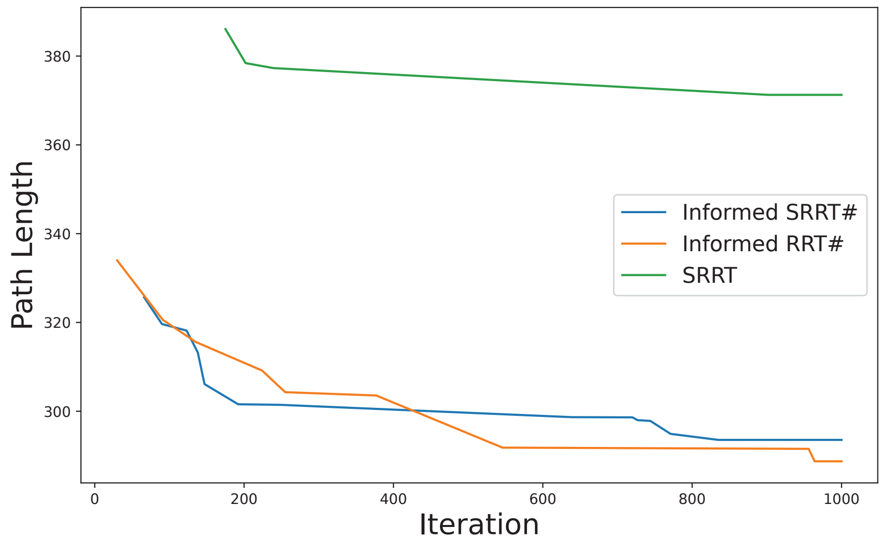

(c) Case 3

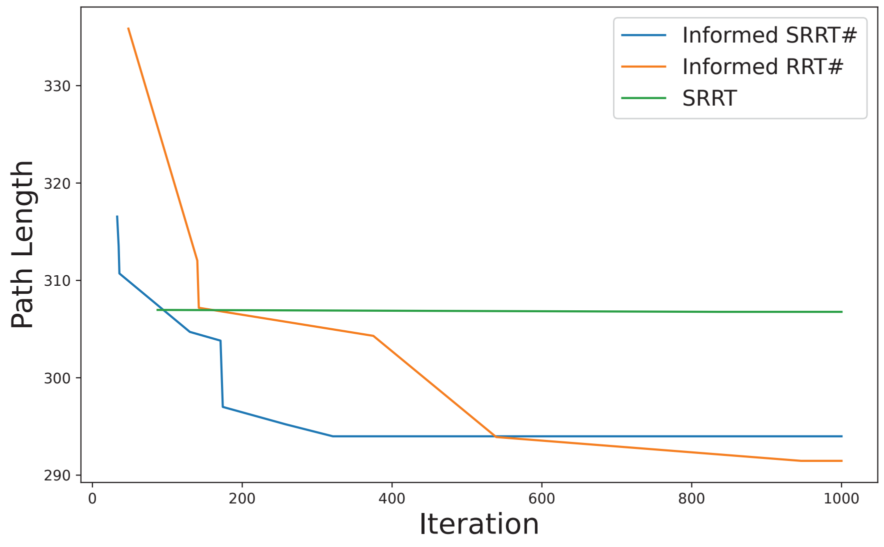
Figure 4.5: Final Path of SRRT After 1000 Iterations



(a) Case 1

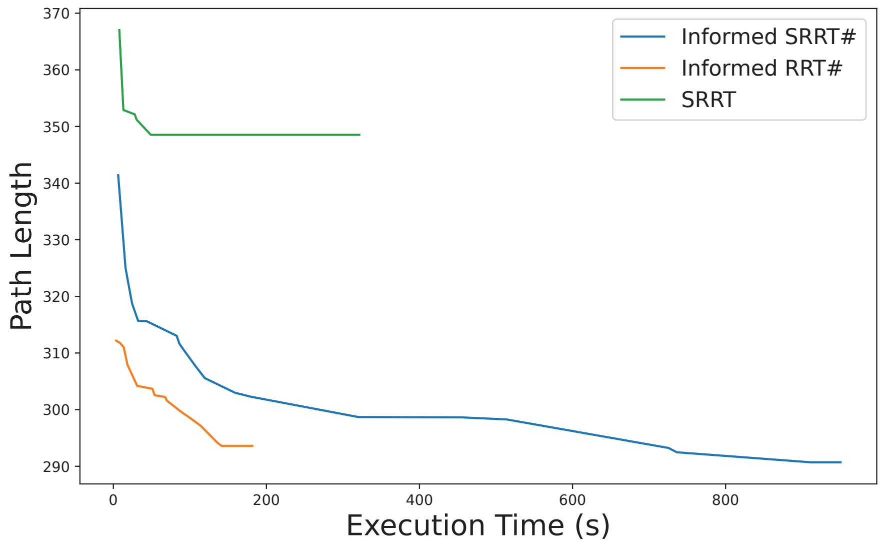


(b) Case 2

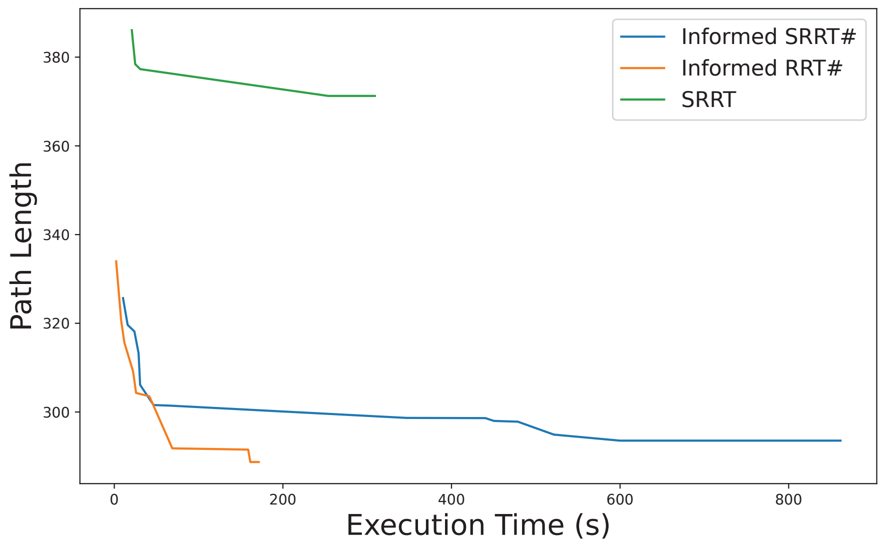


(c) Case 3

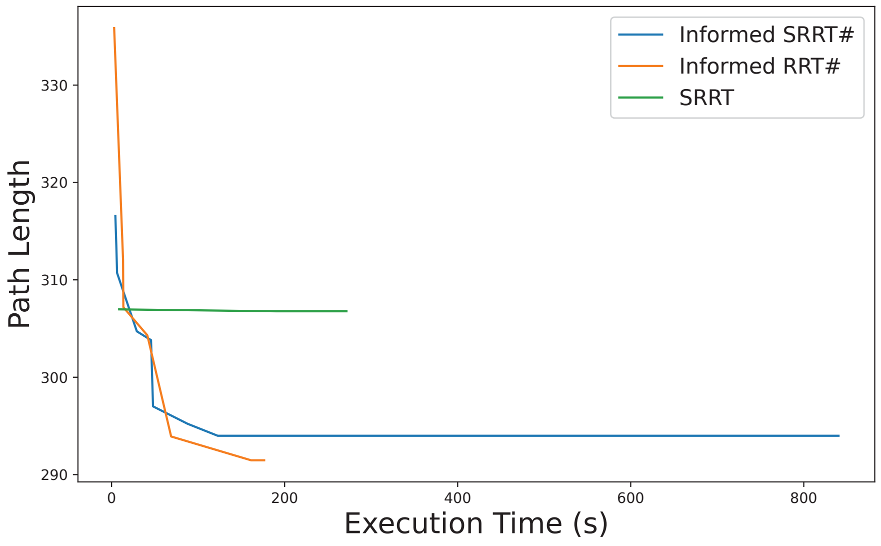
Figure 4.6: Compare Three Algorithms' Path Length over Iterations.



(a) Case 1



(b) Case 2



(c) Case 3

Figure 4.7: Compare Three Algorithms' Path Length over Time.

Finally, we compare these three algorithms in final path lengths, iterations and program running time are presented in Table 4.1.

	SRRT		Informed SRRT#		Informed RRT#	
	Time(s)	Length(m)	Time(s)	Length(m)	Time(s)	Length(m)
1	321.384	348.535	950.147	290.702	181.499	293.612
2	309.13	371.258	861.515	293.547	171.354	288.726
3	271.479	306.771	840.171	293.99	176.379	291.47

Table 4.1: Comparison of Three Algorithms' Path Length and Computational Time.

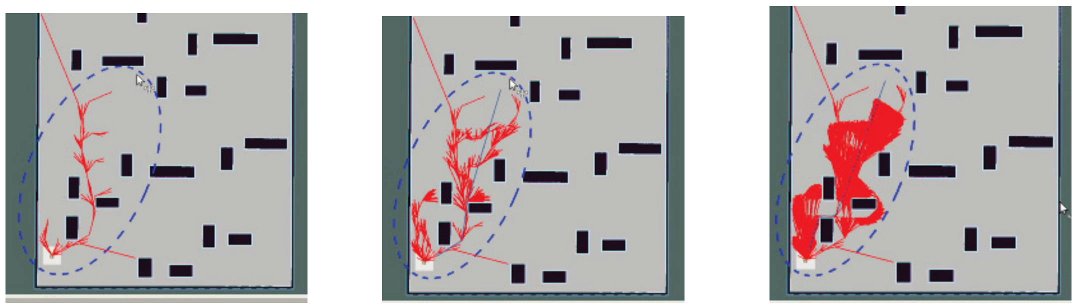
The analytical results demonstrate that our approach can shorten the path length compared to SRRT. However, it does come with a trade-off: the computational time is increased. To run the same 1000 iterations, our method requires triple the time that SRRT does. And when compared to conventional methods like Informed RRT* or RRT*, the time factor increases by approximately four or five times.

Nevertheless, as shown in Fig. 4.6 and Fig. 4.7, even if we allow the same number of iterations (assume maximum iterations: 500) or time (assume maximum time: 350 sec) for path planning, the path length reduced compared to SRRT while it maintains the same level smoothness of G^2 continuity.

4.2 Simulation in Robotics Operation System

We evaluated the performance of both Informed SRRT$^{\#}$ and Informed RRT$^{\#}$ in ROS1, utilizing Gazebo and Rviz. The environment presented a map size of $(280m, 450m)$ populated with clustered obstacles, yet we can assume ideal conditions for all other environmental settings. Recognizing the impracticality of allowing path calculation to exceed one minute in real-world applications, we set time constraints for the planner to generate a final path. The efficiency of the C++ programming language enabled us to complete the calculation significantly quicker than would have been done with Python.

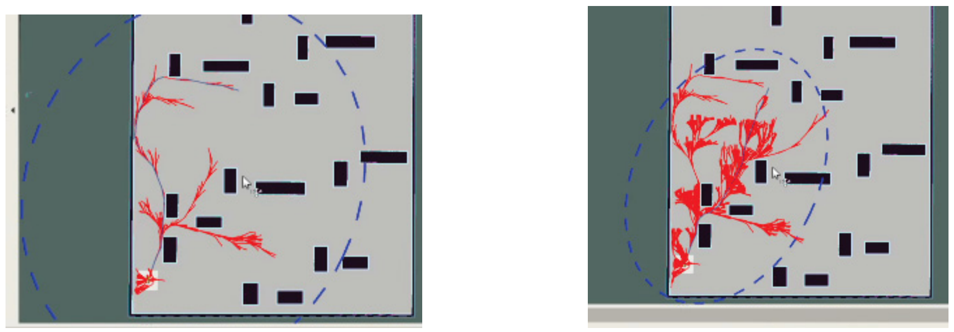
Simulation 1 sets the initial point of the USV as $(0m, 0m)$ and the goal point as $(92m, 187m)$. The iterative process of Informed RRT$^{\#}$ is shown in Figs. 4.8a to 4.8c.



(a) Initial Path (b) Improved Path (c) Final Path

Figure 4.8: ROS Simulation 1 of Informed RRT$^{\#}$

The simulation of Informed SRRT$^{\#}$ is shown in Figs. 4.9a and 4.9b.

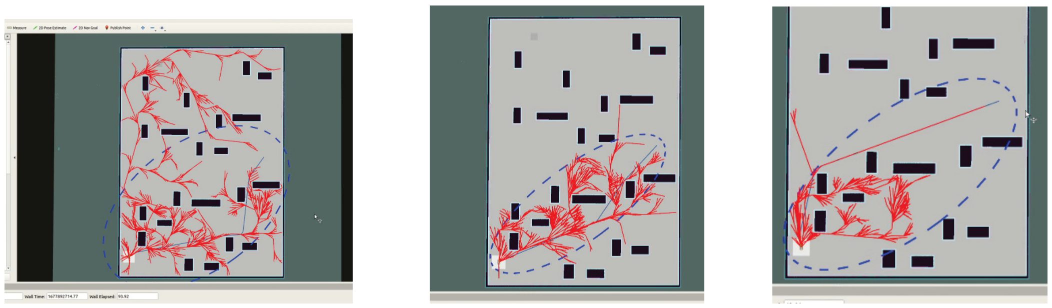


(a) Initial Path (b) Final Path

Figure 4.9: ROS Simulation 1 of Informed SRRT$^{\#}$

Within a time frame of 20 seconds, the final path length of Informed RRT$^{\#}$ is 238.5m, while Informed SRRT$^{\#}$ has a slightly longer final path length of 264.8m. Despite its longer length, the path generated by Informed SRRT$^{\#}$ exhibits smoother steering and navigation around these three obstacles.

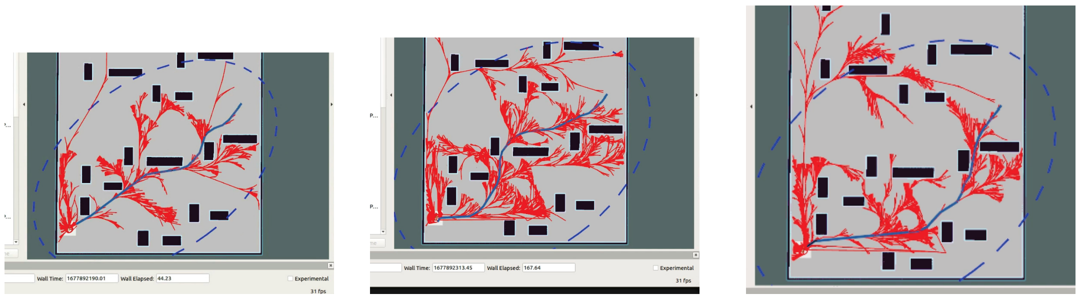
Simulation 2 sets the initial point of the USV as $(0m, 0m)$ and the goal point as $(256m, 167m)$. The simulation results of Informed RRT$^{\#}$ is shown in Fig. 4.10.



(a) Simulation Result 1 (b) Simulation Result 2 (c) Simulation Result 3

Figure 4.10: ROS Simulation 2 of Informed RRT$^{\#}$

The simulation results of Informed SRRT$^{\#}$ is shown in Fig. 4.11.



(a) Simulation Result 1 (b) Simulation Result 2 (c) Simulation Result 3

Figure 4.11: ROS Simulation 2 of Informed SRRT$^{\#}$

Using the same 20-second time frame from Simulation 1 as a baseline, Simulation 2 showed some similarity in results of the final path lengths for Informed RRT$^{\#}$ and Informed SRRT$^{\#}$. The numerical results are listed in Table 4.2. In the same amount of time required to search for a path, Informed SRRT$^{\#}$ has a slightly longer path length compared to Informed RRT$^{\#}$, but it always guarantees G^2 continuity, which Informed RRT$^{\#}$ does not.

	Informed SRRT$^{\#}$	Informed RRT$^{\#}$
1	376.7m	405.1m
2	403.8m	342.6m
3	410.6m	385.9m

Table 4.2: Comparison of Two Algorithms' Path Lengths Within 20 Seconds.

Through the analysis of our Python and ROS simulation outcomes, we have demonstrated that the optimization methods modified from Informed RRT$^{\#}$ are not only accurate but also efficient. Our method consistently produces shorter paths, outperforming SRRT in this aspect. The application of the Bézier spline results in a smoother path compared to the use of a straight line, although there is a slightly lower rate of convergence due to the additional kinematic constraints. These trade-offs can be considered in future optimization.

Chapter 5

Experimental Results

5.1 The TurtleBot3 Platform

We conducted our field test using TurtleBot3 to simulate real-world conditions. The TurtleBot3 is a compact, customizable, and programmable mobile robot that has become an industry standard for educational and research applications. This versatile platform is built around ROS, enabling users to take full advantage of the broad range of software libraries and tools available in ROS ecosystem.

TurtleBot3 is available in multiple configurations, including Burger and Waffle models, each offering different capabilities and characteristics. TurtleBot3 Burger, for instance, is a two-wheeled differential drive robot with encoders, while TurtleBot3 Waffle is a four-wheeled robot with a more robust structure and higher payload capacity.

TurtleBot3 platform is equipped with a variety of sensors, including a 360-degree LIDAR sensor, an Inertial Measurement Unit (IMU), and encoders on each wheel. These sensors allow the robot to perceive its environment and track its motion, enabling complex tasks such as navigation, mapping, and object tracking.

Overall, TurtleBot3 is a powerful and versatile platform that is well-equipped for such tasks. The choice of TurtleBot3 for testing our motion planning algorithm is motivated by several factors. First, the compatibility with ROS allows us to implement and test our algorithm in a well-supported software environment. Moreover, one of its core technologies is the SLAM (Simultaneous Localization and Mapping) algorithm, which allows the robot to create a map of its environment while simultaneously keeping track of its location within the map. The robot also boasts robust hardware and an extensive sensor suite. Finally, the compact size and mobility of the TurtleBot3 make it suitable for testing in a variety of indoor environments.

The subsequent sections of this chapter will detail the integration of our algorithm into the TurtleBot3 platform, the experimental setup, and the results obtained.

5.2 Integration of the Proposed Algorithm with ROS Navigation Stack

The implementation of the proposed motion planning algorithm involves the integration with ROS Navigation Stack and the TurtleBot3 platform. ROS Navigation Stack is a 2D navigation software that takes information from odometry, Lidar sensor streams, and a goal pose to generate safe velocity commands that are transmitted to the mobile base [50].

The Navigation Stack presumes that the robot is configured in a specific manner, as detailed in ROS Navigation tutorials [51]. To ensure compatibility with it, several ROS packages are set up to meet the following requirements:

ROS Installation: The Navigation Stack presupposes that the robot is using ROS. Consequently, Core ROS packages must be installed on TurtleBot3.

Transform Configuration: The Navigation Stack requires the robot to publish information about the relationships between coordinate frames using tf, a package to record multiple coordinate frames. Accordingly, this configuration has to be set up on TurtleBot3.

Sensor Information: The Navigation Stack employs sensor data to avoid obstacles. It presumes that these sensors publish either LaserScan or PointCloud messages over ROS.

Odometry Information: The Navigation Stack necessitates odometry information to be published using tf and the nav_msgsOdometry message. This is ensured on TurtleBot3.

Mapping: In the context of ROS, the creation of a map for the environment is an essential step for the implementation of the Navigation Stack. For this purpose, there are many tools such as gmapping and cartographer, which facilitate the generation of accurate and detailed maps.

AMCL: Adaptive Monte Carlo Localization, is a ROS probabilistic localization system for a robot moving in two dimensional space. It implements an adaptive Monte Carlo localization approach, also known as KLD-sampling. This approach utilizes a particle filter to track the position and orientation of a robot. In essence, AMCL enables the robot to understand its position within the environment represented by the known map.

Base Controller: The Navigation Stack assumes that it can dispatch velocity commands using a geometry_msgsTwist message to the "cmd_vel" topic. This means there must be a ROS node subscribing to the "cmd_vel" topic capable of transforming velocities into motor commands to send to the mobile base. This was integrated on TurtleBot3 using both a Raspberry Pi 4b (Linux server system and ROS) and an OpenCR (a 32-bit microcontroller board).

Once all the necessary software packages and hardware configurations are in place, we can proceed to implement the move_base package, which interfaces directly with the ROS Navigation Stack. The move_base package in ROS is designed to facilitate the navigation for a mobile robot. It attempts to reach a specified goal in the world map. This is accomplished through the move_base node, which links together a global planner and a local planner to complete its global navigation task.

The move_base package is designed to work with any global planner that adheres to the nav_core::BaseGlobalPlanner interface and any local planner that adheres to the nav_core::BaseLocalPlanner interface, both specified in the nav_core package. This allows for flexibility with a wide range of planning algorithms.

Furthermore, the move_base node maintains two costmaps, one for the global planner and one for the local planner. These costmaps are used to facilitate navigation tasks, such as avoiding obstacles and planning optimal paths. The costmaps are created and updated using the data provided by the costmap_2d package, which is another crucial component of the ROS navigation stack.

In the integration of our motion planning algorithm with ROS Navigation Stack, we found that the structured modularity of the Navigation Stack greatly facilitated the process. Each separate module of the Navigation Stack communicates with each other using ROS nodes. The Navigation Stack provides a well-defined and comprehensive API, which allowed for the seamless integration of our algorithm.

This setup allowed us to test and validate our motion planning algorithm in room based environment, ensuring its reliability and robustness.

5.3 Experimental Setup and Results with TurtleBot3 Burger

There are 10 obstacles within an approximate 150×300 cm environment as shown in Fig. 5.1.

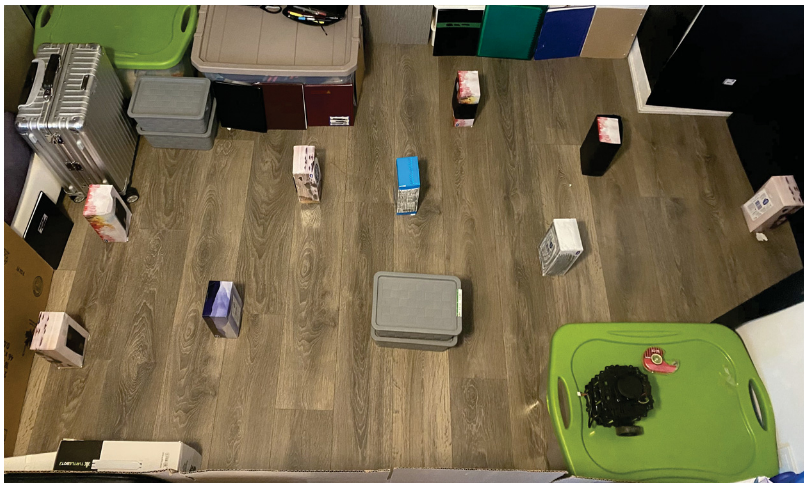


Figure 5.1: The Real Indoor Testing Environment for Path Planners

Initially, we constructed our map utilizing the ROS Gmapping package. The Gmapping package offers laser-based SLAM (Simultaneous Localization and Mapping) functionality via a ROS node known as slam_gmapping. By deploying slam_gmapping, it generates a two-dimensional occupancy grid map, using laser and pose data from sensors. To streamline the experiments, the planning method is deployed after the map is thoroughly explored. The virtual map is depicted in Fig. 5.2.

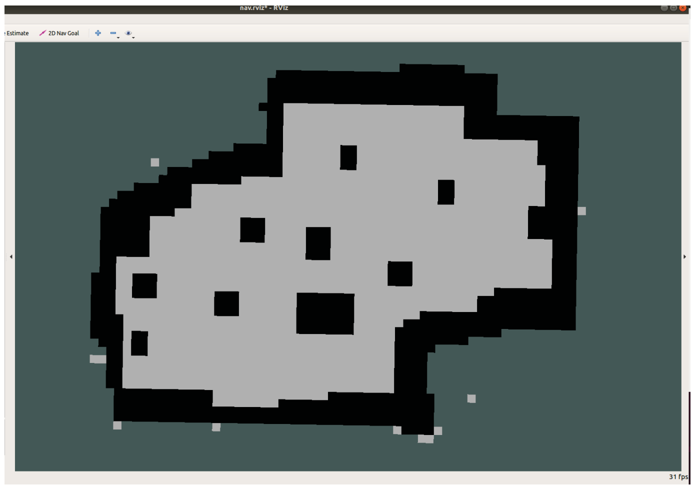
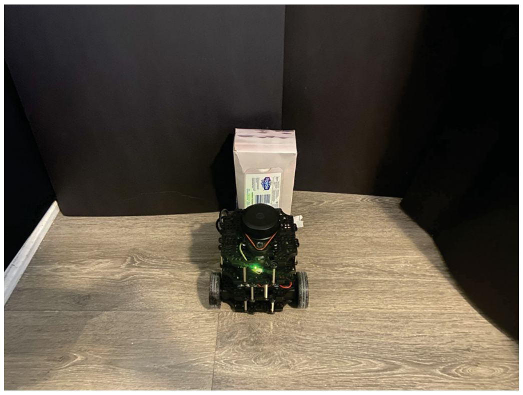
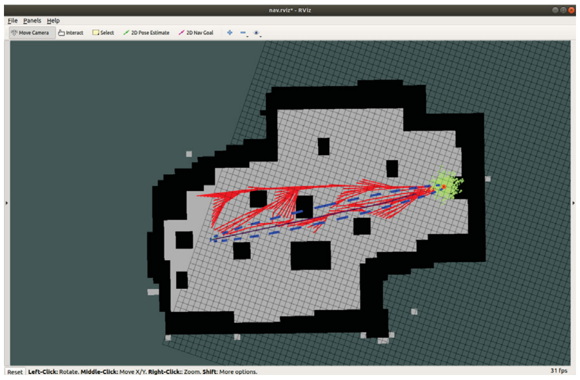


Figure 5.2: The Virtual Map using the SLAM Technology

Two experiments are carried out. The first experiment is to set the destination point that is not directly obstructed by the obstacles. The initial distance between the start and goal point is 1.6m. In Fig. 5.3, the robot started from the right side of the map.

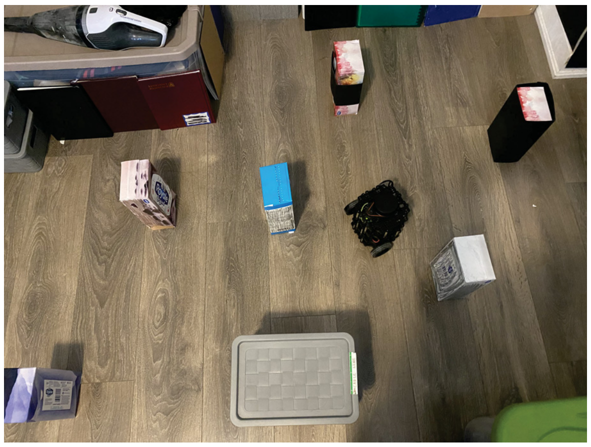

(a) The Physical Map

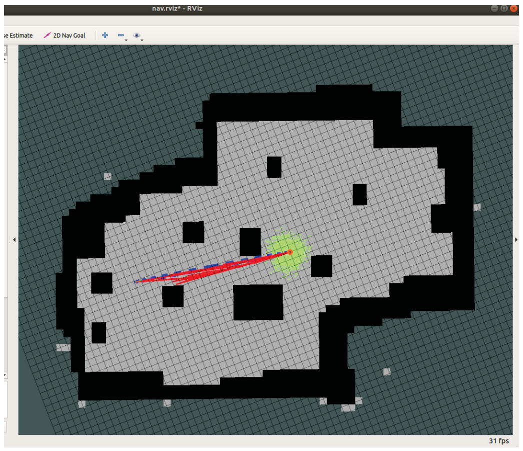


(b) The Virtual Map

Figure 5.3: Test 1: TurtleBot3 at the Origin of Map

The robot successfully navigated past the first obstacle, and the planner quickly converged. As a result, the search space continued to shrink, eventually approximating a straight line. This process is illustrated in Fig. 5.4.

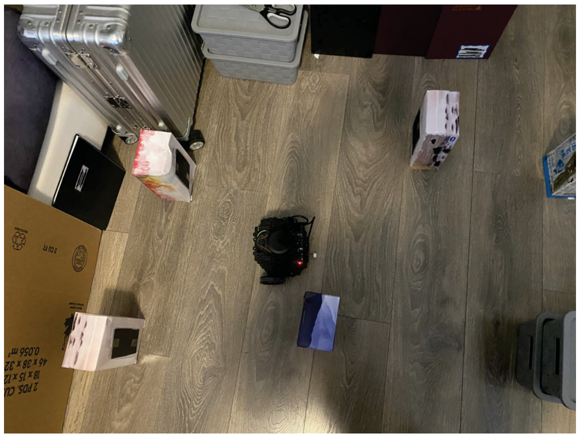

(a) The Physical Map

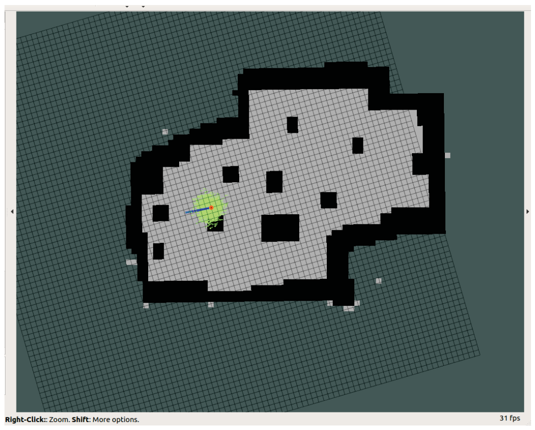


(b) The Virtual Map

Figure 5.4: Test 1: the First Obstacle is Passed.

The robot passed the narrow passage between two obstacles and get close to the goal point near the fourth obstacles as shown in Fig. 5.5.

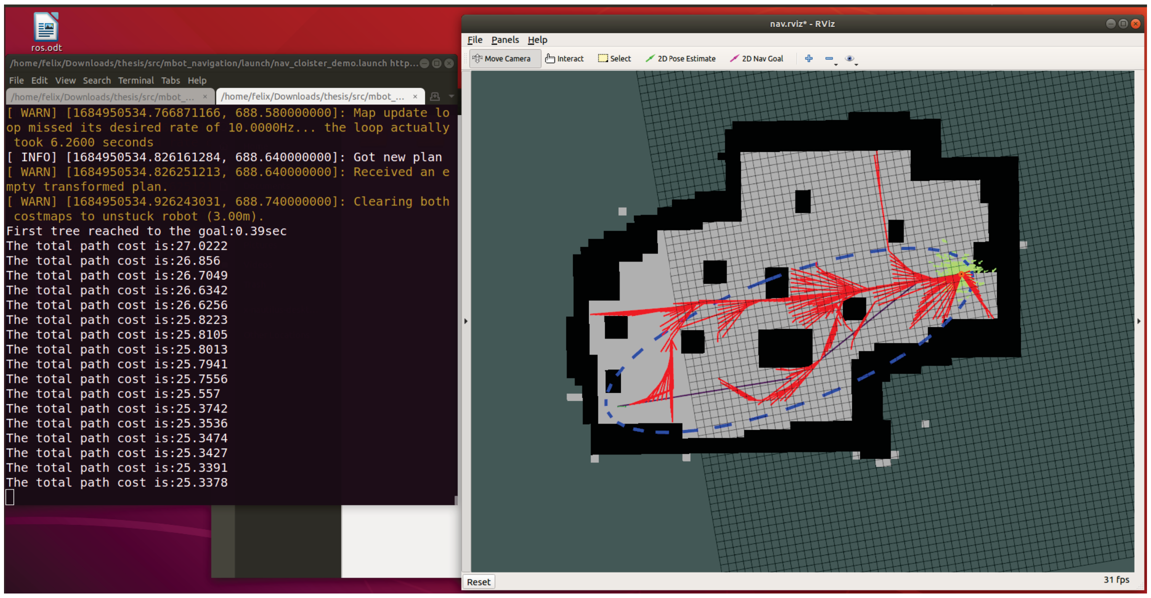

(a) The Physical Map



(b) The Virtual Map

Figure 5.5: Test 1: the Narrow Passage is Passed and the Goal is Reached.

In Experiment 2, the destination region is put in a narrow passage and blocked by two obstacles. The straight line distance between the start and goal point is 2.2m. In Fig. 5.6, the robot started from the same position as Experiment 1.

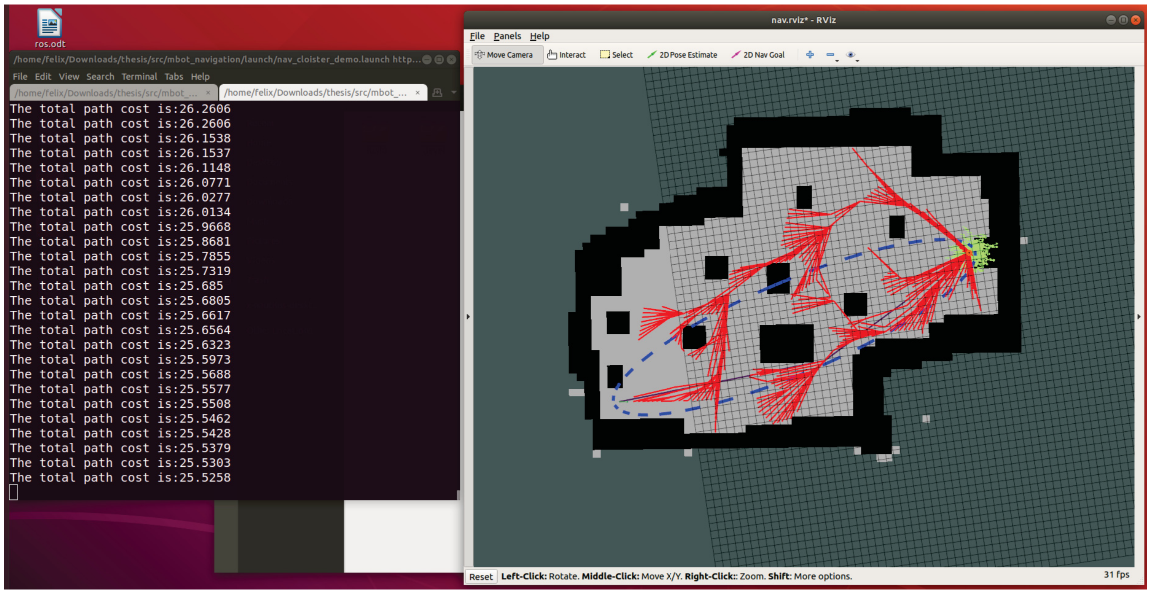

ros.odt
/home/felix/Downloads/thesis/src/mbot_navigation/launch/nav_cloister_demo.launch http...
File Edit View Search Terminal Tabs Help
/home/felix/Downloads/thesis/src/mbot_...
/home/felix/Downloads/thesis/src/mbot_...
nav.rviz* - RViz
File Panels Help
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Reset
31 fps

```
[ WARN] [1684950534.766871166, 688.580000000]: Map update lo
op missed its desired rate of 10.0000Hz... the loop actually
 took 6.2600 seconds
[ INFO] [1684950534.826161284, 688.640000000]: Got new plan
[ WARN] [1684950534.826251213, 688.640000000]: Received an e
mpty transformed plan.
[ WARN] [1684950534.926243031, 688.740000000]: Clearing both
 costmaps to unstuck robot (3.00m).
First tree reached to the goal:0.39sec
The total path cost is:27.0222
The total path cost is:26.856
The total path cost is:26.7049
The total path cost is:26.6342
The total path cost is:26.6256
The total path cost is:25.8223
The total path cost is:25.8105
The total path cost is:25.8013
The total path cost is:25.7941
The total path cost is:25.7556
The total path cost is:25.557
The total path cost is:25.3742
The total path cost is:25.3536
The total path cost is:25.3474
The total path cost is:25.3427
The total path cost is:25.3391
The total path cost is:25.3378
```

(a) The Virtual Map

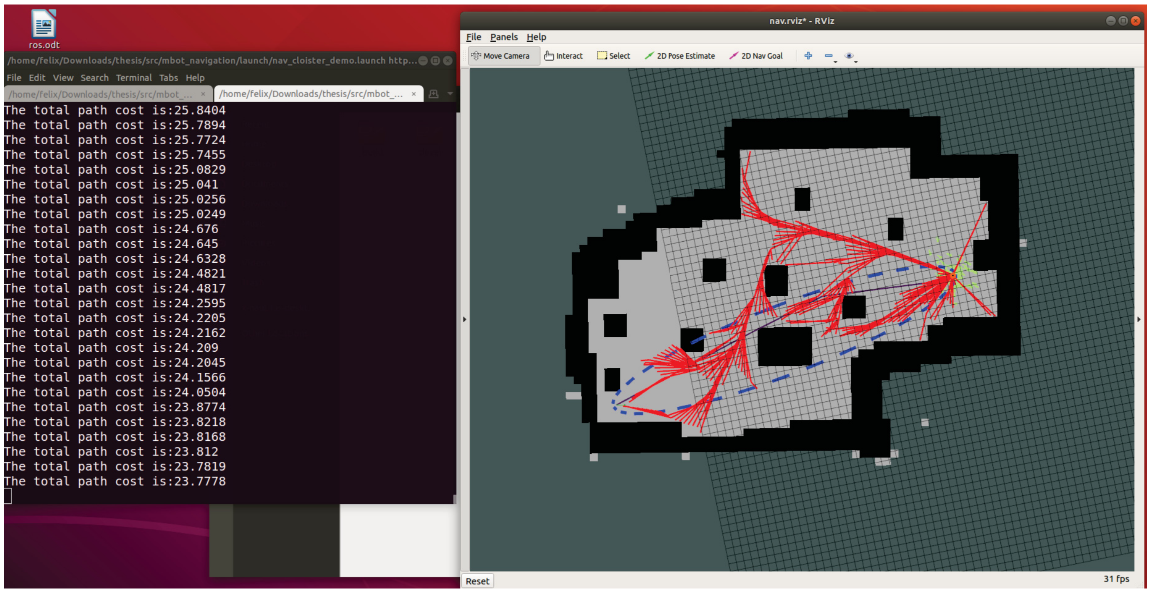
Figure 5.6: Test 2: TurtleBot3 at the Origin of Map

Through re-planning, the search space is quickly converged to a smaller area when the robot is still in the initial region. When the localization module encounters issues, such as providing inaccurate readings as shown in Fig. 5.7 for recalibration, the process of Replanning comes into play. Replanning involves generating new plans in response to changes in the robot's knowledge of the world or when its existing plans no longer align with the current reality.


ros.odt
/home/felix/Downloads/thesis/src/mbot_navigation/launch/nav_cloister_demo.launch http...
File Edit View Search Terminal Tabs Help
/home/felix/Downloads/thesis/src/mbot_...
/home/felix/Downloads/thesis/src/mbot_...
nav.rviz* - RViz
File Panels Help
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.
31 fps

```
The total path cost is:26.2606
The total path cost is:26.2606
The total path cost is:26.1538
The total path cost is:26.1537
The total path cost is:26.1148
The total path cost is:26.0771
The total path cost is:26.0277
The total path cost is:26.0134
The total path cost is:25.9668
The total path cost is:25.8681
The total path cost is:25.7855
The total path cost is:25.7319
The total path cost is:25.685
The total path cost is:25.6805
The total path cost is:25.6617
The total path cost is:25.6564
The total path cost is:25.6323
The total path cost is:25.5973
The total path cost is:25.5688
The total path cost is:25.5577
The total path cost is:25.5508
The total path cost is:25.5462
The total path cost is:25.5428
The total path cost is:25.5379
The total path cost is:25.5303
The total path cost is:25.5258
```

(a) The Virtual Map: Initial Planning

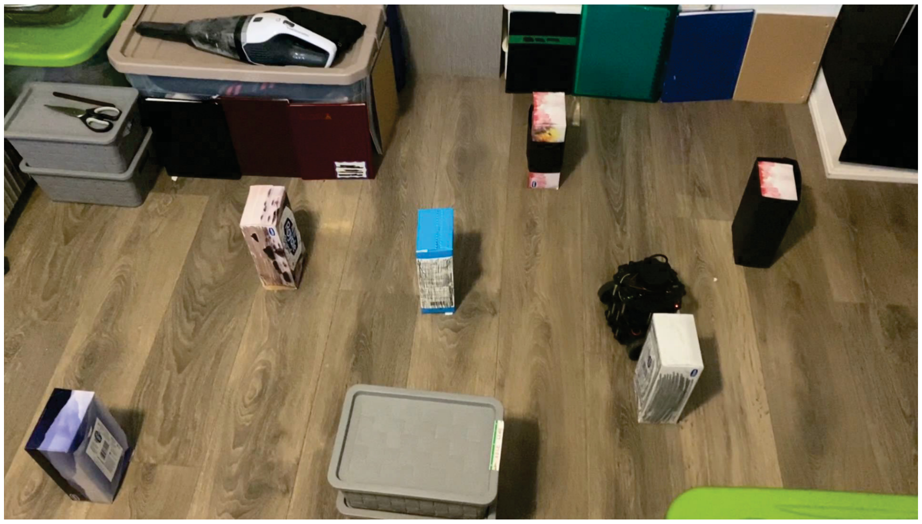

ros.odt
/home/felix/Downloads/thesis/src/mbot_navigation/launch/nav_cloister_demo.launch http...
File Edit View Search Terminal Tabs Help
/home/felix/Downloads/thesis/src/mbot_...
/home/felix/Downloads/thesis/src/mbot_...
nav.rviz* - RViz
File Panels Help
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Reset
31 fps

```
The total path cost is:25.8404
The total path cost is:25.7894
The total path cost is:25.7724
The total path cost is:25.7455
The total path cost is:25.0829
The total path cost is:25.041
The total path cost is:25.0256
The total path cost is:25.0249
The total path cost is:24.676
The total path cost is:24.645
The total path cost is:24.6328
The total path cost is:24.4821
The total path cost is:24.4817
The total path cost is:24.2595
The total path cost is:24.2205
The total path cost is:24.2162
The total path cost is:24.209
The total path cost is:24.2045
The total path cost is:24.1566
The total path cost is:24.0504
The total path cost is:23.8774
The total path cost is:23.8218
The total path cost is:23.8168
The total path cost is:23.812
The total path cost is:23.7819
The total path cost is:23.7778
```

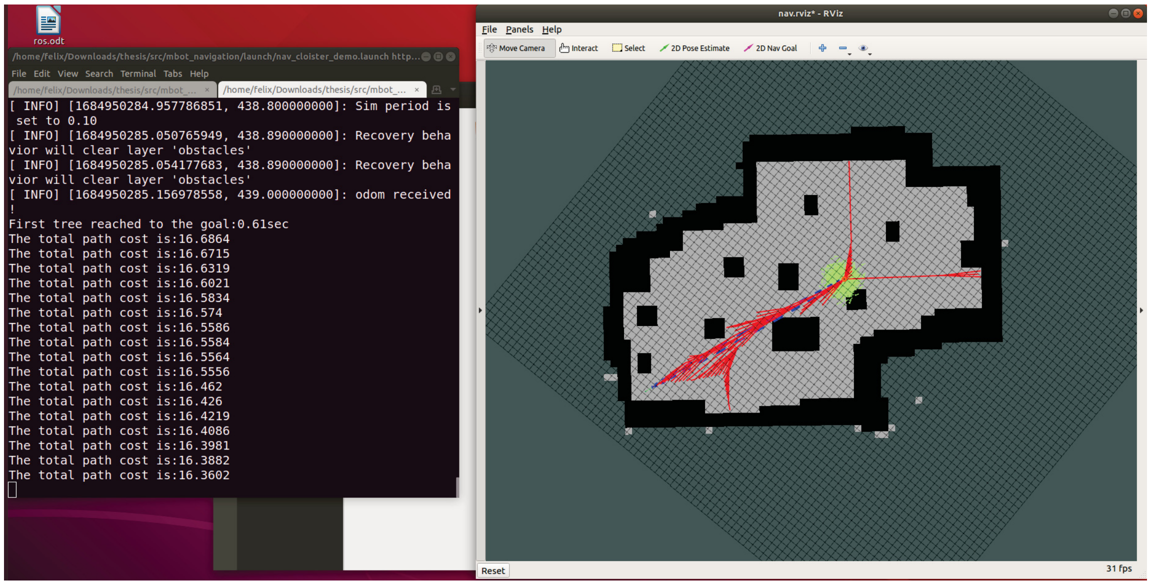
(b) The Virtual Map: Re-planning

Figure 5.7: Test 2: TurtleBot3 with Re-Planning

The robot passed the first obstacle in Fig. 5.8 and aggressively optimized the path length. It successfully and quickly found a straight line optimal path towards the goal point.



(a) The Physical Map

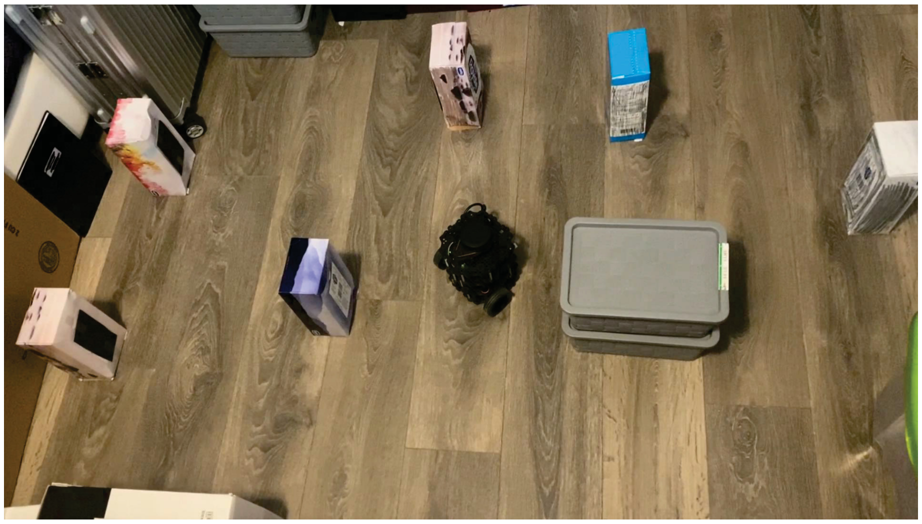

ros.odt
/home/felix/Downloads/thesis/src/mbot_navigation/launch/nav_cloister_demo.launch http...
File Edit View Search Terminal Tabs Help
/home/felix/Downloads/thesis/src/mbot_...
/home/felix/Downloads/thesis/src/mbot_...
nav.rviz* - RViz
File Panels Help
Move Camera
Interact
Select
2D Pose Estimate
2D Nav Goal
Reset
31 fps

```
[ INFO] [1684950284.957786851, 438.800000000]: Sim period is
 set to 0.10
[ INFO] [1684950285.050765949, 438.890000000]: Recovery beha
vior will clear layer 'obstacles'
[ INFO] [1684950285.054177683, 438.890000000]: Recovery beha
vior will clear layer 'obstacles'
[ INFO] [1684950285.156978558, 439.000000000]: odom received
!
First tree reached to the goal:0.61sec
The total path cost is:16.6864
The total path cost is:16.6715
The total path cost is:16.6319
The total path cost is:16.6021
The total path cost is:16.5834
The total path cost is:16.574
The total path cost is:16.5586
The total path cost is:16.5584
The total path cost is:16.5564
The total path cost is:16.5556
The total path cost is:16.462
The total path cost is:16.426
The total path cost is:16.4219
The total path cost is:16.4086
The total path cost is:16.3981
The total path cost is:16.3882
The total path cost is:16.3602
```

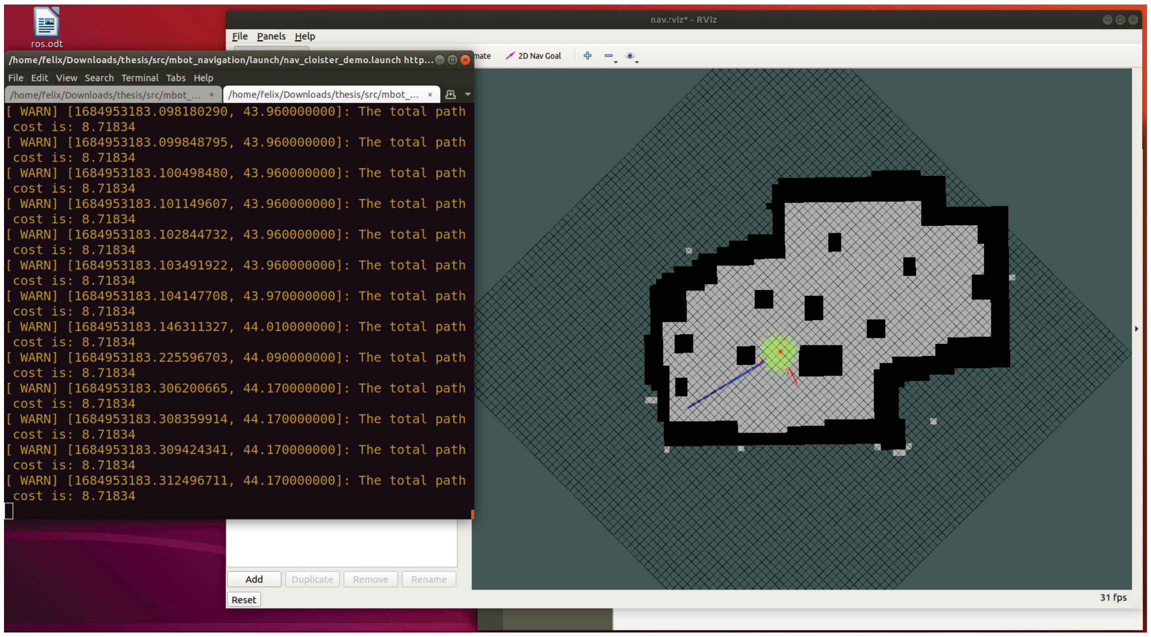
(b) The Virtual Map

Figure 5.8: Test 2: the First Obstacle is Passed.

The robot navigated the narrow passage in Fig. 5.9 and updated the previous path, which can pass through the last obstacle ahead.



(a) The Physical Map



ros.odt
nav.rviz* - RViz
File Panels Help
/home/felix/Downloads/thesis/src/mbot_navigation/launch/nav_cloister_demo.launch http...
File Edit View Search Terminal Tabs Help
/home/felix/Downloads/thesis/src/mbot_...
/home/felix/Downloads/thesis/src/mbot_...
2D Nav Goal
Add
Duplicate
Remove
Rename
Reset
31 fps

```
[ WARN] [1684953183.098180290, 43.960000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.099848795, 43.960000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.100498480, 43.960000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.101149607, 43.960000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.102844732, 43.960000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.103491922, 43.960000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.104147708, 43.970000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.146311327, 44.010000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.225596703, 44.090000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.306200665, 44.170000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.308359914, 44.170000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.309424341, 44.170000000]: The total path
 cost is: 8.71834
[ WARN] [1684953183.312496711, 44.170000000]: The total path
 cost is: 8.71834
```

(b) The Virtual Map

Figure 5.9: Test 2: the Narrow Passage is Passed.

The TurtleBot3 reached to the goal point in Fig. 5.10 and Experiment 2 is finished.

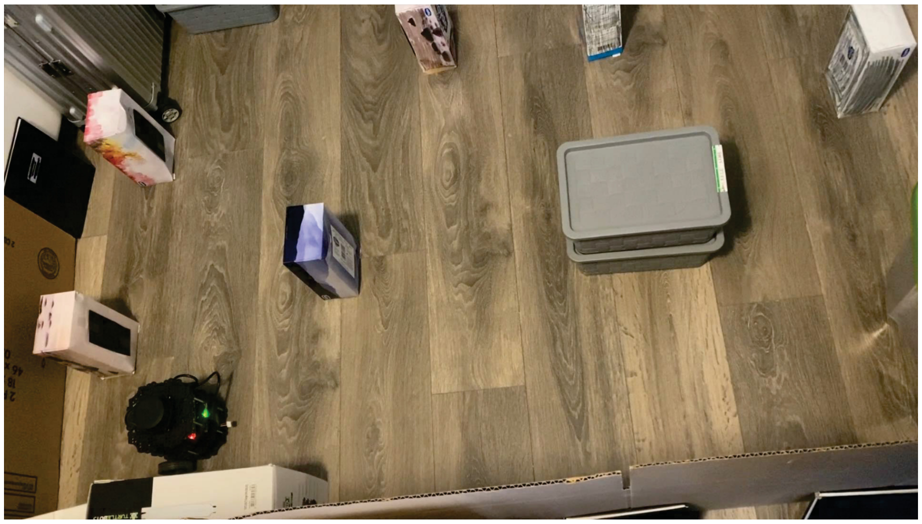
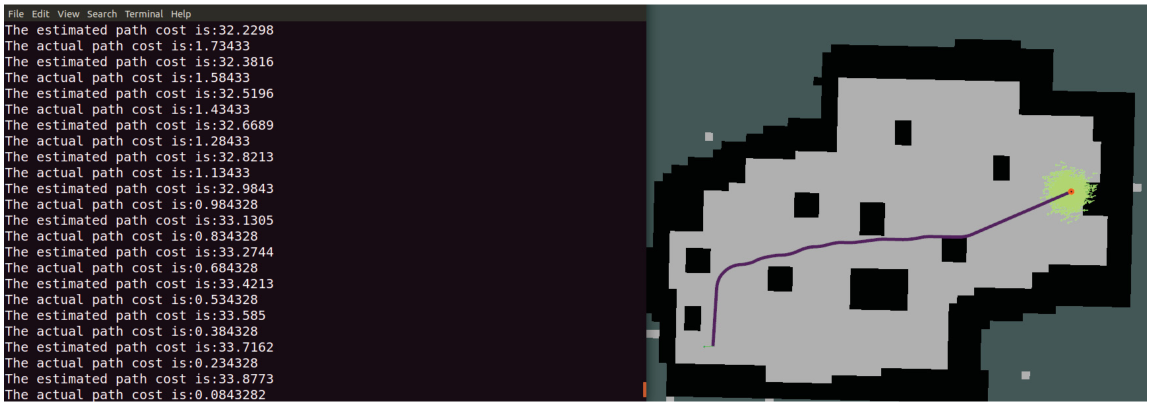


Figure 5.10: Test 2: TurtleBot3 Reached the Designated Goal Region.

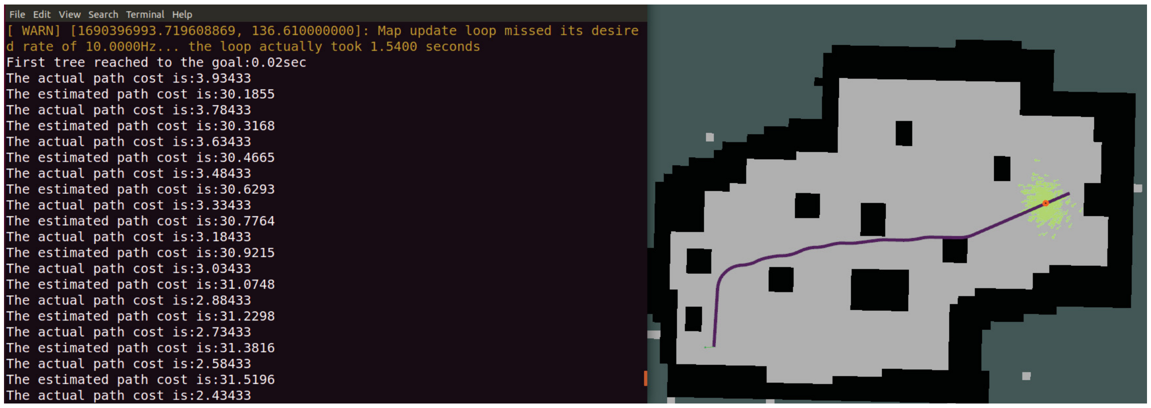
In the final phase of our study, we conducted a comparative analysis between our method used in Experiment 2 and the conventional method from SRRT implemented in Experiment 3. Though the global planners employed in each experiment differ, all other modules remained constant, incorporating the DWA local planner and AMCL for localization. A fixed number of nodes, specifically 10,000, were sampled in both cases. Whenever a shorter path emerged in Experiment 3, it was employed to replace the previously longer one.

After we tested 10 times using SRRT, we identified the most effective initial solutions. As depicted in Fig. 5.11 and Fig. 5.12, the derived solution, although not optimal, provided a feasible path. Notably, this path navigated through obstacles positioned centrally rather than skirting them from either side.



```
The estimated path cost is:32.2298
The actual path cost is:1.73433
The estimated path cost is:32.3816
The actual path cost is:1.58433
The estimated path cost is:32.5196
The actual path cost is:1.43433
The estimated path cost is:32.6689
The actual path cost is:1.28433
The estimated path cost is:32.8213
The actual path cost is:1.13433
The estimated path cost is:32.9843
The actual path cost is:0.984328
The estimated path cost is:33.1305
The actual path cost is:0.834328
The estimated path cost is:33.2744
The actual path cost is:0.684328
The estimated path cost is:33.4213
The actual path cost is:0.534328
The estimated path cost is:33.585
The actual path cost is:0.384328
The estimated path cost is:33.7162
The actual path cost is:0.234328
The estimated path cost is:33.8773
The actual path cost is:0.0843282
```

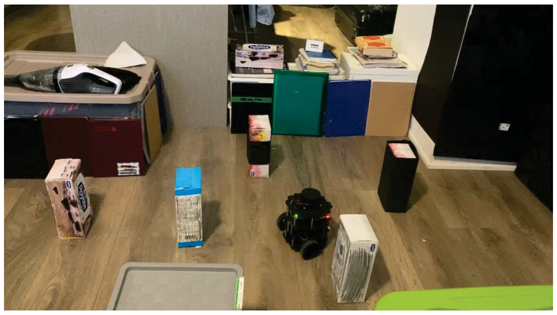
Figure 5.11: Test 3 with SRRT: Initial State

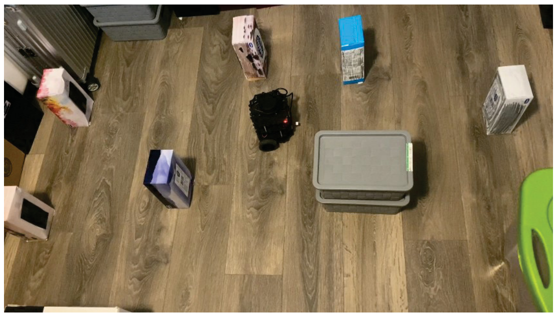


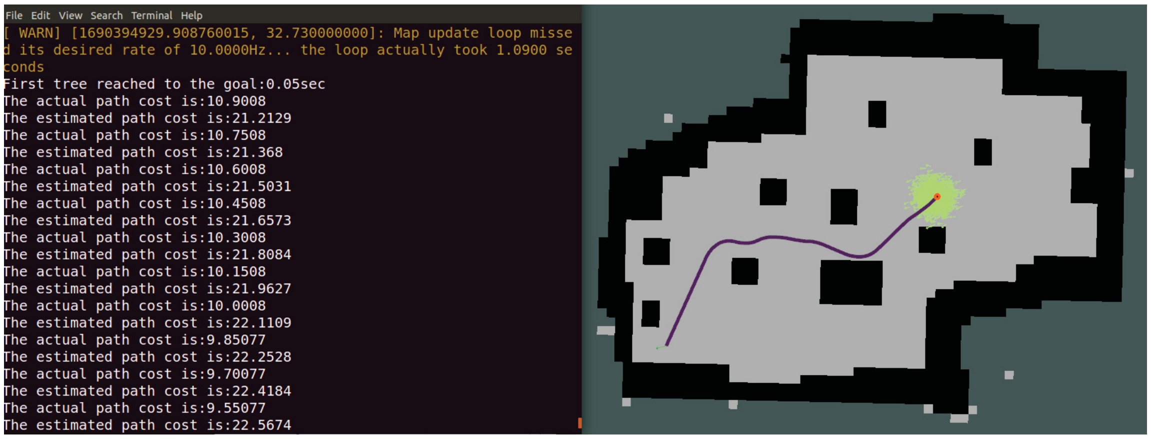
```
[ WARN] [1690396993.719608869, 136.610000000]: Map update loop missed its desire
d rate of 10.0000Hz... the loop actually took 1.5400 seconds
First tree reached to the goal:0.02sec
The actual path cost is:3.93433
The estimated path cost is:30.1855
The actual path cost is:3.78433
The estimated path cost is:30.3168
The actual path cost is:3.63433
The estimated path cost is:30.4665
The actual path cost is:3.48433
The estimated path cost is:30.6293
The actual path cost is:3.33433
The estimated path cost is:30.7764
The actual path cost is:3.18433
The estimated path cost is:30.9215
The actual path cost is:3.03433
The estimated path cost is:31.0748
The actual path cost is:2.88433
The estimated path cost is:31.2298
The actual path cost is:2.73433
The estimated path cost is:31.3816
The actual path cost is:2.58433
The estimated path cost is:31.5196
The actual path cost is:2.43433
```

Figure 5.12: Test 3 with SRRT: Initial State 2

The robot diverged from the initially planned route, initiating its first replanning process. This yielded a new trajectory, as depicted in Fig.,5.14, with its corresponding physical map illustrated in Fig. 5.13a. When contrasted with our method represented in Fig. 5.8, it becomes apparent that the SRRT struggles to consistently reduce the length of its path.


(a) The Physical Map of Replanning 1

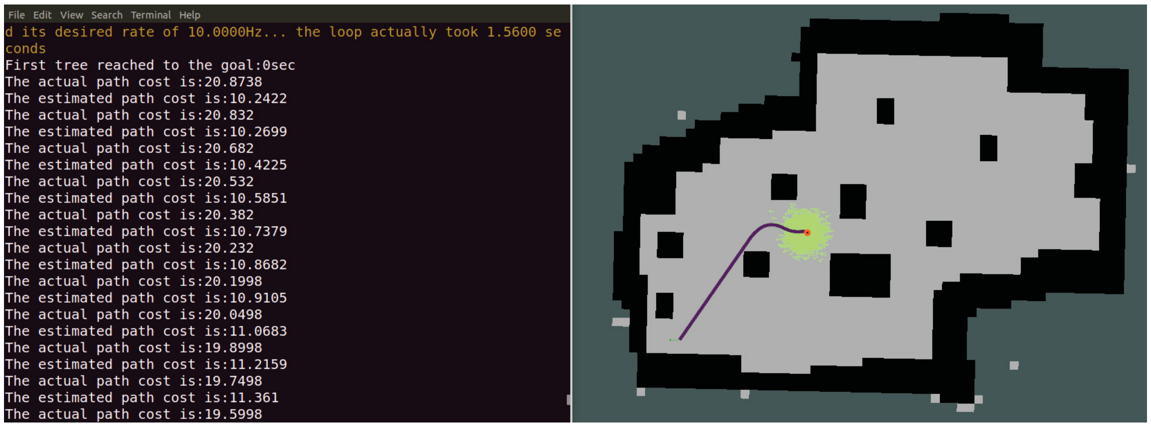

(b) The Physical Map of Replanning 2



```
[ WARN] [1690394929.908760015, 32.730000000]: Map update loop misse
d its desired rate of 10.0000Hz... the loop actually took 1.0900 se
conds
First tree reached to the goal:0.05sec
The actual path cost is:10.9008
The estimated path cost is:21.2129
The actual path cost is:10.7508
The estimated path cost is:21.368
The actual path cost is:10.6008
The estimated path cost is:21.5031
The actual path cost is:10.4508
The estimated path cost is:21.6573
The actual path cost is:10.3008
The estimated path cost is:21.8084
The actual path cost is:10.1508
The estimated path cost is:21.9627
The actual path cost is:10.0008
The estimated path cost is:22.1109
The actual path cost is:9.85077
The estimated path cost is:22.2528
The actual path cost is:9.70077
The estimated path cost is:22.4184
The actual path cost is:9.55077
The estimated path cost is:22.5674
```

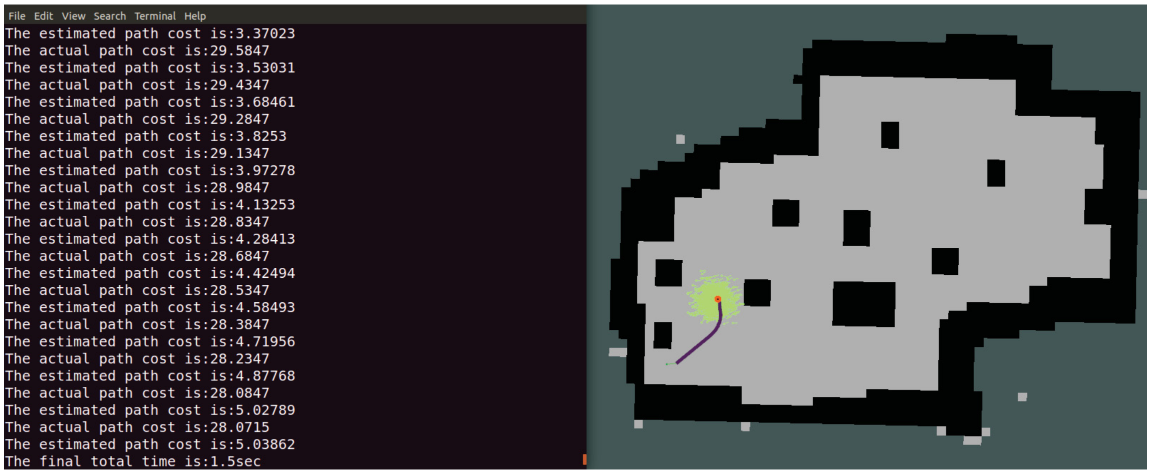
Figure 5.14: Test 3: Replanning 1 After Deviation from Planned Path

Despite not traversing the third narrow passage, the robot bypassed the third obstacle using a side route, as illustrated in Fig. 5.15 and Fig. 5.13b. Ultimately, this maneuver brought it closer to its final destination.



```
d its desired rate of 10.0000Hz... the loop actually took 1.5600 se
conds
First tree reached to the goal:0sec
The actual path cost is:20.8738
The estimated path cost is:10.2422
The actual path cost is:20.832
The estimated path cost is:10.2699
The actual path cost is:20.682
The estimated path cost is:10.4225
The actual path cost is:20.532
The estimated path cost is:10.5851
The actual path cost is:20.382
The estimated path cost is:10.7379
The actual path cost is:20.232
The estimated path cost is:10.8682
The actual path cost is:20.1998
The estimated path cost is:10.9105
The actual path cost is:20.0498
The estimated path cost is:11.0683
The actual path cost is:19.8998
The estimated path cost is:11.2159
The actual path cost is:19.7498
The estimated path cost is:11.361
The actual path cost is:19.5998
```

Figure 5.15: Test 3: Replan Twice and Bypass the Third Obstacle.



```
The estimated path cost is:3.37023
The actual path cost is:29.5847
The estimated path cost is:3.53031
The actual path cost is:29.4347
The estimated path cost is:3.68461
The actual path cost is:29.2847
The estimated path cost is:3.8253
The actual path cost is:29.1347
The estimated path cost is:3.97278
The actual path cost is:28.9847
The estimated path cost is:4.13253
The actual path cost is:28.8347
The estimated path cost is:4.28413
The actual path cost is:28.6847
The estimated path cost is:4.42494
The actual path cost is:28.5347
The estimated path cost is:4.58493
The actual path cost is:28.3847
The estimated path cost is:4.71956
The actual path cost is:28.2347
The estimated path cost is:4.87768
The actual path cost is:28.0847
The estimated path cost is:5.02789
The actual path cost is:28.0715
The estimated path cost is:5.03862
The final total time is:1.5sec
```

Figure 5.16: Test 3: Get Closer to the Goal.

In comparing the estimated and actual paths from Experiments 2 and 3, it is observed that the actual path is longer due to occasional inaccuracies within the robot's odometry and AMCL localization module. The final path executed by the robot using SRRT falls within the range of 3.5m and 4.5m. Contrastingly, the path derived through our method is consistently approximately 3m in length.

The results of the planned paths, originating from the initial state, are demonstrated in Fig. 5.6 and Fig. 5.11. The path planned from the initial state using SRRT in Experiment 3 is $3.3m$, whereas the path determined by our method measures $2.7m$.

Our path planning algorithm performed successfully in the real scenarios. The algorithm guided the robot's local planner and controller around obstacles, which contributed to a new navigation strategy that reduces the path length and remains smoothness. In the real testing, the localization module sometimes detected a sudden change in the robot's position due to a sensor error or some external factor. In that case, the robot's existing plan could become invalid. Replanning came into play to generate a new plan that reflected its new understanding of the world. This can involve re-calculating a path to a goal. TurtleBot3 continued to function effectively even when AMCL or odometry module was not working correctly or robot itself encountered unexpected changes in its environment. The search space continued to shrink and path length became shorter during the operation, demonstrating the algorithm's capability to refine and focus its path planning effectively. This successful field test is a milestone, confirming the robustness of our algorithm and its readiness for more complex applications.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

The primary contribution of this thesis is the creation and demonstration of an innovative path planning algorithm tailored for non-holonomic robot systems. This algorithm has been proven to locate a near-optimal path that maintains continuous curvature at all points.

Our approach ensures the G^2 continuity of curvature stays within pre-set constraints. For paths longer than the minimum extended edges from SRRT between three consecutive states' intermediate points and under maximum curvature constraints, the algorithm can be subdivided into piece-wise Bézier spline segments.

During the initial steering process, the approach accommodates a newly sampled point even if it is unable to directly connect to the nearest point due to path collision. The optimization function of our algorithm encompasses two main parts: the first seeks to connect the new sampled node to the best neighbor node in a manner that satisfies kinematic constraints, ensuring collision-free paths; the second part introduces three alternative methods that can rewire tree nodes when at least three consecutive nodes satisfy the kinematic constraints.

Two modified rewiring processes from RRT* are integral to our proposed method. The first process involves the optimization of the initial path; during this phase, subtrees might be partially removed or reused as “new” sample nodes if previous vertices and edges conflict with the kinematic constraints. To maintain curvature continuity, a second rewiring process was introduced that mandates the consideration of at least three or even four consecutive states.

To ensure the robustness and efficacy of the proposed path planning algorithm, comprehensive simulations were conducted using Python 3 and ROS with C++. These simulations demonstrated that the algorithm's capacity to create efficient paths under conditions mimicking those of ground robots. In addition, a field test was performed using a TurtleBot3, providing evidence of the algorithm's real-world applicability and effectiveness.

The success of this work is exhibited in simulation and field-test results, validating the effectiveness and efficiency of the modified optimization processes adapted from Informed RRT* and RRT$^\#$, offering a similar degree of smoothness as SRRT. The application of path smoothing techniques further refined the final path.

6.2 Future Work

The potential applications and developments of this work are vast, where our approach can pass through the sampled local control nodes, a contrast to the traditional use of B-splines with RRT*. We also plan to investigate and delve into other spline methods. Additionally, we are intrigued by the prospect of extending this method to higher dimensional space or non-Euclidean state spaces. Such a move could open up new dimensions of possibilities, allowing for more diverse applications and addressing more complex robotic path planning scenarios.

Looking ahead, our research can also harness the power of other artificial intelligence (AI) technologies, especially machine learning (ML) to enhance our proposed path planning algorithm, potentially solving several of its present limitations. The integration of ML in sampling-based motion planners has been gaining attention due to their ability to effectively tune several key parameters based on the environment or the limits of the robots. It can also mitigate the "brute force" approach often employed by these planners.

In this context, AI and ML could greatly improve this process by automatically determining these tuning parameters and optimizing their sizes according to a predefined metric. By doing so, it is expected that the AI could effectively trim the unnecessary growth of the search space while still ensuring the optimal path planning.

Particularly in the case when our algorithm apply some traits of SRRT, there are two essential parameters: curvature and angles, which are bound by the robot's kinematic limitations. Tuning these parameters manually can be a daunting task but it is important to decide the minimum expanded length, often requiring a deep understanding of the system and extensive trials. An AI-driven system could intuitively learn and adapt the tuning of these parameters, optimizing the path planning process according to the unique characteristics of each robot and its environment.

Bibliography

[1] A. Gasparetto, P. Boscariol, A. Lanzutti, and R. Vidoni, "Path planning and trajectory planning algorithms: A general overview," *Motion and Operation Planning of Robotic Systems: Background and Practical Approaches*, pp. 3–27, 2015.

[2] D. González, J. Pérez, V. Milanés, and F. Nashashibi, "A review of motion planning techniques for automated vehicles," *IEEE Transactions on intelligent transportation systems*, vol. 17, no. 4, pp. 1135–1145, 2015.

[3] B. Paden, M. Čáp, S. Z. Yong, D. Yershov, and E. Frazzoli, "A survey of motion planning and control techniques for self-driving urban vehicles," *IEEE Transactions on intelligent vehicles*, vol. 1, no. 1, pp. 33–55, 2016.

[4] C. Niederberger, D. Radovic, and M. Gross, "Generic path planning for real-time applications," in *Proceedings Computer Graphics International, 2004.*, pp. 299–306, IEEE, 2004.

[5] C. Zhou, B. Huang, and P. Fränti, "A review of motion planning algorithms for intelligent robots," *Journal of Intelligent Manufacturing*, vol. 33, no. 2, pp. 387–424, 2022.

[6] R. Meyes, H. Tercan, S. Roggendorf, T. Thiele, C. Büscher, M. Obdenbusch, C. Brecher, S. Jeschke, and T. Meisen, "Motion planning for industrial robots using reinforcement learning," *Procedia CIRP*, vol. 63, pp. 107–112, 2017.

[7] Z. Liu, Y. Zhang, X. Yu, and C. Yuan, "Unmanned surface vehicles: An overview of developments and challenges," *Annual Reviews in Control*, vol. 41, pp. 71–93, 2016.

[8] M. Panda, B. Das, B. Subudhi, and B. B. Pati, "A comprehensive review of path planning algorithms for autonomous underwater vehicles," *International Journal of Automation and Computing*, vol. 17, no. 3, pp. 321–352, 2020.

[9] X. Li, Z. Sun, D. Cao, D. Liu, and H. He, "Development of a new integrated local trajectory planning and tracking control framework for autonomous ground vehicles," *Mechanical Systems and Signal Processing*, vol. 87, pp. 118–137, 2017.

[10] E. W. Dijkstra, "A note on two problems in connexion with graphs," in *Edsger Wybe Dijkstra: His Life, Work, and Legacy*, pp. 287–290, 1959.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE transactions on Systems Science and Cybernetics*, vol. 4, no. 2, pp. 100–107, 1968.

[12] A. Stentz, “Optimal and efficient path planning for partially known environments,” in *Intelligent unmanned ground vehicles*, pp. 203–220, Springer, 1997.

[13] S. Koenig and M. Likhachev, “Fast replanning for navigation in unknown terrain,” *IEEE Transactions on Robotics*, vol. 21, no. 3, pp. 354–363, 2005.

[14] M. Likhachev, G. J. Gordon, and S. Thrun, “Ara*: Anytime a* with provable bounds on sub-optimality,” *Advances in neural information processing systems*, vol. 16, 2003.

[15] I. Pohl, “First results on the effect of error in heuristic search,” *Machine Intelligence*, vol. 5, pp. 219–236, 1970.

[16] D. Harabor and A. Grastien, “Online graph pruning for pathfinding on grid maps,” in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 25, pp. 1114–1119, 2011.

[17] S. M. LaValle *et al.*, “Rapidly-exploring random trees: A new tool for path planning,” 1998.

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

[19] R. Bohlin and L. Kavraki, “Path planning using lazy prm,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 1, pp. 521–528 vol.1, 2000.

[20] J. J. Kuffner and S. M. LaValle, “Rrt-connect: An efficient approach to single-query path planning,” in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, vol. 2, pp. 995–1001, IEEE, 2000.

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

[22] O. Arslan and P. Tsiotras, “Use of relaxation methods in sampling-based algorithms for optimal motion planning,” in *2013 IEEE International Conference on Robotics and Automation*, pp. 2421–2428, IEEE, 2013.

[23] L. Janson, E. Schmerling, A. Clark, and M. Pavone, “Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions,” *The International journal of robotics research*, vol. 34, no. 7, pp. 883–921, 2015.

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

[25] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, vol. 79, no. 3, pp. 497–516, 1957.

[26] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[27] C. Runge, "Über empirische funktionen und die interpolation zwischen äquidistanten ordinaten," *Zeitschrift für Mathematik und Physik*, vol. 46, no. 224-243, p. 20, 1901.

[28] C. De Boor and C. De Boor, *A practical guide to splines*, vol. 27. springer-verlag New York, 1978.

[29] E. Catmull and R. Rom, "A class of local interpolating splines," in *Computer aided geometric design*, pp. 317–326, Elsevier, 1974.

[30] M. Brezak and I. Petrović, "Real-time approximation of clothoids with bounded error for path planning applications," *IEEE Transactions on Robotics*, vol. 30, no. 2, pp. 507–515, 2013.

[31] K. Yang, D. Jung, and S. Sukkarieh, "Continuous curvature path-smoothing algorithm using cubic b zier spiral curves for non-holonomic robots," *Advanced Robotics*, vol. 27, no. 4, pp. 247–258, 2013.

[32] J. P. Rastelli, R. Lattarulo, and F. Nashashibi, "Dynamic trajectory generation using continuous-curvature algorithms for door to door assistance vehicles," in *2014 IEEE Intelligent Vehicles Symposium Proceedings*, pp. 510–515, IEEE, 2014.

[33] D. H. Kochanek and R. H. Bartels, "Interpolating splines with local tension, continuity, and bias control," in *Proceedings of the 11th annual conference on Computer graphics and interactive techniques*, pp. 33–41, 1984.

[34] K. Solovey, L. Janson, E. Schmerling, E. Frazzoli, and M. Pavone, "Revisiting the asymptotic optimality of rrt," in *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2189–2195, IEEE, 2020.

[35] G. Ramalingam and T. Reps, "An incremental algorithm for a generalization of the shortest-path problem," *Journal of Algorithms*, vol. 21, no. 2, pp. 267–305, 1996.

[36] S. Koenig, M. Likhachev, and D. Furcy, "Lifelong planning a*," *Artificial Intelligence*, vol. 155, no. 1-2, pp. 93–146, 2004.

[37] O. Arslan and P. Tsiotras, "The role of vertex consistency in sampling-based algorithms for optimal motion planning," *arXiv preprint arXiv:1204.6453*, 2012.

[38] W. Kabsch, "A solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 32, no. 5, pp. 922–923, 1976.

[39] H. Sun and M. Farooq, "Note on the generation of random points uniformly distributed in hyper-ellipsoids," in *Proceedings of the Fifth International Conference on Information Fusion. FUSION 2002.(IEEE Cat. No. 02EX5997)*, vol. 1, pp. 489–496, IEEE, 2002.

[40] G. Wahba, "A least squares estimate of satellite attitude," *SIAM review*, vol. 7, no. 3, pp. 409–409, 1965.

[41] A. H. de Ruiter and J. R. Forbes, "On the solution ofwahba's problem on so (n)," *The Journal of the Astronautical Sciences*, vol. 60, pp. 1–31, 2013.

[42] W. Kabsch, "A discussion of the solution for the best rotation to relate two sets of vectors," *Acta Crystallographica Section A: Crystal Physics, Diffraction, Theoretical and General Crystallography*, vol. 34, no. 5, pp. 827–828, 1978.

[43] J. D. Gammell, T. D. Barfoot, and S. S. Srinivasa, "Informed sampling for asymptotically optimal path planning," *IEEE Transactions on Robotics*, vol. 34, no. 4, pp. 966–984, 2018.

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

[45] K. Yang, S. Moon, S. Yoo, J. Kang, N. L. Doh, H. B. Kim, and S. Joo, "Spline-based rrt path planner for non-holonomic robots," *Journal of Intelligent & Robotic Systems*, vol. 73, no. 1-4, pp. 763–782, 2014.

[46] K. Yang and S. Sukkarieh, "An analytical continuous-curvature path-smoothing algorithm," *IEEE Transactions on Robotics*, vol. 26, no. 3, pp. 561–568, 2010.

[47] B. A. Barsky and T. D. DeRose, "Geometric continuity of parametric curves: three equivalent characterizations," *IEEE Computer Graphics and Applications*, vol. 9, no. 6, pp. 60–69, 1989.

[48] W. Cai, M. Zhang, and Y. R. Zheng, "Task assignment and path planning for multiple autonomous underwater vehicles using 3d dubins curves," *Sensors*, vol. 17, no. 7, p. 1607, 2017.

[49] G. Vailland, V. Gouranton, and M. Babel, "Cubic bézier local path planner for non-holonomic feasible and comfortable path generation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 7894–7900, IEEE, 2021.

[50] ROS, “Navigation stack.” `http://wiki.ros.org/navigation`, 2023. Accessed: 2023-05-25.

[51] ROS, “Setup and configuration of the navigation stack on a robot.” `http://wiki.ros.org/navigation/Tutorials/RobotSetup`, 2023. Accessed: 2023-05-25.

Appendix A

Author's Publication List

Peer-Reviewed

Z. Fei and Y.J. Pan, "A Parameterized Cubic Bézier Spline-based Informed RRT* for Non-holonomic Path Planning," in *IEEE/ASME International Conference on Advanced Intelligent Mechatronics*, Seattle, USA, 2023, pp. 1267-1272.