


Article

Efficient Path Planning Based on Dynamic Bridging Rapidly Exploring Random Tree

Shulei Qiu ^{1,*}, Baoquan Li ¹, Ruiyang Tong ¹, Xiaojing He ¹  and Chuanjing Tang ²

¹ School of Control Science and Engineering, Tiangong University, Tianjin 300387, China; libq@tiangong.edu.cn (B.L.); tong166159@163.com (R.T.); 2130081007@tiangong.edu.cn (X.H.)

² School of Computer Science and Communication Engineering, Jiangsu University, Zhenjiang 212013, China; 2222008055@stmail.ujs.edu.cn

* Correspondence: qslqer@163.com; Tel.: +86-1886-201-3989

Abstract: In the domain of mobile robotic navigation, the real-time generation of low-cost, executable reference trajectories is crucial. This paper propounds an innovative path planning strategy, termed Dynamic Bridging Rapidly Exploring Random Tree (DBR-RRT), which endeavors to enable safe and expedited path navigation. Initially, a heuristic discrimination method is engaged in the path search phase, whereby the issue of sluggish search velocity is tackled by evaluating whether sampled points reside at “bridging locations” within a free space, and by assessing the spatial–geometric relationships between proximate obstacles and auxiliary points. Subsequently, by leveraging extended speed, additional sampling points are generated in the vicinity of existing points to augment the search’s efficacy. Ultimately, the path is optimized and pruned by synthesizing the local curvature of the sampling points and the proximity to obstacles, assigning varied priorities to nodes, thus ensuring that the path’s quality and smoothness is upheld.

Keywords: RRT; path planning; dynamic sampling; path trimming; mobile robot



Citation: Qiu, S.; Li, B.; Tong, R.; He, X.; Tang, C. Efficient Path Planning Based on Dynamic Bridging Rapidly Exploring Random Tree. *Appl. Sci.* **2024**, *14*, 2032. <https://doi.org/10.3390/app14052032>

Academic Editor: Jonghoek Kim

Received: 2 February 2024

Revised: 22 February 2024

Accepted: 27 February 2024

Published: 29 February 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Mobile robots have gained widespread application in fields such as industrial production, transportation, and the service industry, becoming a focal point in current technological research [1,2]. Path planning refers to the process by which robots navigate autonomously based on environmental perception through their sensors [3,4], and its algorithm design and optimization cannot be separated from the effective deployment of sensors [5]. The autonomous navigation of mobile robots is an intelligent system encompassing technologies such as environmental perception, path planning, and navigation positioning [6,7]. Path planning is a core research issue within this system, with its goal being to find a path from a starting point to a destination while ensuring that the robot avoids collisions with obstacles in the environment. Moreover, the path should meet one or more criteria, such as shortest distance, least time, or minimum energy consumption [8], ensuring the efficiency and reliability of pathways is also critical to achieving a highly automated urban transportation network [9]. Over the past few decades, numerous path planning algorithms have been proposed and widely applied in various fields, including, but not limited to, autonomous driving, drone navigation, and industrial robots. The study of path planning problems was initially carried out based on a grid-based approach, which first divides the environmental map into a series of grid cells. For example, graph search algorithms like A* [10,11] and D* [12], while theoretically robust and offering optimal solutions at a consistent resolution, are hampered by a significant drawback due to the challenges in selecting an appropriate a priori resolution. When the resolution is set too low, the quality of the resultant paths tends to be suboptimal. Conversely, setting a high-resolution leads to an exponential increase in computational costs, especially in the context of high-dimensional space problems, necessitating prolonged processing times. Another issue is

their lack of real-time efficiency, as these algorithms require re-graphing and replanning, resulting in reduced efficiency. Optimization-based methods, such as artificial potential field methods [13], are widely used due to their ease of implementation but may fall into local minima.

With the advancement of technology, computing power has been greatly improved and more complex algorithms have been developed, so geometry-based methods have begun to be studied and applied. For example, the Voronoi diagram [14]. The Voronoi diagram divides the space into a series of polygonal regions; each region contains a generating point, and the distance from any point in the region to the generating point is less than the distance to other generating points. The Voronoi diagram guides the mobile robot to avoid obstacles by generating paths within a safe distance from the edges of obstacles, but in highly complex environments, the Voronoi diagram may generate too many Voronoi diagrams, and the generated paths tend to bypass the midpoint of the obstacle, making it difficult to find an optimal solution. This is especially true in environments with many small obstacles or obstacles with complex shapes.

Subsequently, sampling-based path planning methods have received a lot of attention. Sample-based planning methods, due to their ability to provide feasible paths for robots in the short term, have garnered considerable interest. For instance, the Rapidly Exploring Random Tree (RRT) [15] method consistently samples independent states in the search space and incrementally extends towards the goal state, offering the robot a path around obstacles. Another approach, RRT-Connect, proposed by Klemm et al. [16], improves the speed of finding feasible paths by expanding two search trees simultaneously from the starting point and the endpoint. However, although RRT exhibits good computational efficiency, the path it produces is often not optimal, causing the robot to take more time to reach the target location. In [17], Kang et al. proposed an improved RRT-Connect algorithm, which enhances the efficiency of the traditional RRT algorithm by adopting a rewire-tree method based on the triangle inequality, effectively reducing the redundancy commonly seen in paths generated by the standard RRT algorithm. Similarly, in [18], Karaman et al. introduced the improved RRT* algorithm. By updating the tree through reselecting parent nodes, it can find suboptimal paths. However, this approach is characterized by high computational costs and lower efficiency.

There are also artificial intelligence-based path planning techniques such as the ant colony algorithm [19], or the use of machine learning [20] and deep learning [21] models to predict and optimize paths. The ant colony algorithm is computationally intensive and has a slow convergence rate, although it can obtain a global optimal solution. Machine learning and deep learning approaches learn strategies for path planning from large amounts of data, and while they perform well in some scenarios, as shown Wang et al. in [22], who proposed a convolutional neural network (CNN)-based path planning algorithm that trains a model of a CNN network capable of non-uniform sampling, they typically require large amounts of training data and have poorly interpreted models.

2. Related Work

To address these challenges, researchers have gradually introduced more sophisticated strategies. In [23], Kun et al. employed the strategy of the directed expansion of new nodes, making the algorithm more efficient, combined with curvature constraints for path smoothing. However, this method's drawback is the excessive path search time. Other researchers also offered their solutions. Based on the RRT method, in [24], Karaman et al. introduced a rewire process and path cost function to improve the nodes in the existing search tree and their connections. This adjustment made RRT achieve an asymptotically optimal performance, but the process of finding the optimal path can greatly increase the time required for path planning. Gammell et al. proposed the improved algorithm Informed-RRT* in [25], which utilizes an elliptical heuristic search for optimal sampling. This approach addresses the strong randomness inherent in the traditional RRT algorithm. However, it tends to encounter local extremum issues in complex scenarios. Subsequently,

in [26], Gammell et al. introduced Bi-RRT*, which constrains the sampling range and employs dual trees between the start and goal points to communicate information, thereby further enhancing the overall convergence speed, but this method still suffers from the problems of a low success rate for path planning under small samples, and the planning efficiency still needs to be improved under large samples. In [27], Yin effectively reduces the number of mobility simulations required for path planning by combining the RRT algorithm with adaptive surrogate modeling. Using the surrogate model to guide the exploration of random trees under mobility reliability constraints, and then employing these exploration trees and reliability assessments to refine the surrogate model. However, a high dependence on the accuracy of the agent model can generate problems such as, for example, insufficient preparation of the initial agent model or the inability to update it efficiently during the exploration process. Islam et al. proposed RRT*-Smart in [28], which, after generating the initial path, improves RRT*'s convergence speed by removing redundant nodes and optimizing sampling. In [29], Dai proposed a novel algorithm based on bidirectional Rapidly Exploring Random Trees and direct connection. Initially, an expansion strategy based on artificial potential fields was designed in the joint space, which was then integrated with the GB-RRT algorithm. Additionally, a direct connection strategy was developed to enhance the efficiency of expansion, ensuring larger safety margins between the obstacles and the system.

In specific environments such as narrow spaces or areas with dense obstacles issues like difficulty in node expansion and generation of convoluted paths arise. Ji proposed the Ellipsoidal Rapidly exploring Random Tree (E-RRT*) in [30]. This approach replaces line segments with ellipsoids to connect adjacent nodes and introduces a slow informed guidance method to optimize the sampling process, effectively addressing path planning challenges in confined spaces. In [31], Zhang made distinct optimizations to the sampling and proposed a flow-based VF-RRT* algorithm. This method quantifies the process of feasible path countercurrent using a parameter called the up-flow coefficient, constructs a heuristic space based on the flow function, and adaptively rejects sampling nodes outside the flow value range with an adjustable probability.

Another noteworthy method is the Fast-Marching Tree (FMT*) method [32], which combines probabilistic roadmap and RRT approaches by using a set + of sample points for tree expansion. However, the FMT method has the particular issue of redundant exploration, which leads to a decline in its path-searching performance. To solve this problem, Wu Zhen et al. proposed a direction-selective heuristic function that can evaluate the cost gradient of the samples, adjust the ordering of samples, and guide the expansion of FMT. Furthermore, in [33], the Safe Tunnel-based FMT* (ST-FMT*) method generates a preprocessed initial path before method expansion and then constructs a safe tunnel for sampling, which accelerates the convergence speed of the algorithm. Among the sampling-based path planning algorithms, the RRT planning method with a better search capability is suitable for use in complex environments, and the method only requires real-time local data to dynamically construct the search tree. However, RRT node expansion slows down its search efficiency due to the large number of collision detections required. In addition, RRT-generated paths are often redundant and convoluted, making them difficult for mobile robots to execute.

To address these issues, an efficient path planning method based on Dynamic Bridging RRT (DBR-RRT) is proposed, aiming to optimize the efficiency of the RRT-Connect algorithm, and, at the same time, to reduce the speed mutation phenomenon that occurs in the rapid operation of the mobile robot, and to achieve a good balance between the speed and the smoothness of the operation. The main contributions are as follows:

1. Firstly, a heuristic discrimination method is used in path planning to solve the problem of a slow search speed due to node extension collision detection by evaluating whether the sampling points are in free space and assessing the spatial location relationship between neighboring obstacles and setup assistance points.

2. Then, for the extended search phase, the sampling points with the slowest extension speed are searched for, and the set of sampling points is generated in their vicinity to improve the search efficiency.
3. Finally, for the path optimization and pruning phases, different priorities are assigned to the nodes by comprehensively evaluating the sampling point connectivity and proximity to obstacles, which ultimately makes the generated path smoother and easier to execute.

3. Problem Formulation

Despite significant advancements in path planning algorithms, they still encounter critical challenges. A primary issue is that tree expansion near obstacles involves extensive collision detection, which significantly hampers search efficiency. Additionally, the generated paths often lack smoothness and tend to be excessively tortuous. This shortcoming contradicts the fundamental principles of vehicle kinematics. Consequently, the pressing question is how to enhance path planning algorithms to meet the stringent requirements of robot kinematics while addressing these inefficiencies and ensuring smoother trajectories. This paper will delve deeply into this issue with the aim of balancing the generation of efficient paths as well as the stability of robot operation.

The structure of this paper is as follows: the remaining part of this section reviews the existing path planning methods and their advantages and disadvantages; the fourth part details our method and its implementation process; the fifth part presents our experimental results and compares the performance of our method with other methods; finally, the sixth part summarizes the main contributions of this paper and discusses future research directions.

3.1. Traditional Framework

This section will briefly review the RRT-Connect path planning method. The RRT-Connect algorithm, a bidirectional version of the RRT algorithm, does not require pre-sampling and storage of the entire configuration space, but explores the entire state space through function iteration.

The random tree growth process of RRT-Connect is shown in Figure 1, with the specific steps as follows:

1. Initialize the starting point q_{start} , the endpoint q_{goal} , and obstacles, and initialize the starting search tree T_{start} and the ending search tree T_{goal} , as well as the step length step.
2. A randomly selected sampling point q_{rand} in space is used to guide the expansion of the random tree. This point needs to satisfy motion constraints and capture collisions with objects in the environment. Then, iterate over all the nodes on the random tree T_{start} , compute the distance between them and the sampling point q_{rand} , and filter the node $q_{nearest1}$ that is closest to that point.
3. The node $q_{nearest1}$ on the random tree T_{start} grows a fixed number of steps in the direction of the sampling point q_{rand} to obtain a new leaf node q_{new1} . If there is no collision with an obstacle during the movement in the direction from the node $q_{nearest1}$ to the new node q_{new1} , the node q_{new1} will be added to the random tree T_{start} . Otherwise, the point will be discarded, and the process will revert to step 2.
4. Then, iterate over all the nodes on the random tree T_{goal} , and calculate their distances to the new node q_{new2} . Perform the same process as in step 3 on the random tree T_{goal} to obtain the new node q_{new2} on it.
5. Using the greedy search algorithm, step 4 is repeated on the random tree T_{goal} . When an obstacle is encountered during the growth of the random tree, the growth is stopped.
6. Then, exchange the two random trees and repeat step 2–step 6. Within the specified number of searches, when q_{new1} and q_{new2} are the same, a path connecting the initial point and the goal point can be obtained, indicating successful path planning.

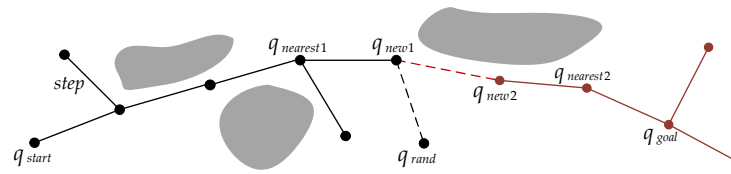


Figure 1. RRT-Connect extension steps. The nodes in T_{start} are displayed as black nodes, while the nodes in T_{goal} are displayed as red nodes.

3.2. Proposed Framework for Path Planning Algorithm

Like the RRT-Connect algorithm, the DBR-RRT algorithm uses two trees, T_{start} and T_{goal} , growing correspondingly from the starting state and target state. What differentiates our new algorithm is the use of heuristic sampling and dynamic sampling strategies.

The DBR-RRT algorithm proposed in this paper is composed of a path search (Bridge Test [34]) strategy. Firstly, DBR-RRT introduces a heuristic method to determine whether a sample point is in the free space “Bridge”. This method evaluates the expansion value of the sampled point based on the relationship between the auxiliary point’s location and obstacles. The tree is only expanded when the Bridge Test concept is passed, to solve the problem of slow search due to extension collision detection. Moreover, DBR-RRT introduces a dynamic sampling strategy based on expansion speed, generating sampling points set around nodes with slower expansion speed. This allows the tree to expand into obstacle-dense areas or areas with complex shapes at a faster rate, thus enhancing the search efficiency. Finally, rapid path smoothing is applied to the path generated by this technique. Through continuous iterations, the original path is optimized to produce a smoother and higher-quality path.

For initialization, two empty trees are initialized and expanded from the start node and goal node respectively. DynamicRandomNode() is used to generate an adaptive random sampling point. This function considers the current structure of the starting tree T_{start} and the goal tree T_{goal} , thereby more intelligently choosing sampling points.

The following is the Pseudocode of the DBR-RRT algorithm (Algorithm 1):

Algorithm 1. DBR-RRT

Notation: start point S , goal point G , obstacle O , vertices and edges of tree

$T_{start}, T_{goal}, V_{start}, E_{start}, V_{goal}, E_{goal}$, random node X_r , nearest node X_{near} , new node X_{new} , connect node X_{con} , path P .

```

1:  $(V_a, E_a) \leftarrow \{S\}, \{\}; V_b = \{G\}; E_b = \{\}$ 
2: while LocalPlanning() do
3:    $X_r \leftarrow \text{GenerateRandomNode}()$ 
4:    $X_{near} \leftarrow \text{FindNearestPoint}()$ 
5:    $X_{new} \leftarrow \text{BridgeTest}()$ 
6:   if  $X_{new} \neq \text{NULL}$  and  $\text{IsFree}(X_{new})$  then
7:      $V_{start} \leftarrow V_{start} \cup \{X_{new}\}; E_{start} \leftarrow E_{start} \cup \{(X_{near}, X_{new})\}$ 
8:     UpdateExpandSpeed()
9:      $X_{con} \leftarrow \text{FindNearestPoint}()$ 
10:    if  $X_{con} \neq \text{NULL}$  then
11:       $(E_{goal}, V_{goal}) \leftarrow \text{ConnectTrees}()$ 
12:      if TreesConnected() then
13:         $P \leftarrow \text{ConstructPath}()$ 
14:        return PathSmoothing()
15:      end if
16:    end if
17:  end if
18: end while
19: if  $V_{goal}.size() < V_{start}.size()$  then
20:   Swap( $V_{start}, V_{goal}$ ); Swap( $E_{start}, E_{goal}$ )
21: end if
22: GenerateExtraSamples()

```

The algorithm will attempt to identify areas in the search space that have not been fully explored (i.e., where node expansion is slow), and generate sampling points in these areas. In addition, if the starting tree and the goal tree are close to each other in space, the function may be more inclined to generate sampling points in the area between the two trees to increase the chance of tree connection.

4. Methodology

In this section, the method proposed in this paper for improving the RRT-Connect algorithm will be detailed, including the improved Bridge Test, dynamic sampling strategy, and path-smoothing techniques. Our goal is to achieve efficient, high-quality path planning.

4.1. Bridge Test

The Bridge Test is a heuristic method used to assess the likelihood of a sampling point being in the “bridge” concept of free space. The core principle of the Bridge Test is based on Voronoi diagram heuristics. Voronoi diagram is a graphical structure that partitions space into several regions, each containing a seed point, and any point within that region is closer to the seed point than to other seed points. In path planning, the Voronoi diagram can help find a path that is as far away from obstacles as possible.

The basic idea of the Bridge Test is to identify sampling points located on the edge of the free space Voronoi diagram [35], as these points are more likely to be part of an excellent path. To achieve this, a geometric analysis method is adopted, combining the spatial relationships between the sampling points, nearby obstacles, and auxiliary test points, to determine whether the sampling points align with the conditions of the Voronoi edge.

The theoretical concept of the Bridge Test is shown in Figure 2. A passage exists from the A direction to B direction. P_c represents a random sampling point on the map. P_d represents the nearest point in the known tree. P_e and P_f represent newly generated test points in two directions. Bridge Test is an algorithmic test method for quickly recognizing and passing through narrow passages (which we call “bridges”) in complex environments. As shown in Figure 2, in the test we follow several steps to evaluate whether a new sampling point (P_c) can help find and enter these narrow passages. The steps in the Bridge Test are as follows:

1. Find the line between the sampled point P_c and the nearest neighbor point P_d in the search tree and calculate angle θ of this line with respect to the horizontal.
2. Depending on the angle θ , the sampling point P_c is offset by a fixed angle in each of the positive and negative directions, thus constructing two predetermined test points P_e and P_f that explore two different directions from P_c .
3. Collision detection is used to verify that the path between the test points, P_c , P_e , P_f is not blocked by obstacles.
4. If the P_c 's path to one of the test points is open (i.e., unobstructed) and the path to the other test points is blocked, this indicates that the P_c is located at the entrance to a potentially narrow channel.
5. P_c is considered to have passed the Bridge Test and will be added to the current search tree.

In Figure 2b, there is an unobstructed path from P_c to P_e , which means that there are no narrow passages obstructing passage in this direction. On the contrary, the path from P_c to P_f partially enters inside the obstacle. This observation suggests that, if the path from P_c to one test point is blocked by an obstacle while the path to another test point is open, it can be determined that the point P_c passed the Bridge Test. By this means of creating handling points, the tree expansion can enter the narrow passage more quickly. This result suggests that the sampling point may be located on the Voronoi boundary of free space, and therefore has a higher expansion priority. During the search process, this priority allocation helps the algorithm to explore those possible effective paths which lead more

efficiently to the target and pushes the nodes into the usually neglected narrow passages with a biasing concept, thus significantly improving search efficiency.

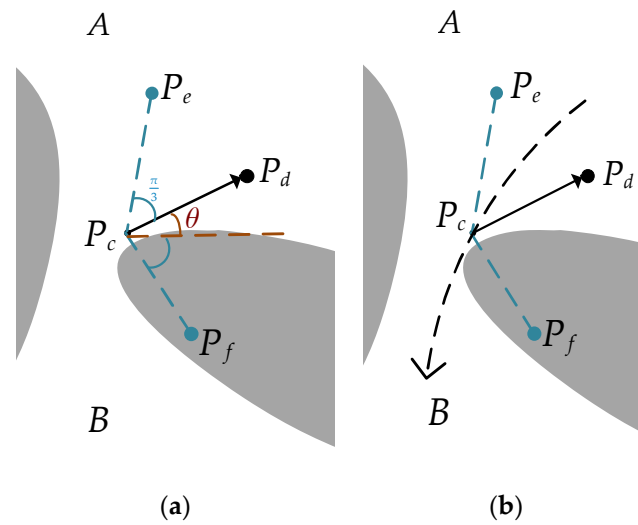


Figure 2. Bridge Test theory: (a) Generate additional sampling points near the sampling points at angle θ (b) Connect and detect any obstacles between the newly generated two paths.

Here is the Pseudocode of the Bridge Test (Algorithm 2):

Algorithm 2. Bridge Test

Notation: sample X_s , angle α , angleOffset α_o , distance d , testPoint1 P_e , testPoint2 P_f .

- 1: $\alpha \leftarrow \text{ComputeAngle}(X_s)$
 - 2: $P_e \leftarrow \text{ComputeTestPoint}(X_s, \alpha + \alpha_o, d)$
 - 3: $P_f \leftarrow \text{ComputeTestPoint}(X_s, \alpha - \alpha_o, d)$
 - 4: **if** $\text{testPass} \leftarrow \text{IsBlocked}(P_e, X_s) \text{ XOR } \text{IsBlocked}(P_f, X_s)$ **then**
 - 5: **return** testPass
 - 6: **else**
 - 7: **return** **false**
 - 8: **end if**
-

The functions `ComputeAngle` and `ComputeTestPoint` have been defined, where the `TestPoint` function is used to generate test points.

`ComputeAngle`: This function calculates the angle between the line from the nearest node to the current sample (X_s) and the horizontal direction. The resulting angle is used as a basis for calculating the test points in subsequent steps.

`ComputeTestPoint`: This function takes the result of `ComputeAngle` and adds and subtracts a given `angleOffset` (α_o) to it to produce two new angles. Using these two new angles and a given distance (d), i.e., the distance from the sampling point to the test point, the two preset test points are calculated.

`IsBlocked`: This function checks if the path from the sample point (X_s) to both test points is blocked by obstacles. This is handled by performing a logical XOR operation on the results of the obstacle detection at both test points, i.e., if only one test point's path is blocked by an obstacle, then the test is considered to have passed (`testPass` is true). This indicates that the sampling point successfully passed the Bridge Test. If the paths to both test points are blocked or neither of them are blocked, then the sampling point is considered to have failed the Bridge Test and the function will return false.

In the fourth step of the pseudo-code, the returned result of the `IsBlocked` function is used to determine whether the test point passes the Bridge Test, and the result of `testPass` is returned in the fifth step.

4.2. Dynamic Sampling Strategy

To overcome the dilemma of the slow expansion speed of the tree structure when the traditional RRT-Connect algorithm deals with high-density environments and areas with many obstacles, an innovative dynamic sampling strategy based on node expansion speed is proposed. In each iteration cycle, the expansion speed of each node is meticulously calculated. This is an insightful metric that accurately reflects the expansion of the search tree in a specific area, which is positively correlated with the priority of the node in the search process. In each iteration, the node with the most lethargic expansion speed $x_{slowest}$ is selected to generate a batch of additional sample points N around it, taking the expansion speed (expandSpeed) as a key evaluation criterion.

This unique strategy significantly reduces the ineffective sampling points caused by random sampling. Particularly when the robot is in a dense environment and trapped in a local minimum, the tree's growth can be steered by tweaking the node generation range, allowing the robot to quickly leave the dangerous area. This strategy not only greatly improves the efficiency of the algorithm, but also enhances its adaptability to complex environments.

It is hypothesized that there is a collision-free path from the start point to the target point. Even if this path is not located in the initial iteration, the algorithm will still select the node with the slowest expansion speed according to the rules (Algorithm 1) and take this node as the center to construct a sector with a certain radius. Within this sector, the algorithm will generate a series of sub-nodes, and then enter the next round of iteration. In Figure 3, yellow nodes represent failure to pass the collision detection, and green nodes represent successful passage. The newly generated nodes in the figure are crucial for connecting the start and target points.

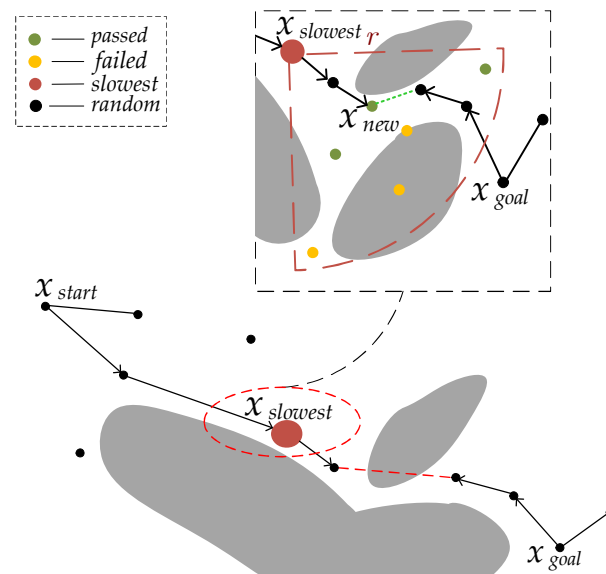


Figure 3. Generate new nodes within the slowest node region.

Generating additional sample points can be regarded as an efficient optimization strategy, which aims to further improve the execution efficiency of the algorithm on an existing basis. In each iteration cycle, the algorithm first tries to execute normal extension steps (Algorithm 1, steps 3–14). Only when the normal extension steps fail to find the path, and the set size of the target point is smaller than the set size of the start point (i.e., the tree range starting from the target point is still relatively small), does the algorithm need to generate additional sample points at the node with the slowest expansion speed to accelerate the expansion of the tree. This implies that, barring the inability to pinpoint unexplored regions, additional sampling will not be conducted, but regular search steps will continue to be executed.

This method achieves effective management of the search range, allowing us to reasonably allocate and optimize unexplored areas while ensuring efficiency, avoiding invalid searches and waste of resources, and further improving the efficiency and effects of path planning. The steps for Dynamic Sampling are as follows (Algorithm 3):

Algorithm 3. Dynamic Sampling

Notation: number of extra samples N_e , the slowest expand node X_{se} , extra samples S_e , tree of samples T_s

```

1: Initialize  $N_e \leftarrow N$ 
2: if  $X_{se} \neq NULL$  then
3:    $N_e \leftarrow \text{StaticCast}(N_e)$ 
4:    $S_e \leftarrow \text{GenerateExtraSamples}(N_e)$ 
5:   for each sample  $X$  in  $S_e$  do
6:     if  $\neg \text{IsAnyObstacleInPath}(X)$  then
7:       Insert  $X$  into  $T_s$ 
8:     end if
9:   end for
10: else
11: skip to the next iteration
12: end if

```

Initialize $N_e \leftarrow N$: The number of dynamic sampling points is first set to N . This is the initial setting for dynamic sampling, preparing for subsequent additional sampling.

GenerateExtraSample: The function verifies if a node with the slowest expansion speed exists (perhaps in an insufficiently explored area), if there is such a node, extra sampling points N are produced at this node. This step achieves adding more sampling points in unexplored areas, thereby improving the depth and breadth of the search.

isAnyObstacleInPath: these extra sampling points are processed individually. For each sample point, an initial check determines if obstacles exist in the path from the slowest expanding node to this sample point. If the path is clear (i.e., there are no obstacles), this sample point is added to the search tree. This step ensures that the extra sampling points generated have relevance, meaning they can be directly reached from the current node. (Algorithm 1. 5–9)

If there is no node with the slowest expansion speed, the current iteration is skipped, moving to the subsequent iteration. This means that, without a clear determination of unexplored areas, no additional sampling will be carried out, but regular search steps will continue to be performed. This step is carried out to ensure the robustness of the algorithm, which can function normally even in special circumstances. (Algorithm 1. 11–15)

While there is a need to spend time calculating the expansion speed of existing nodes in the tree to find the node with the slowest expansion speed, this also allows us to concentrate our resources on the most promising nodes. With this strategy, our tree expansion strategy can effectively avoid falling into local optimal solutions, and, to some extent, this also accelerates the search process.

In addition, this adaptive optimization method aims to dynamically adjust the sampling strategy according to the information obtained during the search process and the current environment, to improve the efficiency of the path planning algorithm in complex environments and improve the quality of the generated path. This is an optimized allocation of computational resources, aiming to maximize the possibility of global optimization, reduce search time, and provide strong support for robot navigation in complex environments. It helps to improve the efficiency of the path planning algorithm in complex environments and the quality of the generated path. This strategy can adaptively adjust the sampling distribution, making the search process more focused on difficult-to-expand areas, thereby improving the efficiency and quality of path planning given the updated node expansion speeds in each iteration, and ensuring that the search process always focuses on the most challenging area currently.

4.3. Path Smoothing

To solve the problems of redundant turns and frequent oscillations in path planning, a prominent path-smoothing strategy has been incorporated. This strategy uses an iterative optimization method to gradually optimize the original path through continuous iterations. This phase weights both local geometric contains and overarching path optimization indicators.

In the path-smoothing procedure, two non-adjacent nodes from the original path nodes are chosen. An attempt is then made to directly connect these two nodes to construct a potential new path to replace the path segment between these two nodes in the original path. This procedure must ensure that the fresh path segment does not intersect with any obstacles, preserving the path's validity. Only when the new path segment meets this criterion is it approved to replace the original path section.

Path smoothing is a key part of this research, aiming to improve the initial path generated by the search algorithm, making it more continuous, smooth, and efficient while meeting the requirements of feasibility and safety. This method helps to optimize the algorithm by directly connecting non-adjacent nodes, eliminating redundant bends in the original path, and finally generating a relatively smooth and high-quality path. The process continues until it reaches a predetermined number of iterations or the path converges. This method aims to gradually achieve global path optimization by adjusting the non-optimized parts of the path, thereby improving the navigation efficiency of robots in complex environments.

4.3.1. Curvature Calculation

It is known that the tangent vector or normal deflection angle of the curve's start and end equals the integral of the curvature length. For the polyline in Figure 4 below, the normal vector only changes at each vertex, so our goal is to find out the curvature at each vertex, but the change in the normal vector at the vertex jumps, as shown in Figure 4; the gray arrow denotes the discernible change in the vertex on the line segment, recording the turning angle of the curve.

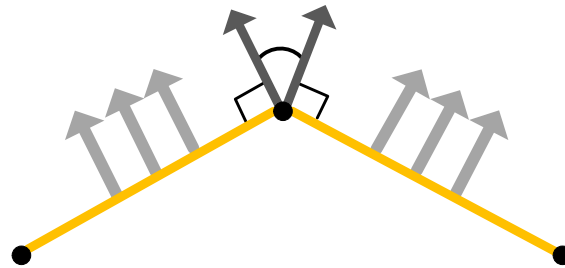


Figure 4. Normal changes at vertices record the turning angle.

The path-smoothing strategy employs a local optimization-based approach to recalibrate pivotal points in the path, minimizing its curvature. Its initial task is to scrutinize the produced preliminary path and identify possible redundant nodes and discontinuous parts. Eradicating these nodes can diminish the path's complexity while preserving essential nodes that ensure the path's feasibility.

An initial step involves assessing the local curvature between various nodes to generate our x_{new} ; if the local curvature is below our preset value, then the node is deemed promising. The curvature at the vertices of a folded line is approximated by applying the Gauss–Bonnet theorem. For a simple closed curve in the plane, the total curvature and the sum of the angles at all vertices are equal to 2π . Applying this principle to the folded line model, the folded line is regarded as a polygon consisting of straight-line segments, with a corner α_i defined at each vertex, which can be found by calculating the angle between the vectors of the two neighboring sides.

The length of its arc is 0, a drawing from the Gauss-Bonnet theorem, the simple loop formed by curves on the plane satisfies the equation:

$$\sum_0^k \int_i^{i+1} k ds + \sum_0^k \theta_i = 2\pi \tag{1}$$

where, k is the curvature function and θ_i is the angle of turn at vertex i . For a single simple closed curve, it is:

$$\oint k ds = 2\pi \tag{2}$$

If this simple closed curve is discretized into a polyline, it still satisfies the Gauss-Bonnet theorem, that is, the sum of the turning angles of all vertices is 2π :

$$\sum_1^n \alpha_i = 2\pi \tag{3}$$

where α_i is the corner at vertex i , and n is the total number of vertices, which represents the integral of mean curvature to arc length on a segment of the real curve, namely:

$$\alpha_i = \int k_i ds = k_i A_i \tag{4}$$

Among these terms, k_i is the curvature at the vertex, and A_i is the arc length of the curve. Consequently, once the arc length A_i is discerned, k_i can be determined. A reasonable A_i value is the sum of half of the two edges of a vertex, as shown in the following Figure 5:

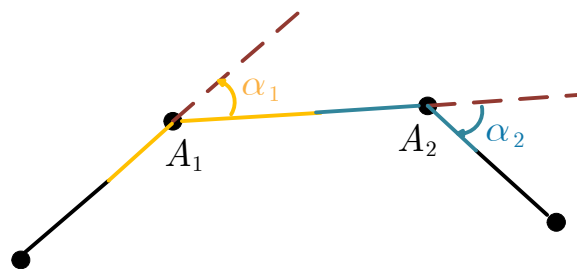


Figure 5. The folded line is formed by joining vertices A_1 and A_2 to form two interior angles α_1 and α_2 .

Leading to a specific result:

$$k_i = \frac{\alpha_i}{A_i} = \frac{2\alpha_i}{l_{i1} + l_{i2}} \tag{5}$$

where, k_i is the local curvature at vertex i , α_i is the angle formed by the two neighboring edges, and A_i is the length of the path from vertex i to the next vertex, i.e., the sum of the halves of the two edges.

4.3.2. Candidate Node Sorting

In this schematic, a succession of green dots marks candidate nodes. Priority is set for these adjacent candidate nodes. Every candidate node undergoes an evaluation, which includes measuring its local curvature and its proximity to the closest obstacle. According to our algorithm, candidate nodes with lower local curvature and further distance from obstacles will be assigned a higher priority.

Moreover, the color of these candidate nodes also symbolizes their priority: the brighter the color, the higher the corresponding priority. Based on this priority, and given this hierarchy, these nodes are prioritized and integrated into the tree structure.

As shown in Figure 6, the newly generated X_{new1} and X_{new2} will be inserted into the original path, providing more possibilities for path planning.

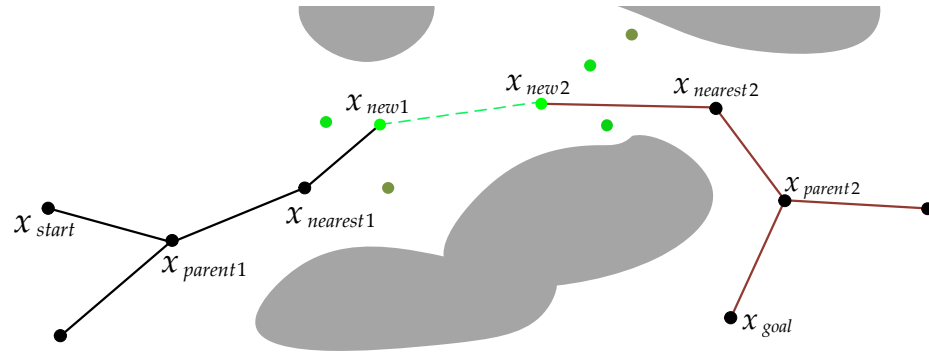


Figure 6. Prioritized selection of candidate nodes. The nodes in both T_{start} and T_{goal} are represented as black nodes, and the green nodes are represented with different brightness according to their distance from the obstacle, the brighter the color, the higher the priority.

For each reachable new node in the path, the angle between adjacent paths is calculated. For T_{start} , the turning angle α at the new node is composed of $X_{new1} \vec{X}_{nearest1}$ and $X_{new1} \vec{X}_{new2}$; For T_{goal} , the turning angle β at the new node is composed of $X_{new2} \vec{X}_{new1}$ and $X_{new2} \vec{X}_{nearest2}$. The formula for calculating the T_{start} vector is shown in (7) and (8), and the formulas for calculating the steering angle α and β are shown in (9) and (10).

$$X_{new1} \vec{X}_{nearest1} = [X_{nearest}(x, y) - X_{new}(x, y)] \tag{6}$$

$$X_{new2} \vec{X}_{nearest2} = [X_{nearest2}(x, y) - X_{new2}(x, y)] \tag{7}$$

$$\alpha = ar \cos \left(\frac{X_{new1} \vec{X}_{nearest1} \cdot X_{new1} \vec{X}_{new2}}{\|X_{new1} \vec{X}_{nearest1}\| \|X_{new1} \vec{X}_{new2}\|} \right) \tag{8}$$

$$\beta = ar \cos \left(\frac{X_{new2} \vec{X}_{new1} \cdot X_{new2} \vec{X}_{nearest2}}{\|X_{new2} \vec{X}_{new1}\| \|X_{new2} \vec{X}_{nearest2}\|} \right) \tag{9}$$

With a specified maximum handover constraint of θ defined as $0 \leq \theta \leq 90^\circ$. For T_{start} , when $\alpha \geq \pi - \theta$, passes the handover constraint, a new node can be injected into T_{start} . If $0 \leq \alpha < \pi - \theta$, the algorithm abandons the new node. Next, enter the extension of T_{goal} , pass through when $\beta \geq \pi - \theta$, and inject the new node into T_{goal} .

Employing this smoothing approach enables the generation of more fluid, superior-quality paths that adhere to the robot’s motion stipulations. This method can not only improve the feasibility of the path, but also reduce the cost of controlling the robot during execution. In the following section, the effectiveness of the proposed method will be showcased across diverse different environments and scenarios. The following is the fast-smoothing pseudocode (Algorithm 4).

Algorithm 4. PathSmoothing

Notation: node N , candidate C , candidate nodes N_c , number of candidate nodes Q_c , curvatureThreshold K .

```

1: for each  $N$  in path
2:   if curvature of node  $> K$ 
3:      $N_c \leftarrow \text{GenerateCandidateNodes}(N, Q_c)$ 
4:     for each  $C$  in  $N_c$ 
5:        $\text{AssignPriority}(C)$ 
6:      $N \leftarrow \text{HighestPriority}$ 
7:   end for
8: end if
9: end for
10: return  $P$ 

```

5. Simulation and Experimental Results**5.1. General Framework of ROS**

The system framework of ROS is divided into three main parts: the file system level, the computational graph level, and the open-source community level, with each part representing a hierarchical concept.

1. File system level: Different components in the ROS program are to be placed in different folders, and each folder also has its corresponding function. Usually we name the workspace `catkin_ws`, which is a folder containing function packages, compiled packages, and compiled executables. A function package is a combination of a specific file structure and a folder containing running nodes, configuration files, etc. We use the command `catkin_make` to compile the workspace. A function package mainly contains the files shown on the rightmost side of Figure 7, in which the `src` is the place where we store the source files, the `build` folder includes the project cache information, configurations and other intermediate files, and the `devel` folder is used to save the compiled program.
2. Computational graph level: The ROS creates a network to connect all of the nodes, through which any node can interact with other nodes, obtain information published by other nodes, and send its own data to the network. The basic concepts at this level include nodes, node managers, parameter servers, messages, services, topics, and message logs.
3. Open-source community level: The open-source community level of the ROS shares software and knowledge mainly through independent online communities, including ROS distributions, software repositories, ROS wiki, ROS Answer, blogs, and so on.

In Figure 8, The core of our navigation architecture is the `move_base` node, which fully integrates global path planning, local path planning, global cost maps, and local cost map features. These different functions are jointly implemented by subscribing to `/tf/odom`, `/map` topics and publishing the `/cmd_vel` topic for motion control under specific conditions. Figure 8 shows that `move_base` is the core of robot navigation, providing an interface for the ROS for configuration and the operation of and interaction with navigation. The robot encounters obstacles in the environment and recalculates the path by subscribing to data from LiDAR, map information, and Monte Carlo localization to convert the path into robot velocity information and plan a new path. Figure 8 shows a diagram of the navigation function architecture which can be found on the left side of the `tf` information through the ROS navigation and localization module. The command `amcl` releases the robot pose for use with `move_base`; below it, `odom` is the robot's odometer information. The upper right corner shows the `map_server` which operates through the analysis of the slam built maps released; the mainstream slam algorithms are `gmapping`, `hector`, and `cartographer`. The sensor in the lower right corner plays a role in local path planning. In this framework, the DWA algorithm serves as the tool for local path planning, while the DBR-RRT algorithm is utilized for global path planning.

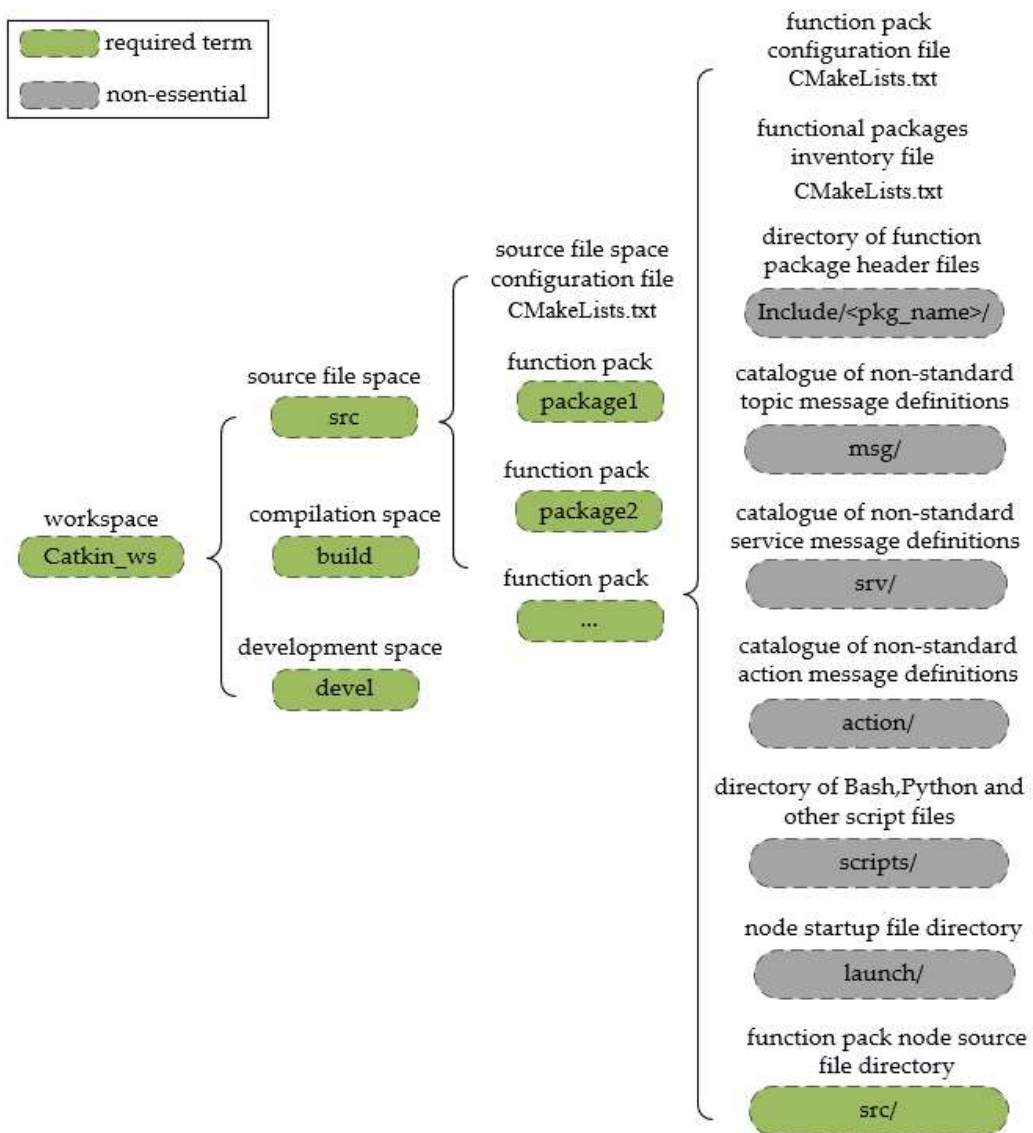


Figure 7. File system-level architecture for ROS.

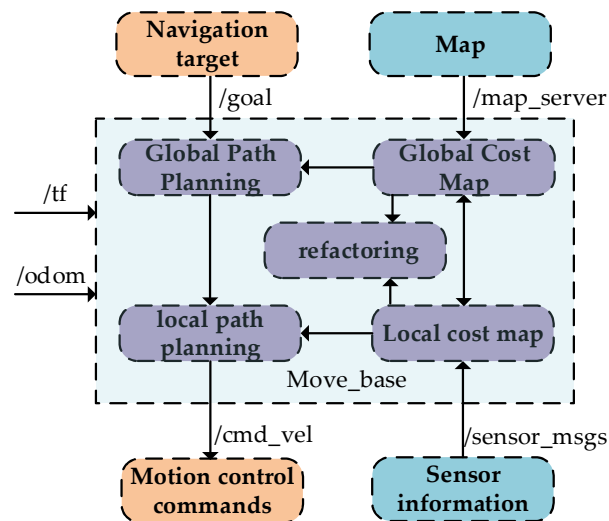


Figure 8. Navigation Framework.

5.2. Simulation I

To validate the simulation experiment, a series of simulation experiments were conducted using the above-mentioned navigation architecture on the TurtleBot3 Waffle mobile robot platform. These experiments aim to verify whether the robot can still maintain a fast path planning speed and robustness in different obstacle environments. Simulation experiments were crafted based on the Ubuntu 20.04-ROS-Neotic system, with the establishment of a densely populated obstacle environment to simulate point-to-point navigation tasks. In simulation I, the simulation environment was 13.2×10.8 m in size, and two rectangular obstacles of $4.8 \times 1.2 \times 2$ m and an L-shaped obstacle composed of two rectangular obstacles were placed, with dimensions of $8.4 \times 0.2 \times 2$ m and $1.8 \times 0.3 \times 2$ m, respectively. There were also a number of irregularly shaped obstacles in the environment. We used a TurtleBot3 Waffle Pi robot with dimensions of $281 \times 306 \times 141$ mm, $v_{\max} = 0.26$ m/s, and $w_{\max} = 1.82$ rad/s. The model of the laser ranging sensor used is LDS-01; the scanning range of the laser radar is set to 3.0 m, the expansion radius is set to 1.0 m, and the cost scaling factor is set to 3.0. By comparing the traditional RRT-Connect algorithm with our improved algorithm, and recording the speed curve, motion trajectory, and time consumption of the mobile robot, a deeper understanding of the convergence speed and robustness of our improved algorithm emerges. The simulation experiment platform used was Gazebo, as shown in Figure 9a,b.

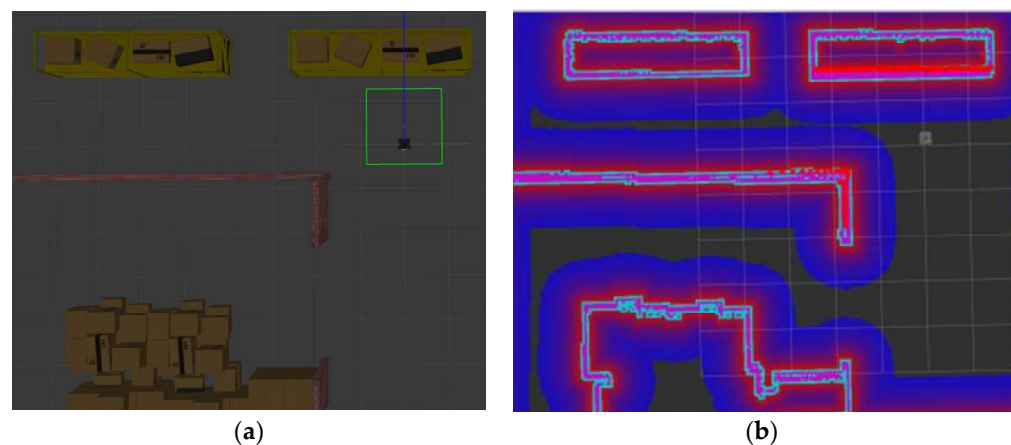


Figure 9. Under the L-shaped obstacle environment. (a) Gazebo environment in simulation I; (b) RVIZ environment in simulation I.

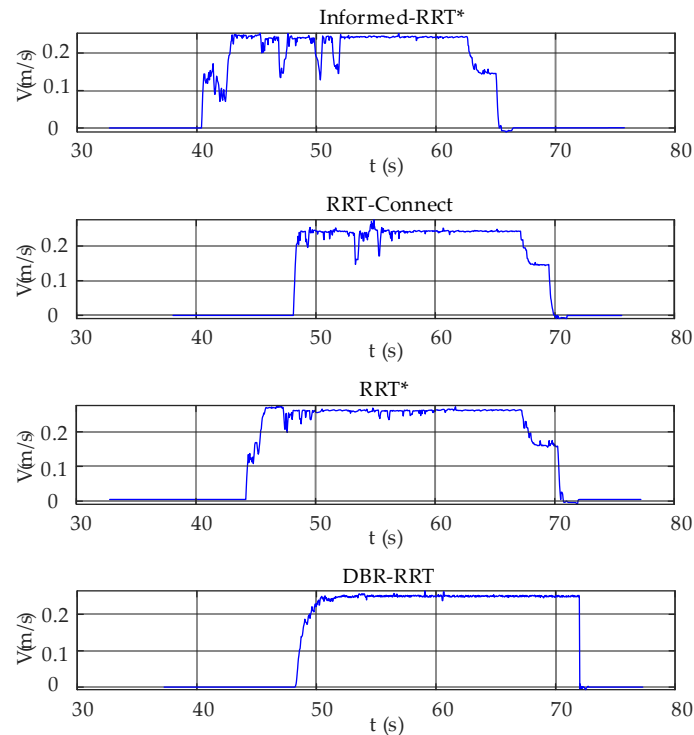
Figure 9 shows that the physical platform is built in Gazebo; the map was constructed by the Gmapping algorithm and visualized using Rviz to simulate a real environment with different shapes of obstacles in the scene, forming a large concave obstacle. The robot had to bypass the obstacles and move to the target location in the lower left corner.

Pre-designed scripts were employed to establish and publish the target coordinates of the mobile robot, replacing the process of manually selecting the target point through the Rviz visualization tool. This ensures the consistency of the target point's position in each experiment, thereby reducing the error within the experimental results. In this experiment, our script set the target point in the lower left corner of the image. A total of 30 simulation experiments were conducted this time, and the statistical indicators used for the comparative evaluation are as follows:

Four different global path planning algorithms are thoroughly evaluated in Simulation 1: Informed-RRT*, RRT-Connect, RRT*, and the modified DBR-RRT. To ensure the fairness and accuracy of the comparison, all algorithms were run in the same test environment, and their running times and standardized speed deviation calculations are meticulously documented in Table 1, and Figure 10 shows a detailed record of each of the four speed profiles.

Table 1. Comparison between Informed-RRT*, RRT-Connect, RRT*, and proposed method in terms of time, average velocity, maximum velocity, and standard deviation of executed velocities.

Planner	Vel. (m/s)		STD Vel.	Time (s)
	Avg	Max		
Informed-RRT*	0.163	0.253	0.051	25.03
RRT-Connect	0.154	0.276	0.043	21.83
RRT*	0.157	0.256	0.039	24.16
DBR-RRT	0.189	0.301	0.037	22.53

**Figure 10.** Speed curves generated by the Informed-RRT*, RRT-Connect, RRT*, and DBR-RRT algorithms in simulation I.

In Table 1 and Figure 10, it is shown that DBR-RRT shows an improvement of 15.95% in average speed and a 27.45% improvement in speed standard deviation compared to Informed-RRT*. Compared to RRT-Connect, DBR-RRT shows a 22.73% improvement in average speed and a 13.95% improvement in speed standard deviation. In addition, DBR-RRT results in 20.38% improvement in average speed and 5.13% improvement in standard deviation compared to RRT*. These results indicate that the proposed algorithm exhibits a high consistency and low speed fluctuations over multiple runs. In addition, DBR-RRT achieves a maximum speed of 0.301 m/s, which is slightly higher than the 0.276 m/s of RRT-Connect, indicating that the proposed algorithm maintains stability without sacrificing navigation performance.

The experimental results show that the RRT-Connect algorithm takes the least amount of time among all the reference algorithms, this is due to the bidirectional search strategy adopted by the RRT-Connect algorithm, which can quickly discover effective paths from the starting point to the end point. However, its standard deviation is large, and the speed profile shown in Figure 10 is rugged, and the excessively fast search speed sacrifices some stability. In addition, the proposed DBR-RRT algorithm inherits the bidirectional search strategy of RRT-Connect, and although the average running time is slightly longer than that of the initial RRT-Connect algorithm, the speed has a smaller standard deviation and possesses the largest maximum speed, which indicates that the speed fluctuation be-

tween each run is small, and it achieves a balance between increasing the average speed and maintaining less speed fluctuations. Although the RRT algorithm needs to consider more candidate paths in the process of finding the optimal path, and thus consumes the longest time, it is clearly unnecessary to sacrifice time to find the optimal path in a simple L-shaped environment. Compared with DBR-RRT, Informed RRT* can theoretically improve the efficiency of path planning through conducting a more informed search, but in this test environment, its efficiency does not exceed that of the other algorithms, and the large fluctuations in the velocity curves shown in Figure 10. indicate that the algorithm has a large variation in the robot's velocity when planning paths.

5.3. Simulation II

In simulation II, the simulation environment is 12×10.8 m in size, with three rectangular obstacles of $4.8 \times 1.2 \times 2$ m and five circular obstacles with a diameter of 0.5 m. The robot and other environmental parameters are the same as in simulation I. Five smaller obstacles were strategically placed to surround the robot in a dense environment. The robot's task was to move to the bottom right corner of the image while avoiding these obstacles. To better simulate obstacles that may suddenly appear in the real environment, these hindrances were intentionally excluded from grid maps constructed using SLAM technology.

Figure 11 shows a path planning process using the DBR-RRT algorithm. This design makes the robot unaware of these obstacles at the beginning of the initial navigation task. However, when the robot approaches these obstacles, its internal LiDAR can effectively detect them and successfully use our algorithm to plan new paths to avoid these obstacles. Real-time robot planning trajectories were acquired and visualized by subscribing to pertinent topics, offering a clear perspective of the robot's current state. In this experiment, the target point was set in the lower right corner. In this case, the robot had to detour around five "suddenly appearing" obstacles. By incorporating such challenges, the robot was predisposed to encounter obstacles early on, positioning itself in potentially hazardous zones surrounded by impediments. This setting allows us to visually observe and compare the robustness and planning efficiency of the two algorithms.

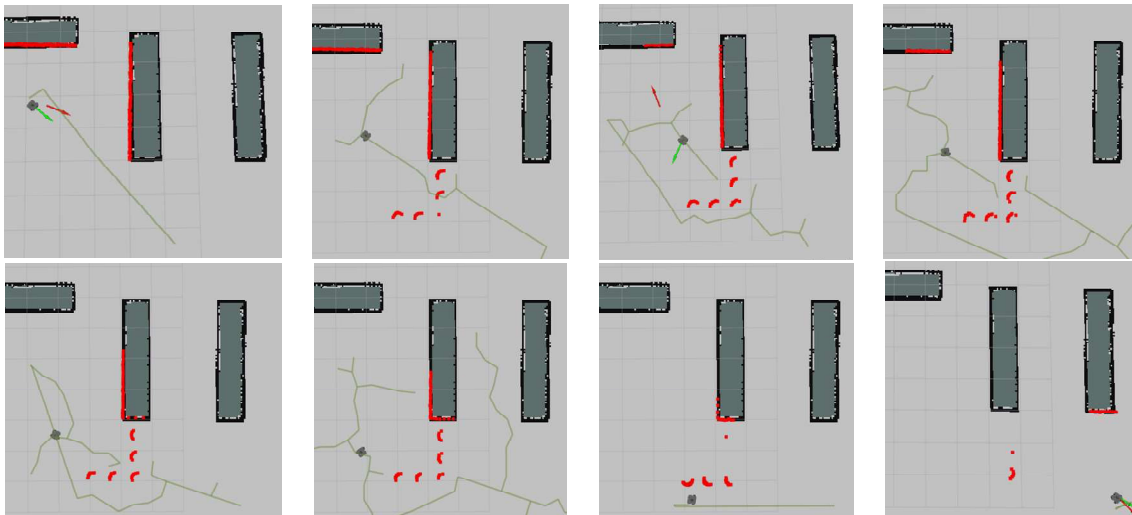


Figure 11. One experiment of DBR-RRT algorithm for path planning.

RGB color mapping was employed to vividly convey the robot's speed information on its travel path. As shown in Figure 12b, the proposed algorithm enabled the robot to have a faster average speed compared to the RT-Connect algorithm, and the speed could be smoothly transitioned during the overall path planning process. In Figure 12a, there are meanders in the robot path and there are multiple instances of sudden velocity mutations. The traditional RRT-Connect algorithm exhibits serious problems when dealing with local optimal solutions, causing the robot to fall into a state of spinning in place. In this case, the

robot’s speed exhibits an extremely unstable acceleration and deceleration process, even approaching zero. Therefore, the repeated rotation of the robot resulted in a significant increase in the overall path planning time.

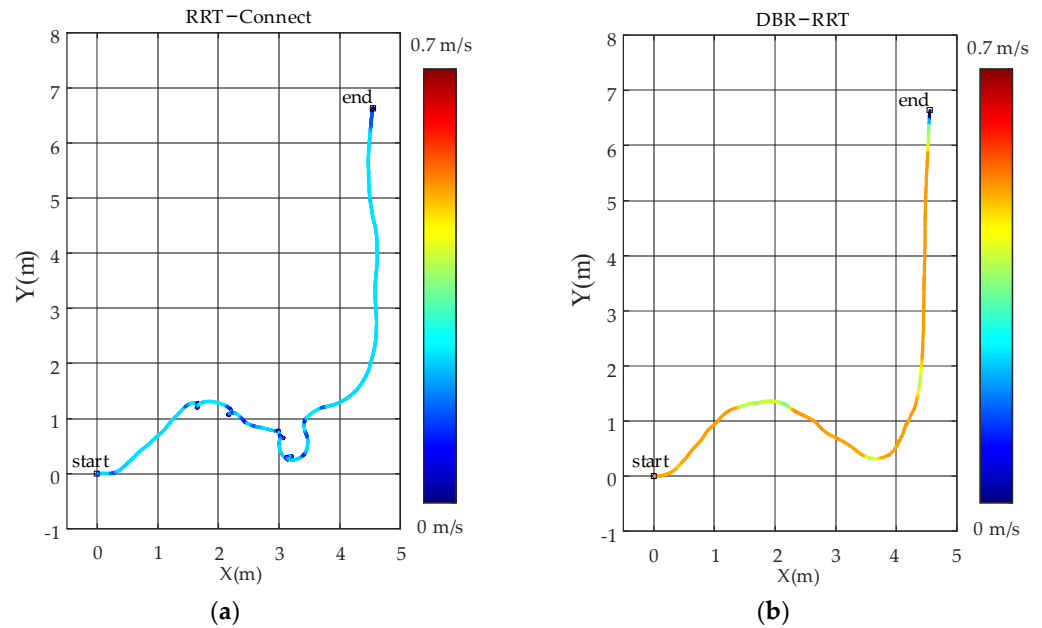


Figure 12. Trajectory velocity curves of two algorithms: (a) RRT-Connect and (b) DBR-RRT.

Figure 13 shows a comparison of the trajectories generated by the two algorithms. The robot using the RRT-Connect algorithm for path planning had three detours in the trajectory from (1.7, 1.3) to (3.5, 1.0), and there was a large slewing behavior at (3.5, 1.0), which greatly increased the path length. For unknown obstacles in a known environment, traditional algorithms perform inefficient mass collision detection and inefficiently generate paths in real time. In contrast, the algorithm proposed in this paper will first perform a Bridging Test for obstacles, select the points surrounded by obstacles, and generate a virtual node set in the iterative process to quickly guide the robot away from the dangerous area surrounded by obstacles.

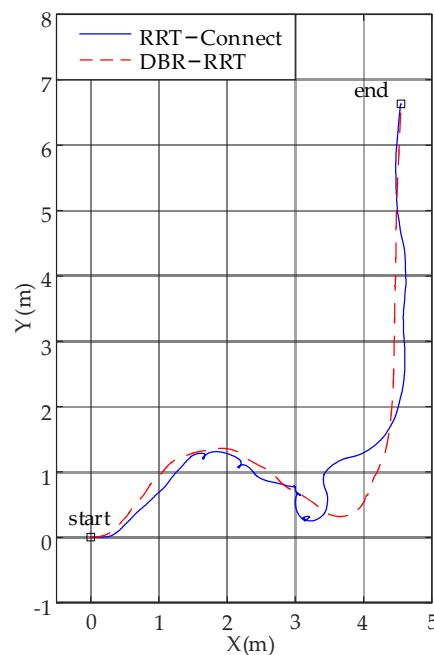


Figure 13. Comparison of trajectories of the two algorithms.

Therefore, the algorithm proposed in this paper performs better in this trap environment and can quickly plan a new path when a sudden obstacle is detected. In addition, the overall planning process maintains a stable speed, allowing for stable acceleration and deceleration even during unavoidable steering processes. As shown in Figures 12 and 13, the robot's movement trajectory is a smooth curve that remains stable after turning, and the final trajectory approaches a straight line.

5.4. Real-World Experiment I

After the navigation architecture was successfully configured, not only was a precise local area network constructed, but a rigorous network configuration for the robot was also implemented. Particular attention was given to ensuring that both the host and the mobile robot operate within the same network segment, prioritizing communication reliability and efficiency. With this robust foundation, the phase of experimental verification commenced. The size of the experimental environment was 4.2×5.2 m. A $281 \times 306 \times 141$ mm TurtleBot3 (Waffle Pi) robot was selected for algorithm experiments. The safety distance $l = 0.25$ m was chosen for the experiment, and the maximum allowable linear and angular velocities of the robot were 0.26 m/s and 1.82 rad/s, respectively, and the peak parameters $v_{\max} = 0.23$ m/s and $w_{\max} = 1.5$ rad/s were set for this experiment.

This experiment was conducted to validate the scenario simulated in Experiment 1: avoiding unknown obstacles in a known map. A physical experimental environment was set up, where black objects represent known obstacles in the known environment, and brown obstacles represent the unknown obstacles. The experimental site contained four black rectangular obstacles of $20 \times 20 \times 30$ cm, two black rectangular obstacles of $26 \times 8 \times 16$ cm, one black rectangular obstacle of $30 \times 13 \times 40$ cm, one black rectangular obstacle of $35 \times 15 \times 30$ cm, and two brown obstacles of $20 \times 35 \times 30$ cm. Additionally, the robot's navigation process was captured in a sequence of 80 frames, documenting the real robot's trajectory.

Figures 14 and 15 show that DBR-RRT algorithm can avoid obstacles with stability and relatively high speed in environments with unknown obstacles. It exhibits fewer sudden changes in speed and maintains stability even in situations requiring turns, while also planning a smooth path.



Figure 14. Avoiding unknown obstacles in a known environment.

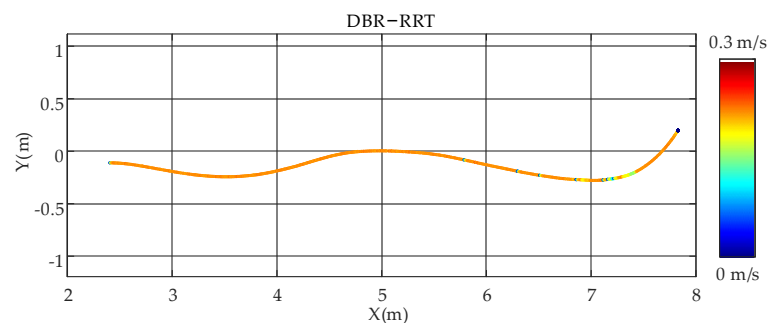


Figure 15. Speed-trajectories diagram for robots in experiments.

5.5. Real-World Experiment II

To test the effectiveness of the proposed algorithm in path planning over long periods of time, an environment full of obstacles was designed. In this experimental site, as shown in Figure 16, four rectangular obstacles of $20 \times 20 \times 30$ cm, one L-shaped obstacle consisting of two long rectangles of $100 \times 6 \times 30$ cm, and two L-shaped obstacles consisting of $85 \times 6 \times 30$ cm and $60 \times 6 \times 30$ cm were placed. The parameters and environmental parameters set by the robot are consistent with Experiment 1. The starting point was anchored in the map's bottom right corner. To enhance the evaluation of the robot's performance in the real environment, goal commands were issued at strategic points on the map to enable the robot to navigate once in the complex environment.



Figure 16. Experiment environment.

The actual mobile robot's going and returning using the two algorithms are shown in Figure 17, respectively. Given the overlap between the robot's outbound and return paths, to prevent any visual confusion within a singular image, the decision was made to separate the actual trajectories of the two path-planning methods. These are represented in two distinct images, showcased through frame extraction and overlay. An optimal selection of 80 frames was made to authentically depict the robot's movement trajectory.

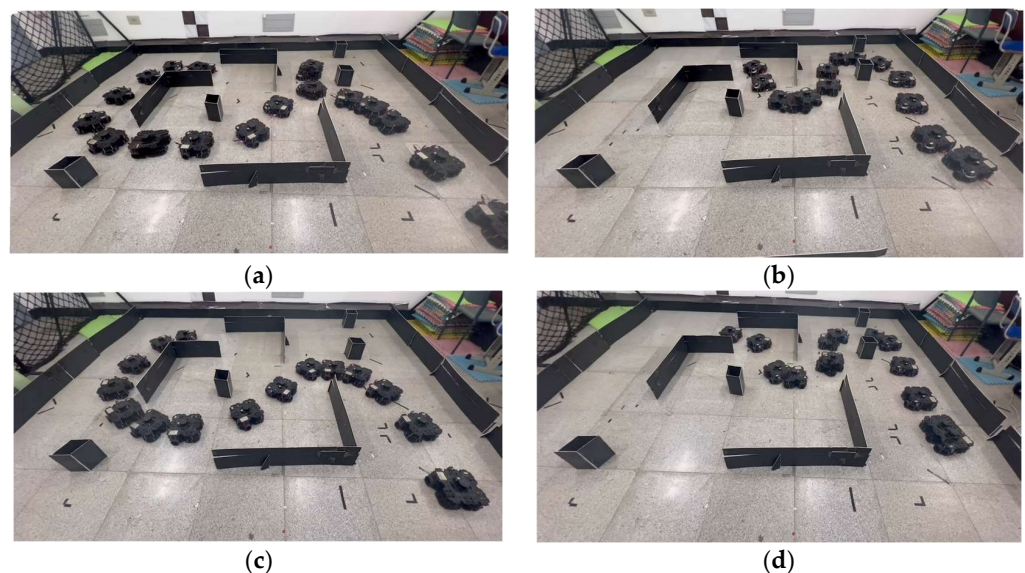


Figure 17. One-time path planning using the RRT-Connect and DBR-RRT algorithm. (a) RRT-Connect to planned outbound; (b) RRT-Connect planned return trip; (c) DBR-RRT to planned outbound; (d) DBR-RRT planned return trip.

In Figure 18, the robot's movement trajectory is displayed on the Rviz visualization platform. The actual motion path of the DBR-RRT robot is represented by green lines, while the actual motion path of the RRT-Connect robot is represented by red lines.

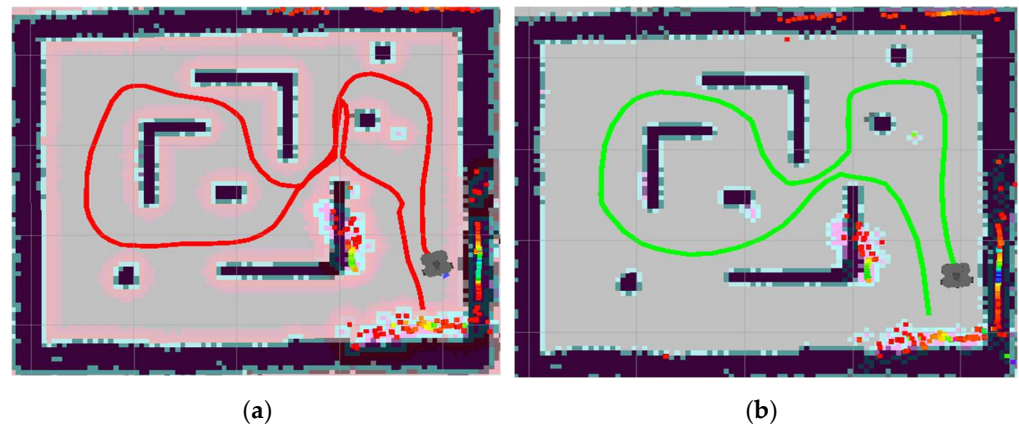


Figure 18. The trajectory generated by the RRT-Connect and DBR-RRT algorithms are displayed in Rviz: (a) RRT-Connect. (b) DBR-RRT.

Upon close examination of the red trajectory, it was discerned that a judgment error occurred at the first turn. Rather than making a left at the intersection, the robot chose to proceed straight, causing an unnecessary path extension. In addition, at the point where the robot is about to enter a turn, its trajectory shows a series of obvious twists and turns, revealing that the path planning is not smooth enough. And being too close to the obstacle in the upper right corner may cause potential safety issues.

As can be seen in Figure 19, after mapping the robot’s speed onto the trajectory for depth comparison, the traditional RRT-Connect algorithm exhibits significant speed fluctuations in multiple key areas. Particularly in areas requiring turns, a marked difference in robot speed was observed. Moreover, on sections devoid of significant turns, the speed presented unstable fluctuation characteristics. The speed parameters used in this experiment are as follows:

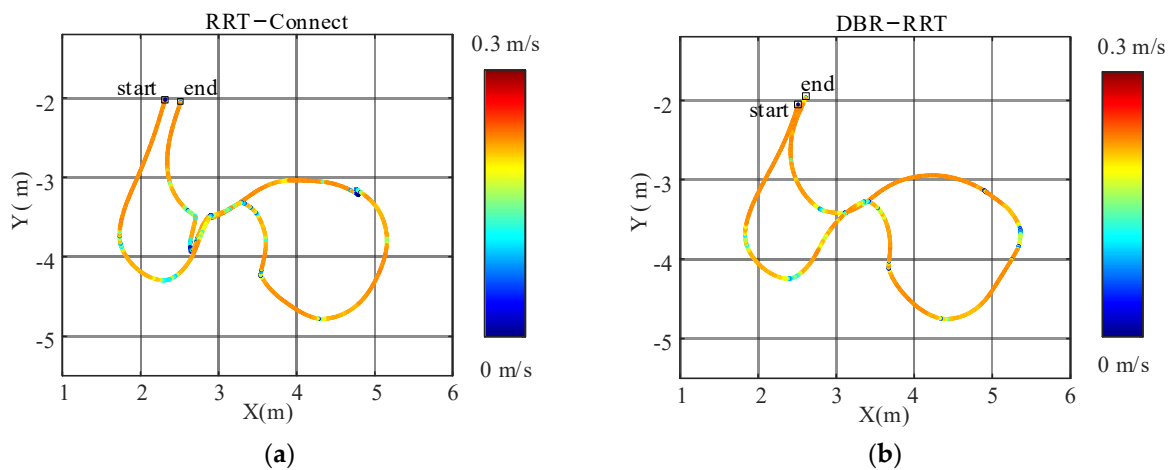


Figure 19. Speed–trajectory curves of RRT–Connect and DBR–RRT. (a) RRT–Connect (b) DBR-RRT.

Following the experimental verification, Table 2 shows that the proposed DBR-RRT shows higher efficiency than the traditional RRT-Connect algorithm. Firstly, its average completion time is 9 s faster than the traditional RRT-Connect algorithm. Secondly, the average running speed of the DBR-RRT algorithm is also higher than that of the traditional algorithms, indicating that the robot can maintain higher operating efficiency throughout the entire operation process. Finally, it is worth noting that the standard deviation of the DBR-RRT algorithm is relatively low, which means that the robot’s operating speed changes are more stable and less susceptible to sudden factors, thus maintaining a good driving performance in different environments and conditions.

Table 2. Comparison between RRT-Connect and proposed method in terms of time, max velocity, average velocity, and standard deviation of executed velocities.

Planner	Times (s)	Max Vel. (m/s)	Avg Vel. (m/s)	STD Vel.
RRT-Connect	106	0.230	0.100	0.103
DBR-RRT	97	0.230	0.139	0.097

6. Conclusions

In this paper, an advanced path-planning method based on Dynamic Bridging RRT was successfully developed and demonstrated, dedicated to addressing the limitations of existing strategies in complex environments and real-time requirements. The research results indicate that, based on the dynamic sampling strategy of expanding speed, key achievements include faster search speeds in regions with dense obstacles or intricate bottleneck shapes. Meanwhile, in the concluding path-generation phase, a swift path-smoothing strategy was adopted, which further optimized the path, producing a more streamlined and high-quality route. This approach makes path planning for mobile robots in complex environments more efficient and is a useful complement to existing path-planning algorithms. However, due to time and resource constraints, we were not able to independently demonstrate the potential improvement effect that the path-smoothing component may have on the RRT-Connect algorithm. This is an important dimension of comparison that will be added in future work. Future research will include applying the smoothing strategy to RRT-Connect individually under the same conditions and performing a careful comparative performance analysis to fully assess the impact of the smoothing approach. Through such work, we hoped to provide deeper insights and further validate the stability and efficiency of the methods, providing a more complete performance evaluation framework.

Author Contributions: Conceptualization, S.Q. and R.T.; methodology, S.Q.; software, S.Q. and R.T.; validation, S.Q. and B.L.; formal analysis, S.Q.; investigation, S.Q., R.T. and C.T.; resources, S.Q., B.L., and X.H.; data curation, S.Q. and R.T.; writing—original draft preparation, S.Q.; writing—review and editing, S.Q., B.L. and C.T.; visualization, S.Q. and C.T.; supervision, B.L.; project administration, S.Q. and B.L.; funding acquisition, B.L.; All authors have read and agreed to the published version of the manuscript.

Funding: This work is supported by the National Natural Science Foundation of China (grant number 61973234 and 62203326), and in part by the Tianjin Natural Science Foundation (grant number 20JCYBJC00180).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The source code presented in this study is available on request from the corresponding author.

Acknowledgments: The authors would like to thank Tiangong University for technical support and all members of our team for their contribution to the mobile robot experiments. The authors acknowledge the anonymous reviewers for their helpful comments on the manuscript.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Gul, F.; Mir, I.; Abualigah, L.; Sumari, P.; Forestiero, A. A Consolidated Review of Path Planning and Optimization Techniques: Technical Perspectives and Future Directions. *Electronics* **2021**, *10*, 2250. [[CrossRef](#)]
- Zhang, H.-y.; Lin, W.-m.; Chen, A.-x. Path Planning for the Mobile Robot: A Review. *Symmetry* **2018**, *10*, 450. [[CrossRef](#)]
- Sánchez-Ibáñez, J.R.; Pérez-del-Pulgar, C.J.; García-Cerezo, A. Path Planning for Autonomous Mobile Robots: A Review. *Sensors* **2021**, *21*, 7898. [[CrossRef](#)] [[PubMed](#)]
- Li, X.; Tong, Y. Path Planning of a Mobile Robot Based on the Improved RRT Algorithm. *Appl. Sci.* **2024**, *14*, 25. [[CrossRef](#)]
- Owais, M. Traffic Sensor Location Problem: Three Decades of Research. *Expert Syst. Appl.* **2022**, *208*, 118134. [[CrossRef](#)]

6. Silva Ortigoza, R.; Marcelino-Aranda, M.; Silva Ortigoza, G.; Hernandez Guzman, V.M.; Molina-Vilchis, M.A. Wheeled Mobile Robots: A Review. *IEEE Lat. Am. Trans.* **2012**, *10*, 2209–2217. [[CrossRef](#)]
7. Gao, X.; Li, J.; Fan, L.; Zhou, Q.; Yin, K.; Wang, J.; Song, C.; Huang, L.; Wang, Z. Review of Wheeled Mobile Robots' Navigation Problems and Application Prospects in Agriculture. *IEEE Access* **2018**, *6*, 49248–49268. [[CrossRef](#)]
8. Khaksar, W.; Vivekananthen, S.; Saharia, K.S.M.; Yousefi, M.; Ismail, F.B. A Review on Mobile Robots Motion Path Planning in Unknown Environments. In Proceedings of the IEEE International Symposium on Robotics and Intelligent Sensors (IRIS), Langkawi, Malaysia, 18–20 October 2015; pp. 295–300.
9. Owais, M.; Alshehri, A. Pareto Optimal Path Generation Algorithm in Stochastic Transportation Networks. *IEEE Access* **2020**, *8*, 58970–58981. [[CrossRef](#)]
10. Dong, G.; Yang, F.; Tsui, K.-L.; Zou, C. Active Balancing of Lithium-Ion Batteries Using Graph Theory and A-Star Search Algorithm. *IEEE Trans. Ind. Inform.* **2021**, *17*, 2587–2599. [[CrossRef](#)]
11. Zhang, H.; Tao, Y.; Zhu, W. Global Path Planning of Unmanned Surface Vehicle Based on Improved A-Star Algorithm. *Sensors* **2023**, *23*, 6647. [[CrossRef](#)] [[PubMed](#)]
12. Wang, S.J.; Hu, L.K.; Wang, Y.F. Path Planning of Indoor Mobile Robot Based on Improved D* Algorithm. *Comput. Eng. Des.* **2020**, *41*, 1118–1124.
13. Warren, C.W. Multiple Robot Path Coordination Using Artificial Potential Fields. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Cincinnati, OH, USA, 13–18 May 1990; pp. 500–505.
14. Wang, J.; Meng, M.Q.-H. Optimal Path Planning Using Generalized Voronoi Graph and Multiple Potential Functions. *IEEE Trans. Ind. Electron.* **2020**, *67*, 10621–10630. [[CrossRef](#)]
15. LaValle, S.M.; Kuffner, J.J., Jr. Randomized Kinodynamic Planning. *Int. J. Robot. Res.* **2001**, *20*, 378–400. [[CrossRef](#)]
16. Kuffner, J.J.; LaValle, S.M. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In Proceedings of the 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation (ICRA), San Francisco, CA, USA, 24–28 April 2000; pp. 995–1001.
17. Kang, J.G.; Lim, D.W.; Choi, Y.S. Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning. *Sensors* **2021**, *21*, 333. [[CrossRef](#)] [[PubMed](#)]
18. Karaman, S.; Walter, M.R.; Perez, A.; Frazzoli, E.; Teller, S. Anytime Motion Planning Using the RRT*. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Shanghai, China, 9–13 May 2011; pp. 1478–1483.
19. Qi, J.; Yang, H.; Sun, H. MOD-RRT*: A Sampling-Based Algorithm for Robot Path Planning in Dynamic Environment. *IEEE Trans. Ind. Electron.* **2021**, *68*, 7244–7251. [[CrossRef](#)]
20. Sarker, I.H. Machine Learning: Algorithms, Real-World Applications and Research Directions. *Comput. Sci.* **2021**, *2*, 160. [[CrossRef](#)]
21. Janiesch, C.; Zschech, P.; Heinrich, K. Machine Learning and Deep Learning. *Electron. Markets* **2021**, *31*, 685–695. [[CrossRef](#)]
22. Wang, J.; Chi, W.; Li, C.; Wang, C.; Meng, M.Q.H. Neural RRT*: Learning-Based Optimal Path Planning. *IEEE Trans. Autom. Sci. Eng.* **2020**, *17*, 1748–1758. [[CrossRef](#)]
23. Kun, W.; Ren, B. A Method on Dynamic Path Planning for Robotic Manipulator Autonomous Obstacle Avoidance Based on an Improved RRT Algorithm. *Sensors* **2018**, *18*, 571–586.
24. Karaman, S.; Frazzoli, E. Sampling-Based Algorithms for Optimal Motion Planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [[CrossRef](#)]
25. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Informed RRT*: Optimal Sampling-Based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Chicago, IL, USA, 14–18 September 2014; pp. 2997–3004.
26. Gammell, J.D.; Srinivasa, S.S.; Barfoot, T.D. Batch Informed Trees (BIT*): Sampling-Based Optimal Planning via the Heuristically Guided Search of Implicit Random Geometric Graphs. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Seattle, WA, USA, 26–30 May 2015; pp. 3067–3074.
27. Yin, J.; Hu, Z.; Mourelatos, Z.P.; Gorsich, D.; Singh, A.; Tau, S. Efficient Reliability-Based Path Planning of Off-Road Autonomous Ground Vehicles Through the Coupling of Surrogate Modeling and RRT*. *IEEE Trans. Intell. Transp. Syst.* **2023**, *24*, 15035–15050. [[CrossRef](#)]
28. Islam, F.; Nasir, J.; Malik, U.; Ayaz, Y.; Hasan, O. RRT*-Smart: Rapid convergence implementation of RRT* towards optimal solution. In Proceedings of the IEEE International Conference on Mechatronics and Automation (ICMA), Chengdu, China, 5–8 August 2012; pp. 1651–1656.
29. Dai, J.; Zhang, Y.; Deng, H. Novel Potential Guided Bidirectional RRT* With Direct Connection Strategy for Path Planning of Redundant Robot Manipulators in Joint Space. *IEEE Trans. Ind. Electron.* **2024**, *71*, 2737–2747. [[CrossRef](#)]
30. Ji, H.; Xie, H.; Wang, C.; Yang, H. E-RRT*: Path Planning for Hyper-Redundant Manipulators. *IEEE Robot. Autom. Lett.* **2023**, *8*, 8128–8135. [[CrossRef](#)]
31. Zhang, W.; Shan, L.; Chang, L.; Dai, Y. SVF-RRT*: A Stream-Based VF-RRT* for USVs Path Planning Considering Ocean Currents. *IEEE Robot. Autom. Lett.* **2023**, *8*, 2413–2420. [[CrossRef](#)]
32. Janson, L.; Schmerling, E.; Clark, A. Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *Int. J. Robot. Res.* **2015**, *34*, 883–921. [[CrossRef](#)] [[PubMed](#)]

33. Wu, Z.; Chen, Y.; Liang, J. ST-FMT*: A fast optimal global motion planning for mobile robot. *IEEE Trans. Ind. Electron.* **2021**, *69*, 3854–3864. [[CrossRef](#)]
34. Lee, J.; Kwon, O.; Zhang, L.; Yoon, S. SR-RRT: Selective Retraction-based RRT Planner. In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Saint Paul, MN, USA, 14–18 May 2012; pp. 2543–2550.
35. Chi, W.; Ding, Z.; Wang, J.; Chen, G.; Sun, L. A Generalized Voronoi Diagram-Based Efficient Heuristic Path Planning Method for RRTs in Mobile Robots. *IEEE Trans. Ind. Electron.* **2022**, *69*, 4926–4937. [[CrossRef](#)]

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.