



## Article

# CERRT: A Mobile Robot Path Planning Algorithm Based on RRT in Complex Environments

Kun Hao, Yang Yang , Zhisheng Li , Yonglei Liu and Xiaofang Zhao

School of Computer and Information Engineering, Tianjin Chengjian University, Tianjin 300384, China; kunhao@tcu.edu.cn (K.H.); yhpl01@163.com (Y.Y.); sanxiong\_1@163.com (Y.L.); zhaoxf@tcu.edu.cn (X.Z.)  
\* Correspondence: lzs@tcu.edu.cn

**Abstract:** In complex environments, path planning for mobile robots faces challenges such as insensitivity to the environment, low efficiency, and poor path quality with the rapidly-exploring random tree (RRT) algorithm. We propose a novel algorithm, the complex environments rapidly-exploring random tree (CERRT), to address these issues. The CERRT algorithm builds upon the RRT approach and incorporates two key components: a pre-allocated extension node method and a vertex death mechanism. These enhancements aim to improve vertex utilization and overcome the problem of becoming trapped in concave regions, a limitation of traditional algorithms. Additionally, the CERRT algorithm integrates environment awareness at collision points, enabling rapid identification and navigation through narrow passages using local simple sampling techniques. We also introduce the bidirectional shrinking optimization strategy (BSOS) based on the pruning optimization strategy (POS) to further enhance the quality of path solutions. Extensive simulations demonstrate that the CERRT algorithm outperforms the RRT and RRV algorithms in various complex environments, such as mazes and narrow passages. It exhibits shorter running times and generates higher-quality paths, making it a promising approach for mobile robot path planning in challenging environments.

**Keywords:** path planning; RRT; path optimization; complex environments



**Citation:** Hao, K.; Yang, Y.; Li, Z.; Liu, Y.; Zhao, X. CERRT: A Mobile Robot Path Planning Algorithm Based on RRT in Complex Environments. *Appl. Sci.* **2023**, *13*, 9666. <https://doi.org/10.3390/app13179666>

Academic Editor: Jonghoek Kim

Received: 4 August 2023

Revised: 25 August 2023

Accepted: 25 August 2023

Published: 26 August 2023



**Copyright:** © 2023 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

Path planning is a crucial research field in the robotics industry. Its purpose is to find a safe, collision-free path for a mobile robot to traverse from its starting position to its destination in a specified area that contains obstacles [1]. It has widespread applications in complex environments such as urban roads, factory production lines, and outdoor exploration. Currently, path planning mainly uses algorithms based on search, heuristics, and sampling. Among them, sampling-based path planning algorithms have become a research hotspot due to their wide applicability, ease of implementation, and lack of need to construct complex structures [2].

In the class of sampling-based algorithms, the rapidly-exploring random tree (RRT) algorithm, which is widely used, can avoid complex space constructions by implementing a collision check module, making it suitable for solving high-dimensional or multi-constraint planning problems [3]. However, the efficiency of the RRT algorithm is typically affected when it faces complex environments such as multiple obstacles, mazes, narrow passages, and concave traps. In recent years many researchers have proposed improved RRT algorithms to address these issues. For instance, Kuffner et al. propose a straightforward and efficient bidirectional random tree algorithm, denoted as RRT-Connect [4], alternately expands two trees to improve the algorithm's efficiency. However, its performance still suffers in complex environments. Tahirovic et al. introduced a rapid exploration algorithm named Rapid Random Vine (RRV) [5] for efficient exploration. It determines the local environment type using principal component analysis (PCA), a dimensionality reduction technique that captures the most significant variations in the data. By analyzing

the relationships among environmental features, RRV selects an appropriate direction in which to expand, thereby solving the narrow passage problem well. However, its performance is poor in complex environments without passages. Hsu et al. [6] augmented the Rapidly-Exploring Random Tree (RRT) algorithm with a bridge-testing technique, which increased the sampling probability in narrow passages and decreased the sampling probability in non-interest regions, thereby addressing narrow passage issues. However, the unevenness of the sampling probability may cause the algorithm to ignore some feasible paths. Wu et al. proposed the Fast-RRT [7] algorithm, which detects narrow passages by re-randomizing the expansion direction at collision points, but the algorithm's stability is poor. Cai et al. combined RRV with bridge testing [8], enabling efficient identification and expansion in complex environments without the need for additional collision detection, greatly reducing computational intensity, but the algorithm can generate a large number of useless vertices in open areas. Building upon RRV and RRT-Connect, Li et al. proposed an adaptive random tree algorithm called ARRT-Connect [9], which effectively improves the algorithm's performance, but the algorithm may fall into concave traps. Chi et al. [10] introduce a heuristic path-planning algorithm based on the Generalized Voronoi Diagram (GVD), which significantly improves the algorithm's performance in maze environments but requires preprocessing of the map and does not consider the narrow passage problem. Taheri et al. proposed a Fuzzy Greedy Rapidly Exploring Random Tree (FG-RRT) [11] algorithm, which significantly reduces computation time in maze, narrow passage, and convex obstacle environments, but the algorithm requires the setting of nine fuzzy rules, and the parameter settings are complex.

To address the problem of low-quality generated paths, the RRT\* algorithm was introduced by Karaman et al. [12], which introduced the ChooseParent and Rewire processes when adding new nodes to the tree, making the algorithm asymptotically optimal. As the number of iterations tends to infinity, the probability of finding the optimal solution approaches 100%. RRT\* is a milestone in the development of RRT. To improve the convergence speed of the RRT\* algorithm, numerous scholars have conducted extensive research, mainly optimizing the sampling, ChooseParent, and Rewire processes of RRT\*. Islam et al. put forth an intelligent sampling tree named RRT\*-Smart [13] to expedite the convergence rate of the algorithm, but the quality of the generated path depends largely on the initial solution. Inspired by node exclusion, Gammell et al. [14] employ a direct sampling method within the hyperellipsoid to enhance algorithm performance, but the algorithm is no longer applicable when the ellipsoid is larger than the planning domain. P-RRT\* [15] combines APF and RRT\* to provide feasible directions for sampling exploration, which speeds up the convergence speed. Jeong et al. improved the ChooseParent and Rewire procedures using the triangle inequality to propose the Quick-RRT\* [16], which generates better initial paths and faster convergence. Inspired by Quick-RRT\*, F-RRT\* [17] creates a parent node near the obstacle for each sampled point, obtaining better initial solutions and faster convergence speed than Quick-RRT\* and RRT\* under the same conditions. Although algorithms based on the RRT\* framework can find the optimal or approximate optimal solution, they all require a large number of samples to gradually search for the optimal path. Therefore, when the important parameter for an algorithm is its running speed, optimizing the path directly generated by RRT is necessary. To optimize the initial path generated by RRT, Qian et al. [18] proposed a method to optimize the initial path generated by RRT by merging trees based on the initial path to form a closed-loop path and then performing optimization to obtain the relatively optimal path. Chen et al. [19] introduced a bidirectional pruning optimization strategy that prunes redundant nodes from both the starting and ending points of the path and selects the shortest optimized path, effectively improving the quality of the path.

In conclusion, extensive research has been conducted on path planning utilizing the RRT algorithm in complex environments. However, no algorithm currently exists that effectively and simply addresses the dual issues of subpar performance in complex environments and inferior path quality. To remedy this, the present paper proposes the complex

environments rapidly-exploring random tree (CERRT) path planning algorithm inspired by RRV, which greatly improves the efficiency of the algorithm in complex environments and optimizes the generated initial feasible paths.

The main contributions of this paper are as follows:

- (1) We have designed a new process for environmental perception. This process determines the type of environment by sampling the local area, eliminating the need for principal component analysis and significantly reducing computational complexity.
- (2) We propose a pre-allocated vertex expansion method in conjunction with a vertex death mechanism. This approach foregoes the expansion of inactive tree vertices to prevent the algorithm from getting stuck in concave areas. When combined with the environment-aware capability, the algorithm deftly navigates complex environments such as mazes, narrow passages, and concave regions.
- (3) We also suggest a bidirectional contraction optimization strategy. Once a feasible path is identified, its points are contracted in both directions, yielding a more streamlined and efficient path.

The rest of this paper is structured as follows: Section 2 outlines the mathematical definition of the planning problem along with a brief introduction to the core principles of RRT, RRV and Fast-RRT. Section 3 offers an in-depth description of our proposed CERRT algorithm framework. Section 4 presents simulation experiments that compare the new algorithm against RRT and RRV. Finally, Section 5 concludes the paper.

## 2. Background

In this section, we first introduce the mathematical definition of the path planning problem and then briefly describe the RRT and RRV algorithms.

### 2.1. Problem Definition

Let  $X$  be the configuration space,  $X_{obs}$  be the obstacle region, and  $X_{free} = X/X_{obs}$  be the feasible region.  $(X, X_{start}, X_{goal})$  defines a path planning problem, where  $x_{start} \in X_{free}$  is the initial state and  $X_{goal} \subset X_{free}$  is the goal area. Let a continuous function  $\sigma: [0, n] \rightarrow X$  of bounded variation be a path, where  $n$  is the path point number. If  $\forall \tau \in [0, n], \sigma(\tau) \in X_{free}$ , then  $\sigma$  is a feasible path, defined as  $\sigma_{free}$ .

#### Definition 1. Feasible Path Solution.

For the  $(X, X_{start}, X_{goal})$  problem, if  $\exists \sigma \in \sigma_{free}$ , where  $\sigma(0) = X_{start}$  and  $\sigma(n) \in X_{goal}$ , then the path is called a feasible path solution  $\sigma^*$ ; otherwise, report a path planning failure.

#### Definition 2. Approximate Optimal Path Solution.

For the  $(X, X_{start}, X_{goal})$  problem, if  $\exists \sigma^*$  satisfies  $C(\sigma^*) \leq \min\{C(\sigma) : \sigma \in \sigma_{free}\} * 1.05$ , then output path  $\sigma^*$  is the approximate optimal path solution; otherwise, report a failure.

### 2.2. RRT

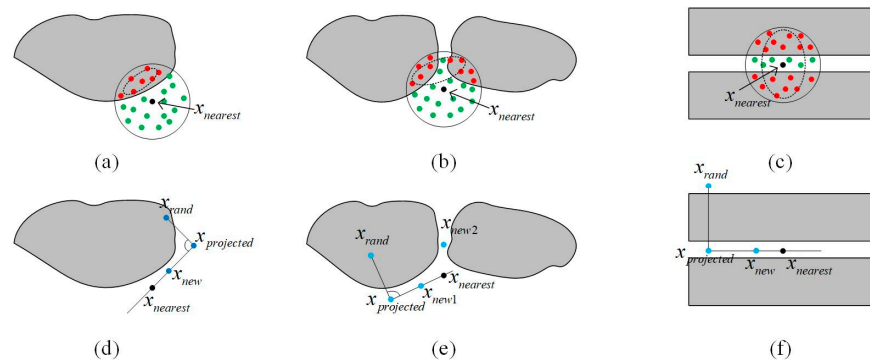
RRT explores the configuration space by maintaining a tree  $T$ . The algorithm sets the root node of the tree as  $x_{start}$  and performs an iterative expansion. In each iteration, the sampler randomly selects a sample  $x_{rand}$  from the configuration space, finds the vertex  $x_{nearest}$  in  $T$  closest to  $x_{rand}$ , and extends a step size  $d_{stepsize}$  from  $x_{nearest}$  toward  $x_{rand}$  to obtain the node  $x_{new}$  for expansion. If the local path from  $x_{nearest}$  to  $x_{new}$  is collision-free, then  $x_{new}$  is added to the tree. The algorithm terminates either when a feasible path is obtained or when the maximum number of iterations ' $N$ ' is exceeded.

However, the randomness of the sampler often results in a low sampling probability in narrow passages, leading to fewer sampling points in such areas. Consequently, it becomes challenging for the expansion tree to detect these narrow passages. This limitation hampers the effectiveness of the RRT algorithm in complex environments.

### 2.3. RRV

RRV is an algorithm developed to overcome the narrow passage problem found in the RRT algorithm. It uses principal component analysis to identify the local environment type, as it can effectively map high-dimensional data to a lower-dimensional space while maximizing information retention, which facilitates the extraction of data features. Specifically, this process involves mapping sampled points within obstacles into a single feature vector that maximally retains information from the obstacle points. Subsequently, the  $x_{rand}$  point is projected onto the feature vector passing through  $x_{nearest}$ . The resulting projected point,  $x_{projected}$ , serves as a novel direction for the expansion of RRV. This strategic extension enables the random tree to circumvent obstacles, similar to the growth of a vine, along the obstacle boundaries.

As illustrated in Figure 1, the RRV algorithm generates local random sampling points (shown in red and green) and performs principal component analysis on the red obstacle points. The confidence ellipse is then used to determine the type of environment. If the ellipse does not contain a green point, it is classified as a convex obstacle environment (Figure 1a). If it contains a green point but not  $x_{nearest}$ , it is identified as a passage entrance environment (Figure 1b). If it contains both a green point and  $x_{nearest}$ , it is a passage interior environment (Figure 1c).



**Figure 1.** Environment judgment and expansion in RRV. (a,d) represent convex obstacles, (b,e) represent the entrance of the passage, and (c,f) represent the interior of the passage.

Then, the  $x_{rand}$  point is projected onto a principal component analysis feature vector passing through  $x_{nearest}$  to obtain  $x_{projected}$ , and the tree is expanded toward this point to avoid growing toward obstacles. If the environment is identified as a convex obstacle or a passage interior, the tree is expanded along the obstacle (as shown in Figure 1d,f). If it is identified as a narrow passage entrance, as shown in Figure 1e, the tree is further expanded along the obstacle to  $x_{new1}$  and toward the interior of the passage to  $x_{new2}$ . This enables RRV to discover narrow passages more effectively than the classic RRT algorithm, and once a narrow passage is discovered, the expansion tree can grow quickly.

However, when applied to environments without narrow passages, the performance of RRV falls short compared to the original RRT algorithm. This indicates a high degree of environmental dependence in its performance.

### 3. CERRT

The CERRT algorithm seeks to correct the insensitivity of traditional RRT algorithms to the environment. It utilizes a novel node expansion strategy to improve expansion efficiency and incorporates new environmental awareness capabilities to address narrow passage problems. Moreover, it introduces a path optimization strategy to enhance the quality of the paths generated, making it a more efficient solution overall.

### 3.1. Algorithm Framework

The CERRT algorithm is an optimization of the RRT algorithm. It uses an array  $V$  to store expandable points in tree  $T$  and constantly deletes non-extensible points, known as dead nodes.

The CERRT algorithm initially stores the starting point  $x_{start}$  point as the root node in the tree and pre-allocates a corresponding set of expandable points,  $x_{start}.CAND$  for it (Lines 1–3 in Algorithm 1). The sampler is adjusted to amplify the tree's growth orientation by systematically sampling points within the goal region with a certain probability. (Line 5 in Algorithm 1).

Following sampling, the algorithm selects the nearest point  $x_{nearest}$  to the sampling point from the array  $V$  and queries the closest vertex to the sampling point  $x_{rand}$  from the expansion point set  $x_{nearest}.CAND$  of  $x_{nearest}$  to obtain the new node  $x_{new}$ . Once found, the point is removed from  $x_{nearest}.CAND$ . If  $x_{nearest}.CAND$  is empty after removal, and the vertex  $x_{nearest}$  is deemed dead and removed from the vertex array  $V$  (Lines 6–11 in Algorithm 1).

Should a collision happen during expansion, the algorithm enters the environment perception phase to determine the type of environment where  $x_{nearest}$  is located. If  $x_{nearest}$  is near a channel, a new  $x_{new}$  point is calculated for expansion; otherwise, resampling is performed (Lines 13–14 in Algorithm 1).

If no collision occurs during the expansion,  $x_{new}$  is added to tree  $T$ , and a corresponding set of candidate nodes for  $x_{new}$  is pre-allocated. If the set of candidate nodes overlaps with any existing nodes in the tree, no allocation is performed. If the set of candidate nodes for  $x_{new}$  is empty after allocation, it is not added to array  $V$  (Lines 17–21 in Algorithm 1).

Finally, after obtaining a feasible path, the algorithm optimizes it to reduce the cost  $C(\sigma)$  (Line 24 in Algorithm 1). A more detailed description of the algorithm process is given in the subsequent section.

---

#### Algorithm 1 CERRT( $x_{start}, x_{goal}, d_{step}, d_{gap}, N, Map$ )

---

```

1:  $T = \{x_{start}\};$ 
2:  $V = T;$ 
3:  $x_{start}.CAND = \text{initTree}(x_{start});$ 
4: for  $i = 1$  to  $N$  do
5:    $x_{rand} = \text{GetSample}(i);$ 
6:    $(x_{nearest}, near_{idx}) = \text{GetNearest}(V, x_{rand});$ 
7:    $(x_{new}, new_{idx}) = \min(\text{distance}(x_{nearest}.CAND(:), x_{rand}));$ 
8:    $\text{Delete}(x_{nearest}.CAND(new_{idx}));$ 
9:   if  $\text{Empty}(x_{nearest}.CAND)$  then
10:      $\text{Delete}(V(near_{idx}));$ 
11:   end if
12:   if  $\text{Overlap}(x_{new}, T)$  then  $\text{continue};$  end if
13:   if  $\text{Collision}(x_{new}, x_{nearest}, Map)$  then
14:      $(x_{new}, Fig) = \text{Aware}(x_{nearest}.parent, x_{nearest}, Map, d_{step}, d_{gap});$ 
15:     if  $\sim Fig$  then  $\text{continue};$  end if
16:   end if
17:    $x_{new}.CAND = \text{SetCAND}(x_{nearest}, x_{new}, d_{step}, T, V);$ 
18:   if  $\text{NoEmpty}(x_{new}.CAND)$ 
19:      $V = V \cup \{x_{new}\};$ 
20:   end if
21:    $T = T \cup \{x_{new}\};$ 
22:   if  $x_{new} \in X_{goal}$  then  $\text{break};$  end if
23: end for
24:  $\sigma = \text{GetPath}(\sigma);$ 
25: return  $\sigma = \text{OptPath}(\sigma);$ 

```

---

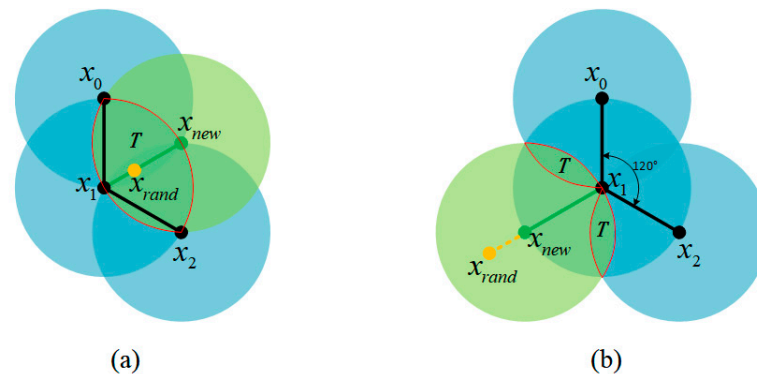
### 3.2. Vertex Expansion Method

The traditional rapidly-exploring random tree (RRT) algorithm randomly samples the configuration space to guide the tree's expansion and exploration. When the RRT algorithm expands a new vertex, it calculates the distance ' $d$ ' between the point and the goal. If ' $d$ ' is less than a predetermined threshold ' $r$ ', it means the algorithm has found the goal point and the search ceases. If not, the search continues.

In this regard, each time a new node is expanded, the RRT algorithm explores the region with a radius of ' $r$ ' centered on that node. The unexplored regions of tree nodes are referred to as the 'unknown regions', while the explored areas are defined as 'exploration regions'. With each expansion, new exploration regions are created within the unknown regions.

The RRT algorithm achieves the exploration of the entire space by continuously sampling and expanding. However, this process brings about a scenario where some exploration regions are revisited, some even more than twice. The exploration efficiency is thus measured by the area of novel exploration regions explored by newly expanded vertices and the frequency of re-exploration of already known regions. Efficiency is high when the area of unexplored regions explored is large, but it is low if the area of already known regions explored is large or if they are explored multiple times.

Figure 2 illustrates the issue of redundant exploration in known regions. In this figure, the blue region represents the explored area, while the overlapped area denotes the repeated examination. The green zone denotes the exploration area for new vertices, and new vertices conduct repeated exploration on region ' $T$ ' enclosed by the red circle more than three times.



**Figure 2.** Issue of redundant exploration in known regions. (a) Redundant exploration using the traditional extension strategy. (b) Improved extension strategy to reduce redundancy.

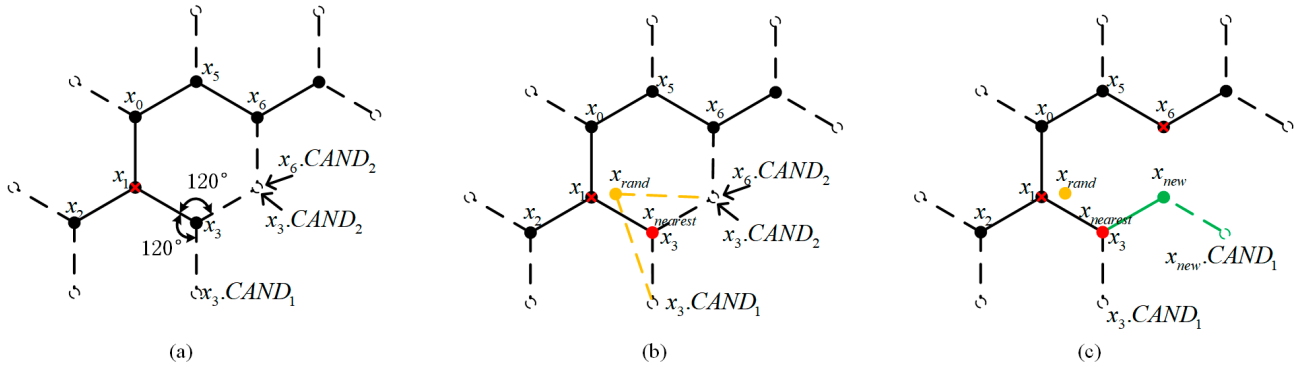
Figure 2a elucidates the traditional RRT expansion process. This diagram illustrates that vertices  $x_0$ ,  $x_1$ , and  $x_2$  have already examined the ' $T$ -region' twice. Following this, vertex  $x_1$  extends toward the  $x_{rand}$  point and produces a fresh vertex  $x_{new}$ . Regrettably, this new vertex instigates a third redundant exploration of the ' $T$ -region'. Multiple explorations of the same area are pointless.

In response to the problem mentioned earlier, our research suggests that setting the angle at 120 degrees between each vertex and its connected points can drastically decrease unnecessary exploration in space, as shown in Figure 2b. When a new point,  $x_{new}$ , extends from vertex  $x_1$ , the angles between the three edges  $x_1-x_0$ ,  $x_1-x_2$  and  $x_1-x_{new}$  are all 120 degrees. This decreases the ' $T$ -region' to a mere 30% of what it is in Figure 2a. The exploration process also avoids repeating exploration areas over three times, which greatly increases efficiency.

To achieve this, we propose pre-allocated expansion points and a vertex dead strategy to ensure that the angle between each vertex's edges is 120 degrees. Algorithm 2 presents the detailed process of allocating candidate extension points. Initially, the angle  $ang_0$  of the edge  $x_{new}-x_{nearest}$  is computed within the Cartesian coordinate system. Following

this, candidate extension points are created with a 120-degree bias angle (Lines 1–4 in Algorithm 2). If the candidate extension point coincides with a tree node, the allocation fails (Lines 6–10 in Algorithm 2).

Figure 3 depicts the improved vertex expansion strategy. In this figure, the black, red, green, and yellow dots, respectively, represent standard tree vertices,  $x_{nearest}$ ,  $x_{new}$ , and  $x_{rand}$  sample points. Dashed circles are potential expansion points for each vertex, while red-crossed points signify discarded vertices removed from array  $V$ .



**Figure 3.** Improved vertex expansion process. (a) Before expansion. (b) Selection of  $x_{nearest}$  after sampling. (c) After expansion.

Upon expanding a new vertex  $x_{new}$ ,  $x_{new}.CAND_j$  (where  $j$  is the candidate point index  $j = 0, 1, 2$ ) is pre-assigned to maintain each vertex’s edge angle at 120 degrees. Additionally, if a vertex’s candidate point set is vacant, it is classified as a discarded vertex and removed from array  $V$ . Figure 3a shows that every vertex logs its corresponding candidate expansion points  $x_i.CAND_j$  (where  $i$  is the vertex index in the growing tree  $T$ ,  $i = 0, 1, \dots, n$ ). If  $x_i$ ’s candidate point set is void, it is deemed a discarded vertex and excluded from array  $V$ .

The expansion process of the CERRT is conveyed in Figure 3b. After sampling  $x_{rand}$  randomly, the closest vertex  $x_{nearest}$  is selected from array  $V$  as the starting point. Since  $x_1$  is a discarded vertex and has been eliminated from array  $V$ ,  $x_3$  is chosen as the nearest vertex. Subsequently, vertex  $x_{new}$  is selected from the candidate expansion points of  $x_3$  based on its proximity to  $x_{rand}$ . After calculation,  $x_3.CAND_2$  is chosen as  $x_{new}$  for expansion and  $x_3.CAND_2$  is removed from the candidate set of  $x_3$ . As  $x_6.CAND_2$  coincides with  $x_3.CAND_2$ , it is also removed from the candidate set of  $x_6$ . Since the candidate set of vertex  $x_6$  is now empty,  $x_6$  becomes a dead vertex and is deleted from array  $V$ .

If edge  $x_{new}-x_{nearest}$  collides with obstacles, the algorithm advances to the environment awareness stage. Otherwise,  $x_{new}$  is added as a new vertex to tree  $T$ , and a set of candidate expansion points is pre-allocated for  $x_{new}$ . It is worth noting that the candidate points must not overlap with any existing points in tree  $T$ . This process is illustrated in Figure 3c.

---

**Algorithm 2** SetCAND( $x_{nearest}$ ,  $x_{new}$ ,  $d_{step}$ ,  $T$ ,  $V$ )

---

- 1:  $ang_0 = \text{GetCartesianAngle}(x_{nearest}, x_{new});$
  - 2:  $ang_1 = ang_0 + 120; ang_2 = ang_0 + 240;$
  - 3:  $new_1 = x_{new} + d_{step} * [\sin(ang_1), \cos(ang_1)];$
  - 4:  $new_2 = x_{new} + d_{step} * [\sin(ang_2), \cos(ang_2)];$
  - 5:  $CAND = [];$
  - 6: **if** NoOverlap( $new_1$ ,  $T$ ) **then**
  - 7:      $CAND(\text{end} + 1) = new_1;$
  - 8: **else if** NoOverlap( $new_2$ ,  $T$ ) **then**
  - 9:      $CAND(\text{end} + 1) = new_2;$
  - 10: **end if**
  - 11: **return** CAND;
-

### 3.3. Environmental Awareness

Algorithms based on Sampling often encounter difficulties when navigating narrow passages due to their lack of environmental sensitivity. However, whether a passage is deemed narrow is contingent on the extension step length. If the step length is much smaller than the narrow passage, it can be considered a spacious road. Conversely, if the step length is too small, the search accuracy will be too high, resulting in lower algorithm efficiency. To combat this, an environment perception strategy is proposed to enable the random tree to quickly identify and pass through narrow passages without decreasing the step length. When a collision occurs with an obstacle during the tree expansion process, expansion is halted and enters the environment perception stage. Algorithm 3 provides a comprehensive outline of this phase.

To begin the environmental perception process, local spatial information is collected around the  $x_{nearest}$  point through local sampling. Local sampling uniformly samples  $n$  points around the vertex to be expanded using the expansion step length as the radius and stores them in a point set  $S$ . Here,  $n = 16$  is used as an example. Set  $S$  is then separated into two subsets,  $S_{obs}$  and  $S_{free}$ , based on whether the position of  $S$  lies within the obstacle area. The boundary points between  $S_{free}$  and  $S_{obs}$  are selected and stored in  $S_{bdry}$ , where  $S_{bdry} \subseteq S_{free}$  (Lines 2–3 in Algorithm 3). Figure 4 illustrates the schematic diagram of local sampling, where the red point represents the expanding vertex where a collision occurred, the blue points represent  $S_{obs}$  sampling points, the green points represent boundary points  $S_{bdry}$ , and the yellow and green points represent  $S_{free}$  points. In Figure 4, the obstacle is recognized as a wall obstacle.

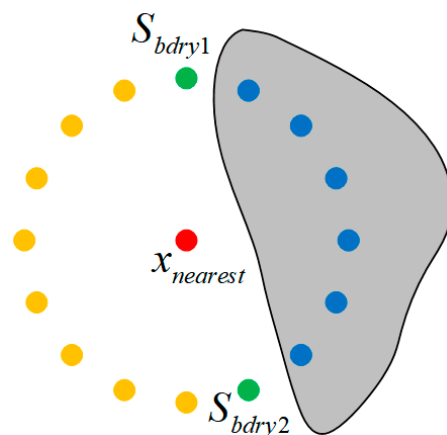


Figure 4. Local sampling process and wall obstacle types.

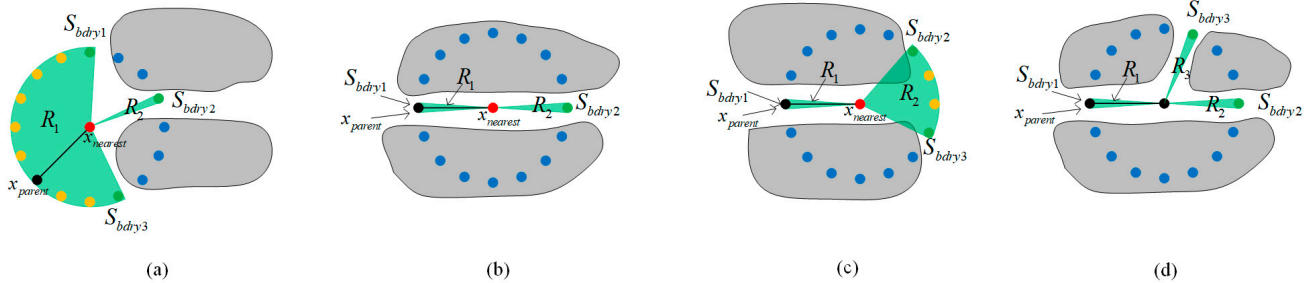
For the local sampling to accurately identify narrow passages, it is crucial to further clarify the quantity of local sampling points  $n$  and the extension step size  $d_{step}$ . Given the width  $d_{gap}$  of the minimum feasible driving passage and the extension step size  $d_{step}$ , to ensure that the local sampling can sample the interior of the passage, the spacing between adjacent sampling points should not exceed  $d_{gap}$ . The number of sampling points  $n$  must align with Equation (1).

$$n \geq \frac{2\pi}{\arccos\left(1 - \frac{d_{gap}^2}{2d_{step}^2}\right)} \tag{1}$$

After local sampling, environments are categorized based on the numerical relation between the point sets  $S_{free}$  and  $S_{bdry}$ . The environment is split into two primary classifications: wall obstacles and passage environments. If  $S_{bdry}$  has only two vertices and  $S_{free}$  has more than two vertices, the environment is classified as a wall obstacle, as shown in Figure 4. In this case, the algorithm exits the environment perception and performs resampling (Lines 4–6 in Algorithm 3). Otherwise, the environment is considered a narrow passage environment, which can be further classified into entrance, interior, exit, and



multi-branching regions, as shown in Figure 5. In this figure, the red point represents the  $x_{nearest}$  point, the yellow points represent  $S_{free}$  sample points, the green points represent the boundary points  $S_{bdry}$ , and the blue points represent the  $S_{obs}$  sample points. The green sector area  $R$  contains a continuous set of  $S_{free}$  points.



**Figure 5.** Narrow passage perception. (a) Entrance of the passage; (b) Interior of the passage; (c) Exit of the passage; (d) Multiple branching passages.

To traverse the narrow passage, multiple sector regions are divided using  $x_{nearest}$  as the fulcrum and the expansion step as the radius. Each sector region contains only a continuous set of  $S_{free}$  points. The resulting  $n$  sector regions are denoted as  $R_i$  ( $i = 1, 2, \dots, n$ ). The sector area that does not contain the  $x_{parent}$  point is selected as the area for expansion. An  $S_{free}$  point that will not cause a collision is then selected from it as the  $x_{new}$  point for expansion (Lines 8–13 in Algorithm 3).

For example, Figure 5a illustrates the entrance of a narrow passage. After the area is divided into two sector regions ( $R_1, R_2$ ),  $S_{bdry2}$  is selected from  $R_2$  as the  $x_{new}$  point for expansion since  $x_{parent}$  is positioned in  $R_1$ . The same expansion approach is applied to the narrow passage interior shown in Figure 5b and the exit in Figure 5c.

If the number of sector areas  $n$  is greater than 2, it is considered a multi-passage branching situation, as shown in Figure 5d. After division, three sector areas ( $R_1, R_2, R_3$ ) are obtained. Since  $R_1$  contains the  $x_{parent}$  point, two  $S_{free}$  points are selected from the  $R_2$  and  $R_3$  regions that will not cause collisions as  $x_{new}$  for expansion.

---

**Algorithm 3**  $Aware(x_{parent}, x_{nearest}, Map, d_{step}, d_{gap})$

---

```

1:  $x_{new} = []$ ;  $Fig = false$ ;
2:  $(S_{free}, S_{obs}) = LocalSample(x_{nearest}, Map, d_{step}, d_{gap})$ ;
3:  $S_{bdry} = GetBoundary(S_{free}, S_{obs})$ ;
4: if  $size(S_{bdry}) == 2$  &&  $size(S_{free}) > 2$  then
5:    $Fig = false$ ;
6:   return  $(x_{new}, Fig)$ ;
7: else
8:    $(R, n) = DivideRegion(S_{free}, S_{obs}, S_{bdry})$ ;
9:   for  $i = 1$  to  $n$  do
10:    if  $x_{parent} \in R_i$  then  $R = R - R_i$  end if
11:  end for
12:   $x_{new} = ChoseNew(R, S_{free}, n - 1)$ ;
13:   $Fig = true$ ;
14: end if
15: return  $(x_{new}, Fig)$ ;

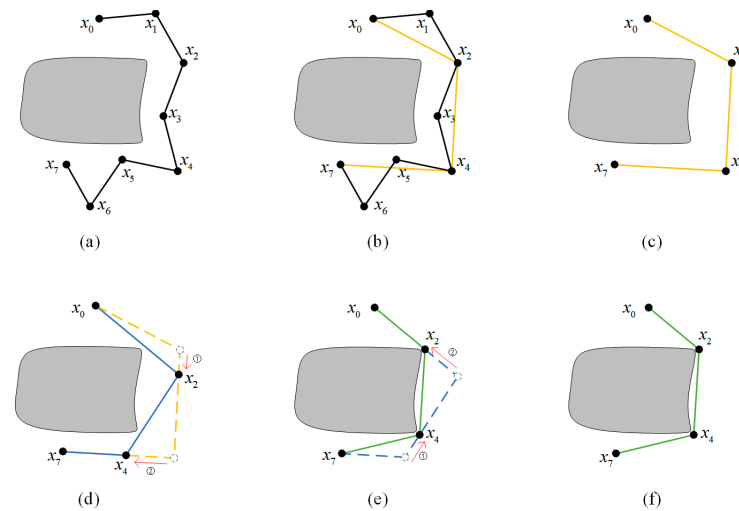
```

---

### 3.4. Path Optimization Strategy

Sampling-based algorithms for path planning can often generate redundant nodes due to their random nature, potentially lowering the quality of path planning. In this paper, a bidirectional shrinking optimization strategy (BSOS) is proposed based on the pruning optimization strategy (POS) to further enhance the quality of the path.

In Figure 6a, moving from  $x_0$  to  $x_7$  requires avoiding an obstacle, and the black line path  $x_0-x_1-x_2-x_3-x_4-x_5-x_6-x_7$  in the figure is the original planned path. This path contains a large number of redundant nodes. By using the pruning strategy and based on the triangle inequality theorem, the robot can move directly from  $x_0$  to  $x_2$  without passing through  $x_1$ , thus identifying  $x_1$  as a redundant node.



**Figure 6.** Path optimization strategy. (a–c) show pruning optimization of the original path, where the yellow path represents the pruned optimized path. (d–f) show bidirectional shrinkage optimization of the path, where the green path represents the final optimized path.

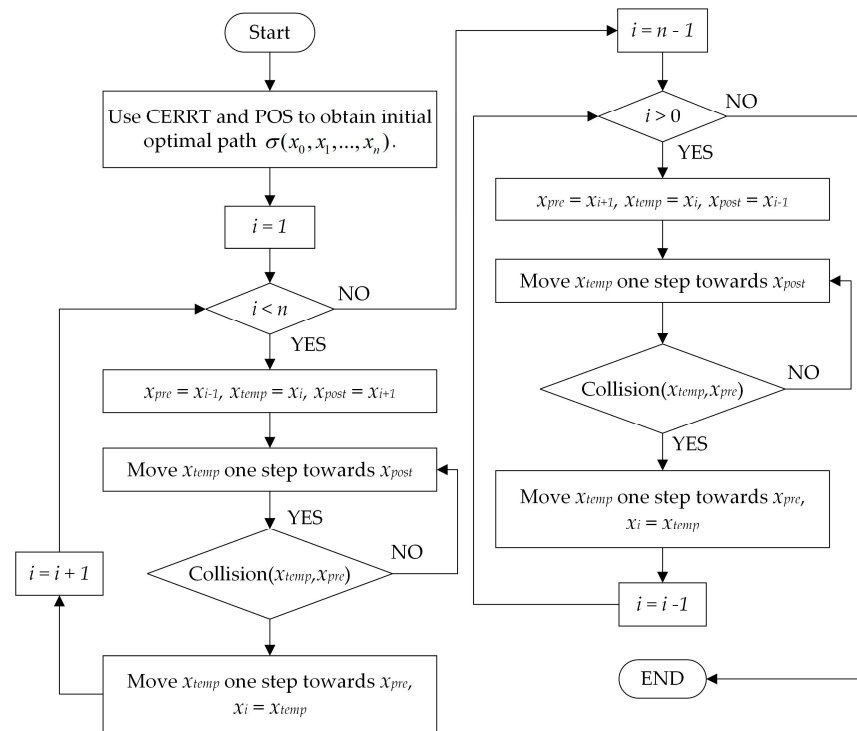
Figure 6b illustrates the application of the pruning operation on the entire initial path, eliminating superfluous nodes  $x_1$ ,  $x_3$ ,  $x_5$ , and  $x_6$ . The remaining nodes are connected to obtain the pruned and optimized yellow path  $x_0-x_2-x_4-x_7$ , as shown in Figure 6c. The yellow path obtained by POS has not only fewer path points but also a shorter path length, but it is not an approximate optimal path.

This paper proposes a bidirectional shrinking-based optimization of path points on the basis of pruning the path  $x_0-x_2-x_4-x_7$ . The endpoints  $x_0$  and  $x_7$  are excluded from the shrinking process, leaving the remaining points to participate in two phases of reduction.

In the first round, each path point moves toward the next point with a higher number according to the sequence of the point number. At the same time, it constantly checks whether there is any collision with the previous path point. If a collision is about to occur, the point stops moving, as shown in Figure 6d. For instance, path point  $x_2$  moves toward  $x_4$  until the  $x_0-x_2$  segment is about to collide with the obstacle, and then the point stops. After that, path point  $x_4$  moves toward  $x_7$  until the  $x_2-x_4$  segment is about to collide with the obstacle, and then the point stops. Once all path points have completed the shrinking movement, the blue path  $x_0-x_2-x_4-x_7$  in Figure 6d is obtained, indicating that the first round of shrinking is completed.

In the second round of contraction, the order is reversed, with the points moving from high to low according to their vertex number. As shown in Figure 6e, path point  $x_4$  moves toward  $x_2$  first, and then it stops when the  $x_4-x_7$  segment is about to collide with the obstacle. Then, path point  $x_2$  moves toward  $x_0$  until the  $x_2-x_4$  segment is about to collide with the obstacle. Once all path points have completed the shrinking movement, the green path  $x_0-x_2-x_4-x_7$  in Figure 6f is obtained, which is the final path obtained by the bidirectional search optimization strategy. The bidirectional shrinking method results in a path that is shorter and closer to the optimal configuration.

The specific process of using the bidirectional shrinking path optimization strategy to optimize the CERRT planning path is shown in Figure 7.



**Figure 7.** Bidirectional shrinking optimization strategy flowchart.

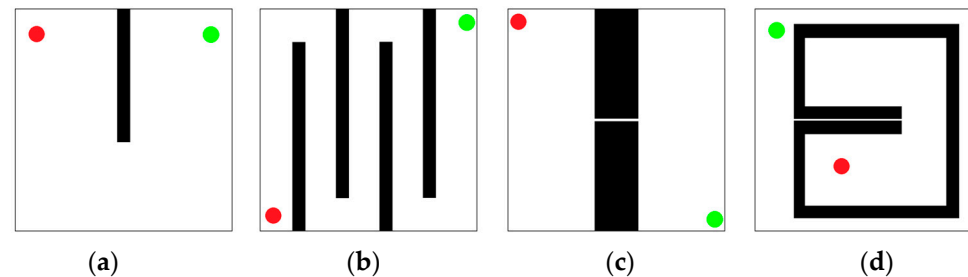
- (1) Obtain the initial optimized path  $\sigma(x_0, x_1, \dots, x_n)$  using the CERRT and pruning optimization strategies, where  $x_i$  ( $i = 0, 1, \dots, n$ ) is a path point and  $i$  is the path point number.
- (2) Initialize  $i = 1$ .
- (3) Check whether  $i$  is less than  $n$ . If it is, put  $x_i$  into the temporary variable  $x_{temp}$ , put  $x_{i-1}$  into the variable  $x_{pre}$ , and put  $x_{i+1}$  into the variable  $x_{post}$ . If not, go to step (6).
- (4) Move the current variable point  $x_{temp}$  one step toward the variable point  $x_{post}$ .
- (5) Check whether the path segment  $x_{temp}-x_{pre}$  collides with any obstacles. If there is a collision, move  $x_{temp}$  one step toward the opposite direction of  $x_{post}$ , update the path point  $x_i$  to  $x_{temp}$ , set  $i = i + 1$ , and go to step (3). If there is no collision, go to step (4).
- (6) Set  $i = n - 1$ .
- (7) Check whether  $i$  is greater than 0. If it is, put  $x_i$  into the temporary variable  $x_{temp}$ , put  $x_{i+1}$  into the variable  $x_{pre}$ , and put  $x_{i-1}$  into the variable  $x_{post}$ . If not, the optimized path is obtained, and the process ends.
- (8) Move the current variable point  $x_{temp}$  one step toward the variable point  $x_{post}$ .
- (9) Check whether the path segment  $x_{temp}-x_{pre}$  collides with any obstacles. If there is a collision, move  $x_{temp}$  one step toward the opposite direction of  $x_{post}$ , update the path point  $x_i$  to  $x_{temp}$ , set  $i = i - 1$ , and go to step (7). If there is no collision, go to step (8).

After the entire process is executed, the path points are updated by shrinking, and the obtained path  $\sigma(x_0, x_1, \dots, x_n)$  is the final optimized path. Note, that “moving one step” in the process refers to moving one pixel.

#### 4. Simulation and Experiment

To verify the algorithm performance of CERRT, this study conducted a comparative analysis of the RRT, RRV [5], Fast-RRT [7] and CERRT algorithms in a simple environment, a maze environment, a narrow passage environment, and a bug environment. The accessible passage width was set to  $d_{gap} = 10$  px, and the map size was  $1000 \text{ px} \times 1000 \text{ px}$ , as shown in Figure 8. The tree expansion step was set to  $d_{step} = 30$  px, and the detection radius was  $r = d_{step}$ . The sampler probability of sampling in the target area was 0.05, and the probability of sampling in the random area was 0.95. The maximum sampling value was

set to 80,000, and if the number of samples exceeded the maximum value, the path planning was considered a failure.



**Figure 8.** Environments for the simulations. (a) Simple. (b) Maze. (c) Narrow. (d) Bug Trap.

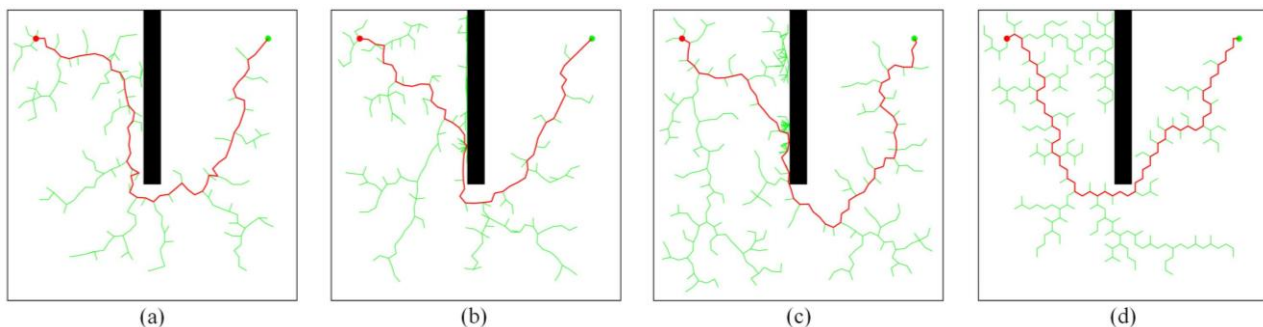
The algorithm performance was evaluated using three criteria: execution time, number of tree vertices generated, and success rate. These evaluations were conducted by repeating each simulation 100 times. The execution time and number of vertices were only counted for successful path planning. In the path optimization experiment, this study compared the original planned path, the pruning optimized path, and the proposed bidirectional shrinking optimized path and evaluated the path quality  $C(\sigma^*)$  using two indicators: path length and smoothness. All simulations were performed on a machine with an Intel (R) Core (TM) i7-12700H 2.30 GHz CPU and 16 GB of RAM. The simulation platform was MATLAB R2022a, and the function min was employed in all algorithms to find the nearest neighbor in all algorithms. The collision detection program used the linear trial method [20].

#### 4.1. Path Planning Simulation

In the path planning simulations, we tested the performance of the RRT, RRV, Fast-RRT and CERRT algorithms in four 2D environments. The objective of using a simple environment was to assess if the new algorithm's performance was significantly impacted by additional computation. On the other hand, the maze environment, narrow environment, and Bug Trap environment were utilized to assess the algorithms' performance in complex scenarios. The red dots in the map represent the starting points, the green dots represent the target points. The entire exploration process is represented by the green lines, and the generated path is represented by the red lines. The sampling method was consistent across all experiments.

##### 4.1.1. Simple Environment

The purpose of testing the algorithm in a simple environment was to evaluate the performance loss of the new algorithm with additional computational costs. The planning scenarios are shown in Figure 9, and the expansion shapes of the four random trees were generally similar.



**Figure 9.** Performance comparison of four algorithms in sample environment. (a) RRT; (b) RRV; (c) Fast-RRT; (d) CERRT.

Table 1 presents the performance of the algorithm in a simple environment. The success rates of the four algorithms are all 100%, and the average number of tree nodes was similar. The average running time of RRV was four times that of RRT, the average running time of Fast-RRT was 1.5 times that of RRT, and the average running time of CERRT was 1.7 times that of RRT. The time variance of CERRT was close to that of RRT and much smaller than that of RRV.

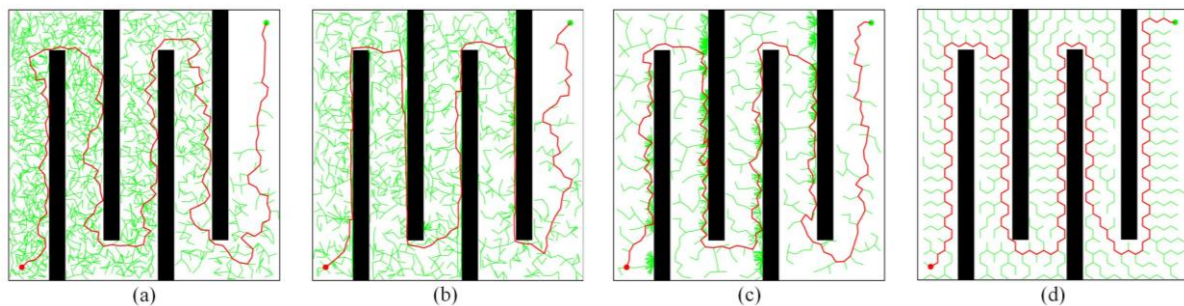
**Table 1.** Results for planning in a simple environment.

| Algorithm | Avg. Time (ms) | Min Time (ms) | Max Time (ms) | Std   | Avg. Nodes | Success Rate |
|-----------|----------------|---------------|---------------|-------|------------|--------------|
| RRT       | 9.08           | 5.61          | 14.48         | 2.04  | 263        | 1.00         |
| RRV       | 37.04          | 7.33          | 71.00         | 10.90 | 281        | 1.00         |
| Fast-RRT  | 13.15          | 6.58          | 32.45         | 3.94  | 355        | 1.00         |
| CERRT     | 15.86          | 9.28          | 24.63         | 2.81  | 286        | 1.00         |

The data indicate that the performance loss of CERRT was significantly more than that of RRV, and its overall performance was almost identical to RRT, indicating that the performance loss caused by the additional computational cost of CERRT was minimal.

#### 4.1.2. Maze Environment

The maze environment was designed to evaluate the performance of algorithms in complex environments without passages. Figure 10 illustrates the planning situations of the four algorithms, indicating that the utilization rates of tree nodes were low for RRV, RRT, and Fast-RRT, and there were numerous repeated exploration points on the left side of the map. In contrast, CERRT's tree nodes were evenly distributed, enabling the exploration of a broader area with fewer points.



**Figure 10.** Performance comparison of four algorithms in maze environment. (a) RRT; (b) RRV; (c) Fast-RRT; (d) CERRT.

Table 2 presents the algorithms' performance in the maze environment. RRT's time consumption was 2.7 times that of CERRT, Fast-RRT's time consumption was 1.7 times that of CERRT, and RRV's time consumption was 15 times that of CERRT. CERRT had the shortest running time and the smallest number of tree nodes. Furthermore, CERRT's time standard deviation was significantly lower than those of RRT and RRV, further indicating that this algorithm was most stable in the Maze environment.

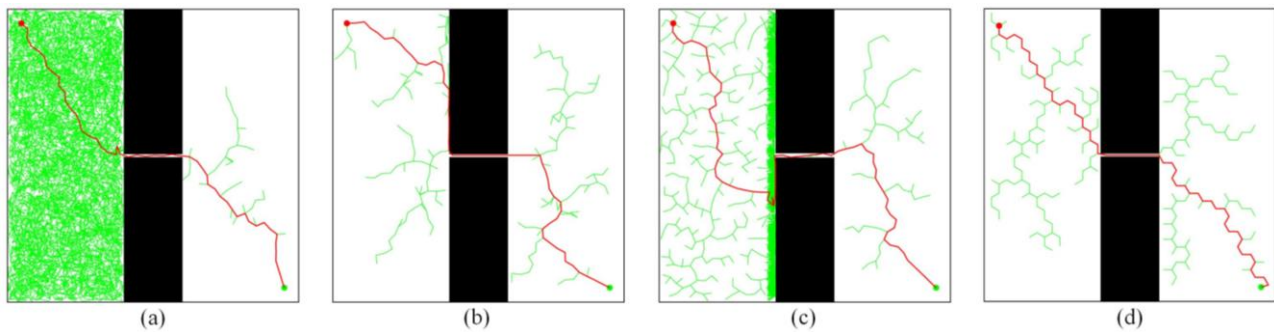
**Table 2.** Results for planning in a maze environment.

| Algorithm | Avg. Time (ms) | Min Time (ms) | Max Time (ms) | Std    | Avg. Nodes | Success Rate |
|-----------|----------------|---------------|---------------|--------|------------|--------------|
| RRT       | 108.29         | 83.66         | 153.99        | 13.97  | 2284       | 1.00         |
| RRV       | 603.24         | 438.86        | 1001.21       | 120.67 | 3042       | 1.00         |
| Fast-RRT  | 68.93          | 48.75         | 135.38        | 11.58  | 1368       | 1.00         |
| CERRT     | 39.73          | 34.59         | 112.35        | 8.66   | 610        | 1.00         |

Based on the above, it can be concluded that the new CERRT algorithm outperformed the RRT, RRV, and Fast-RRT algorithms in various aspects in complex non-navigable environments. The main reason for this is the improved vertex expansion strategy, which effectively improved the utilization rate of vertices by pre-allocating them for expansion. This strategy reduced repeated exploration of the same region and introduced a vertex death mechanism to eliminate useless vertices, thus reducing the time cost of selecting the optimal neighboring vertex.

#### 4.1.3. Narrow Environment

The RRT algorithm faced challenges in solving narrow passage problems, while the RRV algorithm was specifically designed to address this issue. Additionally, the Fast-RRT has also proposed solutions for narrow passages. In this study, we used a narrow environment to test the performance of the new algorithm and evaluate its environmental adaptability. Figure 11 shows the planning process of the four algorithms. It can be seen that RRT had a large number of vertices on the left side of the map, and the algorithm could not effectively detect the narrow passages on the walls. Fast-RRT expands multiple times near obstacles to find passages. RRV and CERRT were quickly able to discover and pass through the narrow passage.



**Figure 11.** Performance comparison of four algorithms in narrow environment. (a) RRT; (b) RRV; (c) Fast-RRT; (d) CERRT.

Table 3 showcases the performance of the algorithms in the narrow passage environment. The planning success rate of the RRT algorithm was 0.97, while the other algorithms were both 1.00, highlighting the shortcomings of RRT in narrow environments. The average running time of RRT was 27 times that of CERRT, the average running time of RRV was four times that of CERRT, and the average running time of Fast-RRT was slightly higher than that of CERRT. The new algorithm had a shorter planning time. The tree vertex numbers of CERRT and RRV were similar, and much lower than that of the RRT and Fast-RRT algorithm, indicating that both derived algorithms can solve narrow passage problems. RRT's time standard deviation was 135 times higher than CERRT, RRV's time standard deviation was 20 times higher than CERRT, and Fast-RRT's time standard deviation was 75 times higher than CERRT, suggesting that CERRT exhibited optimal stability.

**Table 3.** Results for planning in a narrow environment.

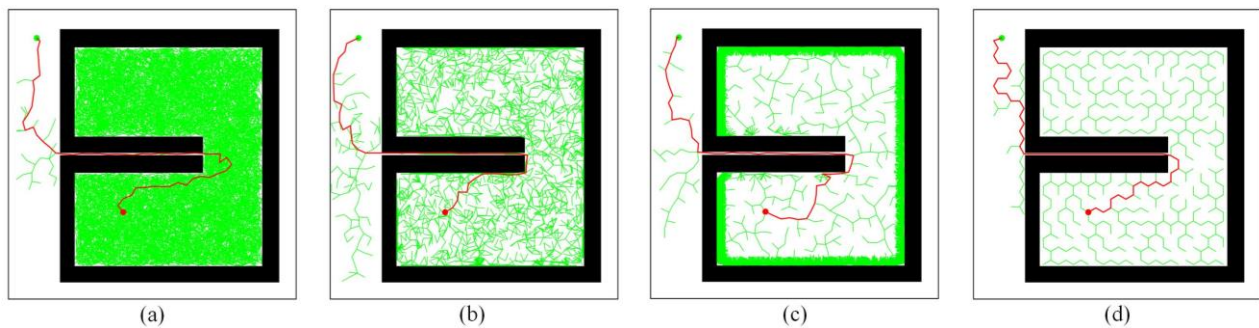
| Algorithm | Avg. Time (ms) | Min Time (ms) | Max Time (ms) | Std    | Avg. Nodes | Success Rate |
|-----------|----------------|---------------|---------------|--------|------------|--------------|
| RRT       | 455.13         | 7.52          | 5189.01       | 871.53 | 2931       | 0.97         |
| RRV       | 65.72          | 25.04         | 1287.83       | 127.12 | 259        | 1.00         |
| Fast-RRT  | 58.03          | 4.87          | 3779.04       | 487.47 | 869        | 1.00         |
| CERRT     | 16.70          | 11.66         | 74.69         | 6.42   | 255        | 1.00         |

Based on the above analysis, we can conclude that the new CERRT algorithm outperformed the RRT, RRV and Fast-RRT algorithms in narrow environments. The main

reason for this is that RRV requires complex principal component analysis to determine the environment, while CERRT's environmental perception ability only requires simple sampling to determine the surrounding environment and select new expansion points without complex calculations, effectively reducing the environmental recognition cost.

#### 4.1.4. Bug Trap Environment

The bug trap environment is created by adding concave traps to a narrow environment to assess the algorithm's ability to handle such obstacles. Figure 12 shows the planning process of the four algorithms. Notably, both the RRT and Fast-RRT algorithms performed extremely poorly in the Bug Trap environment, as evidenced by a significant accumulation of tree vertices inside the trap. The RRV algorithm can only recognize convex obstacles and mistakenly identified the concave traps as entryways, resulting in multiple attempts to expand within the trap area, which greatly reduced the algorithm's performance, with the number of tree vertices still high. The CERRT algorithm effectively utilizes each vertex to explore the space and can quickly discover and pass through the real channel.



**Figure 12.** Performance comparison of four algorithms in Bug Trap environment. (a) RRT; (b) RRV; (c) Fast-RRT; (d) CERRT.

Table 4 illustrates the performance of the algorithms in the Bug Trap environment. The planning success rates of RRT and Fast-RRT were only 0.90 and 0.92, respectively, while the other two algorithms achieved rates of 1.00. The average running time of RRT was 101 times that of CERRT, the average running time of Fast-RRT was 20 times that of CERRT, and the average running time of RRV was 52 times that of CERRT, with the new algorithm having the shortest planning time. The average number of tree vertices of RRT was 29 times that of CERRT, the average number of tree vertices of Fast-RRT was 21 times that of CERRT, and the average number of tree vertices of RRV was six times that of CERRT, indicating that the new algorithm had the highest vertex utilization rate. Moreover, RRT's time standard deviation was 526 times higher than CERRT, Fast-RRT's time standard deviation was 371 times higher than CERRT, and RRV's time standard deviation was 329 times higher than CERRT, underscoring the superior stability of the CERRT algorithm.

**Table 4.** Results for planning in a Bug Trap environment.

| Algorithm | Avg. Time (ms) | Min Time (ms) | Max Time (ms) | Std     | Avg. Nodes | Success Rate |
|-----------|----------------|---------------|---------------|---------|------------|--------------|
| RRT       | 2725.83        | 504.12        | 8782.20       | 1856.74 | 13902      | 0.90         |
| RRV       | 1418.43        | 31.42         | 6234.62       | 1160.86 | 2679       | 1.00         |
| Fast-RRT  | 1674.35        | 267.01        | 9574.50       | 1313.36 | 9798       | 0.92         |
| CERRT     | 26.97          | 13.36         | 35.28         | 3.53    | 474        | 1.00         |

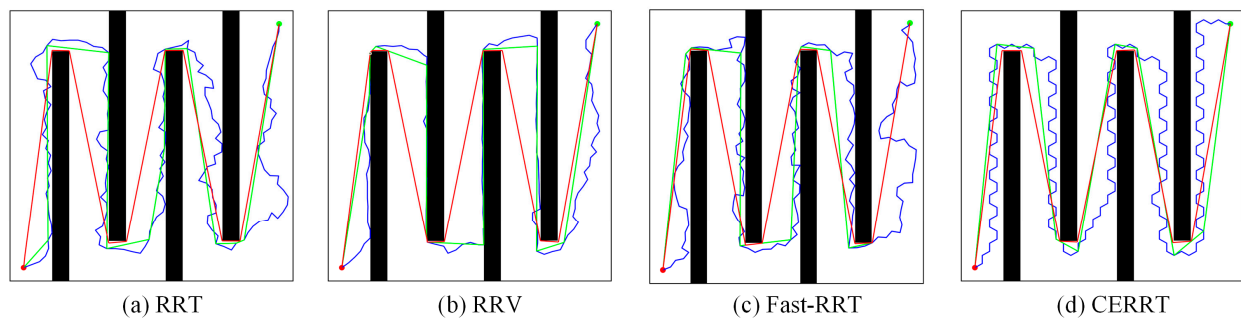
Based on the above analysis, it can be concluded that the new CERRT algorithm outperformed the RRT and RRV algorithms in the Bug Trap environment. The reason is that the vertex death mechanism can deactivate the vertices inside the traps, preventing the algorithm from becoming stuck in the concave traps. Combined with environmental awareness, the CERRT algorithm was quickly able to break through the bug trap environment.

### 4.2. Path Optimization Simulation

In the path optimization experiments, complex Maze and Bug Trap environments were used to test the quality of the four algorithms' paths after pruning and bidirectional shrinking optimization. To assess the stability of the optimization strategy, the experiment was repeated 100 times. The quality characteristics of the generated paths were evaluated by comparing the average length and smoothness values. The path lengths were calculated using the Euclidean distance, which refers to the straight-line distance connecting two points on a plane, and can be computed using the Pythagorean theorem. Additionally, the path smoothness values were obtained by accumulating the turning angles of each path, measured in radians.

#### 4.2.1. Maze Environment Path Optimization

The purpose of the experiments in the Maze environment was to test the performance of the proposed optimization strategy. Figure 13 illustrates the results of pruning and bidirectional shrinking optimizations. It is evident that the original paths generated by all algorithms were convoluted and intricate. However, after the pruning optimization, the quality of the paths improved, but they were not optimal, while bidirectional shrinking optimization generated paths that were close to the optimal path. A detailed comparison of the paths is provided in Table 5. The path lengths of bidirectional shrinking optimization were the shortest, and the path smoothness values were the lowest, indicating that the generated paths were optimal. The paths generated by the four algorithms were all able to be optimized to approximate the optimal path. Hence, it can be concluded that the bidirectional shrinking path optimization strategy outperformed the pruning optimization strategy in complex environments.



**Figure 13.** Pruning and bidirectional shrinking optimization in the Maze environment. The blue lines indicate the original paths, the green lines indicate the pruned optimized paths, and the red lines indicate the bidirectional shrinking optimized paths.

**Table 5.** Results for Optimization in a Maze Environment.

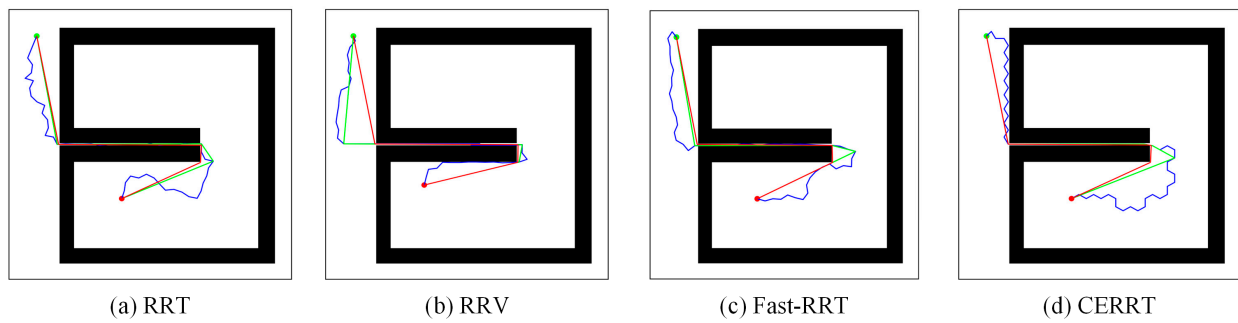
| Algorithm | Path Cost | POS Cost | BSOS Cost | Path Smoothness | POS Smoothness | BSOS Smoothness |
|-----------|-----------|----------|-----------|-----------------|----------------|-----------------|
| RRT       | 5346.20   | 4413.83  | 4076.84   | 78.24           | 12.20          | 11.16           |
| RRV       | 4838.35   | 4392.98  | 4056.15   | 33.77           | 12.50          | 11.31           |
| Fast-RRT  | 5426.26   | 4408.72  | 4062.23   | 65.32           | 11.78          | 11.24           |
| CERRT     | 6033.47   | 4397.60  | 4071.37   | 226.11          | 11.85          | 11.12           |

#### 4.2.2. Bug Trap Environment Path Optimization

The purpose of the experiments conducted in the Bug Trap environment was to test the performance of the proposed optimization strategies in narrow passage environments. Figure 14 displays the results of pruning and bidirectional contraction optimization. The pruned paths were not able to determine the optimal route, while the bidirectional contraction path generated the optimal paths close to the wall. A detailed comparison of the



paths is provided in Table 6. The paths generated by the four algorithms all had relatively low quality, and after pruning optimization, the path smoothness values were effectively improved, but the path lengths were not optimal. After bidirectional contraction path optimization, the paths had the lowest smoothness values and the shortest lengths and they were close to the optimal path. Therefore, it is evident that the bidirectional contraction path optimization strategy still outperformed the pruning optimization strategy in narrow environments and was applicable to all initial paths generated by sampling algorithms.



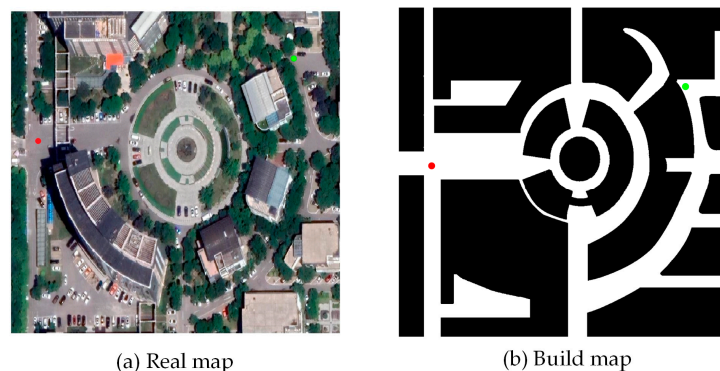
**Figure 14.** Pruning and bidirectional shrinking optimization in the Bug Trap environment. The blue lines represent the original paths, the green lines represent the pruned paths, and the red lines represent the paths optimized by bidirectional shrinking.

**Table 6.** Results for Optimization in the Bug Trap Environment.

| Algorithm | Path Cost | POS Cost | BSOS Cost | Path Smoothness | POS Smoothness | BSOS Smoothness |
|-----------|-----------|----------|-----------|-----------------|----------------|-----------------|
| RRT       | 1723.38   | 1386.87  | 1289.38   | 32.75           | 4.48           | 4.15            |
| RRV       | 1618.49   | 1383.94  | 1281.70   | 13.58           | 4.78           | 4.34            |
| Fast-RRT  | 1682.35   | 1375.54  | 1285.46   | 29.74           | 4.36           | 4.21            |
| CERRT     | 1619.61   | 1323.14  | 1283.38   | 55.53           | 4.08           | 4.05            |

### 4.3. Evaluation of Algorithms in Real Environment

In order to evaluate the performance of the algorithm, we have chosen an actual map scenario, which is Tianjin Central Square with geographical coordinates of  $117^{\circ}04'56.88''$  E,  $39^{\circ}05'49.19''$  N, as shown in Figure 15a. The map covers an area of  $175\text{ m} \times 175\text{ m}$  and is divided into a grid of  $1000\text{ px} \times 1000\text{ px}$ , where each pixel represents an actual area of  $0.03\text{ m} \times 0.03\text{ m}$ . Figure 15b depicts the map generated based on the real scene, where black represents the obstacle areas and white represents the free space.



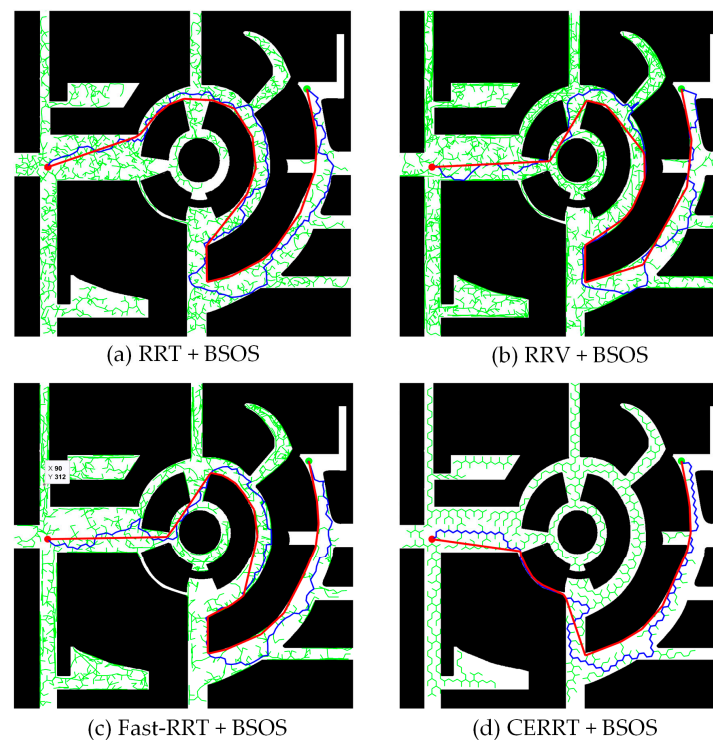
**Figure 15.** Actual map environment.

The coordinates of the starting point are marked with a red circle at  $[100, 480]$ , and the coordinates of the target point are marked with a green circle at  $[870, 240]$ . We will

perform path planning tasks 100 times for a mobile robot in this scenario and evaluate the algorithm's performance by taking the average.

#### Algorithm Comparison

In Figure 16, the path planning results of four algorithms in a real-world environment are depicted. Notably, a curved narrow passage shortcut can be observed on the map. All four algorithms can generate feasible paths for the mobile robot, but only CERRT is able to discover and navigate through the curved passage shortcut in the actual environment. The other algorithms struggle to solve the narrow passage problem in the real environment, further confirming the practical value of the proposed algorithm. Additionally, the path quality is significantly improved after optimizing with BSOS compared to the initial paths.



**Figure 16.** Performance comparison of four algorithms in real environment. The blue lines represent the original paths and the red lines represent the paths optimized by bidirectional shrinking.

Since RRT-based algorithms have probabilistic completeness, the success rate of all four algorithms in planning paths in the actual environment is 100% when the sampling limit is not restricted. Table 7 displays the performance parameters of these algorithms. The average runtime of RRT is approximately five times that of CERRT, RRV is approximately 50 times that of CERRT, and Fast-RRT is approximately seven times that of CERRT. CERRT has the shortest average runtime with a standard deviation of only 4.68, which is significantly lower than the other three algorithms, indicating excellent performance of the proposed algorithm in the actual environment.

**Table 7.** Results for planning in actual map.

| Algorithm | Avg. Time (ms) | Std    | Avg. Nodes | Path Cost | BSOS Cost |
|-----------|----------------|--------|------------|-----------|-----------|
| RRT       | 278.31         | 139.61 | 3126       | 2499.48   | 2104.93   |
| RRV       | 2782.19        | 435.89 | 3149       | 2446.17   | 2036.01   |
| Fast-RRT  | 365.72         | 138.94 | 2674       | 2545.87   | 2124.81   |
| CERRT     | 55.23          | 4.68   | 907        | 1990.01   | 1706.98   |

Regarding path optimization, a horizontal comparison reveals that the path lengths after applying BSOS are all smaller than the initial paths, confirming the practicality of BSOS. A vertical comparison shows that the path length generated by CERRT is smaller than the other three algorithms. This is because CERRT is able to quickly discover and navigate through the curved passage shortcut in the actual environment, while the other algorithms struggle to pass through the curved passage efficiently.

In conclusion, the CERRT algorithm outperforms the other three algorithms in a real-world environment. The main reason is that the expansion principle of CERRT is inspired by the hexagonal honeycomb structure found in nature. Reference [21] mentions that a hexagonal honeycomb provides the least-perimeter way to enclose and separate infinitely many regions of unit area, indicating that using a hexagonal expansion strategy can greatly improve the utilization of each tree node and thus enhance the algorithm's performance.

## 5. Conclusions

In this paper, we propose a sampling-based path planning algorithm, CERRT, which performs better than other algorithms in complex environments and effectively solves the narrow passage problem. CERRT consists of two important parts: the first part limits the selection and expansion of tree vertices to maximize the utilization of each vertex, while the second part involves environment perception, where vertices are sampled near obstacles to ensure feasible passages are discovered. By combining the vertex selection method of the first part with the sampling strategy of the second part, the algorithm avoids redundant and useless exploration near trap-type obstacles, significantly improving planning performance. Numerical simulations comparing the proposed algorithm with others validate its effectiveness. Since sampling-based algorithms generally generate lower-quality paths, we propose the BSOS strategy to optimize the initial paths. The suitability of the algorithm has been verified through path optimization for different sampling algorithms. The optimized paths produced by the BSOS algorithm were superior to those produced by the well-known pruning optimization strategy (POS). However, for the algorithm proposed in this paper, the number of locally sampled points in three-dimensional environments may increase dramatically, limiting the environmental perception capability. Therefore, the next step is to study the performance of the algorithm in high-dimensional environments.

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

**Funding:** This work was supported in part by the Chunhui Cooperation Program of the Ministry of Education, HZKY20220590-202200265; National Natural Science Foundation of China under Grant 61902273.

**Institutional Review Board Statement:** Not applicable.

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** No data were used for the research described in the article.

**Conflicts of Interest:** The authors declare no conflict of interest.

## References

1. Hichri, B.; Gallala, A.; Giovannini, F.; Kedziora, S. Mobile robots path planning and mobile multirobots control: A review. *Robotica* **2022**, *40*, 4257–4270. [[CrossRef](#)]
2. Öztürk, Ü.; Akdağ, M.; Ayabakan, T. A review of path planning algorithms in maritime autonomous surface ships: Navigation safety perspective. *Ocean. Eng.* **2022**, *251*, 111010. [[CrossRef](#)]
3. Wang, X.; Wei, J.; Zhou, X.; Xia, Z.; Gu, X. AEB-RRT\*: An adaptive extension bidirectional RRT\* algorithm. *Auton. Robot.* **2022**, *46*, 685–704.

4. Kuffner, J.J.; LaValle, S.M. RRT-connect: An efficient approach to single-query path planning. In *Proceedings of the Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065), San Francisco, CA, USA, 24–28 April 2000*; IEEE: Piscataway, NJ, USA, 2000; pp. 995–1001.
5. Tahirovic, A.; Ferizbegovic, M. Rapidly-exploring random vines (RRV) for motion planning in configuration spaces with narrow passages. In *Proceedings of the 2018 IEEE International Conference on Robotics and Automation (ICRA), Brisbane, QLD, Australia, 21–25 May 2018*; IEEE: Piscataway, NJ, USA, 2018; pp. 7055–7062.
6. Hsu, D.; Jiang, T.; Reif, J.; Sun, Z. The bridge test for sampling narrow passages with probabilistic roadmap planners. In *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (cat. no. 03CH37422), Taipei, Taiwan, 14–19 September 2003*; IEEE: Piscataway, NJ, USA, 2003; pp. 4420–4426.
7. Wu, Z.; Meng, Z.; Zhao, W.; Wu, Z. Fast-RRT: A RRT-Based Optimal Path Finding Method. *Appl. Sci.* **2021**, *11*, 11777. [[CrossRef](#)]
8. Cai, P.; Yue, X.; Zhang, H. ADD-RRV for motion planning in complex environments. *Robotica* **2022**, *40*, 136–153. [[CrossRef](#)]
9. Li, B.; Chen, B. An adaptive rapidly-exploring random tree. *IEEE/CAA J. Autom. Sin.* **2021**, *9*, 283–294. [[CrossRef](#)]
10. 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.* **2021**, *69*, 4926–4937. [[CrossRef](#)]
11. Taheri, E.; Ferdowsi, M.H.; Danesh, M. Fuzzy greedy RRT path planning algorithm in a complex configuration space. *Int. J. Control. Autom. Syst.* **2018**, *16*, 3026–3035. [[CrossRef](#)]
12. Karaman, S.; Frazzoli, E. Sampling-based algorithms for optimal motion planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894. [[CrossRef](#)]
13. Islam, F.; Nasir, J.; Malik, U.; Ayaz, Y.; Hasan, O. Rrt\*-smart: Rapid convergence implementation of rrt\* towards optimal solution. In *Proceedings of the 2012 IEEE International Conference on Mechatronics and Automation, Chengdu, China, 5–8 August 2012*; IEEE: Piscataway, NJ, USA, 2012; pp. 1651–1656.
14. 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 2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, 14–18 September 2014*; IEEE: Piscataway, NJ, USA, 2014; pp. 2997–3004.
15. Qureshi, A.H.; Ayaz, Y. Potential functions based sampling heuristic for optimal path planning. *Auton. Robot.* **2016**, *40*, 1079–1093. [[CrossRef](#)]
16. Jeong, I.-B.; Lee, S.-J.; Kim, J.-H. Quick-RRT\*: Triangular inequality-based implementation of RRT\* with improved initial solution and convergence rate. *Expert Syst. Appl.* **2019**, *123*, 82–90. [[CrossRef](#)]
17. Liao, B.; Wan, F.; Hua, Y.; Ma, R.; Zhu, S.; Qing, X. F-RRT\*: An improved path planning algorithm with improved initial solution and convergence rate. *Expert Syst. Appl.* **2021**, *184*, 115457. [[CrossRef](#)]
18. Qian, K.; Liu, Y.; Tian, L.; Bao, J. Robot path planning optimization method based on heuristic multi-directional rapidly-exploring tree. *Comput. Electr. Eng.* **2020**, *85*, 106688. [[CrossRef](#)]
19. Chen, Y.; Fu, Y.; Zhang, B.; Fu, W.; Shen, C. Path planning of the fruit tree pruning manipulator based on improved RRT-Connect algorithm. *Int. J. Agric. Biol. Eng.* **2022**, *15*, 177–188. [[CrossRef](#)]
20. Hao, K.; Zhao, J.; Wang, B.; Liu, Y.; Wang, C. The Application of an Adaptive Genetic Algorithm Based on Collision Detection in Path Planning of Mobile Robots. *Comput. Intell. Neurosci.* **2021**, *2021*, 5536574. [[CrossRef](#)]
21. Morgan, F. The hexagonal honeycomb conjecture. *Trans. Am. Math. Soc.* **1999**, *351*, 1753–1763. [[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.