

## Article

# Fast Path-Planning Algorithm for Mobile Robots via Straight Strategy

Jihong Jeong <sup>1</sup> and Jin-Woo Jung <sup>2,\*</sup>

<sup>1</sup> Department of Computer Science and Artificial Intelligence, Dongguk University, Seoul 04620, Republic of Korea; jeongjihong0303@naver.com

<sup>2</sup> Department of Computer Science and Engineering, Dongguk University, Seoul 04620, Republic of Korea

\* Correspondence: jwjung@dongguk.edu; Tel.: +82-2-2260-3812

## Abstract

RRT\*, which augments the RRT with the ChooseParent and Rewire steps, is a widely used algorithm for path planning. RRT\*-based algorithms are effective for improving the quality of paths in the Rewire step. However, as the expansion continues, there are more nodes to be inspected, which can slow down the path search. In addition, due to the parameters set in the rewire step, a trade-off issue between path quality and planning time arises. In this paper, we propose Straight-RRT to improve upon these limitations. Straight-RRT applies the Straight strategy to rapidly obtain an initial path and then returns a refined path using the MoveParent strategies. Accordingly, Straight-RRT adopts the following mechanisms. (1) The Straight strategy is applied for rapidly finding an initial path. This procedure quickly finds a feasible initial path. (2) MoveParent is a strategy inspired by parametric equations for path optimization. This complements the weaknesses of the Straight strategy and the limitations of the triangle inequality. The MoveParent strategy is applied bidirectionally. These procedures progressively refine the path and improve efficiency. We propose an algorithm that is faster than other algorithms using these strategies and minimizes the trade-off caused by parameter settings. In the experimental comparison results across most environments, our approach achieved shorter planning times than the compared baseline algorithms and produced paths of comparable quality.

**Keywords:** path planning; sampling based algorithms; rapidly explored random tree (RRT); triangular inequality; linear interpolation;

## 1. Introduction

Driven by rapid advances in science and technology, the era of human–robot coexistence is approaching. In everyday life, mobile robots can be found in various forms, including patrol, service, delivery, inspection, and agricultural robots. Accordingly, path planning for mobile robots is becoming increasingly important. For a mobile robot, path planning refers to computing a path from a start position to a goal position. The path must be collision-free and of as low a cost as possible. In addition, mobile robots must respond to user commands and environmental changes without delay, and they must be able to plan paths rapidly to cope with unforeseen dynamic obstacles during navigation. In essence, path planning must be able to generate efficient paths quickly.

Path planning can be broadly classified into classical, intelligent, and other approaches [1]. The classical category includes A\* search, artificial potential fields (APFs),

Received: 29 December 2025

Revised: 8 February 2026

Accepted: 12 February 2026

Published: 15 February 2026

**Copyright:** © 2026 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 \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

probabilistic roadmaps (PRMs), and rapidly exploring random trees (RRTs). The proposed algorithm is a variant of RRT, one of the classical path-planning approaches. RRT-based algorithms have several advantages.

Since RRT\* introduced the Rewire strategy, most variants have adopted a Rewire process because of its benefits. However, rewiring depends on the hyperparameter settings. As a result, it has the drawback of reduced generalization [2]. In addition, the Rewire strategy requires considering an increasing number of nodes as the tree grows, which causes the planning time to be delayed as time goes on. That is, the existing algorithm has limitations due to planning time delay caused by rewiring and due to hyperparameters. The planning time delay can be dangerous in environments where the robot must respond immediately. For example, if a fast-moving obstacle is approaching from the opposite side, the robot must be able to quickly plan a path to avoid it. In addition, there are many different environments, and if the hyperparameters must be set differently for each environment, generalization becomes weak.

Therefore, to address these issues, the objectives of this study are as follows. First, we achieve faster planning times than existing algorithms to enable rapid path generation in response to motion commands and unexpected events. Second, we minimize parameter-dependent trade-offs by eliminating the rewiring step. To achieve the above goals, we propose an RRT variant without rewiring that plans faster while producing paths of comparable quality. The proposed algorithm quickly computes a drivable path in the early stage. The path is then post-processed to shorten it. The existing method adds a node at a point that is a fixed distance from  $x_{\text{nearest}}$  in the direction of  $x_{\text{rand}}$ . However, if there is an obstacle in this process, the computational cost used for collision-checking and the computational cost used for sampling are wasted. Unlike the existing method, the proposed strategy finds the last collision-free node position toward  $x_{\text{rand}}$  direction. Then, it adds the midpoint between that position and  $x_{\text{nearest}}$  to the tree. One representative post-processing method used in existing approaches is based on the triangle inequality. However, it cannot be applied if there is even a very small obstacle between node [i] and node [i+2]. The proposed strategy uses a parametric equation such that, if a direct connection from node [i] to node [i+2] is not possible, it updates the position of node [i+1] to a location as close as possible to node [i+2]. With this strategy, we aim to reduce reliance on rewiring in prior studies and to generate a comparable or better path while computing it more efficiently.

To evaluate the validity of the proposed algorithm, we set “length of path,” “cumulative heading change,” and “planning time” as the evaluation criteria. Length of path is an indicator used to determine how short the travel distance is. Cumulative heading change represents the total amount of direction change required during the robot’s path-following process, thereby indirectly reflecting the burden of rotational motion. Planning time is an indicator used to assess the feasibility of a real-time application in an environment with limited computation time. Based on the experimental results, it was overall superior to all comparison algorithms. Although the path quality was slightly lower than that of other algorithms, the planning time was noticeably reduced. That is, it shows a reduced planning time while maintaining similar path quality, even as it alleviates parameter dependence and removes the rewiring process. This means that, when applied to a mobile robot, it can respond to user commands or environmental changes with much less delay than other algorithms.

The remainder of this paper is organized as follows. Section 2 introduces related work on existing RRT-based algorithms. Section 3 describes the mechanisms of the comparison algorithms, Q-RRT\*, F-RRT\*, and MQ-RRT\*, the reasons for selecting these algorithms, and the limitations of each algorithm. Section 4 presents the core of the proposed

algorithm, focusing on the mechanisms of the Straight strategy and the MoveParent strategy. Section 5 presents experiments conducted using two scenarios across three different environments. Specifically, it describes the comparative performance evaluation and results of the algorithms based on three evaluation metrics. Section 6 discusses the conclusion, the limitations of the study, and future directions.

## 2. Related Works

In this section, we introduce several variants of RRT-based algorithms. Compared with many path planners, RRT offers key advantages, such as rapid exploration, robustness in obstacle-dense environments, and efficiency in high-dimensional spaces [3]. Despite these advantages, RRT often yields high-cost, non-smooth paths. Consequently, there has been active research aimed at improving path quality and reducing cost.

In RRT-Connect, once a tree creates  $x_{new}$ , the opposite tree repeatedly extends toward it via Connect, which typically yields faster path-finding than the standard RRT [4]. In RRT-Connect, the extend strategy grows toward  $x_{rand}$ , and when an obstacle is detected, it stops the tree expansion without adding the node. It then performs another sampling step and continues expanding the tree by extending again toward the newly sampled  $x_{new}$ . Encountering an obstacle or wall during this expansion causes the growth toward the sampled node to terminate, after which the algorithm resamples and resumes tree expansion. This repeated stopping and resampling process results in wasted computational cost. The connect strategy can rapidly meet the tree on the opposite side because it extends as far as possible only in the direction of  $x_{new}$ . However, this process can reduce exploration capability, leading to search bias and, consequently, degrading space coverage. To mitigate the computational cost issues of the extend strategy and the reduced exploration of the connect strategy, the proposed strategy adds the midpoint to the tree after extending as far as possible. In RRT-Connect, after one tree expands, the opposite tree executes the connect strategy. During this process, it repeatedly attempts to reach  $x_{new}$  by extending toward it until it is reached or trapped. In contrast, the proposed strategy checks connectivity only at regular intervals for the expanding tree. This approach can reduce computational cost by avoiding repeated nearest-neighbor searches and collision-checking steps with the identified nearby node.

An Anytime RRTs approach has been proposed that first rapidly finds an executable path and then incrementally improves its cost within the permitted time. This yields paths of higher quality than the standard RRT [5]. Similarly, the RRT-Back method first finds a path and applies shortcutting based on the triangle inequality [6]. This applies the triangle inequality to the completed path starting from the goal position. The proposed MoveParent strategy using a parametric equation is performed bidirectionally, and it can also be applied to segments where the triangle inequality cannot optimize the path.

RRT\* was proposed, applying the ChooseParent and Rewire strategies to RRT to lower the path cost. It has been shown to converge to the optimal path as the number of samples increases. This addresses the limitation of RRT, which does not guarantee convergence to an optimal solution [7]. However, although RRT\* guarantees optimality, it often suffers from long planning times. Consequently, numerous RRT\*-based variants have been proposed to improve path quality and shorten the planning time.

When the robot is moving, RT-RRT\* improves real-time performance by adopting an online tree-rewiring strategy that relocates the tree's root along with the robot [8]. AM-RRT\*, which extends RT-RRT\* by employing a diffusion-distance-based assisting metric to reduce path cost and planning time, was proposed [9]. Fast AM-RRT\* extends AM-RRT\* by adding distance and directionality weighting and adopting a weight-based node selection strategy and bidirectional tree expansion. This yields faster planning [10]. The method applies Ant Colony Optimization (ACO) to maintain an ant table that preserves

promising samples and to generate the next samples from a Gaussian mixture model. This adaptively refines the sampling distribution [11]. To address planning time slowdowns and excessive node generation in RRT and RRT\*, a strategy was proposed that guides tree growth using intermediate guide nodes derived from the Generalized Voronoi Diagram (GVD) [12]. To improve RRT's low sampling efficiency, GEO-RRT\* was proposed [13]. It combines explored-region rejection with obstacle-information-based expansion to reduce unnecessary sampling and expansion failures, thereby accelerating exploration and improving path quality. Direct-DRRT\* applies a direct heuristic after the Rewire step to link the nearest node to the sampled node and the sampled node to the goal with straight lines when both segments are collision-free [14]. However, this method relies on rewiring. Accordingly, it is difficult to address the time delay that increases as the tree grows and the parameter dependence. Attempts have also been made to extend Direct-DRRT\* from 2D to 3D [15]. Informed-RRT\* was proposed [16]. It first quickly finds a path using RRT and then samples only within an admissible region to optimize the path.

RRT\*-Smart was proposed [17]. It uses intelligent sampling and the triangle inequality to generate paths rapidly. RRT\*-Smart showed that the triangle inequality can optimize paths quickly and effectively. After that, several algorithms based on the triangle inequality were proposed. Improved RRT-Connect was proposed. It introduces a rewiring strategy that applies the triangle inequality [18]. Q-RRT\* was proposed [19]. During ChooseParent, it considers the ancestry of neighboring nodes to select a parent. F-RRT\* was proposed, mentioning the drawbacks of the ChooseParent step and introducing the FindReachest and CreateNode strategies to improve it [20]. MQ-RRT\* was proposed to improve sampling efficiency by introducing a sparse sampling strategy and a dynamically goal-biased strategy using a sigmoid function and adopting the RemoveTips and CreateNode strategies [21]. FHQ-RRT\* was proposed as an extension of MQ-RRT\*, assuming the availability of sensor-preprocessed information for  $c_{obs}$  and  $c_{free}$  [22]. It dynamically changes the sparse sampling, and it introduces a NewRewire strategy that excludes co-origin nodes from the optimization during the Rewire process. A summary of the principal algorithms is provided in Table 1.

**Table 1.** Overview of major RRT-based algorithms.

| Algorithms              | Principle                                                                                                                                                                                                                                                                                                                                                                                                                                                                        |
|-------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| RRT*-Smart              | After finding an initial feasible path, the path is optimized using triangle inequality-based PathOptimization. Then, path optimization is carried out by repeatedly performing IntelligentSample using the nodes of the optimized path as biasing points.                                                                                                                                                                                                                       |
| Q-RRT*                  | We expand the parent candidate set beyond the RRT* near set ( $X_{near}$ ) by including the ancestry of nearby vertices. This enables improved parent selection. The same expansion is applied to the rewiring.                                                                                                                                                                                                                                                                  |
| F-RRT*                  | This algorithm is intended to reduce the impact of $r_{near}$ . For tree expansion, when adding $x_{rand}$ , it introduces FindReachest (finding a reachable parent via ancestor search) instead of $r_{near}$ . In addition, it generates the parent of $x_{rand}$ using a bisection-based CreateNode strategy. However, this process introduces an additional parameter, $D_{dichotomy}$ .                                                                                     |
| MQ-RRT*                 | Planning time performance was improved through sparse sampling and dynamic goal bias. A bisection-based CreateNode and RemoveTips were introduced to generate new parent nodes. As a result, the method reduces path cost and improves smoothness. However, an additional parameter $R_{commit}$ is introduced, and performance varies depending on its value.                                                                                                                   |
| Straight-RRT (Proposed) | The Straight strategy does not use a rewiring process during the tree expansion stage, thereby generating an initial feasible path very quickly. This reduces dependence on nearby-neighbor search parameters, such as $r_{near}$ , which affect performance and computational cost in RRT*-based methods, and alleviates the issue that the computational burden increases over time due to repeated neighbor searches and collision checks. Subsequently, the MoveParent post- |

processing improves the path by reconstructing node connections using the principle of a parametric equation, effectively enhancing the path length.

### 3. Overview of Comparator Algorithms

This section provides a brief review of the comparator algorithms Q-RRT\*, F-RRT\*, and MQ-RRT\*. We selected these algorithms because they commonly refine the path based on the triangle inequality, which is known to reduce planning time and improve path quality compared with standard RRT\*. The notation used throughout the paper is summarized in Table 2.

**Table 2.** The definition of notations.

| Notation        | Description                                                                   |
|-----------------|-------------------------------------------------------------------------------|
| $x_{start}$     | Start position                                                                |
| $x_{goal}$      | Goal position                                                                 |
| $C_{free}$      | The collision-free region of the configuration space with respect to the map. |
| $C_{obs}$       | The obstacle region of the configuration space with respect to the map        |
| $T_{limit}$     | Limit path planning time                                                      |
| $l_{cc}$        | Collision-check step size                                                     |
| $x_{rand}$      | Randomly sampled configuration in the configuration space                     |
| $x_{new}$       | New node (steered from $x_{nearest}$ toward $x_{rand}$ )                      |
| $x_{nearest}$   | The nearest tree node to $x_{rand}$                                           |
| $r_{near}$      | Rewiring radius                                                               |
| $X_{near}$      | The set of nodes within distance $r_{near}$ of $x_{new}$                      |
| $D_{ancestry}$  | Depth parameter when use in Q-RRT*                                            |
| $D_{dichotomy}$ | Dichotomy parameter used in F-RRT*                                            |
| $R_{commit}$    | Sparse-sampling commit radius used in MQ-RRT*                                 |

#### 3.1. Q-RRT\*

Q-RRT\* was proposed to address the trade-off between planning time and path quality induced by the rewiring radius  $r_{near}$  in RRT\*. The key idea is to expand the candidate parent set in the ChooseParent and Rewire steps from only  $X_{near}$  to include their ancestors as well, thereby reducing overall planning time while maintaining solution quality. The Q-RRT\* pseudocode is presented in Algorithms 1 and 2.

#### Algorithm 1. Pseudocode of the Q-RRT\* algorithm.

**Input:**  $x_{start}$ ,  $x_{goal}$ ,  $C_{free}$ ,  $C_{obs}$ ,  $T_{limit}$ ,  $r_{near}$ ,  $D_{ancestry}$

**Output:** Tree

**Initialize:** Tree  $\leftarrow$  Null Tree,  $(T_{now}, T_{start}) \leftarrow$  Initialize

**Begin Q-RRT\* Procedure**

```

1 Tree  $\leftarrow$  Insert Root Node  $x_{start}$ 
2 While  $T_{now} - T_{start} \leq T_{limit}$  do
3    $x_{rand} \leftarrow$  RandomSampling()
4    $x_{nearest} \leftarrow$  Nearest( $x_{rand}$ )
5    $x_{new} \leftarrow$  Steer( $x_{nearest}$ ,  $x_{rand}$ )
6   If CollisionFree( $x_{nearest}$ ,  $x_{new}$ ) then
7      $X_{near} \leftarrow$  Near( $x_{new}$ ,  $r_{near}$ )
8      $X_{parent} \leftarrow$  Ancestry( $X_{near}$ ,  $D_{ancestry}$ )
9      $x_{parent} \leftarrow$  ChooseParent( $X_{near} \cup X_{parent}$ ,  $x_{nearest}$ ,  $x_{new}$ )

```

---

```

10   Tree  $\leftarrow$  Connect(Tree,  $x_{parent}$ ,  $x_{new}$ )
11   Tree  $\leftarrow$  Rewire-Q-RRT*(Tree,  $x_{new}$ ,  $X_{near}$ )
12   If Distance( $x_{new}$ ,  $x_{goal}$ ) <  $goal_{threshold}$  then
13     Return Tree
14    $T_{now} \leftarrow UpdateTime()$ 
End Q-RRT*

```

---



---

**Algorithm 2.** Pseudocode of the Rewire-Q-RRT\*.

---

**Input:** Tree,  $x_{new}$ ,  $X_{near}$ 
**Output:** Tree

**Begin Rewire-Q-RRT\* Procedure**

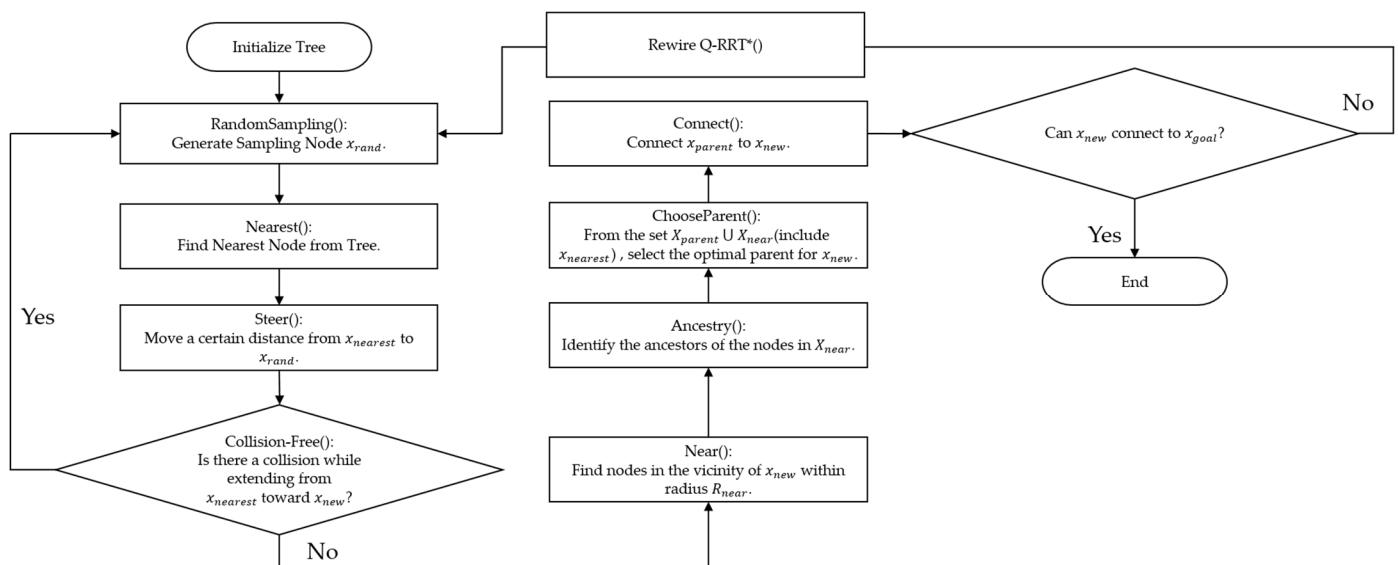
```

1  for  $x_{near}$  in  $X_{near}$ 
2    If Cost( $x_{new}$ ) + Distance( $x_{new}$ ,  $x_{near}$ ) < Cost( $x_{near}$ ) then
3      If CollisionFree( $x_{new}$ ,  $x_{near}$ ) then
4        Tree  $\leftarrow$  Reconnect(Tree,  $x_{new}$ ,  $x_{near}$ )
5  Return Tree
End Rewire-Q-RRT* from Q-RRT*

```

---

In RRT\*, the ChooseParent step considers only  $X_{near}$  when selecting a parent for  $x_{new}$ . In contrast, Q-RRT\* considers not only  $X_{near}$  but also their ancestors, with the maximum ancestry depth determined by  $D_{ancestry}$ . Once the candidate parent set  $X_{parent}$  has been obtained through the preceding steps,  $x_{new}$  selects the optimal parent  $x_{parent} \in X_{parent}$  and performs connect. Subsequently, the Rewire step is performed on the neighborhood. For each node in  $X_{near}$ , if a lower-cost collision-free connection via  $x_{new}$  or any of  $x_{new}$ 's ancestors exists, it updates its parent using Reconnect. We selected Q-RRT\* as the baseline because it substantially mitigates the limitations of RRT\*. An overview of the overall algorithmic pipeline is provided in the flowchart in Figure 1.



**Figure 1.** Flowchart of the Q-RRT\* algorithm.

### 3.2. F-RRT\*

F-RRT\* reduces the trade-off caused by  $r_{near}$ . It replaces the  $r_{near}$ -based ChooseParent step with two strategies called FindReachest and CreateNode. It aims to optimize path quality by generating nodes near obstacle boundaries. The F-RRT\* pseudocode is presented in Algorithms 3 and 4.

---

#### Algorithm 3. Pseudocode of the F-RRT\* algorithm.

---

**Input:**  $x_{start}, x_{goal}, C_{Free}, C_{Obs}, T_{limit}, r_{near}, D_{dichotomy}$

**Output:** Tree

**Initialize:** Tree  $\leftarrow$  Null Tree,  $(T_{now}, T_{start}) \leftarrow$  Initialize

**Begin F-RRT\* Procedure**

```

1  Tree  $\leftarrow$  Insert Root Node  $x_{start}$ 
2  While  $T_{now} - T_{start} \leq T_{limit}$  do
3     $x_{rand} \leftarrow$  RandomSampling()
4     $x_{nearest} \leftarrow$  Nearest( $x_{rand}$ )
5    If CollisionFree( $x_{nearest}, x_{rand}$ ) then
6       $x_{near} \leftarrow$  Near( $x_{rand}, r_{near}$ )
7       $x_{reachest} \leftarrow$  FindReachest(Tree,  $x_{nearest}, x_{rand}$ )
8       $x_{create} \leftarrow$  CreateNode(Tree,  $x_{reachest}, x_{rand}, D_{dichotomy}$ )
9      If  $x_{create}$  is not Null then
10       Tree.InsertNode( $x_{create}, x_{rand}$ )
11    Else
12       Tree.InsertNode( $x_{rand}$ )
13    If Distance( $x_{new}, x_{goal}$ )  $< goal_{threshold}$  then
14       Return Tree
15    Rewire(Tree,  $x_{rand}, x_{near}$ )
16   $T_{now} \leftarrow$  UpdateTime()
End F-RRT*

```

---



---

#### Algorithm 4. Pseudocode of the CreateNode.

---

**Input:** Tree,  $x_{reachest}, x_{rand}, D_{dichotomy}$

**Output:**  $x_{create}$

**Begin CreateNode Procedure**

```

1   $x_{allow} \leftarrow x_{reachest}$ 
2  If  $x_{allow}$  is not  $x_{start}$  then
3     $x_{forbid} \leftarrow$  Parent( $x_{reachest}$ )
4    While Distance( $x_{allow}, x_{forbid}$ )  $> D_{dichotomy}$  do
5       $x_{mid} \leftarrow$  MiddleNode( $x_{allow}, x_{forbid}$ )
6      If CollisionFree( $x_{rand}, x_{mid}$ ) then
7         $x_{allow} \leftarrow x_{mid}$ 
8      Else
9         $x_{forbid} \leftarrow x_{mid}$ 
10      $x_{forbid} \leftarrow x_{rand}$ 
11  While Distance( $x_{allow}, x_{forbid}$ )  $> D_{dichotomy}$  do

```

---

```

12    $x_{mid} \leftarrow \text{MiddleNode}(x_{allow}, x_{forbid})$ 
13   If CollisionFree( $x_{mid}$ , Parent( $x_{reachest}$ )) then
14        $x_{allow} \leftarrow x_{mid}$ 
15   Else
16        $x_{forbid} \leftarrow x_{mid}$ 
17   If  $x_{allow}$  is not  $x_{reachest}$  then
18        $x_{create} \leftarrow x_{allow}$ 
19   Else
20        $x_{create}$  is Null
21   Return  $x_{create}$ 
End CreateNode

```

FindReachest find  $x_{nearest}$ , the nearest neighbor of  $x_{rand}$  in the tree. It then ascends the ancestor chain of  $x_{nearest}$  while the straight-line segment to  $x_{rand}$  remains collision-free. The highest ancestor reachable under this condition is returned and denoted as  $x_{reachest}$ .

CreateNode locally optimizes the subpath Parent( $x_{reachest}$ )  $\rightarrow x_{reachest} \rightarrow x_{rand}$ . This step sets the upper bound of  $x_{create}$  to  $x_{reachest}$  and, in turn, assigns the lower bound to Parent( $x_{reachest}$ ) and  $x_{rand}$ . And a binary search is applied until the distance between the upper and lower bounds is below  $d_{dichotomy}$ . If the interval is less than  $d_{dichotomy}$ , it assigns the upper bound to  $x_{create}$ . Then the tree connects  $x_{reachest}$  to  $x_{create}$  and  $x_{create}$  to  $x_{rand}$ . And if  $x_{goal}$  is not reached, it performs Rewire on  $x_{near}$ . Although CreateNode boosts optimization, both sensitivity to  $d_{dichotomy}$  and the trade-off remain [23]. In other words, attempts to reduce dependence on  $r_{near}$  resulted in a  $d_{dichotomy}$  induced speed-quality trade-off. F-RRT\* was selected as a comparison algorithm because it is faster than RRT\* and produces higher-quality solutions. An overview of the overall algorithmic pipeline is provided in the flowchart in Figure 2.

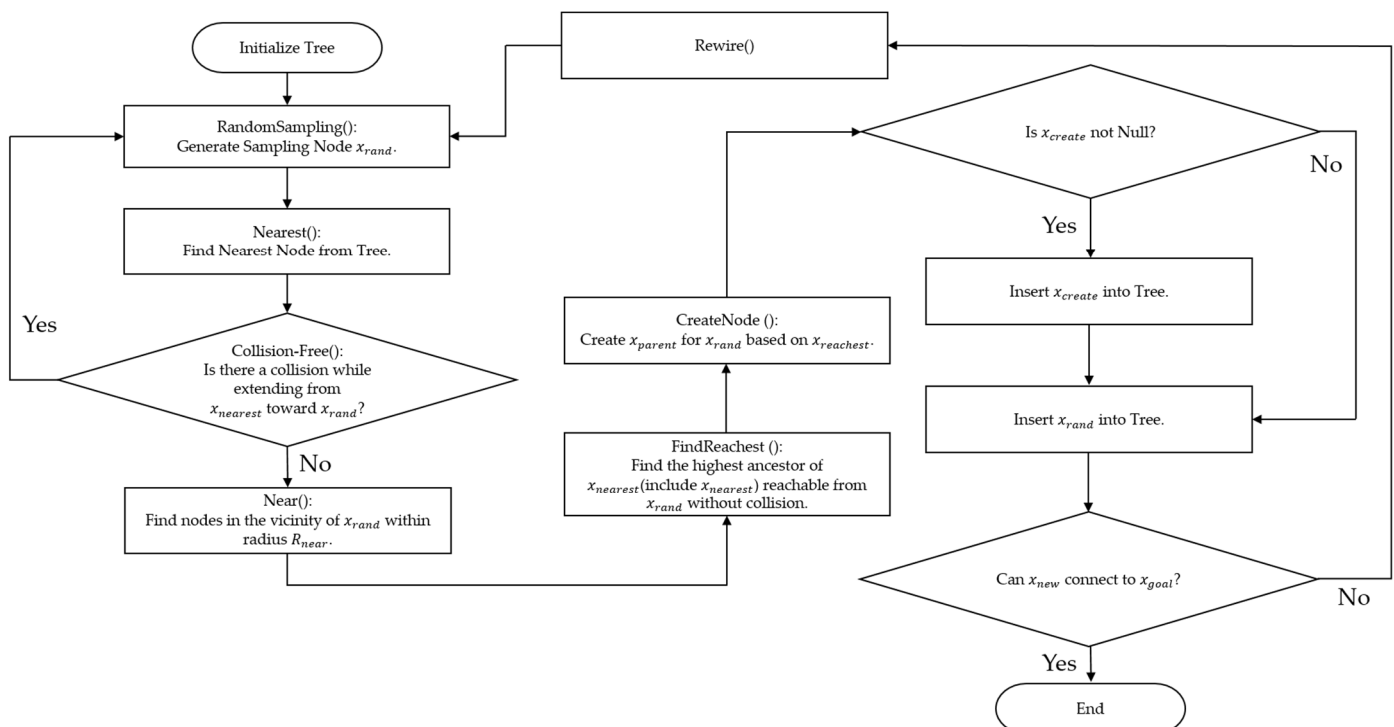


Figure 2. Flowchart of the F-RRT\* algorithm.

### 3.3. MQ-RRT\*

MQ-RRT\* introduces sparse sampling, a sigmoid function-based dynamic goal bias, RemoveTips, and CreateNode, and yields significant speed improvements over the two earlier algorithms. Sparse sampling is a strategy that discards  $x_{rand}$  if the tree already has a node within an  $r_{commit}$  radius of  $x_{rand}$ . This enables more efficient exploration of the search space. Although sparse sampling contributes to much faster performance than the two preceding algorithms, its performance can degrade in narrow passages depending on the choice of  $R_{commit}$ . Additionally, the goal bias is adjusted dynamically using a sigmoid function and  $n_1$ .  $n_1$  indicates whether the region is obstacle-dense.  $n_1$  increases when  $x_{rand}$  is collision-free, and accordingly, the output of the sigmoid function increases. This, in turn, raises the probability of sampling  $x_{goal}$  in the next sampling step. Conversely, when collisions are frequent,  $n_1$  decreases, the sigmoid function returns a lower value, and the probability of sampling  $x_{goal}$  decreases.

RemoveTips applies the principle of dichotomy to smooth sharp points along the path  $\text{Parent}(x_{parent}) \rightarrow x_{parent} \rightarrow x_{rand}$ . After removing the sharp points and obtaining  $x_{cornerA}$  and  $x_{cornerB}$ , it further applies the principle of dichotomy in the CreateNodes procedure to produce a more optimized path that stays close to obstacles. If successful, we obtain the path  $(\text{parent}(x_{parent}), x_{createA}, x_{createB}, x_{rand})$  instead of the direct  $(\text{parent}(x_{parent}), x_{parent})$  segment. Because it computes paths much faster than RRT\* by applying sparse sampling and a dynamic goal bias, we selected MQ-RRT\* as a comparison algorithm. An overview of the overall algorithmic pipeline is provided in the flowchart in Figure 3. The MQ-RRT\* pseudocode is presented in Algorithm 5.

---

#### Algorithm 5. Pseudocode of the MQ-RRT\*.

---

**Input:**  $x_{start}, x_{goal}, C_{Free}, C_{Obs}, T_{limit}, r_{near}, R_{commit}, D_{dichotomy}, D_{ancestor}$

**Output:** Tree

**Initialize:** Tree  $\leftarrow$  Null Tree

$n_1 \leftarrow 0$

IterCount  $\leftarrow 0$

Sigmoid( $n_1$ )  $\leftarrow 1/(1 + \exp(-(n_1 - 5)))$

$(T_{now}, T_{start}) \leftarrow \text{Initialize}$

**Begin MQ-RRT\* Procedure**

1 Tree  $\leftarrow$  Insert Root Node  $x_{start}$

2 **While**  $T_{now} - T_{start} \leq T_{limit}$  **do**

3   **If** Rand() < Sigmoid( $n_1$ ) **then**

4      $x_{rand} \leftarrow x_{goal}$

5   **Else**

6      $x_{rand} \leftarrow \text{SparseSampling}(\text{Tree}, R_{commit})$

7   IterCount += 1

8   **If** IterCount mod 10 is zero **then**

9      $N_1 \leftarrow 0$

10     $x_{nearest} \leftarrow \text{Nearest}(\text{Tree}, x_{rand})$

11   **If** CollisionFree( $x_{rand}, x_{nearest}$ ) **then**

12      $N_1 += 1$

13      $x_{near} \leftarrow \text{Near}(x_{rand}, r_{near})$

14      $x_{parent} \leftarrow \text{Ancestry}(\text{Tree}, x_{near}, D_{ancestor})$

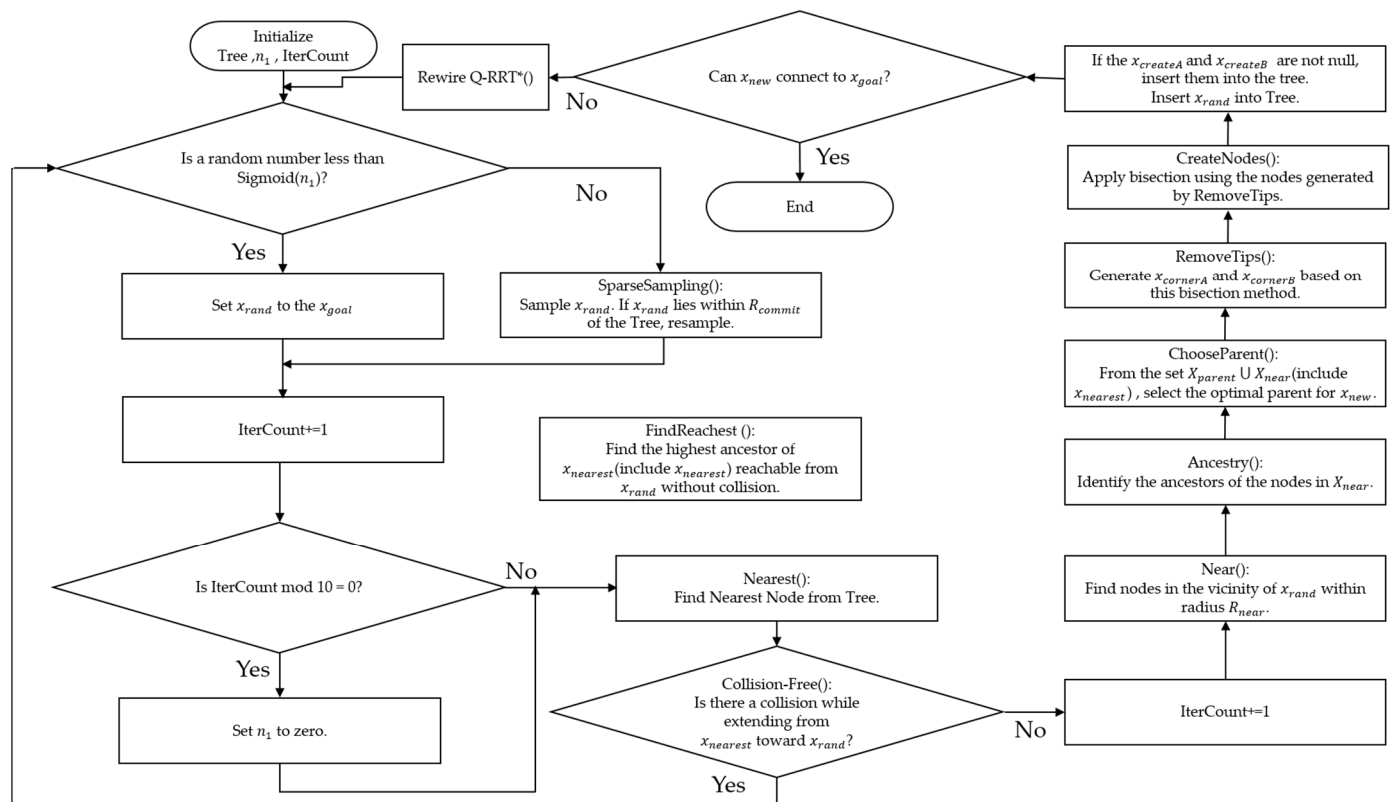
15      $x_{parent} \leftarrow \text{ChooseParent}(x_{near} \cup x_{parent}, x_{nearest}, x_{rand})$

---

```

16   ( $x_{cornerA}, x_{cornerB}$ )  $\leftarrow$  RemoveTips(Tree,  $x_{parent}, x_{rand}, D_{dichotomy}$ )
17   ( $x_{createA}, x_{createB}$ )  $\leftarrow$  CreateNodes(Tree,  $x_{parent}, x_{rand}, x_{cornerA}, x_{cornerB},$ 
     $D_{dichotomy}$ )
18   If ( $x_{createA}$  is not Null) and ( $x_{createB}$  is not Null) then
19       Tree.InsertNode(Parent( $x_{parent}$ ),  $x_{createA}$ )
20       Tree.InsertNode( $x_{createA}, x_{createB}$ )
21       Tree.InsertNode( $x_{createB}, x_{rand}$ )
22   Else
23       Tree.InsertNode( $x_{parent}, x_{rand}$ )
24   If Distance( $x_{rand}, x_{goal}$ ) <  $goal_{threshold}$  then
25       Return Tree
26   Else
27       Tree  $\leftarrow$  Rewire-Q-RRT*(Tree,  $x_{rand}, X_{near}$ )
28    $T_{now} \leftarrow UpdateTime()$ 
End MQ-RRT*

```



**Figure 3.** Flowchart of the MQ-RRT\* algorithm.

#### 4. Proposed Algorithm: Straight-RRT

In this section, we introduce the proposed Straight-RRT algorithm. The design goal of the algorithm is to achieve results that are as good as or better in path quality than those of other algorithms while being faster. The key idea of Straight-RRT is based on three observations. First, if the tree is expanded while being optimized through a process such as rewiring, it takes a long time to compute a feasible path. In other words, there is a problem in that the path cannot be computed when the path must be computed within a very short time. Second, existing algorithms have parameters that the user must set, such

as  $r_{\text{near}}$  and  $D_{\text{ancestor}}$ , when the tree is expanded, and accordingly, their computational speed and quality are affected by these parameter values. Third, the triangle inequality is efficient but cannot be applied if there is a very small obstacle at a point on the straight line between  $x_{\text{new}}$  and  $\text{Parent}(x_{\text{parent}})$ . To complement this, we applied the following strategies. First, the Straight strategy finds a feasible path faster than other algorithms. Second, the MoveParent strategy complements the triangle-inequality strategy, which cannot be applied if there is even a very small obstacle between a node and its ancestor node. It was inspired by parametric equations.

The proposed algorithm excludes the Rewire process and accordingly minimizes dependence on parameters. However, since there is no rewiring, the problem remains that it is difficult to find an efficient path. Therefore, a path is quickly found using the Straight strategy, and the MoveParent strategy is applied bidirectionally to optimize the overall path. The proposed algorithm finds a good-quality path faster than other algorithms by using these strategies. The overall workflow of the algorithm is shown in Figure 4. The trees are alternately expanded using the Straight strategy. If the two trees can be connected, the MoveParent strategy is executed once from the start direction of the path and then once more from the goal direction. The overall procedure of the Straight strategy is shown in Figure 5. The Straight strategy returns whether the trees have been connected. The overall procedure of the MoveParent strategy is shown in Figure 6. MoveParent returns the optimized path.

The overall workflow of the algorithm is shown in Figure 4. The trees are alternately expanded using the Straight strategy. If the two trees can be connected, the MoveParent strategy is executed once from the start direction of the path and then once more from the goal direction. The overall procedure of the Straight strategy is shown in Figure 5. The Straight strategy returns whether the trees have been connected. The overall procedure of the MoveParent strategy is shown in Figure 6. MoveParent returns the optimized path.

The pseudocode of the Straight-RRT algorithm is given in Algorithm 6. In Algorithm 6, the RandomSampling function samples a node uniformly at random and returns  $x_{\text{rand}}$ . The Nearest function finds the node in the given tree that is closest to  $x_{\text{rand}}$ . Straight and MoveParent are described in Algorithms 7 and 8, respectively. The Reconstruct function returns a complete path from the connected trees.

---

**Algorithm 6.** Pseudocode of the Straight-RRT algorithm.

---

**Input:**  $x_{\text{start}}, x_{\text{goal}}, C_{\text{free}}, C_{\text{obs}}, T_{\text{limit}}, D_{\text{connect}}, t_{\text{incre}}$

**Output:**

Path  $\leftarrow$  Result of Algorithm

**Initialize:**

Tree<sub>start</sub>  $\leftarrow$  Null Tree

Tree<sub>goal</sub>  $\leftarrow$  Null Tree

ExpandTree<sub>start</sub>  $\leftarrow$  True

( $T_{\text{now}}, T_{\text{start}}$ )  $\leftarrow$  Initialize

**Begin Straight RRT Procedure**

1 Tree<sub>start</sub>  $\leftarrow$  Insert Root Node  $x_{\text{start}}$  to Tree<sub>start</sub>

2 Tree<sub>goal</sub>  $\leftarrow$  Insert Root Node  $x_{\text{goal}}$  to Tree<sub>goal</sub>

3 **While**  $T_{\text{now}} - T_{\text{start}} \leq T_{\text{limit}}$  **do**

4    $x_{\text{rand}} \leftarrow \text{RandomSampling}()$

5   **If** ExpandTree<sub>start</sub> **then**

6      $x_{\text{nearest}} \leftarrow \text{Nearest}(x_{\text{rand}}, \text{Tree}_{\text{start}})$

---

---

```

7   Else
8        $x_{nearest} \leftarrow \text{Nearest}(x_{rand}, \text{Tree}_{goal})$ 
9   isConnect  $\leftarrow \text{Straight}(x_{rand}, x_{nearest}, D_{connect}, \text{ExpandTree}_{start})$ 
10  If isConnect then
11      Path  $\leftarrow \text{Reconstruct}_{path}(\text{Tree}_{start}, \text{Tree}_{goal})$ 
12      MoveParent(Path,  $t_{incre}$ )
13      Path  $\leftarrow \text{Path.Reverse}()$ 
14      MoveParent(Path,  $t_{incre}$ )
15      Path  $\leftarrow \text{Path.Reverse}()$ 
16      Return Path
17  ExpandTreestart  $\leftarrow \text{Not ExpandTree}_{start}$ 
18   $T_{now} \leftarrow \text{UpdateTime}()$ 
19  End While
End Straight RRT

```

---



---

**Algorithm 7.** Pseudocode of Straight.

---

**Input:**  $x_{start}, x_{target}, D_{connect}, \text{ExpandTree}_{start}$ 
**Output:** isConnect

**Initialize:**
 $\theta \leftarrow \text{atan2}((x_{target} - x_{start})_y, (x_{target} - x_{start})_x)$ 

distance  $\leftarrow 0$ 
**Begin Straight Procedure**

1  $last_x \leftarrow (x_{start})_x$ 

2  $last_y \leftarrow (x_{start})_y$ 

3 **While**  $(last_x, last_y) \in C_{free}$  **do**

4     distance  $\leftarrow distance + l_{cc}$ 

5      $tmp_x \leftarrow (x_{start})_x + \text{distance} \times \cos(\theta)$ 

6      $tmp_y \leftarrow (x_{start})_y + \text{distance} \times \sin(\theta)$ 

7     **If not** CollisionFree( $(last_x, last_y), (tmp_x, tmp_y)$ ) **then**

8         **Break**

9     **If**  $\text{int}(\text{distance}) \bmod D_{connect} = 0$  **then**

10         **If** ExpandTree<sub>start</sub> **then**

11              $x_{nearest} \leftarrow \text{NearestNodeInOpposite}(\text{Tree}_{goal}, (tmp_x, tmp_y), D_{connect})$ 

12             **If**  $x_{nearest}$  **is not** Null **then**

13                 **If** CollisionFree( $(tmp_x, tmp_y), x_{nearest}$ ) **then**

14                      $x_{new} \leftarrow (tmp_x, tmp_y)$ 

15                     Tree<sub>goal</sub>  $\leftarrow \text{Connect}(\text{Tree}_{goal}, x_{start}, x_{new})$ 

16                     **Return True**

17         **Else**

18              $x_{nearest} \leftarrow \text{NearestNodeInOpposite}(\text{Tree}_{start}, (tmp_x, tmp_y), D_{connect})$ 

19             **If**  $x_{nearest}$  **is not** Null **then**

20                 **If** CollisionFree( $(tmp_x, tmp_y), x_{nearest}$ ) **then**

21                      $x_{new} \leftarrow (tmp_x, tmp_y)$ 

22                     Tree<sub>start</sub>  $\leftarrow \text{Connect}(\text{Tree}_{start}, x_{start}, x_{new})$

---

```

23      Return True
24  End While
25   $last_x \leftarrow tmp_x$ 
26   $last_y \leftarrow tmp_y$ 
27   $x_{new} \leftarrow \text{MiddleNode}(last_x, last_y)$ 
28  If ExpandTreestart then
29    Treestart  $\leftarrow \text{Connect}(\text{Tree}_{start}, x_{start}, x_{new})$ 
30  Else
31    Treegoal  $\leftarrow \text{Connect}(\text{Tree}_{goal}, x_{start}, x_{new})$ 
32  Return False
End Straight

```

---



---

**Algorithm 8.** Pseudocode of the MoveParent.

---

**Input:** Path,  $t_{incre}$   
**Output:**  $\mathcal{P}_{refined}$   
**Initialize:**  
 $\mathcal{P}_{refined} \leftarrow [ ]$   
 $i \leftarrow 0$   
 $n \leftarrow \text{length of Path}$   
isTZero  $\leftarrow$  **False** //If the path is improved at  $t = 0.0$ , it is set to True.  
**Begin**  
1  $\mathcal{P}_{refined}.\text{add}(\text{Path}[0])$   
2 **While**  $i < (n - 2)$  **do**  
3    $a \leftarrow \text{Path}[i+2]$   
4    $b \leftarrow \text{Path}[i+1]$   
5    $c \leftarrow \mathcal{P}_{refined}[-1]$  //The value -1 indicates the last index.  
6    $(\text{new}_b, \text{isTZero}) \leftarrow \text{ParametricEquation}(a, b, c, t_{incre})$   
7   **If** isTZero **then**  
8      $i \leftarrow i+1$   
9     **Continue**  
10    $\mathcal{P}_{refined}.\text{add}(\text{new}_b)$   
11    $i \leftarrow i+1$   
12 **End While**  
13  $\mathcal{P}_{refined}.\text{add}(\text{Path}[-1])$  //The value -1 indicates the last index.  
14 **Return**  $\mathcal{P}_{refined}$   
**End MoveParent**

---

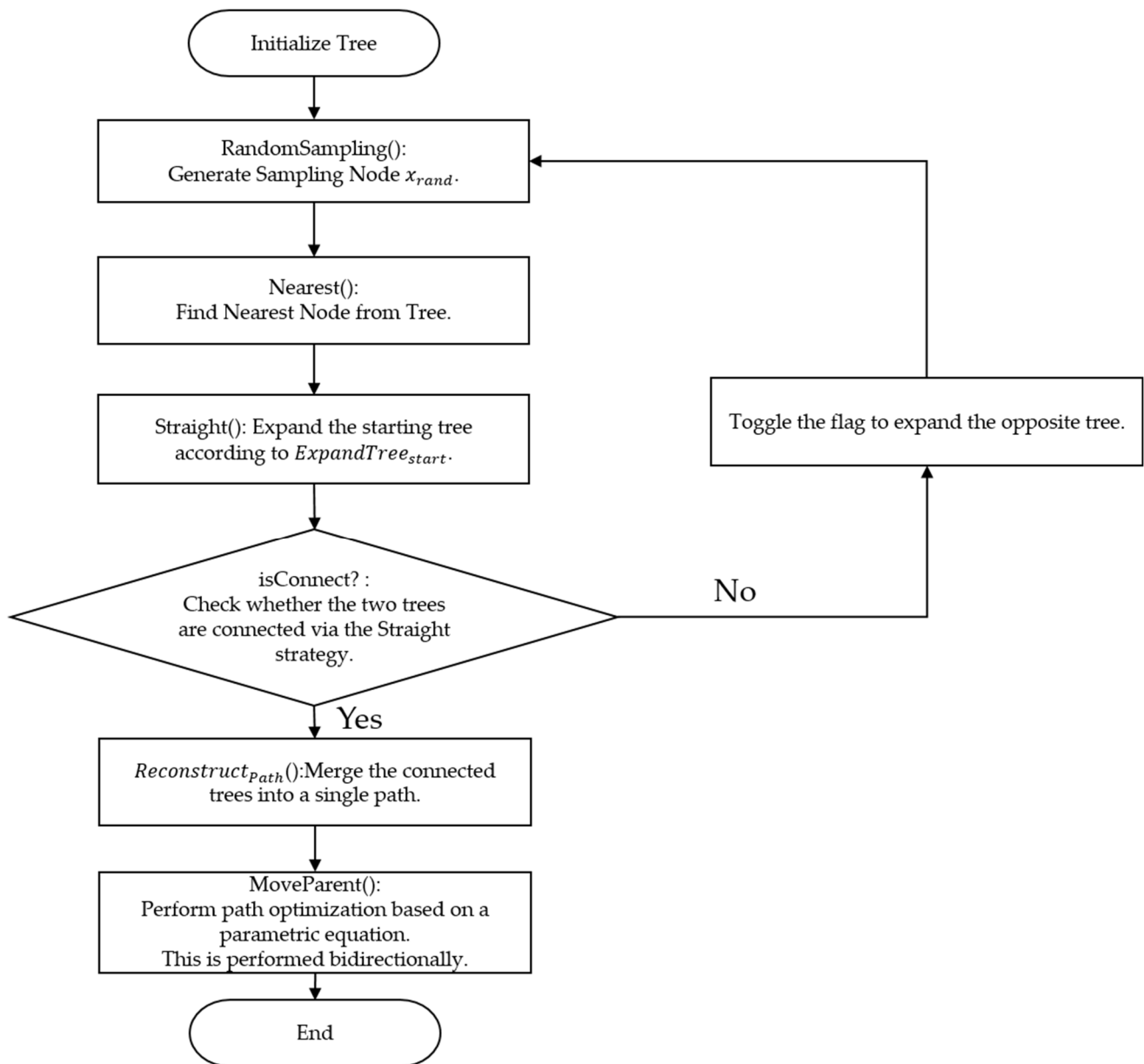
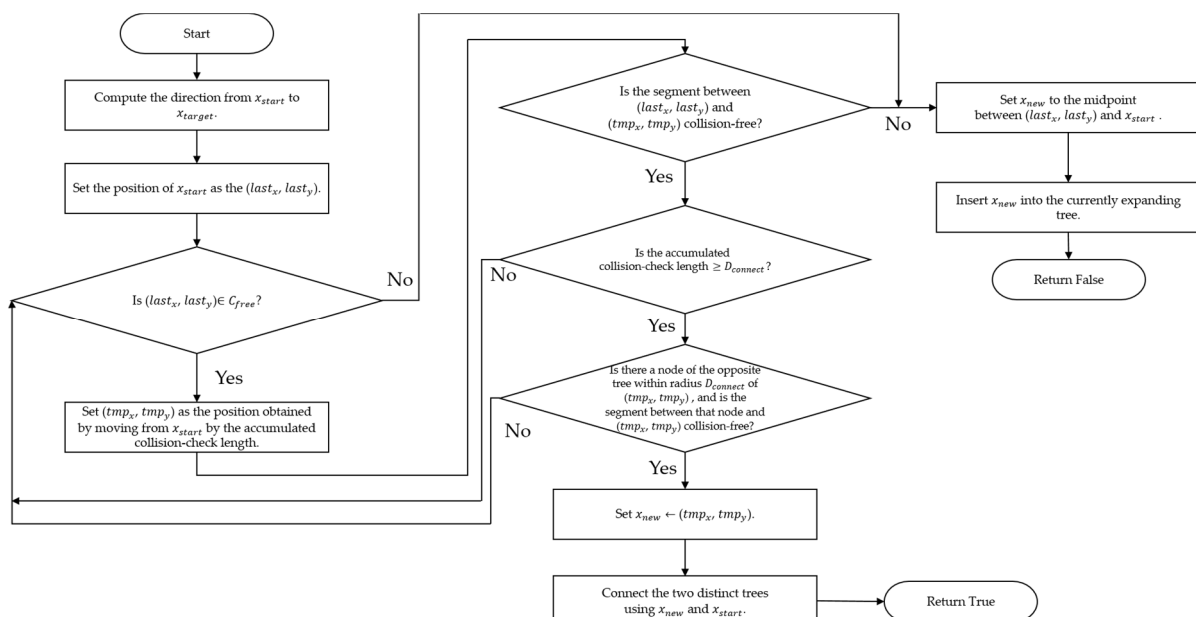
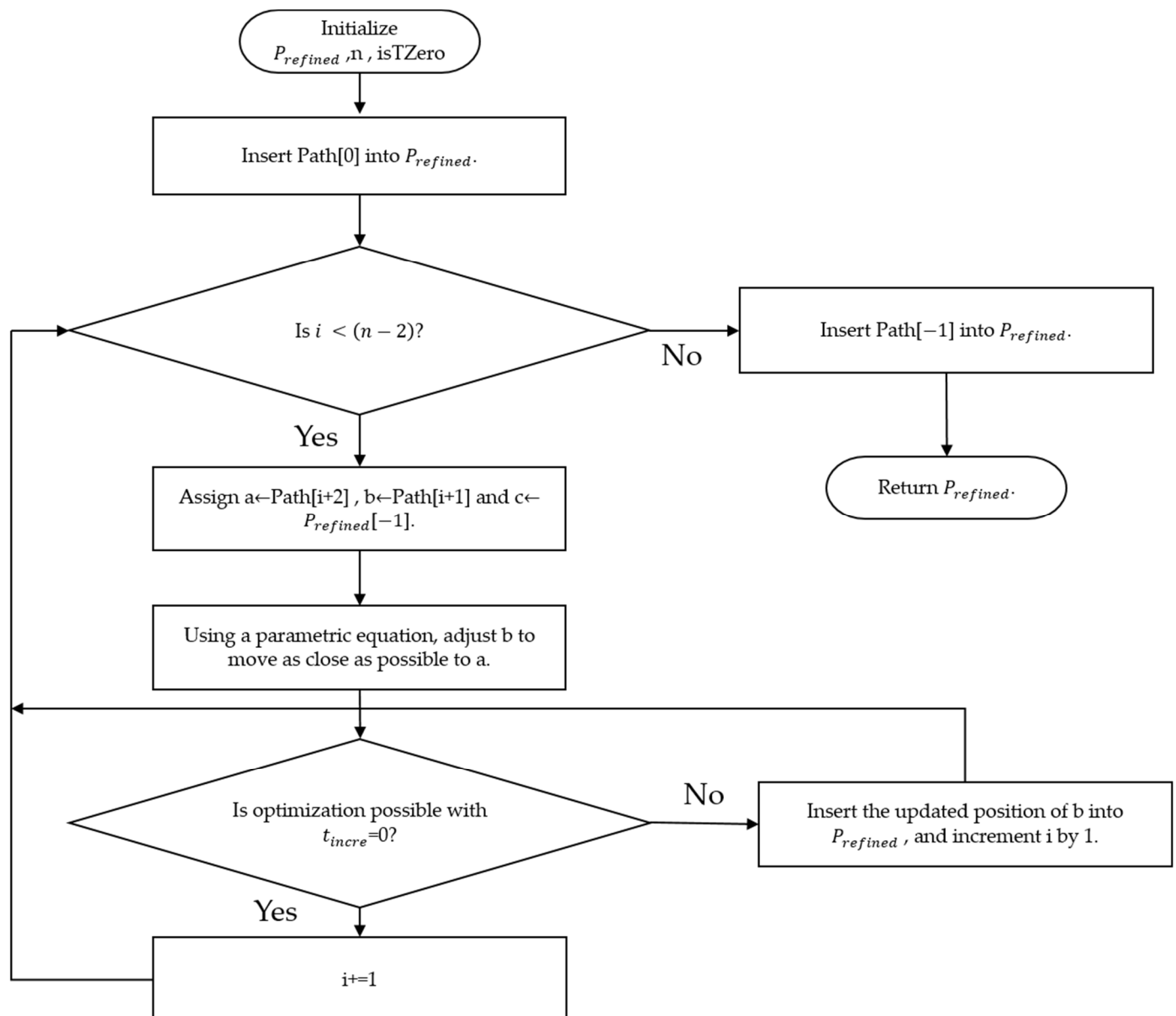


Figure 4. Flowchart of the Straight-RRT algorithm.



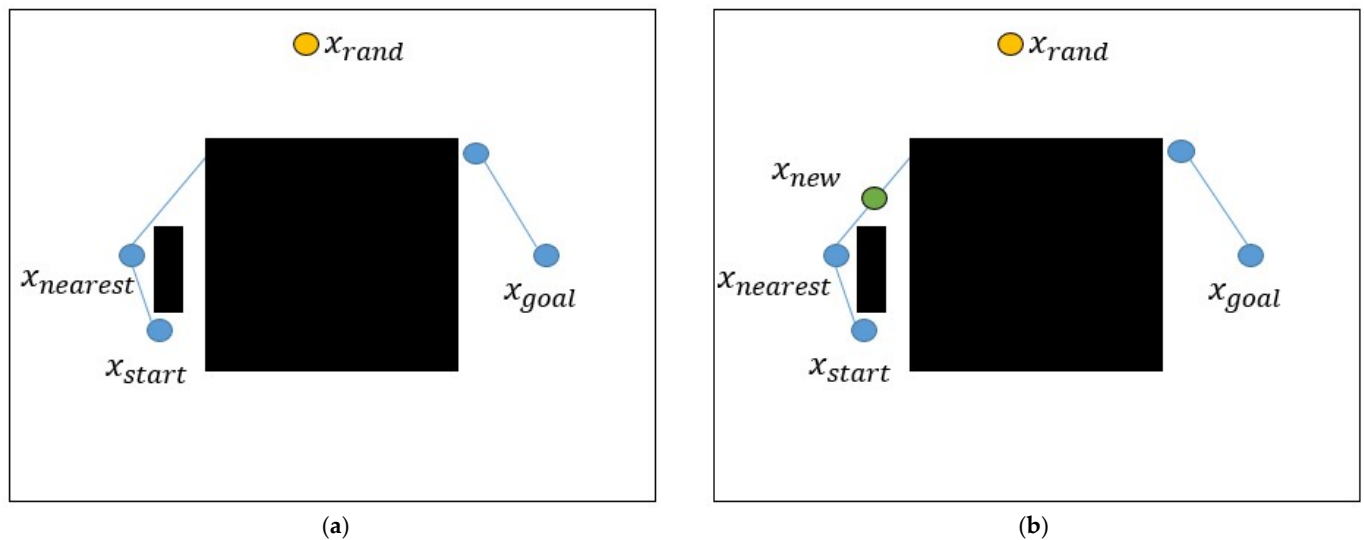
**Figure 5.** Flowchart of the Straight strategy.**Figure 6.** Flowchart of the MoveParent strategy.

#### 4.1. Straight Strategy

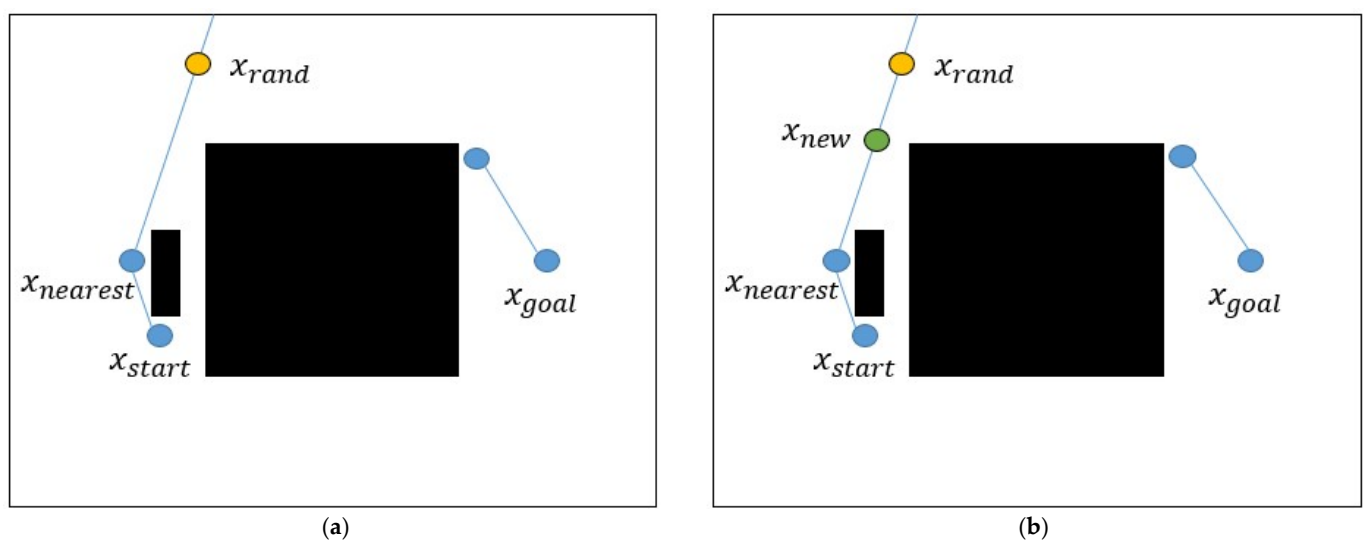
Existing RRT and its variations can only be extended by the step-size parameter value in the  $x_{rand}$  direction from  $x_{nearest}$  to  $x_{new}$ . If there is an obstacle on the straight path from  $x_{nearest}$  to  $x_{new}$ , the point is discarded. If it is discarded, the inspection cost used for sampling and collision-checking is wasted, increasing planning time. To address this issue, this section introduces the Straight strategy. Algorithm 7 presents the Straight strategy procedure. In Algorithm 7, the CollisionFree function checks whether there is an obstacle between the given nodes. The NearestNodeInOpposite function finds the nearest node in the opposite tree within a distance  $D_{connect}$  from the given node. If no such node exists, it returns null. The Connect function inserts  $x_{new}$  into the opposite tree and connects the two trees.

The Straight strategy looks for points that can be extended in the direction of  $x_{rand}$  until a collision with the boundaries of obstacles, walls, or the map is detected. (Figures 7a and 8a). If a collision is detected, the point immediately before the collision and the

midpoint between  $x_{nearest}$  and that point are added to the tree as  $x_{new}$  (Figures 7b and 8b). Adding the midpoint rather than directly using the furthest extended point is intended to mitigate search bias and improve exploration capability. If the furthest-extended point is greedily added as a node, the node is more likely to be placed near obstacles, which reduces the clearance available in the next expansion step and can negatively affect tree growth. Therefore, the straight strategy inserts the midpoint into the tree to account for both expansion stability and exploration of the free space.



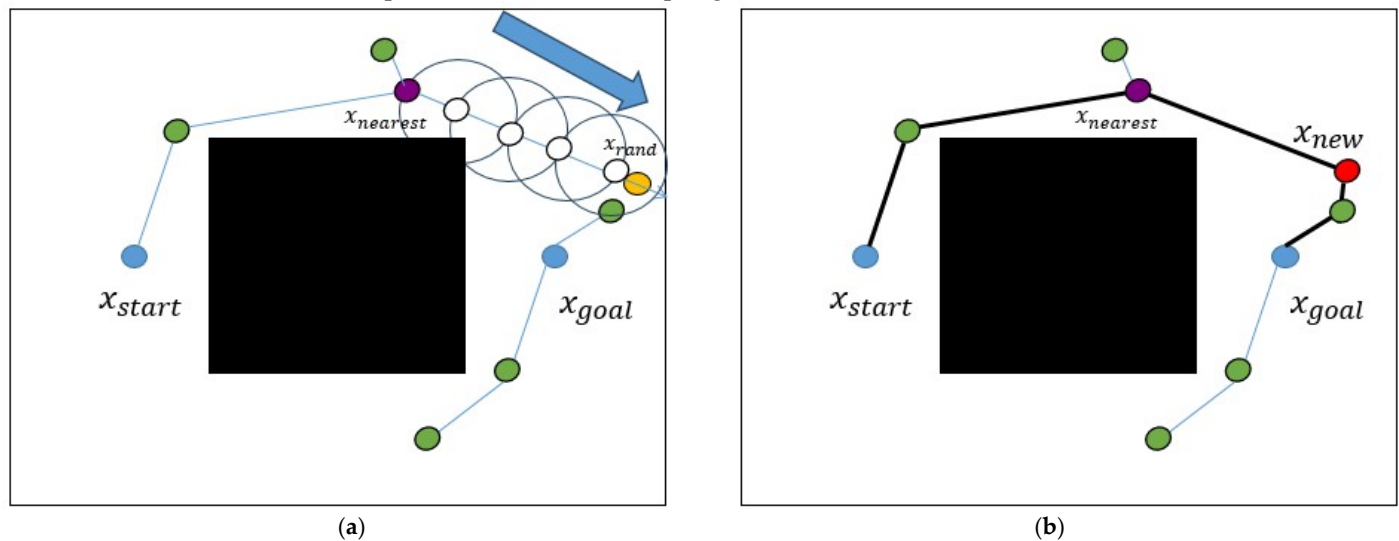
**Figure 7.** Example of the Straight strategy when a collision with an obstacle occurs. (a) A case where the extension in the  $x_{rand}$  direction hits an obstacle. (b) The midpoint node between  $x_{nearest}$  and the point immediately before the collision with the obstacle is added as  $x_{new}$ .



**Figure 8.** Example of the Straight strategy when a collision with the map boundary occurs. (a) A case where the extension in the  $x_{rand}$  direction reaches the map boundary. (b) The midpoint node between  $x_{nearest}$  and the point immediately before the collision with the map boundary is added as  $x_{new}$ .

In addition, as it extends in the  $x_{rand}$  direction, it looks for nodes that can be connected to the opposite tree within a  $D_{connect}$  radius. The inspection cycle is not performed every time for efficiency, but only when it has moved by a distance of  $D_{connect}$  (Figure 9a). And if there is a connectable node within a  $D_{connect}$  radius, it connects the two trees (Figure 9b). This process eliminates the need for step-size parameters, thereby minimizing

parameter dependence and allows faster tree expansion. In addition, it reduced wasted inspection costs when sampling nodes where collisions occur.



**Figure 9.** Example of tree connection: (a) While extending in the  $x_{rand}$  direction, it searches for a connectable node of the opposite tree within a  $D_{connect}$  radius. The blue circular outline is a circle with a radius of  $D_{connect}$ , i.e., the inspection range. The inspection is performed whenever it moves by the  $D_{connect}$  radius. (b) Since there exists a connectable node in the opposite tree within the radius, the position of the current node is set to  $x_{new}$ . Then, the two trees are connected.

#### 4.2. MoveParent Strategy

The Straight strategy above can quickly find a path, but it may produce a path with many nodes and curves. Therefore, the MoveParent strategy is applied to the found path as a post-processing step. Many RRT-based algorithms use a triangle inequality-based strategy for post-processing. This performs computations on the found path. According to the triangle inequality principle, the length of the path moving from path[i] to path[i+1] to path[i+2] is longer than or equal to the length of the path going from path[i] to path[i+2]. Therefore, if the straight-line path between path[i] and path[i+2] is collision-free, the path length is improved, so the path is reconstructed to go directly from path[i] to path[i+2] rather than from path[i] to path[i+1]. However, a drawback of the triangle inequality is that it cannot be applied if there is a very small obstacle in the segment between path[i] and path[i+2].

To address this, MoveParent uses a parametric equation and places candidate positions of path[i+1] on the segment between path[i+1] and path[i+2], and it refines the segment when an improvement is expected. Starting with 0.0 as the initial value, candidate positions of path[i+1] are found based on the  $t$  value. Therefore, if the candidate position according to the  $t$  value has no collision with path[i], the position of path[i+1] is updated. Depending on the  $t$  value, the candidate position changes, and the closer it is to 0, the more it indicates a region where optimization performs better. The MoveParent strategy shows that if there is no collision when  $t$  is greater than 0 and less than 1.0, path[i+1] can be optimized even in segments where the triangle inequality cannot optimize. The meaning of this strategy is that it is called MoveParent because the position of path[i+1] is updated as if it moves along the segment between path[i+1] and path[i+2].

The MoveParent strategy improves the quality of the segment between path[i], path[i+1], and path[i+2] by using a parametric equation. First, it derives the parametric equation for moving from path[i+2] to path[i+1]. And it sets the starting point as path[i+2] and the goal point as path[i+1]. Based on Equations (1) and (2), a parametric equation like Equation (3) is obtained.

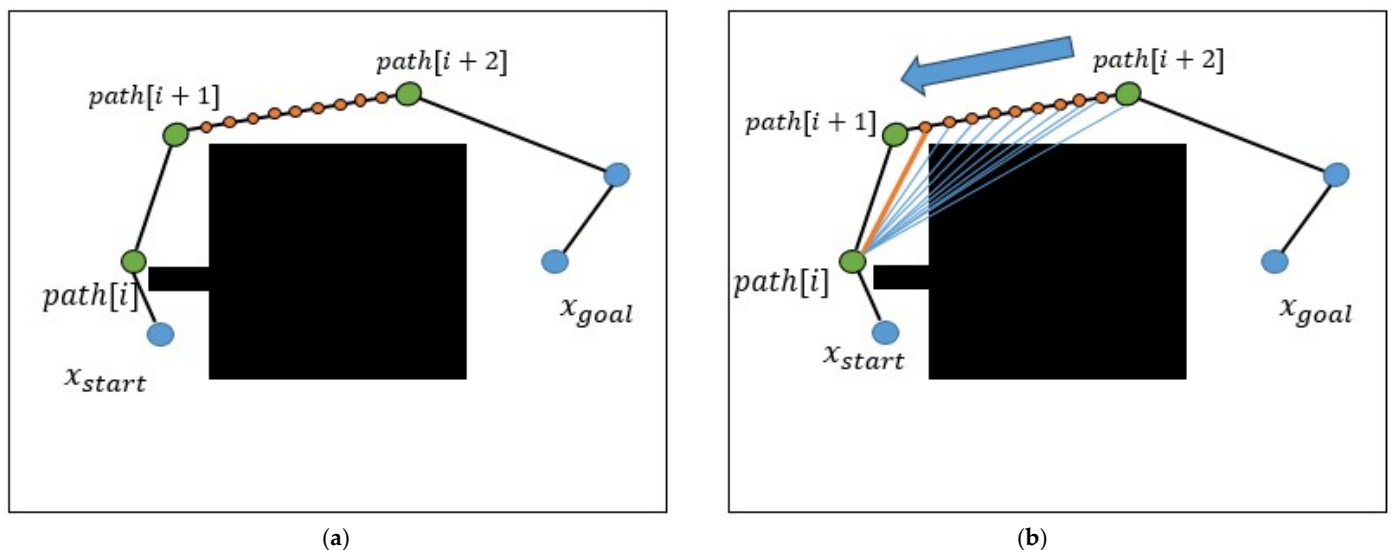
$$P_{\text{start}} = \text{Path}[i+2] \quad (1)$$

$$P_{\text{goal}} = \text{Path}[i+1] \quad (2)$$

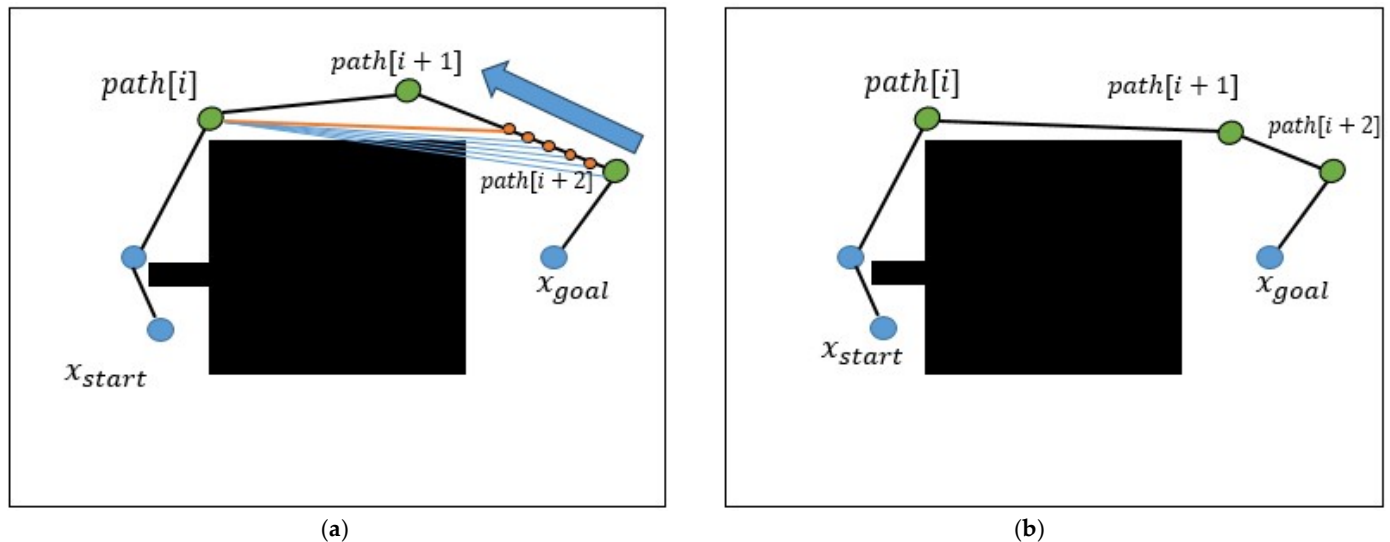
$$P_{\text{newPosition}}(t) = P_{\text{start}} + t(P_{\text{goal}} - P_{\text{start}}), 0.0 \leq t < 1.0 \quad (3)$$

The number of candidate positions of  $\text{path}[i+1]$  to be checked on the straight-line segment between  $\text{path}[i+1]$  and  $\text{path}[i+2]$  is determined by the increment of  $t$ .  $t$  always starts from 0.0. For example, if it is divided into 10 segments,  $t$  is varied within the range [0.0, 1.0) by incrementing it by 0.1 each time. If there is no collision between  $\text{path}[i]$  and the candidate position of  $\text{path}[i+1]$  at each increment of  $t$ , it is updated to that position, and the check proceeds to the next path segment.

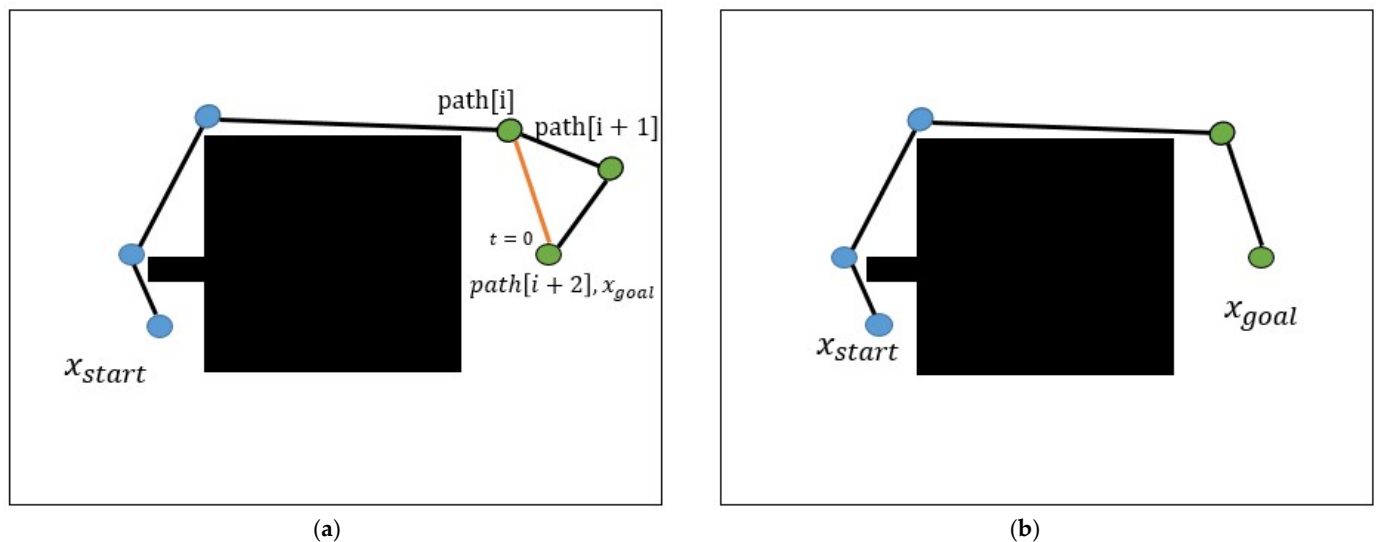
Using Equation (3), candidate positions of  $\text{path}[i+1]$  according to the  $t$  value are calculated as shown in Figure 10a. If the candidate position for a given  $t$  value is collision-free with  $\text{path}[i]$ , the candidate position is updated as the new position of  $\text{path}[i]$ , as shown in Figure 10b. With the MoveParent strategy, as can be seen in Figure 11a, the updated  $\text{path}[i+1]$  also affects the next MoveParent iteration, further optimizing the path as shown in Figure 11b. Figure 12 shows that the path is improved at  $t = 0.0$ . In this case,  $\text{path}[i]$  is connected to  $\text{path}[i+2]$ . When  $t = 0.0$ ,  $\text{path}[i+1]$  is removed, and  $\text{path}[i]$  is connected to  $\text{path}[i+2]$ . Then, instead of checking the next segment, the MoveParent process is performed again based on the current position of  $\text{path}[i]$ .



**Figure 10.** Example with a  $t$  increment of 0.1, when  $\text{path}[i+1]$  is updated at  $t = 0.9$ . (a) According to the parametric equation,  $\text{path}[i+1]$  has 10 candidate positions. (b) The candidate positions of  $\text{path}[i+1]$  are searched from  $\text{path}[i+2]$  to  $\text{path}[i+1]$  according to the parametric equation. It shows an example of finding a new  $\text{path}[i+1]$  position at  $t = 0.9$ .



**Figure 11.** Example with a  $t$  increment of 0.1, when  $path[i+1]$  is updated at  $t = 0.6$ . (a) A collision-free case when  $t = 0.6$ . (b) The path is improved by updating  $path[i+1]$  to the new position at  $t = 0.6$ .



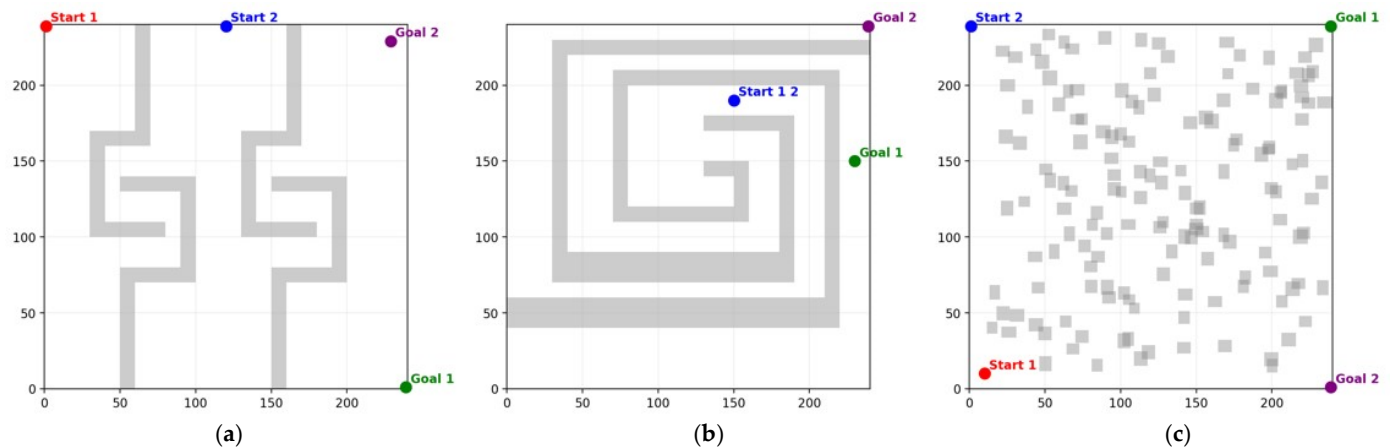
**Figure 12.** Example of  $t = 0$ . (a) A collision-free case when  $t = 0$ . (b) As a result, the path is optimized compared to the initial path.

This strategy is applied bidirectionally to the path found by the Straight strategy, excluding the start and goal positions. In addition, the number of potential candidate positions for  $path[i+1]$  is discussed in Section 5.

## 5. Experimental Results

In this section, we evaluate the performance of the algorithms on the map shown in Figure 13. The map size is 240 (width)  $\times$  240 (height) pixels, and the experiments are conducted in a Python 3.9 environment. The gray areas represent obstacles. For each map, the experiments are conducted separately for Case 1 and Case 2. In Case 1, the start position is start 1, and the goal position is goal 1. In Case 2, the start position is start 2, and the goal position is goal 2. Each start and goal position is shown in Table 3.

Each algorithm is executed 100 times in the given environment, and the mean values are compared. The evaluation metrics are the path length (pixels), the time required to compute the path (seconds), and cumulative heading change (radians).



**Figure 13.** Maps for the experiment. The size of each map is  $240 \times 240$ . The gray areas are the obstacles: (a) Map 1; (b) Map 2; and (c) Map 3.

**Table 3.** The start and goal position for each map.

|      | Start 1    | Goal 1     | Start 2    | Goal 2     |
|------|------------|------------|------------|------------|
| Map1 | (1, 239)   | (239, 1)   | (120, 239) | (229, 229) |
| Map2 | (150, 190) | (230, 150) | (150, 190) | (239, 239) |
| Map3 | (10, 10)   | (239, 239) | (1, 239)   | (239, 1)   |

The comparative experimental results are compared by normalizing the Q-RRT\* result to 100, as in Equation (4). That is, values closer to 100 indicate performance similar to Q-RRT\*. Values smaller than 100 indicate better performance. In contrast, values larger than 100 indicate worse performance.

$$\text{Normalized} = \frac{\text{mean}(\text{Result}_{\text{others}})}{\text{mean}(\text{Result}_{\text{Q-RRT}^*})} \times 100 \quad (4)$$

For the comparison parameters,  $D_{\text{ancestor}}$  in Q-RRT\* was set to 2. The dichotomy value in F-RRT\* was set to 2. In MQ-RRT\*,  $R_{\text{commit}}$  was set to 10, and the dichotomy and  $D_{\text{ancestor}}$  were set to 2. These values were all set to the recommended values suggested in the original papers. The proposed algorithm sets the  $t$  increment to 0.1. The experimental setup for the  $t$ -increment is presented in Section 5.4. Table 4 shows the specifications of the computer used for the experiments.

**Table 4.** Experimental environment.

| Hardware | Specification                         |
|----------|---------------------------------------|
| CPU      | AMD Ryzen 7 3700X (8 cores, 3.60 GHz) |
| RAM      | 32.0 GB                               |
| VGA      | NVIDIA GeForce RTX 2070 SUPER (8 GB)  |

### 5.1. Experimental Results on Map 1

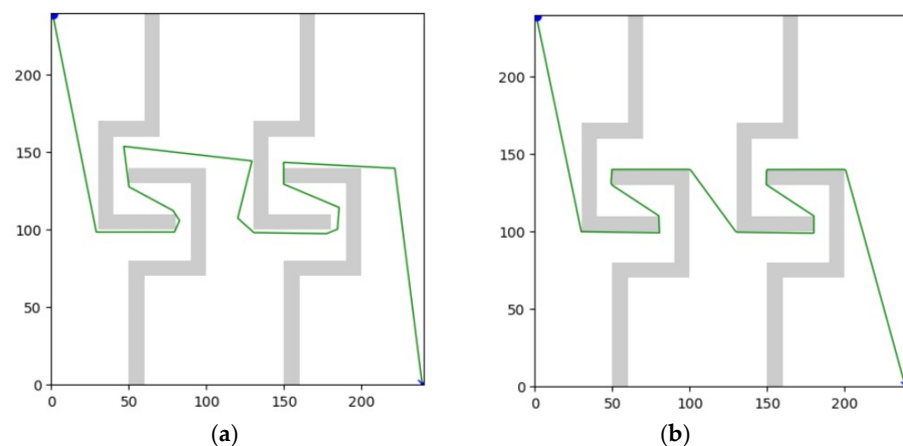
Figures 14 and 15 show the experimental results for Case 1 in the Map 1 environment. Figure 14 shows the experimental results of Q-RRT\* and F-RRT\*. Figure 15 shows the experimental results of MQ-RRT\* and Straight-RRT.

The straight strategy can find an initial path very quickly, but it may generate an inefficient path. Therefore, when the MoveParent strategy is applied, the sharp segments are smoothed, and the path length is reduced, as can be seen in Figure 15b. In this way, we confirmed that an initial path can be obtained quickly using the Straight strategy and that the path quality can then be improved rapidly using the MoveParent strategy.

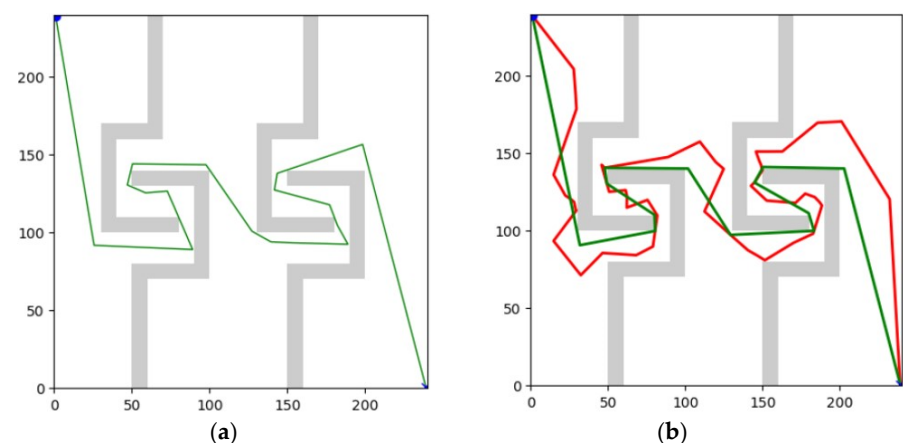
Figures 16 and 17 show the experimental results for Case 2 in the Map 1 environment. Figure 16 shows the experimental results of Q-RRT\* and F-RRT\*. Figure 17 shows the experimental results of MQ-RRT\* and Straight-RRT.

The values in Table 5 are relative comparison results normalized by setting the result of Q-RRT\* to 100. This means that smaller values indicate better performance. In Case 1, the path length of the proposed algorithm is 96.5402, which is improved compared to Q-RRT\* (100) and better than MQ-RRT\* (108.0243), but a longer path is generated than by F-RRT\* (89.5748). The planning time is 12.4823, which is greatly reduced compared to Q-RRT\* and F-RRT\*. In addition, we confirmed that it is slightly faster than MQ-RRT\* (14.0567). In addition, the cumulative heading change in the proposed algorithm in Case 1 is 85.389, which is improved compared to Q-RRT\* (100) and lower than F-RRT\* (90.4439) and MQ-RRT\* (114.2131). This result indicates that the proposed algorithm can reduce the overall amount of direction change along the path.

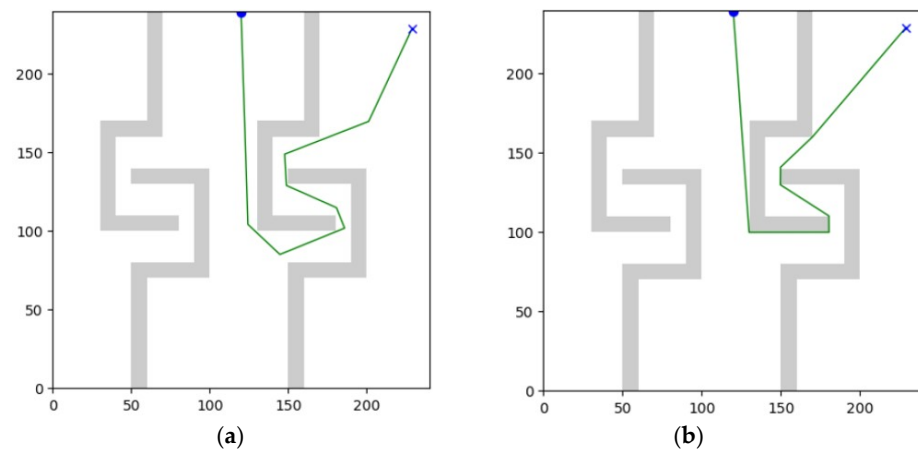
In Case 2 as well, the path length of the proposed algorithm is 92.5402, which is improved compared to Q-RRT\* and better than MQ-RRT\* (103.5305), but the path quality is slightly lower than that of F-RRT\* (90.3169). In contrast, the planning time is 7.6401, which is the fastest among the compared algorithms. For Case 2, the proposed algorithm also shows improved performance in terms of cumulative heading change. As shown in Table 5, the cumulative heading change is 72.7848, which is lower than Q-RRT\* (100), F-RRT\* (79.7509), and MQ-RRT\* (115.2701). From the results for Map 1, it was confirmed that the proposed algorithm is overall faster than the other algorithms, while the path quality is good or comparable.



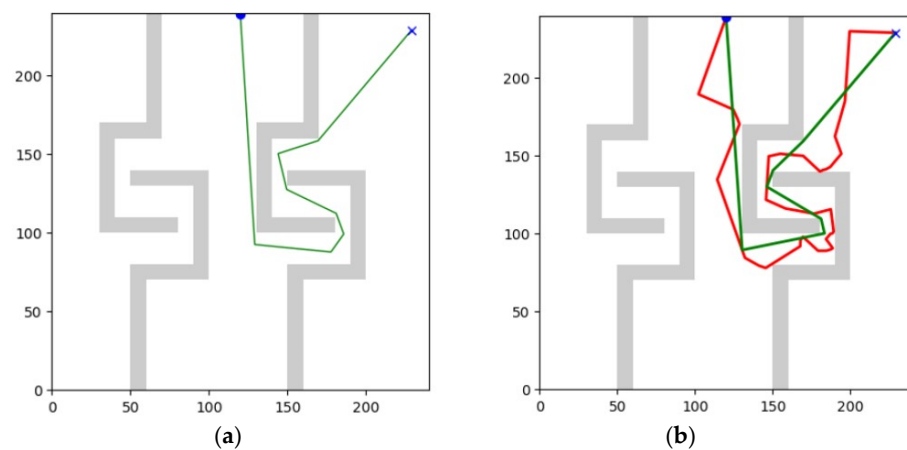
**Figure 14.** Experimental results for Case 1 in Map 1. The green line represents the path computed by each algorithm: (a) Q-RRT\*; (b) F-RRT\*.



**Figure 15.** Experimental results for Case 1 in Map 1: (a) MQ-RRT\*; (b) Straight-RRT: The red line represents the initial path computed using the Straight strategy. The green line represents the path after applying the MoveParent strategy.



**Figure 16.** Experimental results for Case 2 in Map 1. The green line represents the path computed by each algorithm: (a) Q-RRT\*; (b) F-RRT\*.



**Figure 17.** Experimental results for Case 2 in Map 1: (a) MQ-RRT\*; (b) Straight-RRT: The red line represents the initial path computed using the Straight strategy. The green line represents the path after applying the MoveParent strategy.

**Table 5.** Experimental results for Map 1 (normalized to Q-RRT = 100%).

|                               |                               | Q-RRT* | F-RRT*  | MQ-RRT*  | Proposed Algorithm |
|-------------------------------|-------------------------------|--------|---------|----------|--------------------|
| Case 1<br>(Start 1 to Goal 1) | Length of path (%)            | 100    | 89.5748 | 108.0243 | 96.5402            |
|                               | Planning time (%)             | 100    | 57.3709 | 14.0567  | 12.4823            |
|                               | Cumulative heading change (%) | 100    | 90.4439 | 114.2131 | 85.389             |
| Case 2<br>Start 2 to Goal 2   | Length of path (%)            | 100    | 90.3169 | 103.5305 | 92.5256            |
|                               | Planning time (%)             | 100    | 55.1641 | 20.0516  | 7.6401             |
|                               | Cumulative heading change (%) | 100    | 79.7509 | 115.2701 | 72.7848            |

Table 6 reports the experimental results as mean (std) over repeated runs. For planning time, the proposed method achieves low average computation time with relatively low variability (7.87 (2.74) in Case 1 and 2.30 (1.20) in Case 2). In comparison, Q-RRT\* shows much larger average time and variability (63.11 (29.25) and 30.19 (26.98)), and F-RRT\* also has larger mean and variability (36.21 (11.30) and 16.65 (9.81)). MQ-RRT\* has a

similar mean time but relatively small variability in Case 1 (8.87 (1.16)) and a higher mean time in Case 2 (6.05 (1.78)).

**Table 6.** Experimental results for Map 1 (mean (std.)).

|                               |                                 | Q-RRT*         | F-RRT*        | MQ-RRT*        | Proposed Algorithm |
|-------------------------------|---------------------------------|----------------|---------------|----------------|--------------------|
| Case 1<br>(Start 1 to Goal 1) | Length of path (px)             | 730.53 (19.97) | 654.37 (1.30) | 789.15 (22.70) | 705.26 (21.38)     |
|                               | Planning time (s)               | 63.11 (29.25)  | 36.21 (11.30) | 8.87 (1.16)    | 7.87 (2.74)        |
|                               | Cumulative heading change (rad) | 17.82 (0.73)   | 16.12 (0.13)  | 20.3612 (1.96) | 15.22 (0.32)       |
| Case 2<br>Start 2 to Goal 2   | Length of path (px)             | 421.98 (15.78) | 381.12 (1.23) | 436.88 (20.50) | 390.44 (11.57)     |
|                               | Planning time (s)               | 30.19 (26.98)  | 16.65 (9.81)  | 6.05 (1.78)    | 2.30 (1.20)        |
|                               | Cumulative heading change (rad) | 8.5364 (0.66)  | 6.80 (0.12)   | 9.84 (1.35)    | 6.21 (0.31)        |

### 5.2. Experimental Results on Map 2

Figures 18 and 19 show the experimental results for Case 1 in the Map 2 environment. Figure 18 shows the experimental results of Q-RRT\* and F-RRT\*. Figure 19a shows the experimental results of MQ-RRT\*. Figure 19b shows the experimental results of Straight-RRT.

When only the straight strategy was applied, the resulting path was winding. By applying the MoveParent strategy to this path, the winding path became closer to a straight line. In addition, we confirmed that it was improved into a more efficient path overall. Figures 20 and 21 show the experimental results for Case 2 in the Map 2 environment. Figure 20 shows the experimental results of Q-RRT\* and F-RRT\*. Figure 21 shows the experimental results of MQ-RRT\* and Straight-RRT.

The values in Table 7 are relative comparison results normalized by setting the result of Q-RRT\* to 100. In Case 1, the path length of the proposed algorithm was 103.8429, which was a slightly longer path than that generated by the baseline algorithm, Q-RRT\*. However, the proposed algorithm achieved a planning time of 1.0969, and we confirmed that although the path was slightly inefficient, the planning time was fast enough to mitigate this drawback. The comparison results with F-RRT\* are similar to those of Q-RRT\*, but it is much faster in terms of planning time. We confirmed that, like Q-RRT\*, it can mitigate the drawback in terms of path length. We confirmed that the proposed algorithm achieves better results than MQ-RRT\* in terms of both planning time and path length.

**Table 7.** Experimental results for Map 2 (normalized to Q-RRT = 100%).

|                               |                               | Q-RRT* | F-RRT*  | MQ-RRT*  | Proposed Algorithm |
|-------------------------------|-------------------------------|--------|---------|----------|--------------------|
| Case 1<br>(Start 1 to Goal 1) | Length of path (%)            | 100    | 96.8872 | 112.4546 | 103.8429           |
|                               | Planning time (%)             | 100    | 41.7648 | 3.4434   | 1.0969             |
|                               | Cumulative heading change (%) | 100    | 96.2589 | 112.4961 | 85.9243            |
| Case 2<br>Start 2 to Goal 2   | Length of path (%)            | 100    | 95.2605 | 101.2851 | 97.7381            |
|                               | Planning time (%)             | 100    | 89.7849 | 21.4039  | 4.9181             |
|                               | Cumulative heading change (%) | 100    | 98.4794 | 118.1316 | 98.2255            |

A similar tendency can be observed in Case 2. The path length of the proposed algorithm is 97.7381, which is shorter than Q-RRT\* (100) and MQ-RRT\* (101.2851), and is at a similar level to F-RRT\* (95.2605). The planning time is 4.9181, which is smaller than that of Q-RRT\* (100), F-RRT\* (89.7849), and MQ-RRT\* (21.4039), showing a clear advantage in time performance. From the results for Map 2, the proposed algorithm maintains a path

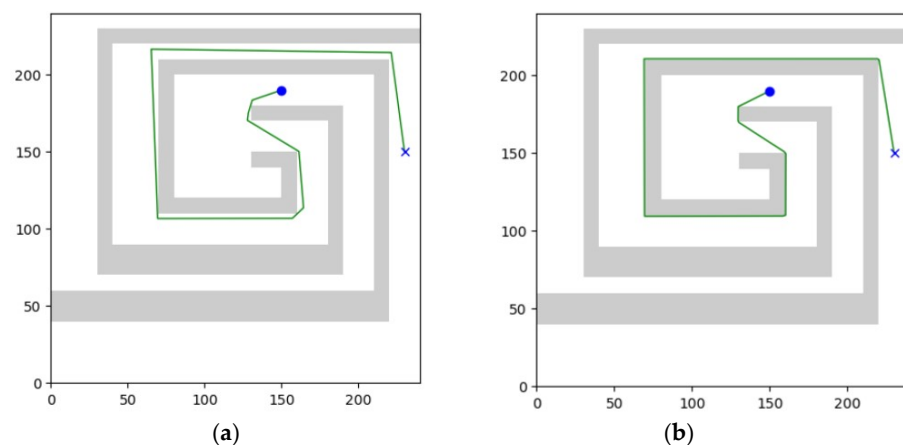
length comparable to or slightly longer than that of F-RRT\* while achieving superior planning time. The path quality was comparable to that of Q-RRT\*, but the planning time was much faster. Compared to MQ-RRT\*, it showed better results in all aspects.

In addition, Table 7 shows the cumulative heading change (rad.) normalized to Q-RRT\* (100). In Case 1, the proposed algorithm achieves 85.9243, which is lower than Q-RRT\* (100), F-RRT\* (96.2589), and MQ-RRT\* (112.4961), indicating a reduced overall direction-change demand along the path. In Case 2, the proposed algorithm achieves 98.2255, which is slightly lower than Q-RRT\* (100) and comparable to F-RRT\* (98.4794), while being lower than MQ-RRT\* (118.1316).

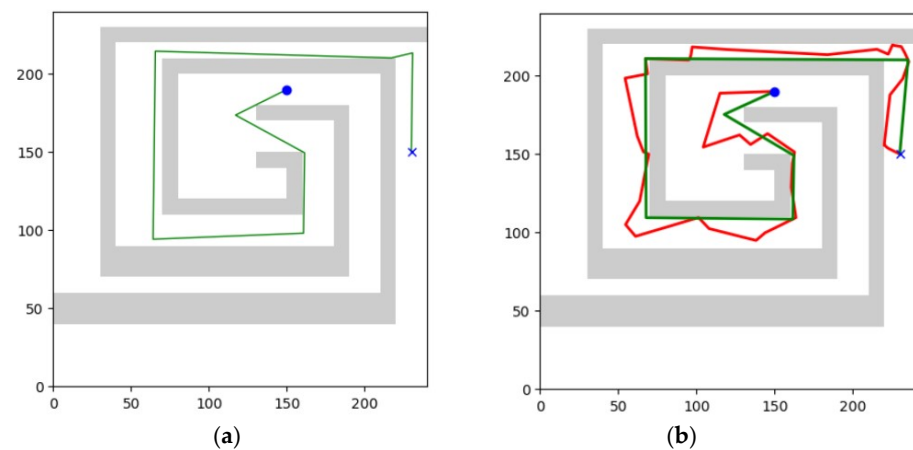
Table 8 reports the experimental results as mean (std) over repeated runs. For planning time, the proposed algorithm achieves low mean values with relatively small variability (Case 1: 2.55 (0.78), Case 2: 1.24 (0.34)), while Q-RRT\* and F-RRT\* show much larger variability (Q-RRT\*: 233.20 (94.06) and 25.33 (34.88), F-RRT\*: 97.39 (28.30) and 22.75 (25.22)). For the length of the path, the proposed algorithm shows competitive mean values (Case 1: 549.31 (11.96), Case 2: 700.45 (7.28)) and remains more consistent than MQ-RRT\* in both cases.

**Table 8.** Experimental results for Map 2 (mean (std.)).

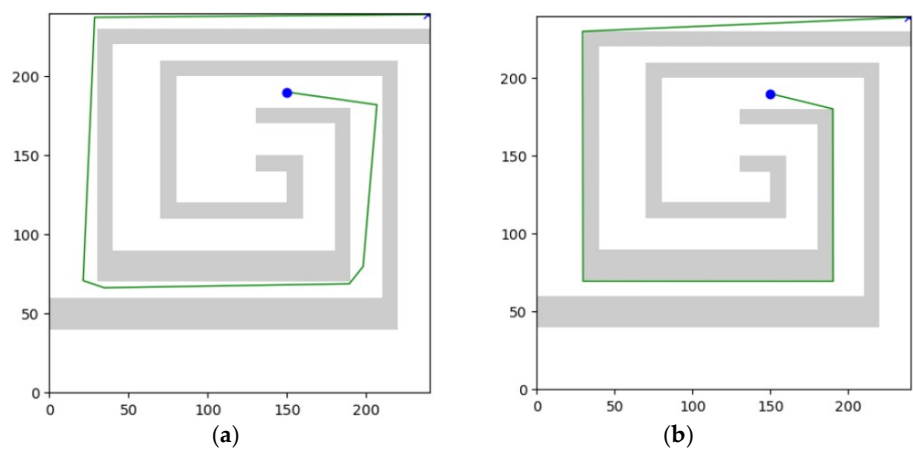
|                               |                                 | Q-RRT*         | F-RRT*        | MQ-RRT*        | Proposed Algorithm |
|-------------------------------|---------------------------------|----------------|---------------|----------------|--------------------|
| Case 1<br>(Start 1 to Goal 1) | Length of path (px)             | 528.98 (6.38)  | 512.52 (1.07) | 594.87 (22.55) | 549.31 (11.96)     |
|                               | Planning time (s)               | 233.20 (94.06) | 97.39 (28.30) | 8.03 (1.13)    | 2.55 (0.78)        |
|                               | Cumulative heading change (rad) | 11.06 (0.26)   | 10.64 (0.13)  | 12.44 (1.15)   | 9.50 (0.15)        |
| Case 2<br>Start 2 to Goal 2   | Length of path (px)             | 716.66 (10.68) | 682.69 (0.64) | 725.87 (10.68) | 700.45 (7.28)      |
|                               | Planning time (s)               | 25.33 (34.88)  | 22.75 (25.22) | 5.42 (2.80)    | 1.24 (0.34)        |
|                               | Cumulative heading change (rad) | 6.14 (0.07)    | 6.04 (0.02)   | 7.25 (1.13)    | 6.03 (0.02)        |



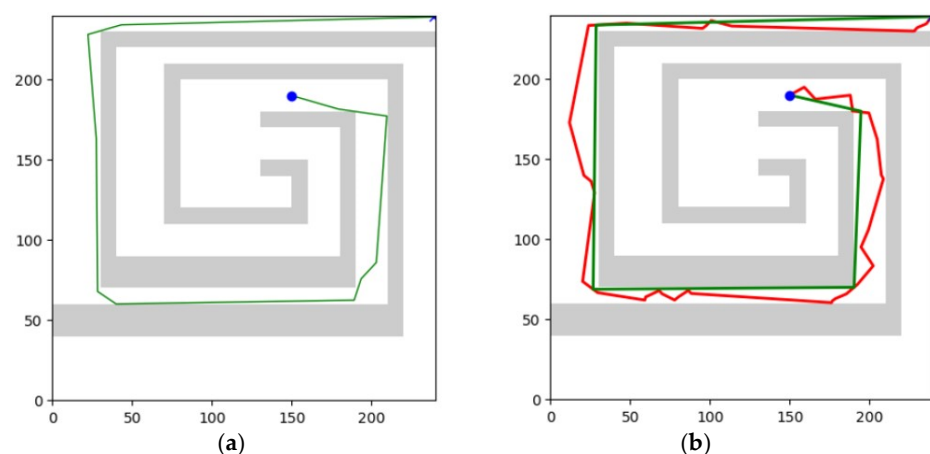
**Figure 18.** Experimental results for Case 1 in Map 2. The green line represents the path computed by each algorithm: (a) Q-RRT\*; (b) F-RRT\*.



**Figure 19.** Experimental results for Case 1 in Map 2: (a) MQ-RRT\*; (b) Straight-RRT: The red line represents the initial path computed using the Straight strategy. The green line represents the path after applying the MoveParent strategy.



**Figure 20.** Experimental results for Case 2 in Map 2. The green line represents the path computed by each algorithm: (a) Q-RRT\*; (b) F-RRT\*.



**Figure 21.** Experimental results for Case 2 in Map 2: (a) MQ-RRT\*; (b) Straight-RRT: The red line represents the initial path computed using the Straight strategy. The green line represents the path after applying the MoveParent strategy.

### 5.3. Experimental Results on Map 3

Figures 22 and 23 show the experimental results for Case 1 in the Map 3 environment. The Map 3 environment contains many obstacles and is therefore unfavorable for tree

expansion. Figure 22 shows the experimental results of Q-RRT\* and F-RRT\*. Figure 23 shows the experimental results of MQ-RRT\* and Straight-RRT. The values in Table 9 are relative comparison results normalized by setting the result of Q-RRT\* to 100.

In Case 1, the path length of the proposed algorithm is 100.5065, which is similar to Q-RRT\* (100) and MQ-RRT\* (102.7431), but tends to be slightly longer than F-RRT\* (94.5312). In contrast, the planning time of the proposed algorithm is 7.5980, which is smaller than those of Q-RRT\* (100), F-RRT\* (82.3475), and MQ-RRT\* (48.8431), showing the fastest planning time among the compared algorithms.

Figures 24 and 25 present the experimental results for Case 2 of Map 3. Figure 25b shows the results of the Straight-RRT. A similar tendency can be observed in Case 2. The path length of the proposed algorithm is 97.4838, which is shorter than Q-RRT\* (100) and MQ-RRT\* (100.9559), but somewhat longer than F-RRT\* (95.0389). However, the planning time is 18.8191, which is smaller than those of Q-RRT\* (100), F-RRT\* (155.5519), and MQ-RRT\* (87.5403), exhibiting the best time performance. Therefore, although the proposed algorithm maintains a path length that is similar to or slightly longer than that of F-RRT\*, it provides a much shorter planning time compared to the other algorithms, indicating a clear advantage in terms of speed.

**Table 9.** Experimental results for Map 3 (normalized to Q-RRT = 100%).

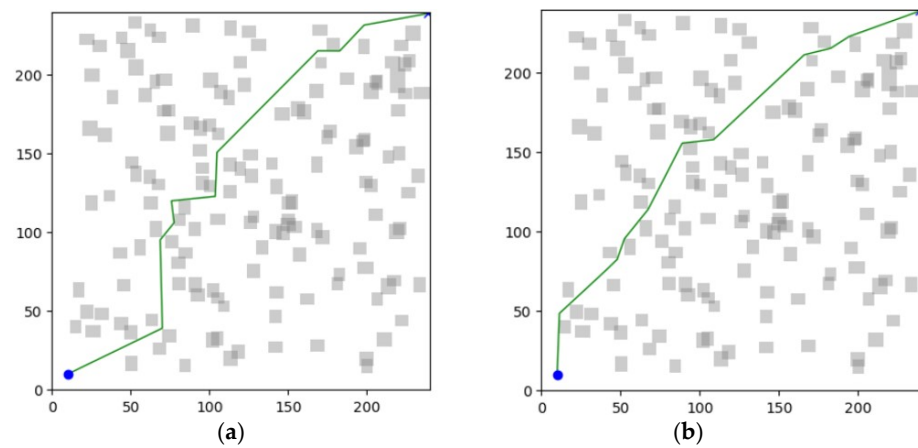
|                               |                               | Q-RRT* | F-RRT*   | MQ-RRT*  | Proposed Algorithm |
|-------------------------------|-------------------------------|--------|----------|----------|--------------------|
| Case 1<br>(Start 1 to Goal 1) | Length of path (%)            | 100    | 94.5312  | 102.7431 | 100.5065           |
|                               | Planning time (%)             | 100    | 82.3475  | 48.8431  | 7.5980             |
|                               | Cumulative heading change (%) | 100    | 56.8343  | 125.5068 | 54.3261            |
| Case 2<br>Start 2 to Goal 2   | Length of path (%)            | 100    | 95.0389  | 100.9559 | 97.4838            |
|                               | Planning time (%)             | 100    | 155.5519 | 87.5403  | 18.8191            |
|                               | Cumulative heading change (%) | 100    | 56.6577  | 127.7667 | 49.6513            |

In addition, Table 9 reports the cumulative heading change (rad.) normalized to Q-RRT\* (=100). In Case 1, the proposed algorithm achieves 54.3261, which is lower than Q-RRT\* (100), F-RRT\* (56.8343), and MQ-RRT\* (125.5068), indicating a reduced overall direction-change demand along the path. In Case 2, the proposed algorithm achieves 49.6513, which is lower than Q-RRT\* (100), F-RRT\* (56.6577), and MQ-RRT\* (127.7667).

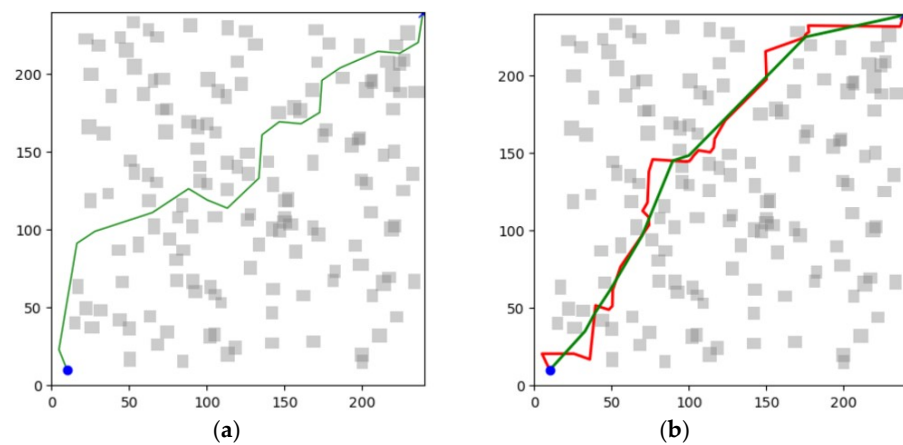
Table 10 reports the experimental results as mean (std) over repeated runs. For planning time, the proposed algorithm shows a low mean with small variability (Case 1: 2.92 (1.33), Case 2: 2.10 (0.35)), while Q-RRT\* and F-RRT\* exhibit much larger variability (Q-RRT\*: 38.46 (35.13) and 11.17 (8.82), F-RRT\*: 31.67 (20.33) and 17.38 (8.58)). For the length of the path, the proposed algorithm shows a similar standard deviation across repeated runs (Case 1: 405.17 (37.24), Case 2: 380.75 (27.16)).

**Table 10.** Experimental results for Map 3 (mean (std.)).

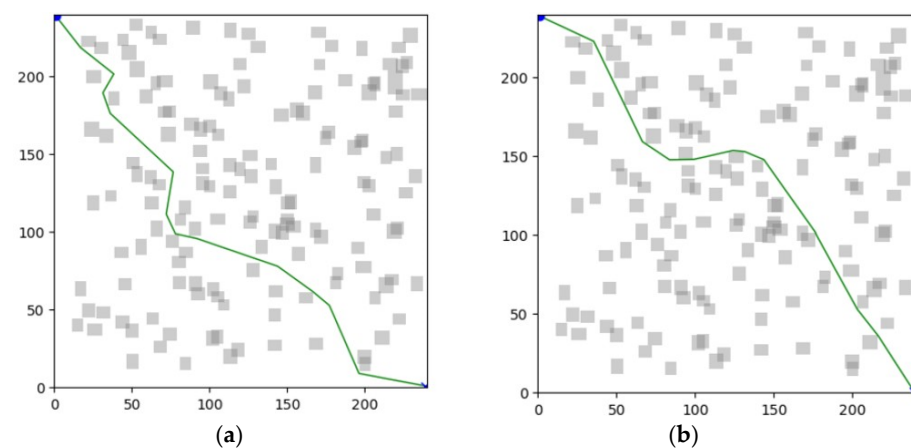
|                               |                                 | Q-RRT*         | F-RRT*         | MQ-RRT*        | Proposed Algorithm |
|-------------------------------|---------------------------------|----------------|----------------|----------------|--------------------|
| Case 1<br>(Start 1 to Goal 1) | Length of path (px)             | 403.12 (33.03) | 381.08 (32.44) | 414.18 (37.48) | 405.17 (37.24)     |
|                               | Planning time (s)               | 38.46 (35.13)  | 31.67 (20.33)  | 18.78 (8.03)   | 2.92 (1.33)        |
|                               | Cumulative heading change (rad) | 8.58 (2.62)    | 4.88 (1.66)    | 10.77 (3.00)   | 4.66 (1.60)        |
| Case 2<br>Start 2 to Goal 2   | Length of path (px)             | 390.57 (27.05) | 371.20 (23.05) | 394.31 (27.22) | 380.75 (27.16)     |
|                               | Planning time (s)               | 11.17 (8.82)   | 17.38 (8.58)   | 9.78 (4.94)    | 2.10 (0.35)        |
|                               | Cumulative heading change (rad) | 6.97 (2.56)    | 3.95 (1.62)    | 8.91 (2.96)    | 3.46 (1.42)        |



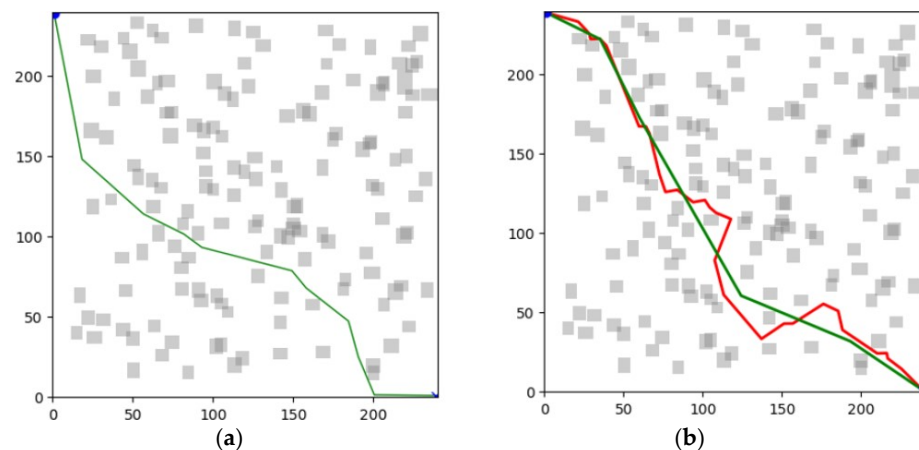
**Figure 22.** Experimental results for Case 1 in Map 3. The green line represents the path computed by each algorithm: (a) Q-RRT\*; (b) F-RRT\*.



**Figure 23.** Experimental results for Case 1 in Map 3: (a) MQ-RRT\*; (b) Straight-RRT: The red line represents the initial path computed using the Straight strategy. The green line represents the path after applying the MoveParent strategy.



**Figure 24.** Experimental results for Case 2 in Map 3: (a) Q-RRT\*; (b) F-RRT\*.



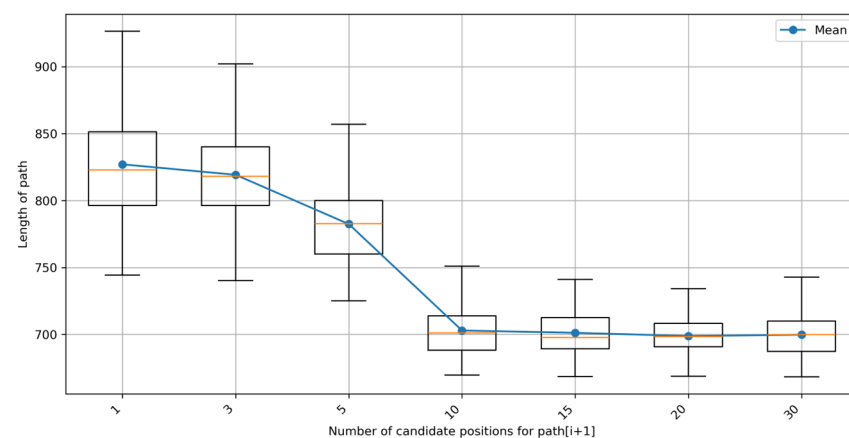
**Figure 25.** Experimental results for Case 2 in Map 3. The green line represents the path computed by each algorithm: (a) MQ-RRT\*; (b) Straight-RRT. The red line represents the initial path computed using the Straight strategy. The green line represents the path after applying the MoveParent strategy.

#### 5.4. Experiment on Candidate Positions of $\text{path}[i+1]$ by $t$ Increment

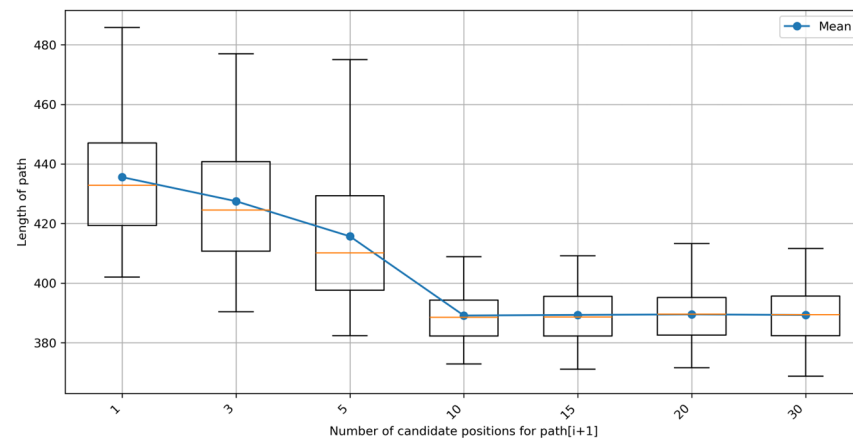
In MoveParent, the increment of the  $t$  value indicates how finely the candidate positions of  $\text{path}[i+1]$  are examined. Therefore, we conducted experiments by dividing the total number of candidate checks according to the increment into 1, 3, 5, 10, 15, 20, and 30. In this case, each increment is 1, 0.34, 0.2, 0.1, 0.067, 0.05, and 0.034. As the increment of the  $t$  value decreases, the number of candidates to be checked increases. An increase in the number of candidates means that the inspection is performed more finely.

The results for Map 1 are shown in Figures 26 and 27, which correspond to Case 1 and Case 2, respectively. As the number of candidates of  $\text{path}[i+1]$  increases, the mean value and dispersion of the length of path decrease, but after 10, the results show a similar level.

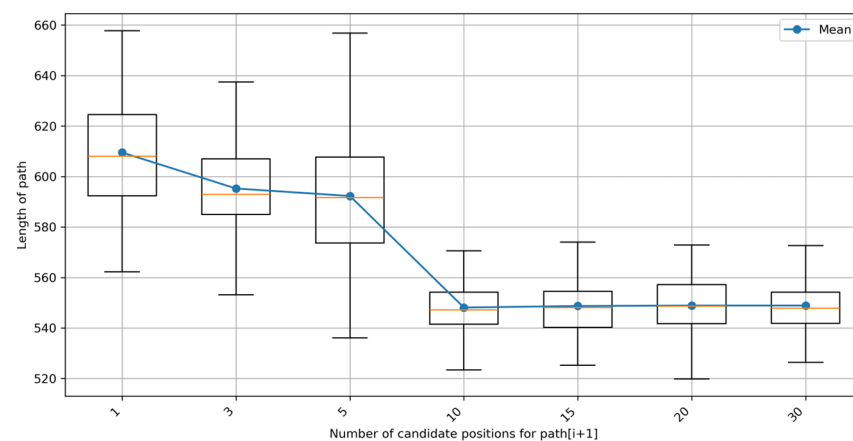
The results for Map 2 are shown in Figures 28 and 29, which correspond to Case 1 and Case 2, respectively. Similar to Map 1, for Map 2, as the number of candidates to be checked increases, the mean and dispersion of the length of the path decrease. However, after 10 candidates, the results remain at a similar level.



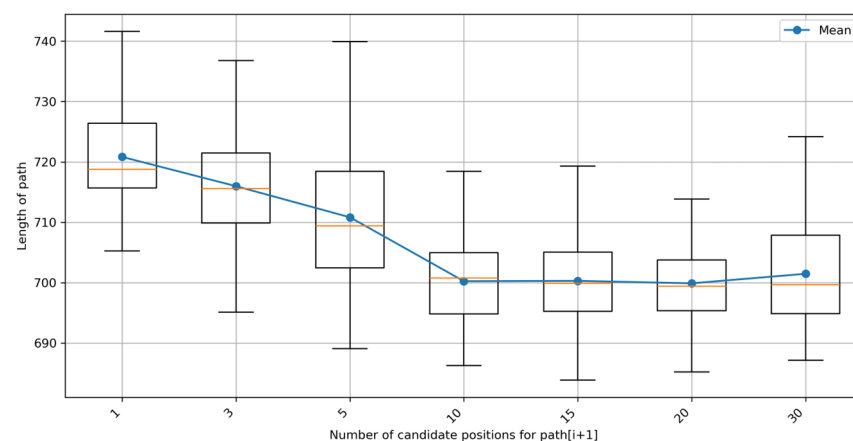
**Figure 26.** Experimental results for the number of candidate positions of  $\text{path}[i+1]$  according to the  $t$  increment in Case 1 of Map 1. The blue points indicate the mean. The orange line indicates the median (Q2).



**Figure 27.** Experimental results for the number of candidate positions of path[i+1] according to the  $t$  increment in Case 2 of Map 1.



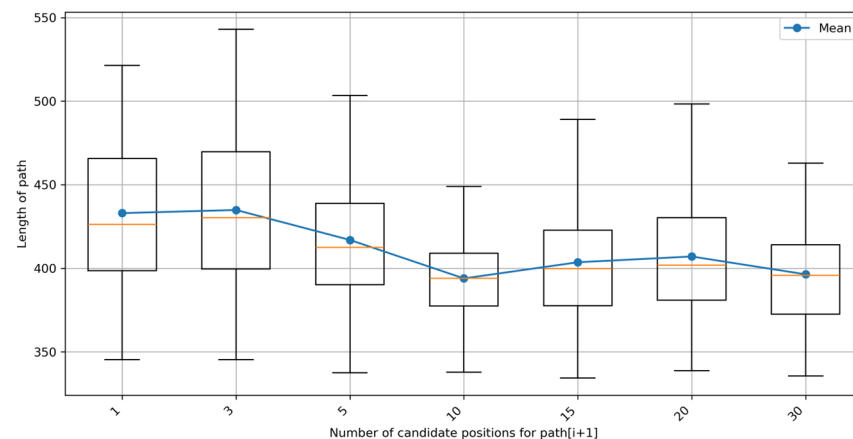
**Figure 28.** Experimental results for the number of candidate positions of path[i+1] according to the  $t$  increment in Case 1 of Map 2.



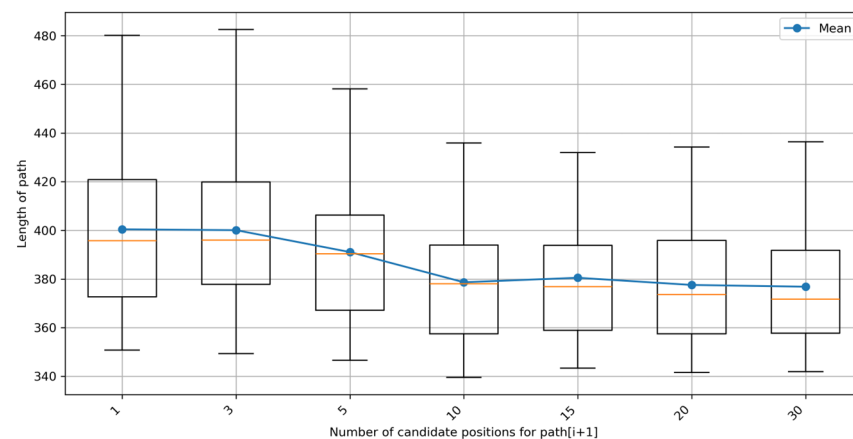
**Figure 29.** Experimental results for the number of candidate positions of path[i+1] according to the  $t$  increment in Case 2 of Map 2.

The results for Map 3 are shown in Figures 30 and 31, which correspond to Case 1 and Case 2, respectively. Since Map 3 is an obstacle-dense environment, we confirmed that the dispersion of the length of the path is improved only slightly, unlike in Map 1 and Map 2. The mean of the length of the path continues to decrease, but after 10 candidates, it shows a phenomenon similar to the results of Map 1 and Map 2. Therefore, based on

this experiment, we set the increment of  $t$  to 0.1 and conducted the experiments. The values of each experimental result are shown in Table 11.



**Figure 30.** Experimental results for the number of candidate positions of path[i+1] according to the  $t$  increment in Case 1 of Map 3.



**Figure 31.** Experimental results for the number of candidate positions of path[i+1] according to the  $t$  increment in Case 2 of Map 3.

**Table 11.** Results of the mean path length for the number of candidate positions of path[i+1] according to the  $t$  increment. The mean value is based on 100 repeated trials.

|       |        | 1      | 3      | 5      | 10     | 15     | 20     | 30     |
|-------|--------|--------|--------|--------|--------|--------|--------|--------|
| Map 1 | Case 1 | 828.26 | 822.27 | 785.00 | 705.26 | 703.44 | 700.47 | 703.07 |
|       | Case 2 | 439.20 | 429.99 | 418.86 | 390.44 | 389.34 | 390.10 | 390.16 |
| Map2  | Case 1 | 610.85 | 598.15 | 592.94 | 549.32 | 549.74 | 548.90 | 550.27 |
|       | Case 2 | 722.08 | 716.73 | 711.94 | 700.45 | 700.53 | 701.65 | 701.47 |
| Map3  | Case 1 | 438.18 | 436.87 | 427.34 | 405.17 | 407.00 | 410.90 | 403.85 |
|       | Case 2 | 405.61 | 403.06 | 399.20 | 380.75 | 383.38 | 381.37 | 378.94 |

## 6. Conclusions

This paper proposes the Straight-RRT algorithm, motivated by the observation that the Rewire procedure widely used in RRT-based path planning improves path quality but increases computational cost and implementation complexity due to hyperparameters, potentially becoming a bottleneck in planning speed. Straight-RRT rapidly obtains an initial solution without rewiring and subsequently improves the path through post-processing. The proposed method rapidly finds an initial path during the tree expansion

phase using a straight strategy, and then efficiently improves the path length via MoveParent post-processing. The straight strategy provides an approach that fills the gap in achieving rapid acquisition of an initial path while reducing parameter dependence. In addition, MoveParent improves the limitations of the triangle-inequality-based method. Experimental results show that the Straight strategy generates an initial path very quickly, and that MoveParent also effectively improves the path.

From the results of six types of experiments, the proposed algorithm shows an average of 8.75% in planning time compared to the baseline algorithm Q-RRT\*. This shows a much lower value than other algorithms, F-RRT\* and MQ-RRT\*, which show averages of 80.33% and 32.55%, respectively. This indicates that the proposed method is significantly more advantageous in terms of planning speed. In the length of the path, it also shows an average of 98.10%, showing that it is more efficient than MQ-RRT\* (104.83%) and Q-RRT\*. Although the proposed method computes a slightly longer path than F-RRT\* (93.6%), it achieves a lower cumulative heading change, with an average of 74.38 compared to 79.73 for F-RRT\*. This indicates that, even if the path is somewhat longer, the proposed method can sufficiently offset this disadvantage in terms of planning time and cumulative heading change. It also shows an advantage in cumulative heading change over Q-RRT\* and MQ-RRT\* (118.89).

Straight-RRT can be effectively applied in environments with weak curvature constraints on obstacles and few dynamic obstacles, such as humans. The scope of this paper is to plan paths faster than existing algorithms while generating shorter path lengths. In environments with stringent curvature constraints, by leveraging processing methods that take the dynamics into account [24,25], the proposed approach is expected to handle abrupt changes in heading angle. This is because the generated path may include segments that are sharp or involve abrupt turns, which may not adequately satisfy kinodynamic constraints. Although the evaluation metrics include the cumulative heading change, it simply represents the accumulation of angular changes. In other words, it does not consider physical criteria caused by abrupt changes in angles. In addition, realistic physical factors, such as the robot's mass, acceleration, inertia, and energy consumption, were not considered. To address these limitations, future work needs to compute a realistic path by considering the physical world. Also, there is the aspect of applicability in a three-dimensional environment. The experiments were conducted as path planning of a mobile robot and were carried out on a two-dimensional environment map without a z coordinate. Therefore, the applicability in a three-dimensional environment with a z coordinate has not been confirmed. To compensate for these limitations, it is necessary to extend the study by considering not only two dimensions but also three dimensions.

**Author Contributions:** Idea and conceptualization: J.J. and J.-W.J.; methodology: J.J. and J.-W.J.; software: J.J. and J.-W.J.; experiment: J.J. and J.-W.J.; validation: J.J. and J.-W.J.; investigation: J.J. and J.-W.J.; resources: J.J. and J.-W.J.; writing: J.J. and J.-W.J.; visualization: J.J. and J.-W.J.; project administration: J.-W.J. All authors have read and agreed to the published version of the manuscript.

**Funding:** This research was financially supported by the Ministry of Trade, Industry, and Energy (MOTIE) and Korea Institute for Advancement of Technology (KIAT) through the International Cooperative R&D program. Project No. P0026318 was supported by the Ministry of Trade, Industry, and Energy (MOTIE) and Korea Institute for Advancement of Technology (KIAT) through the International Cooperative R&D program. Project No. P0030503 was supported by the Commercialization Promotion Agency for R&D Outcomes (COMPA) grant funded by the Korean government (Ministry of Science and ICT) (2710086167), by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2025-2020-0-01789) supervised by the IITP (Institute for Information and Communications Technology Planning and Evaluation), and by the Artificial Intelligence Convergence Innovation Human Resources Development

(IITP-2025-RS-2023-00254592), supervised by the IITP (Institute for Information and Communications Technology Planning and Evaluation).

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

**Informed Consent Statement:** Not applicable.

**Data Availability Statement:** The original contributions presented in this study are included in the article. Further inquiries can be directed to the authors.

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

## References

1. Gao, H.; Liu, D.; Hu, J. A Survey on Path Planning for Mobile Robot Systems. In Proceedings of the 2023 IEEE 12th Data Driven Control and Learning Systems Conference (DDCLS), Xiangtan, China, 12–14 May 2023; pp. 1176–1181.
2. Xu, T. Recent Advances in Rapidly-Exploring Random Tree: A Review. *Heliyon* **2024**, *10*, e32451.
3. LaValle, S.M. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*; Springer: London, UK, 1998.
4. Kuffner, J.J., Jr.; LaValle, S.M. RRT-Connect: An Efficient Approach to Single-Query Path Planning. In Proceedings of the IEEE International Conference on Robotics and Automation, San Francisco, CA, USA, 24–28 April 2000; Volume 2, pp. 995–1001.
5. Ferguson, D.; Stentz, A. Anytime RRTs. In Proceedings of the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Beijing, China, 9–15 October 2006; pp. 5369–5375.
6. Gao, J.; Zhang, M. Rapidly Exploring Random Tree-Back (RRT-Back). In Proceedings of the 2025 International Conference on Artificial Life and Robotics (ICAROB 2025), Oita, Japan, 13–16 February 2025; Volume 30, pp. 83–86.
7. Karaman, S.; Frazzoli, E. Sampling-Based Algorithms for Optimal Motion Planning. *Int. J. Robot. Res.* **2011**, *30*, 846–894.
8. Naderi, K.; Rajamäki, J.; Hämmäläinen, P. RT-RRT\*: A Real-Time Path Planning Algorithm Based on RRT\*. In Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games (MIG '15), Paris, France, 16–18 November 2015; pp. 113–118.
9. Armstrong, D.; Jonasson, A. AM-RRT\*: Informed Sampling-Based Planning with Assisting Metric. In Proceedings of the 2021 IEEE International Conference on Robotics and Automation (ICRA), Xi'an, China, 30 May–5 June 2021; pp. 10093–10099.
10. Li, W.; Wang, C. Fast AM-RRT\*: A Fast and Efficient Path Planning Algorithm Based on AM-RRT\* in Complex Environments. In Proceedings of the 2024 International Conference on Virtual Reality and Visualization (ICVRV), Macao SAR, China, 18–20 October 2024; pp. 94–99.
11. Viseras, A.; Losada, R.O.; Merino, L. Planning with Ants: Efficient Path Planning with Rapidly Exploring Random Trees and Ant Colony Optimization. *Int. J. Adv. Robot. Syst.* **2016**, *13*, 1729881416664078.
12. Wang, J.; Zheng, E. Path Planning of Mobile Robot Based on the Improved RRT-Connect Algorithm. In Proceedings of the 2024 3rd International Conference on Service Robotics (ICoSR), Hangzhou, China, 2024; pp. 34–38.
13. Wen, J.; Zhang, Y.; Tian, J.; Chen, Q. An Improved RRT\* Algorithm for Mobile Robot Path Planning. In Proceedings of the 2025 8th International Conference on Advanced Algorithms and Control Engineering (ICAACE), Shanghai, China, 21–23 March 2025; pp. 109–114.
14. Coelho, F.O.; Carvalho, J.P.; Pinto, M.F.; Marcato, A.L. Direct-DRRT\*: A RRT Improvement Proposal. In Proceedings of the 2018 13th APCA International Conference on Automatic Control and Soft Computing (CONTROLO), Ponta Delgada, Portugal, 4–6 June 2018; pp. 154–158.
15. Lopes, M.S.; Silva, M.F.; de Souza, J.P.C.; Costa, P. Direct-Steered-DRRT\*: A 3D RRT-Based Planner Improvement. In Proceedings of the 2024 IEEE International Conference on Autonomous Robot Systems and Competitions (ICARSC), Paredes de Coura, Portugal, 2–4 May 2024; pp. 100–105.
16. 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, Chicago, IL, USA, 14–18 September 2014; pp. 2997–3004.
17. Nasir, J.; Islam, F.; Malik, U.; Ayaz, Y.; Hasan, O. RRT\*-Smart: Rapid Convergence Implementation of RRT\* towards Optimal Solution. *Int. J. Adv. Robot. Syst.* **2013**, *10*, 299.
18. Kang, J.-G.; Lim, D.-W.; Choi, Y.-S.; Jang, W.-J.; Jung, J.-W. Improved RRT-Connect Algorithm Based on Triangular Inequality for Robot Path Planning. *Sensors* **2021**, *21*, 333.
19. 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.

20. 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.
21. Cui, X.; Wang, C.; Xiong, Y.; Mei, L.; Wu, S. More Quickly-RRT\*: Improved Quick Rapidly-Exploring Random Tree Star Algorithm Based on Optimized Sampling Point with Better Initial Solution and Convergence Rate. *Eng. Appl. Artif. Intell.* **2024**, *133*, 108246.
22. Dong, X.; Wang, Y.; Fang, C.; Ran, K.; Liu, G. FHQ-RRT\*: An Improved Path Planning Algorithm for Mobile Robots to Acquire High-Quality Paths Faster. *Sensors* **2025**, *25*, 2189.
23. Cong, J.; Hu, J.; Wang, Y.; Zhou, W. FF-RRT\*: A Sampling-Improved Path Planning Algorithm for Mobile Robots against Concave Cavity Obstacle. *Complex Intell. Syst.* **2023**, *9*, 7249–7267.
24. Kang, J.-G.; Choi, Y.-S.; Jung, J.-W. A Method of Enhancing Rapidly-Exploring Random Tree Robot Path Planning Using Midpoint Interpolation. *Appl. Sci.* **2021**, *11*, 8483.
25. Kang, T.-W.; Kang, J.-G.; Jung, J.-W. A Bidirectional Interpolation Method for Post-Processing in Sampling-Based Robot Path Planning. *Sensors* **2021**, *21*, 7425.

**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.