Journal of Computational Design and Engineering, 2022, 9(2), 755–771

https://doi.org/10.1093/jcde/qwac025
Journal homepage: www.jcde.org

RESEARCH ARTICLE

Target-biased informed trees: sampling-based method for optimal motion planning in complex environments

Xianpeng Wang[1], Xinglu Ma[1], Xiaoxu Li[1], Xiaoyu Ma[2] and Chunxu Li[3,*]

[1]School of Information Science and Technology, Qingdao University of Science and Technology, Qingdao 266061, China; [2]School of Cyberspace Security, Beijing University of Posts and Telecommunications, Beijing 100876, China and [3]College of Mechanical and Electrical Engineering, Hohai University, Changzhou 213000, Jiangsu Province, China

*Corresponding author. E-mail: chunxuli90@gmail.com

Abstract

Aiming at the problem that the progressively optimized Rapidly-exploring Random Trees Star (RRT*) algorithm generates a large number of redundant nodes, which causes slow convergence and low search efficiency in high-dimensional and complex environments. In this paper we present Target-biased Informed Trees (TBIT*), an improved RRT* path planning algorithm based on target-biased sampling strategy and heuristic optimization strategy. The algorithm adopts a combined target bias strategy in the search phase of finding the initial path to guide the random tree to grow rapidly toward the target direction, thereby reducing the generation of redundant nodes and improving the search efficiency of the algorithm; after the initial path is searched, heuristic sampling is used to optimize the initial path instead of optimizing the random tree, which can benefit from reducing useless calculations, and improve the convergence capability of the algorithm. The experimental results show that the algorithm proposed in this article changes the randomness of the algorithm to a certain extent, and the search efficiency and convergence capability in complex environments have been significantly improved, indicating that the improved algorithm is feasible and efficient.

Keywords: path planning; rapidly exploring random trees; improved RRT*; target bias; heuristic

1. Introduction

With the evolution of a new round of technological revolution and industrial transformation, the robotics industry is booming. Scholars have carried out in-depth research on robots in various fields. Su *et al.* proposed an incremental learning framework for human-like redundancy optimization of anthropomorphic manipulators (Su *et al.*, 2022), and the trajectory control of redundant robotic manipulators using remote center of motion constraints is performed through an improved recurrent neural network scheme (Su *et al.*, 2020). By integrating technologies, Su *et al.* propose a novel system that enables robots to highly intelligently learn the skills of advanced surgeons and perform the learned surgical manipulations in future semi-autonomous surgeries (Su *et al.*, 2021). With the growth of market demand, various types of service robots, such as commercial robots, medical robots, household robots, distribution robots, and robots for the elderly and disabled, have rapidly penetrated into our life or work scenarios.

Path planning technology is one of the key technologies to realize mobile robot navigation, which affects the efficiency and safety of mobile robot navigation process. For the problem of robot path planning, scholars/researchers have carried out a lot of research works. According to the implementation of the algorithm, path planning algorithms can be divided into traditional planning algorithms, intelligent planning algorithms, and sampling-based planning algorithms. Traditional planning

Received: 17 October 2021; **Revised:** 4 February 2022; **Accepted:** 22 February 2022

algorithms mainly include artificial potential field method (Zhang *et al.*, 2006), bug algorithm (Lumelsky & Stepanov, 1986), vector histogram method (Borenstein & Koren, 1991), grid method (Li *et al.*, 2019), visual method (Willms & Yang, 2006), and so on. Traditional planning algorithms tend to fall into local minimums when solving complex problems in narrow spaces. With the increase of the robot's degree of freedom, the planning efficiency of traditional algorithms is greatly reduced, and the memory consumption is significantly increased. Intelligent planning algorithms include fuzzy logic method (Lei & Li, 2007), genetic algorithm (Baba & Kubota, 1993), and neural network method (Liu *et al.*, 2007). Although intelligent algorithms overcome the local minimum problem of traditional algorithms and have good adaptability to the environment, similar to traditional algorithms, as obstacles become more complicated and spatial dimensions increase, their planning efficiency and success rate decrease.

To solve the above problems, scholars have proposed a path planning algorithm based on sampling. The sampling-based algorithm does not use the explicit representation of the environment, but obtains the node information of the feasible trajectory through collision detection. Then, the collision-free nodes in the space are connected to form a feasible trajectory graph (road map). Finally, a feasible path from the initial state to the target state is obtained in the road map. The two most commonly used sampling-based path planning algorithms are the probabilistic roadmaps (PRM) algorithm and the rapidly exploring random trees (RRT; LaValle, 1998). The PRM algorithm is a multiple query method, which has been proven to have good planning performance in high-dimensional spaces (Kavraki *et al.*, 1996). However, the randomness of PRM sampling results in a greatly increased probability of path planning failures in maps containing narrow passages. Additionally, a priori calculated roadmap may be computationally challenging or even not feasible in some applications. The RRT algorithm has been widely used and developed in various fields. The reason is that the RRT algorithm has probabilistic completeness and is suitable for differential constraints. The sampling method of the RRT algorithm is completely random, which also leads to its low planning efficiency and poor path optimization ability. Therefore, in view of the limitations of the RRT algorithm, researchers have proposed many improved algorithms in hopes of optimizing the algorithm in different aspects. The most classic is the rapidly exploring random trees star (RRT*; Karaman & Frazzoli, 2011) algorithm proposed by Sertac and Emilio in 2011. The algorithm gradually reduces the cost of generating the path by reselecting the parent node and rewiring. After the initial solution is generated, the algorithm will continue to execute to optimize the initial path until the end of the iteration. As the number of iterations increases, the executable path is continuously optimized and gradually converges to the optimal value. Since the sampling space of the RRT* algorithm is still the entire space region, as the number of iterations increases, some useless points will be added to the random search tree. This will lead to excessive memory consumption and unnecessary computation, which will slow down the convergence speed in the subsequent optimization process. The convergence speed during the optimization process is reduced. In response to the above shortcomings, Qureshi introduced artificial potential field into the RRT* algorithm (Qureshi & Ayaz, 2016) and increased the goal orientation of random tree growth based on preserving the randomness of random tree growth, thereby accelerating the convergence speed of random tree. Iram introduced offline planning algorithm RRT*-adjustable bounds (RRT*-AB) based on RRT*. This method improves computational efficiency by quickly aiming at the target area (Noreen *et al.*, 2011). The RRT*-smart algorithm (Nasir *et al.*, 2013) can adjust parameters according to online information, thereby reducing the cost of random tree generation and speeding up convergence. Drawing lessons from the growth process of bidirectional RRT random trees, bidirectional rapidly exploring random trees algorithm (Jordan & Perez, 2013) is proposed, which accelerates the convergence of random trees. The rapid expansion of random tree RRT (Informed RRT*) algorithm based on heuristic guidance (Mashayekhi *et al.*, 2020) uses heuristic functions to limit sampling points to specific areas and reduce the growth cost of random trees. Janson proposed Fast Marching Trees (FMT*; Janson *et al.*, 2015), which uses a marching method to process a single set of samples. The resulting search is ordered on cost-to-come but must be restarted if a higher resolution is needed. Gammell proposed the Batch Informed Trees (BIT*) algorithm (Gammell *et al.*, 2020), which samples batch of states and views these sample states as an increasingly dense edge implicit random geometric graph. BIT* effectively reuses information from previous searches and approximations by using incremental search technology, but does not update its heuristic search during the search process.

The above algorithms have played a certain role in optimizing RRT*, but the strong randomness of the RRT* algorithm under high-dimensional space or complex constraints will still cause problems such as many invalid nodes, huge amount of redundant calculation, and slow convergence speed, which reduce the working efficiency of the robot. Therefore, this paper proposes an improved RRT* path planning algorithm target-biased informed trees (TBIT*) based on target-biased sampling strategy and heuristic optimization strategy. By optimizing the sampling point generation method and improving the path optimization rules, the algorithm can quickly plan a high-quality path in a complex environment containing complex obstacles and narrow passages, and finally verify the feasibility of the algorithm in a complex environment through simulation experiments.

2. Path Planning Problem Definition

Suppose the state space of the planning task is X, $X_{\text{obs}} \subset X$ represents the area where all obstacles in the workspace are located, and $X_{\text{free}} = X/X_{\text{obs}}$ denotes the free state space in the workspace, which refers to all reachable areas in the workspace except obstacles. $x_{\text{init}} \subset X_{\text{free}}$ is the initial pose or starting point, and $x_{\text{goal}} \subset X_{\text{free}}$ represents the target pose or target point. The continuous function $\pi : [0, 1] \to X_{\text{free}}$ is a collision-free feasible path from the starting point to the target point in free space, where $\pi(0) = x_{\text{init}}$ and $\pi(1) = x_{\text{goal}}$. The path cost function is represented by $C(\pi)$, $C_{\text{best}} = \min\{C(\pi)\}$ represents the cost of the best feasible path, and $\pi_* = \mathit{argmin}\ \{C(\pi)\}$ denotes the current best feasible path. This paper only considers off-line planning in the Euclidean space and positive Euclidean distance between any two states. The path planning problem is defined as: finding a collision-free feasible path π_* from the starting point x_{init} to the target point x_{goal} with the minimum cost $C(\pi)$ in the free state space X_{free}.

3. Principles of Related Algorithms

The TBIT* algorithm proposed in this paper improves the sampling method of the RRT* algorithm through the target bias strategy, and improves the speed of searching the initial path; the heuristic optimization strategy in the Informed RRT* algorithm is applied to optimize the initial path, which greatly increases the rate of path optimization. The following describes the implementation principles of related algorithms.

3.1. Rapidly exploring random trees star

The basic structure of the RRT* algorithm is similar to that of the RRT algorithm. They both use the starting point as the root node of the random tree and perform random sampling in the free space with no obstacles to find appropriate leaf nodes to add to the random tree. When the target point is also added to the random tree, a collision-free feasible path from the initial point to the target point can be found. If the collision-free feasible path from the initial point to the target point is not found when the maximum number of iterations is reached, the plan is considered mission failed. Based on the RRT algorithm, the RRT* algorithm has made two changes: one is that when a new leaf node is added, the RRT* algorithm ensures that the path corresponding to the new node has a local minimum by reselecting the parent node; the second is to reroute the random tree after adding new nodes to ensure the gradual optimality of the generated path. The pseudo-code of the RRT* algorithm is as follows:

Algorithm 1. RRT* algorithm pseudo-code

```
T ← RRT*($x_{init}$, $x_{goal}$)
1.  T ← EmptyTree()
2.  for i = 0 to N
3.    $x_{rand}$ ← RandomSample($X_{free}$)
4.    $x_{nearest}$ ← NearestNode($x_{rand}$, T)
5.    $x_{new}$ ← Steer($x_{nearest}$, $x_{rand}$)
6.    if CollisionFree($x_{nearest}$, $x_{new}$) then
7.       $X_{near}$ ← NearNeighbors($x_{new}$, T)
8.       $x_{parent}$ ← ChooseParent($X_{near}$, $x_{new}$)
9.       T ← InsertNode($x_{new}$, $x_{parent}$, T)
10.      T ← Rewire($x_{new}$, $x_{parent}$, $X_{near}$)
11.      if CheckToGoal($x_{new}$, $x_{goal}$) then
12.          T ← InsertNode($x_{goal}$, $x_{new}$, T)
13.  return T
```

The following are the functions of the main functions in the RRT* algorithm:

InsertNode: Insert a node into the random tree.

RandomSample: Random sampling in the free state space.

NearestNode: Find the node closest to the sampling point in the random tree.

CollisionFree: Detect whether the path between two points collides with obstacles in the environment.

Steer: The node closest to the random sampling point in the random tree extends a certain distance to the random sampling point to generate a new node.

NearNeighbors: Calculate the set of nodes within a specific range from a node in the random tree.

ChooseParent: Calculate the node closest to the newborn node in the random tree and use it as the parent node of the newborn node.

Rewire: Rewiring operation. Calculate the path cost of all nodes in the set of neighboring nodes in turn when the new node is the parent node. If the new path has a lower path cost compared with the original path, the new node is taken as its parent node.

CheckToGoal: Check whether the path between the new node and the target point collides with obstacles in the environment.

3.2. Informed rapidly exploring random trees star

The RRT* algorithm samples in the entire state space, so as the number of iterations increases, useless sampling points gradually increase, which will bring a huge number of useless calculations and slow the convergence of the algorithm. The principle of the Informed RRT* algorithm to find the initial path is the same as that of the RRT* algorithm, but after the Informed RRT* algorithm obtains the initial feasible solution, it uses heuristic sampling to limit the sampling points to the hyper-ellipsoid sampling space generated by the minimum cost of the current feasible solution, reducing the sampling of useless areas and accelerating the convergence of the algorithm.

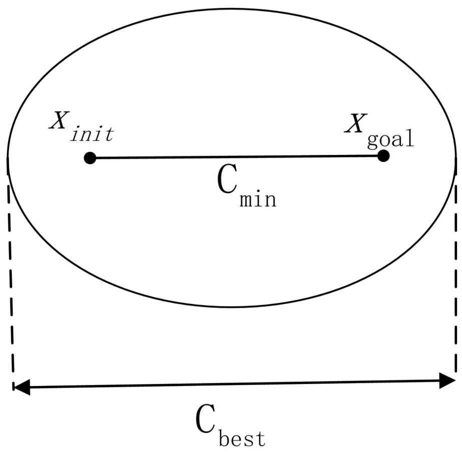

Figure 1: Schematic diagram of heuristic sampling area.

As shown in Fig. 1, when the algorithm is sampling in 2D space, the heuristic sampling area is an ellipse, the focal length C_{min} of the ellipse is the distance between x_{init} and x_{goal}, and the long axis is the shortest path cost C_{best}. As the number of iterations increases, the current path cost C_{best} decreases continuously, which makes the sampling area to gradually decrease, useless sampling points decrease, and the algorithm convergence speed increases.

The pseudo-code of the Informed RRT* algorithm is shown in Algorithm 2. Among them, X_{soln} is a heuristic subset, and *HeuSample* function is a heuristic sampling function. When C_{best} is less than positive infinity, the sampling area will be limited to the heuristic subset. From the comparison of the pseudo-code of the two algorithms, the Informed RRT* algorithm only changes the sampling method.

Algorithm 2. Informed RRT* algorithm pseudo-code

```
T ← Informed RRT*($x_{init}$, $x_{goal}$)
1.  T ← EmptyTree()
2.  T ← InsertNode($x_{init}$, T)
3.  $C_{best}$ ← ∞
4.  $X_{soln}$ ← ∅
5.  for i = 0 to N
6.    $C_{best} = \min_{x \in X_{soln}}$ (Cost(x))
7.    $x_{rand}$ ← HeuSample($x_{init}$, $x_{goal}$, $C_{best}$)
8.    $x_{nearest}$ ← NearestNode($x_{rand}$, T)
9.    $x_{new}$ ← Steer($x_{nearest}$, $x_{rand}$)
10.   if CollisionFree($x_{nearest}$, $x_{new}$) then
11.      $X_{near}$ ← NearNeighbors($x_{new}$, T)
12.      $x_{parent}$ ← ChooseParent($X_{near}$, $x_{new}$)
13.      T ← InsertNode($x_{new}$, $x_{parent}$, T)
14.      T ← Rewire($x_{new}$, $x_{parent}$, $X_{near}$)
15.      if CheckToGoal($x_{new}$, $x_{goal}$) then
16.         T ← InsertNode($x_{goal}$, $x_{new}$, T)
17.         $X_{soln}$ ← $X_{soln}$ ∪ $x_{new}$
18. return T
```

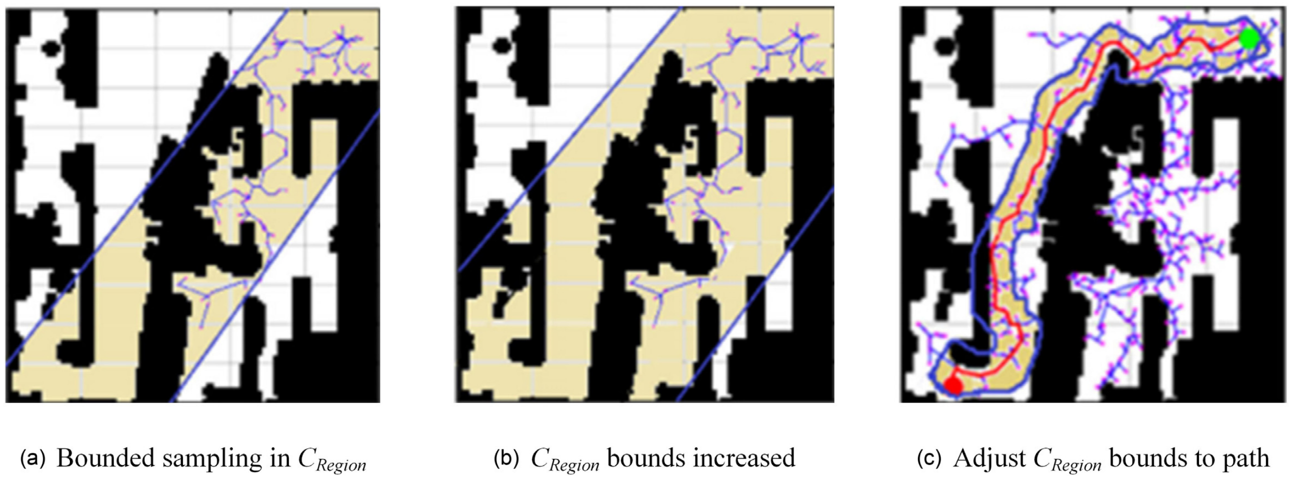
(a) Bounded sampling in C_{Region}
(b) C_{Region} bounds increased
(c) Adjust C_{Region} bounds to path

Figure 2: RRT*-AB schematic diagram.

3.3. Rapidly exploring random trees star-adjustable bounds

The RRT*-AB algorithm improves the efficiency of the algorithm through three strategies: connectivity region, goal-biased bounded sampling, and path optimization. Goal-biased bounded sampling is performed within the boundary of connectivity region to find the initial path. Once the initial path is found, it is optimized gradually using node rejection and concentrated bounded sampling. Final path is further improved using global pruning to erode extra nodes.

When the algorithm is running, first determine the sectioned sampling area C_{Region} and sample in this area, as shown in Fig. 2a. When the path cannot be found in the first scan of C_{Region}, the C_{Region} is doubled, as shown in Fig. 2b. When the initial path is searched, the algorithm redefines the connection area along the initial path, and optimizes the initial path through sampling in this area until the end of the iteration, as shown in Fig. 2c.

4. Target-Biased Informed Trees

To reduce the path search time and reduce the final feasible path cost, this paper proposes an improved RRT* algorithm based on the target bias strategy and heuristic optimization strategy. The algorithm is divided into a search phase and an optimization phase. In the search phase, the algorithm uses a combined target bias strategy to obtain sampling points, which reduces the randomness of sampling points while reducing the cost of adding sampling points to the random tree, so as to quickly search for an initial path with a relatively low path cost; in the optimization phase, the heuristic optimization strategy defines the sampling area as a hyperellipsoid region composed of waypoints in the initial path, which makes the collision-free sampling points obtained in this sampling area can definitely reduce the path cost. It overcomes the redundant calculations generated when sampling points are added to the random tree in the optimization stage of the Informed RRT* algorithm, which greatly improves the efficiency of path optimization.

The pseudo-code of TBIT* is as shown in Algorithm 3. After initializing the parameters, the algorithm enters the search phase (lines 2–7), and the random tree is expanded using the combined target bias strategy (line 3). The random tree expansion method is determined according to the failure rate of random tree node expansion. In the search phase, the target bias strategy is used to reduce the randomness of random tree growth, and the initial path can be obtained faster. After finding the initial path P, the algorithm enters the optimization phase (lines 8–10). In this stage, the initial path is optimized, instead of optimizing all the nodes in the initial random tree. The initial path is globally pruned by the greedy algorithm (line 8) to reduce the path cost. Then, the algorithm uses heuristic sampling (line 9) through iterative methods to obtain sampling points that do not collide with obstacles in the hyper-ellipsoid area formed by nodes in the initial path to reduce the cost of the initial path. The B-spline is used to optimize the path (line 10) to make the final path more consistent with the laws of robot kinematics.

Algorithm 3. TBIT* algorithm pseudo-code

```
P ← TBIT*($x_{init}$, $x_{goal}$)
1.  T ← EmptyTree()
2.  while PathNotFound(T) do
3.     $x_{rand}$ ← TargetBiasSample($x_{goal}$, T)
4.     $x_{nearest}$ ← NearestNode($x_{rand}$, T)
5.     $x_{new}$ ← Steer($x_{nearest}$, $x_{rand}$)
6.     if CollisionFree($x_{nearest}$, $x_{new}$) then
7.        T ← InsertNode($x_{new}$, T)
8.  P ← GlobalPruning(P)
9.  P ← Optimize_HSample(T)
10. P ← BSpline(P)
11. return P
```

4.1. Searching stage

In the search stage, the combined target bias sampling includes a sunflower area sampling and a target pulling area sampling. The algorithm dynamically selects the sampling method according to the current node expansion failure rate. Expansion failure rate R_{ef} is the ratio of the number of expansion failures of the random tree to the total number of expansions. When the expansion failure rate is low, it means that obstacles have little effect on the expansion of the random tree, and the random tree tends to sample in the sunflower area and the target traction area, so that the random tree stretches toward the target point. On the contrary, it shows that obstacles have a greater impact on the expansion of the random tree, and the algorithm tends to sample randomly throughout the space and strengthen the algorithm's exploration of unknown areas so as to escape the obstacle area at a relatively small cost. The pseudo-code of the sampling algorithm in the search phase is shown in Algorithm 4, where r_1 and r_2 are constants. In this paper, r_1 and r_2 are, respectively, 0.4 and 0.8.

Algorithm 4. Combined target bias strategy pseudo-code

```
$x_{rand} \leftarrow$ TargetBiasSample($x_{goal}$, T)
1.  if $R_{ef} \le r_1$
2.      $x_{rand} \leftarrow$ SunFlowerSample($x_{goal}$, T)
3.  else if $R_{ef} \le r_2$
4.      $x_{rand} \leftarrow$ GoalTractionSample($x_{goal}$, T)
5.  else
6.      $x_{rand} \leftarrow$ RandomSample($x_{goal}$, T)
```

4.1.1. *Sunflower regional sampling*

The sunflower area is a hyper-hemispherical area with the maximum expected expansion node x_{max} as the center of the sphere. S' is the hyper-ellipsoid cross-section through the center of the sphere x_{max} and the line S connecting the center of the circle x_{max} and x_{goal} is the normal line of S'. The section S' divides the hypersphere area into two hyper hemisphere areas, where the hyperhemisphere area intersecting the line segment S is the sunflower sampling area, and sampling in this area can improve the efficiency of the random tree expansion to the target point. The radius R of the sunflower area is dynamically changed by the obstacles encountered during the expansion. The initial value of the radius R is $\sigma \cdot L$, where L is the length of the map edge and σ is a constant.

Algorithm 5. Pseudo-code of sunflower area sampling

```
$x_{rand} \leftarrow$ SunFlowerSample($x_{goal}$, T)
1.  $r \leftarrow \sigma \cdot L$
2.  $x_{max} \leftarrow$ MaxCost($x_{goal}$, T)
3.  if IsCollision == False then
4.      $x_{rand} \leftarrow x_{max} + r \cdot (\cos\theta, \sin\theta)$
5.  else
6.      $r \leftarrow k \cdot r$
7.      $x_{rand} \leftarrow$ RandCirle(r, $x_{max}$)
8.      while Diameter($x_{max}$, $x_{goal}$, $x_{rand}$)
9.          $x_{rand} \leftarrow$ RandCirle(r, $x_{max}$)
```

The pseudo-code of the sunflower area sampling algorithm is shown in Algorithm 5. During the expansion of the random tree, the radius value is dynamically adjusted by the coefficient k. In the random tree expansion process, when the obstacles in the sampling area increase, the expansion failure rate increases. At this time, the radius of the sampling area should be reduced to accelerate the escape from the multi-obstacle area. When there are fewer obstacles in the sampling area, the expansion failure rate is reduced. At this time, increasing the radius of the sampling area is beneficial to improve the path search efficiency. In this paper, the coefficient k is determined by the failure expansion rate. When $r_1 < R_{ef} \le (r_1 + r_2)/2$, k takes the value $1 + 4R_{ef}$, and when $(r_1 + r_2)/2 < R_{ef} \le r_2$, k takes the value $1.5R_{ef}$.

The functions $\hat{g}_T(x)$ and $\hat{h}(x)$ represent admissible estimates of the cost-to-come to a state through the random tree, $x \in X$, from the start and the cost-to-go from a state to the goal. The function $\hat{f}(x)$ represents the total estimated cost from the starting point to the target point constrained to pass through x, and $\hat{f}(x) = \hat{g}_T(x) + \hat{h}(x)$. We assume that the estimated cost $\hat{f}(x_{init})$ of the starting point x_{init} is infinite before the initial path is found. Before sampling the sunflower area, the function *MaxCost* calculates and compares the total estimated cost $\hat{f}(x)$ of all nodes in the random tree to find the node x_{max} with the smallest estimated cost.

Figure 3 shows the node expansion diagram in a simple 2D environment, where the sunflower sampling area is a semicircular area, S' is a straight line passing through the center of the circle, and the shaded part is an obstacle. In the process of random tree expansion, the flag bit *IsCollision* will change according to the result of collision detection. As shown in Fig. 3a, when the random tree is expanded in the sunflower area, the first expansion of x_{max} will extend the distance r along the line segment S to the target point to generate a new node x_{new}. When this method fails to expand, the collision detection result resets the flag *IsCollision*, and a new node x_{new} will be randomly generated in the sunflower area, as shown in Fig. 3b. When sampling randomly in the sunflower area, first randomly generate a point in a circle with x_{max} as the center and R as the radius, and then calculate the linear equation of the line S'. When a random point appears in the semicircular area where the circle and the lower area of the line S' intersect, x_{rand} is returned. The function *Diameter* is used to determine whether a random point is in the area below the straight line S'. During the sampling process, the radius of the sunflower area is dynamically adjusted according to the expansion failure rate.

The *RandCirle* function is mainly used for random and uniform sampling in a circle with x_{max} as the center and r as the radius. The coordinate transformation of points in the polar coordinate system in a 2D environment is shown in formulas (1) and (2):

$$x = r \cdot \cos\theta \tag{1}$$

$$y = r \cdot \sin\theta. \tag{2}$$

Assuming that the probability density function of polar coordinates is $f(r, \theta)$, the corresponding Jacobian matrix when calculating the rectangular coordinate system is shown in formula (3):

$$\begin{pmatrix} \frac{\partial x}{\partial r} & \frac{\partial x}{\partial \theta} \\ \frac{\partial y}{\partial r} & \frac{\partial y}{\partial \theta} \end{pmatrix}. \tag{3}$$

Supposing the determinant value to be $r\,(\cos^2\theta + \sin^2\theta) = r$, therefore, the formula (4) can be used to realize the conversion from the polar coordinate system to the rectangular coordinate system:

$$f(r, \theta) = r \cdot f(x, y). \tag{4}$$

When sampling uniformly in the unit circle, $f(x, y) = \frac{1}{\pi}$, which is a constant, then $f(r, \theta) = \frac{r}{\pi}$, from which formulas (5) and (6) can be obtained:

$$f(r) = \sum_{0}^{2\pi} f(r, \theta)\, d\theta = \sum_{0}^{2\pi} \frac{r}{\pi} d\theta = 2r \tag{5}$$

$$f(\theta|r) = \frac{f(r, \theta)}{f(r)} = \frac{1}{2\pi}. \tag{6}$$

From the above formula, when r is determined, $f(\theta|r)$ is a constant, we can further calculate the distribution function and

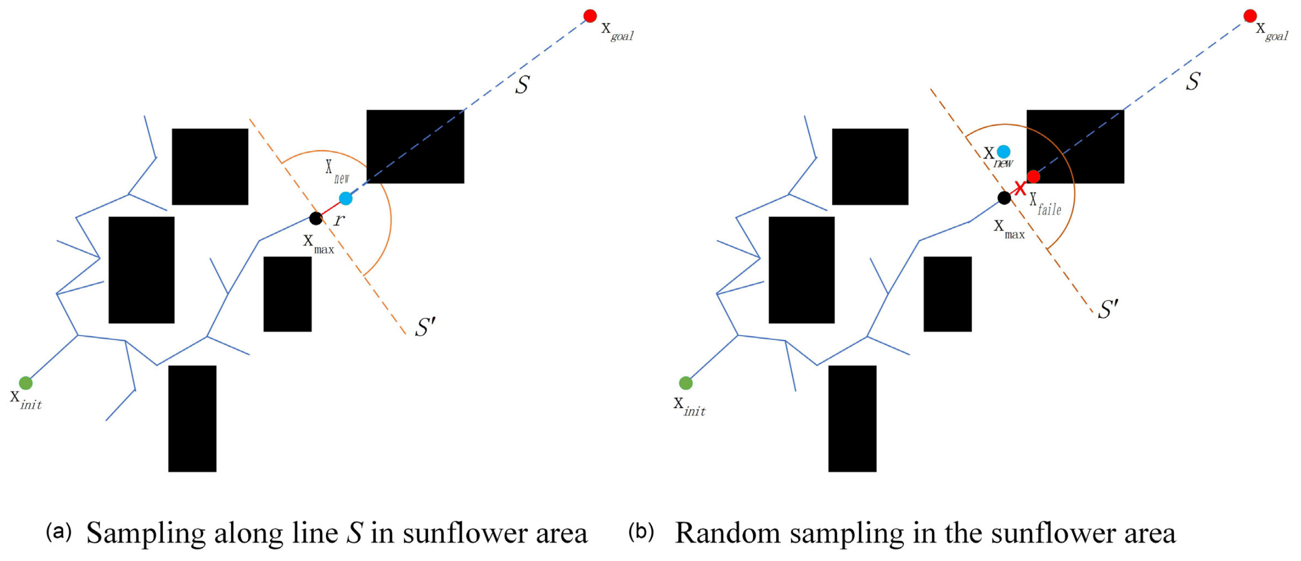

Figure 3: Sampling in sunflower area.

take the inverse function to get $r = \sqrt{\xi_1}, \theta = 2\pi\xi_2$, where ξ_1 and ξ_2 are random numbers. From the above derivation, when randomly and uniformly sampling in a circle with a radius of r, the coordinates of random points can be obtained according to the following formulas (7) and (8), where $0 \le \xi_1 \le r^2$, $0 \le \xi_2 \le 1$.

$$x = \sqrt{\xi_1}\cos(2\pi\xi_2) \tag{7}$$

$$y = \sqrt{\xi_1}\sin(2\pi\xi_2) \tag{8}$$

According to the formula derivation, the pseudo-code of *RandCirle* is shown in Algorithm 6.

Algorithm 6. Random sampling pseudo-code within a circle

```
$x_{rand} \leftarrow$ RandCirle(r, $x_{center}$)
1. $\xi_1 \leftarrow$ random(0, $r^2$)
2. $\xi_2 \leftarrow$ random(0, 1)
3. $x_{tem} \leftarrow (\sqrt{\xi_1}\cos 2\pi\xi_2,\ \sqrt{\xi_1}\sin 2\pi\xi_2)$
4. $x_{rand} \leftarrow x_{tem} + x_{center}$
```

4.1.2. *Sampling of target towing area*

When the influence of obstacles on the expansion of the random tree increases, the failure rate of the random tree expansion also increases. At this time, it is necessary to change the sampling strategy and appropriately increase the randomness of the sampling to speed up the random trees escape from the obstacle area. The target traction area is the hyperball area with x_{goal} as the center of the sphere. In 2D space, this area is a circular area with x_{goal} as the center; in three-dimensional (3D) space, this area is a spherical area with x_{goal} as the center of the sphere. Figure 4 is a schematic diagram of the target towing area in a 2D environment.

When sampling in the target pulling area, the sampling points are generated in the circular area centered on x_{goal}, which makes the sampling points guide the random tree to grow in the target direction. By sampling the target pulling area, while increasing the randomness of sampling, it reduces the generation probability of useless sampling points far away from the target point. This improves the quality of the sampling points and speeds up the growth of the random tree, so that the initial path can be obtained faster. Different from the sunflower area sampling, when the expansion failure rate of the target towing area sampling increases, the search radius will gradually increase to ensure quick escape from the obstacle area. The pseudo-code for sampling the target traction area is shown in Algorithm 7, where ε is a constant.

Algorithm 7. Pseudo-code of target pulling area sampling

```
$x_{rand} \leftarrow$ GoalTractionSample($x_{goal}$, T)
1. $x_{rand} \leftarrow$ RandCirle(r, $x_{goal}$)
2. if IsCollision = = False
3.     $r \leftarrow \varepsilon \cdot r$
```

4.2. Optimization stage

In the Informed RRT* algorithm, when the initial path is found, the algorithm starts to shrink the sampling space and restrict the sampling points to the ellipse. As the number of iterations increases, nodes with relatively small path costs are continuously added to the random tree, and the initial path is continuously optimized. However, with the increase of nodes, the Informed RRT* algorithm still performs re-parenting and re-wiring operations on each node, which brings a lot of redundant calculations. Based on the above shortcomings, in the optimization stage, this paper proposes the following three improvements to reduce the path cost and speed up the algorithm convergence.

4.2.1. *Changing the optimization object*

The Informed RRT* algorithm takes the starting point and the target point as the focus and the initial path cost as the major axis to construct the ellipse space. When sampling in this space, the sampling points need to go through a lot of calculations before they can be added to the random tree, but these points added to the random tree may not directly optimize the path, which also generates a lot of redundant calculations.

To speed up the algorithm convergence and reduce the redundant calculation caused by useless sampling points, this

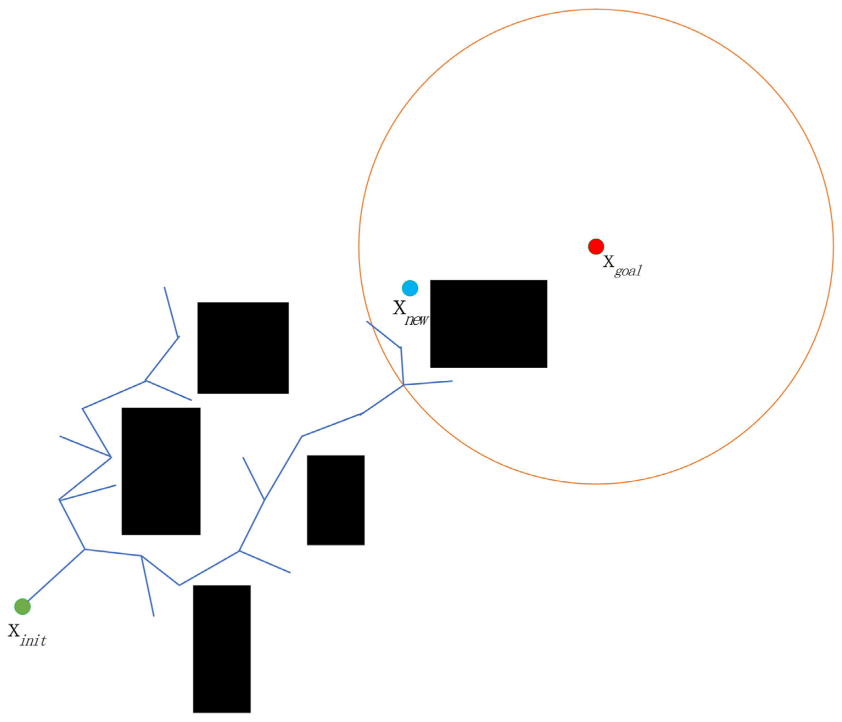

Figure 4: Schematic diagram of the target towing area.

paper finds the initial path and adjusts the optimization of the random tree to the optimization of the initial path. Using heuristic subsets to limit the sampling points to an ellipsoid area composed of nodes in the path, the collision-free nodes generated by sampling in this area must be able to optimize the path. This greatly reduces redundant calculations such as collision detection, rerouting, and reselection of parent nodes generated by sampling points that cannot optimize the path, thereby improving the efficiency of path cost convergence. The pseudo-code of the sampling function in the optimization stage is shown in Algorithm 8.

Algorithm 8. Pseudo-code of initial path optimization

```
P ← Optimize_HSample(T)
1.  P ← FindPath(T)
2.  N ← |P|
3.  i = 1
4.  while i < N − 1
5.     j = i + 2
6.     $C_{min} = (P_i - P_j)_2$
7.     $C_{max} = (P_{i+1} - P_j)_2 + (P_{i+1} - P_i)_2$
8.     $q_{center} = P_i + P_j/2$
9.     C ← RatationToWorldFrame($P_i$, $P_j$)
10.    $r_1 \leftarrow C_{max}/2$
11.    $\{r_k\}_{k=1,2,3,\ldots,n} \leftarrow \sqrt{C_{max}^2 - C_{min}^2}/2$
12.    $L = \mathrm{diag}\{r_1, r_2, r_3, \ldots, r_n\}$
13.    $q_{ball} =$ SampleUnitBall($Q_{ball}$)
14.    $q_{ellipse} \leftarrow (CL \cdot q_{ball} + q_{center}) \cap Q$
15.    if CollisionFree($q_{ellipse}$, $q_i$)
16.       if CollisionFree($q_{ellipse}$, $q_j$)
17.          Replace($q_{ellipse}$, $q_{i+1}$)
18.    i = i + 1
19. return P
```

In Algorithm 8, starting from the starting point, three nodes on the path are selected as the optimization objects for optimization until the target node is optimized. During the optimization process, the sampling area is a hyper-ellipsoid area (lines 6–14) composed of three nodes, which is an elliptical area in a 2D environment; in a 3D environment, this area is an ellipsoid area. First, randomly and uniformly sample in the unit circle (line 13), and then map the random sampling point q_{ball} to the point $q_{ellipse}$ (line 14) in the specified ellipse through matrix transformation, where P_i and P_j are the two focal points of the ellipse and P_{i+1} is a point on the ellipse, C_{min} is the focal length, and C_{max} is the length of the major axis. The transformation matrix L is a diagonal matrix, which can be obtained by the Cholesky decomposition of the hyperellipsoid matrix S, namely $LL^T \equiv S$, where $S \in {}^{n\times n}$, and the constraint condition is

$$S = diag\left\{\frac{C_{best}^2}{4}, \frac{C_{best}^2 - C_{min}^2}{4}, \ldots, \frac{C_{best}^2 - C_{min}^2}{4}\right\}. \tag{9}$$

From S, the decomposition matrix L can be obtained as

$$L = diag\left\{\frac{C_{best}}{2}, \frac{\sqrt{C_{best}^2 - C_{min}^2}}{2}, \ldots, \frac{\sqrt{C_{best}^2 - C_{min}^2}}{2}\right\}. \tag{10}$$

Obtain the rotation transformation matrix C from the hyperellipsoid coordinate system to the world coordinate system through the function $RatationToWorldFrame$, which is defined as

$$C = U \cdot diag\{1, 1, \ldots, \det(U) \cdot \det(V)\} \cdot V^T, \tag{11}$$

where det(·) represents the determinant and the constraint matrix $\sum V^T \equiv \frac{P_j - P_i}{(P_j - P_i)_2} \cdot e_1^T$ is subjected to singular value decomposition to obtain $U \in {}^{n\times n}$, $V \in {}^{n\times n}$, where e_1 is the product outside the first column of the identity matrix. Finally, by rotating and translating q_{ball} through the matrix, the random point $q_{ellipse}$ in the corresponding ellipse can be obtained (line 14). When

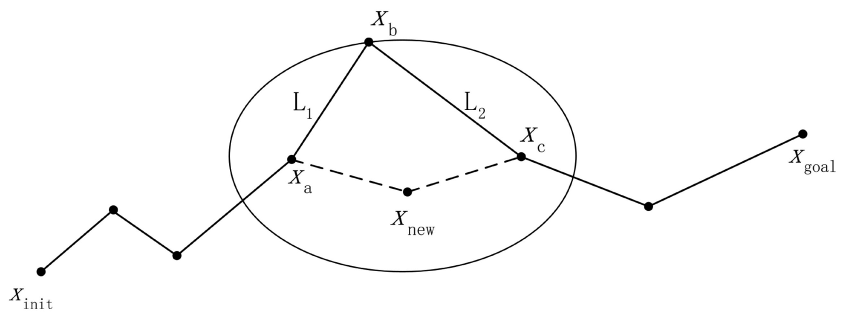

Figure 5: Schematic diagram of path optimization in 2D environment.

$q_{ellipse}$ does not collide with the two focal points, $q_{ellipse}$ is used to replace the node q_{i+1} in the original path.

Figure 5 is a schematic diagram of path optimization in 2D environment. The initial path is the black solid line, the inflection point is the node in the path, L_1 is the path length between x_a and x_b, and L_2 is the path length between x_c and x_b. During the optimization process, the algorithm will sequentially select three nodes in the path to form an ellipse for path optimization. Take Fig. 5 as an example, take x_a and x_c in the path as the focal points of the ellipse, and take $L_1 + L_2$ as the long axis of the ellipse to form a unique ellipse, and use this area as the sampling area of the optimized path. According to the definition of an ellipse, when the new node is inside the ellipse, the sum of the distance between itself and the two focal points must be less than the major axis, so the newly generated path must be better than the original path. When there is no collision, the connection between x_b, x_a, and x_c is disconnected, and x_{new} is connected to x_a and x_c to generate a better new path.

4.2.2. *Greedy pruning algorithm*

The greedy pruning algorithm is shown in Algorithm 9, P is the initial path, N is the number of nodes in the path P, and P_{op} is the optimized path. The algorithm starts from the starting point and performs collision detection with the nodes behind the path in the order of nodes in the path. If there is a collision, the detection continues from the parent node of the collision node until the two nodes pass the collision detection. If two nodes pass the collision detection and there are other nodes between the two nodes, at this time, the two nodes are directly connected, and the middle node is discarded. Greedy pruning algorithm can reduce unnecessary nodes and reduce path cost.

Algorithm 9. Pseudo-code of greedy pruning algorithm

```
P ← $GlobalPruning(P)$
1.  N ← |P|
2.  i = 1
3.  $P_{op}$ ← P(1)
4.  while i ≠ N
5.    j = N
6.    while i ≠ j
7.      if CollisionFree(P(i), P(j))
8.        $P_{op}$ ← Reconnection(P(i), P(j))
9.        i = j
10.       break
11.   j = j − 1
12. return $P_{op}$
```

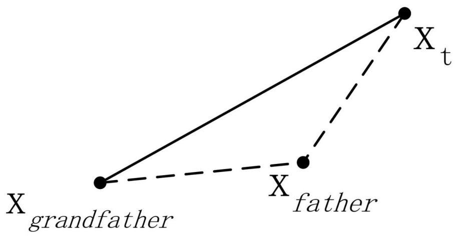

Figure 6: Schematic diagram of greedy pruning algorithm.

Figure 6 is a schematic diagram of the greedy pruning algorithm. The dotted line in the figure is the clipped path. When there is no collision between x_t and $x_{grandfather}$, x_{father} will be discarded and $x_{grandfather}$ will be used as the parent node of x_t. According to the principle of triangle inequality, the path cost after pruning will be smaller.

4.2.3. *Path smoothing based on B-spline curve*

The final path generated by the path planning algorithm usually has many inflection points, which is not conducive to the movement of the robot and increases the complexity of controlling the robot. Splines are piecewise polynomial functions and are widely used to interpolate data points or approximate functions, curves, and surfaces. The B-spline curve overcomes the shortcomings of the Bezier curve, it can specify the order, and moving the control point only changes part of the shape of the curve. Therefore, B-spline curves are widely used in path smoothing. Elbanhawi *et al.* proposed a B-Spline based smoothing approach (Elbanhawi *et al.*, 2015), which proved to be more robust than aforementioned approaches due to its ability to maintain continuity and order. It also offers a point insertion-based strategy to avoid collision; however, it is only effective in a simple structured environment and fails in un-structured, complex, or narrow passage scenarios. It also lacks a post collision curve improvement mechanism and generates longer paths. Two categories of curve generation schemes are prominent in a path smoothing state of art, i.e. interpolation and approximation. Interpolation-based schemes pass from all the way points of path, generating a longer curve. Approximation-based schemes interpolate end points and approximate middle points, generating a shorter curve than interpolation. Iram-proposed approach has introduced an economical point insertion scheme with automated knot vector generation while eliminating post smoothness collisions with obstacles (Noreen, 2020). However, this method is prone to useless collision detection in the process of smoothing the path. Qian used the cubic spline for the path planning of fork-car AGV (Donghai *et al.*, 2022); Gu proposed

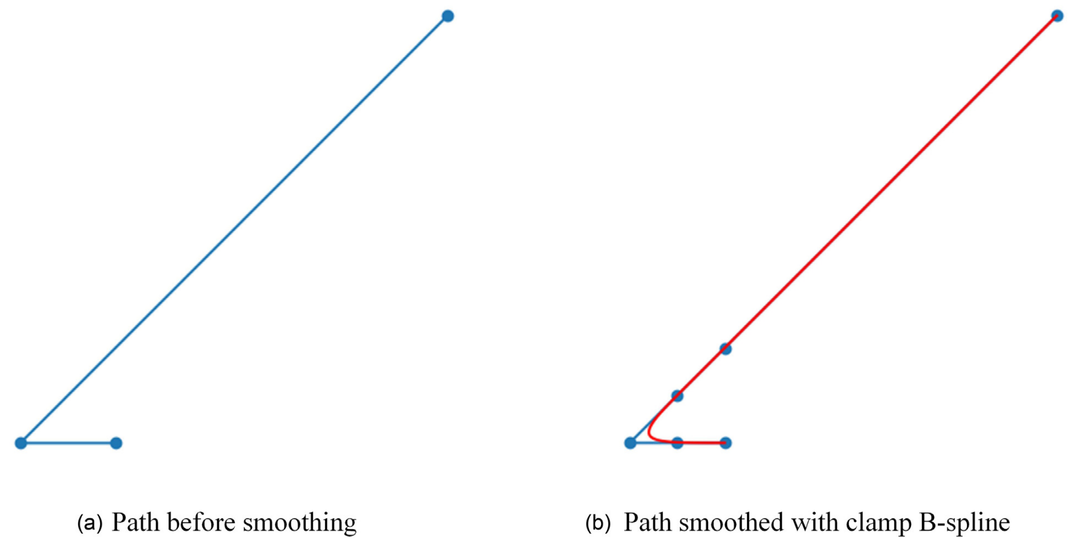
(a) Path before smoothing
(b) Path smoothed with clamp B-spline

Figure 7: Schematic diagram of path smoothing.

an improved cubic B-spline algorithm and used it to smooth the trajectory of shrub pruning (Gu *et al.*, 2021).

Let U be a set of m nondecreasing trees, $u_0 \le u_1 \le u_2 \le \ldots \le u_{m-1}$, where u_i is called a node, set U is a node vector, and half-open interval $[u_i,\ u_{i+1})$ is the i_{th} node interval. p represents the degree of the basis function; then, the basis function of the B-spline curve is defined as

$$N_{i,0}(u) = \begin{cases} 1, & u_i \le u \le u_{i+1} \\ 0, & \text{else} \end{cases} \tag{12}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u), \tag{13}$$

where $N_{i,\ p}(u)$ represents the i_{th} p_{th} B-spline basis function. Suppose $p_0,\ p_1,\ p_2,\ \ldots,\ p_n$ are $n+1$ control points; then, the p-order ($p-1$ degree) B-spline curve is expressed as

$$C(u) = \sum_{i=0}^{n} N_{i,p}(u)\, P_i. \tag{14}$$

In this paper, the interpolation-based clamping cubic B-spline is used to smooth the path. The method of this smoothing algorithm is described in Algorithm 10. The smooth path algorithm first linearly interpolates near the path nodes to increase the number of control points. The path interpolation function receives the nodes in the path and the distance between the two nodes. The function selects the number of insertion points and the separation distance by judging the distance between the two nodes. When the distance between nodes is large, the newly added control point is inserted into the position closer to the inflection point through linear interpolation. When the distance between the two nodes is small, the midpoint of the two nodes is inserted into the path as a new control node. As shown in Fig. 7, smoothing the path after increasing the number of control points near the inflection point in the path can better preserve the shape of the original path, which greatly reduces the probability of collision between the smoothed path and the obstacle.

Before the path is flattened, the node vector $\hat{U}$ is generated using the properties defined by equation (15), where k is the order and n is the number of nodes in the path.

$$m = n + k + 1 \tag{15}$$

The curve is defined using the Cox–de Boor algorithm. There are n basis functions for n number of control points with strong local control at that point. These basis functions are calculated using a recursive algorithm as shown in formulas (12) and (13). Once the initial smooth path SP is generated, it will be evaluated that it may collide with surrounding obstacles due to the introduction of smoothing; see step 9 in Algorithm 10. If a collision occurs in the new path, the next step is to identify the path segment facing the collision, as shown in step 10.

Once the collision segment is determined, an iterative process will insert new control points between the control points of the corresponding segment to approach the planned path; see steps 11–14 in Algorithm 10. As a new control node is inserted, the node vector will be updated accordingly; see step 15. After inserting new control points in the collision segment, regenerate the smooth path SP; see step 16. Collision assessment and control points update for a smooth path until the entire path is collision free.

Algorithm 10. Pseudo-code of path smoothing algorithm

```
P ← B_Spline(P)
1.  N ← |P|
2.  i = 0
3.  while i ≠ N
4.     lenth = getlenth(P(i), P(i + 1))
5.     P ← Interpolation(P(i), P(i + 1), lenth)
6.     Update(i, N)
7.     Û ← CreatKnotVector(N)
8.     SP ← BSpline(Û, P, k)
9.  while CollisionExit(SP)
10.    collision_SP ← Getsegments(SP)
11.    P(i), P(j) ← Getpathnode(collision_SP)
12.    lenth = getlenth(P(i), P(j))
13.    P ← Interpolation(P(i), P(j), lenth)
14.    N ← |P|
15.    Û ← CreatKnotVector(N)
16.    SP ← BSpline(Û, P, k)
17. return SP
```

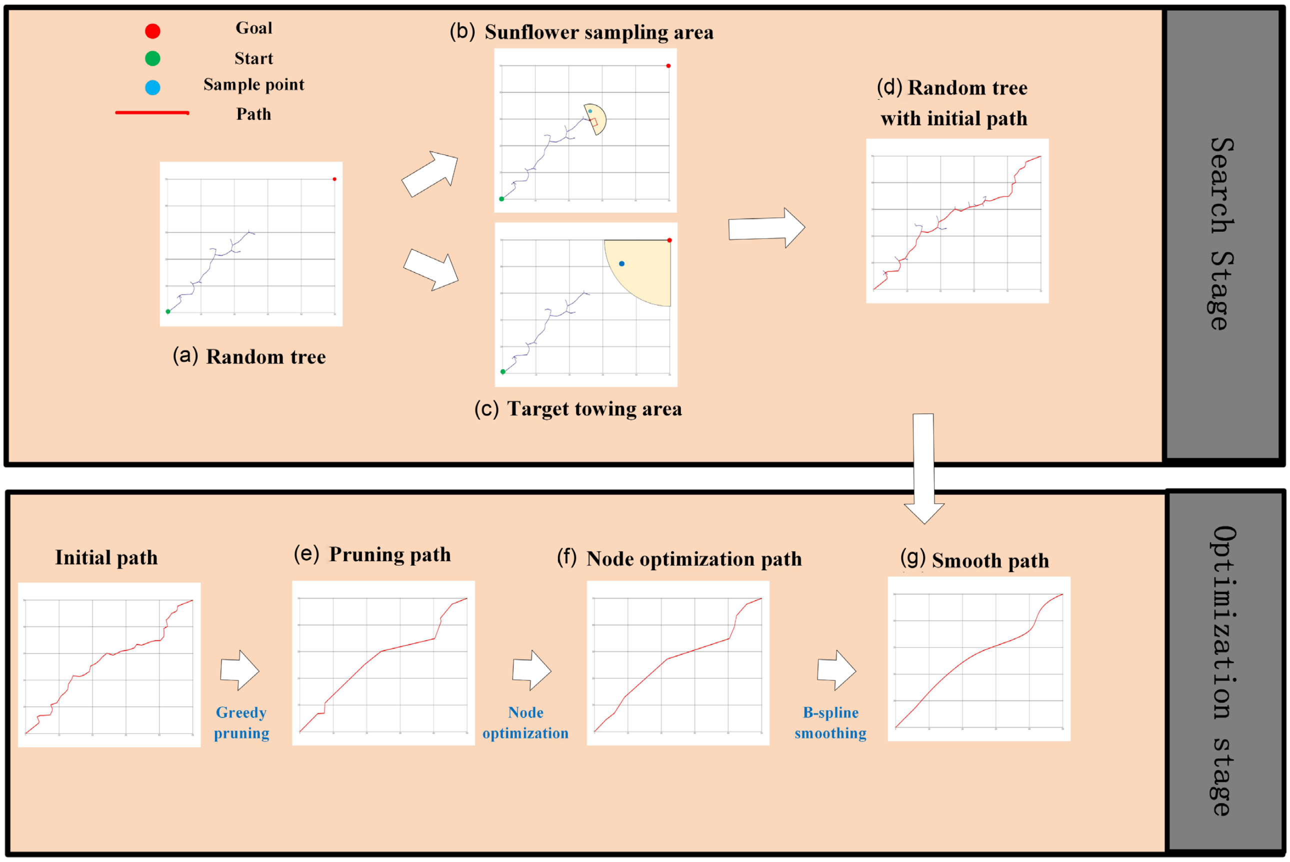

Figure 8: Process of proposed algorithm TBIT*.

Figure 8 shows the top-level sequence of major steps in proposed approach. Figure 8a shows the random tree generated during the search phase. In the search stage, the sampling method is dynamically selected according to the expansion failure rate R_{ef} during the random tree expansion process. When $R_{ef} \le r_1$, sampling points are generated in the sunflower area, as shown in Fig. 8b, where the semicircular area is the sunflower sampling area. When $r_1 < R_{ef} \le r_2$, sample points are generated in the target traction area, as shown in Fig. 8c, where the shaded part is the target traction area. When $r_2 < R_{ef}$, the sampling points are obtained through random sampling. Figure 8d shows the initial path obtained during the search phase. The optimization phase optimizes the initial path. Figure 8e shows the optimization process of the greedy pruning algorithm. At this stage, redundant nodes in the path can be pruned to reduce the path cost. Figure 8f is the result of using heuristic optimization strategy to optimize the path. In this stage, the nodes in the path are optimized by using the elliptic nature to further reduce the path cost. Finally, the clamped cubic B-spline curve is used to smooth the path, so that the final path is more in line with the laws of robot kinematics, as shown in Fig. 8g.

5. Simulation Results and Experimental Analysis

The TBIT* algorithm proposed in this paper uses a combined target bias strategy for sampling in the search stage. The combined target bias strategy includes sunflower area sampling and target pulling area sampling. In the path optimization stage, the greedy pruning algorithm is used to delete useless nodes in the path; heuristic sampling is used to obtain potential optimized sampling points to reduce the path cost, and the path is smoothed by the cubic B-spline curve. Some scholars have done similar work, and the RRT*-AB algorithm also uses the target bias strategy. However, the sampling limited area of this algorithm will only change when the initial path is not found at the end of scanning the area or when the initial path is found in the area. In the optimization path stage, RRT*-AB redefines the connection area along the initial path, and optimizes the selection of sampling points through the node rejection mechanism to optimize the path. Therefore, the two algorithms are essentially different. The RRT*-AB algorithm has a good application effect in the structured complex 2D environment and the unstructured indoor environment. This section will also verify the applicability of the algorithm in different environments.

This section demonstrates the experiments of the algorithm proposed in this article in different environments, and each experimental scene has different obstacles. The experimental data of RRT, RRT*, Informed RRT*, FMT*, BIT*, and TBIT* are also presented. These algorithms are implemented using the 64-bit version of PyCharm Community 2020 and run on a PC with Intel Core i5-8250U CPU@1.60GHz CPU and 8GB internal RAM. Experiments were carried out using different environment maps M1, M2, M3, M4, and M5 to verify the robustness of the algorithm. During the experiment, the experimental data and simulation results were recorded, and they were compared and analysed. Considering that the algorithm has a certain degree of

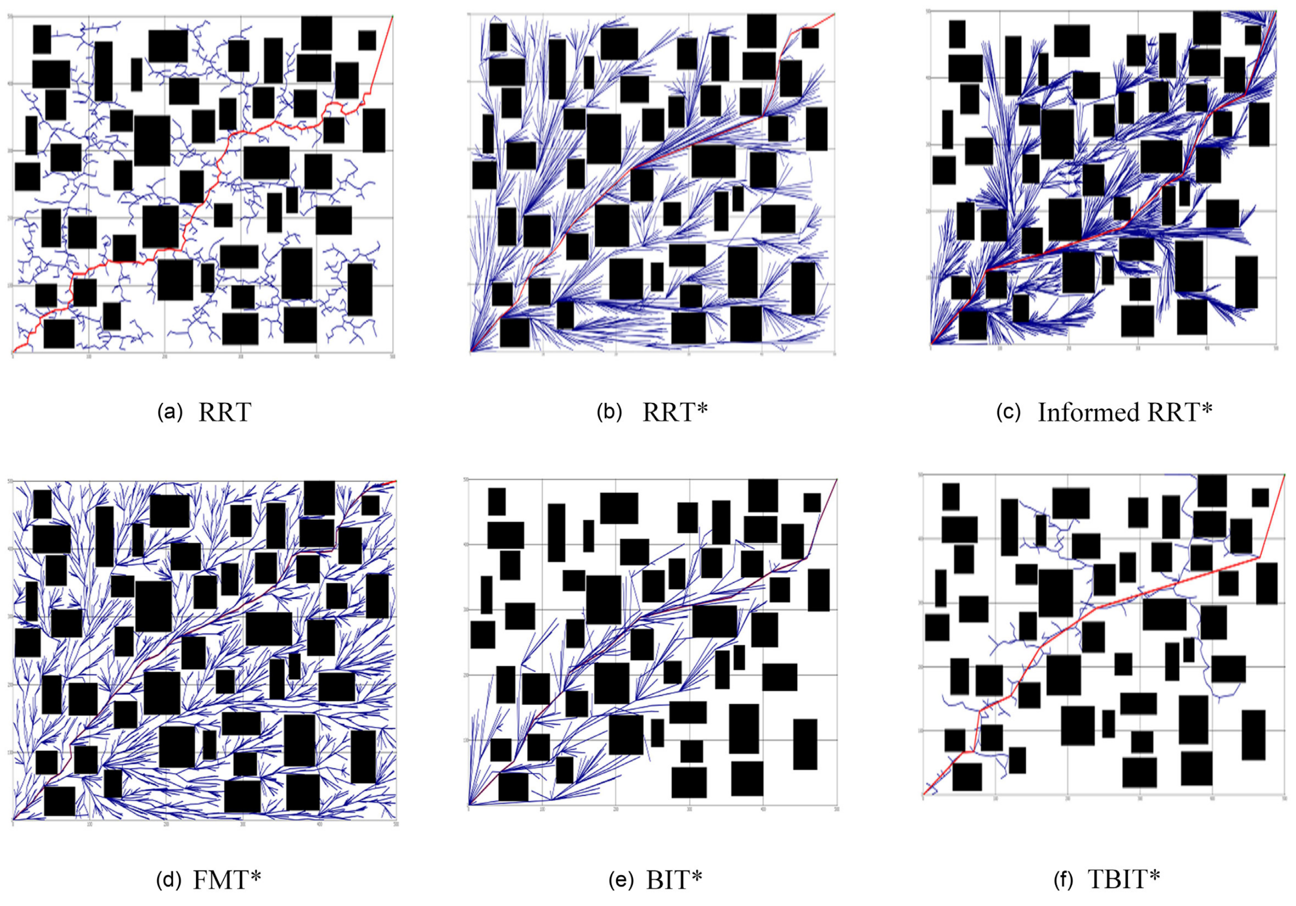

Figure 9: Results comparison of map M1.

Table 1: Simulation results of six algorithms in M1.

Algorithms	Time to generate the initial path (s)	Final path cost	Total number of random tree nodes
RRT	1.71	940	1133
RRT*	18.38	748	3880
Informed RRT*	20.49	765	3518
FMT*	22.08	762	2998
BIT*	1.75	772	310
TBIT*	0.99	744	63

randomness, each experiment set has carried out 30 iterations on the map, and the simulation data presented are the average of these 30 experiment results.

In order to compare the algorithm optimization effects, the numbers of optimization sampling iterations after finding the initial path for RRT*, Informed RRT*, BIT*, and TBIT* are all 3000 times. In the iterative process of BIT*, the number of samples in each batch is taken as 200. In view of the implementation principle of the FMT* algorithm, it is not easy to distinguish the optimization stage of the path, so the sampling point of the FMT* algorithm is set to 3000 in this article.

5.1. Structured complex 2D environment

Maps M1 and M2 are complex scenes with different obstacle densities in a 2D environment. The density of obstacles in M1 is higher. It can be seen from Fig. 9 and Table 1 that the RRT, RRT*, Informed RRT*, and FMT* algorithms sample in the entire space. However, the RRT* algorithm and the Informed RRT* algorithm are added to the optimization strategy of reselecting the parent node and rerouting, which makes the path optimization effect more obvious. However, due to the increase in the amount of calculation, the time for the RRT* algorithm and the Informed RRT* algorithm to obtain the initial path is greatly increased. After the Informed RRT* and BIT* algorithms obtain the initial path, the sampling is limited to the ellipse, so the random tree is concentrated near the optimal path. However, BIT* uses a series of informed graph searches to process states in order of potential solution quality, which speeds up its construction of random trees.

The combined target bias strategy of the TBIT* algorithm makes the random tree grow toward the target point. Therefore, the generated nodes are concentrated near the optimal path, and the success rate of node expansion will be greatly

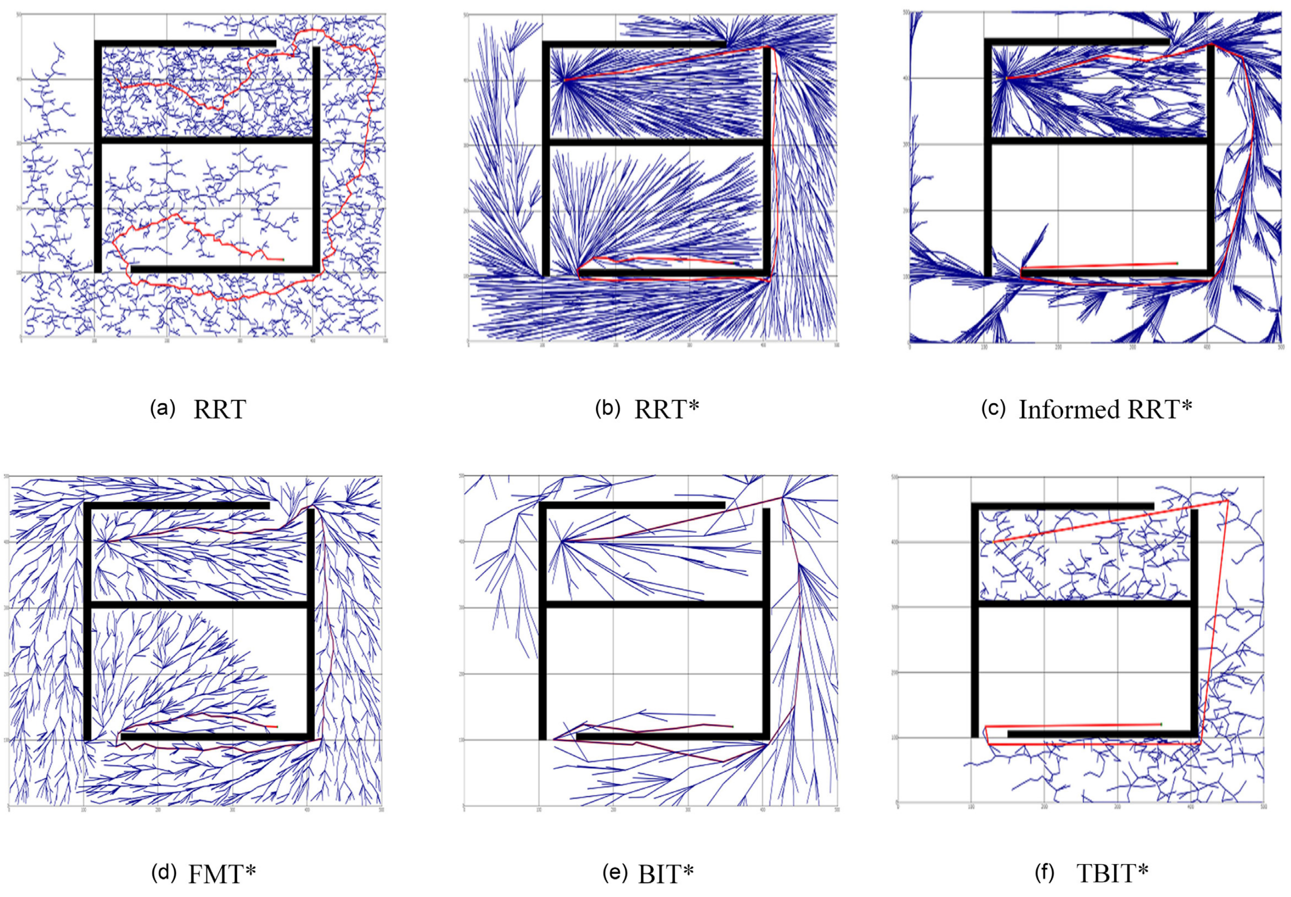

Figure 10: Result comparison of map M2.

Table 2: Simulation results of the six algorithms in M2.

Algorithms	Time to generate the initial path (s)	Final path cost	Total number of random tree nodes
RRT	4.6	1640	3354
RRT*	81.0	1154	6151
Informed RRT*	42.26	1184	4770
FMT*	16.21	1176	2826
BIT*	3.74	1218	654
TBIT*	3.38	1131	871

increased. Therefore, the TBIT* algorithm not only obtains a path that is not much different from the RRT*, Informed RRT*, FMT*, and BIT* algorithms, but also greatly reduces the number of nodes in the random tree. Since the reselection of the parent node and the rewiring strategy are omitted, the amount of calculation at this time is greatly reduced, so the time cost of obtaining the initial path is also greatly reduced. The above experimental results show that the TBIT* algorithm is more efficient than the other five algorithms in a complex and dense environment.

Figure 10 is a graph of the results of different algorithms running in M2. M2 is a special complex 2D simulation environment with U-shaped obstacles. All six algorithms can find the optimal solution. Except for RRT, the final path cost obtained by convergence is basically the same. In terms of time cost, because the TBIT* algorithm adds a combined target bias strategy, the initial path can be obtained faster. Compared with the RRT algorithm, the time cost of the TBIT* algorithm is reduced by 43%. Compared with the BIT* algorithm, while the time cost is reduced by 10%, the final path cost obtained by TBIT* is also reduced accordingly.

When the TBIT* algorithm uses sunflower area sampling, the expansion failure rate increases due to the increase of expansion failure nodes, so the algorithm uses target traction area sampling and random sampling to quickly escape the obstacle area. The expansion success rate outside the U-shaped obstacle is relatively large, which makes the algorithm bias toward combined target sampling, and the random tree grows quickly to the target point. Therefore, the number of nodes in the random tree of the TBIT* algorithm is greatly reduced compared to the RRT, RRT*, Informed RRT*, and FMT* algorithms. The analysis results are shown in Table 2.

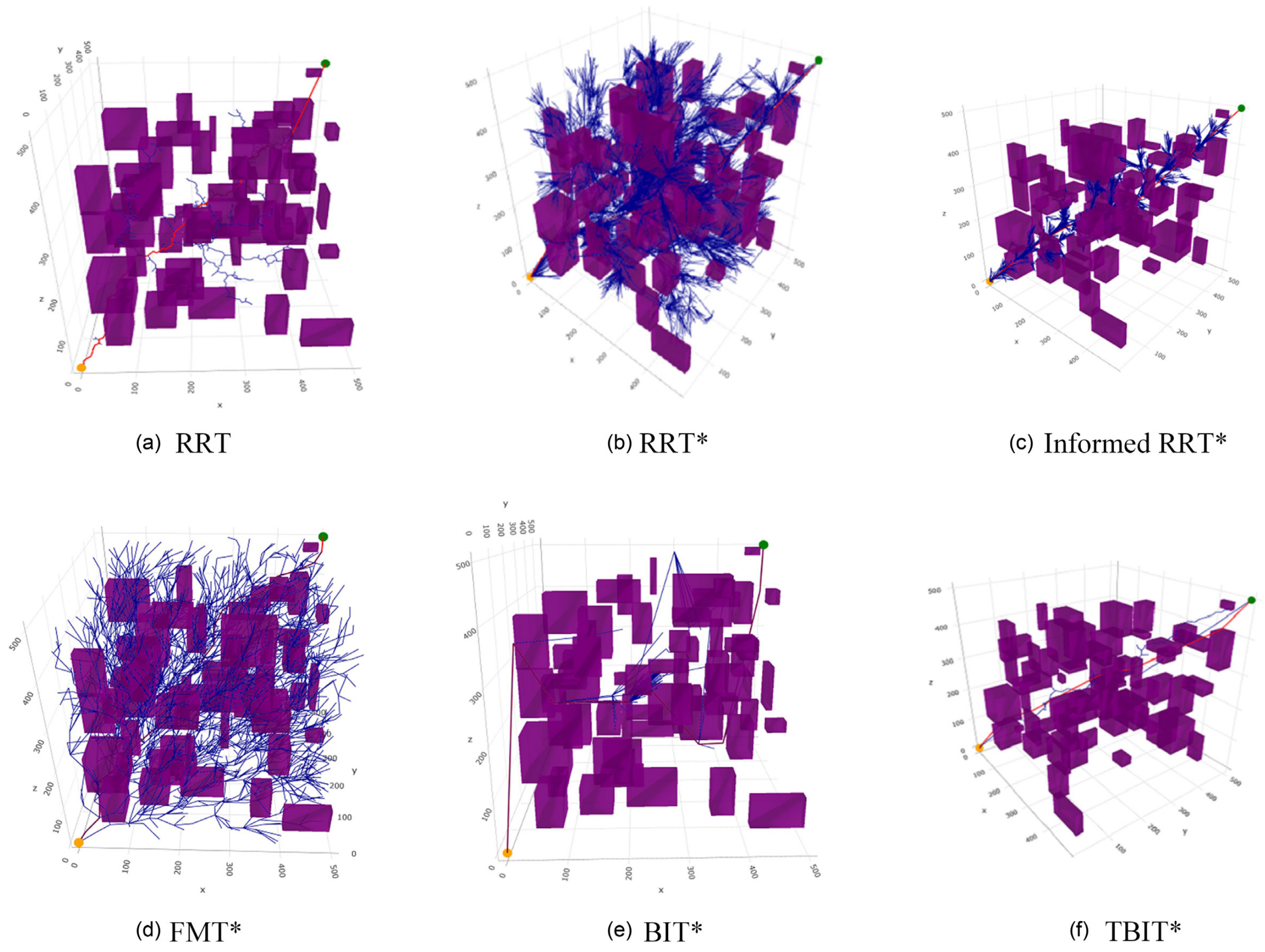

Figure 11: Result comparison of map M3.

Table 3: Simulation results of the six algorithms in M3.

Algorithms	Time to generate the initial path (s)	Final path cost	Total number of random tree nodes
RRT	2.04	1119	983
RRT*	10.47	910	3674
Informed RRT*	8.99	921	3076
FMT*	26.27	990	2998
BIT*	1.06	868	230
TBIT*	0.81	707	68

5.2. Structured complex three-dimensional environment

Figure 11 is a graph of the results of different algorithms running in M3. M3 is a complex 3D simulation environment with dense obstacles. The six algorithms can get the optimal path in M3, and the fast convergence of BIT* and TBIT* algorithms is the best. Due to the addition of an optimization strategy, the sampling points generated by the Informed RRT* algorithm in the optimization stage are also near the optimal path. The RRT, RRT*, and FMT* algorithms are all exploring the entire space. Although the optimization process makes the path cost of the RRT* algorithm gradually decrease, as the number of nodes increases, the calculation amount of the RRT* algorithm increases, so the convergence speed decreases.

The combined target biased sampling method significantly increases the speed of the TBIT* algorithm to obtain the initial solution. Due to the high efficiency of sampling, the number of nodes in the random tree is greatly reduced. Compared with the RRT and BIT* algorithms, the final path cost obtained by the TBIT* algorithm is reduced by 37% and 20%. This demonstrates the ability of the TBIT* algorithm to quickly pass through complex 3D spaces. The analysis results are shown in Table 3.

Figure 12 is a graph of the results of different algorithms running in M4. M4 is a 3D simulation environment with three narrow passages. The six algorithms have good passability in the 3D narrow channel space. Compared with the RRT and RRT* algorithms, the Informed RRT* algorithm has a better optimization effect on random trees. However, the number of random tree nodes is large, and the useless calculations in-

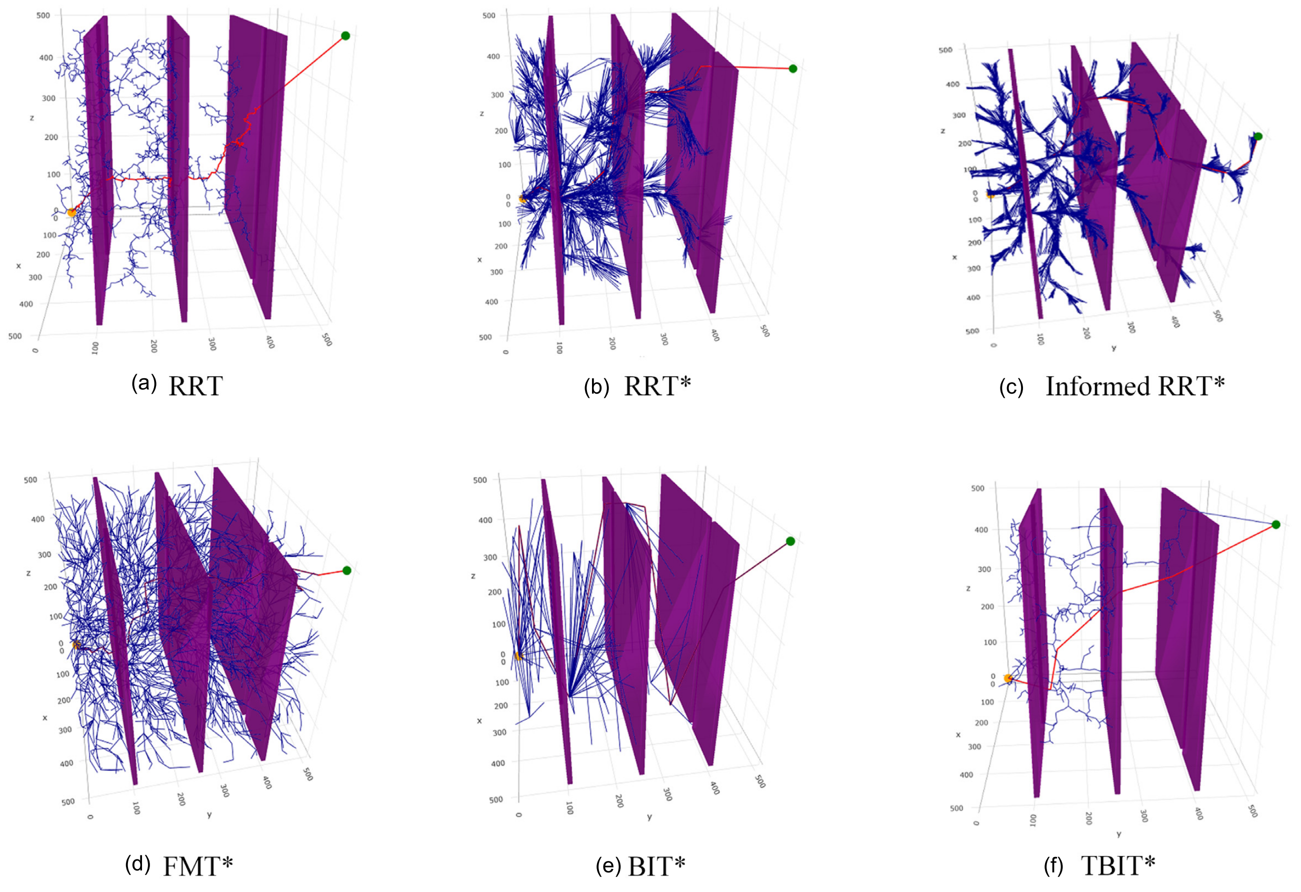

Figure 12: Result comparison of map M4.

Table 4: Simulation results of the six algorithms in M4.

Algorithms	Time to generate the initial path (s)	Final path cost	Total number of random tree nodes
RRT	4.04	1528	2649
RRT*	28.39	1132	4263
Informed RRT*	61.39	1169	6184
FMT*	43.33	1227	2836
BIT*	12.32	885	154
TBIT*	3.75	735	369

crease, resulting in a decrease in the convergence speed of the algorithm. The time cost of the TBIT* algorithm when passing through narrow passages in 3D space is significantly reduced, which also speeds up the convergence speed of the algorithm.

The TBIT* algorithm has a good optimization effect on the path. Except for TBIT*, BIT* has the lowest path cost, and the final path obtained by RRT*, Informed RRT, and FMT* algorithms is not much different. Compared with the BIT* algorithm, the TBIT* path cost is reduced by 27%. The TBIT* algorithm has a great improvement in this convergence efficiency, and the node utilization rate in its random tree is relatively high. The above illustrates the obvious efficiency improvement compared to the other five algorithms in a 3D environment containing narrow passages. The analysis results are shown in Table 4.

5.3. Unstructured indoor environment

Experiments on unstructured indoor environment are also conducted. For this purpose, map M5 is adapted from Intel research lab datasets. We extracted the map data, filtered it, and finally converted it into an occupancy map for simulation experiments. Figure 13 shows the map in the source dataset (Fig. 13a) and the processed occupancy map (Fig. 13b).

It can be seen from Fig. 14 that all six algorithms can obtain paths in an unstructured environment, but the optimization effects of BIT* and TBIT* are relatively obvious, and their sampling points in space are more efficient, thus reducing the amount of calculation. Therefore, these two algorithms can obtain the initial path in a relatively short time.

It can be seen from Table 5 that in the optimization stage of the initial path, the RRT* and Informed RRT* algorithms have

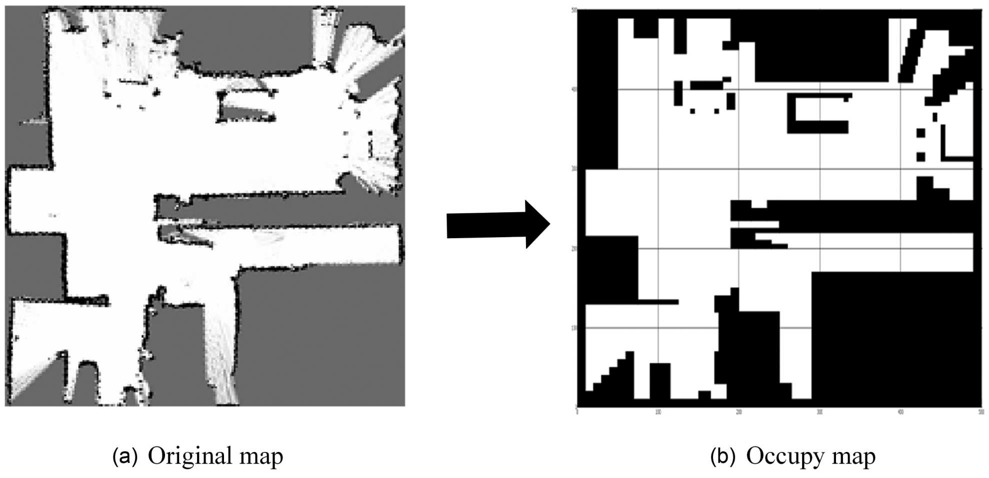

Figure 13: Original map and occupancy map.

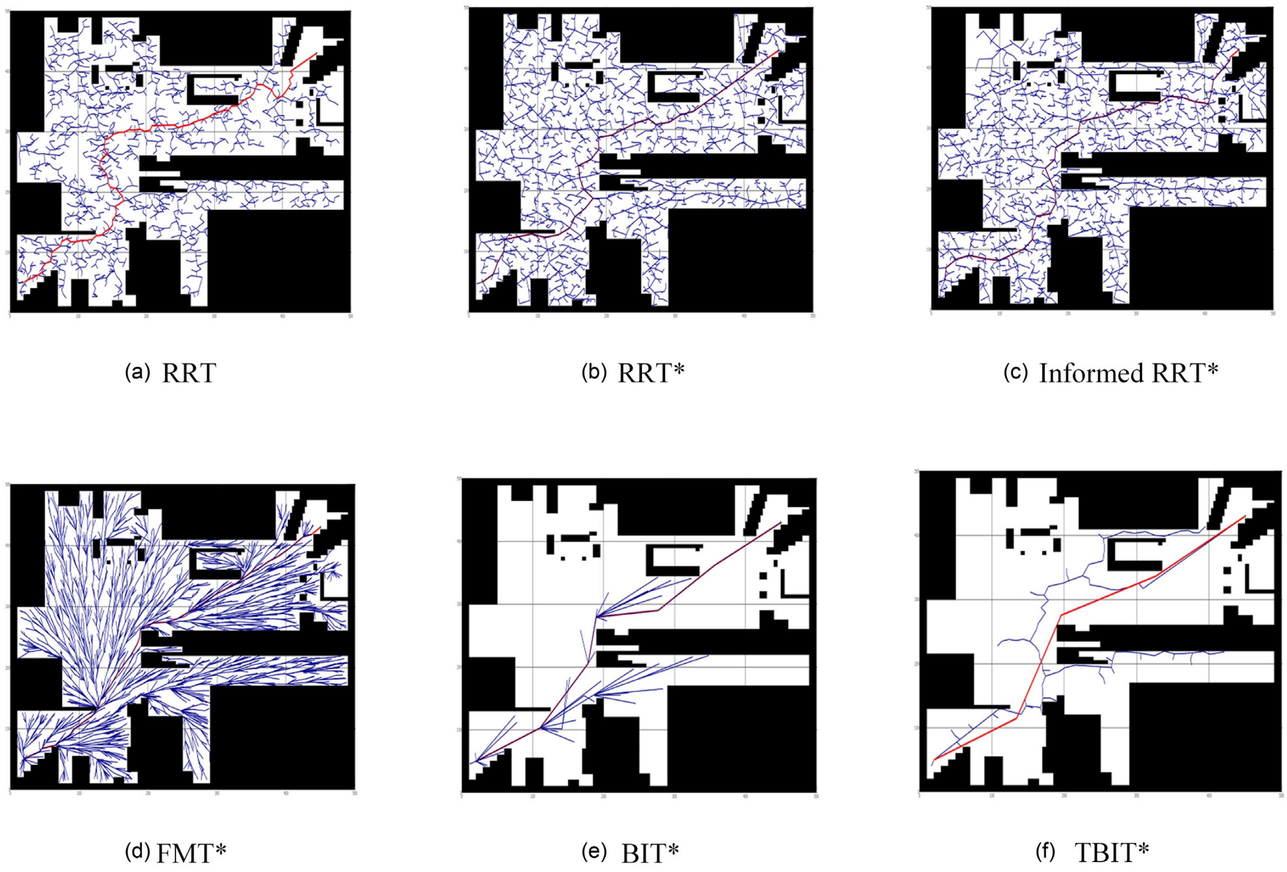

Figure 14: Result comparison of unstructured map M5.

similar effects. BIT* and TBIT* obtained similar final paths, but the time for the TBIT* algorithm to obtain the initial path increased by 40% compared to BIT*. Although the final path obtained by FMT* is similar to BIT*, it takes a longer time for FMT* to find a path due to the way it obtains the random tree.

Since the process of convergence to the optimal solution is started after the initial feasible solution is found, the convergence rate is calculated after the initial path is calculated. Since RRT and FMT* cannot strictly distinguish the optimization stages of the path, we compared the convergence rates of RRT*, Informed RRT*, BIT*, and TBIT*. Figure 15 shows the comparison of the convergence rate of these four algorithms in M5 for 3000 iterations. Let the initial feasible path, denoted by π_{init}, be computed in t_{init} time with C_{init} path cost while the optimal path solution, denoted as π_*, is computed in t_* time with C_* path cost. The convergence rate is defined as

$$R_{\text{convergence}} = \frac{C_{\text{init}} - C_*}{t_* - t_{\text{init}}}. \tag{16}$$

Table 5: Simulation results of the six algorithms in M5.

Algorithms	Time to generate the initial path (s)	Final path cost	Total number of random tree nodes
RRT	0.96	751	353
RRT*	2.89	758	2833
Informed RRT*	3.38	705	2833
FMT*	28.7	597	2910
BIT*	1.54	593	191
TBIT*	0.91	580	90

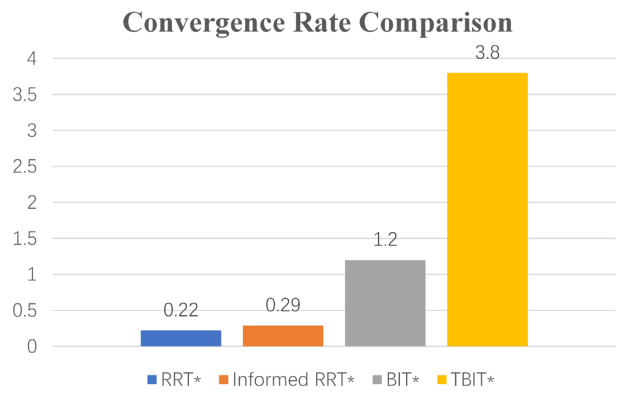

Figure 15: Convergence rate comparison.

It can be clearly seen from Fig. 15 that the proposed TBIT* method has the highest convergence rate, followed by BIT*. The convergence rates of RRT* and Informed RRT* are similar. The results show that once the initial path is found, this method can be improved more quickly than other methods to make it converge to the optimal solution.

6. Conclusion

The improved TBIT* path optimization algorithm proposed in this paper optimizes the RRT* algorithm in the search phase and the optimization phase. In the initial stage, the growth of the tree is guided by the combined target bias strategy to reduce the generation of useless nodes, thereby reducing the time cost of the algorithm and speeding up the convergence of the algorithm. In the optimization stage, directly optimizing the nodes in the path can reduce the number of nodes in the random tree and avoid a large number of invalid calculations, thereby reducing the time cost of obtaining the optimal path. The simulation results in the complex 2D and 3D simulated environments show that, compared with the basic RRT, RRT*, Informed RRT*, FMT*, and BIT* algorithms, the TBIT* algorithm proposed in this paper can form the initial path faster, and has a lower number of nodes in the random tree and a smaller the path cost. This shows that TBIT* has better versatility in both 2D and 3D environments. In a complex and special 2D environment, the superiority of the algorithm shows that it has certain application value in robots that work in 2D environments such as sweeping robots; in a complex and special 3D environment, the superiority of the algorithm shows that it can be applied to robots that work in 3D scenes, such as manipulators.

The algorithm in this paper only considers the optimization of the path in terms of path cost, number of nodes, and time cost, and does not consider the handover constraints in the path; it leads to excessive turning angles in certain positions in the path, which is not conducive to the movement of the robot. Subsequent handovers will be integrated into the generated path constraints in our future works.

Conflict of interest statement

None declared.

References

Baba, N., & Kubota, N. (1993). Path planning and collision avoidance of a robot manipulator using genetic algorithm. *Journal of the Robotics Society of Japan*, 11(2), 299–302.

Borenstein, J., & Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3), 278–288.

Donghai, Q., Linlin, S., & Wei, Z. (2022) Research on fork vehicle AGV path planning based on cubic B-spline curve. *Computer Measurement and Control*, 1–7. doi: 10.1360/112012-392.

Elbanhawi, M., Simic, M., & Jazar, R. (2015) Continuous path smoothing for car-like robots using B-Spline curves. *Journal of Intelligent and Robotic Systems*, 80, 23–56.

Gammell, J. D., Barfoot, T. D., & Srinivasa, S. S. (2020). Batch informed trees (BIT*): Informed asymptotically optimal anytime search. *The International Journal of Robotics Research*. 39(5), 543–567.

Gu, J., Wu, T., Li, C., Zhang, B., & Zhang, Y. (2021). Study on smoothing algorithm of shrub pruning trajectory based on improved cubic B-spline. *Transactions of the Chinese Society of Agricultural Machinery*, 52(S1), 89–97.

Janson, L., Schmerling, E., Clark, A., & Pavone, M. (2015). Fast marching tree: A fast marching sampling-based method for optimal motion planning in many dimensions. *The International Journal of Robotics Research*, 34(7), 883–921.

Jordan, M., & Perez, A. (2013). *Optimal bidirectional rapidly-exploring random trees, Series/report no. MIT-CSAIL-TR-2013-021*.

Karaman, S., & Frazzoli, E. (2011). Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7), 846–894.

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

LaValle, S. M. (1998). *Rapidly-exploring random trees: A new tool for path planning (Technical report)*.

Lei, B., & Li, W. (2007). A fuzzy behaviours fusion algorithm for mobile robot real-time path planning in unknown environment. In *2007 IEEE International Conference on Integration Technology*(pp. 173–178). IEEE.

Li, X., Ma, Z., Chu, X., & Liu, Y. (2019). A cloud-assisted region monitoring strategy of mobile robot in smart greenhouse. *Mobile Information Systems*, 2019, 5846232.

Liu, L., Wang, Y. N., & Kuang, F. (2007). Path planning of mobile robot based on neural network and genetic algorithm. *Jisuanji*

Yingyong Yanjiu/Application Research of Computers, *24*(2), 264–265.

Lumelsky, V., & Stepanov, A. (1986). Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE Transactions on Automatic Control*, *31*(11), 1058–1063.

Mashayekhi, R., Idris, M. Y. I., Anisi, M. H., Ahmedy, I., & Ihsan, A. (2020). Informed RRT*-connect: An asymptotically optimal single-query path planning method. *IEEE Access*, *8*, 19842–19852.

Nasir, J., Islam, F., & Ayaz, Y. (2013). Adaptive rapidly-exploring-random-tree-star (RRT*)-smart: Algorithm characteristics and behavior analysis in complex environments. *Asia-Pacific Journal of Information Technology and Multimedia*, 2, 39.

Noreen, I. (2020). Collision free smooth path for mobile robots in cluttered environment using an economical clamped cubic B-Spline. *Symmetry*, *12*(9), 1567.

Noreen, I., Khan, A., Ryu, H., Doh, N. L., & Habib, Z. (2011). Optimal path planning in cluttered environment using RRT*-AB. *Intelligent Service Robotics*, *11*(1), 41–52.

Qureshi, A. H., & Ayaz, Y. (2016). Potential functions based sampling heuristic for optimal path planning. *Autonomous Robots*, *40*(6), 1079–1093.

Su, H., Hu, Y., Karimi, H. R., Knoll, A., Ferrigno, G., & De Momi, E. (2020). Improved recurrent neural network-based manipulator control with remote center of motion constraints: Experimental results. *Neural Networks: The Official Journal of the International Neural Network Society*, *131*, 291–299.

Su, H., Mariani, H., Ovur, S. E., Menciassi, E., Ferrigno, G., & De Momi, E. (2021). Toward teaching by demonstration for robot-assisted minimally invasive surgery. *IEEE Transactions on Automation Science and Engineering*, *18*(2), 484–494.

Su, H., Qi, W., Hu, Y., Karimi, H. R., Ferrigno, G., & De Momi, E. (2022). An incremental learning framework for human-like redundancy optimization of anthropomorphic manipulators. *IEEE Transactions on Industrial Informatics*, *18*(3), 1864–1872.

Willms, A. R., & Yang, S. X. (2006). An efficient dynamic system for real-time robot-path planning. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *36*(4), 755–766.

Zhang, B., Chen, W., & Fei, M. (2006). An optimized method for path planning based on artificial potential field. In *Sixth International Conference on Intelligent Systems Design and Applications*(Vol. 3, pp. 35–39). IEEE.