International Core Journal of Engineering Volume 12 Issue 3, 2026
ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

Mobile Robot Path Planning based on Improved RRT Algorithm

Yi Li[1], Chi Zhang[1], Wei Wei[2], Xiaobo Liu[1]

[1] School of Aeronautical Manufacturing and Mechanical Engineering, Nanchang Hangkong University, Nanchang 330063, China

[2] School of Mechanical Engineering&Automation, Beihang University, Beijing 100191, China

Abstract

To address the limitations of the Rapidly-exploring Random Tree(RRT) algorithm in mobile robot path planning, specifically issues such as a large sampling range, long search times, and insufficient path smoothness, an improved RRT algorithm is proposed. This algorithm introduces a dynamic sampling strategy to reduce random sampling redundancy and accelerate convergence. Furthermore, redundant waypoints are eliminated to enhance path quality while reducing memory consumption. Finally, Bézier curves are employed to optimize path smoothness, replacing traditional linear connections with continuous curves for smoother turns. Simulation experiments demonstrate that the improved RRT algorithm achieves average performance and stability enhancements of 48% and 50%, respectively, with a 16% reduction in path length and superior path quality.

Keywords

Path Planning; RRT; RRT*; Path Optimization; Random Sampling; Path Smoothing.

1. Introduction

Mobile robots, known for their high flexibility and autonomy, can operate in hazardous environments in place of humans. Recent advances in artificial intelligence have accelerated their development. Path planning is crucial for a robot to complete tasks efficiently. As a core function of the navigation system, it determines an optimal or sub-optimal route to enable autonomous movement from a start to a goal point[1]. Ineffective path planning algorithms-those that are computationally slow or yield unnecessarily long paths-can significantly degrade a robot's operational efficiency and task performance[2].

Classical graph-based search algorithms, such as A*[3] and Dijkstra[4-5], perform well but become computationally expensive in high-dimensional spaces[6]. Consequently, researchers have developed various improvements. For instance, Wang et al.[7] integrated a convolutional neural network (CNN) with the A* algorithm to collect path data, predict optimal paths, and guide path growth. The artificial potential field method is simple to implement but prone to local minima. Ant colony optimization typically generates high-quality paths, but it is computationally intensive and slow to converge.

Sampling-based planners, such as the Rapidly-exploring Random Tree (RRT)[8] and Probabilistic Roadmap (PRM) algorithms, offer an alternative approach. By optimizing memory usage and planning efficiency, they effectively solve high-dimensional, large-scale problems and are widely used in motion planning. These algorithms are probabilistically complete, meaning that if a feasible path exists, the probability of finding it converges to one as the number of samples approaches infinity[9]. However, without heuristic guidance, they often produce sub-optimal paths, and their probabilistic nature can lead to topological discontinuities and a lack of path smoothness.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

To address the sub-optimality of the basic RRT algorithm, RRT* was introduced[10]. This variant incorporates a cost function and a rewiring process, transforming path planning into a minimum-cost optimization problem. Through iteration, RRT* asymptotically converges toward an optimal path, albeit at the cost of longer search times[11].

To overcome the limitations of RRT and RRT*, such as long convergence times and non-smooth paths, this paper proposes an improved RRT-based algorithm. Our approach first progressively reduces the sampling range as new nodes are generated, which effectively shortens convergence time and reduces sampling randomness. Second, we apply a smoothing process to eliminate redundant waypoints. Finally, Bézier curve fitting is used to generate a smooth, continuous trajectory. The effectiveness of our improved algorithm is validated through simulation experiments.

2. Basic RRT Algorithm

2.1 Problem Description

Definition $X \in \mathbb{R}^d$,where d denotes the spatial dimension， and $d \in \mathbb{N}, d \geq 2$. Let X_{obs} be the obstacle space and X_{free} be the free space, $X_{\text{free}} = X / X_{\text{obs}}$. Let x_{init} be the initial state and x_{goal} be the goal state and x_{init} , $x_{\text{goal}} \in X_{\text{free}}$. The goal region is defined as a circle with a radius of *r*: $X_{\text{goal}} = \left\{ x \in X \mid \left\| x\text{-}x_{\text{goal}} \right\| < r \right\}$. A path is defined as the continuous function $\sigma : [0,T] \to X_{\text{free}}$,and $\sigma(0) = x_{\text{init}}$, $\sigma(T) = x_{\text{goal}}$ .If $\sigma(\tau) \in X_{\text{free}}$, $\tau \in [0,T]$,hence, under this definition, the path σ is feasible.

Problem 1 (Feasible Path Planning): The feasibility problem in path planning is to find a feasible path. Given a state space where the free space is X_{free} , the start state is $x_{\text{init}} \in X_{\text{free}}$, and the goal region is $X_{\text{goal}} \in X_{\text{free}}$, the problem is to find a path $\sigma : [0,T] \to X_{\text{free}}$ such that condition $\sigma(0) = x_{\text{init}}$, $\sigma(T) \in X_{\text{goal}}$ is satisfied. If no such path exists, the algorithm should return failure. As shown in Figure 1, if the goal point is entirely surrounded by obstacles, leaving no feasible path, then the algorithm returns a failure.

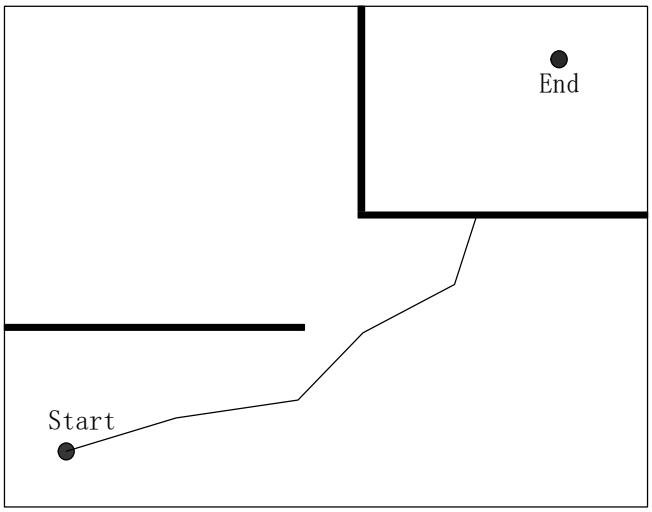


Figure 1. Illustration of Problem 1

Problem 2 (Optimal Path Planning): The cost associated with path planning may vary with each iteration of the algorithm. Let *c* denote a cost function. The objective of optimal path planning, therefore, is to identify a path that yields the minimum non-negative cost. Consider a state space denoted by $X \in \mathbb{R}^d$, with its obstacle-free subset represented as X_{free} . Given an initial state

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

$x_{\text{init}} \in X_{\text{free}}$, a target region $X_{\text{goal}} \in X_{\text{free}}$, and a cost function $c \in \mathbb{R}^+$, the path planning problem can be formally defined. The objective is to identify an optimal path σ^* that satisfies condition $c(\sigma^*) = \min\{c(\sigma) : \sigma \in \sum_{\text{feasible}}\}$; if no such path exists, the algorithm returns failure. As illustrated in Figure 2, the algorithm evaluates the cost functions of two candidate paths. Path 2, being shorter in length and incurring a lower cost, is selected as the final output.

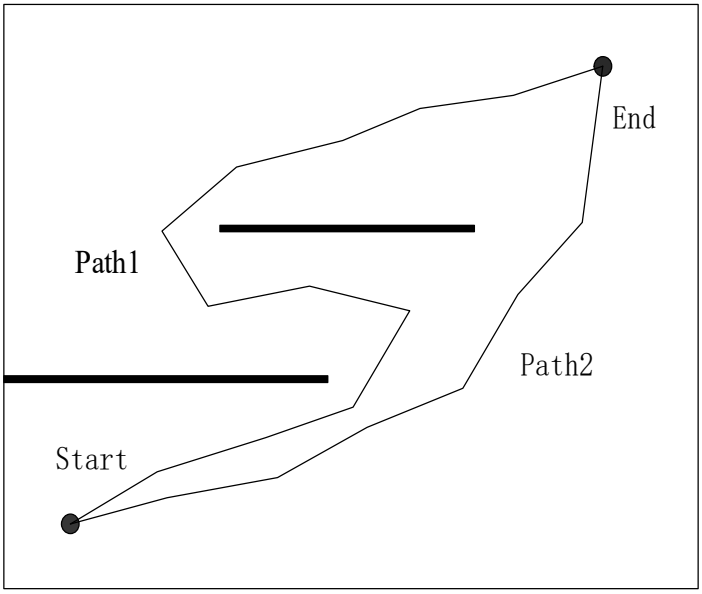


Figure 2. Illustration of Problem 2

2.2 RRT Algorithm

The RRT algorithm is a rapid sampling-based search method, as shown in Figure 3.In the initial phase, the RRT algorithm builds a tree rooted at the initial state x_{init}. Following this, a random sample x_{rand} is generated in the free configuration space X_{free}, and the node x_{near} closest to x_{rand} within the tree is identified. The algorithm then generates a new node x_{new} based on a steering function and performs a collision detection check. If the path between $\{x_{\text{near}}, x_{\text{new}}\}$ and the obstacle space X_{obs} is collision-free, x_{new} is added to the tree with x_{near} as its parent node. Otherwise, a new node x_{rand} is randomly generated for the next attempt. The algorithm concludes successfully if a new node x_{new} is generated within the goal region X_{goal}, at which point the path is deemed complete and the tree is returned. It terminates as failed if the maximum number of iterations is reached. The complete RRT pseudocode is detailed in Algorithm 1

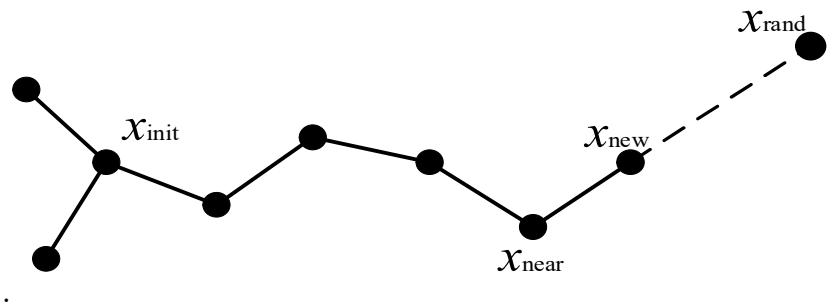


Figure 3. RRT Algorithm Schematic

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

Algorithm 1 RRT Algorithm Pseudocode

Input: $M, x_{\text{init}}, x_{\text{goal}}$

Result: A path σ from x_{init} to x_{goal}

$T.\text{init}();$

for $i = 1$ **to** n **do**

  $x_{\text{rand}} \leftarrow \text{Sample}(M);$

  $x_{\text{near}} \leftarrow \text{Near}(x_{\text{rand}}, T);$

  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{rand}}, x_{\text{near}}, \text{StepSize});$

  $E_i \leftarrow \text{Edge}(x_{\text{new}}, x_{\text{near}});$

  if $\text{CollisionFree}(M, E_i)$ **then**

    $T.\text{addNode}(x_{\text{new}});$

    $T.\text{addEdge}(E_i);$

    if $x_{\text{new}} = x_{\text{goal}}$ **then**

      Success();

2.3 RRT* Algorithm

The RRT* algorithm is an asymptotically optimal variant of RRT. It introduces a rewiring mechanism that continuously optimizes the tree structure during expansion, enabling asymptotic convergence toward the globally optimal path. The pseudocode for the RRT* algorithm is presented in Algorithm 2.

Algorithm 2 RRT* Algorithm Pseudocode

Input: $M, x_{\text{init}}, x_{\text{goal}}$

Result: A path σ **from** x_{init} **to** x_{goal}

$T.\text{init}();$

for $i = 1$ **to** n **do**

  $x_{\text{rand}} \leftarrow \text{Sample}(M);$

  $x_{\text{near}} \leftarrow \text{Near}(x_{\text{rand}}, T);$

  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{rand}}, x_{\text{near}}, \text{StepSize});$

  $E_i \leftarrow \text{Edge}(x_{\text{new}}, x_{\text{near}});$

  if $\text{CollisionFree}(M, E_i)$ **then**

    $X_{\text{rand}} \leftarrow \text{Near}(T, x_{\text{new}});$

    $x_{\text{min}} \leftarrow \text{SampleChooseParent}(X_{\text{rand}}, x_{\text{near}}, x_{\text{new}});$

    $T.\text{addNodeEdge}(x_{\text{min}}, x_{\text{new}});$

    $T.\text{rewire}();$

3. Improved RRT Algorithm

An analysis of the standard RRT structure reveals that its random sampling across the entire free space leads to slow convergence and high memory usage. This global sampling also detrimentally impacts path quality; for instance, even when the tree grows close to the goal, the randomness of the next sample may cause subsequent growth to diverge away from the target. To address these issues, this paper proposes a greedy strategy based on a dynamic sampling space, which systematically reduces the sampling area by modifying the algorithm's core structure. The pseudocode for the improved algorithm is detailed in Algorithm 3.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

Algorithm 3 Improved RRT Algorithm Pseudocode

Input: $M, x_{\text{init}}, x_{\text{goal}}$

Result: A path σ from x_{init} to x_{goal}

T.init( );
for $i = 1$ **to** n **do**
  $x_{\text{rand}} \leftarrow$ DynamicSpaceSampling( );
  $x_{\text{near}} \leftarrow \text{Near}(x_{\text{rand}}, T)$;
  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{rand}}, x_{\text{near}}, \text{StepSize})$;
  $E_i \leftarrow \text{Edge}(x_{\text{new}}, x_{\text{near}})$;
  if CollisionFree(M, E_i) **then**
    T.addNode(x_{new});
    T.addEdge(E_i);
  else T.deleteNode(x_{new});
    if $x_{\text{new}} = x_{\text{goal}}$ **then**
      SmoothedTree $\leftarrow$ Tree;
      return SmoothedTree;

3.1 Dynamic Sampling Strategy

During the first random sampling step from the initial state x_{init}, the sampling range encompasses the entire configuration space. However, once the first new node x_{init} is successfully added to the tree, the improved RRT algorithm dynamically adjusts the sampling space. The pseudocode for this dynamic sampling strategy is outlined in Algorithm 4, and a schematic diagram is provided in Figure 4.

The initial sample is drawn from the entire configuration space. Once the first x_{new} is added to the tree, the improved RRT algorithm dynamically adjusts the sampling space. The pseudocode for this dynamic sampling strategy is presented in Algorithm 4, and a schematic illustration is shown in Figure 4.

Algorithm 4 Dynamic Space SamplingPseudocode

DynamicSpace = ConfigurationSpace $\leftarrow$ Sample(DynamicSpace);
if CollisionFree(M, E_i) **then**
  DifSpace $\leftarrow$ Reduce(DynamicSpace, NextSpace);
  DynamicSpace = NextSpace;
else
  $x_{\text{rand}} \leftarrow$ BackwithForwardSample(DifSpace);

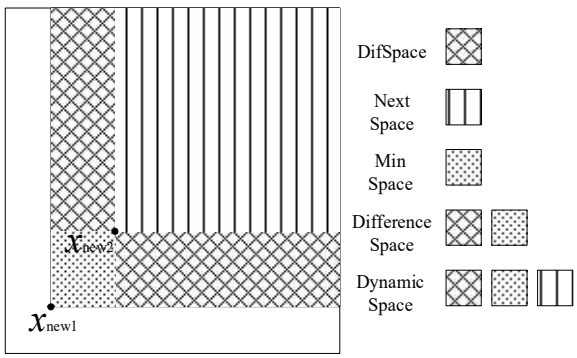


Figure 4. Schematic Diagram of DifSpace

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

The initial configuration space is considered the first Dynamic Space. After the node x_{new} is added to the tree, its x and y coordinates are used as boundaries to partition this space. The resulting quadrant that contains the goal region is defined as the Next Space. In the subsequent sampling iteration, this Next Space is then assigned to be the new Dynamic Space. The area within the Dynamic Space that does not intersect with the Next Space is termed the Difference Space. A small region, called Min Space, is then removed from the corner of the Difference Space diagonally opposite to the Next Space. The remaining area is referred to as DifSpace. This method effectively biases the sampling direction in the next iteration, thereby mitigating the issue of the tree growing backward.

The DifSpace is defined as follows:

$$\text{DifSpace} = \text{DynamicSpace} \setminus (\text{DynamicSpace} \cap \text{NextSpace}) - \text{MinSpace}$$

A key challenge is the potential creation of a confined, obstacle-blocked space as the sampling area shrinks, which traps the planner at node x_{new} with continual collisions. We propose an intelligent backtracking mechanism to overcome this. The mechanism is triggered when consecutive collisions are detected at a single x_{new}, signifying a blocked path within a specific DifSpaceN. The algorithm then prunes this trapped node x_{new} from the tree and reverts the sampling focus to the previously recorded DifSpaceN for subsequent exploration.

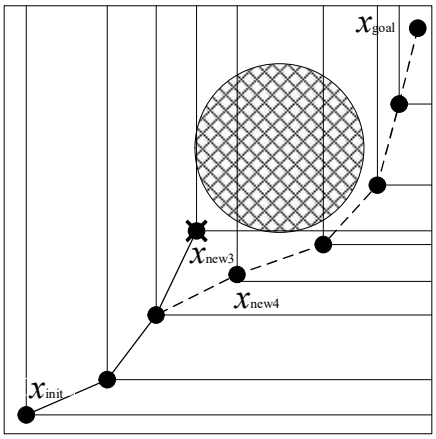


Figure 5. llustration of Blocked Path Resolution

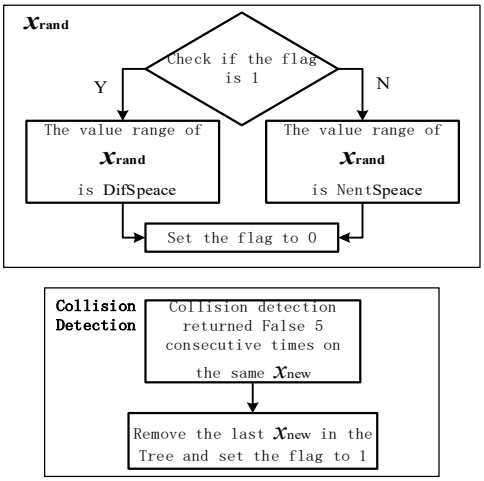


Figure 6. Collision Detection Flowchart

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

A blocked path scenario is illustrated in Figure 5. In this figure, when a collision occurs, the current node x_{new3} is deleted. In the subsequent sampling step, a new node x_{new4} is selected, and subsequent path generation proceeds from this new point, as shown by the dashed line in the diagram. The complete workflow for this process is provided in Figure 6.

A flowchart diagramming the improved algorithm is presented in Figure 7.

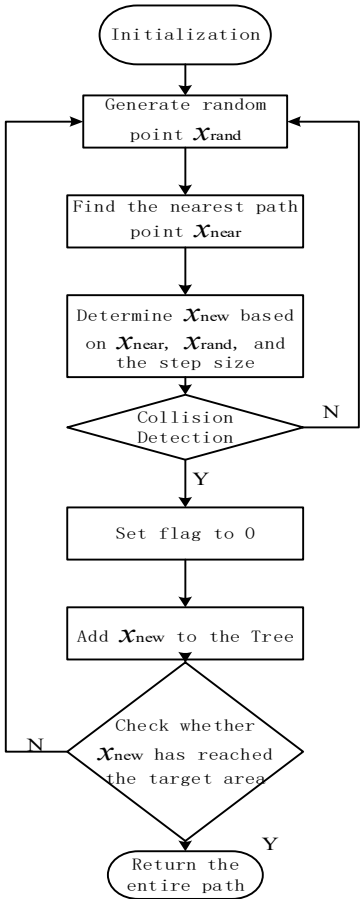


Figure 7. Improved Algorithm Flowchart

3.2 SmoothedTree

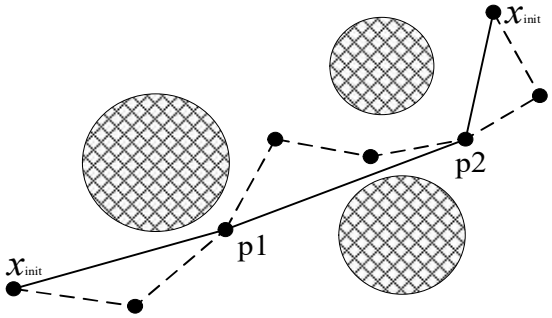


Figure 8. Path Smoothing Schematic

To address the issue of redundant waypoints in paths generated by the RRT algorithm, our improved algorithm performs path smoothing by iterating through and optimizing these waypoints, as illustrated in Figure 8. The process starts from the initial state x_{init} and connects to the other path points in sequence. When the connection to the n-th point is attempted, if an intersection between $\{x_{init}, x_n\}$ and an obstacle X_{obs} is detected, the algorithm instead connects x_{init} directly to the (n-1)-th point,

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

replacing the previous path segment. It then iterates by setting the n-th point as the new starting point and continues connecting subsequent points until the goal state x_{goal} is reached.

4. Simulation Experiments and Analysis

A comparative analysis was performed between the RRT algorithm and the improved RRT algorithm through simulation experiments to validate the performance advantages of the latter, including path curve fitting. The simulations were conducted in PyCharm 2022 on a computer with the following configuration: Windows 10 OS, an Intel Core i7-8750H CPU, and 8 GB of RAM. To ensure an objective evaluation, each algorithm was run 50 times, and the average, maximum, and minimum values of the performance metrics were compared.

4.1 Methodology and Process

To validate the effectiveness of the improved algorithm, a rectangular simulation environment of 12 m × 10 m was constructed, containing six circular obstacles of varying sizes randomly placed within it. The start point was set at (1, 1), and the goal point at (9, 9). The goal region was defined as a circle with a radius of 0.5 centered on the goal point, as shown in Figure 9. To investigate the impact of step size on algorithmic complexity, comparative experiments were conducted using four different step sizes: 0.5, 1, 1.5, and 2.

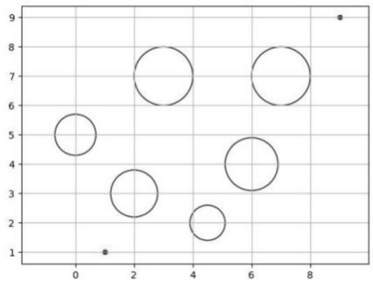
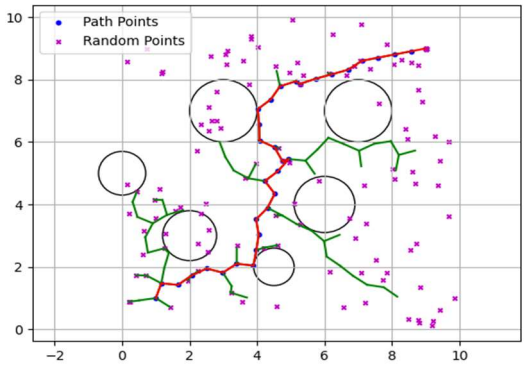


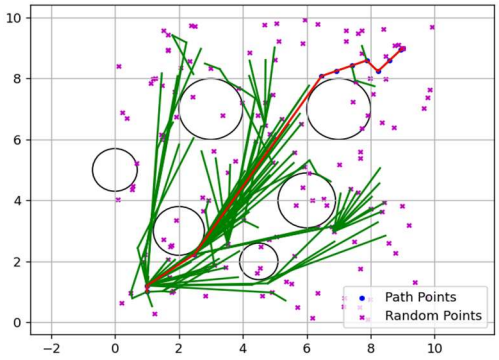
Figure 9. Simulation Environment

The experimental comparison involved the classic RRT algorithm, the RRT* algorithm, and the improved RRT algorithm. The improved version incorporates a dynamic target-biased sampling strategy, which steers the random tree's growth toward the goal region, thereby reducing ineffective search. The simulation results are presented in Figure 10. To mitigate the effects of randomness, each experiment was conducted 50 times. Performance was evaluated using the following three metrics: (1) Time Efficiency: The mean and variance of the search time were calculated, with the variance indicating algorithmic stability; (2) Memory Consumption: Measured by the number of random nodes generated; (3) Path Quality: Assessed by the final path length.

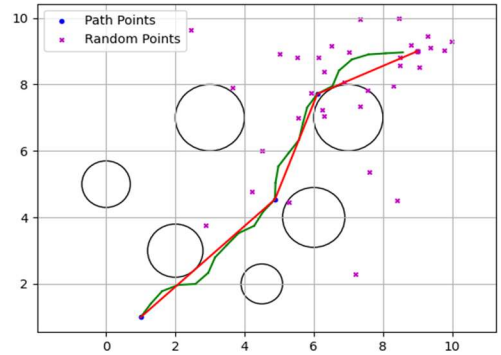


(a) RRT Algorithm

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018



(b) RRT* Algorithm



(c) Improved RRT Algorithm

Figure 10. Simulation Results in a Static Environment

4.2 Results and Analysis

The search time statistics for the three algorithms from simulation experiments are shown in Figure 11. In the figure, the square, circular, and triangular line markers represent the RRT, RRT, and improved RRT algorithms, respectively. Compared to RRT and RRT, the improved RRT algorithm exhibits a significantly shorter search time and a substantially lower variance.

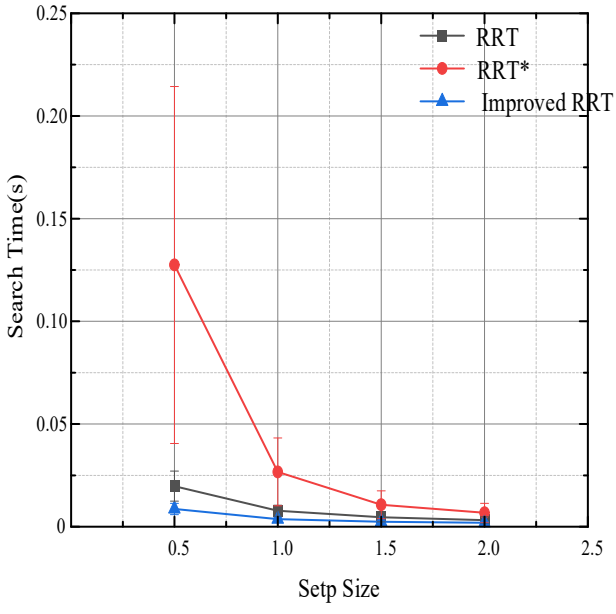


Figure 11. Search Time Comparison

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

In simulations with a step size of 0.5, the improved RRT algorithm significantly outperformed the others, achieving an average time of 0.009 s (variance: 0.003 s) compared to 0.020 s for RRT (variance: 0.007 s) and 0.127 s for RRT* (variance: 0.087 s). This represents a 56.35% and 93.25% faster average search time, with dramatic variance reductions of 61.11% and 96.78%. These superior performance trends held at a step size of 2, with time improvements of 39.18% and 59.56% and variance lowered by 71.47% and 85.90%.

A further comparison of memory consumption among the RRT, RRT, and improved RRT algorithms is presented in Figure 12. The results indicate a significant reduction in memory usage for our improved algorithm. This is because the standard RRT and RRT algorithms sample random points across the entire map, leading to disorganized search paths, the generation of unproductive nodes that consume memory, and potential tree growth away from the goal. In contrast, our improved RRT algorithm employs a dynamic target-biased sampling strategy to confine the sampling area, which progressively steers the tree growth toward the goal and thereby reduces memory consumption.

At a step size of 0.5, the improved RRT algorithm achieved reductions in memory consumption of 50.00% and 37.85% compared to the RRT and RRT* algorithms, respectively, with corresponding variance decreases of 10.77% and 0.85%. When the step size was increased to 2, the memory consumption was reduced by 50.00% and 43.50%, while the variance was lowered by 52.49% and 29.09%, respectively.

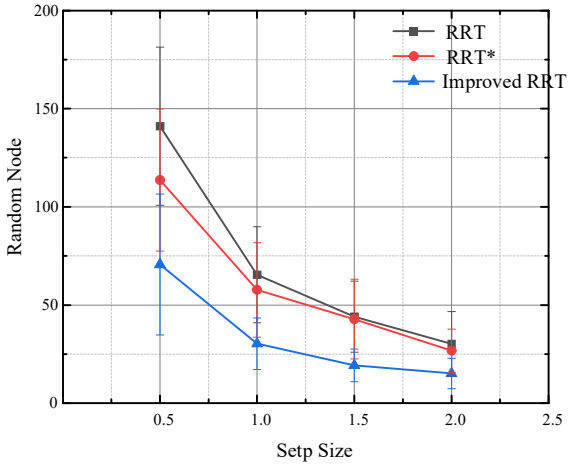


Figure 12. Memory Consumption Comparison

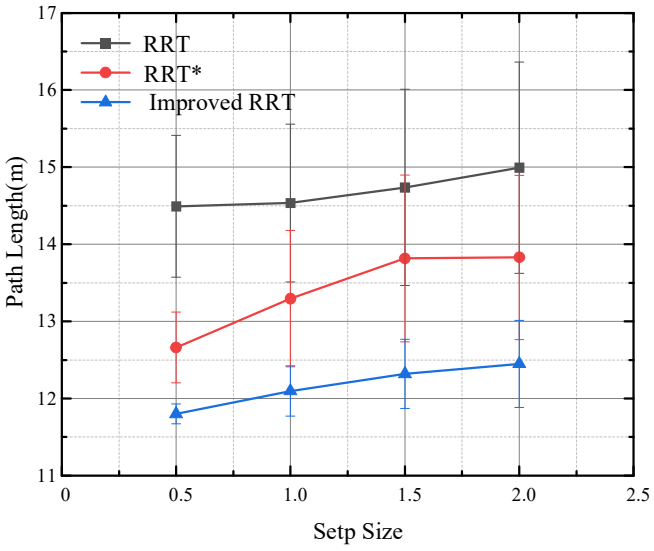


Figure 13. Path Length Comparison

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

Figure 13 compares the path lengths of the three algorithms after fitting. It is evident that the improved RRT algorithm generates a shorter path than both the standard RRT and RRT* algorithms. By applying smoothing and eliminating redundant waypoints, our method effectively shortens the path length, leading to reductions of 16.00% and 6.87% compared to RRT and RRT*, respectively. Furthermore, the path length variance is reduced by 65.00% and 71.89%.

The simulation data for the three algorithms, with a search step size of 0.5,are presented in Table 1.

Table 1. Simulation Results

Algorithm	Number of Nodes	Path Length /m	Computation Time /s
RRT	141	14.5	0.020
RRT*	114	12.7	0.127
Improved RRT	71	11.8	0.009

4.3 Path Smoothing

The initially generated path may contain overly sharp turns, which are problematic for the practical control of robots or UAVs. To address this, we fit the final path into a smooth curve to meet real-world application requirements. This is achieved by employing a Bézier curve for path fitting; the resulting smoothed path is shown in Figure 14.

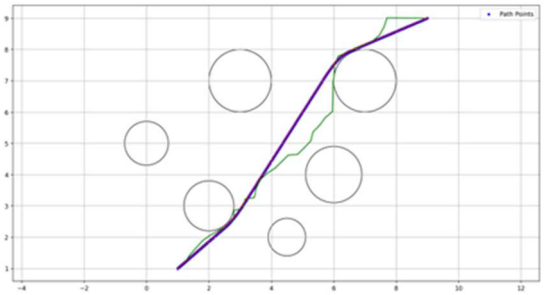


Figure 14. Path Smoothing Result

the thin line denotes the original path, while the thickened line represents the fitted one. It can be observed that the path after curve fitting is significantly smoother.

5. Conclusion

This paper presents an improved RRT algorithm that enhances search speed through a dynamic target-biased sampling strategy and path smoothing. The algorithm also optimizes path length by pruning redundant nodes. Furthermore, it addresses the issue of excessively large turning angles via curve fitting, a feature of significant practical value for UAV and robot control. Experimental results demonstrate that the improved algorithm reduces search time by 49%, memory consumption by 50%, and path length by 12%.

Compared to the classic RRT and RRT* algorithms, the proposed method shows significant advantages in search speed, stability, and path quality, indicating substantial potential for real-world control applications. Future work will explore the integration of our improved RRT with artificial potential fields and ant colony optimization to fully leverage its strengths in complex environments and enhance its overall performance. Additionally, we plan to extend the algorithm into three-dimensional space to better accommodate kinematic analysis requirements.

ISSN: 2414-1895 DOI: 10.6919/ICJE.202603_12(3).0018

References

[1] MA G J,DUAN Y L,LI M Z,et al. A probability smoothing Bi-RRT path planning algorithm for indoor robot[J]. Future generation computer systems,2023,143(5):349-360.

[2] SHI K,WU P,LIU M. Research on path planning method of forging handling robot based on combined strategy[C]//Proceedings of the 2021 IEEE international conference on industrial technology and engineering management. Beijing, China: IEEE, 2021: 123-128.

[3] ZHANG H,TAO Y,ZHU W. Global path planning of unmanned surface vehicle based on improved A-star algorithm[J]. Sensors,2023,23(14):6647.

[4] ALSHAMMREI S,BOUBAKER S. Improved dijkstra algorithm for mobile robot path planning and obstacle avoidance[J]. Computers, materials & continua,2022,72(3):5939-5954.

[5] DIJKSTRA E W. A note on two problems in connexion with graphs[J]. Numerische mathematik,1959, 1(1):269-271.

[6] TAMAS H,BALAZS N,PETER G. Design of a low-complexity graph-based motion-planning algorithm for autonomous vehicles[J]. Applied sciences-basel,2020,10(21):7716.

[7] WANG J K,CHI W Z,LI C M,et al. Neural RRT*:learning-based optimal path planning[J]. IEEE transactions on automation science and engineering,2020,17(4),1-11.

[8] LAVALLE S M. Rapidly-exploring random trees: A new tool for path planning[J]. The annual research report,1998,98(11):1-24.

[9] WEI K,REN B. A met hod on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved RRT algorithm[J]. Sensors,2018,18(2):67-81.

[10] SHI Y Y,LI Q Q,BU S Q,et al. Research on intelligent vehicle path planning based on rapidly-exploring random tree[J]. Mathematical problems in engineering, 2020,20(9):1-10.

[11] HSU D,LATOMBE J C,MOTWANI R. Path planning in expansive configuration spaces[C]//Proceedings of the 1997 IEEE international conference on robotics and automation. Albuquerque, NM, USA:IEEE,1997:2719-2726.