Transactions on Computer Science and Intelligent Systems Research
ISSN: 2960-1800, eISSN: 2960-2238 | Volume 5
AIDML 2024

Comparative Analysis of Hybrid A* and RRT* Algorithms and Optimizing the RRT* Algorithm in Autonomous Maze Navigation

Yanfeng Li

Department of automation, Northwestern Polytechnical University, Xian, Shaanxi, China

1391386134lyf@mail.nwpu.edu.cn

Abstract. The path planning problem of autonomous mobile robots is a very important problem in today's robotics field, and after decades of development, many path planning algorithms have been derived, among which the A* algorithm and the Rapidly Exploring Random Trees (RRT) algorithm are more widely used. These two methods use completely different basic principles, the A* algorithm relies on global search, while the RRT algorithm uses random search, in order to explore the advantages and disadvantages of the two algorithms, this paper mainly analyzes the principles of the A* algorithm and the RRT* algorithm and compares their performances in the maze navigation of robots. After that, the RRT* algorithm is also improved to make up for the shortcomings of the lower efficiency of the RRT* algorithm by utilizing a bidirectional search strategy, and then finally curve fitting is used to obtain a path that is more suitable for the robot's motivation. After comparative analysis, the A* algorithm is more suitable for precise path planning given a global map, and it can calculate the best path quickly. The RRT* algorithm is more suitable for the path planning problem of an unknown map, and due to the principle of stochastic search, it can make the algorithm calculate a reachable path faster.

Keywords: A* algorithm; RRT* algorithm; Path planning.

1. Introduction

The problem of path planning for robots is a very important problem in the field of mobile robotics nowadays. Path planning refers to the algorithms or techniques to determine the optimal or the path that satisfies specific conditions from the starting point to the end point while avoiding obstacles. This field has gained significant attention due to its wide range of applications in various industries, including manufacturing, logistics, transportation, and even space exploration.

In the past, robot path planning was primarily focused on static environments where the obstacles' positions remained constant. However, with the advancement of technology and the increasing complexity of environments, there has been a shift towards developing algorithms capable of handling dynamic and uncertain environments. Attempts have been made to allow robots to explore their surroundings on their own and generate a feasible path.

In general, geometry-based search algorithms can be categorized into certainty search algorithms and random search algorithms. The search behavior of deterministic search algorithms is predictable and repeatable, and the optimal path that meets the performance index can be obtained; while the search behavior of stochastic search algorithms is unforeseeable for the use of random probability factors, and it is not guaranteed to obtain the optimal path, but only a satisfactory path [1]. Usually, certainty search algorithms include shortest path algorithms and dynamic programming algorithms, etc. Shortest path-based methods mainly include Dijkstra's algorithm, Floyd's algorithm and A* algorithm. Common random search algorithms include genetic algorithms, particle swarm algorithms, Ant colony algorithms and other intelligent optimization algorithms [2]. This study centers on contrasting the A* algorithm with the RRT* algorithm, both of which represent distinct approaches to path planning.

One of the fundamental algorithms in robot path planning is the A* (A-star) algorithm, which is widely used for its efficiency and effectiveness in finding the shortest path between two points. A* uses a heuristic to guide the search process [3], making it particularly suitable for static environments.

For dynamic and complex environments, the Rapidly-exploring Random Tree (RRT) algorithm and its variants, such as RRT*, have become popular choices. These algorithms are designed to quickly explore the environment and adapt to changes, making them ideal for real-time applications where the environment is continuously changing.

In this research, a comparative analysis of two algorithms, Hybrid A* algorithm and the RRT* algorithm. Using MATLAB's simulink function for maze modeling and write code for the A* algorithm and the RRT* algorithm to test the robot's ability to plan paths in mazes of varying complexity, witch can be expressed in terms of the time taken to solve the maze as well as the number of nodes to generate the paths.

This research aims to compare and analyze the efficiency of the A* algorithms and RRT* algorithms in maze navigation using MATLAB. A maze environment will be created by MATLAB's simulink function to simulate real-world navigation challenges for robots. Finally, the RRT* algorithm will be optimized using dynamic node step size adjustment and curve optimization methods, such as 3rd order B-spline curves, to improve its performance in finding optimal paths. The study will provide insights into the strengths and weaknesses of each algorithm in dynamic environments, contributing to the advancement of path planning techniques in robotics.

2. Related Work and Introduction to Algorithms

2.1. Maze Modeling

This research use MATLAB’s Robotics Playground Toolbox, The algorithm tested in this article was chosen to be applied to the problem of getting a mobile robot to plan a path through a maze. First, the toolbox's own Mazecreat function is used to generate a random 400×400 size maze, the complexity of which depends on the parameters set by branching; the larger the branching, the higher the complexity of the maze. Firstly, we create a random maze of branching is 11, witch is shown in Figure 1.

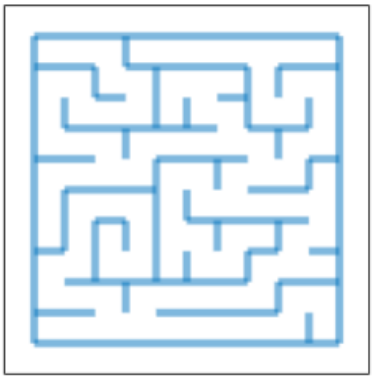


Fig. 1 Branching = 11 random maze (Photo credited: Original)

Then, in performing the simulation experiments, the maze image is transformed into a binary map, which was shown in the Figure 2. Binary map can distinguish between traversable and non-traversable zones. By default, a binary Occupancy Map in MATLAB is expressed as 2D matrix of logical values. Values of 0 corresponds to an obstacle-free area in the map, while values of 1 represent an obstacle. The resolution parameter here is used to provide a mapping between the number of pixels and the physical size of the grid that will be used for path planning.

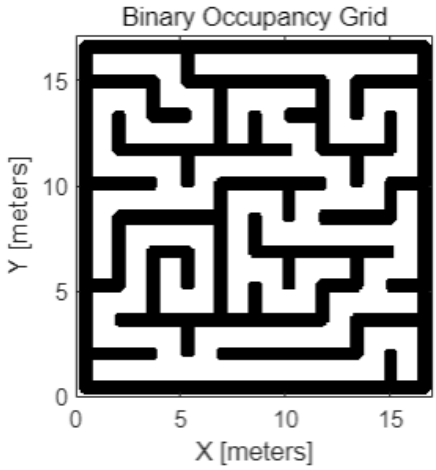


Fig. 2 Binary Maze (Photo credited: Original)

2.2. A-star Algoriths

The A* algorithm, a path planning algorithm [4], is often seen as an enhancement of the Dijkstra algorithm, as it introduces the concept of "expected cost" to node selection. This expected cost, represented in equation (1), guides the algorithm towards the most promising nodes, improving its efficiency in finding optimal paths.

$$f(x) = g(x) + h(x) \tag{1}$$

In the A* algorithm, the total cost f(x) for reaching a node x is the sum of the actual cost from the start node to node x (g(x)) and an estimate of the cost from node x to the goal node (h(x)). The estimate h(x) can be calculated using various methods, such as Euclidean distance or energy loss, depending on the application. The A* algorithm selects nodes to expand based on the sum of g(x) and h(x), aiming to find the path with the lowest total cost.

The A* algorithm is capable of finding the optimal path, which is the path with the lowest total cost among all possible paths from the start node to the goal node. However, the optimality of the path found by A* depends on the accuracy of the map representation used. A higher rasterization accuracy of the map leads to longer computation times for A* but results in a path that is closer to the absolute optimal path for the given map [5].

The A* algorithm used in this study is the Hybrid A* algorithm. The Hybrid A* algorithm is a variant of the A* algorithm designed specifically for autonomous vehicle path planning in continuous state spaces. It combines the benefits of both discrete and continuous state space representations, making it particularly effective for navigating complex environments with obstacles.

One of the key differences between Hybrid A* and A* is the representation of the state space. While A* operates in a discrete state space, Hybrid A* uses a grid-based representation for discretization but maintains continuous state variables for position and orientation. This allows for a more accurate representation of the vehicle's motion in continuous space. In addition, Hybrid A* use of Dubins paths for generating feasible paths in Hybrid A*. Dubins paths are simple curves that connect two configurations of a vehicle while respecting its kinematic constraints [6]. By utilizing Dubins paths, Hybrid A* can efficiently explore the state space and find feasible paths for the vehicle.

2.3. RRT* Algorithm

The Rapidly-exploring Random Tree (RRT) algorithm is a random algorithm. It operates by selecting a new node at random within the search space and extending a path from the current node to this new

node [7]. If the path is obstructed or otherwise invalid, the algorithm discards the result and selects a new random node. However, if the path is valid, it merges the new path with the existing paths. This process continues iteratively until a path to the goal is found or it is determined that no valid path exists

The basic principle is shown in Figure 3. In a state space A an initial node Z_{init} is the start node of the random tree T, and a point Z_{rand} is found in the state space A by a random extension function. And $Z_{rand} \in$ A then traverse the random tree T to find a point Z_{near} closest to Z_{rand} , where Z_{near} $\in$ T, in the direction d formed by the line Z_{near} and Z_{rand} , expands a distance to get a new node Z_{new} , the direction d is the minimum extension length generated for RRT. The new node Z_{new} is added to the node tree when $Z_{new} \in$ A. Otherwise, the new node is re-randomly expanded. The above process is repeated until the target node Z_{goal} is expanded and the random tree construction is completed. Then, the nodes from Z_{init} to Z_{goal} are searched from the random tree, and connect these nodes to form a complete route tree T to get a path.

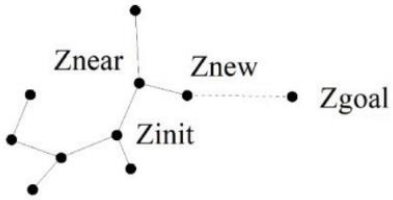


Fig. 3 Principle of the standard RRT algorithm [5]

Building upon the RRT algorithm, the RRT* algorithm has been developed to improve path optimization [5]. While the traditional RRT algorithm focuses solely on expanding new nodes to ensure reachability, the RRT* algorithm introduces additional steps for reselecting parent nodes and rebuilding paths to optimize the trajectory. Reselecting parent nodes process is shown in Figure 4 However, these added steps increase computational complexity, leading to longer running times compared to the basic RRT algorithm.

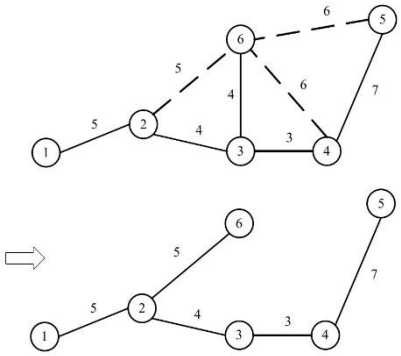


Fig. 4 Reselecting parent nodes of RRT* algorithm [5]

3. Methods

Author wrote A* algorithm and RRT algorithm using MATLAB. In order to verify the feasibility of two algorithms, this section will analyze the simulation results of the RRT* algorithm and A* algorithm under the same experimental conditions, and give the random tree and the effective path of the connected nodes. The execution time and path planning length of the algorithm are compared.

The object-oriented implementation for path planners in MATLAB requires to pre-configure your environment. First, creating a state validator object for collision checking. This contains the environment map and the possible state space the robot can exist in.

After that the algorithm has to be initialized. The path planner must then be initialised object within the state validator object. Using the plannerHybridAStar function and the plannerRRTstar function construct to create the planner and specify its properties, for example, the parameters of MinTurningRadius and MotionPrimitiveLength in A* algorithm; Maximum step size, maximum number of iterations, maximum number of nodes in RRT* algorithm.

Finally, we define the start and goal poses for the vehicle as (x, y, θ) vectors.

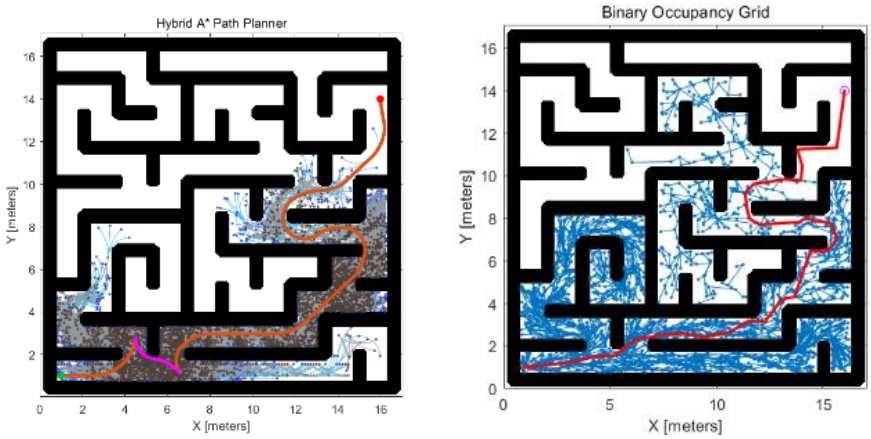
By passing the planner into the plan function, along with the start and goal poses, we plan a path from the start pose to the goal pose. It may be that there is no possible solution depending on the state spaces and map specified.

In this paper, the simulation environment for the two algorithms is a maze that the X-axis coordinate range (0, 17). The Y-axis coordinate range (0, 17), the black branching in the figure is an obstacle, and the (1, 1) and (14, 16) are the starting position and the target position, respectively.

For comparative analysis, this study compares the A* algorithm and RRT* algorithm for autonomous maze navigation. After initializing the algorithm parameters, 11 mazes of varying complexity were created. The two algorithms were then executed and their path planning times and path lengths were compared. In addition, 20 mazes of similar complexity were created to further evaluate the performance of the algorithms by comparing the same parameters.

4. Results

By running the two algorithms, a map of the robot's path through the maze can be obtained as shown in the following figures. Figure 5 (a) is the path generated using the A* algorithm and Figure 5 (b) is the path generated using the RTT* algorithm.

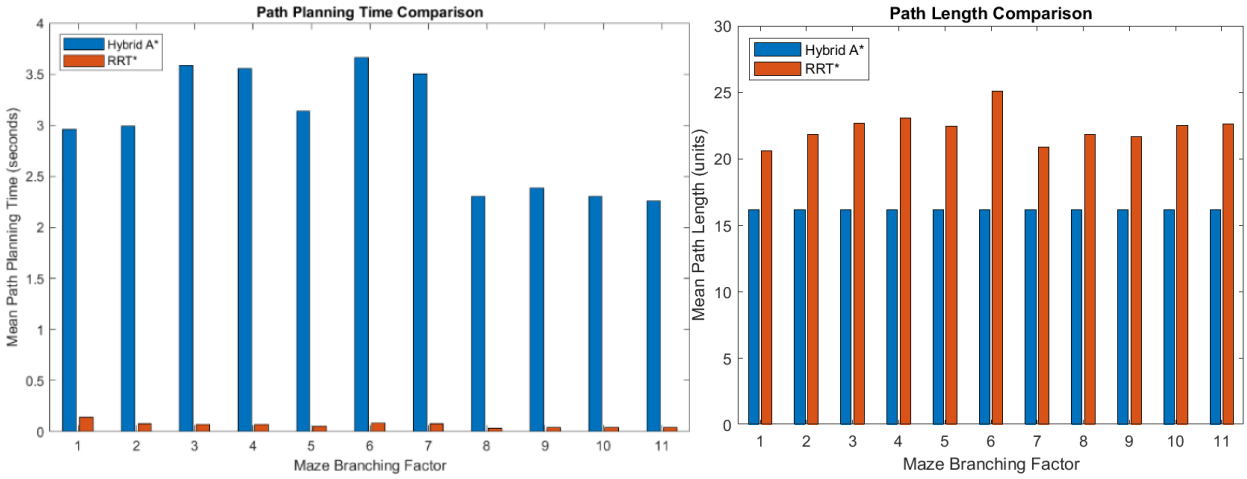


(a) (b)

Fig. 5 Reselecting parent nodes of RRT* algorithm (Photo credited: Original)

Observation of the two path planning graphs reveals that the A* algorithm nodes are covered more closely than the RRT* algorithm, without many invalid nodes like the RRT* algorithm. And the planned curves are smoother and the paths are relatively optimal. Moreover, the paths generated by the RRT* algorithm are more random, and the paths may be different each time. However, the RRT* algorithm is significantly faster than the A* algorithm in terms of path planning time, with the RRT* algorithm taking roughly 2.3 seconds, while the A* algorithm takes 12.5 seconds.

After that, we repeated the test several times for mazes of different complexity and compared the two algorithms in terms of two parameters: path planning time and path length. As shown in Figure 6 below.

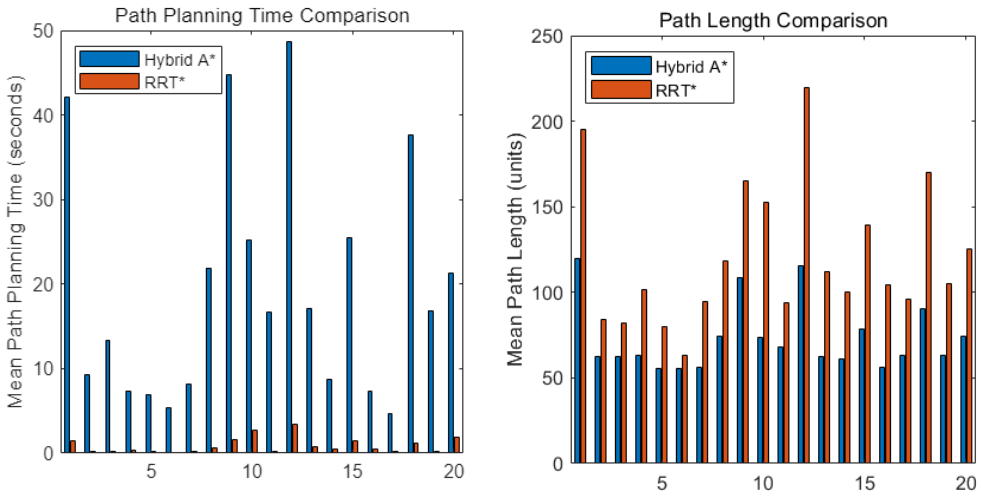


(a) (b)

Fig. 6 Reselecting parent nodes of RRT* algorithm (Photo credited: Original)

The result shows that RRT* algorithm is much shorter than A* algorithm in terms of time and its path planning is faster, but its planned path may not be the optimal path due to the principle of random search, and it also requires more nodes, with a high degree of randomness and more redundancy. The A* algorithm, on the other hand, is based on the principle of grid search, and the path is probable to be the optimal solution, but the computation time is longer.

The authors set the branching parameter to 10 to create mazes of comparable complexity but with distinct structures. This mazes are all 341×341 in size. Twenty path planning trials were conducted using both algorithms, with identical start and goal points. The path planning times and path lengths were then compared, and the results are presented in Figure 7



(a) (b)

Fig. 7 Path Planning Time and Path Length (Photo credited: Original)

From Figure 7, it can be observed that in mazes of similar complexity, the performance of the A* algorithm and the RRT* algorithm is consistent with the previous findings. The RRT* algorithm exhibits significantly shorter path planning times compared to the Hybrid A* algorithm. However, the paths generated by the RRT* algorithm are slightly longer than those generated by the A* algorithm in all 20 times

5. Improving RRT* Algorithm

5.1. Bidirectional Search Strategy

The traditional RRT* algorithms for searching in the workspace are unidirectional randomized expansion from the initial node to the whole workspace, but as the map environment increases and the map becomes more complex, the computational efficiency of the RRT* algorithms becomes lower, and the algorithms' convergence time also becomes longer. In order to further improve the efficiency of the algorithm, this paper adopts a two-way search strategy for path planning. The bidirectional search strategy is that one random tree searches in the workspace with the initial node as the root node, and the other random tree searches in the workspace with the target node as the root node [8]. The bidirectional search strategy can effectively improve the computational efficiency of the RRT* algorithm and reduce the planning time.

The rough process is

Step 1: Initialize the information of start node x_start, goal node x_goal and obstacles in the environment.

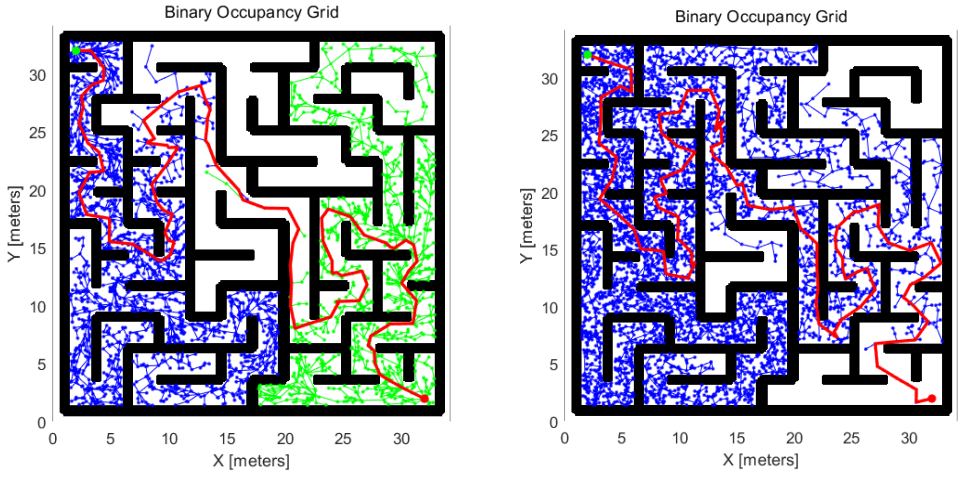
Step 2: Grow a random tree Tree1 or Tree2 from each of the start and stop points, and the two random trees sample randomly in the workspace to generate a random sampling point x_rand.

Step 3:The two random trees Tree1orTree2 each find the nearest tree node x_near to x_rand, and use x_near as the starting point to expand a fixed step in the direction of x_rand to generate a new node x_new.

Step 4: Determine whether x_near and x_new can be connected by collision detection, if they can, add x_new to the random tree, otherwise return to step 2.

Step 5: Determine whether the new node x_new of random tree Tree1 and the new node x_new of random tree Tree2 satisfy the condition of being connected, if they can be, connect them, and carry out path replanning until the number of iterations set by the algorithm is reached, and if they cannot be connected, return to step 2.

The RRT* algorithm after using two-way search is shown in Figure 8



(a) RRT*algorithm (b) using bidirectional search

Fig. 8 Path planning about different algorithms on same maze (Photo credited: Original)

Figure 8 demonstrates that the RRT* algorithm, utilizing a bidirectional search strategy, displays reduced redundant nodes and shorter computation times for path generation. This underscores the enhanced efficiency achieved by the RRT* algorithm, and the overall paths converge more towards the optimal path.

5.2. B-spline Curves Optimization

B-spline curves are widely used parametric curves for curve fitting and interpolation. They offer advantages over Bezier curves, particularly in their ability to make local adjustments without affecting the entire curve's shape, overcoming the limitation of global control [9].

The key principle behind B-spline curve fitting is defining a set of control points that influence the curve's shape. While the curve passes through the first and last control points, the intermediate points are used to smoothly shape the curve.

The smoothness of a B-spline curve is determined by its degree. Higher-degree curves are smoother but require more control points for definition. This trade-off allows for greater flexibility in controlling the curve's shape while maintaining smoothness.

The formula for a B-spline curve of degree p with n control points is given by:

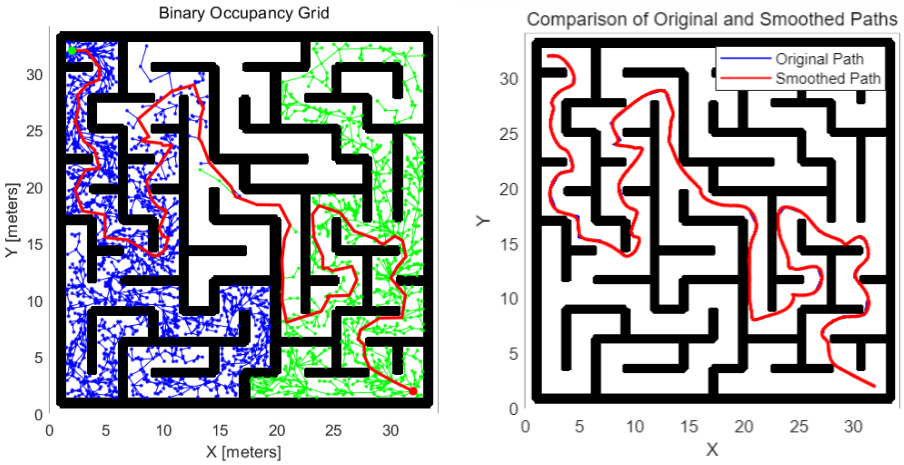
$$C(u) = \sum_{n}^{i=0} N_{i,p}(u) P_i, u \in [0,1] \quad (2)$$

$P(u)$ is the point on the curve at parameter value u.

$N_{i,p}(u)$ are the B-spline basis functions of degree .

P_i are the control points.

In this paper, we use cubic B-spline curve fitting to make the paths obtained by the RRT* algorithm smoother. Figure 9 shows the comparison between the optimized and unoptimized paths using the B-spline curve.



(a) original path (b) smoothed path

Fig. 9 Comparison of original and smoothed path (Photo credited: Original)

Comparing the original path with the smoothed path, it can be observed that B-spline curve smoothing primarily affects local areas. Since the entire route is still based on the composition of each path point, there won't be significant changes to the overall path. However, in some local areas, it can eliminate path spikes and make the route smoother, which is more suitable for the actual path of a mobile robot.

6. Conclusion

In this study, the authors investigate the path planning performance of the A* algorithm and the RRT* algorithm for solving mazes for mobile robots. The experimental setup consists of implementing both algorithms in MATLAB and comparing their performance based on path planning time and path length.

The A* algorithm is known for its efficiency in finding the shortest, optimal paths and is suitable for searching for the best path when the global map is known. In experiments, especially in relatively simple mazes, it is able to compute the optimal path quickly. However, in more complex maze configurations, the A* algorithm struggles due to its deterministic nature and tendency to fall into local minima.

On the other hand, the RRT algorithm, as a probabilistic method, is more adaptable to complex maze environments. Its randomness allows it to explore the maze more efficiently, often finding obscure paths. However, compared to the A* algorithm, the RRT algorithm sometimes produces longer paths, especially if exploration is not efficient.

To address the shortcomings of the RRT* algorithm, two other improvements are proposed in this experiment, and the RRT* algorithm with the addition of a two-way search strategy can plan feasible paths more quickly. Afterwards, in using curve optimization, the obtained paths are smoothed to make them more suitable for the real physical model of the mobile robot.

Overall, the choice between the A-star and RRT algorithms depends on the specific requirements of the application. In situations where finding the shortest path is critical and the maze structure is relatively simple, the A-star algorithm may be more suitable. However, in complex maze environments where exploration and adaptation are key, the RRT algorithm may provide better performance.

References

[1] Z. He and L. Zhao, The Comparison of Four UAV Path Planning Algorithms Based on Geometry Search Algorithm, 2017 9th International Conference on Intelligent Human-Machine Systems and Cybernetics (IHMSC), Hangzhou, China, pp. 33-36, 2017.

[2] S. Xiaoqiang, L. Zhichao, Z. Xuan and N. Xinchao, Research on Robot Path Planning Based on Improved Adaptive Ant Colony Algorithm, 2019 Chinese Control And Decision Conference (CCDC), Nanchang, China, pp. 506-510, 2019.

[3] K. Tu, S. Yang, H. Zhang and Z. Wang, Hybrid A∗ Based Motion Planning for Autonomous Vehicles in Unstructured Environment, 2019 IEEE International Symposium on Circuits and Systems (ISCAS), Sapporo, Japan, pp. 1-4, 2019.

[4] Fda B , Aba B , Mka B , et al. Path Planning with Modified a Star Algorithm for a Mobile Robot - ScienceDirect[J]. Procedia Engineering, 96:59-69, 2014.

[5] Q. Zhou and G. Liu, "UAV Path Planning Based on the Combination of A-star Algorithm and RRT-star Algorithm," 2022 IEEE International Conference on Unmanned Systems (ICUS), Guangzhou, China, pp. 146-151, 2022.

[6] X. Li, Y. Fang and W. Fu, "UAV Path Planning Based on Shuffled Frog-Leaping Algorithm and Dubins Path," 2020 39th Chinese Control Conference (CCC), Shenyang, China, pp. 3990-3995, 2020.

[7] L. Zhao, J. Liu and J. Wang, "Path Planning of Sand Blasting Robot Based on Improved RRT Algorithm," 2019 IEEE International Conference on Mechatronics and Automation (ICMA), Tianjin, China, pp. 1901-1906, 2019.

[8] C. Li, H. Ma, J. Wang and M. Q.-H. Meng, "Bidirectional Search Strategy for Incremental Search-based Path Planning," 2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Detroit, MI, USA, pp. 7311-7317, 2023.

[9] Y. Xie, J. Yang, M. Feng, W. Huang and J. Li, "Path planning of grinding robot with force control based on B-spline curve," 2019 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dali, China, pp. 2732-2736, 2019.