A Method Integrating RRT and A-Star Algorithms to Enhance Obstacle Navigation and Optimization

Shichao Yin

Information Department, Shanghai Waigaoqiao Shipbuilding Co., Ltd, Shanghai, China

563479882@qq.com

Abstract. This paper addresses the issue of path planning for robots in complex environments by proposing a hybrid path planning method that integrates the Rapidly-Exploring Random Trees Algorithm(RRT) and the A-STAR Search Algorithm(A-Star). The method leverages the rapid exploration capability of the RRT algorithm to generate an initial path, and combines it with the heuristic strategy of the A-Star algorithm to optimize the path, particularly in the areas near obstacles. Experimental results show that, compared to using RRT or A-Star algorithms alone, the proposed hybrid algorithm can generate shorter, smoother, and near-optimal paths while maintaining planning efficiency. Specifically, the hybrid algorithm demonstrates good performance in terms of path length, computation time, and path smoothness. Future work will focus on further optimizing the algorithm to enhance its adaptability in more complex environments and considering its application in dynamic obstacle environments.

Keywords: Hybrid Path Planning Algorithm, Path Optimization, RRT, A-STAR.

1 Introduction

In recent years, with the advancement of robotics, particularly in the fields of autonomous vehicles, drones, and industrial robots, the study of path planning algorithms has become increasingly important. Path planning algorithms assist robots in finding a safe path from a starting point to a target point in unknown or partially known environments. Among these algorithms, RRT and the A-Star are the two most commonly used methods, although RRT is favored for its ability to effectively explore high-dimensional spaces and find solutions in complex environments [1], it suffers from low search efficiency and non-optimal paths [2]. On the other hand, the A-Star algorithm is renowned for its capability to efficiently find optimal paths, but its applicability is mainly limited to low-dimensional grid environments and it tends to become trapped in local optima [3].

To overcome these limitations, various improvements have been proposed by scholars. For example, in [4], the authors introduced an improved RRT algorithm, which guarantees asymptotic optimality of the path, although it is more complex to implement and has a slower convergence rate. Additionally, [5] presented a hybrid approach that combines the artificial potential field method with the A-Star algorithm, aiming to enhance path smoothness and avoid obstacles. However, this method still faces limitations in complex environments, especially when dealing with dense obstacles.

CONF-CIAP 2025, January 17-23, Eskisehir, Turkey

DOI 10.4108/eai.17-1-2025.2355326

Another improvement approach is to hybridize different types of algorithms to compensate for the deficiencies of a single algorithm. In [6], a hybrid algorithm combining RRT and A-Star was proposed, utilizing the heuristic information from A-Star to guide RRT's exploration, thereby improving search efficiency and path quality. Although these methods have made some progress, they still face challenges in practical applications, such as high computational costs and lack of path smoothness.

In light of this, the paper proposes a new method that integrates RRT and the A-Star algorithms, aiming to combine their strengths to address the issues in existing algorithms. Specifically, this method leverages RRT's rapid exploration capabilities while incorporating A-Star's heuristic strategies to optimize path selection, with the goal of obtaining shorter, smoother, and safer paths. This method not only effectively avoids obstacles but also significantly reduces the path planning time.

2 Algorithm Principles

2.1 RRT

RRT is a path planning algorithm that efficiently explores complex, high-dimensional spaces by randomly sampling points within the space, connecting each new point to the nearest node in the tree, and incrementally building a tree structure that extends towards the goal. The specific steps of the algorithm are as follows.

(1) Initialization. As shown in figure 1, set the start and end points, and create a tree that initially contains only the start point.

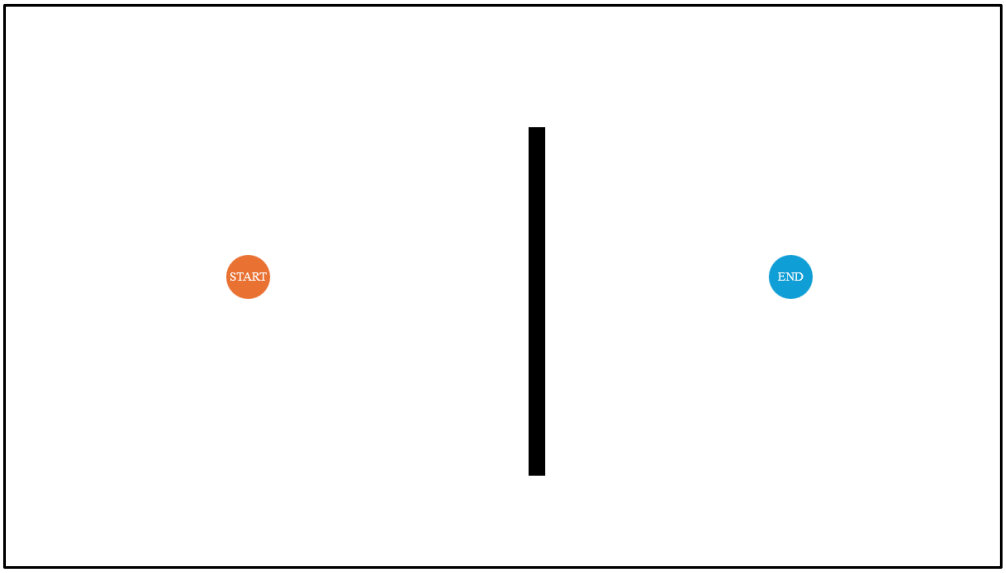

Fig. 1. Setting Start and End Points.

(2) Random Sampling. Randomly sample a point within the configuration space, as illustrated by the green point in figure 2.

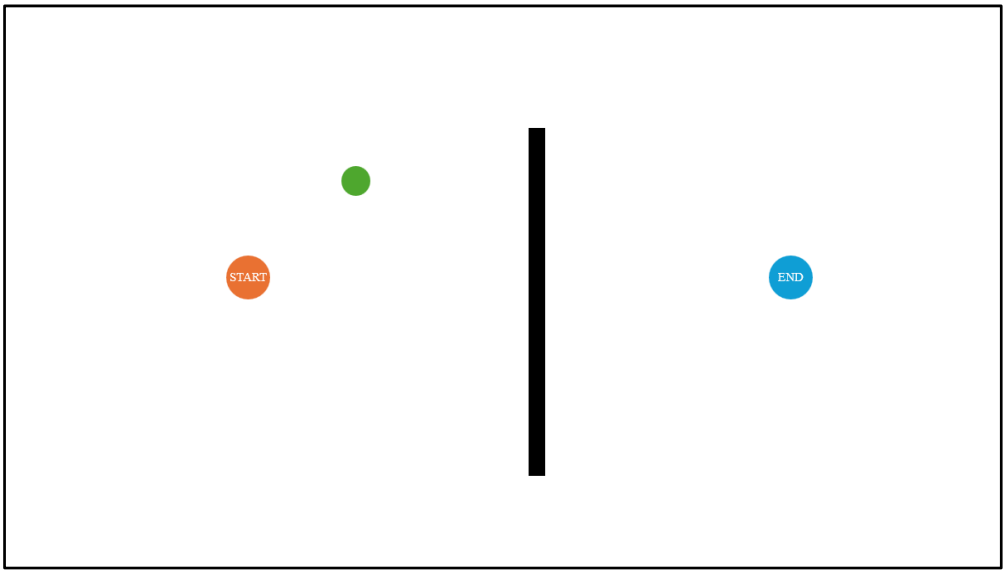

Fig. 2. Randomly Sampling a Point.

(3) Selecting the Nearest Node. Identify the node in the tree that is closest to the sampled point. As shown in figure 3, the tree currently contains only the start point.

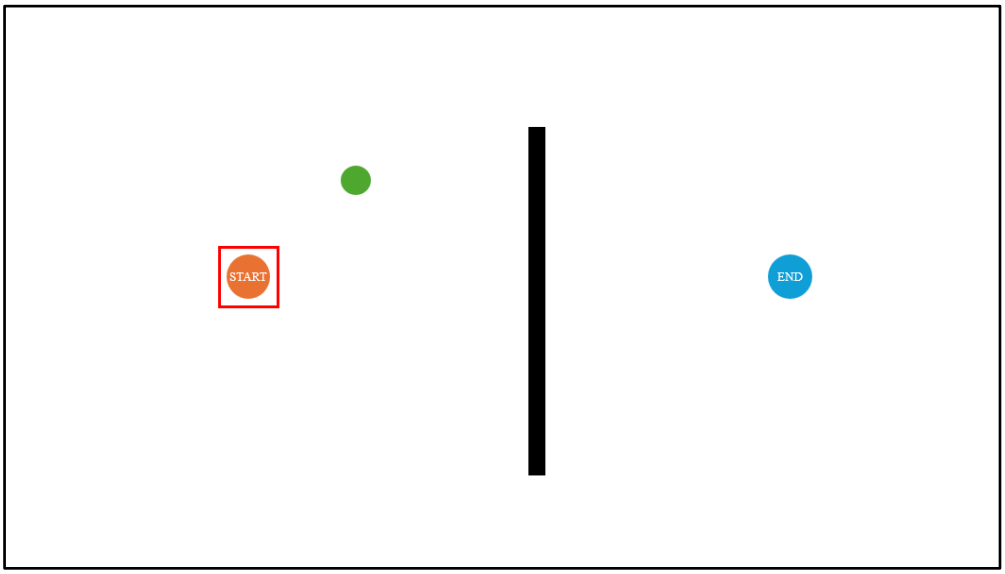

Fig. 3. Selecting the Nearest Node.

(4) Expanding the Tree. From the nearest node, extend the tree towards the sampled point by a certain step size, generating a new node, as depicted in figure 4, where the dashed circle represents the step size range.

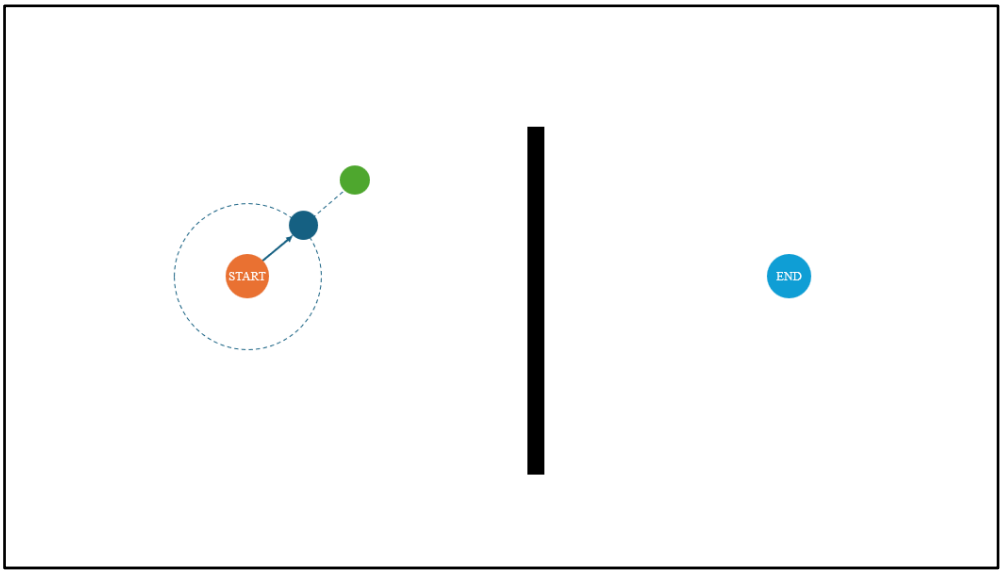

Fig. 4. Expanding the Tree Towards the Sampled Point.

(5) Collision Detection. Check whether the path between the new node and the nearest node collides with any obstacles. As shown in figure 5, if the newly generated path collides with an obstacle, the path is discarded. If there is no collision, add the new node to the tree.

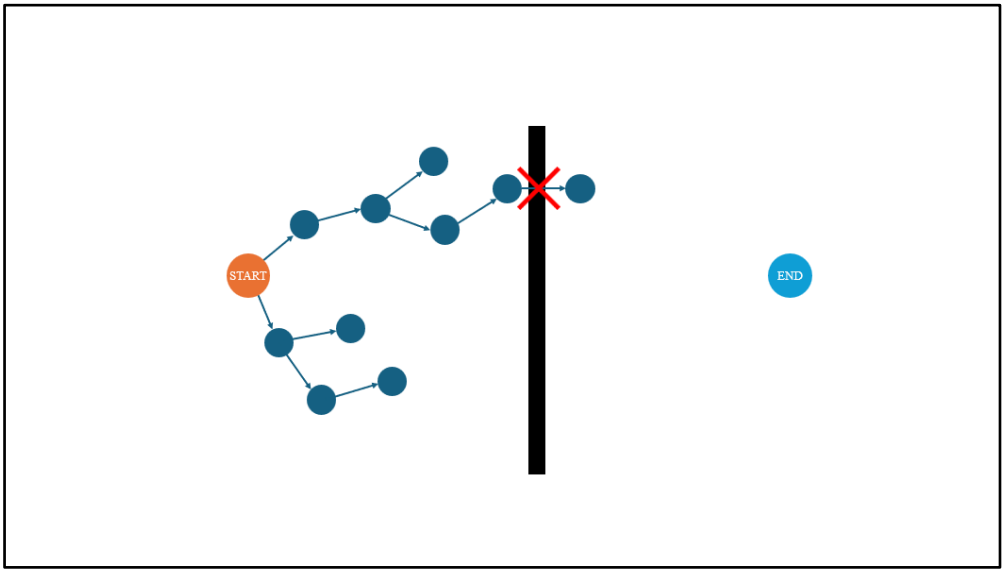

Fig. 5. Checking for Obstacles.

(6) Checking Goal Achievement. If the new node is close enough to the goal, stop the algorithm and return the path, as illustrated in figure 6, where a complete path has been generated. If not, repeat steps 2 through 5 until either the maximum number of iterations is reached or a path is found.

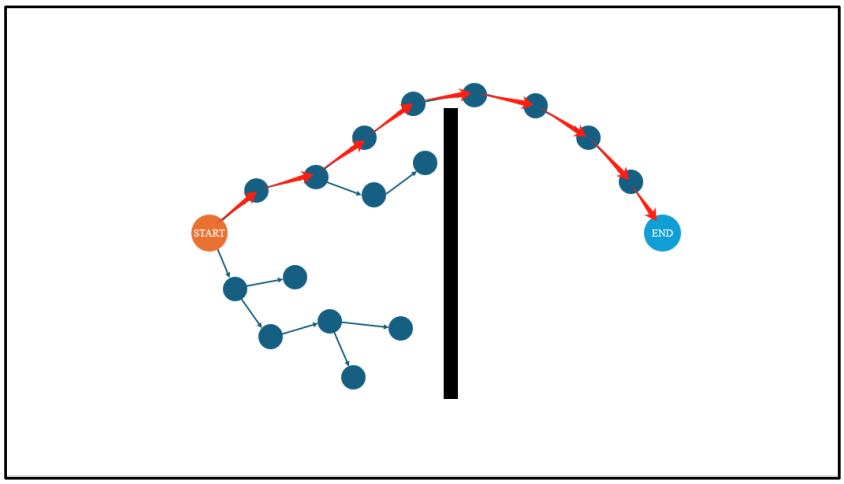

Fig. 6. Expanding the Tree to Reach the Goal.

2.2 A-Star

The A-Star algorithm is a shortest path search algorithm based on heuristic information. Its core concept is to prioritize the expansion of paths that are most likely to reach the goal by combining actual costs with estimated costs until the shortest path is found.

Explanation of the Cost Function. Details are as follows.

(1) Actual Cost $g_{neighbor}$.

This represents the actual cost of reaching the neighbor node from the start node, calculated as shown in equation (1).

$$g_{neighbor} = g_{current} + d(current, neighbor) \tag{1}$$

Where.

$g_{current}$ is the actual cost of the parent node current of the newly added node neighbor.

$d(current, neighbor)$ is the distance between the newly added node neighbor and its parent node current, i.e., the step size.

(2) Heuristic Cost $h_{neighbor}$.

This represents the estimated cost from the neighbor node to the goal node, calculated as shown in equation (2).

$$h_{neighbor} = heuristic(neighbor, goal) \tag{2}$$

Where.

$heuristic(neighbor, goal)$ is the estimated cost function from the newly added node neighbor to the goal node goal. This is typically calculated using the Euclidean distance or Manhattan distance, such as.

$$heuristic(neighbor, goal) = [(X_{goal} - X_{neighbor})^2 - (Y_{goal} - Y_{neighbor})^2]^{1/2};$$

(3) Total Cost Function $f_{neighbor}$.

The comprehensive cost x of node neighbor is calculated as shown in equation (3).

$$f_{neighbor} = g_{neighbor} + h_{neighbor} \tag{3}$$

Cost Function Usage. The algorithm iterates over the nodes based on the total cost function until the goal node is found. The detailed steps are as follows.

(1) Initialization. Define the start node start and the goal node goal. Initialize the open list open_set, and add the start node to it. Set the cost start of the start node to 0. Initialize the closed list close_set to store nodes that have already been processed.

(2) Node Expansion. Remove the node with the smallest cost, referred to as current (i.e., the node with the smallest $f_{current}$), from the open list and place it in the closed list. Check if current is the goal node; if it is, the algorithm terminates and returns the path. Otherwise, generate all neighboring nodes of current.

(3) Cost Calculation. For each neighboring node neighbor, calculate the actual cost $g_{neighbor}$ from the start point to that node, the heuristic cost $h_{neighbor}$, and the total cost $f_{neighbor}$. The detailed algorithm is provided in the formulas above.

(4) Node Processing. If neighbor is already in the closed list and the g value of the new path is greater, skip this node. If neighbor is not in the open list, add it and set its parent node to current, then record the g, h, and f values. If neighbor is already in the open list but the g value of the new path is smaller, update its parent node to current and update the g, h, and f values.

(5) Repeat Steps 2-4. Until the goal node goal is found or the open list is empty (indicating that no solution path was found).

2.3 Hybrid Algorithm

This paper presents a hybrid path planning algorithm that combines the strengths of RRT and the A-Star to overcome their respective limitations. Specifically, the algorithm leverages the exploration capability of RRT to generate an initial path, then employs the A-Star algorithm to optimize the path in areas near obstacles, and finally applies path smoothing techniques. The detailed principles and implementation process of this algorithm are as follows.

The algorithm begins by using the RRT method to generate a basic path, following the principles described previously. Next, for each path node, it checks whether the node is near an obstacle (i.e., whether there is an obstacle within a certain radius). For regions near obstacles, the algorithm replans the path between these nodes using the A-Star algorithm. The A-Star algorithm takes into account the actual cost from the current node to the goal node and a heuristic estimate, always choosing the path with the lowest cost. This approach allows for the identification of safer and smoother paths in areas near obstacles. Finally, the algorithm

performs path smoothing by checking if the line connecting adjacent nodes in the path collides with any obstacles. If there is no collision, these two nodes can be directly connected, thereby eliminating any unnecessary intermediate nodes.

3 Experimental Plan

3.1 Experimental Environment

This experiment was conducted on a Windows 11 (64-bit) operating system with 32GB of Random Access Memory and an Intel Core i5-12500H processor. The algorithm was implemented using Python 3.10, and the visualization was carried out using the pyplot module from the matplotlib library. The specific configurations of the map used in the experiment are as follows.

- The map configuration includes the size and shape of the grid, as well as the placement of obstacles that vary in size and shape to simulate a realistic environment;
- The simulation environment is a two-dimensional plane with a grid size of 100*100;
- Several obstacles were placed within the grid to simulate a complex environment. The layout of obstacles remained consistent across all experiments to facilitate the comparison of algorithm performance. The generated map is shown in figure 7.

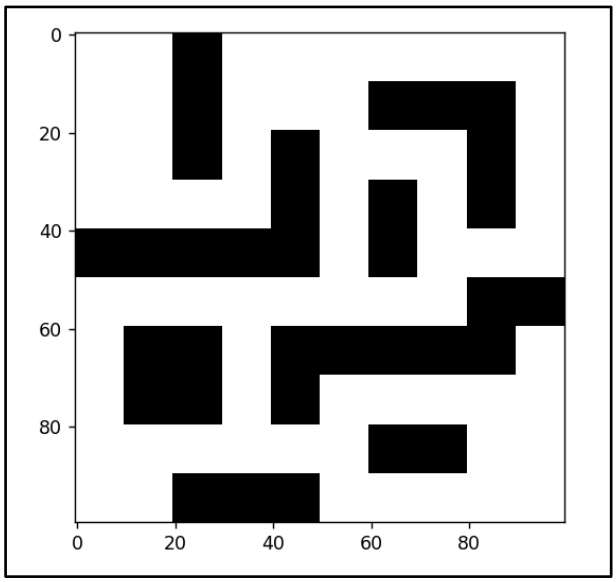

Fig. 7. Map Used in the Experiment.

3.2 Performance Metrics

Path length, path smoothness, and computation time were selected as the criteria to comprehensively evaluate the algorithm's performance across different aspects. These criteria each focus on different performance indicators and collectively influence the practical effectiveness and utility of path planning.

Average Path Length. Path length measures the total distance of the path from the start point to the goal. The reasons for choosing path length as a criterion include: efficiency, feasibility, and practical application:

- **Efficiency.** A shorter path usually indicates higher efficiency and lower resource consumption (such as energy and time). For mobile systems like robots or autonomous vehicles, a shorter path can save fuel or power and reduce wear and tear;
- **Feasibility.** In certain scenarios, a shorter path may be easier to achieve or safer, especially in resource-limited or complex environments;
- **Practical Application.** In real-world applications, the goal is often to find the shortest path from the start point to the goal, which can enhance the efficiency and effectiveness of task execution.

Number of Path Turns. The number of path turns measures the quantity and degree of turns along the path by counting the number of corners or sharp bends. The reasons for selecting the number of path turns as a criterion include:

- **Motion Comfort.** For passenger or cargo transport, a more comfortable experience, reducing discomfort caused by sharp turns;
- **Mechanical Constraints.** For certain robots or vehicles, frequent sharp turns may not align with the design requirements of the mechanical structure, and a smoother path can reduce stress and wear on the steering system;
- **Dynamic Response.** A smoother path is easier to predict and track, allowing control systems to maintain stability and reduce vibrations and oscillations during movement.

Average Computation Time. Computation time measures the duration from the start of the algorithm the discovery of a feasible path. The reasons for choosing computation time as a criterion include:

- **Real-Time Performance.** In many applications, such as autonomous driving or robot navigation, path planning must be completed within a limited time frame to respond to dynamic environmental changes or emergencies;
- **Resource Constraints.** Computation time is directly related to the efficient use of computational resources. Shorter computation times indicate a more efficient algorithm, making it suitable for resource-constrained environments like embedded systems or low-power devices;
- **Practicality.** In practical operations, path planning algorithms need to not only find an effective path but also complete the task within a reasonable time, especially in dynamic environments where frequent replanning is required.

3.3 Implementation Methods

Average Path Length

(1) Single Path Length Calculation. In each run of the algorithm, calculate the distance between each pair of adjacent nodes in the path, then sum these distances to obtain the total length of the path. The specific calculation formula is. path_length = sum(distance(path[i],

path[i + 1])), where distance(path[i], path[i + 1]) represents the Euclidean distance between the ith and i+1th nodes.

(2) Total Path Length Accumulation. For multiple experiments, accumulate the path lengths calculated from each run to obtain the total path length for all experiments.

(3) Average Path Length Calculation. Divide the total path length by the number of experiments to obtain the average path length.

Number of Path Turns. The calculation of the number of turns is based on the change in the path vector at each node. If the angle between two consecutive path segments is less than 170 degrees (i.e., the vector's turn angle is greater than 10 degrees), it is considered a turn. The detailed calculation method is as follows.

(1) Define Vectors. For each segment in the path, define a vector from the previous node to the current node.

(2) Calculate Angles Between Vectors. Compute the angle between each pair of consecutive vectors. If the angle is less than 170 degrees (i.e., the deflection angle is greater than 10 degrees), count it as a turn.

(3) Count the Number of Turns. Traverse all nodes in the path, calculate and accumulate the number of turns.

(4) Calculate the Average Number of Turns. Divide the total number of turns by the number of experiments to obtain the average number of turns.

(3) Average Experiment Time

Average Experiment Time

(1) Single Experiment Time Measurement. For each run of the algorithm, record the start and end times of the algorithm and calculate the difference between them to obtain the time used for that experiment. The specific calculation method is. end_time - start_time, where start_time and end_time are the timestamps of the algorithm's start and end, respectively.

(2) Total Experiment Time Accumulation. For multiple experiments, accumulate the time recorded for each experiment to obtain the total time for all experiments.

(3) Average Experiment Time Calculation. Divide the total time by the number of experiments to obtain the average experiment time.

3.4 Experimental Procedure

Initialization. Set up the experimental environment, including the grid, obstacles, starting point, and destination.

Algorithm Execution. Run multiple experiments (set at 1,000 times) for each of the three algorithms (RRT, A-Star, and the hybrid algorithm). Record the path length, path smoothness, and runtime for each experiment.

Data Collection. Gather the result data from each experiment and calculate the average path length, path smoothness, and runtime.

Data Analysis. Compare the results of the three algorithms, analyzing the strengths and weaknesses of the hybrid algorithm across different metrics.

4 Experimental Results

The visual path diagram generated by the algorithm in the experiment is shown in figure 8 below:

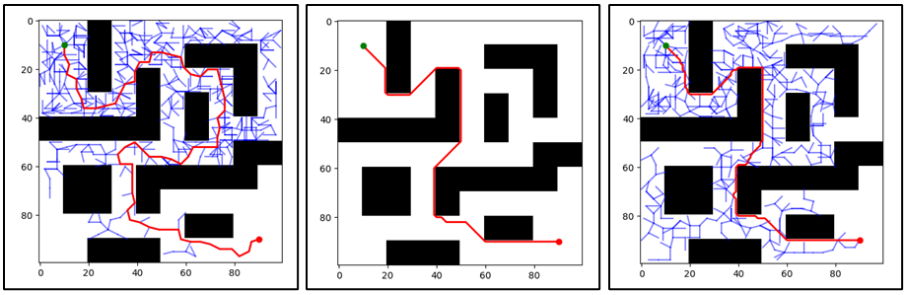

Fig. 8. Visual path diagrams of (a) RRT and (b) A-Star and (c) Hybrid Algorithm

Each experiment compares the performance of the algorithms by analyzing the average path length, average computation time, and average number of turns in the generated paths. The specific results are shown in figure 9 Experimental Results.

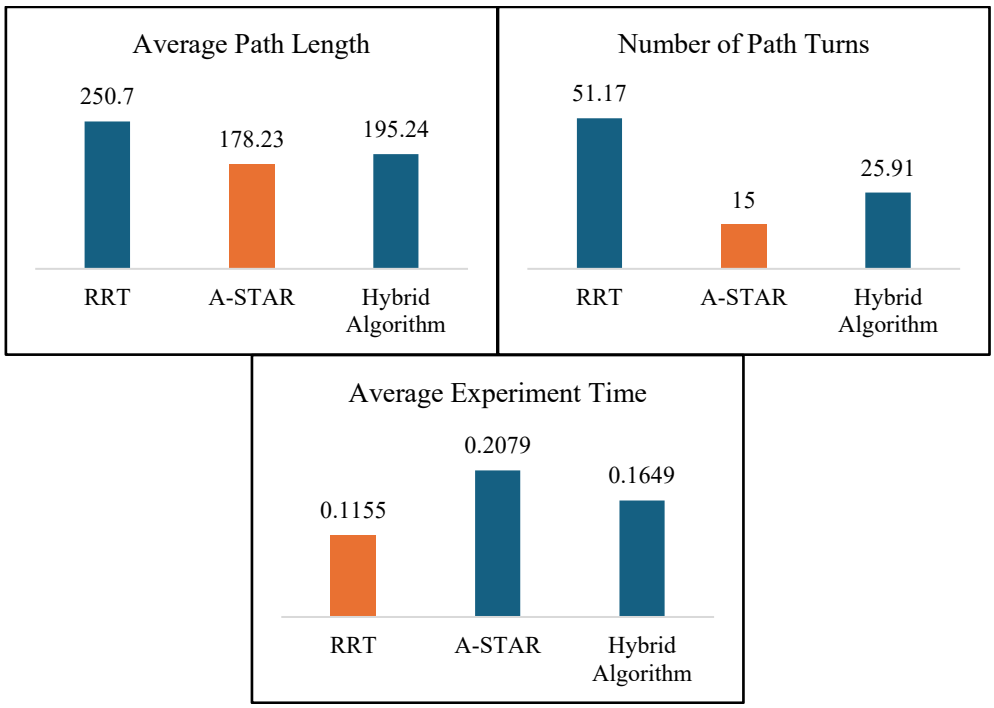

Fig. 9. Experimental Results.

Through the analysis of the path diagram and experimental data, the performance of the hybrid algorithm under different complex environments is demonstrated. The key areas of focus are.

- **Average Path Length.** The hybrid method generates a significantly shorter average path length compared to RRT. This indicates that the hybrid method can find shorter paths while avoiding obstacles, thus improving path optimality;
- **Number of Path Turns.** The hybrid method results in significantly fewer turns compared to RRT. This is because the hybrid method successfully reduces unnecessary turns through the use of the A-Star algorithm and path smoothing steps, resulting in a smoother path;
- **Average Experiment Time.** The hybrid method's average computation time is slightly longer than that of the RRT due to the path optimization using the A-Star algorithm near obstacle areas. However, this increase is acceptable, especially when greater path smoothness and optimality are required in practical applications.

The hybrid algorithm strikes a balance between path length and runtime, offering an optimal compromise. Compared to standalone RRT and A-Star algorithms, the hybrid algorithm produces paths that are slightly longer than A-Star but significantly shorter than RRT, indicating improvement in path optimization. Although its runtime is not as fast as RRT, it is still better than A-Star, demonstrating good computational efficiency. Finally, the hybrid algorithm's path smoothness is between the two, with the number of turns being fewer than RRT but more than A-Star. Overall, the algorithm achieves a favorable trade-off among path length, computation time, and path smoothness.

5 Conclusion

This study proposes a hybrid path planning method that combines RRT and A-Star algorithms to overcome the limitations of each when used independently. The RRT algorithm excels at rapidly exploring large search spaces, while the A-Star algorithm is renowned for its optimal pathfinding capability. By integrating these two approaches, the hybrid method not only improves search efficiency but also generates smoother paths, avoiding lengthy routes and frequent turns. Future research should focus on further optimizing the algorithm to enhance its adaptability in more complex environments and extending its application to dynamic obstacle environments. This proposed hybrid method holds promise for widespread application in fields such as autonomous driving and robot navigation.

References

[1] ABDEL-RAHMAN, Ahmed S., et al. Enhanced hybrid path planning algorithm based on apf and A-Star. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2023, 48. 867-873.

[2] WANG, Dong, et al. Path Planning Based on the Improved RRT* Algorithm for the Mining Truck. *Computers, Materials & Continua*, 2022, 71.2.

[3] ZHANG, Jing, et al. Autonomous land vehicle path planning algorithm based on improved heuristic function of A-Star. *International Journal of Advanced Robotic Systems*, 2021, 18.5. 17298814211042730.

[4] NASIR, Jauwairia, et al. RRT*-SMART. A rapid convergence implementation of RRT. *International Journal of Advanced Robotic Systems*, 2013, 10.7. 299.

[5] ABDEL-RAHMAN, Ahmed S., et al. Enhanced hybrid path planning algorithm based on apf and A-Star. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2023, 48. 867-873.

[6] AL-ANSARRY, Suhaib; AL-DARRAJI, Salah. Hybrid RRT-A*. An Improved Path Planning Method for an Autonomous Mobile Robots. *Iraqi Journal for Electrical & Electronic Engineering*, 2021, 17.1.