

Robot Path Planning Problem by Using A* Algorithm

Jingbo Ni

*Beijing YuYing School, Beijing, China
Jasperni827@asu.edu.pl*

Abstract. Path planning is widely used in today's life, with the A* algorithm being one of the principal methods. As applications become more complex, improving the efficiency of robot path planning has become a crucial challenge. This article mainly introduces the use of the A* algorithm in path planning. The article first introduces the basic logic and expression formula of the A* algorithm. Secondly, the article introduces the application of the A* algorithm in the robot module of MATLAB, demonstrating the route exploration of robots under the A* algorithm. Finally, the article provides a detailed introduction to the cutting-edge research achievements of the A* algorithm, mainly focusing on how to improve the algorithm, shortens time, and enhance efficiency. The article introduces four different methods: Slope comparison method; Reduce node method; Bidirectional search method; the jump point search method. Slope comparison method identifies obstacles on a point-to-point straight line, take the shortest path to reach the obstacle, and then re plan in front of the obstacle to repeatedly reach the endpoint. Reduce node method improves efficiency by decreasing the number of nodes while maintaining straight line segments in the path. Bidirectional search method uses a dual search approach of endpoint and starting point to find duplicate routes to improve efficiency. The jump point search method improves efficiency by identifying and omitting irrelevant nodes along the path based on symmetry in the grid-based maps.

Keywords: A* algorithm, Path Planning, Robot, MATLAB

1. Introduction

Science, technology, and economy are constantly evolving, robots have become assistants to humans. Robots represent an integration of multiple technologies, encompassing various disciplines such as computer science, electronic engineering, control engineering, and mathematics. Robots with high-tech integration have been widely used, manufacturing intelligent machines with perception, decision-making, reasoning, and execution functions to assist or even replace humans in completing dangerous, arduous, and complex tasks. Among them, the widely used robots are mobile robots. One of the main functions of mobile robots is autonomous navigation, which means they can independently move in actual scenes, start from their current position, and autonomously travel according to pre-set target points without human intervention. But to achieve this function, three key issues need to be addressed: the current position of the robot, the surrounding environment, and the driving route. Mobile robot navigation relies on three core technologies: robot positioning, map

construction, and path planning [1]. Robot positioning and map construction serve as fundamental prerequisites for path planning. In intelligent robots, sensors perceive environmental parameters such as physical quantities, chemical quantities, and biomass, and convert them into electrical or digital signals [2]. These input commands become important data for robot position determination and map construction. The input data is processed and analyzed to generate a model and determine its location. After obtaining location and map data, the robot can plan its path. Global and local path planning are the two categories of path planning. Global path planning refers to avoiding obstacles and planning a safe path from the starting point to the final destination point while accurately obtaining environmental information. Local path planning is aimed at avoiding unknown obstacles in the environment, where global environmental information is completely or partially unknown [3].

This article mainly focuses on the introduction of global path planning. The ant colony algorithm imitates how natural ants use pheromones to search for paths. Ants secrete pheromones along the paths they travel, and the concentration of pheromones is higher on shorter paths. Ants tend to choose paths with higher concentrations of pheromones, resulting in a continuous increase in pheromone concentration along shorter paths, forming positive feedback [4]. Dijkstra algorithm selects the node closest to the source node from the previously unvisited nodes and updates the shortest path of all neighboring nodes of that node [5]. The A* algorithm is an efficient direct search method, particularly suitable for finding the shortest path in known and static map environments. It can quickly find the optimal solution within a finite state space, while theoretically ensuring the acquisition of the global optimal solution. This algorithm incorporates Dijkstra's theoretical foundation, utilizing a cost estimation function for path decision-making and gradually expanding the child nodes around the parent node [6].

2. The working principle of A* algorithm

In this part, the paper introduces what is A* algorithm, and A* algorithm working logic. In a fully positioned environment, if this paper want to travel from the began point to the ending point, this paper need to first plan the path. However, because this paper is not clear about unfamiliar environments, this paper needs to explore and try to find the correct route. To solve this problem, the A* algorithm is employed. By reading the literature, the following content has been summarized: the A* algorithm is a heuristic search algorithm and its applications in path planning in a large number of situations. It calculates heuristic function's value at all nodes on the work region and relates the checking of many adjoint nodes for finding the best solution with zero incidence of collision [7].

The function of A* algorithm can be writhed as:

$$f'(n) = g'(n) + h'(n) \quad (1)$$

At this equation, any node on the path can be represented by n, the distance between the original point and the goal point is the estimate function of node n called f'(n), g'(n) is the actual point which from the original point to node n distance, and the estimated distance between node n and the target node is represented by h'(n).

3. Application of algorithm A*

By using the robot extension package in MATLAB, this paper can intuitively feel the working logic of the A* algorithm. The Figure 1 shows the path selection of the robot after applying the A*

algorithm. There are sensors placed in front of the robot, on the left and right sides respectively. When the sensors detect that the distance to the obstacle is too close, they will issue an alarm, which allows the robot to understand the existence of the obstacle. And calculate the path based on the data of these obstacles. It can be seen that the robot did not hit the wall and chose a suitable path to reach the endpoint.

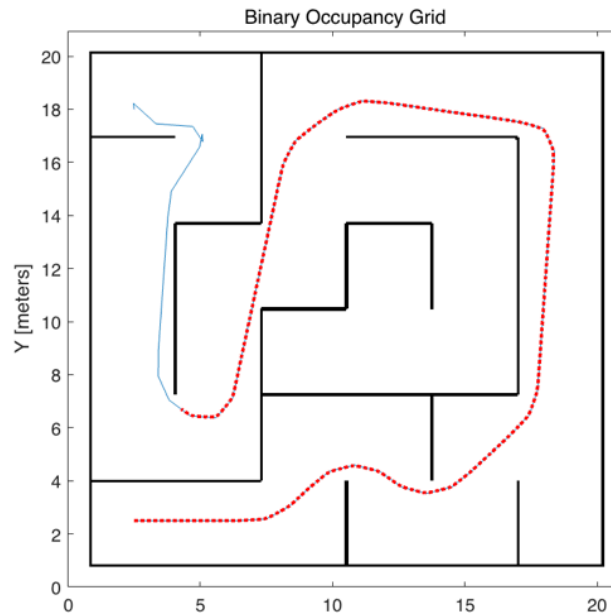


Figure 1. Using the A-star algorithm to navigate the route (photo/picture credit: original)

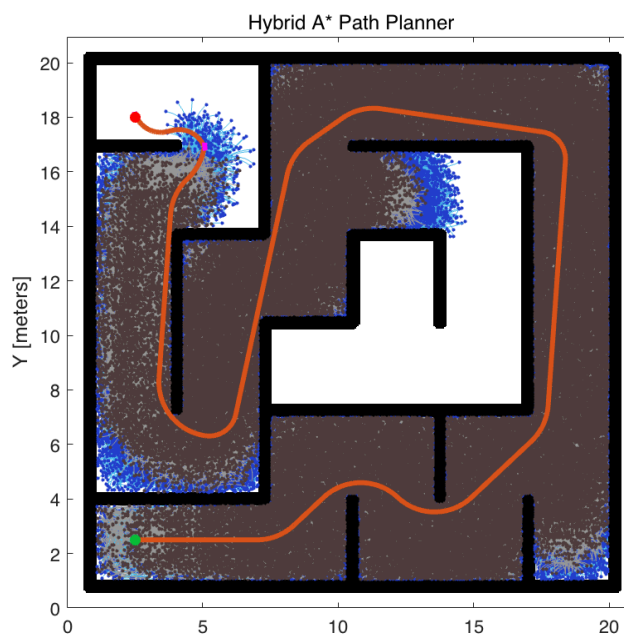


Figure 2. Show the real path of the robot (photo/picture credit: original)

The robot has many more testing paths in the real path display, as can be seen in Figure 2. The robot will first divide into different directional paths and then choose a path in one direction. After a period of time, it will divide into two directions and choose again. Similarly, when the robot reaches

the blue point, its sensors will sense the presence of the wall, which requires the robot to return to the previous branch node and then choose the next path. In this way, the robot will find the correct route through multiple learning and attempts.

4. Current research on the A * algorithm

The A* algorithm has problems such as large storage capacity and slow time. In this section, this paper introduces the improvements made by people to the A* method and the experimental results.

4.1. Slope comparison method

To decrease the time needed for the A* algorithm, Akshay's team has made the following changes. The process begins by converting the robot's workspace into a binary image representation, where obstacles are marked in black. The workspace image is aligned with the coordinate system origin for symmetric processing. Provide the coordinates of the source and target and input these data. Then, examining the distance between the starting point and the end goal. The A* phase progresses further when the proximity condition is satisfied, and the distance is less. Next, the slope of the line with the end point being the source and goal is figured out. The pixels will be projected onto an imaginary line connecting the beginning and the end displays the image from the source if the slope calculated is equal. The modified A* algorithm's usefulness for the set of source and goal coordinates is determined by checking the condition based on those coordinates. Firstly, under the condition that the distance between the starting point and the target is close, the robot operates A* algorithm. But if the slope is found to be inconsistent, the pixel will be projected onto any slope calculated from the values of h and g. After projecting the pixels to the target, a straight-line path will be obtained between the source and the target. During the path, the coordinates of the point before the collision are obtained when the projection comes into contact with an obstacle, which is called the switching point (abbreviated as S point), because after S point, the algorithm turns to the A * algorithm.

After identifying point S, A* algorithm is applied with S as the new source point to find the shortest path to the destination. The switching phase is characterized by the algorithm starting to predetermine the A* algorithm for path planning. The total path is formed by combining the straight-line path from the start to point S and the pathway that leads from point S to the destination. Simply put, can understand this process as repeatedly planning the path (Figure 3) [8].

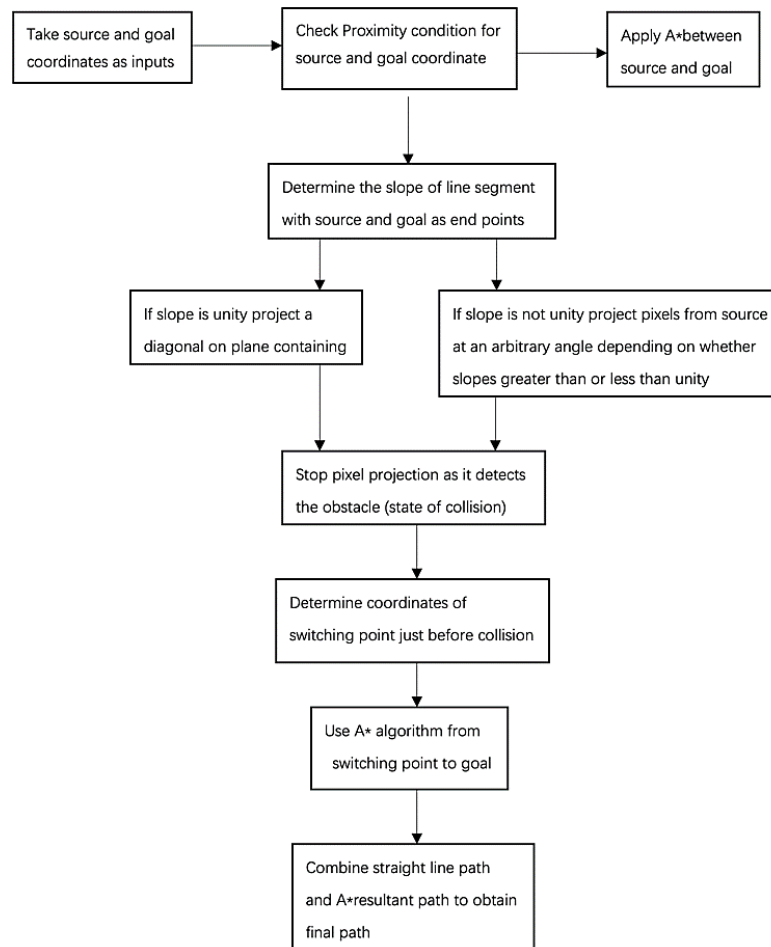


Figure 3. Proposed time-efficient A* algorithm [8]

In this way, the modified A* algorithm has been discovered to reduce processing time by at least 65% and result in an absolute 3.4% increase in path length. The modified A* algorithm is superior to the original A* algorithm. When the source and the first obstacle are separated by more than the straight-line path and eliminates the need to check each adjacent value used in the heuristic algorithm employed, such as the A* algorithm.

4.2. Reduce node method

Based on the principle that straight-line paths minimize distance, node reduction can optimize computational efficiency. To help reduce time, and it can make the moving route smoother, reduce the occurrence of pauses, and thus improve efficiency.

An improved A* algorithm has been proposed to enable robots to quickly and effectively plan a global path away from obstacles. Based on the different distances from the extended node to the target point, a distance based variable weight evaluation function is modified on the basis of the traditional A* algorithm, which improves the efficiency of path search and reduces the required storage capacity. Among them, the impact of $h(n)$ on the A* algorithm is as follows: when the current node is farther away from the target node, the search efficiency should be improved, that is, the proportion of $h(n)$ in the evaluation function should be increased; When the current node is closer to the target node, the accuracy of the algorithm should be improved, that is, the proportion of

the influence of $g(n)$ in the evaluation function should be increased. Select the node closest to the target point, the node with the smallest $h(n)$, as the current expansion node. When there are multiple nodes with the smallest $h(n)$, select the node closest to the starting point, the node with the smallest $g(n)$, as the current expansion node (Table 1).

Table 1. Different result in different ways [9]

	Method	Length/m	Nodes	Second/s
Simple environment	Traditional A*	72.924 65	764	0.139 566
Simple environment	Improved A*	74.459 73	220	0.040 787
Complex environment	Traditional A*	154.985 6	4 111	1.209 12
Complex environment	Improved A*	167.487 1	442	0.089 61

By improving the traditional A* algorithm, a safe and reliable global path is obtained. Firstly, a distance based variable weight evaluation function is proposed, which improves the efficiency of path search and reduces the number of nodes. In both complex and simple environments, the improved version of the A* algorithm takes less time than the regular A* algorithm.

4.3. Bidirectional search method

To enhance the A* algorithm's search efficiency and ensure the optimality of the path, an improved Algorithm A* based on bidirectional search is proposed, which uses the current nodes of forward and backward search as target points for bidirectional search.

The improved algorithm introduces weighted Manhattan distance as the heuristic function. This allows dynamic adjustment of cost function weights, ensuring the real-time performance of the algorithm and the superiority of the path; secondly, to address the issue of local path optimization in bidirectional path search, a cost factor of deviating from the optimal distance is added to the heuristic function. The principle of the bidirectional A* search algorithm is to establish two A* search algorithms, namely forward and reverse. The algorithm initiates forward search from the start point while simultaneously conducting reverse search from the goal point; Starting from the endpoint in reverse, search with the current point of forward search as the target until the forward and backward paths intersect.

The running time of this algorithm ranges from 0.164 38, 0.224 42 seconds shortened to 0.152 87, 0.145 55 seconds. When the map size increases to 50×50 , the number of nodes in this algorithm is reduced by 8 compared to the bidirectional A* algorithm 3%, reducing the path cost by 14.7%, reduced runtime by 19.8% [10]. The integration of weighted adjustment and deviation cost factors yields substantial improvements in computational efficiency and path optimization.

4.4. The jump point search method

Through the previous method, this paper found that reducing the number of points can shorten time, so the jump point search method is also a way to reduce time.

Jump point search represents an advanced variant of heuristic pathfinding methods. Its difference from the A* algorithm is that it selects representative nodes for expansion, and through a certain regular expansion formula, reduces the access of many unnecessary grid points, thereby reducing the computational load. In the path planning process of the A* algorithm, the skip point search strategy is used to handle the process of expanding nodes. It does not expand all nodes adjacent to the current

node but filters out nodes at specific positions. These selected nodes are called skip points. The jump point search method reduces the number of nodes in the path, increasing the efficiency of pathfinding. The improvement of A * algorithm based on the idea of sub node jumping in the jump point method is to perform pre operation on the environment grid before the path planning of A * algorithm: The algorithm identifies jump points based on specific criteria and maintains them in an Open List for evaluation, which reduces the operation of many redundant nodes and lowers the number of redundant nodes. Unlike, A* which explores adjacent nodes within short distances, jump point search enables longer-range exploration from parent nodes, the optimized algorithm with child node jumping can explore for a long distance. Therefore, less time is required for preprocessing in the path planning process, greatly reducing the computational load on the *Openlist* tables and *Closetlist* tables, effectively reducing the memory usage in the path planning process (Table 2).

Table 2. Comparison of experimental results on jumping point improvement of A * algorithm [11]

Map size/m ²	Expand nodes/grids		Search time/s Path length/m		Search time/s Path length/m	
	A * Algorithm	Jumping Point Improvement A * Algorithm	A * Algorithm	Jumping Point Improvement A * Algorithm	A * Algorithm	Jumping Point Improvement A * Algorithm
15×15	25	4	0.0587 7	0.03538	16.1421	172356
20×20	81	4	0.1122 8	0.04147	22.1421	22.3245
25×25	161	10	0.2022 3	0.06115	32.2132	34.5698

Experimental results demonstrate that incorporating jump point search into A* algorithm enhances both computational efficiency and memory management not only has a significant improvement in path planning speed compared to the traditional A * algorithm, but also greatly reduces the memory consumption of the algorithm during runtime.

5. Conclusion

This paper provides a detailed introduction to the A* algorithm, mainly including its basic logic and application in MATLAB. Secondly, the article introduces the improvement methods and results of various scholars on the A* algorithm today. Through this article, this paper can discover the differences and advantages and disadvantages of different improvement methods. In this paper, all the four methods focus on reducing the number of nodes to improve algorithm efficiency. But the methods are different, and the results presented are also different. It can be seen that one of the fundamental principles of improving algorithms is to reduce the number of turning points, which can make straight lines occupy most of the paths in the route and reduce the time consumption of unnecessary paths. Based on the findings presented in this paper, several prospects for future path planning research can be identified. Firstly, as an algorithm that integrates multiple paths planning methods, the A* algorithm has many areas for research and improvement. In addition to focusing on reducing nodes, this paper can also use more methods to improve efficiency, such as making routes smoother and improving search efficiency. This is important for improving algorithms. The A* algorithm shows potential for expansion beyond basic scenario planning into more sophisticated applications. This paper can use it in express delivery and other complex scenarios. Not only can the

A* algorithm be enhanced, but biomimetic knowledge such as animal migration patterns can also be incorporated to develop more scientific and natural path planning approaches.

References

- [1] W. Bian, Taiyuan University of Science and Technology (2023)
- [2] F. R. Yi, Appl. Home Appliance Maintenance Letters 12, 89–91 (2024)
- [3] X. Li, P. S. Zhu, Y. Q. Zhang, Y. X. Chang, Appl. Automation Applications Letters 65, 126–129 (2024)
- [4] A. W. Yu, L. G. Guo, H. B. Zhu, Appl. Journal of Heilongjiang Institute of Technology (Comprehensive Edition) Letters 24, 147–151 (2024)
- [5] K. J. Jin, Appl. Information and Computers (Theoretical Edition) Letters 36, 47–49 (2024)
- [6] C. Y. Tan, Changchun University of Technology (2024)
- [7] R. Yonetani, T. Taniai, M. Barekatin, "Learning sparse representations for machine learning, " in International Conference on Machine Learning, Proceedings of Machine Learning Research 12 (PMLR, 2021), pp. 29–39
- [8] A. K. Guruji, H. Agarwal, D. K. Parsediya, "Analysis of technological innovations in data transmission, " in Procedia Technology, Elsevier Proceedings 23 (Elsevier, 2016), pp. 144–149
- [9] Y. B. He, L. S. Cai, L. Xu, Appl. Southern Agricultural Machinery Letters 55, 31–34 (2024)
- [10] J. L. Zhang, B. Jia, L. Nie, Appl. Journal of Chongqing University of Science and Technology (Natural Science Edition) Letters 26, 89–97 (2024)
- [11] P. Zhang, North University of China (2021)