Scholar Publishing Group
International Journal of Multimedia Computing
https://doi.org/10.38007/IJMC.2025.060102
ISSN 2789-7168 Vol. 6, Issue 1: 19-28

Improved A Algorithm Based on Bidirectional Jump Point Search*

Qingyuan Xiao[1,a] , Meng Qin[1,b], Huiheng Suo[1,c], Guangjun Lai[1,d], Zuteng Chen[1,e], Jian Wu[1,f,*], Yuanhao Pan[2,g], Xie Ma[2,h], Yingping Bai[3,i], Weihong Zhong[3,j]
[1]*Nanchang Hangkong University, Nanchang, China*
[2]*Ningbo University of Finance & Economics, Ningbo, China*
[3] *NingboTech University, Ningbo, China*
[a]*xiaoqy0918@163.com,*[b]*alphenqin@163.com,* [c]*suohuiheng@163.com,* [d]*l2647080898@163.com,*
[e]*2293747855@qq.com,* [f]*flywujian@qq.com,* [g]*1090824061@qq.com,* [h]*maxie88@163.com,*
[i]*3459951916@qq.com,* [j]*zwh@nbt.edu.cn*
[*]*corresponding author*

Keywords: A* algorithm; Path planning; Jump point search; Bidirectional search

Abstract: To address the issues of low path planning efficiency and poor path smoothness of the traditional A* algorithm in complex environments, an improved A* algorithm combining bidirectional jump point search with a direction penalty factor is proposed. This approach enhances search efficiency by changing the traditional A* algorithm's unidirectional search strategy to a bidirectional search, reduces redundant node expansions through jump point search mechanisms, and optimizes path smoothness by introducing a direction penalty factor. Experimental results show that the improved algorithm reduces the number of node expansions by approximately 90%, shortens the search time by 50%, and decreases the path length by 13%, while ensuring that the path length remains close to optimal.

1. Introduction

Path planning refers to the best path formed from the pre-set start point to the specified target point by relying on specific policy methods and meeting certain performance indicators[1]. There are some common path planning algorithms such as Dijkstra algorithm[1], best priority search, A* algorithm[2], D* algorithm, and intelligent algorithms like genetic algorithm[3], particle swarm algorithm[4], etc. Among so many path planning algorithms, A* algorithm is the most widely used for approaching a near optimal solution, fast solution speed and high efficiency, which still needs to be improved. In recent years, researchers have improved algorithm performance by optimizing heuristic functions[5], employing dynamic weighting strategies[6], and incorporating jump point search[7]. However, these enhancements often sacrifice path optimality or struggle to balance real-time response requirements in dynamic environments[8]. To address the limitations of the

traditional A* algorithm, literature[9] proposes a new and enhanced A* algorithm incorporating the Floyd trajectory optimization algorithm, but this approach proved time-consuming. Literature[10] proposed a method to optimize path length by detecting whether the line connecting adjacent nodes crosses obstacles.

To address the high computational cost and the lack of path smoothness in the A* algorithm, this paper proposes an enhanced version of the A* algorithm that incorporates a bidirectional jump point search mechanism to boost search speed and optimize the search process by minimizing unnecessary node expansions. Additionally, the algorithm employs a direction penalty factor to improve path smoothness, thereby significantly enhancing path planning performance in complex environments. This strategy not only increases search efficiency but also ensures the smoothness and safety of the path, making it particularly suitable for applications sensitive to turning constraints.

2. Traditional A* algorithm

A* algorithm is widely used in robot path planning because of its high efficiency. The nodes to be detected in the path planning process are stored in the OpenList, and the detected nodes are stored in the CloseList. From the starting point, the cost of the surrounding grid is calculated by Formula 1. The point with small cost is selected as the next child node, and the list is updated. The updated child node is used as the parent node to recalculate the cost of the surrounding grid until the path is searched to the target point.

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

Where, *g(n)* represents the actual cost from the start node to the current node, while *h(n)* is the heuristic function estimating the predicted cost from the current node to the goal. For the evaluation function *f(n)*, when *g(n)=0*, the original expression becomes the heuristic function *h(n)*, which estimates the cost from node n to the goal. In this case, the A* algorithm degenerates into a greedy Best-First Search (*BFS*), which is computationally fast but does not guarantee an optimal solution. When *h(n)=0*, the original expression reduces to the function *g(n)*, which estimates the cost from the start node to node n. Here, the A* algorithm transforms into Dijkstra's algorithm, requiring the computation of a large number of nodes and resulting in low efficiency. During the search process, the A* algorithm simultaneously computes *g(n)* and *h(n)*, balancing search efficiency while ensuring the discovery of the optimal path.

Traditionally, the A* algorithm uses either Euclidean distance or Manhattan distance as the metric for the movement cost between two points. The distance function is expressed as:

$$h_1(n) = \sqrt{(X_j - X_i)^2 + (Y_j - Y_i)^2} \tag{2}$$

$$h_2(n) = |X_j - X_i| + |Y_j - Y_i| \tag{3}$$

Where, *(Xi,Yi)* and *(Xj,Yj)* represent the coordinates of points n_1 and n_2, respectively. This paper employs the Manhattan distance to measure the movement cost between two points. Using the Manhattan distance provides a way to estimate the cost based on a grid-like path where only vertical and horizontal movements are allowed, making it particularly suitable for scenarios with such constraints[11].

3. Improved A* algorithm

3.1 Jump Point Search

The jump point search is to find the point that can jump between the starting point and the target

point, and find the path by searching between jump points. This is the search strategy of the jump point search algorithm[12]. As shown in

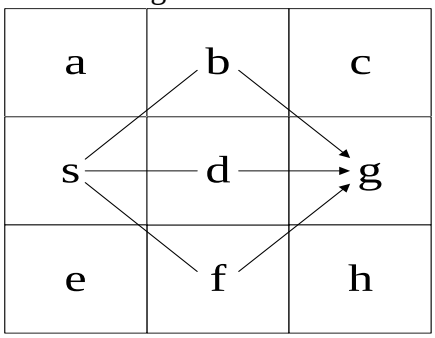

Figure 1 Node expansion process in search, the *g* point can be directly searched from the *s* point, without passing through the nodes *b* or *f*. The intermediate nodes on the path formed by the jump points (like *s* and *g*) are not expanded, which also greatly reduces the amount of calculation and storage during the expansion process and speeds up the search.

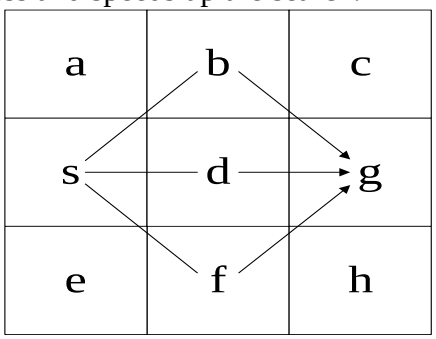

Figure 1 Node expansion process in search

3.2 Bidirectional Search

The traditional A* algorithm uses a unidirectional search from the start to the goal, which is inefficient in large maps with numerous obstacles[13]. To address this limitation, integrated the bidirectional search strategy with the traditional A* algorithm, resulting in the Bidirectional A* (BA*) algorithm. Then, searches are initiated simultaneously from both the start and goal points, with each treating the other as the target, until the paths meet in the middle. Then, the complete path is generated by backtracking from the meeting point to the start and goal. Specific Steps Are as Follows:

(1)Initialization

Add the start node to the OPEN1 list for forward search and the goal node to the OPEN2 list for backward search. Initialize the corresponding CLOSE1 and CLOSE2 lists as empty, which will store nodes that have already been explored in each direction.

(2)Iterative Search

Select the nodes with the lowest total cost, Node1 from OPEN1 and Node2 from OPEN2, remove them from their respective OPEN lists, and add them to CLOSE1 and CLOSE2. For Node1, examine its neighboring nodes: if a neighbor is already present in CLOSE1 or marked as an obstacle, it is ignored. If the neighbor is not found in OPEN1, set Node1 as its parent node, calculate its total cost *F*, and then add it to OPEN1. If the neighbor is already in OPEN1, compare the new path's actual cost with the previously recorded one; if the new cost is lower, update the neighbor's total cost *F*, actual cost, and parent node information accordingly. Similarly, perform the same operations for Node2's neighbors, updating the CLOSE2 and OPEN2 lists.

(3)Termination Condition

The search terminates when a common node appears in both the CLOSE1 and CLOSE2 lists, indicating that a path has been found. The complete path can be constructed by backtracking through the parent pointers from the intersecting node to both the start and goal nodes. If either the OPEN1 or OPEN2 list becomes empty, or no common node exists between the CLOSE1 and CLOSE2 lists, it is determined that no valid path exists, and the search fails.

By employing a bidirectional search strategy, the algorithm significantly reduces the search space on large-scale maps, thereby markedly improving the efficiency of path planning. This approach not only accelerates the search process but also ensures optimal path discovery while minimizing computational resource consumption.

3.3 Direction Penalty Factor

In the A* algorithm, the core of path planning is to select the optimal path through heuristic function *h(n)* and cost function *g(n)*. To improve the A* algorithm and introduce a directional penalty factor, consider the impact of direction changes on the total cost. Specifically, when the path changes direction, an additional penalty value is added to adjust the priority of path selection.

Let the direction vector from node A to node B be denoted as $\vec{v}_{AB}$,The direction vector from node B to node C can be denoted as$\vec{v}_{BC}$.By calculating the angle θ between these two vectors, the direction penalty factor can be determined. A larger angle indicates a more significant change in direction, and consequently, a higher penalty should be applied. The improved algorithm incorporates the direction penalty factor into the cost function *g(n)* as follows:

$$f(n) = h(n) + g'(n) \tag{4}$$

Where,

$$g'(n) = g(n) + DirectionPenalty\ (\theta) \tag{5}$$

First, calculate the angle θ between the two direction vectors:

$$cos\ (\theta) = \frac{\vec{v}_{AB} \cdot \vec{v}_{BC}}{\|\vec{v}_{AB}\| \cdot \|\vec{v}_{BC}\|} \tag{6}$$

Where, the penalty function is defined based on the magnitude of the angle θ. A common linear penalty function can be expressed as:

$$DirectionPenalty\ (\theta) = k \cdot \theta \tag{7}$$

In the penalty function, there is a penalty coefficient, denoted here as *k*, where is used to adjust the sensitivity to changes in direction. This coefficient plays a crucial role in determining how heavily directional changes are penalized during the pathfinding process.

3.4 Pathfinding Process of the Improved A* Algorithm

The improved A* algorithm integrates Jump Point Search (JPS), bidirectional search, and a direction penalty factor to filter out unnecessary nodes during the pathfinding process. The remaining nodes are those that satisfy specific screening criteria. The flowchart of the improved A* algorithm is shown in Figure 2, with the detailed steps outlined below:

Step 1:Initialization:Initialize the forward search queue OPEN1 and the backward search queue OPEN2, along with their corresponding CLOSE1 and CLOSE2 lists.Add the start node to OPEN1 and the goal node to OPEN2.Calculate the initial direction vectors for both directions.Set the direction penalty coefficient *k* and the jump point expansion threshold.

Step 2: Bidirectional Jump Point Expansion:From OPEN1 and OPEN2, select the nodes Node1

and Node2 with the smallest total cost *F*. Prioritize processing the queue with the smaller *F*-value.Apply the JPS strategy to expand along the parent node's direction:If the current node has a forced neighbor (e.g.,an obstacle blocking the natural path), mark it as a jump point and add it to the corresponding OPEN list.Otherwise, move directly to the next jump point along the straight-line direction (e.g.,node *g* in

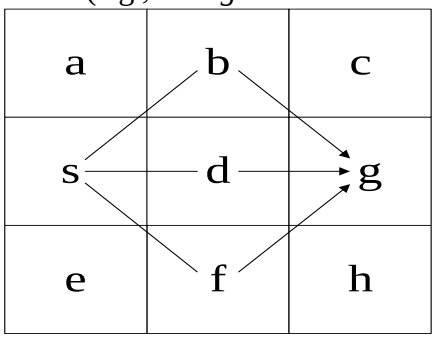

Figure 1).For each expanded jump point, calculate the penalty value Δp caused by the change in path direction using equations (4)-(7).Update the actual cost *g(n)* as:

$$g(n)=g(parent)+step\_cost+\Delta p \quad (8)$$

Step 3:Bidirectional Search Coordination:Add the current jump point Node1 to CLOSE1 and Node2 to CLOSE2.Check the neighboring nodes of Node1 and Node2:If a neighboring node has not been visited and is not an obstacle, update its parent node and cost, then insert it into the corresponding OPEN queue.If the neighboring node already exists in the OPEN queue, compare the new path cost and update it only if the new cost is lower.Continuously compare the nodes in CLOSE1 and CLOSE2 in real-time.If a common node is found, terminate the search and generate the complete path.If either OPEN1 or OPEN2 becomes empty or no common node is detected, conclude that no feasible path exists.

Step 4: Path Backtracking and Optimization:Starting from the intersection node identified by the bidirectional search, backtrack along the parent pointers to reconstruct the full path from the start node to the goal node.Remove redundant intermediate nodes (e.g.,nodes between JPS jump points that are directly reachable).Adjust the path angles by incorporating the direction penalty factor to minimize unnecessary turns and ensure smooth transitions.

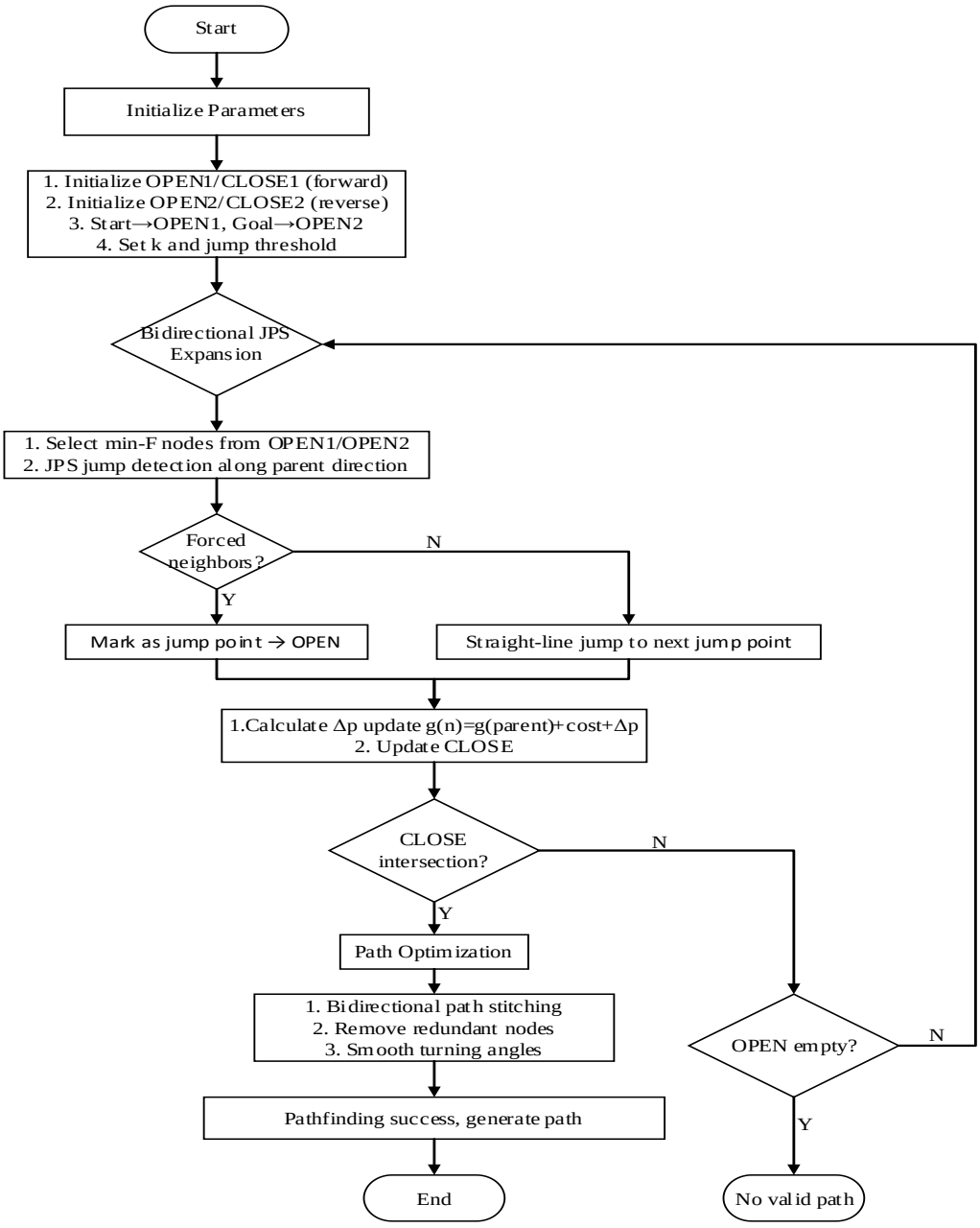

Figure 2 Improved A algorithm flow chart*

4. Simulation Experiments

4.1 Experimental Design

To validate the effectiveness of the improved A* algorithm, this paper conducts simulation experiments comparing the traditional A* algorithm with the enhanced A* algorithm that integrates bidirectional search and Jump Point Search (JPS). The experiments are divided into two parts:

1. MATLAB Grid Map Simulation: This part involves simulating grid maps in MATLAB to compare the traditional A* algorithm with the improved A* algorithm in terms of path length, the number of expanded nodes, and search time.

2. ROS Robot Simulation: This part evaluates the performance of the algorithms in a realistic

map environment using the Robot Operating System (ROS). The assessment focuses on path smoothness, obstacle avoidance success rate, and execution efficiency in real-world scenarios.

4.2 MATLAB Grid Map Simulation

The experimental platform was MATLAB R2022b, and experiments were conducted on four grid maps of different sizes. The obstacle density of the grid maps used in the experiments was uniformly set at 40%, with a direction penalty factor $k=0.5$. Figure 3 shows the simulation experiment on a 40×40 grid map, where black cells represent obstacles, the black circle indicates the starting node, and the yellow circle represents the goal node. Figure 4 and Figure 5 present the comparative experimental results of the traditional A* algorithm and the improved A* algorithm proposed in this paper. Green cells indicate nodes visited (i.e., added to the operation list) during the search process by the pathfinding algorithms, while the red polyline represents the final generated path. It can be observed that both pathfinding algorithms successfully generated paths in the same environment, and the number of nodes operated on during the search process by the improved A* algorithm was significantly fewer than that of the traditional A* algorithm. Simulation results for other map environments are shown in Table 1.

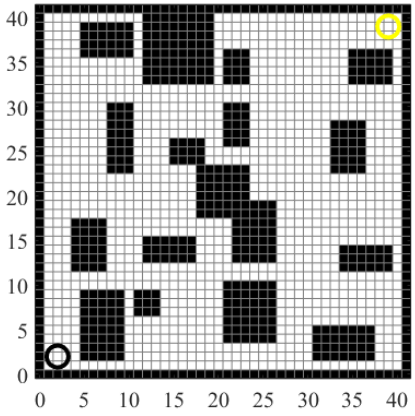

Figure 3 Simulation Experiment Map

According to Table 1, on the basis of ensuring the same path quality, the improved A* algorithm reduces the number of expanded nodes by an order of magnitude during the pathfinding process. The traditional A* algorithm involves visiting and evaluating each neighboring node, followed by sorting based on heuristic values. In contrast, the improved A* algorithm operates on a significantly smaller number of jump points, thereby greatly reducing computational complexity and minimizing memory consumption during the pathfinding process. The experimental results demonstrate that the improved A* algorithm not only effectively enhances the search speed of the traditional A* algorithm but also drastically decreases the number of expanded nodes during the search process. The improved A* algorithm reduces the search time by approximately 48%, decreases the number of node expansions by about 93%, and shortens the path length by roughly 13% compared to the traditional A* algorithm.

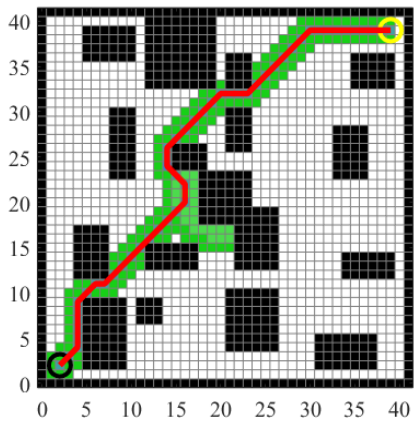

Figure 4 Traditional A algorithm*

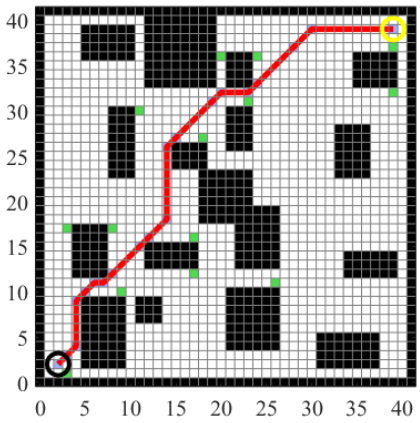

Figure 5 Improved A algorithm*

Table 1 MATLAB raster map simulation experiment comparison

Map size	Search time/s		Number of extension nodes		Path length	
	Traditional - A*	Improved - A*	Traditional - A*	Improved - A*	Traditional - A*	Improved - A*
40*40	1.9022	1.1067	245	14	61	57
50*50	2.1644	1.3156	347	23	74	64
60*60	3.7641	1.8064	689	44	102	87
80*80	8.4217	3.4648	968	72	124	102

4.3 ROS Robot Simulation

The experiment was based on the Ubuntu 20.04 operating system, using ROS Noetic as the development platform, and combined with Gazebo 11 for physical simulation and Rviz for path visualization. The robot model was an Ackermann chassis four-wheeled robot, equipped with a Hokuyo LiDAR, with a scanning frequency set to 10Hz for real-time environmental perception. The scene design is shown in Figure 6 as a static indoor maze environment, and Figure 7 shows the map constructed using the Gmapping SLAM[14]. The above configuration ensured the diversity of the simulation experiments and a high degree of fidelity to actual application scenarios.

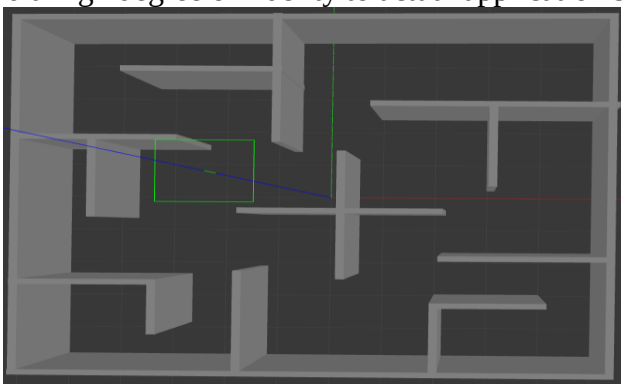

Figure 6 Gazebo simulation environment

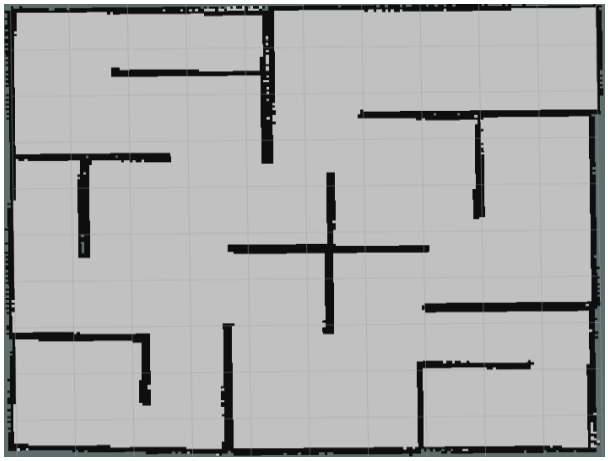

Figure 7 Environmental maps built by Gmapping SLAM

The pathfinding results of the traditional A* algorithm and the improved A* algorithm are shown in Figure 8 and Figure 9 , respectively. The maps display the 2D map and path planning results visualized by Rviz, a visualization tool of ROS. In the figures, the light gray areas represent the 2D form of the constructed map in the simulation environment. For both sets of experiments, identical start and end points were set, with the green lines indicating the generated planned paths.

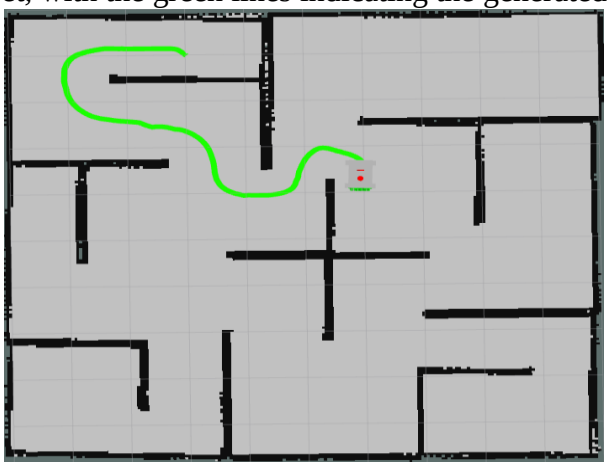

Figure 8 Traditional A algorithm*

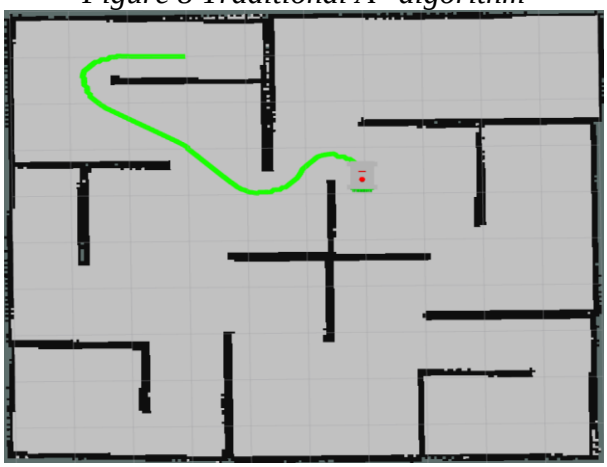

Figure 9 Improved A algorithm*

Table 2 lists the path lengths and search times of the two algorithms for comparison. This paper concludes that the improved A* algorithm can essentially generate optimal paths. Meanwhile, in

terms of search speed and path length, the improved A* algorithm significantly outperforms the traditional A* algorithm. The experimental results demonstrate that the improved A* algorithm, which incorporates bidirectional jump point search, effectively accomplishes path planning for mobile robots. Compared to the traditional A* pathfinding algorithm, the improved A* algorithm optimizes the search strategy, thereby achieving faster computational speed and higher pathfinding efficiency.The improved A* algorithm reduces the search time by approximately 52% and shortens the path length by about 13% compared to the traditional A* algorithm.

Table 2 Comparison of search results

Algorithms	Trajectory length/cm	Time spent/s
Traditional A*	30.68	2
Improved A*	26.74	1.27

5. Conclusions

The improved A* algorithm proposed in this paper transforms the traditional unidirectional search strategy into a bidirectional search, and incorporates a jump point search mechanism to reduce redundant node expansions. Additionally, it introduces a direction penalty factor to optimize path smoothness. Experimental results show that this algorithm achieves significant optimization in terms of the number of node expansions, search time, and path length, particularly demonstrating higher search efficiency and better path quality in complex environments. These improvements make the algorithm more suitable for real-time application scenarios requiring efficient path planning. Compared with the traditional A* algorithm, the improved algorithm demonstrates its practicality and superiority, providing a new solution for path planning in complex environments.

Acknowledgements

This paper is supported by Projects of major scientific and technological research of Ningbo City (2021Z059, 2022Z090(2022Z050), 2023Z050(the second batch)), Projects of major scientific and technological research of Beilun District, Ningbo City(2021BLG002, 2022G009), Projects of scientific and technological research of colleges student's of China(202313022036, 202413001008).

Reference

[1] Tang Y, Zakaria MA, Younas M. Path planning trends for autonomous mobile robot navigation[J]. Sensors, 2025, 25(4): 1206.
Zhou P, Xie Z, Zhou W, Tan Z. A heuristic integrated scheduling algorithm based on improved Dijkstra algorithm[J]. Electronics, 2023, 12(23): 4189.
[2] Liu L, Wang B, Xu H. Research on path-planning algorithm integrating optimization A-star algorithm and artificial potential field method[J]. Electronics, 2022, 11(22): 3660.
[3] Alfaro-Cid E, McGookin EW, Murray-Smith DJ. Optimisation of the weighting functions of an H∞ controller using genetic algorithms and structured genetic algorithms[J]. International Journal of Systems Science, 2008, 39(4): 335-347.
[4] Fakhouri HN, Hudaib A, Sleit A. Multivector particle swarm optimization algorithm[J]. Soft Computing, 2020, 24(15): 11695-11713.
[5] Zhao Q, Liu H, Zhang Y, Wang J. Path planning fusion algorithm based on improved A-star and adaptive dynamic window approach for mobile robot[J]. International Journal of Industrial

Engineering: Theory, Applications and Practice, 2023, 30(5): 1-15.

[6] Liu C, Mao Q, Chu X, Xie S. An improved A-star algorithm considering water current, traffic separation and berthing for vessel path planning[J]. Applied Sciences, 2019, 9(6): 1057.

[7] Mi Z, Xiao H, Huang C. Path planning of indoor mobile robot based on improved A algorithm incorporating RRT and JPS[J]. AIP Advances, 2023, 13(4): 045313.*

[8] Tang C, Claramunt C, Hu X, Zhou P. Geometric A-star algorithm: An improved A-star algorithm for AGV path planning in a port environment[J]. IEEE Access, 2021, 9: 59196-59210.

[9] Bu X, Li G, Tong B, Zhang X. A robot navigation system based on improved A-star algorithm[J]. International Journal of Pattern Recognition and Artificial Intelligence, 2024, 38(16): 2456012.

[10] Huang J, Chen C, Shen J, Liu G, Xu F. A self-adaptive neighborhood search A-star algorithm for mobile robots global path planning[J]. Computers and Electrical Engineering, 2025, 123(Part A): 110018.

[11] Li X, Hu X, Wang Z, Du Z. Path planning based on combination of improved A-star algorithm and DWA algorithm[C]. 2020 2nd International Conference on Artificial Intelligence and Advanced Manufacture (AIAM), 2020: 99-103.

[12] Liu Y, Zhang H, Wang L. Global dynamic path planning fusion algorithm combining Jump-A algorithm and dynamic window approach[J]. IEEE Access, 2021, 9: 19632-19638.*

[13] Pavlik JA, Sewell EC, Jacobson SH. Two new bidirectional search algorithms[J]. Computational Optimization and Applications, 2021, 80(2): 377-409.

[14] Zhao J, Liu S, Li J. Research and implementation of autonomous navigation for mobile robots based on SLAM algorithm under ROS[J]. Sensors, 2022, 22(11): 4172.