applied sciences

MDPI

Article

Research on Path Planning for Robots with Improved A* Algorithm under Bidirectional JPS Strategy

Fujie Wang, Wei Sun, Pengfei Yan, Hongmei Wei and Huishan Lu *

School of Mechanical Engineering, North University of China, Taiyuan 030051, China; wangfujie@nuc.edu.cn (F.W.); 18235908468@163.com (W.S.); ypf5321@163.com (P.Y.); weihmchina@163.com (H.W.)
* Correspondence: 13934597379@139.com

Abstract: Aiming to address the A* algorithm's issues of traversing a large number of nodes, long search times, and large turning angles in path planning, a strategy for multiple improvements to the A* algorithm is proposed. Firstly, the calculation of the heuristic function is refined by utilizing the Octile distance instead of traditional distance, which more accurately predicts the optimal path length. Additionally, environmental constraints are introduced to adaptively adjust the weight of the heuristic function, balancing the trade-off between search speed and path length. Secondly, the bidirectional jump point search method is integrated, allowing simultaneous path searches from both directions. This significantly reduces path search times and the number of nodes traversed. Finally, the path undergoes two rounds of smoothing using a path smoothing strategy until the final path is generated. To validate the effectiveness of the improved A* algorithm, simulations are conducted on ten types of grid maps. Results demonstrate that the improved A* algorithm markedly decreases path search times while maintaining path length, with greater speed improvements observed as the map size increases. Furthermore, the improved algorithm is applied in experiments with mobile robots, achieving significant reductions in average path search times of 79.04% and 37.41% compared to the traditional A* algorithm and the JPS algorithm, respectively. This enhancement effectively meets the requirements for rapid path planning in mobile robotics applications.

Keywords: robot; path planning; A* algorithm; heuristic function; jump point search

check for updates

Citation: Wang, F.; Sun, W.; Yan, P.; Wei, H.; Lu, H. Research on Path Planning for Robots with Improved A* Algorithm under Bidirectional JPS Strategy. *Appl. Sci.* **2024**, *14*, 5622. https://doi.org/10.3390/app14135622

Academic Editor: Jonghoek Kim

Received: 18 May 2024
Revised: 20 June 2024
Accepted: 24 June 2024
Published: 27 June 2024

Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

In today's society, with the continuous development of intelligent manufacturing technology, robots have received widespread attention and have permeated various aspects of human life. Currently, research on robot autonomous navigation is primarily divided into three parts: perception, planning, and execution. Among these, path planning, referred to as "planning", serves as the bridge connecting the other two components and is also the core focus of robotics research [1,2]. The importance of path planning can be reflected in the framework diagram of navigation, as shown in Figure 1. The navigation framework is mainly composed of the following key components. The 'Move_base' node is the core of the navigation framework, being responsible for coordinating and managing other modules. It receives the target position, calls Global_Planner and Local_Planner to generate a path, and sends control commands to Base_controller. The 'Global_Planner' node calculates the optimal path from the start point to the target point based on the static map and the target position. The 'Local_Planner' node is responsible for generating local paths suitable for the current environment during the execution of the global path, performing real-time obstacle avoidance and path adjustment. The 'Base_controller' node transforms high-level path planning instructions into actual motion control signals to ensure that the robot can navigate accurately and safely to the target position. The 'Costmap' node consists of the Global_costmap node and Local_costmap node, representing the traversal costs of different locations in the environment. Global_costmap is based on a static map

while Local_costmap is based on real-time sensor data. The 'Recovery_behaviors' node is used to handle faults in various modules during navigation. Recovery behaviors are activated when there are failures in global planning, local planning, or oscillations. The 'Amcl' node is responsible for robot localization based on sensor data and the map. The 'Map_server' node provides the static environment map, usually generated by SLAM. This map is used for path planning and localization. The 'Odometry source' node provides the robot's position information and motion state in the environment, estimating the robot's posture and position changes relative to its starting position. The 'Sensor sources' node collects data about the robot's surroundings using sources such as LiDAR, depth cameras, and ultrasonic sensors. These data are used to build and update the costmap. The 'Sensor transforms' node converts the data collected by the sensors into a unified coordinate system or reference frame for consistent processing and decision making. Additionally, by using communication mechanisms such as topic communication with topics like "/map", the nodes work collaboratively. The navigation framework achieves key autonomous navigation functions through the coordinated work of Global_Planner and Local_Planner. Path planning specifically means that the robot plans an optimal path from the start position to the goal position without any collision with obstacles in the space when the environment is known or unknown. Path planning according to the degree of mastery of the environment information is divided into two forms. One is global path planning with full knowledge of the environment; at present, the commonly used global path planning algorithms are the Dijkstra algorithm [3], A* algorithm [4–8], ACO algorithm [9], etc. The other is local path planning with only partial or no knowledge of the environment; local path planning algorithms mainly include the DWA algorithm [10,11], artificial potential field method [12], etc.

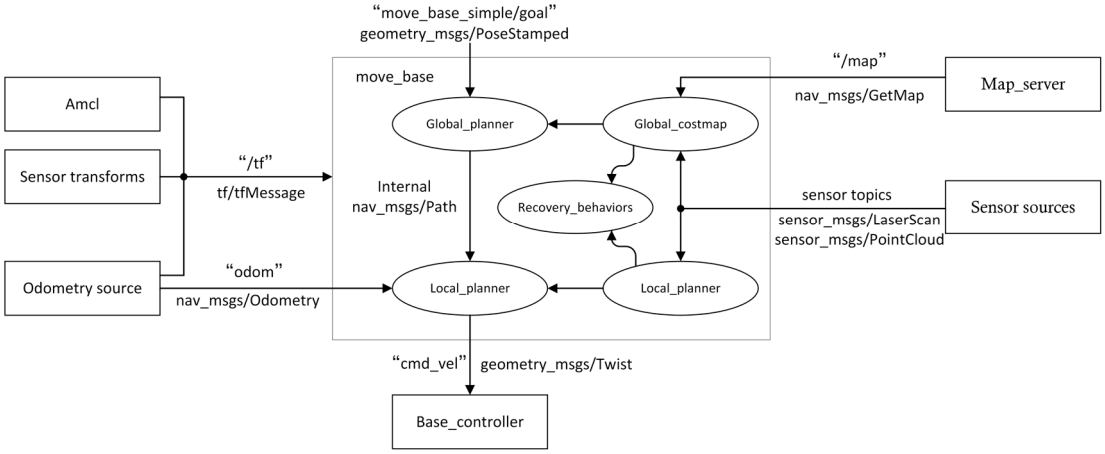


Figure 1. Framework diagram of navigation.

The A* algorithm is a path planning algorithm based on graph search that is widely used in global path planning due to its fast computation speed, high efficiency, and maturity, being the most established path planning algorithm known so far. However, the A* algorithm still has many shortcomings. For example, it constantly visits the surrounding eight neighboring nodes during pathfinding, adding them to OpenList; selects the node with the smallest cost as the current node for expansion after calculating its evaluation function value; and adds it to CloseList after expansion, resulting in a large number of redundant nodes being stored and accessed. When the map scene is large, it will lead to the wastage of both storage space as well as time due to the increase in computation, resulting in inefficient pathfinding. Therefore, many scholars have made a series of improvements to the A* algorithm through research based on these problems. Zhang Mo et al. [13] reduced the number of turns in diagonally planning paths by changing the equation for total cost, i.e., increasing the influence of the predicted cost on the total cost, and smoothed the paths by introducing the five-point cubic smoothing method, but the algorithm still

suffers from the problem of having too many redundant nodes. Gong Yunxin et al. [14] used an extension to include convex corner points that satisfy the adjacency relationship as the pattern of adjacency extension for the A* algorithm. This modification reduces access to unnecessary nodes and effectively improves the algorithm's search efficiency. However, the improved A* algorithm's efficiency is still not high in scenarios with too many convex corner points, and it fails to meet real-time demands. Harabor et al. [15] proposed a jump point search algorithm that removes non-essential nodes through screening and pruning rules, enables jumping between nodes with variable step sizes, and improves the pathfinding speed. However, the path planned by this method still suffers from the problem that the turning angles are not smooth and cannot satisfy the robot's motion characteristics. In Zhao Xiao et al.'s [16] method, which combines the A* algorithm with the JPS algorithm to improve the A* algorithm, a series of representative nodes are screened for expansion by preprocessing the nodes, which not only saves memory but also enhances the algorithm's efficiency. In this paper, we analyze and study the existing improved A* algorithms and a new strategy for the improvement of the A* algorithm is proposed. It involves the following steps:

1. Improve the heuristic function of the A* algorithm: firstly, utilize the Octile distance formula instead of the commonly used distance formula to approximate the predicted cost of the heuristic function to the actual path length. Afterward, environmental constraints, i.e., the percentage of obstacles, are introduced to make the algorithm adaptive to the changes of the environmental map;
2. Incorporating a bidirectional JPS search strategy: the searched jump points are added to OpenList for access and expansion. This replaces the need to access and compute a large number of neighboring nodes in the A* algorithm, thereby reducing memory consumption and significantly accelerating the path search speed through a bidirectional search strategy;
3. Further optimize the generated paths: first reduce the number of redundant nodes on the paths, then eliminate unnecessary inflection points on the paths, and, finally, smooth the paths further using the dynamic circular cutting method to improve the turning angles of the paths.

2. Traditional A* Algorithm

2.1. Modeling of Grid Maps

Establishing a map model of the environment is the primary prerequisite for path planning, and commonly used methods for map modeling include the grid method, visual graphic method, topological map method, Voronoi diagram, and others [17]. The grid method is widely adopted for its intuitive representation, simple operation, and ease of implementation. In this paper, 2D grids of uniform size are used to abstractly represent information about the actual environment, with each grid corresponding to a unique coordinate. Each grid is classified into two states based on whether it is occupied by an obstacle [18]. One is the free state, represented by white grids in Figure 2, indicating areas where the robot can freely pass; the other is the occupied state, represented by black grids in Figure 2, indicating areas impassable for the robot.

Commonly used neighborhood expansion patterns in A* algorithms include four-neighborhood expansion and eight-neighborhood expansion [19]. In actual robot operations, the model's complexity is increased due to uncertainties in the robot's operational state, which affect its movement direction. To mitigate this complexity, this paper selects an eight-neighborhood expansion pattern when there are no boundaries or obstacles around the robot, as depicted in Figure 3.

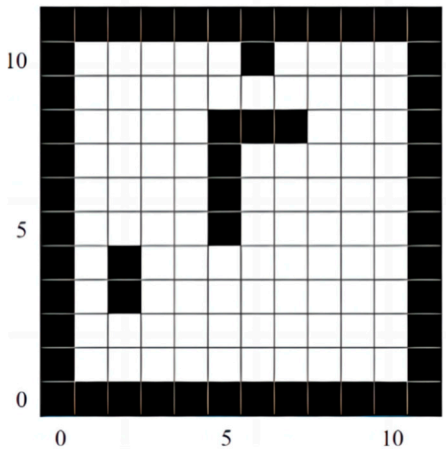


Figure 2. Grid map.

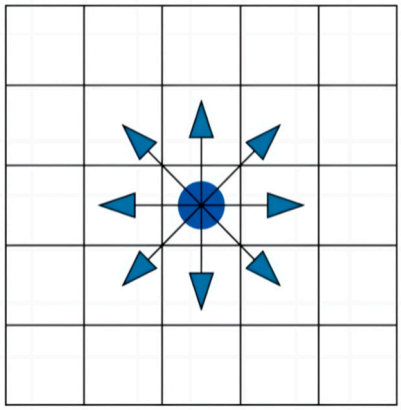


Figure 3. Eight-neighborhood expansion.

2.2. Traditional A* Algorithm

The A* algorithm is a heuristic path planning algorithm designed for static scenarios that introduces a Best First Search algorithm, i.e., heuristic function, based on the Dijkstra algorithm, which enhances the heuristic of the algorithm, makes the algorithm more purposeful in searching towards the goal point, and reduces the algorithm's traversal of the redundant nodes. The evaluation function of the A* algorithm is given in the following equation:

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

In Equation (1), $g(n)$ denotes the total actual cost accumulated from the start node to the current node n; $h(n)$ is a heuristic function representing the predicted cost from the current node n to the target node; and $f(n)$ is an evaluation function representing the total predicted cost for the start node to reach the target node via node n.

The heuristic function $h(n)$ is crucial in determining the efficiency of the algorithm. When $h(n)$ is smaller than the actual cost $h^*(n)$, the algorithm can definitely plan the optimal path, but it may traverse more nodes, slowing down the search speed; when $h(n)$ is 0, $h(n)$ must be less than or equal to $h^*(n)$, i.e., when the algorithm degenerates into the Dijkstra algorithm; when $h(n)$ is greater than the actual cost $h^*(n)$, the algorithm reduces the number of traversed nodes, speeding up the search, but it will ensure that the path planning result is not the optimal solution; when $h(n)$ is much larger than $h^*(n)$, the algorithm evolves to the Best First Search algorithm; the A* algorithm needs to consider the search speed during the search process while ensuring that the optimal path is found. Therefore, for optimal efficiency, $h(n)$ should closely approximate $h^*(n)$.

Commonly used representations of heuristic functions are Euclidean distance and Manhattan distance. Let the coordinates of the current node be (x_n,y_n) and the coordinates of the target point be (x_g,y_g); then, the Euclidean distance formula is as below:

$$h(n) = \sqrt{(x_g - x_n)^2 + (y_g - y_n)^2} \tag{2}$$

The Manhattan distance formula is as follows:

$$h(n) = |x_g - x_n| + |y_g - y_n| \tag{3}$$

Due to the presence of obstacles in the actual environment and the selected expansion method in this paper being eight-neighborhood expansion, when using a Euclidean distance of $h(n)$, the distance value must be smaller than the actual cost $h^*(n)$; when using a Manhattan distance of $h(n)$, the distance value is larger than the actual cost $h^*(n)$. In this paper, we need to select an appropriate distance formula that makes $h(n)$ closer to $h^*(n)$.

The A* algorithm will add the node with the smallest value of the current evaluation function as the current node to CloseList during the path search and, at the same time, visit the neighboring nodes reachable by this node and add them to OpenList, then calculate the node with the smallest value of the evaluation function in OpenList to add it to CloseList. It will repeat this process until the target point becomes the current node, completing the pathfinding. Finally, the final path is generated by backtracking the parent–child relationships of the nodes. The pathfinding process is illustrated in Figure 4, which highlights that the A* algorithm generates a large number of redundant nodes, represented by the light blue grids in the figure. Maintaining and accessing these nodes increases the computational workload of the algorithm and wastes storage space, thereby impacting the efficiency of path planning.

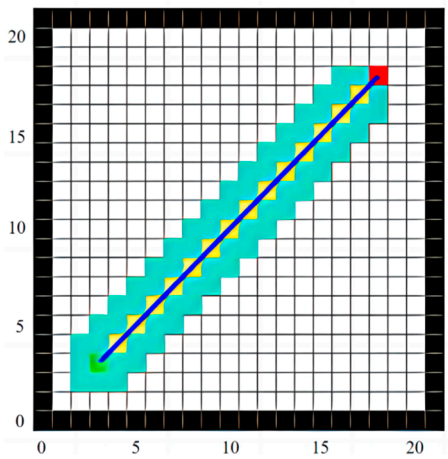


Figure 4. Pathfinding process of A* algorithm.

3. Improved A* Algorithm

3.1. Improvement of the Heuristic Function

3.1.1. Selection of Heuristic Function

The heuristic function, as the core of the A* algorithm, significantly impacts the result of the planned path. In response to the issues with the general heuristic function's expression mentioned above, this paper proposes a new distance expression for the heuristic function, namely Octile distance. Octile distance is a combination of Euclidean distance and Manhattan distance that brings the predicted distance closer to the actual distance, thus improving the efficiency of the algorithmic search. Octile distance is illustrated in Figure 5, where the yellow grid indicates the starting point, the purple grid indicates the target point, the gray grid indicates the actual path, the blue straight line indicates the Octile distance,

the red straight line indicates the Euclidean distance, and the green straight line indicates the Manhattan distance. From the figure, it can be seen that Euclidean distance is suitable for multi-neighborhood expansion, Manhattan distance is suitable for four-neighborhood expansion, and Octile distance is more suitable for the eight-neighborhood expansion proposed in this paper.

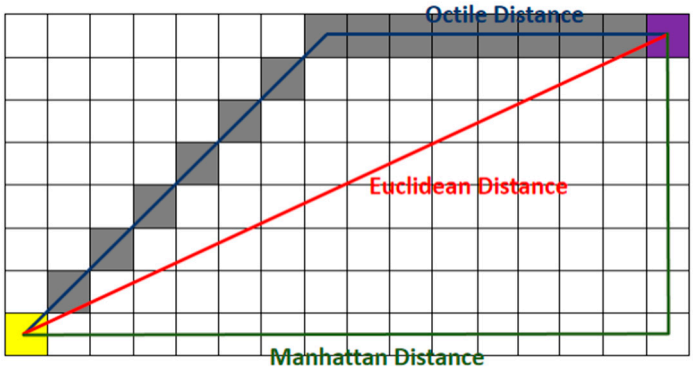


Figure 5. Octile distance.

The Octile distance formula is as below:

$$h(n) = (\left|x_g - x_n\right| + \left|y_g - y_n\right|) + (\sqrt{2} - 2)\min(\left|x_g - x_n\right|, \left|y_g - y_n\right|) \quad (4)$$

In Equation (4), (x_n,y_n) are the coordinates of the current node and (x_g,y_g) are the coordinates of the target point.

To verify the effect of the heuristic function of the A* algorithm on pathfinding efficiency, a simulation test was performed on a 20 × 20 grid map, as shown in Figure 6. Figure 6a represents the result of using Octile distance as the heuristic function, Figure 6b represents the result of using Euclidean distance as the heuristic function, and Figure 6c represents the result of using Manhattan distance as the heuristic function. By comparing the three graphs, it can be seen that the paths obtained in Figure 6a,b are equal in length and shorter than the paths obtained in Figure 6c while the number of nodes traversed in Figure 6a is less than the number of nodes traversed in Figure 6b. In summary, it can be concluded that Octile distance as a heuristic function is more efficient than the traditional distance formulas when choosing the eight-neighborhood expansion for path planning.

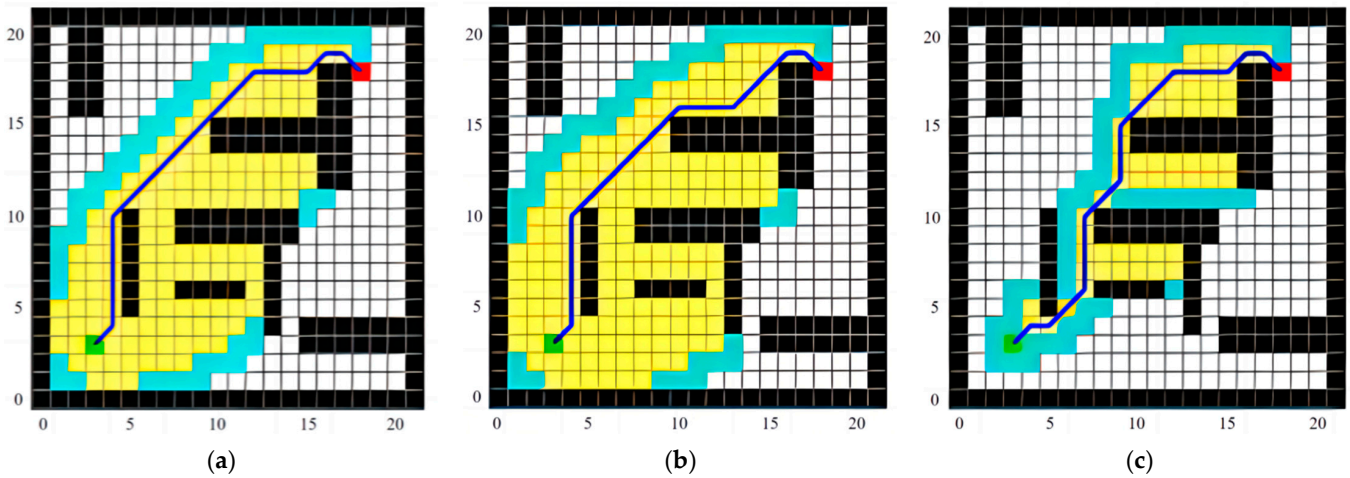


Figure 6. Simulation comparative testing of heuristic functions: (**a**) Octile distance; (**b**) Euclidean distance; (**c**) Manhattan distance.

3.1.2. Introduction of Environmental Constraints

Environmental constraints, i.e., the percentage of obstacles, are also critical factors that affect the efficiency of path planning. When the percentage of obstacles in the environment is high, the planned path is prone to local optimization issues, leading to path search failures. Conversely, when the percentage of obstacles is low, the search speed may decrease due to the broader path search range. To address these issues, this paper introduces an obstacle percentage coefficient O_s to adaptively adjust the heuristic function. Additionally, to ensure data smoothness, the obstacle percentage coefficient O_s is first logarithmically transformed and then taken in its absolute value to ensure non-negativity. The improved heuristic function formula is as follows:

$$h\prime(n) = \alpha h(n) \tag{5}$$

$$\alpha = |\ln O_s| \tag{6}$$

$$O_s = \frac{N}{(|x_g - x_s|+1) \times (|y_g - y_s|+1)} \tag{7}$$

Here, N is the number of obstacle grids between the starting point and the endpoint and (x_s, y_s) and (x_g, y_g) are the coordinates of the starting point and the endpoint, respectively.

From the improved heuristic function, it is evident that as the percentage of obstacles increases (i.e., the occupancy factor of obstacles becomes larger), α decreases. A smaller α indicates a reduced weight of the heuristic function in the evaluation function. At this point, the A* algorithm tends to behave more like the Dijkstra algorithm, expanding its search range and thus avoiding local optimization problems. Conversely, when the percentage of obstacles decreases (i.e., there is a smaller occupancy factor of obstacles), α increases. A larger α indicates a greater weight of the heuristic function in the evaluation function. In this scenario, the A* algorithm leans towards behaving more like the BFS algorithm, narrowing its search range and resulting in a faster search.

The test results of the algorithm before and after the introduction of environmental constraints on a grid map of 20 × 20 are shown in Figure 7. Figure 7a shows the test results before introducing environmental constraints and Figure 7b shows the results after their introduction. From the figures, it can be observed that the nodes expanded by the algorithm increase after the introduction of environmental constraints, resulting in a larger search range. However, the final path length obtained is shorter than that achieved without environmental constraints.

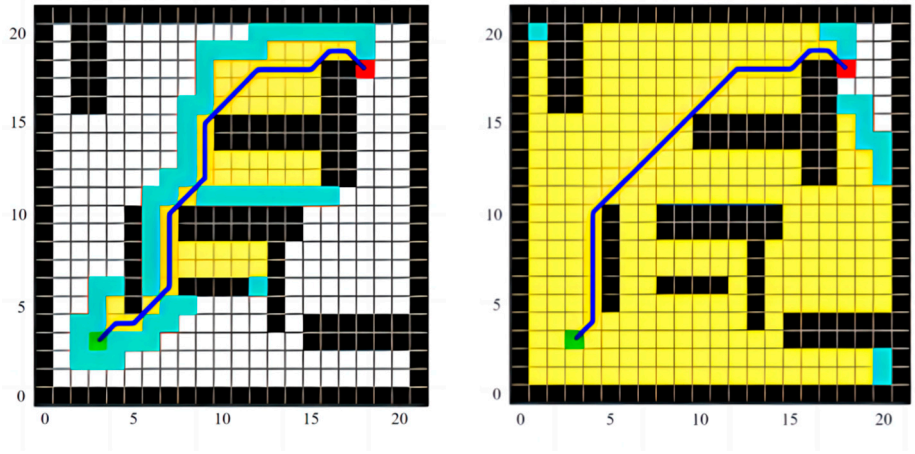


Figure 7. Simulation comparative testing: (**a**) without environmental constraints; (**b**) with environmental constraints.

3.2. Combining the Bidirectional JPS Algorithm

3.2.1. Traditional JPS Algorithm

Jump Point Search (JPS) is a more efficient path planning algorithm that screens natural neighbor nodes and forced neighbor nodes through specific screening rules and further screens to obtain the jump points needed for path planning, which well solves the problem of the A* algorithm accessing more redundant nodes.

Natural neighbor nodes are nodes that extend directly along the current search direction in the hopping point search algorithm; using these nodes can efficiently skip intermediate invalid nodes, reduce the search space, and improve the speed of path planning.

Forced neighbor nodes are the nodes that must be considered in the jump point search algorithm in order to bypass obstacles. By checking these nodes, JPS can effectively avoid obstacles in path search and ensure that it remains the optimal path.

The screening rules are as follows.

(1) No obstacles around the node

As shown in Figure 8, node x represents the current node, node $P(x)$ is the parent of node x, and n is a neighboring node of x. The length of the path taken by the parent node $P(x)$ to reach node n through x, when moving horizontally, must be shorter than the lengths of other paths that do not pass through x to reach node n. Similarly, the length of the path taken by the parent node $P(x)$ to reach node n through x, when moving diagonally, must be less than or equal to the lengths of other paths that do not pass through x to reach node n.

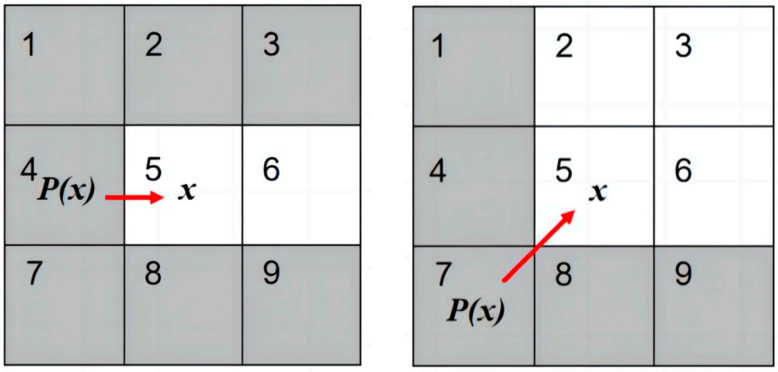


Figure 8. Screening rules without adjacent obstacles: (**a**) horizontal move; (**b**) diagonal move.

Therefore, the following screening rules for natural neighbor nodes can be derived:

- When moving horizontally, the natural neighbor node n satisfies the following equation:

$$L(< P(x), x, \ldots, n >) < L(< P(x), \ldots, n > | x) \tag{8}$$

Here, $L(<P(x), x, \ldots, n>)$ denotes the length of the path from the starting point $P(x)$ through x to reach n and $L(<P(x), \ldots, n> | x)$ denotes the length of the path from the starting point $P(x)$ to reach n without passing through x.

- When moving diagonally, the natural neighbor node n satisfies the following equation:

$$L(< P(x), x, \ldots, n >) \leq L(< P(x), \ldots, n > | x) \tag{9}$$

(2) Presence of obstacles around the node

As shown in Figure 9, the black grid indicates the obstacles. At this time, in addition to screening the natural neighbor nodes, it is also necessary to screen out the forced neighbor nodes. The forced neighbor nodes are shown in the green grid in the figure.

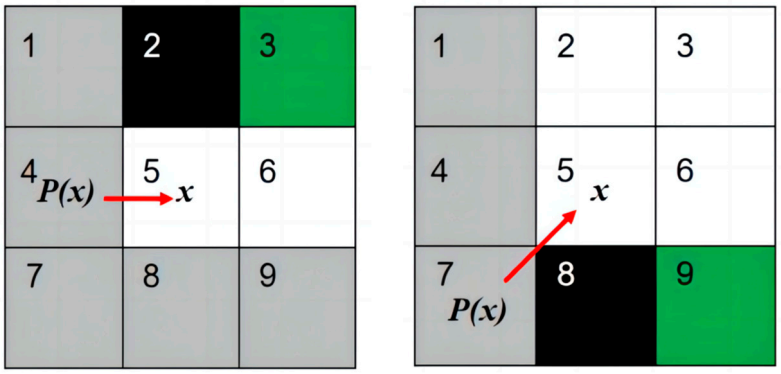


Figure 9. Screening rules with adjacent obstacles: (**a**) horizontal move; (**b**) diagonal move.

The screening rules for natural neighbor nodes are the same as the screening rules when there are no obstacles around a node. The screening rules for the forced neighbor nodes are as follows:

- n is not a natural neighbor of node x;
- n satisfies the following equation:

$$L(< P(x), x, \ldots, n >) < L(< P(x), \ldots, n > | x) \tag{10}$$

For the natural neighbor nodes and forced neighbor nodes obtained by the above screening rules, it is necessary to further screen them to obtain the jump point. The jump point screening rules are as follows:

- If node n is a starting or goal point, then n is a jump point;
- If node n has at least one forced neighbor, then n is a jump point;
- A node n is a jump point if the parent node $P(x)$ is in a diagonal direction from node n and n has nodes in the horizontal or vertical directions that satisfy conditions (1) and (2) above.

The purpose of the integration of the A* algorithm with the JPS algorithm is to first use the JPS algorithm to preprocess to obtain representative jump points, then add these jump points to OpenList and select the node with the smallest cost as the current node for expansion, then continue to screen its jump points to be added to OpenList. At the same time, the jump points after the expansion is completed are added to CloseList. This process is repeated until the current node is the target point, at which point the path planning is complete.

3.2.2. Bidirectional JPS Algorithm

The traditional JPS algorithm can better deal with the problems of the A* algorithm accessing more redundant nodes and having a high computation and memory consumption. However, in larger scenes or with more obstacles, a substantial number of jump points can still be generated, leading to slower pathfinding. Therefore, this paper proposes to improve the A* algorithm by using the bidirectional JPS algorithm on the basis of the traditional JPS algorithm.

Using the bidirectional JPS algorithm means that both the starting point and the goal point are used as the starting point and each takes the other as the goal point for path planning. Typically, the search from the starting point to the goal point is referred to as the forward search while the search from the goal point to the starting point is known as the reverse search. A successful path search is indicated when the currently expanded node of one party is the currently expanded node of the other party or exists in the CloseList of the other party.

The pathfinding process of the A* algorithm incorporating the bidirectional JPS algorithm is shown in Figure 10. Node s represents the start node, node g represents the target node, the black grids denote obstacles, and the red arrows and green arrows represent the paths for the forward and reverse searches, respectively. The flow of the algorithm is as follows:

- Create four lists, *OpenList1*, *OpenList2*, *CloseList1*, and *CloseList2*. Place node *s* into *OpenList1* and node *g* into *OpenList2*;
- Start the bidirectional search; during the forward search, place jump point *a* into *OpenList1*; during the reverse search, place jump point *b* into *OpenList2*. Place node *s* into *CloseList1* and node *g* into *CloseList2;*
- Continuing the forward search with *a* as the current node yields jump points *c* and *d*, which are added to *OpenList1*. In the reverse search, jump point *e* is found for *b* and added to *OpenList2*. Nodes *a* and *b* are placed into *CloseList1* and *CloseList2*, respectively;
- Since *d* is the node with the smallest value of *f(n)* in *OpenList1*, the forward search expands *d* to obtain jump points *e* and *f*, which are added to *OpenList1*. In the reverse search, jump point *h* is found for *e* and added to *OpenList2*. Nodes *d* and *e* are placed into *CloseList1* and *CloseList2*, respectively;
- Node *e* has the smallest value of *f(n)* in *OpenList1*, so *e* is selected as the current node for forward expansion. Since *e* exists in *CloseList2*, the path search is successful.

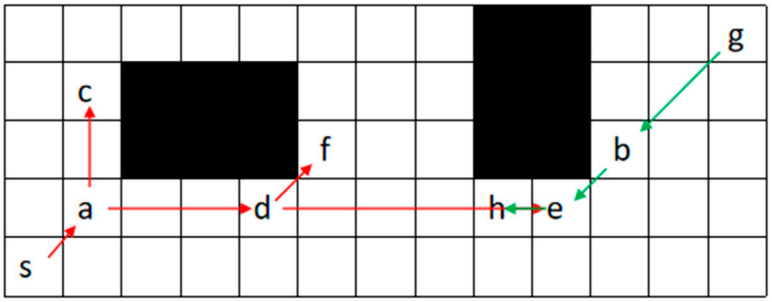


Figure 10. Schematic diagram of bidirectional JPS algorithm.

According to the process diagram of pathfinding using bidirectional JPS shown in Figure 10, theoretically, bidirectional JPS can double the efficiency compared to unidirectional JPS. However, bidirectional JPS also has drawbacks. When there is symmetry in the environment map, bidirectional JPS may plan two symmetric paths. This results in the paths searched in the forward direction not intersecting with the paths searched in the reverse direction. At this time, each search may independently plan a path, which not only fails to improve the efficiency of path planning but may even reduce it. To address this issue, this paper proposes an improved method where each search considers the other's current extended node as the target node.

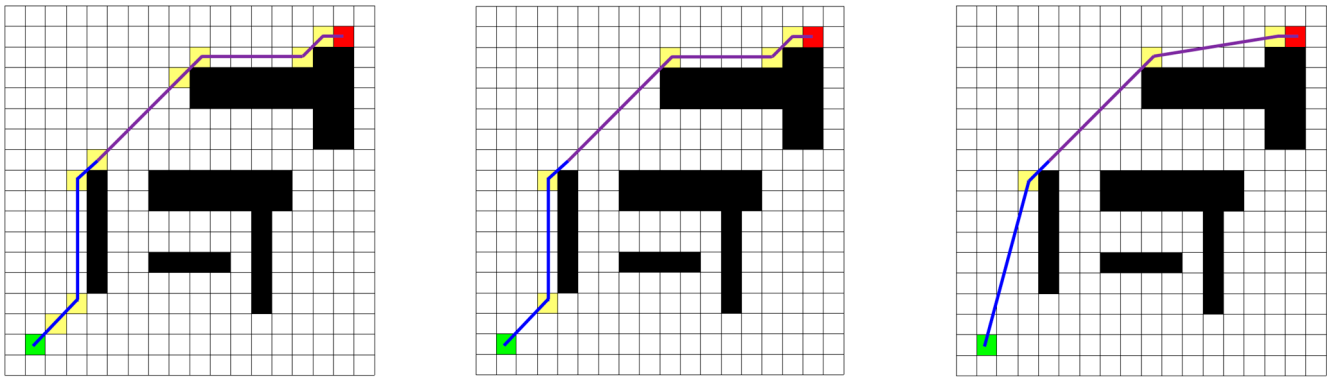
3.3. Path Smoothing

3.3.1. Primary Smoothing

The paths planned by the improved algorithm still exhibit issues such as having numerous redundant nodes and excessive turning points. Therefore, this paper employs a redundant point removal strategy to enhance path smoothness by reducing both the number of redundant nodes and excessive turning angles. The steps of the redundant point removal strategy, i.e., primary smoothing, are as follows:

- Plan the optimal path using the improved algorithm and save information such as the path length and number of nodes;
- Create a collection of path nodes, $R = \{N_i\}$, $i \in [0, n]$, where N_0 is the start node, N_n is the goal node, and $N_1, \ldots, N_{n-1}$ are the intermediate nodes;

- Determine whether the vectors $\overrightarrow{N_0N_1}$ and $\overrightarrow{N_1N_2}$ formed by points N_0, N_1, and N_2 are parallel using the principle of vector parallelism. If they are parallel, the three points are collinear, and the redundant node N_1 is removed. If they are not parallel, then the three points are not collinear, and there is no redundant node. If node N_1 is removed, then continue to evaluate nodes N_2, N_3, and N_4 to determine whether they are collinear. Otherwise, evaluate nodes N_1, N_2, and N_3 for collinearity. Repeat this process until no more collinear nodes exist in the path. Create a new set P of path nodes and add the reserved nodes to set P. Set $P = \{p_j\}, j \in [0, m]$, where p_0 is the start node, p_m is the goal node, and $p_1, \ldots, p_{m-1}$ are the intermediate nodes;
- Create the set K of nodes that the path must pass through. Add p_0 to set K and consider p_0 as the current point for evaluation. First, connect nodes p_0 and p_m to determine whether the straight line segment p_0p_m passes through obstacles. If it does not, p_0p_m is considered a viable path, and one should add p_m to set K. If p_0p_m is obstructed, connect p_0 to p_{m-1} and continue the evaluation. If p_0p_{m-1} is clear, add p_{m-1} to set K. If p_0p_{m-1} is obstructed, connect p_0 to p_{m-2} and continue evaluating. Continue this process until the must-pass node p_{t1} is identified and added to set K. Then, use p_{t1} as the current point and repeat the evaluation process. Repeat these steps until p_m is added to set K;
- Connect the nodes extracted from set K sequentially; the resulting path is the path after one smoothing. The schematic diagram of the redundant node removal process is shown in Figure 11.



(**a**) (**b**) (**c**)

Figure 11. Schematic diagram for removing redundant nodes: (**a**) original path; (**b**) removing collinear nodes; (**c**) removing redundant turning nodes.

3.3.2. Quadratic Smoothing

Although the path after primary smoothing reduces the number of redundant nodes and inflection points, it still exhibits problems such as unsmooth turning angles, which do not satisfy the motion characteristics of the robot. Therefore, to enhance the smoothness of the path, this paper adopts the dynamic circular cutting method for secondary smoothing. The schematic diagram of the dynamic circular cut method is shown in Figure 12.

For the turning corner shown in Figure 12 ∠ABC, the steps of the dynamic circular cut method are as follows:

Compare the lengths of the sides of ∠ABC. Choose the endpoint A of the shorter side AB as the tangent point and draw a perpendicular line through A to intersect the angle bisector BO_0 of ∠ABC at point O_1. Then proceed to step 2;

Make a circle with O_1 as the center and AO_1 as the radius. Judge whether the arc AQ crosses obstacles or not. If it does, perform step 3; otherwise, use the arc AQ instead of the straight lines AB and BQ;

The tangent point A is moved upward along AB by a length λ (the value of λ is chosen according to the actual situation), denoted as A_1. Use A_1 as the tangent point to perform step 1;

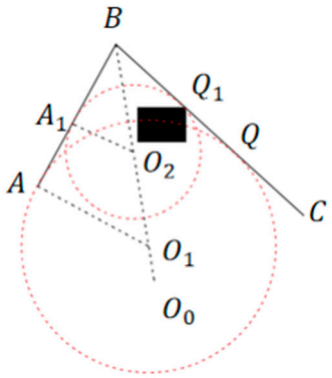


Figure 12. Schematic diagram of the dynamic circle cutting method.

- Determine whether all the turning corners have been smoothed. If yes, the secondary smoothing is completed; otherwise, continue with step 1.

4. Simulation Analysis and Experimental Validation

4.1. Simulation Analysis

In order to verify the effectiveness of the improved algorithm, this paper simulates and tests the A* algorithm, the JPS algorithm, and the improved A* algorithm on ten different grid maps. The computer we use is configured with a Windows 11 operating system, an R7-5800H processor with 3.2 GHz, and 16 GB of RAM.

The results of the three algorithms tested on a grid map of size 40 × 40 with a 30% obstacle ratio are shown in Figure 13. In the figure, the green grid indicates the start node, the red grid indicates the target node, the black grids indicate obstacles, the blue grids indicate nodes added to OpenList, and the yellow grids indicate nodes added to CloseList. The blue solid line represents the planned path, the green solid line represents the path obtained by primary smoothing, and the red solid line represents the final path obtained after secondary smoothing. As can be seen from Figure 13, the length and the number of turns of the optimal path obtained by the improved A* algorithm are better than those of the other two algorithms in the same environment. Other data obtained from the simulations are shown in Tables 1 and 2.

Table 1 compares the three algorithms in terms of the nodes traversed, path lengths, numbers of turns, and search times in grid maps of different sizes with a consistent obstacle percentage of 30%. It is evident that the number of nodes traversed by the improved algorithm is better than that of the A* algorithm and worse than that of the JPS algorithm. However, due to the nature of bidirectional search, it is guaranteed that the search time will outperform the other two algorithms. In 20 × 20 maps, the improved algorithm's search time efficiency improves by 50.29% and 26.37%, respectively, relative to the other algorithms; in 40 × 40 maps, it improves by 62.07% and 30.59%, respectively; in 60 × 60 maps, it improves by 68.66% and 37.13%, respectively; in 80 × 80 maps, it improves by 72.62% and 42.41%, respectively; and in 100 × 100 maps, it improves by 78.37% and 50.43%, respectively. From these data, it can be concluded that as the map size increases, the algorithm's improvement in search speed becomes more significant. However, in some cases, this improvement may lead to longer path lengths. Additionally, the improved algorithm's paths also exhibit significantly fewer turns compared to the other two algorithms.

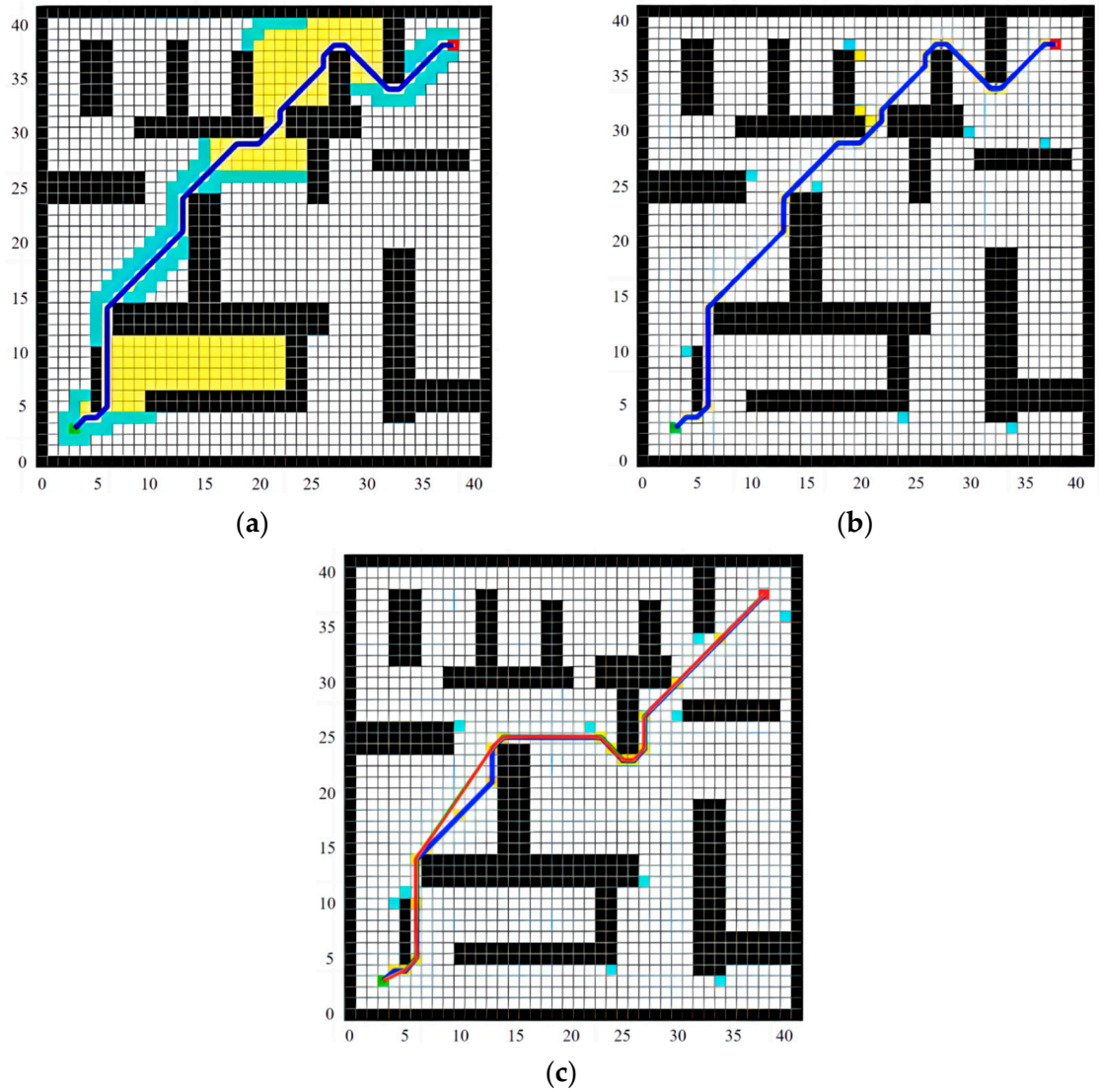


Figure 13. Comparison diagram of simulation results for three algorithms: (**a**) traditional A* algorithm; (**b**) traditional JPS algorithm; (**c**) improved algorithm.

Table 2 compares the three algorithms in terms of the nodes traversed, path lengths, numbers of turns, and search times in grid maps of varying sizes with a consistent obstacle percentage of 70%. It shows that the improved algorithm can generally plan optimal paths, as evidenced by shorter path lengths and fewer turns compared to the other two algorithms. According to Table 2, the search time efficiency of our algorithm improves by 26.14% and 12.70% relative to the other two algorithms in 20 × 20 maps, 39.25% and 20.80% in 40 × 40 maps, 52.47% and 32.10% in 60 × 60 maps, 58.21% and 39.79% in 80 × 80 maps, and 66.58% and 48.22% in 100 × 100 maps. It can be seen that the larger the size of the map is, the more effective the algorithm is in improving search speed. But in cases where the proportion of obstacles is higher, the improvement in search speed is not as significant as when the proportion of obstacles is lower. However, the optimal path is guaranteed to be obtained when the proportion of obstacles is higher.

Table 1. Comparison of three algorithms on grid maps with low proportions of obstacles.

Map Size	Experimental Algorithms	Number of Nodes Traversed (Number)	Path Length (Meters)	Number of Turns (Times)	Search Time (Seconds)	Percentage of Obstacles
20 × 20	A* algorithm	77	24.7279	7	0.233068	30%
	JPS algorithm	20	24.7279	7	0.157325	
	Improved algorithm	26	24.1381	6	0.115842	
40 × 40	A* algorithm	326	61.0122	17	0.707153	
	JPS algorithm	39	61.0122	17	0.386459	
	Improved algorithm	32	58.0070	10	0.268241	
60 × 60	A* algorithm	290	88.3259	11	1.101714	
	JPS algorithm	38	88.3259	11	0.549135	
	Improved algorithm	30	93.5291	6	0.345265	
80 × 80	A* algorithm	836	131.6396	21	3.334070	
	JPS algorithm	51	131.6396	21	1.585116	
	Improved algorithm	52	134.3342	15	0.912868	
100 × 100	A* algorithm	876	160.5097	23	5.128762	
	JPS algorithm	55	160.5097	23	2.237949	
	Improved algorithm	48	159.7931	16	1.109351	

Table 2. Comparison of three algorithms on grid maps with high proportions of obstacles.

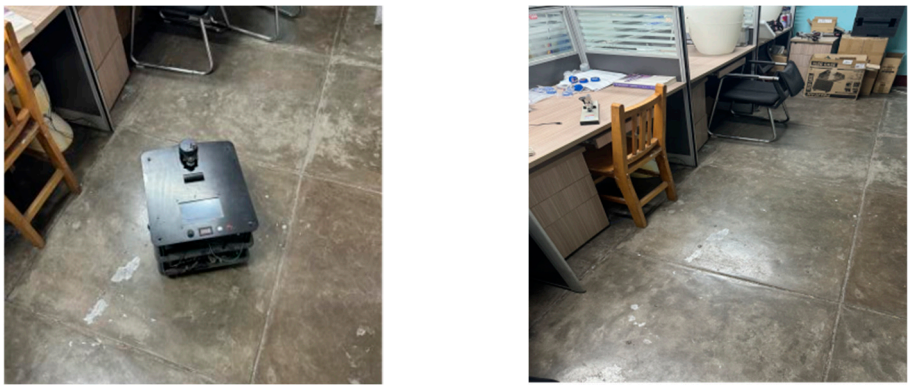
Map Size	Experimental Algorithms	Number of Nodes Traversed (Number)	Path Length (Meters)	Number of Turns (Times)	Search Time (Seconds)	Percentage of Obstacles
20 × 20	A* algorithm	131	29.3137	12	0.448762	70%
	JPS algorithm	40	30.7279	9	0.379653	
	Improved algorithm	57	30.2587	9	0.331437	
40 × 40	A* algorithm	183	62.6274	25	1.247487	
	JPS algorithm	79	62.6274	25	0.956823	
	Improved algorithm	125	60.3749	18	0.757804	
60 × 60	A* algorithm	280	100.7696	34	2.161697	
	JPS algorithm	86	100.7696	32	1.513218	
	Improved algorithm	128	99.3118	29	1.027475	
80 × 80	A* algorithm	694	136.9117	31	4.919608	
	JPS algorithm	113	136.9117	31	3.414556	
	Improved algorithm	132	134.7762	24	2.055904	
100 × 100	A* algorithm	791	173.3970	35	7.117738	
	JPS algorithm	127	173.3970	35	4.593951	
	Improved algorithm	144	169.9316	27	2.378748	

The simulation results indicate that the efficiency of the improved A* algorithm is influenced by the size of the environment map and the ratio of obstacles. Specifically, the improved algorithm demonstrates greater efficiency in improving the search speed with larger maps. Conversely, with smaller maps, the efficiency of the improved algorithm in enhancing the search speed diminishes. When the ratio of obstacles in the map varies, due to the introduction of environmental constraints, if the percentage of obstacles is low, the improved algorithm shows less noticeable improvement in path length, and in some cases, the path length may even worsen. However, the improvement in search time is significant. Conversely, when the percentage of obstacles is higher, the improved algorithm demonstrates more pronounced improvement in path length, but the improvement in search time deteriorates compared to the scenario with a lower obstacle percentage. However, in

general, compared to the other two algorithms, the improved A* algorithm has significantly enhanced both the efficiency of pathfinding and algorithm optimization.

4.2. Experimental Validation

In order to verify the feasibility of the improved A* algorithm in the actual operation of the mobile robot, the algorithm was applied to the two-wheel-drive differential robot built by our team, as shown in Figure 14a.The differential robot utilizes the RPLIDAR A1 laser rangefinder to obtain the 2D point cloud data of the external environment. It integrates odometer data for localization and constructs a 2D map using the Amcl and Gmapping function packages. Finally, it performs global path planning using the global planner in the Move Base function package [20]. The experimental scene described in this paper was a section of aisles in the laboratory, as shown in Figure 14b.



(**a**) (**b**)

Figure 14. Mobile robots and experimental environments: (**a**) differential robot; (**b**) laboratory.

The two path plannings performed by the differential robot at the experimental site are shown in Figure 15. The results of the planning have been displayed using the ROS visualization tool Rviz, where the orange square represents the model of the cart and the green solid line depicts the path planned by the improved A* algorithm.



(**a**) (**b**)

Figure 15. The results of path planning using the improved algorithm: (**a**) planning for Path 1; (**b**) planning for Path 2.

The search times of the three algorithms for the two paths are shown in Tables 3 and 4, respectively. It can be concluded that the improved algorithm proposed in this paper further enhances search speed while essentially producing optimal paths. Therefore, the improved algorithm outperforms the other two algorithms significantly.

Table 3. Comparison of pathfinding times of the three algorithms for Path 1.

	A* Algorithm	JPS Algorithm	The Algorithms in This Paper
Path length/m	16.8	16.7	16.5
Search time/ms	267.89	84.43	56.25

Table 4. Comparison of pathfinding times of the three algorithms for Path 2.

	A* Algorithm	JPS Algorithm	The Algorithms in This Paper
Path length/m	14.7	14.7	15.0
Search time/ms	217.43	78.10	45.47

5. Conclusions

In this paper, for the traditional A* algorithm in path planning, there are many redundant nodes, which lead to problems such as high memory consumption and long search time, etc.; an improved A* algorithm under the combination of bidirectional JPS strategy is proposed. The effectiveness of the improved algorithm has been demonstrated by analyzing and comparing the results of the three algorithms for path planning under different grid maps. The simulation results in different grid maps demonstrate that while ensuring basic path length guarantees, the algorithm proposed in this paper not only significantly improves path search speed but also notably reduces the number of turns in a path. Especially in larger scenarios, the effect of improving the search speed is more pronounced. The improved algorithm has been applied to the robot platform for experiments, and the experimental results fully demonstrate that the improved algorithm meets the requirements for fast path planning, with noticeable optimization effects.

Author Contributions: Project administration, F.W.; writing—original draft preparation, W.S.; writing—review and editing, P.Y. and H.W.; supervision, H.L. All authors have read and agreed to the published version of the manuscript.

Funding: "This research was funded by the Fundamental Research Program of Shanxi Province, grant number 20210302123054" and "Supported by the Opening Foundation of Shanxi Provincial Key Laboratory for Advanced Manufacturing Technology (No. XJZZ202307)".

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data are contained within the article. The dataset is available on request from the authors.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Qu, D.K.; Du, Z.J.; Xu, D.G.; Xu, F. Research on Path Planning for a Mobile Robot. *J. Robot.* **2008**, *30*, 97–101+106.
2. Cui, W.; Zhu, F.Z. Review of Path Planning Algorithms for Robot Navigation. *J. Comput. Eng. Appl.* **2023**, *59*, 10–20.
3. Xu, Y.; Guan, G.F.; Song, Q.W.; Jiang, C.; Wang, L.H. Heuristic and random search algorithm in optimization of route planning for Robot's geomagnetic navigation. *J. Comput. Commun.* **2020**, *154*, 7–12. [CrossRef]
4. Zhi, C.B.; Zhang, A.J.; Du, X.Y.; Peng, P. Research on Global Path Planning of Mobile Robot Based on Improved A* Algorithm. *J. Comput. Simul.* **2023**, *40*, 486–491.
5. Qi, F.L.; Wang, X.Q.; Zhang, W.Y. Research on AGV Obstacle Avoidance Path Planning Based on Improved A* Algorithm. *J. Mach. Tool Hydraul.* **2023**, *51*, 34–39.
6. Chen, C. Research on Robot Shortest Path Planning with Improved A* Algorithm. *J. Comput. Digit. Eng.* **2023**, *51*, 1697–1701.
7. Zhong, X.Y.; Tian, J.; Hu, H.S.; Peng, X.F. Hybrid Path Planning Based on Safe A* Algorithm and Adaptive Window Approach for Mobile Robot in Large-Scale Dynamic Environment. *J. Intell. Robot. Syst.* **2020**, *99*, 65–77. [CrossRef]
8. Sang, T.T.; Xiao, J.C.; Xiong, J.F.; Xia, H.Y.; Wang, Z.Z. Path Planning Method of Unmanned Surface Vehicles Formation Based on Improved A* Algorithm. *J. Mar. Sci. Eng.* **2023**, *11*, 176. [CrossRef]

9. Huang, C.J. Robot Path Planning Design Based on Improved Ant Colony Algorithm. *J. Value Eng.* **2023**, *42*, 51–53.
10. Fu, Q.; Ning, Y.K.; Ji, Y.F.; Sun, X.Y. Path Planning Based on Improved RRT and DWA Fusion Algorithm. *J. Comput. Simul.* **2023**, *40*, 429–435.
11. Wang, Y.; He, T.; Silva, P. Wide-band high-accuracy ADC using segmented DAC with DWA and mismatch shaping. *J. Electron. Lett.* **2017**, *53*, 713–714. [CrossRef]
12. Gao, F.X.; Hao, W.J.; Wu, Y.; Sun, C. Research on Robot Obstacle Avoidance Path Planning Based on Improved Artificial Potential Field Method. *J. Comput. Simul.* **2023**, *40*, 431–436+442.
13. Zhang, M.; Wu, Y.Z. Optimization of transfer robot path planning based on A* algorithm. *J. Mod. Electron. Tech.* **2023**, *46*, 135–139.
14. Gong, Y.X.; Liu, G.H.; Zhang, W.K.; Yu, D.Y.; Cui, Y.X.; Shen, Z.B. Path Planning Method of Improving A* Algorithm Using Convex Corner. *J. Comput. Eng. Appl.* **2023**, *59*, 309–315.
15. Harabor, D.; Grastien, A. The JPS pathfinding system. In Proceedings of the 5th Annual Symposium on Combinatorial Search, Niagara Falls, ON, Canada, 19–21 July 2012.
16. Zhao, X.; Wang, Z.; Huang, C.K.; Zhao, Y.W. Mobile Robot Path Planning Based on an Improved A*Algorithm. *J. Robot.* **2018**, *40*, 903–910.
17. Lv, T.Z.; Zhao, C.X.; Xia, P.P. Global path planning based on simultaneous visibility graph construction and A* algorithm. *J. Nanjing Univ. Sci. Technol.* **2017**, *41*, 313–321.
18. Jeddisaravi, K.; Alitappeh, J.R.; Pimenta, A.C.L.; Guimarães, G.F. Multi-objective approach for robot motion planning in search tasks. *J. Appl. Intell.* **2016**, *45*, 305–321. [CrossRef]
19. Zhang, W.M.; Zhang, Y.; Zhang, H. Path planning of coal mine rescue robot based on improved A*algorithm. *J. Coal Geol. Explor.* **2022**, *50*, 185–193.
20. Harabor, D.; Grastien, A. Online graph pruning for pathfinding on grid maps. In Proceedings of the 25th AAAI Conference on Artificial Intelligence, Menlo Park, CA, USA, 7–11 August 2011.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.