applied sciences

MDPI

Article

Global Path Planning for Differential Drive Mobile Robots Based on Improved BSGA* Algorithm

Ming Yao [1], Haigang Deng [2,*], Xianying Feng [1], Peigang Li [1], Yanfei Li [1] and Haiyang Liu [1]

[1] School of Mechanical Engineering, Shandong University, Jinan 250061, China; ym1601065487@163.com (M.Y.); fxying@sdu.edu.cn (X.F.); pgli@vip.sina.com (P.L.); yanfei-li@foxmail.com (Y.L.); 18943653361@163.com (H.L.)
[2] School of Instrumentation Science and Engineering, Harbin Institute of Technology, Harbin 150001, China
* Correspondence: hgdeng_hit@163.com

Abstract: The global path planner is an important part of the navigation system for autonomous differential drive mobile robots (DDMRs). Aiming at the problems such as long calculation time, large number of search nodes, and poor smoothness of path when A* is applied to global path planning, this study proposes an improved bidirectional search Gaussian-A* (BSGA*) algorithm. First, the Gaussian function is introduced to realize the dynamic weighting of the heuristic function, which reduces the calculation time. Secondly, the bidirectional search (BS) structure is adopted to solve the problem of nodes' repeated search when there are large obstacles between the starting point and the target point. Finally, a multi-layer turning point filter strategy is proposed to further smooth the path. In order to verify the performance of the improved BSGA* algorithm, experiments are carried out in simulation environments with the size of 15 × 15 and 30 × 30, respectively, and compared with the five common global path planning algorithms including ant colony optimization (ACO), D* lite algorithm, and genetic algorithm (GA). The results show that the improved BSGA* algorithm has the lowest calculation time and generates the shortest and smoothest path in the same environment. Finally, the program of the improved BSGA* algorithm is embedded into the LEO ROS mobile robot and two different real environments were built for experimental verification. By comparing with the A* algorithm, Dijkstra algorithm, ACO, D* lite algorithm, and GA, the results show that the improved BSGA* algorithm not only outperforms the above five algorithms in terms of calculation time, length, and total turning angle of the generated paths, but also consumes the least time when DDMR drives along the generated paths.

Keywords: differential driven mobile robots; global path planning; improved BSGA* algorithm; bidirectional structure; Gaussian function; turning point filtering strategy

Citation: Yao, M.; Deng, H.; Feng, X.; Li, P.; Li, Y.; Liu, H. Global Path Planning for Differential Drive Mobile Robots Based on Improved BSGA* Algorithm. *Appl. Sci.* **2023**, *13*, 11290. https://doi.org/10.3390/app132011290

Academic Editors: Nadjim Horri, William Holderbaum and Fabrizio Giulietti

Received: 5 September 2023
Revised: 11 October 2023
Accepted: 12 October 2023
Published: 14 October 2023

1. Introduction

DDMRs are widely used in earthwork construction, firefighting, the military, and other fields because of their ability to operate in high-risk, high-intensity environments [1–3]. At present, most DDMRs rely on manual control, which requires strict manipulation skills and experience from the operators. This way of long-term stable operation is difficult, and due to the relatively harsh working environment, it often brings certain risks to the operators. Therefore, the autonomous navigation systems of DDMRs have become a research hotspot [4,5]. Global path planning is the basis of an autonomous navigation system, which can pre-plan a safe and feasible path according to the environmental information collected by the sensors in advance to guide their moving process [6].

Global path planning, as a kind of static planning (also called offline planning), refers to planning an optimal collision-free path from the starting point to the target point for the robot under the consideration of certain evaluation criteria [7]. It is evaluated in three main ways. (1) Optimality: it can use path length, number of turns, driving time, and other standards to evaluate the superiority of the path. (2) Authenticity: the planned path

should meet the driving requirements of the mobile robot. (3) Integrity: the path should connect the starting point and the target point continuously, and achieve the avoidance of obstacles [8].

Many algorithms have been applied to global path planning for mobile robots, including the A* algorithm [9], rapidly exploring random tree (RRT) algorithm [10], ant colony algorithm (ACO) [11], genetic algorithm (GA) [12], etc. On this basis, many scholars have carried out extensive and in-depth research. The RRT algorithm is suitable for continuous domain search because it does not require the discretization of a continuous space during planning [13]. However, this algorithm focuses more on finding feasible paths rather than reducing costs, thus the costs of the path tend to be high [14]. Wang [15] proposed a kinematic constrained bi-directional RRT algorithm (KB-RRT*) for path planning. By introducing kinematic constraints, KB-RRT* can avoid unnecessary tree growth, but the global path obtained based on the KB-RRT* algorithm is still a suboptimal solution. ACO is a heuristic algorithm that simulates the foraging behavior of ant colonies. Although the algorithm can obtain the optimal path, its convergence speed is slow and it is easy to fall into the local optimal. Hou et al. [16] proposed an improved ACO algorithm by adopting the communication mechanism of ants using their antennae to contact each other in nature, and applied the algorithm to mobile robots. However, the algorithm still has problems in calculation efficiency. GA can generate a large number of new individuals when the crossover probability is large, thereby improving the global search scope, but there is a precocious phenomenon and the results are unstable. Tuncer A et al. [17] proposed an improved GA using optimized mutation operators, which simultaneously checked all idle nodes near the mutation nodes, selected nodes according to the fitness value of paths rather than the direction of mutation nodes, and obtained a higher average fit than other methods. However, the paths are too close to the obstacles and are less safe. The Dijkstra algorithm [18] is a single-source path search algorithm that expands outward from the start point until it reaches the target and obtains the shortest path. With the increase in environmental information, the Dijkstra algorithm will consume a lot of computing time. P. E. Art et al. [19] proposed the A* algorithm in 1968 by combining the Dijkstra algorithm with the BFS algorithm. The A* algorithm greatly improves the search efficiency by introducing a heuristic function and the planned path is optimal. In addition, the path planned by the A* algorithm based on the grids has a sudden change in turning angle, while the DDMRs can turn in place by the speed difference between motors on both sides, so the A* algorithm is more suitable for the global path planning of the DDMRs. However, the A* algorithm still has the problems of long calculation time, poor smoothness, and low safety when facing a complex environment.

To solve the problem of the long calculation time of the A* algorithm, many scholars improve the algorithm structure or optimize the heuristic function. Zhang et al. [20] proposed an improved A* algorithm containing a bi-directional sector expansion and a variable-step search strategy, thus improving the computational efficiency. Liu et al. [21] adopted the Delaunary triangulation algorithm to deal with complex obstacles, took the generated Voronoi points as the preferred pathfinding nodes, then designed the dynamic fusion pathfinding algorithm (DFPA) based on the Delaunary triangulation algorithm and the improved A* algorithm. Jiang H et al. [22] introduced a cosine factor into the heuristic function of the A* algorithm to optimize the search direction, and adopted a search strategy that synchronizes the starting point and the target point to realize the path planning for the electric disinfection vehicle, which has higher computational efficiency, but does not consider the kinematic model. For the problem of poor path smoothness, scholars optimize the A* algorithm by improving the heuristic function or adjusting the search step size. Liu et al. [23] designed a global path yaw angle based on the relationship between the real-time position and global path, then introduced it into the heuristic function of the A* algorithm to improve the path smoothness. Tang et al. [9] filter the nodes in the closed list to avoid the irregular path. Meanwhile, a cubic B-spline curve is adopted to smooth the path and improve the stability in turning. Li et al. [24] introduced the jump-point search strategy into

A* algorithm and effectively reduced the number of turning points by ignoring unnecessary nodes. For the problem of low security, some scholars have introduced the influence of obstacles into the A* algorithm. Zhang et al. [25] converted the distance between the mobile robot and the obstacles into the time cost, so as to optimize the cost function and improve the safety. Sang et al. [26] narrow the search scope by imposing constraints of maximum search distance and maximum path length, then maintain a safe distance by reducing the search points near the obstacles. Cui [27] predicted the future movement of the target using Bernstein basis polynomials combined with the information of the obstacle distribution around the target, then utilized the A* algorithm to search for a safe tracking path. Finally, in the case of dynamic constraints, a quadratic programming method is used to optimize the tracking path to improve the smoothness. Dang [28] proposed an Adaptive Back-stepping Hierarchical Sliding Mode Control (ABHSMC) scheme for three-wheeled mobile robots (3WMRs) based on RBF neural networks. By aggregating all uncertain components in specific vectors and estimating using an RBF neural network, the effect of uncertainties will be minimized. Then, combining it with the TEB local planner and A* global planner improves the navigation and obstacle avoidance performance.

Although these studies have optimized the performance of the A* algorithm to some extent, the direct application to DDMRs are still one-sided, because, in addition to the factors mentioned above, the global path planning of DDMRs also needs to consider the total driving time. The former is related to the setting of the heuristic function, algorithm structure, and processor performance, while the latter is related to the motion state, length, and total turning angle of the path. Therefore, the kinematic model of DDMR is established to analyze the motion state on the turning point. On this basis, Gaussian functions are firstly used to dynamically adjust the weight ratio of the heuristic functions, which effectively reduces the calculation time. Secondly, the BS structure is introduced into the A* algorithm, which solves the problems of repeated node search when there is a large area of obstacles between the starting point and the target point. Finally, a multi-layer turning nodes filtering strategy is proposed to reduce the total turning angle and improve the smoothness of the path. The simulation results show that the improved BSGA* algorithm has higher efficiency than the A* algorithm, GA, and ACO algorithm, and can plan a more reasonable driving route. The LEO ROS robot is used for experimental verification. The experimental results show that it has better results than the five common global path planning algorithms.

2. Materials and Methods

This section introduces the kinematic modeling of DDMR, the process of environment modeling using the grid method, and the basic principle of the A* algorithm.

2.1. Analysis of Turning Motion of DDMR

Figure 1 shows the analysis of DDMR's turning motion. The rectangular coordinate system XOY is established, the coordinates of DDMR are set as (x, y), the linear velocity is v, and the angular velocity is ω. The width of DDMR is B, the turning center is ICR, and the turning radius is R. Because DDMR has nonholonomic constraints [29], its motion can be described only through linear velocity v and angular velocity ω [30].

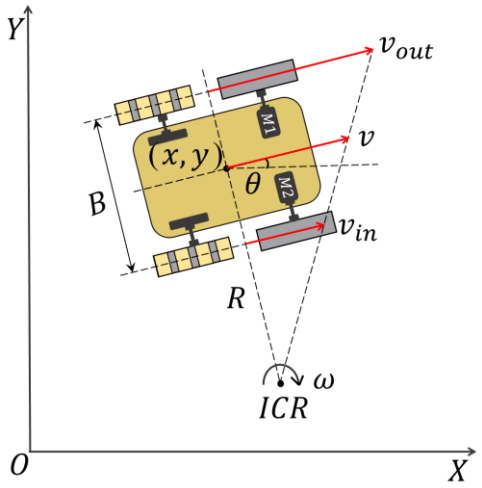


Figure 1. Analysis of DDMR's turning motion.

The controller makes the DDMR reach the expected motion state by controlling the velocities of the drive motors on both sides, the linear velocities of the drive wheels on both sides can be expressed as follows:

$$\begin{cases} v_{in} = (R + B/2)\omega \\ v_{out} = (R - B/2)\omega \end{cases} \tag{1}$$

where v_{in} and v_{out} are, respectively, the inner linear velocity and outer linear velocity of DDMR. From Equation (1), the linear velocity of the center of mass can be expressed as follows:

$$\mathrm{v} = \omega R = (v_{in} + v_{out})/2 \tag{2}$$

Based on Equations (1) and (2), the turning radius R and angular velocity ω of DDMR can be expressed as follows:

$$\begin{cases} \omega = (v_{out} - v_{in})/B \\ R = [B(v_{out} + v_{in})]/[2(v_{out} - v_{in})] \end{cases} \tag{3}$$

According to Equation (3), when v_{out}=$-v_{in}$, $R = 0$, DDMR can realize in place turning. Assuming that the angular velocity of DDMR is constant in the turning process, the turning angle θ is proportional to the turning time Δt. Many studies [31] have lower driving time by reducing the number of turns, but this is not comprehensive. The direct factor affecting the turning time is the total turning angle, and there is no direct functional correspondence between it and the number of turns. Since the DDMR realizes turning through the speed difference of the drive motors arranged on both sides of the body, when the total turning angle is too large, it will aggravate the wear of the drive motors, conveyor belts, and other components, and even reduce the service life. Secondly, the turning of the DDMR is a relatively slow process, which includes three decomposition actions: deceleration, turning, and acceleration, and consumes more time than that of the straight motion. Therefore, reducing the total turning angle can effectively shorten the time required for the DDMR to drive along this path. So the total turning angle should be used as the evaluation index to reduce the turning time.

2.2. Grid Modeling

Based on the driving characteristics of DDMR that can be turned in place, the grid method is used to establish the environment map. The work space is projected as a two-dimensional plane and discretized into a square grid of uniform, continuous, and non-intersecting. According to the environmental information, the grids with obstacle projection

can be set as obstacle grids, represented by 1; the grids without obstacle projection can be set as free grids, represented by 0. A Cartesian coordinate system is established in the grid map, with the lower left corner as the origin of coordinates, the horizontal axis for the X axis, with values increasing from left to right. The vertical axis for the Y axis, with values increasing from bottom to top. The position of each grid is represented by the coordinate $p(x_i, y_j)$ in the upper right corner of the grid, where $i, j = 1, 2, 3 \ldots\ldots, n$.

The environmental information in the grid maps will be different from the actual maps, which mainly depends on the size of the grids. If the grids are too large, the resolution of the environment maps will be reduced, which may make the planned paths deviate from the theoretical optimal paths seriously, or even fail to search for feasible paths. If the grids size are too small, the resolution of the environment maps will be too high, which will occupy a lot of computational resources. In order to simplify the planning problem, DDMR is regarded as a square with side length $l_{edge} = 1$, and the side length l of the grid is also set as 1. In order to improve the safety of the path, DDMR should keep away from obstacles during the driving process. Therefore, the obstacles are puffed up as shown in Figure 2, the grids with only partial obstacles are also set as obstacle grids.

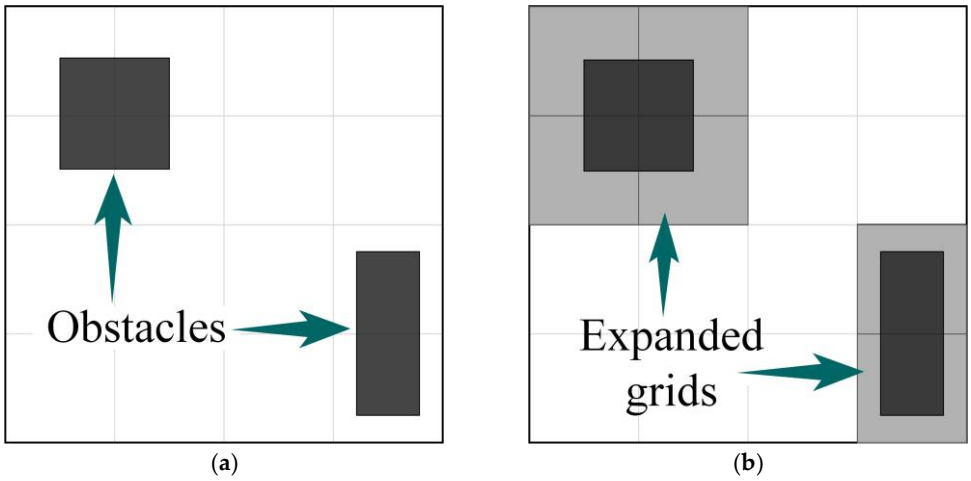


Figure 2. Obstacle puffing process. They should be listed as follows: (**a**) Before puffing process; (**b**) After puffing process.

2.3. Procedure of A Algorithm*

As a heuristic path planning algorithm, the A* algorithm is used to find the optimal path in the static environment [32]. This algorithm searches from the starting point according to the predetermined search strategy, calculates the actual cost of each feasible node around the current node to the starting point and the heuristic cost to the target point, and selects the node with the minimum total cost as the next extended node. The algorithm ends when the extended node is overlapped with the target point [33]. The core of the A* algorithm lies in the cost function, which is as follows:

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

where $f(n)$ represents the total cost from the starting point (X_s, Y_s) to the target point (X_e, Y_e) through the current point (X_n, Y_n); $g(n)$ represents the actual cost from the starting point (X_s, Y_s) to the current point (X_n, Y_n); and $h(n)$ represents the heuristic cost from the current point (X_n, Y_n) to the target point (X_e, Y_e). The traditional A* algorithm adopts the 4-neighborhood search strategy. Because DDMRs can realize in-place turning, it can be reasonably considered to complete the diagonal movement of the grids. In order to make the path generated by the A* algorithm more realistic, the search neighborhood is extended to 8 neighborhoods, as shown in Figure 3, with 8 turning angles $(0°, 45°, -45°, 90°, -90°, 135°, -135°, 180°)$. There are mainly three heuristic functions, in-

cluding Euclidean distance function [34], Chebyshev distance function [35], and Manhattan distance function [36]. They can be expressed, respectively, as follows:

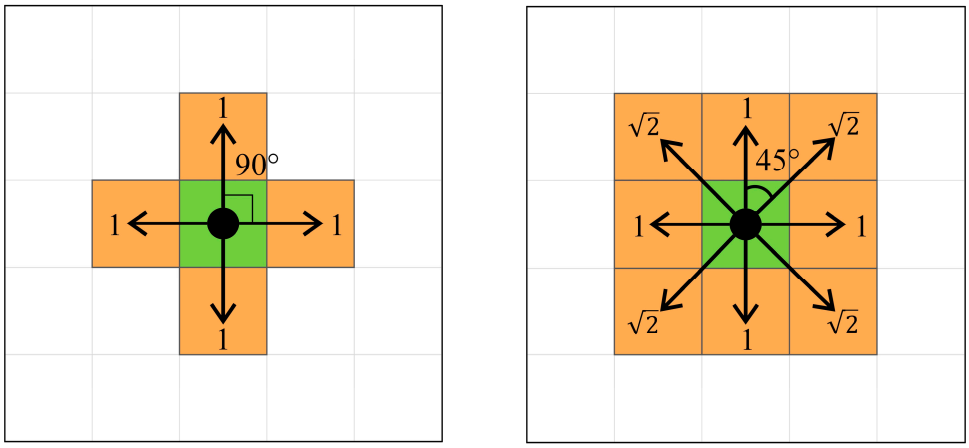


Figure 3. Extended search neighborhood. They should be listed as: (**a**) 4 search neighborhood; (**b**) 8 search neighborhood.

Euclidean distance function:

$$h(n) = \sqrt{(X_e - X_n)^2 + (Y_e - Y_n)^2} \tag{5}$$

Chebyshev distance function:

$$h(n) = |X_e - X_n| + |Y_e - Y_n| \tag{6}$$

Manhattan distance function:

$$h(n) = \max(|X_e - X_n|, |Y_e - Y_n|) \tag{7}$$

For the heuristic function, if $h(n)$ is the same as the actual cost $P(n)$ from the (X_n, Y_n) to the (X_e, Y_e), all nodes extended by the A* algorithm are on the optimal path, it will not expand any remaining unarticulated points, and the search speed is the fastest at this time, but this is difficult to achieve. For the Manhattan distance, its value is often less than $P(n)$, although the optimal path can be found, the search speed will decrease. For the Chebyshev distance, its value is often greater than $P(n)$, although it searches faster, there is no guarantee that the optimal path can be found. Since the 8-neighborhood search strategy is adopted in this study, $h(n)$ obtained by using Euclidean distance is closer to $P(n)$, which ensures that the search speed can be improved on the basis of finding the optimal path. Therefore, Euclidean distance is adopted as the heuristic function.

The A* algorithm divides all extended nodes into two sets:

OpenList: Stores all extension nodes to be detected in the search process, and sorts them according to the $f(n)$.

CloseList: Stores all detected extension nodes in the search process to prevent repeated search.

On the basis of the above concepts, the steps of the A* algorithm are shown in Figure 4.

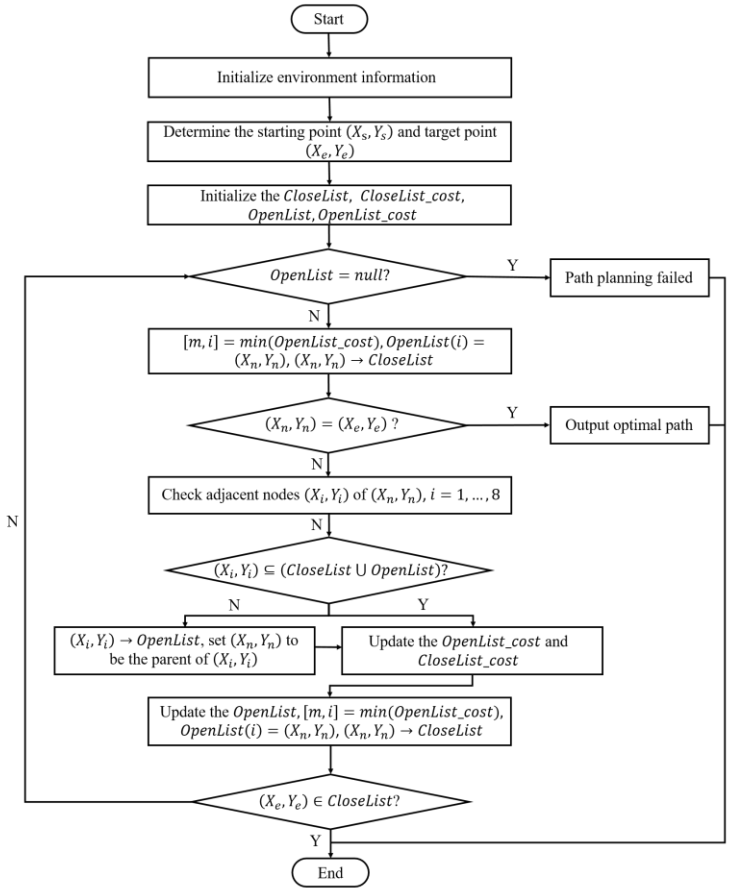


Figure 4. Program flow of A* algorithm.

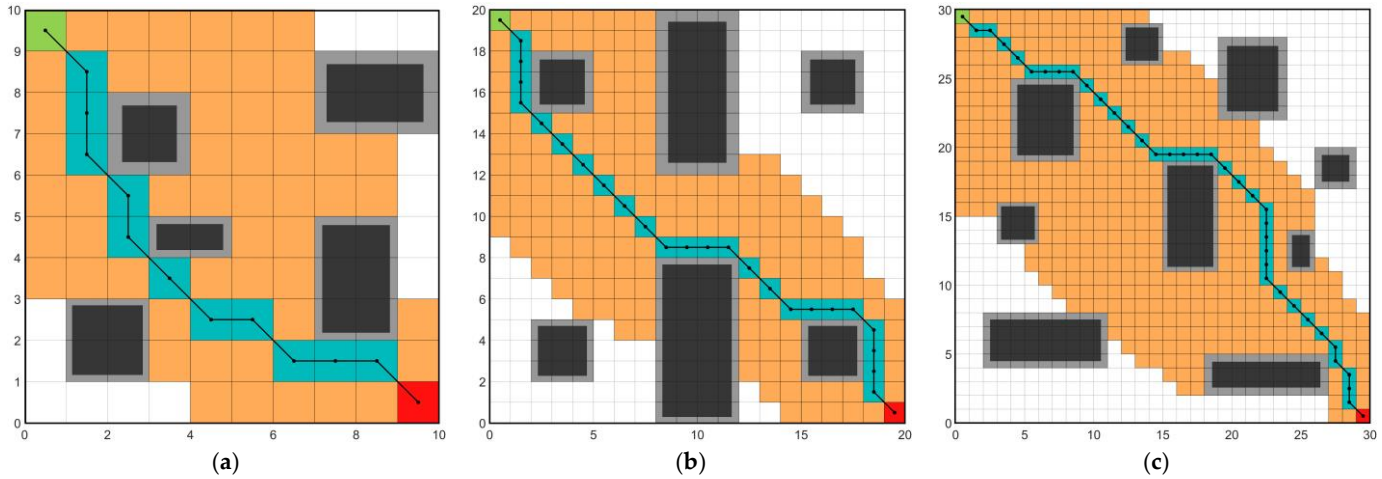
3. Improved BSGA* Algorithm

The traditional A* algorithm is improved to design a new global path planner, whose inputs include the values of grids, the starting point, the target point, and the attitude of the DDMR. The global path calculated by the improved BSGA* algorithm is divided into a series of waypoint coordinates as the outputs of the algorithm, which are relied upon to generate the linear velocity and angular velocity to guide the DDMR to move to the target point. In this planner, the constraints include kinematic constraints, that is, the maximum velocity of 0.6 m/s and the maximum angular velocity of $\pi/6$ rad/s. The environmental constraints, that is, the side lengths of the grids and the locations of the obstacles. Cost constraints: the sum of the actual costs and heuristic costs of each grid.

3.1. Improved Heuristic Functions

The heuristic function determines the quality of the A* algorithm [37]. As shown in Figure 5, the A* algorithm, using Euclidean distance as the heuristic function, can search a feasible path, but there are too many search nodes and the calculation time is too long. Therefore, this study addresses the above shortcomings by improving the heuristic function. When $h(n) = 0$, the A* algorithm is transformed into a Dijkstra algorithm, and a large number of nodes will be searched. Although the total cost in the planning process is the same as the actual cost, it is inefficient [38]. When $g(n) = 0$, the A* algorithm is transformed into the Breadth-First Search (BFS) algorithm, which has high planning efficiency but not an optimal path [39]. In order to make the path planning process both efficient and qualitative, $h(n)$ should usually occupy a larger weight in the beginning of the planning, so that the algorithm can search near the target point more quickly. At the end of the planning process, $g(n)$ should occupy a larger weight, so as to avoid repeated searches of nodes that may occur near the target point and reduce the total turning angle of paths. Since Euclidean

distance is used as the heuristic function in this study, $h(n)$ tends to be slightly smaller than the actual cost from the current node to the target node. As the current node gradually approaches the target node, this gap will become smaller and smaller. In the optimal case, $h(n)$ is always the same as the actual cost, so the algorithm can always select the optimal node. But this is often difficult to achieve.



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

Figure 5. Simulation results of traditional A* algorithm under different map environments. (**a**) Map size is 10 × 10. (**b**) Map size is 20 × 20. (**c**) Map size is 30 × 30.

Therefore, this study combined with the Gaussian function to construct the attenuation coefficient $D(n)$ for dynamically adjusting the weight ratio of the heuristic function; the heuristic function was rewritten as the following equation:

$$\begin{cases} f(n) = g(n) + \mathrm{h}(n)/D(n) \\ D(n) = \frac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(h(n)-\mu)^2}{2\sigma}} \end{cases} \tag{8}$$

Let $\mu = 0$ and $\sigma = 1/\sqrt{2\pi}$. At the beginning of the algorithm, the current node is far from the target point, at this time $D(n) \to 0^+$, so the weight ratio of $h(n)$ increases, thus improving the search efficiency. As the algorithm runs, the current node gradually approaches the target point, $D(n)$ gradually increases, and the weight ratio of $h(n)$ gradually decreases to reduce the total turning angle of the path.

In order to verify the performance of the Gaussian–A* (GA*) algorithm, Matlab2020a software was used to write the algorithm program. The Intel workstation running the program was equipped with an i9-10900 processor, with 64 GB RAM and 3.70 GHz main frequency. The subsequent simulation was also run based on this workstation. As shown in Figures 5 and 6, 50 executions were carried out on grid maps with sizes of 10 × 10, 20 × 20 and 30 × 30, respectively.

Table 1 shows the states represented by the different colored grids. It can be seen that, compared with the A* algorithm, the path generated by the GA* algorithm has fewer turning points and the number of nodes searched is also less. According to Table 2, it can be seen that, on the 10 × 10 grid map, the calculation time, the number of search nodes, and the total turning angle of the GA*s algorithm are, respectively, reduced by 64.43%, 40.00%, and 37.5% relative to the A* algorithm. On the 20 × 20 grid map, the calculation time, number of search nodes, and the total turning angle of the GA*s algorithm are, respectively, reduced by 80.98%, 56.38%, and 25% relative to the A* algorithm. On the 30 × 30 grid map, these three were reduced by 84.09%, 70.91%, and 41.67%, respectively. Although the length of the path generated by the GA* algorithm is not much different from that of the A* algorithm, it has higher efficiency, and with the gradual expansion of the map, this

difference becomes more and more obvious. In addition, the GA* algorithm can generate smoother paths, which is more conducive to DDMR driving.

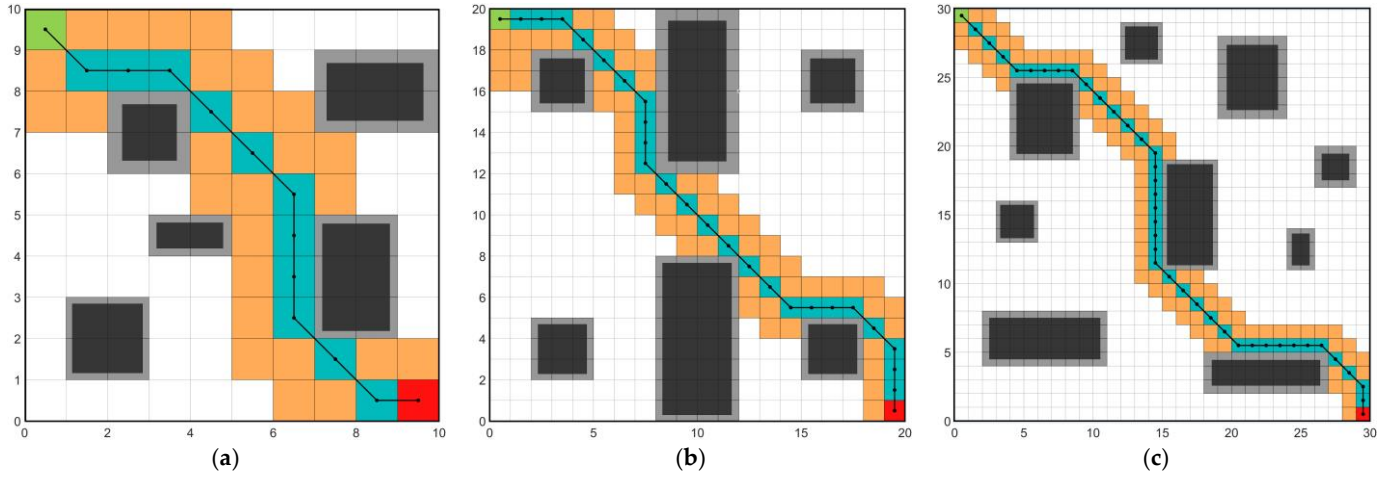


Figure 6. Simulation results of traditional GA* algorithm under different map environments. (**a**) Map size is 10 × 10. (**b**) Map size is 20 × 20. (**c**) Map size is 30 × 30.

Table 1. Grid state correspondence table.

No	Color	State of Grid
1	White	Free grid
2	Black	Obstacle grid
3	Grey	Expanded grid
4	Green	Starting point
5	Red	Target point
6	Orange	Searched point
7	Blue-green	Path point

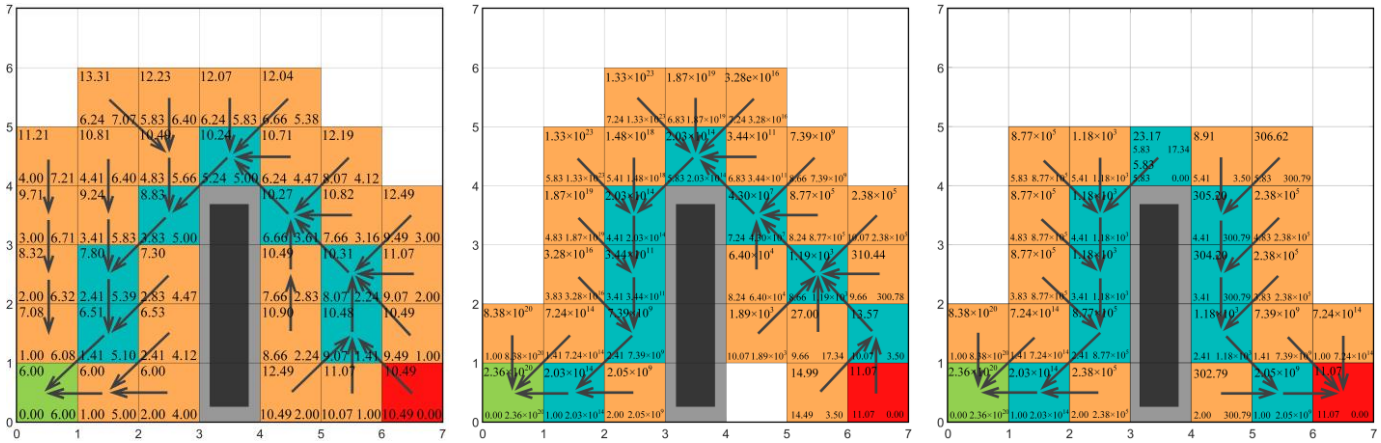
Table 2. Comparison of GA* algorithm and traditional A* algorithm on different maps; the values of average calculation time are reported as mean ± standard deviation for 50 executions.

Map Size	Algorithm	Average Calculation Time/ms	Number of Searched Nodes	Total Angle/°	Path Length
10 × 10	A*	574.94 ± 12.97	65	360	14.49
	GA*	204.51 ± 2.17	39	225	14.49
20 × 20	A*	3119.03 ± 41.53	188	360	30.39
	GA*	593.38 ± 2.76	82	270	30.39
30 × 30	A*	14,194.60 ± 75.32	440	540	45.70
	GA*	2258.70 ± 9.34	128	315	46.87

3.2. Bidirectional Search Structure

Although the GA* algorithm obviously improves the computational efficiency and smoothness of the path, the planning effect in some special cases still needs to be improved. As shown in Figure 7a,b, when there is a large obstacle between the starting point and the target point, DDMR needs to bypass the obstacle to reach the target point, so the A* algorithm and GA* algorithm will search in the direction that deviate from the starting point and the target point at the beginning process. The numbers in the upper-left, lower-left, and lower-right corners of the grids in Figure 7 correspond to the $f(n)$, $g(n)$, and $h(n)$ of the current node, respectively. The arrow points to the parent point of the node where the arrow tail is located. As can be seen from Figure 7a, $f(n)$ gradually increases in the process of avoiding obstacles. However, after avoiding obstacles, $g(n)$ is too large due to the long

path, so $f(n)$ is even higher than $f(n)$ that corresponds to the nodes at the beginning of the search process. Therefore, in the later stage of the A * algorithm, many nodes near the starting point will be repeatedly searched. As shown in Figure 7b, this phenomenon has been improved somewhat in the GA* algorithm but still exists. To address the above problem, this study, using a BS structure to improve the GA* algorithm, which is named the BSGA* algorithm.



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

Figure 7. Comparison of the cost during the calculation process. (**a**) A*. (**b**) GA*. (**c**) BSGA*.

The BSGA* algorithm searches from the starting point and the target point simultaneously, where the nodes to be searched in the forward search process beginning from the starting point are stored in *OpenList_forward*, and the nodes to be searched in the reverse search process beginning from the target point are stored in *OpenList_backward*. The forward and reverse search processes are roughly the same as that of GA*. The difference is that the target point of the forward search is *Current_back*, which is the current node of the reverse search, and the cost function is shown in Equation (9). The target point of the reverse search is *Current_forward*, which is the current node of the forward search, and the cost function is shown in Equation (10). The two search processes are carried out alternately and both adopt the eight neighborhood search strategy until the current nodes of the two search processes meet.

$$\begin{cases} f_{forward}(n_f) = g_{forward}(n_f) + \dfrac{h_{forward}(n_f)}{D_{forward}(n_f)} \\ D_{forward}(n_f) = \dfrac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(h_{forward}(n_f)-\mu)^2}{2\sigma}} \end{cases} \tag{9}$$

$$\begin{cases} f_{backward}(n_b) = g_{backward}(n_b) + \dfrac{h_{backward}(n_b)}{D_{backward}(n_b)} \\ D_{backward}(n_b) = \dfrac{1}{\sigma\sqrt{2\pi}} e^{\frac{-(h_{backward}(n_b)-\mu)^2}{2\sigma}} \end{cases} \tag{10}$$

where n_f is the current node of forward search; n_b is the current node of the reverse search; $g_{forward}(n_f)$ is the actual cost from the starting point to n_f; $h_{forward}(n_f)$ is the Euclidean heuristic cost of forward search from n_f to n_b; and $D_{forward}(n_f)$ is the dynamic attenuation coefficient of the forward heuristic function. $g_{backward}(n_b)$ is the actual cost of the target point to n_b; $h_{backward}(n_b)$ is the Euclidean heuristic cost of the next reverse search from n_b to the n_f; and $D_{backward}(n_b)$ is the dynamic attenuation coefficient of the backward heuristic function.

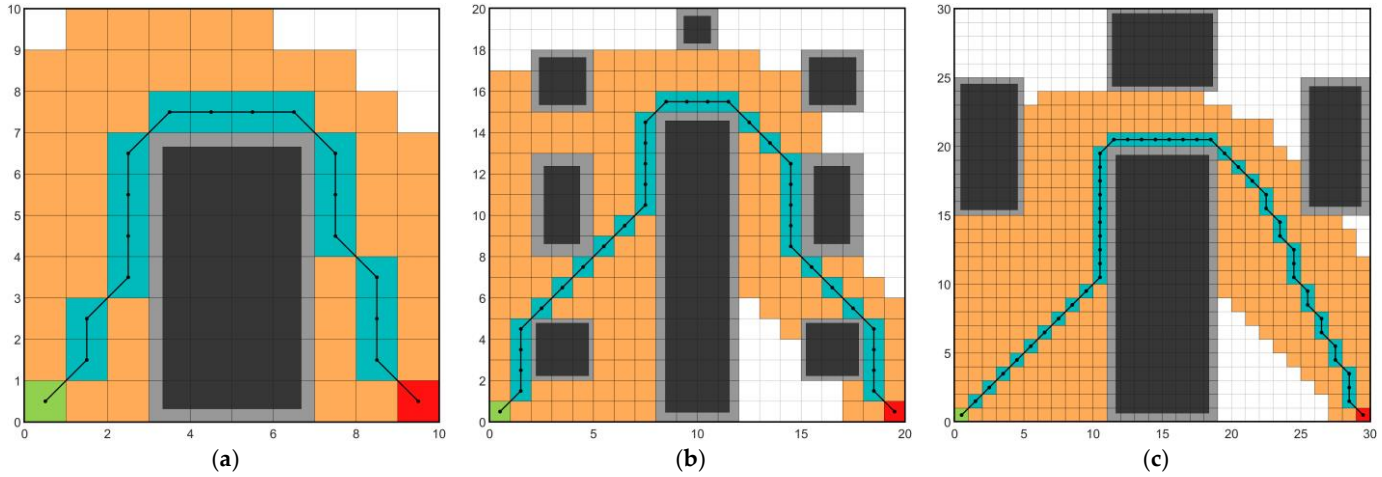
The pseudo code for the BSGA* algorithm is as follows (Algorithm 1).

Algorithm 1

Initialization
Puff up obstacles in the grid map
Determine the starting point (X_s, Y_s) and target point (X_e, Y_e)
$(X_s, Y_s) \rightarrow CloseList\_forward$, $(X_e, Y_e) \rightarrow CloseList\_backward$, set $CloseList\_forward\_cost = 0$, $CloseList\_backward\_cost = 0$
$Child\_nodes\_forward \rightarrow OpenList\_forward$, $Child\_nodes\_backward \rightarrow OpenList\_backward$
$Flag = 1$
$while = Flag$
    If $(Child\_nodes\_forward \subset OpenList\_forward) \cap (Child\_nodes\_backward \subset OpenList\_backward) \neq 1$
    Add nodes to the corresponding set
    Calculate the $f_{forward}$ of all nodes in $OpenList\_forward$, min $\_n_f \rightarrow CloseList\_forward$, set $n_f = current\_forward$, $f_{forward}(n_f) \rightarrow CloseList\_forward\_cost$
    Calculate the $f_{backward}$ of all nodes in $OpenList\_backward$, min $\_n_b \rightarrow CloseList\_backward$, set $n_b = current\_backward$, $f_{backward}(n_b) \rightarrow CloseList\_backward\_cost$
    $if\ current\_forward = current\_backward$
        Flip the elements in $CloseList\_backward$ and add them to $CloseList\_forward$
        $Cost = CloseList\_forward\_cost(end) + CloseList\_backward\_cost(end)$
        Generate the optimal path
        $Flag = 0$
    $else$
        Non-obstacle sub-nodes $current\_forward \rightarrow OpenList\_forward$
        Non-obstacle sub-nodes $current\_backward \rightarrow OpenList\_backward$
    $end\ if$
$end\ while$

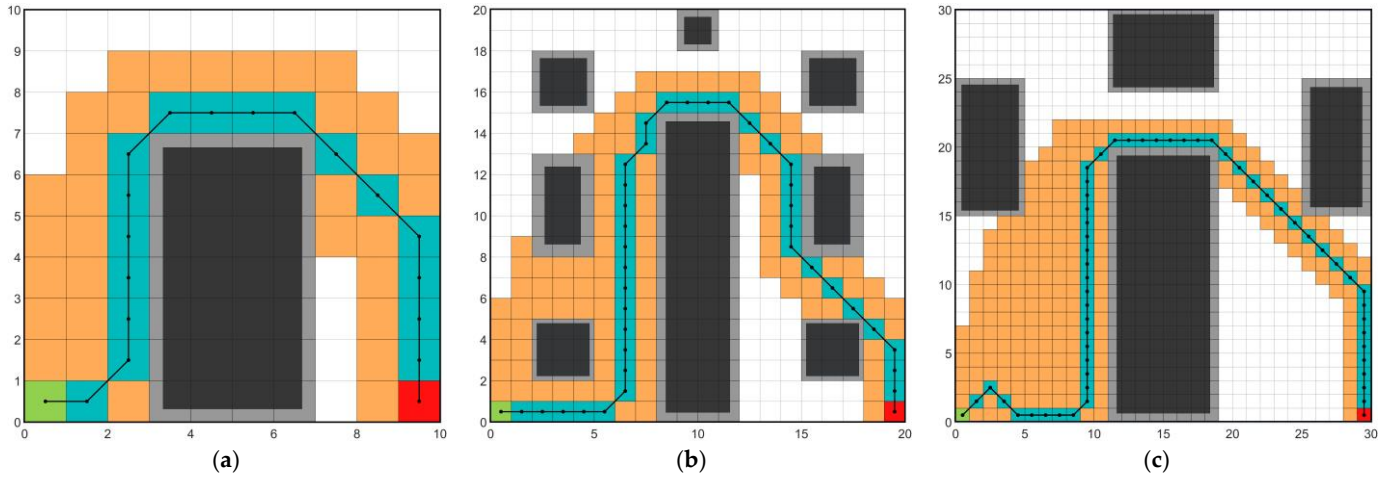
As shown in Figure 7c, the BS structure is introduced to dynamically switch and adjust the starting point and target point in the search process, so that the current node always keeps the trend of being close to the target point. In this process, $f(n)$ decreases rapidly until the two paths meet and falls to the lowest value. The values of the upper and lower levels of the encounter grids represent the total cost, actual cost, and heuristic cost of forward search and reverse search at this node, respectively. The introduction of the BS structure effectively solves the problem of repeatedly searching nodes when avoiding obstacles and reaching the target point, and improves the efficiency of search.

In order to verify the performance of the BSGA* algorithm, as shown in Figures 8–10, the A* algorithm, GA* algorithm, and BSGA* algorithm were executed 50 times on the maps of 10 × 10, 20 × 20, and 30 × 30, respectively. It can be seen that, in the case of large obstacles between the starting point and the target point, the BSGA* algorithm searches fewer nodes compared to the A* algorithm and the GA* algorithm.



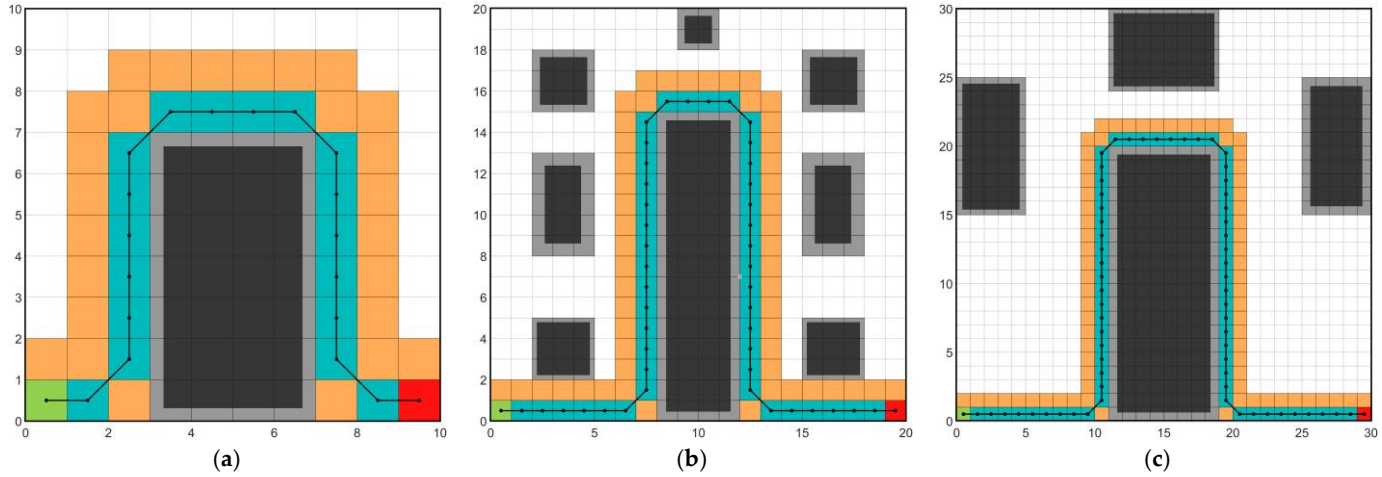
(**a**) (**b**) (**c**)

Figure 8. Performance of A* algorithm with a large area of obstacles: (**a**) Map size is 10 × 10. (**b**) Map size is 20 × 20. (**c**) Map size is 30 × 30.



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

Figure 9. Performance of GA* algorithm with a large area of obstacles: (**a**) Map size is 10 × 10. (**b**) Map size is 20 × 20. (**c**) Map size is 30 × 30.



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

Figure 10. Performance of BSGA* algorithm with a large area of obstacles: (**a**) Map size is 10 × 10. (**b**) Map size is 20 × 20. (**c**) Map size is 30 × 30.

According to Table 3, it can be seen that, on the 10 × 10 grid map, compared with the A* and GA* algorithms, the calculation time of BSGA* is reduced by 61.44% and 21.04% and the number of search nodes is reduced by 28.13% and 9.80%, respectively, but there is no significant advantage in the total turning angle. On the 20 × 20 grid map, compared with the A* algorithm and GA* algorithm, the calculation time of BSGA* is reduced by 59.48% and 41.25%, the number of search nodes is reduced by 45.16% and 28.17%, and the total turning angle is reduced by 20% and 20%, respectively. On the 30 × 30 grid map, compared with the A* algorithm and GA* algorithm, the calculation time of BSGA* is reduced by 80.94% and 69.73% and the number of search nodes is reduced by 64.80% and 49.64%, respectively. The A* algorithm makes several turns in the path after bypassing the obstacles, and the total turning angle rises rapidly, while the total turning angle of the BSGA* algorithm and GA* algorithm is relatively more stable. Although the BSGA* algorithm has obvious advantages over the GA* and A* algorithms in search efficiency, the path length will be slightly longer and the advantages in total turning angle are not obvious.

Table 3. Comparison of the paths generated by the A*, GA*, and BSGA* algorithms under different grid maps, the values of average calculation time are reported as mean ± standard deviation for 50 executions.

Map Size	Algorithm	Average Calculation Time/ms	Number of Searched Nodes	Total Angle/°	Path Length
10 × 10	A*	574.94 ± 12.27	64	450	19.49
	GA*	280.79 ± 4.48	51	270	20.07
	BSGA*	221.71 ± 2.42	46	360	20.66
20 × 20	A*	1593.84 ± 18.83	186	450	39.63
	GA*	1099.21 ± 7.24	142	450	42.56
	BSGA*	645.77 ± 7.24	102	360	46.66
30 × 30	A*	6192.52 ± 38.80	392	810	56.11
	GA*	3898.71 ± 27.79	274	405	62.46
	BSGA*	1180.20 ± 4.77	138	360	66.66

3.3. Multi-Layer Turning Point Filtering Strategy

In order to further reduce the total turning angle and improve smoothness, this study proposed an optimization strategy based on turning point filtering. The specific procedures are as follows:

(1) Store the path points $Grid\_node_i(i = 1, 2, \ldots)$ obtained by BSGA* algorithm in the *CloseList_Path* and store the obstacles and environment boundaries in the *Obstacle_nodes*. Store the $Start\_node = (X_s, Y_s)$ and the $target\_node = (X_e, Y_e)$ in the *Opt_path*.

(2) Calculate the slope k_1 and k_2 of $Start\_node$ and $Grid\_node_1$, $Start\_node$ and $Grid\_node_2$ respectively. If $k_1 = k_2$, then $Grid\_node_1$ is not a turning point, continue to calculate the slope k_i of $Start\_node$ and $Grid\_node_i$ in turn until $k_1 \neq k_i$, then $Grid\_node_{i-1}$ is the turning point. $Grid\_node_i$ will be stored in the set of *Turn_node*s, then repeat the above process from $Grid\_node_i$ until it reaches the target point.

(3) Connect the previous node $Grid\_node_{i-2}$ of the $Grid\_node_{i-1}$ with the next node $Grid\_node_i$. If the connection between them passes through the obstacle, then $Grid\_node_{i-1}$ will be stored in *Opt_path*. If the connection does not pass through the obstacle, $Grid\_node_{i-1}$ will be deleted in *Turn_node*s. Judge all the turning points in *Turn_node*s in turn according to this step.

(4) Repeat steps (2) and (3) until the connection between the previous node and the next node of any of the turns in *Turn_node*s passes through the obstacle and connects the nodes in *Opt_path* to generate the optimized path, then the algorithm stops.

To verify the effectiveness of the turning point filtering strategy, as shown in Figures 11 and 12, 50 executions were conducted, respectively, to compare the A*, BSGA*, and improved BSGA* algorithms under grid maps with sizes of 15 × 15 and 30 × 30. The

purple nodes in Figures 11c and 12c are the filtered turning points. As can be seen from Table 4, the calculation time of improved BSGA* is reduced by 62.45% compared with the A* algorithm and improved by 13.79% compared with the BSGA* algorithm in the 15 × 15 grid map. This is because improved BSGA* is implemented by re-performing the turning point filtering on the paths planned by the BSGA* algorithm, so the calculation time is slightly longer. Compared with the A* and BSGA* algorithms, the total turning angle of paths planned by improved BSGA* is reduced by 89.76% and 86.34%, respectively, the path length is reduced by 1.13. In the 30 × 30 grid map, compared with A* and BSGA*, the calculation time of improved BSGA* is reduced by 92.28% and that of BSGA* is improved by 7.26%, the total turning angle of the path is reduced by 87.89% and 81.83%, the path length is reduced by 2.33, respectively. Therefore, it can be seen that the improved BSGA* path has better smoothness and is more suitable for the driving process of the DDMR.

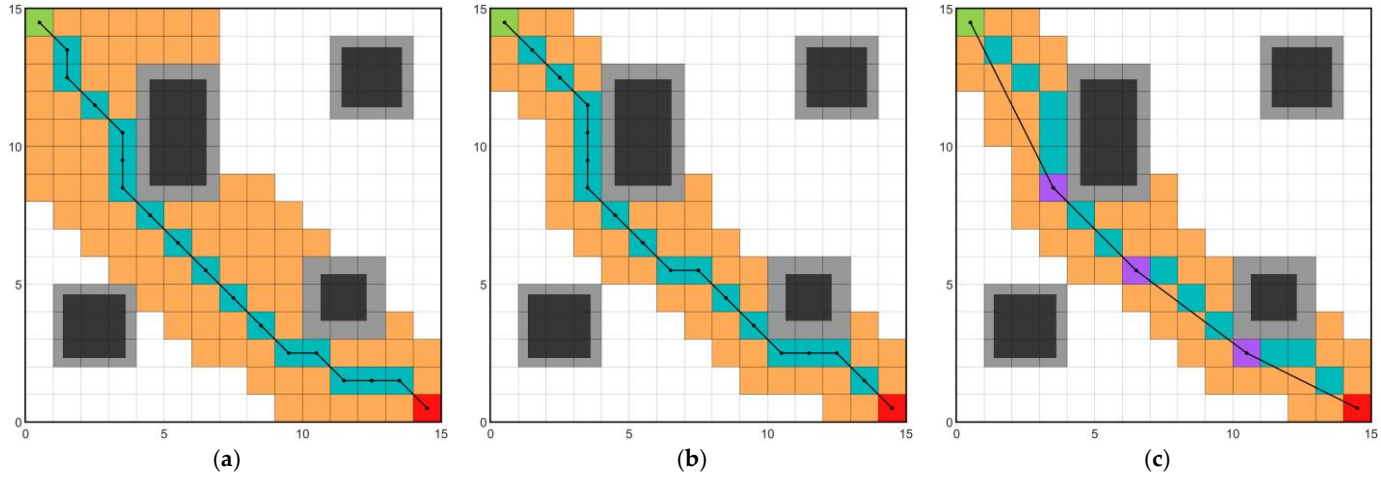


Figure 11. Performance of turning point filtering strategies in a 15 × 15 map: (**a**) A*. (**b**) BSGA*. (**c**) Improved BSGA*.

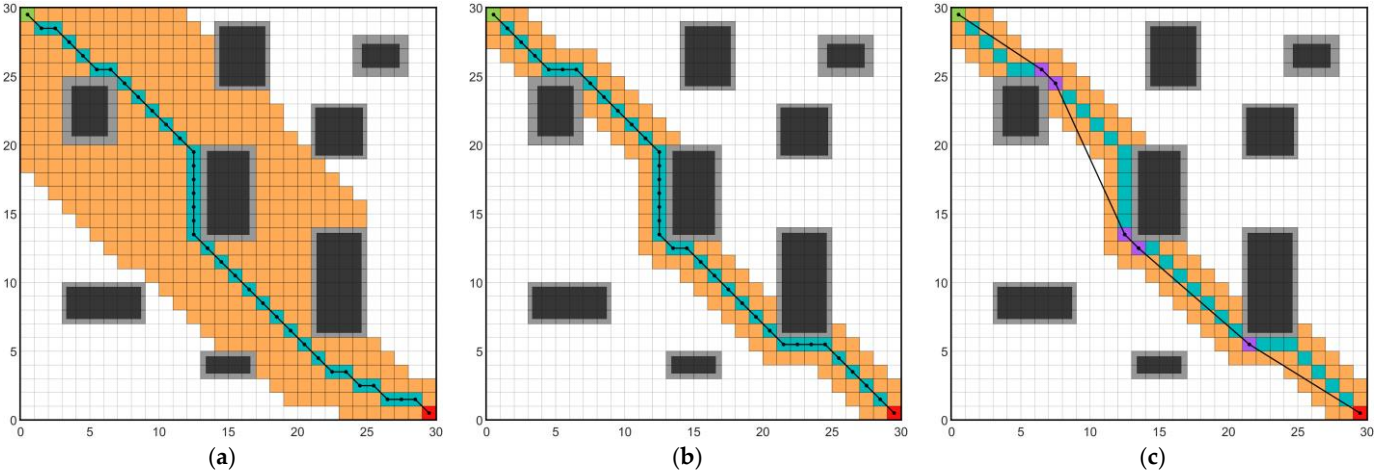


Figure 12. Performance of turning point filtering strategies in a 30 × 30 map: (**a**) A*. (**b**) BSGA*. (**c**) Improved BSGA*.

Table 4. Comparison of paths generated by A*, BSGA*, and improved BSGA* algorithms under 15 × 15 and 30 × 30 grid maps; the values of average calculation time are reported as mean ± standard deviation for 50 executions.

Map Size	Algorithm	Average Calculation Time/ms	Number of Searched Nodes	Total Angle/°	Path Length
15 × 15	A*	975.49 ± 16.29	95	360	21.55
	BSGA*	321.92 ± 7.27	63	270	21.55
	Improved BSGA*	366.31 ± 8.35	63	36.87	20.42
30 × 30	A*	12,316.77 ± 99.37	388	540	44.52
	BSGA*	886.42 ± 12.69	131	360	44.52
	Improved BSGA*	950.80 ± 14.47	131	65.42	42.19

4. Simulation and Experiments

4.1. Performance Comparison of Different Algorithms

In order to verify the performance of the improved BSGA* algorithm, the 15 × 15 and 30 × 30 grid maps constructed in Section 3.3 were used to compare the improved BSGA* algorithm with the ACO algorithm [40,41], D* lite algorithm [42], and GA [43]. The ACO algorithm [44] uses pheromone as the medium to transmit the information learned by ants in the process of moving, so that ants can move to the area with high pheromone concentration spontaneously. The parameter settings of the ACO algorithm are shown in Table 5. The GA [45] retains individuals according to their fitness by simulating the biological evolution process and updates the population through genetic and mutation operations. The parameters settings of GA are shown in Table 6. The D* lite algorithm uses the same heuristic function as the A* algorithm to guide the search [46], but the difference is that the D* lite algorithm first takes the target point as the starting point for reverse search to obtain the shortest path information, and can use the previously obtained information to reduce the search scope when it is necessary to re-plan during a later process.

Table 5. Setting the parameters of ant colony algorithm.

Parameters	Value
Maximum number of iterations	200
Number of ants	50
The expectation heuristic factor	1.5
The pheromone heuristic factor	7
The evaporation coefficient of pheromone	0.3
Initial amount of pheromone concentration	1
The constant coefficient	1

Table 6. Setting of genetic algorithm parameters.

Parameters	Value
Population size	500
Number of generations	200
Crossover probability	0.8
Mutation probability	0.2

Figure 13a,b shows the paths generated by the above-mentioned various algorithms on the 15 × 15 and 30 × 30 grid maps, respectively. It can be seen that, compared with the ACO algorithm, GA, and the D* lite algorithm, the path generated by the improved BSGA* algorithm is smoother and the total turning angle is smaller. It is worth noting that, due to the randomness of the paths generated by the ACO algorithm and GA, the results of each execution may be different. For better comparison, these three algorithms were executed

50 times, respectively, on the above two grid maps; the results are shown in Table 7. It can be seen that, on the 15 × 15 grid map, the improved BSGA* algorithm is 69.84%, 5.81%, and 85.97% lower than the ACO algorithm, 69.76%, 92.50%, and 6.20% lower than the GA, and 89.45%, 5.26%, and 86.34% lower than the D* lite algorithm in terms of calculation time, path length, and total turning angle, respectively. On the 30 × 30 grid map, the improved BSGA* algorithm is 93.74%, 5.45%, and 80.87% lower than the ACO algorithm, 74.34%, 92.64%, and 5.30% lower than the GA, and 86.82%, 5.30%, and 79.23% lower than the D* lite algorithm in these three aspects, respectively. From the above data, it can be seen that the improved BSGA* algorithm has a superior performance compared to the ACO algorithm, GA, and the D* lite algorithm. Therefore, the paths generated by the improved BSGA* algorithm are more suitable for DDMRs.

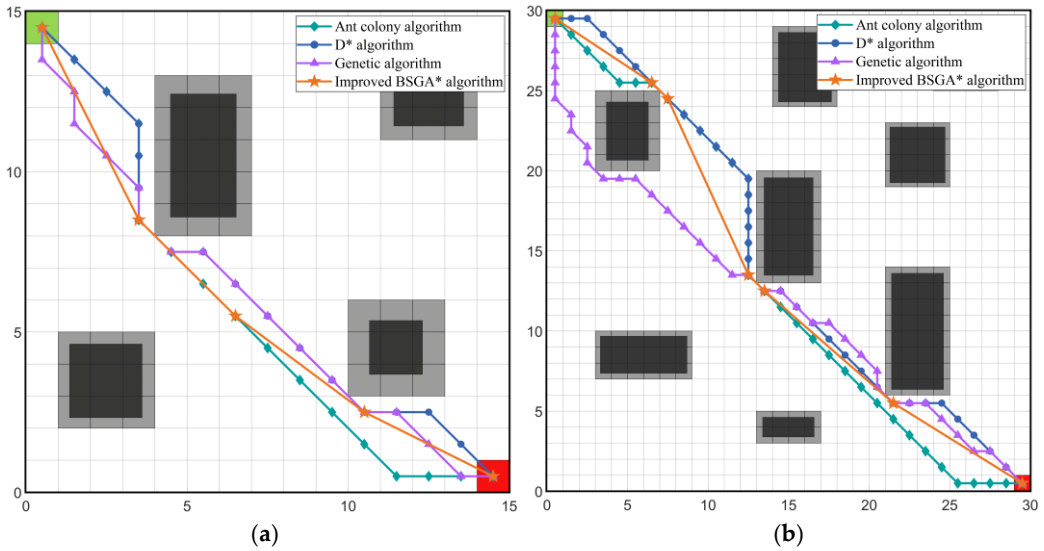


Figure 13. Comparison of paths generated by ACO, GA, D* lite, and improved BSGA* algorithms under different maps: (**a**) Map size is 15 × 15; (**b**) Map size is 30 × 30.

Table 7. Comparison of paths generated by ACO, GA, D* lite, and improved BSGA* algorithms under 15 × 15 and 30 × 30 grid maps; the values of average calculation time are reported as mean ± standard deviation for 50 executions.

Map Size	Algorithm	Average Calculation Time/ms	Total Angle/°	Path Length
15 × 15	ACO	1214.73 ± 89.46	262.80 ± 78.65	21.68 ± 0.23
	GA	1211.47 ± 156.62	491.40 ± 21.75	21.77 ± 0.45
	D* lite	3473.67 ± 69.46	270.00	21.55
	Improved BSGA*	366.31 ± 8.35	36.87	20.42
30 × 30	ACO	15,190.43 ± 376.30	342.00 ± 106.49	44.62 ± 0.27
	GA	3705.64 ± 338.03	888.30 ± 141.11	45.77 ± 0.77
	D* lite	7217.43 ± 84.26	315.00	44.55
	Improved BSGA*	950.80 ± 14.47	65.42	42.19

Figures 14 and 15 are the column graphs and error bars drawn according to the above comparison experiments. Since both the D* lite algorithm and improved BSGA* algorithm are heuristic algorithms, the optimal paths generated in each execution are the same, so the standard deviations of path length and total turning angle are 0 in 50 executions. The paths generated by the ACO algorithm and the GA have randomness. As can be seen from Figures 14 and 15, after enough iterations, although the path generated by the ACO algorithm and GA is close to the improved BSGA* algorithm in length, the standard deviation of total turning angle, however, is larger due to the randomness of path point

selection. This means that both algorithms are not stable enough in terms of path smoothing. In contrast, the paths generated by the improved BSGA* algorithm are not only smoother, but also the optimal path generated is unique and performs more stably when the grid map, starting point, and target point are determined.

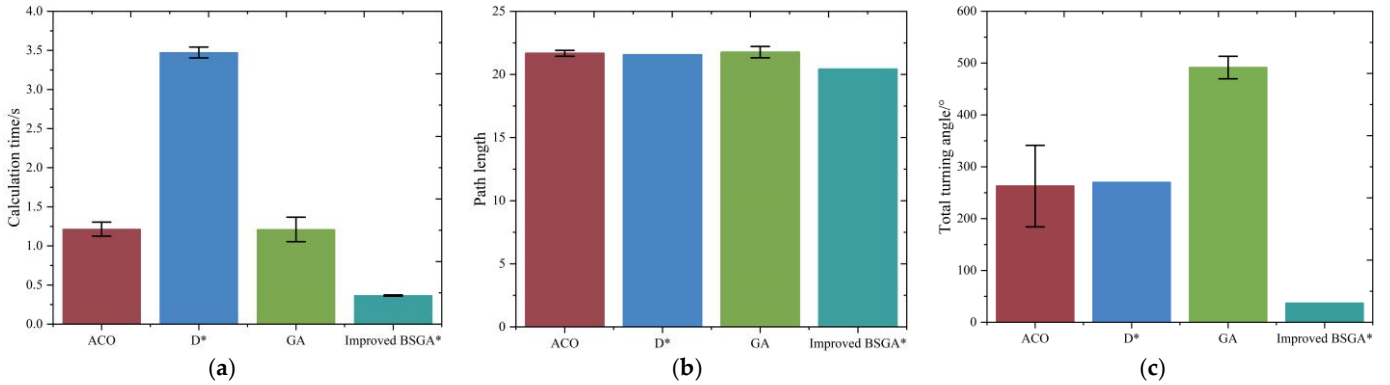


Figure 14. Column graphs and error bars of executing time, path length, and total turning angle in 15 × 15 map: (**a**) Executing time; (**b**) Path length; (**c**) Total turning angle.

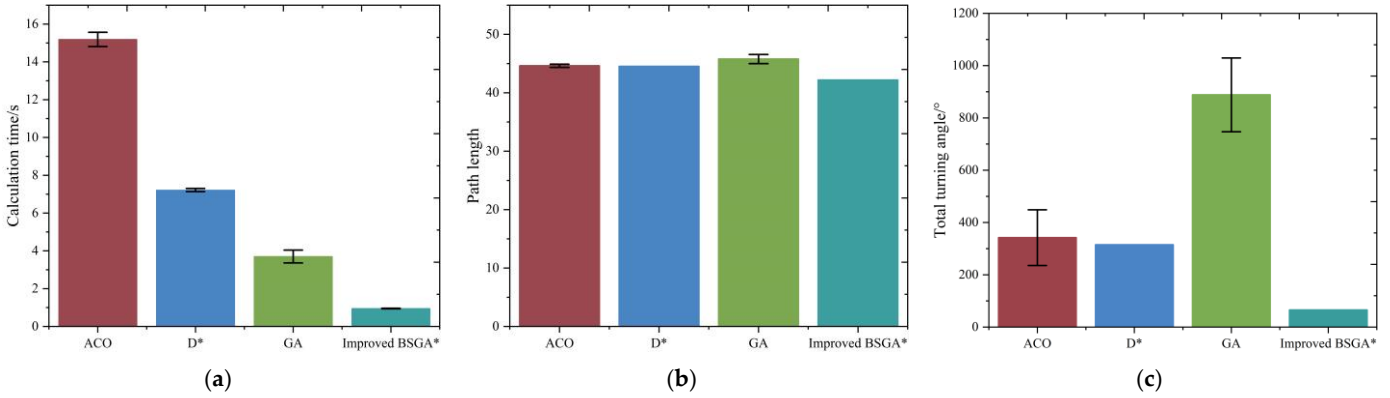


Figure 15. Column graphs and error bars of executing time, path length, and total turning angle in 30 × 30 map: (**a**) Executing time; (**b**) Path length; (**c**) Total turning angle.

4.2. Comparison in Real Environments

In order to verify the effectiveness of the improved BSGA* algorithm in real environments, a LEO ROS mobile robot is selected for the path-planning experiment. As shown in Figure 16, rubber wheels are adopted as the front wheels and two DC three-phase brushless motors are equipped, which can realize the differential control of both sides of the driving wheels. The rear wheels use omnidirectional wheels to achieve turning. The robot is equipped with a US-015 ultrasonic module, which can detect obstacles with a measuring accuracy of 0.1 cm + 1%. In addition, the robot uses EAI G1 LIDAR to test the surrounding environment, obtains obstacle information based on the triangulation principle, constructs environment maps using the *Gmapping* SLAM algorithm, saves them with *map_server*, and generates grid maps through the *map_2d* function package. The AMCL algorithm is used to determine the initial position and pose of the robot before navigation.

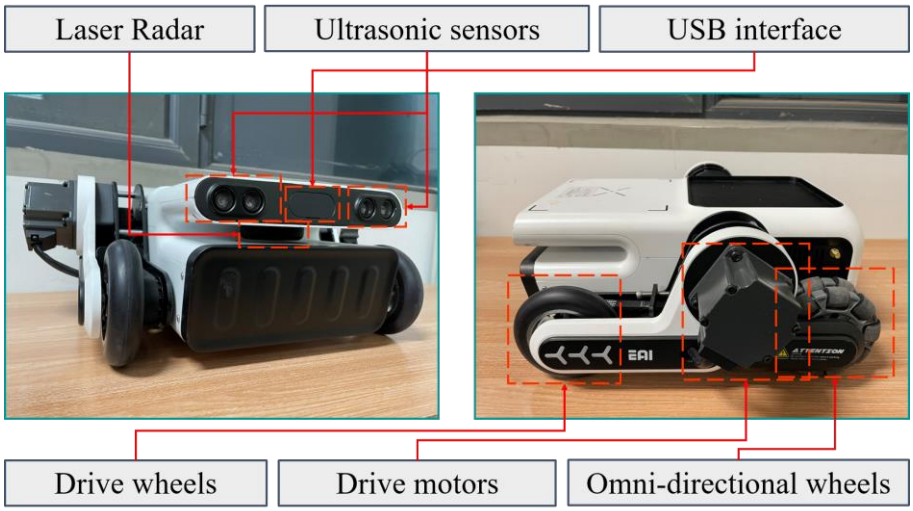
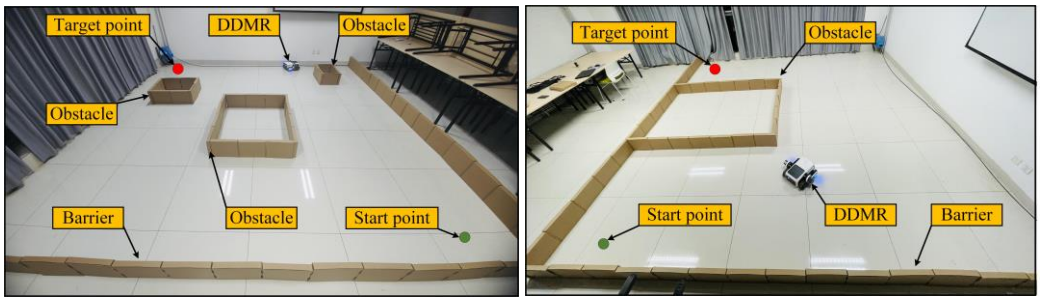


Figure 16. Main structure of the LEO ROS robot.

The global path-planning algorithms embedded in the LEO ROS mobile robots exist in the form of plug-ins in the *move_base* package. By creating and configuring the parameter file *base_global_planner_param.yaml* writing the *GlobalPlanner* class, the selection of different global path-planning algorithms can be realized. In order to apply the improved BSGA* algorithm to the robot, *BSGAstar_global_planner* plug-in is written and added into the *move_base* package based on the current *GlobalPlanner*, and the algorithm was implemented by modifying and configuring the parameter file *move_base_params.yaml*.

The LEO ROS robot is 360 mm in length, 455 mm in width, and 160 mm in height. The LIDAR is 100 mm from the ground. Based on the above size, 360 × 260 × 40 mm cardboard boxes are used to build the experimental environment, as shown in Figure 17. The edge of the square experimental environment consists of 15 cardboard boxes placed side by side. Figure 18a shows environment 1, the starting point and the target point are in a diagonal position, and there are three obstacle areas randomly placed in the environment. Figure 17b shows environment 2, there is a large area of obstacles between the starting point and the target point. The environmental map was constructed using the *Gmapping* SLAM algorithm in the ROS system and displayed on the *rivz* platform in real time. The size of the grid constructed by the robot was set to 360 mm × 360 mm. As the construction of environments is affected by many factors, such as the smoothness of driving, it is difficult to be completely consistent with the actual environment.



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

Figure 17. Experimental environments: (**a**) Environment 1; (**b**) Environment 2.

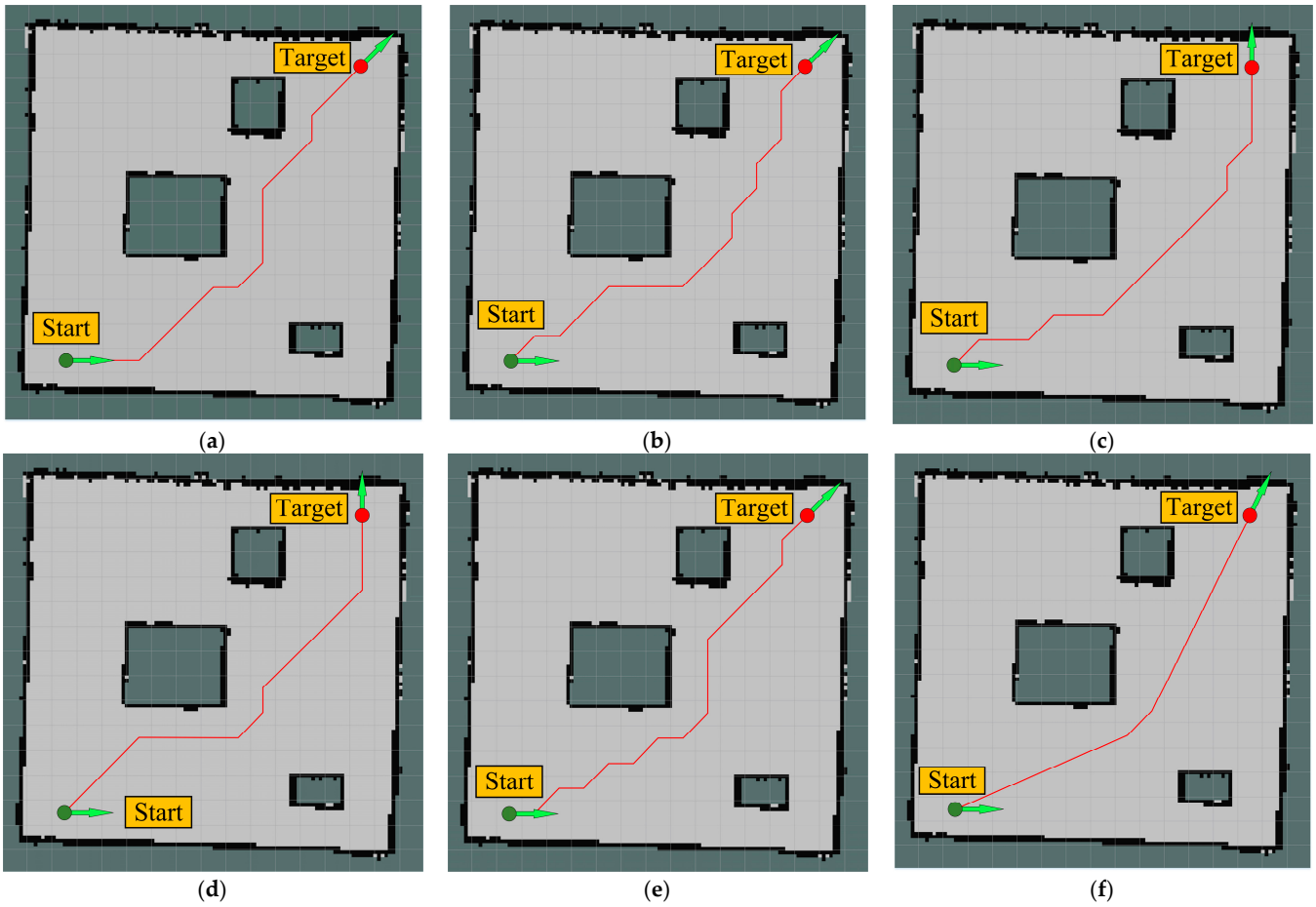


Figure 18. Comparison of paths generated by Dijkstra, A*, and improved BGA* algorithms in environment 1: (**a**) Dijkstra; (**b**) A*; (**c**) D* lite; (**d**) ACO; (**e**) GA; (**f**) Improved BSGA*.

In order to verify the performance of the improved BSGA* algorithm proposed in this study, it is compared with the Dijkstra algorithm, A* algorithm, D* lite algorithm, ACO algorithm, and GA, respectively, in two environments. The above algorithms were executed 10 times in both environments, respectively. Figure 18 shows the paths generated by the above six algorithms in environment 1 and Figure 19 shows the paths generated by the above six algorithms in environment 2. The green arrow in the figure represents the current orientation of the robot, and the red line segment represents the driving path. It can be seen that the paths generated by the improved BSGA* algorithm are smoother in both maps, mainly because the redundant turning points can be removed by using a multi-layer turning point filtering strategy.

Tables 8 and 9 show the comparison of the calculation times of six algorithms and the driving time of the LEO ROS robot in environments 1 and 2.

Table 8. The comparison of the path-planning time of six algorithms and travel time of the LEO ROS robot in environment 1; the values of average calculation time are reported as mean ± standard deviation for 10 executions.

Algorithm	Average Calculation Time/ms	Actual Driving Time/s
Dijkstra	1134.99 ± 12.73	74.82 ± 0.53
A*	981.96 ± 19.28	101.60 ± 0.35
D* lite	4480.39 ± 138.67	54.76 ± 0.35
ACO	3940.42 ± 87.47	58.88 ± 0.41
GA	1049.15 ± 51.95	110.66 ± 0.48
Improved BSGA*	299.69 ± 13.21	35.02 ± 0.28

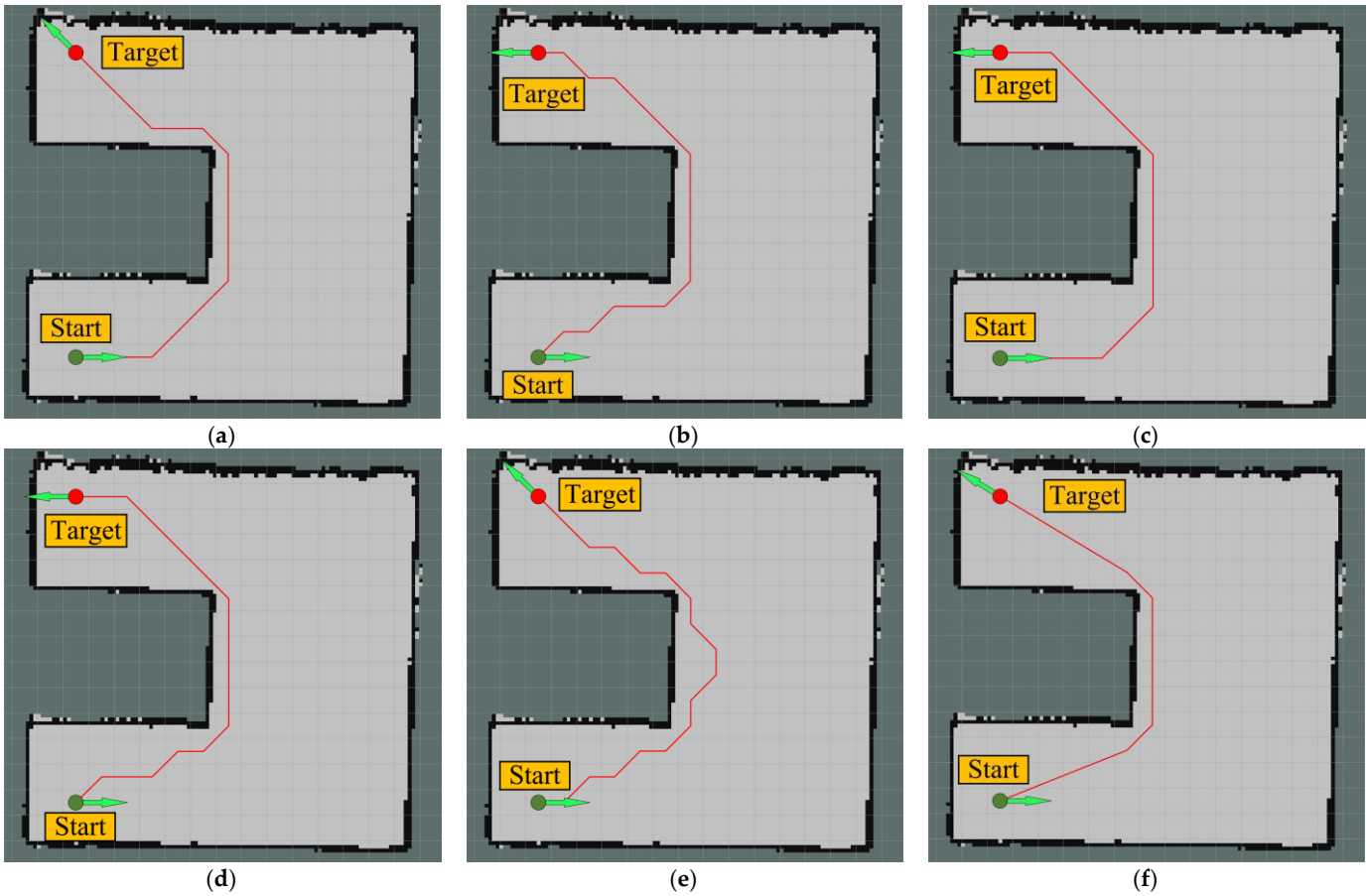


Figure 19. Comparison of paths generated by Dijkstra, A*, and improved BGA* algorithms in environment 2: (**a**) Dijkstra; (**b**) A*; (**c**) D* lite; (**d**) ACO; (**e**) GA; (**f**) Improved BSGA*.

Table 9. The comparison of the path-planning time of six algorithms and travel time of the LEO ROS robot in environment 2; the values of average calculation time are reported as mean ± standard deviation for 10 executions.

Algorithm	Average Calculation Time/ms	Actual Driving Time/s
Dijkstra	1454.97 ± 13.16	60.28 ± 0.34
A*	626.82 ± 8.14	93.61 ± 0.54
D* lite	4976.84 ± 114.65	78.14 ± 0.38
ACO	4426.67 ± 245.95	77.39 ± 0.51
GA	1940.80 ± 138.87	146.24 ± 0.62
Improved BSGA*	324.91 ± 8.02	52.88 ± 0.34

It can be obtained that, compared with the other five algorithms in environment 1, the calculation times of the improved BSGA* algorithm are reduced by 73.60%, 69.48%, 93.31%, 92.39%, and 71.43%, respectively, in turn. In environment 2, the corresponding data are 77.67%, 48.17%, 93.47%, 92.66%, and 83.26%, respectively, in turn. This is because the Gaussian function is used to dynamically weigh the heuristic function of the A* algorithm, and the BS search structure is applied to the A* algorithm, which can effectively reduce the redundant search nodes. From Table 9, it can be concluded that, compared with the other five algorithms in environment 1, the driving time of the LEO ROS robot using the improved BSGA* algorithm is reduced by 53.19%, 65.53%, 36.05%, 40.52%, and 68.35%, respectively, in turn. In environment 2, the corresponding data are 12.28%, 43.51%, 32.33%, 31.67%, and 63.84%, respectively, in turn. This is due to the fact that the LEO ROS robot takes relatively more time during the turning process, whereas the improved BSGA* algorithm

generates smoother and shorter paths, thus effectively reducing the driving time. Therefore, in different types of environments, the improved BSGA* algorithm shows better results than the other five algorithms, which is beneficial for improving the service life of DDMRs, reducing motor loss, and improving driving efficiency.

5. Conclusions

In this paper, an improved BSGA* algorithm is proposed based on the traditional A* algorithm and the kinematic analysis of DDMRs. By introducing a Gaussian function to dynamically weigh the heuristic function and optimizing it using the BS structure, the algorithm performance in the face of different types of obstacles is effectively improved. In addition, a multi-layer turning point filtering strategy is proposed to reduce the redundant nodes and improve the smoothness of the path. The simulation results show that the improved BSGA* algorithm has a better performance than the ACO algorithm, D* lite algorithm, and GA in different maps. Finally, the improved BSGA* algorithm is applied to the LEO ROS mobile robot, and two different environments are built. The experimental results show that the improved BSGA* algorithm is more suitable to provide global path guidance for DDMRs than the Dijkstra algorithm, A* algorithm, ACO algorithm, D* algorithm, and GA. As a static planning method, the improved BSGA* cannot deal with dynamic obstacles, so it should be combined with a local obstacle avoidance algorithm to ensure the safety of driving in future work.

Author Contributions: Conceptualization, M.Y.; Methodology, M.Y.; Software, M.Y.; Investigation, M.Y.; Writing—original draft, M.Y.; Writing—review and editing, M.Y., H.D. and X.F.; Funding acquisition, H.D.; Resources, X.F.; Project administration, X.F.; Form analysis, P.L.; Data curation, Y.L.; Supervision, H.L.; Visualization, H.L.; Validation, H.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Key Technology Research and Development Program of Shandong, grant number 2022CXGC010101 and the Key Technology Research and Development Program of Shandong, grant number 2019JZZY010443.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare that they have no known competing financial interest or personal relationship that could have appeared to influence the work reported in this paper.

References

1. Stefek, A.; Pham, T.V.; Krivanek, V.; Pham, K.L. Energy Comparison of Controllers Used for a Differential Drive Wheeled Mobile Robot. *IEEE Access* **2020**, *8*, 170915–170927. [CrossRef]
2. Felix-Rendon, J.; Bello-Robles, J.C.; Fuentes-Aguilar, R.Q. Control of differential-drive mobile robots for soft object deformation. *ISA Trans.* **2021**, *117*, 221–233. [CrossRef]
3. Xie, H.; Zheng, J.; Chai, R.; Nguyen, H.T. Robust tracking control of a differential drive wheeled mobile robot using fast nonsingular terminal sliding mode. *Comput. Electr. Eng.* **2021**, *96*, 107488. [CrossRef]
4. Haider, M.H.; Wang, Z.; Khan, A.A.; Ali, H.; Zheng, H.; Usman, S.; Kumar, R.; Bhutta, M.U.M.; Zhi, P. Robust mobile robot navigation in cluttered environments based on hybrid adaptive neuro-fuzzy inference and sensor fusion. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 9060–9070. [CrossRef]
5. Raikwar, S.; Fehrmann, J.; Herlitzius, T. Navigation and control development for a four-wheel-steered mobile orchard robot using model-based design. *Comput. Electron. Agric.* **2022**, *202*, 107410. [CrossRef]
6. Bai, Y.; Zhang, B.; Xu, N.; Zhou, J.; Shi, J.; Diao, Z. Vision-based navigation and guidance for agricultural autonomous vehicles and robots: A review. *Comput. Electron. Agric.* **2023**, *205*, 107584. [CrossRef]
7. Jian, Z.; Zhang, S.; Chen, S.; Nan, Z.; Zheng, N. A Global-Local Coupling Two-Stage Path Planning Method for Mobile Robots. *IEEE Robot. Autom. Lett.* **2021**, *6*, 5349–5356. [CrossRef]
8. Mandloi, D.; Arya, R.; Verma, A.K. Unmanned aerial vehicle path planning based on A* algorithm and its variants in 3d environment. *Int. J. Syst. Assur. Eng. Manag.* **2021**, *12*, 990–1000. [CrossRef]

9. Tang, G.; 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. *Ieee Access* **2021**, *9*, 59196–59210. [CrossRef]
10. Liao, B.; Wan, F.; Hua, Y.; Ma, R.; Zhu, S.; Qing, X. F-RRT*: An improved path planning algorithm with improved initial solution and convergence rate. *Expert Syst. Appl.* **2021**, *184*, 115457. [CrossRef]
11. Miao, C.; Chen, G.; Yan, C.; Wu, Y. Path planning optimization of indoor mobile robot based on adaptive ant colony algorithm. *Comput. Ind. Eng.* **2021**, *156*, 107230. [CrossRef]
12. Sarkar, R.; Barman, D.; Chowdhury, N. Domain knowledge based genetic algorithms for mobile robot path planning having single and multiple targets. *J. King Saud Univ. Comput. Inf. Sci.* **2022**, *34*, 4269–4283. [CrossRef]
13. Lavalle, S.M. *Planning Algorithms*; Cambridge University Press: Cambridge, UK, 2006; ISBN 9780511546877.
14. Guo, J.; Xia, W.; Hu, X.; Ma, H. Feedback RRT* algorithm for UAV path planning in a hostile environment. *Comput. Ind. Eng.* **2022**, *174*, 108771. [CrossRef]
15. Wang, J.; Li, B.; Meng, M.Q.H. Kinematic Constrained Bi-directional RRT with Efficient Branch Pruning for robot path planning. *Expert Syst. Appl.* **2021**, *170*, 114541. [CrossRef]
16. Hou, W.; Xiong, Z.; Wang, C.; Chen, H. Enhanced ant colony algorithm with communication mechanism for mobile robot path planning. *Robot. Auton. Syst.* **2022**, *148*, 103949. [CrossRef]
17. Tuncer, A.; Yildirim, M. Dynamic path planning of mobile robots with improved genetic algorithm. *Comput. Electr. Eng.* **2012**, *38*, 1564–1572. [CrossRef]
18. Dijkstra, E.W. A note on two problems in connexion with graphs. *Numer. Math.* **1959**, *1*, 269–271. [CrossRef]
19. Hart, P.E.; Nilsson, N.J.; Raphael, B. A Formal Basis for the Heuristic Determination. *IEEE Trans. Syst. Sci. Cybern.* **1968**, *2*, 28–29.
20. Zhang, Z.; Jiang, J.; Wu, J.; Zhu, X. Efficient and optimal penetration path planning for stealth unmanned aerial vehicle using minimal radar cross-section tactics and modified A-Star algorithm. *ISA Trans.* **2023**, *134*, 42–57. [CrossRef]
21. Liu, Z.; Liu, H.; Lu, Z.; Zeng, Q. A Dynamic Fusion Pathfinding Algorithm Using Delaunay Triangulation and Improved A-Star for Mobile Robots. *IEEE Access* **2021**, *9*, 20602–20621. [CrossRef]
22. Jiang, H.; Sun, Y. Research on global path planning of electric disinfection vehicle based on improved A* algorithm. *Energy Rep.* **2021**, *7*, 1270–1279. [CrossRef]
23. Liu, H.; Zhang, Y. ASL-DWA: An Improved A-Star Algorithm for Indoor Cleaning Robots. *IEEE Access* **2022**, *10*, 99498–99515. [CrossRef]
24. Li, J.; Liao, C.; Zhang, W.; Fu, H.; Fu, S. UAV Path Planning Model Based on R5DOS Model Improved A-Star Algorithm. *Appl. Sci.* **2022**, *12*, 11338. [CrossRef]
25. Zhang, H.; Li, M.; Yang, L. Safe Path Planning of Mobile Robot Based on Improved A* Algorithm in Complex Terrains. *Algorithms* **2018**, *11*, 44. [CrossRef]
26. Sang, H.; You, Y.; Sun, X.; Zhou, Y.; Liu, F. The hybrid path planning algorithm based on improved A* and artificial potential field for unmanned surface vehicle formations. *Ocean. Eng.* **2021**, *223*, 108709. [CrossRef]
27. Cui, S.; Chen, Y.; Li, X. A Robust and Efficient UAV Path Planning Approach for Tracking Agile Targets in Complex Environments. *Machines* **2022**, *10*, 931. [CrossRef]
28. Dang, S.T.; Dinh, X.M.; Kim, T.D.; Xuan, H.L.; Ha, M. Adaptive Backstepping Hierarchical Sliding Mode Control for 3-Wheeled Mobile Robots Based on RBF Neural Networks. *Electronics* **2023**, *12*, 2345. [CrossRef]
29. Latombe, J.C. *Robot Motion Planning*; Springer: Berlin/Heidelberg, Germany, 1991.
30. Klančar, G.; Matko, D.; Blažič, S. A control strategy for platoons of differential drive wheeled mobile robot. *Robot. Auton. Syst.* **2011**, *59*, 57–64. [CrossRef]
31. Li, Y.; Jin, R.; Xu, X.; Qian, Y.; Wang, H.; Xu, S.; Wang, Z. A Mobile Robot Path Planning Algorithm Based on Improved A* Algorithm and Dynamic Window Approach. *IEEE Access* **2022**, *10*, 57736–57747. [CrossRef]
32. Duchoň, F.; Babinec, A.; Kajan, M.; Beňo, P.; Florek, M.; Fico, T.; Jurišica, L. Path Planning with Modified a Star Algorithm for a Mobile Robot. *Procedia Eng.* **2014**, *96*, 59–69. [CrossRef]
33. Hong, Z.; Sun, P.; Tong, X.; Pan, H.; Zhou, R.; Zhang, Y.; Han, Y.; Wang, J.; Yang, S.; Xu, L. Improved A-Star Algorithm for Long-Distance Off-Road Path Planning Using Terrain Data Map. *ISPRS Int. J. Geo-Inf.* **2021**, *10*, 785. [CrossRef]
34. Zhao, H.; Zhang, B.; Sun, J.; Yang, L.; Yu, H. Spot-welding path planning method for the curved surface workpiece of body-in-white based on a memetic algorithm. *Int. J. Adv. Manuf. Technol.* **2021**, *117*, 3083–3100. [CrossRef]
35. Liu, C.; Mao, Q.; Chu, X.; Xie, S. An Improved A-Star Algorithm Considering Water Current, Traffic Separation and Berthing for Vessel Path Planning. *Appl. Sci.* **2019**, *9*, 1057. [CrossRef]
36. Flores-Caballero, G.; Rodriguez-Molina, A.; Aldape-Perez, M.; Villarreal-Cervantes, M.G. Optimized Path-Planning in Continuous Spaces for Unmanned Aerial Vehicles Using Meta-Heuristics. *IEEE Access* **2020**, *8*, 176774–176788. [CrossRef]
37. Li, C.; Huang, X.; Ding, J.; Song, K.; Lu, S. Global path planning based on a bidirectional alternating search A* algorithm for mobile robots. *Comput. Ind. Eng.* **2022**, *168*, 108123. [CrossRef]
38. Soltani, A.R.; Tawfik, H.; Goulermas, J.Y.; Fernando, T. Path planning in construction sites: Performance evaluation of the Dijkstra, A∗, and GA search algorithms. *Adv. Eng. Inform.* **2002**, *16*, 291–303. [CrossRef]
39. Ryu, H.; Chung, W.K. Local map-based exploration using a breadth-first search algorithm for mobile robots. *Int. J. Precis. Eng. Manuf.* **2015**, *16*, 2073–2080. [CrossRef]

40. Dorigo, M.; Gambardella, L.M. Ant colony system: A cooperative learning approach to the traveling salesman problem. *Ieee Trans. Evol. Comput.* **1997**, *1*, 53–66. [CrossRef]
41. Dorigo, M.; Maniezzo, V.; Colorni, A. Ant system: Optimization by a colony of cooperating agents. *IEEE Trans. Syst. Man Cybern. Part B Cybern.* **1996**, *26*, 29–41. [CrossRef]
42. Stentz, A. Optimal and Efficient Path Planning for Partially-Known Environments. In Proceedings of the 1994 IEEE International Conference on Robotics and Automation, San Diego, CA, USA, 8–13 May 1994; pp. 3310–3317.
43. Sugihara, K.; Smith, J. Genetic Algorithms for Adaptive Motion Planning of an Autonomous Mobile Robot. In Proceedings of the IEEE International Symposium on Computational Intelligence in Robotics & Automation, Monterey, CA, USA, 10–11 July 1997.
44. Liu, J.; Anavatti, S.; Garratt, M.; Abbass, H.A. Modified continuous Ant Colony Optimisation for multiple Unmanned Ground Vehicle path planning. *Expert Syst. Appl.* **2022**, *196*, 116605. [CrossRef]
45. Raja, R.; Dutta, A.; Venkatesh, K.S. New potential field method for rough terrain path planning using genetic algorithm for a 6-wheel rover. *Robot. Auton. Syst.* **2015**, *72*, 295–306. [CrossRef]
46. Yu, J.; Yang, M.; Zhao, Z.; Wang, X.; Bai, Y.; Wu, J.; Xu, J. Path planning of unmanned surface vessel in an unknown environment based on improved D*Lite algorithm. *Ocean. Eng.* **2022**, *266*, 112873. [CrossRef]