Scholar Publishing Group
International Journal of Social Sciences and Economic Management
https://doi.org/10.38007/IJSSEM.2023.040213
ISSN 2789-715X Vol. 4, Issue 2: 96-107

Threat Cost Based Multi-level Prediction D-star Algorithm

Huiheng Suo[1,3,a], Bosi Wei[1,b*], Jian Wu[1,c*], Xie Ma[2,d], Tao Yang[2,e], Yaoyu Huang[3,f], Yicheng Lu[3,g], Xiushui Ma[3,h]

[1]*Nanchang Hangkong University, Nanchang, China*
[2]*Ningbo University of Finance & Economics, Ningbo, China*
[3]*NingboTech University, Ningbo, China*
[a]*suohuiheng@163.com,* [b]*wei_bosi@163.com,* [c]*flywujian@qq.com,* [d]*maxie@163.com,* [e]*2669673070@qq.com,* [f]3125823263@qq.com, [g]*1014190067@qq.com,* [h]*mxsh63@aliyun.com*
[*]*corresponding author*

Keywords: Path Planning, MLP D-star Algorithm, D-star Algorithm, Modeling and Simulation

Abstract: In this paper, we propose a Multi-level Prediction D-star algorithm (MLP D-star) based on threat cost to address the path planning problem of mobile robots in local unknown environments. The algorithm improves the node expansion of the D-star algorithm using a multi-level prediction structure, which avoids excessive turning points in the planned path. The cost function of this algorithm incorporates threat cost and heuristic function to prevent the issue of path crossing obstacles. Simulation results demonstrate that the improved MLP D-star algorithm has advantages in terms of real-time performance, practicality of path results, safety, and computational efficiency.

1. Introduction

The path planning of a mobile robot refers to the process of computing a safe path from the current position to a given target point [1]. As part of the autonomous exploration of mobile robots, path planning can be divided into global static path planning and local dynamic path planning based on the robot's operational state [2]. On the one hand, global static path planning involves computing a collision-free path based on prior static map information, typically using the shortest distance as a criterion for evaluating the path's quality. On the other hand, local dynamic path planning involves the robot using real-time sensor data to detect unknown obstacles and making local adjustments to the global path to avoid obstacles and move towards the target point. In 1959, Dijkstra [3] proposed the Dijkstra algorithm, which uses a greedy approach to compute the shortest path from a node to all other nodes. The algorithm is simple to implement and has low time complexity but can be time-consuming. In 1968, Hart et al. [4] extended Dijkstra's algorithm and introduced the A*

algorithm, which uses a heuristic function to guide the search direction and avoid exhaustive search in all directions. A* gradually became the mainstream path planning algorithm due to its improved overall search efficiency. However, it is not suitable for path planning in dynamic environments. To address this issue, Stantz [5] proposed a reverse incremental search algorithm called D-Star in 1994, based on A* and Dijkstra's algorithms. This algorithm is applicable to path planning problems in unknown environments [6]. Over the years, D-Star algorithm has been widely used in various mobile robots and even employed as a path search algorithm in the US Mars rover "Phoenix" [7]. In 2002, Koenig et al. [8] introduced the D* Lite algorithm, which incorporates a heuristic function to significantly improve search efficiency. However, when the forward direction of the current path is impassable, the search efficiency decreases significantly. In 1998, Lavalle et al. [9] proposed the Rapidly-Exploring Random Trees (RRT) algorithm, which is a random sampling-based search algorithm. It gradually expands the tree nodes to cover the entire map, and when the expanded node reaches the target point, the optimal path search is completed. However, the randomness of expansion in this algorithm leads to lower exploration efficiency. In 2000, Kuffner et al. [10] proposed the RRT-Connect algorithm, which utilizes two random trees for simultaneous search. In 2011, Karaman et al. [11] introduced the RRT* algorithm based on incremental sampling. In 1986, Khatib [12] proposed the Artificial Potential Field (APF) method, which describes a virtual force field. By establishing a repulsive field for obstacles and an attractive field for the target point, the robot generates a safe path through the combined effect of attraction and repulsion, enabling autonomous obstacle avoidance and reaching the target point. In 1997, Fox et al. [13] presented the Dynamic Window Approach (DWA), which utilizes sampling to predict the robot's motion using multiple combinations of linear and angular velocities. The trajectories are evaluated using a scoring function, and the optimal combination is selected to drive the robot for obstacle avoidance. This method transforms the path planning problem into a constrained optimization problem in the vector space [14].

Path planning in a partially unknown environment is a complex problem that requires robots to rapidly and accurately plan paths based on limited environmental information, as well as adapt to path replanning caused by environmental changes [15]. This technology assists robots in achieving efficient, precise, and safe movements in complex environments, as well as better accomplishing task objectives.

Currently, the commonly used path planning algorithm is the A-star algorithm. However, the A-star algorithm performs global planning only once. When environmental changes affect the path, such as encountering dynamic obstacles during traversal along the planned path, the A-star algorithm requires re-planning. Therefore, a drawback of this algorithm is its inability to utilize information generated from the previous planning to reduce computation and planning time. The D-star algorithm, also known as the Dynamic A-star algorithm, can utilize cost information from the previous planning during subsequent planning, thereby reducing computation. Consequently, it is considered a more intelligent path planning algorithm suitable for unknown environments. In this chapter, the research on autonomous exploration path planning for robots in unfamiliar environments is conducted based on the D-star algorithm. Improvements are made to the exploration strategy and cost function of the D-star algorithm, resulting in the proposal of the Threat-Cost-Based Multilayer Estimated D-star algorithm.

2. Design Principles and Drawbacks of the D-star Algorithm

2.1The Design Principles of D-star Algorithm

The D-star algorithm is a dynamic reverse path search algorithm based on the foundations of

the A-star algorithm and the Dijkstra algorithm. It is suitable for path planning in unknown environments. Given the current environmental information, the starting point, and the target point, this algorithm expands the search from the target point towards the starting point. Once the starting point is discovered, it backtracks the path based on the backtrace pointers of each node, ultimately completing the path planning from the starting point to the target point. However, when the environmental information changes and affects the nodes along the planned path, the algorithm initiates a local path replanning step. By calculating the cost values of the nodes affected by the environmental changes, the algorithm selects a better trajectory, thus achieving dynamic adaptation.

The D-star algorithm reduces the computational cost of path node backtracking through its reverse search mechanism when solving path planning problems in unknown environments. It only requires local path adjustments when dealing with dynamic obstacle threats, avoiding global replanning. This approach not only enables local obstacle avoidance but also improves the efficiency of the algorithm.

The cost function of the D-star algorithm is shown as follows:

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

Where, $h(n)$ represents the total cost value from the target point to the current node.

$$h(n) = h(o) + c(o,n) \tag{2}$$

Where, $c(o,n)$ represents the estimated cost from node o to node n. The change in cost $h(n)$ during the D-star algorithm's execution occurs in two places. Firstly, during the path search process, when the neighboring nodes of the current node are expanded, if it is possible to achieve a lower cost $h(n)$, it is updated according to Equation (2). Secondly, during the execution of the completed path planning process, when encountering obstacles, the cost $h(y)$ of obstacle nodes is modified by $insert(x, y, val)$.

The D-star algorithm consists primarily of two functions, namely the Process-State( ) function and the Modify-Cost( ) function, referred to as P and M respectively in the following text. The main purpose of the P function is to compute the optimal path cost and generate the optimal path, while the M function dynamically updates the planned path by updating the cost values of nodes in the path and modifying the table entries. The algorithm follows the execution process outlined below:

Step1: Create two lists, namely OPEN list and CLOSE list, and add the goal node to the OPEN list.

Step2: Check if the OPEN list is empty. If there are no nodes in the OPEN list, terminate the algorithm as the path planning has failed. Otherwise, check if the node corresponding to the minimum estimated cost in the OPEN list is the starting point. If true, the path planning is successful, and proceed to step 3. If false, add this node to the CLOSE list and add the neighboring nodes to the OPEN list. Continue with this step.

Step3: Traverse back to the goal node following the backtracking pointers. If there are environmental changes that affect the nodes in the path, perform local replanning by adding the previous node of the changed node to the OPEN list. Then, return to step 2. If there are no changes, proceed to step 4.

Step4: Path planning is complete.

The pseudocode is shown in Figure 1:

```
Algorithm: D_star
Data: start, goal
Result: path
t(all_of_grid) ← NEW
h(goal) ← 0
put_into_open_list(goal, h(goal))
while(t_start isnot CLOSED AND k_min != −1)
        k_min ← PROCESS_STATE( )
end
if (k_min == −1)
   exit
end
while(map_has_no_new_obs AND goal_is_not_reached)
        path ← List_PATH
end
if (goal_is_reached)
    exit
else
    b(y) → x,(x_is_obs)
    MODIFY_COST(x, y, new_c(x, y))
    while(k_min < h(y) AND k_min != −1 )
            k_min ← PROCESS_STATE( )
    end
    if (k_min == −1)
       exit
    end
end
```

Figure 1. Pseudocode Design for the D-star Algorithm

2.2 The Drawbacks of D-star Algorithm

The D-star algorithm belongs to the category of dynamic path planning algorithms. As the robot traverses the initial path, encountering obstacles allows the algorithm to perform local replanning using the previous planning information. This approach avoids redundant calculations, making the algorithm suitable for path planning in unknown environments. However, the D-star algorithm has limitations when applied in practice.

Firstly, in the D-star algorithm, the selection of neighboring nodes for expansion is limited to the 8-neighborhood directions of the current node. Additionally, the original algorithm's cost function includes the cost of moving from the current node to the next node. Consequently, the generated path often exhibits excessive turns or zigzag patterns. This phenomenon can hinder the robot's efficiency during autonomous exploration in safe areas.

Secondly, when performing path planning, the D-star algorithm strictly considers obstacles as occupying the grid cells. As a result, the generated path may be close to or even pass through

narrow gaps between adjacent obstacles. Deploying the D-star algorithm on a robot could lead to instances where the robot scrapes or collides with obstacles.

3. Threat-Cost-Based Multi-level Prediction D-star Algorithm

In this section, we focus on the expansion of nodes and the cost function in the D-star algorithm. Based on this, we propose a Threat-Cost-Based Multi-level Prediction D-star algorithm to achieve path planning for mobile robots in locally unknown environments.

3.1. Exploration Strategy of the MLP D-star Algorithm

The D-star algorithm commonly utilizes the 8 neighborhood method for node expansion, where the algorithm traverses the surrounding area from the current node. In Figure 2, the directions indicated by the black line segments represent the 8 neighborhood expansion directions. It can be observed that the D-star algorithm restricts the minimum change in path angles to $\pi/4$, resulting in frequent rotations when deploying the algorithm within a robotic system for path planning.

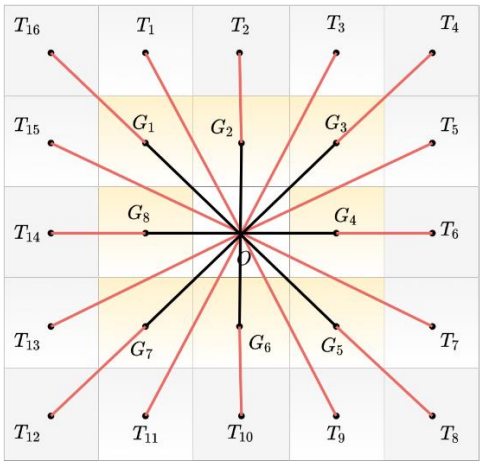


Figure 2. Schematic Diagram of the Multi-Level Predictive Structure

Based on the above description, in this section, following the neighborhood concept, the second-layer outer neighborhood of the node $T_1 : T_{16}$ is also included in the node expansion range. This is illustrated by the red line segments in Figure 2.

The definition of the multi-level predictive structure is shown in Equation (3) as follow:

$$\left\{ s_i = (x_i, y_i) \,|\, |x_i - x_O| \le 2, |y_i - y_O| \le 2 \right\} \tag{3}$$

Where, s_i represents the set of neighboring nodes of the current node, (x_O, y_O) represents the coordinates of the current node, and (x_i, y_i) represents the coordinates of the neighboring nodes of the current node.

In the multi-level predictive structure diagram, each cell's center point represents the location of the robot. Assuming the current position of the robot is at node O, when the D-star algorithm performs neighborhood expansion at the current node, it not only estimates the cost for the neighboring nodes represented by $\{G_1, G_2, \mathrm{L}\ , G_8\}$ (8-neighborhood expansion) but also includes cost estimation for the second-layer neighborhood represented by $\{T_1, T_2, \mathrm{L}\ , T_{16}\}$, where the node O is located. Considering that node O is not directly adjacent to the second-layer neighborhood, the

outer nodes of $\{G_1, G_2, \mathrm{L}\ , G_8\}$ can only be reached through the first-layer neighborhood node O. If G_1 is an obstacle node with an infinite cost estimation, the robot will be unable to reach the three outer nodes of T_1, T_{15}, T_{16}. Therefore, the cost value of the first-layer neighborhood node $\{G_1, G_2, \mathrm{L}\ , G_8\}$ directly affects the passage between node O and the outer neighborhood nodes $\{T_1, T_2, \mathrm{L}\ , T_{16}\}$, and it is numerically represented as an increase in the estimated cost for moving from node O to the outer neighborhood nodes. In light of the aforementioned scenario, the following neighborhood expansion strategy is proposed in this section:

(1) When node G_1 is an impassable obstacle, expansion of node T_1, T_{15}, T_{16} is abandoned. When node G_2 is an impassable obstacle, expansion of node T_1, T_2, T_3 is abandoned. When node G_3 is an impassable obstacle, expansion of node T_3, T_4, T_5 is abandoned. When node G_4 is an impassable obstacle, expansion of node T_5, T_6, T_7 is abandoned. When node G_5 is an impassable obstacle, expansion of node T_7, T_8, T_9 is abandoned. When node G_6 is an impassable obstacle, expansion of node T_9, T_{10}, T_{11} is abandoned. When node G_7 is an impassable obstacle, expansion of node T_{11}, T_{12}, T_{13} is abandoned. When node G_8 is an impassable obstacle, expansion of node T_{13}, T_{14}, T_{15} is abandoned.

(2) When multiple first-layer expanded neighborhood nodes from Expansion Strategy 1 are obstacles, it will impose cumulative constraints on the second-layer expanded neighborhood nodes.

The above expansion strategies increase the expansion directions by a factor of 1, reducing the minimum change in path angles in the D-star algorithm to $\pi/8$. This effectively avoids redundant rotations of the robot, to a certain extent, reducing path cost. At the same time, Expansion Strategy 2 ensures that when the first-layer neighborhood nodes are obstacles, it does not directly generate paths from parent nodes to the second-layer neighborhood nodes. This avoids computing invalid path costs and guarantees the feasibility of generating paths.

3.2. Cost Function of the MLP D-star Algorithm

When conducting path planning, the D-star algorithm strictly considers obstacles as belonging to the grid cells of the grid map. It only avoids neighboring nodes that are obstacles during pathfinding. As a consequence, the generated paths are relatively close to obstacles, and in some cases, they may even pass through the narrow gaps between two adjacent obstacles. Therefore, when the D-star algorithm is deployed on a robot, this phenomenon may result in collisions between the robot and obstacles.

In this section, starting from the cost function of the D-star algorithm, we consider the threat cost of obstacles. By incorporating threat coefficients into the estimated costs of each node, the addition of threat coefficients can reduce the likelihood of selecting nodes around obstacles.

Taking the grid cell where the current robot position node is located as a reference, we establish threat regions for each known obstacle grid cell in the environment. Figure 3 shows an illustrative example of the threat region for obstacle o_1, where the black grid cell represents the obstacle o_1 in the node. We assign threat coefficients to the nodes corresponding to the first-layer neighborhood around the obstacle o_1 grid cell. Nodes farther away from the obstacle o_1 grid cell have lower probabilities of collision.

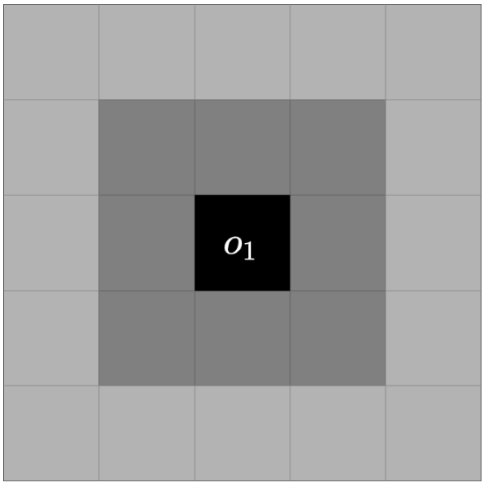


Figure 3. Threat Zone Illustration

To avoid potential collisions between the robot and obstacles while following the planned path, this section proposes an enhancement to the cost function of the D-star algorithm. An heuristic function is added, and a threat cost term is included in the $h(n)$ term. The modified cost estimation function of the improved D-star algorithm is as follows:

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

$$h^*(n) = h(n) + s(n) \tag{5}$$

In equations (4) and (5), $g^*(n)$ represents the cost value from the goal point to the current node n after adding the threat cost term to each grid node. $s(n)$ represents the threat cost term of node n, which is numerically equal to the sum of threat costs induced by surrounding obstacles on the current node n. $g(n)$ represents the estimated value from the current node to the start node, which is the heuristic term, and $n, f(n), h(n)$ is defined the same as in equation (1). The threat cost estimation function $s(n)$ is given by the following expression:

$$s(n) = \sum_{i=1}^{M} a_i s_i = \sum_{i=1}^{M} \frac{a_i}{\alpha \cdot \left((x_n - x_i)^2 + (y_n - y_i)^2 \right)} \tag{6}$$

Where, α is a scaling coefficient because the square root operation is removed for computational efficiency. Here, the scaling coefficient is used to appropriately reduce the algebraic value in the denominator. (x_i, y_i) represents the coordinates of the center of the i th obstacle. (x_n, y_n) represents the coordinates of the current path node n. a_i represents the effective coefficient of the threat cost. If there are no obstacles in the multi-level predicted neighborhood of the current path node, a_i is set to 0; otherwise, it is set to 1.Considering that the path node expansion in the MLP D-star algorithm is in a multi-level structure, the Manhattan distance and diagonal distance are greatly influenced by a single dimension. If there is a large difference in one dimension, the heuristic function value will become large, neglecting the influence of other dimensions. Therefore, the Euclidean distance is used as the heuristic function, and a weighted coefficient ω greater than 1

is introduced to increase the proportion of the cost from the current node to the goal point in the total cost. This helps increase the search depth and prevent the algorithm from getting stuck in local optima. The expression for the heuristic function is as follows:

$$h(n)=\omega\sqrt{(x_n-x_{start})^2+(y_n-y_{start})^2} \tag{7}$$

Where, ω represents the weighting coefficient. (x_{start},y_{start}) represents the coordinates of the starting point in the grid map. (x_n,y_n) represents the coordinates of the current node being expanded, denoted as node n, in the grid map.

3.3. Implementation of the MLP D-star Algorithm

We assume that the starting node for path planning is s and the goal node is g. We define the OPEN list and CLOSE list to store the nodes to be expanded and the nodes that have been expanded, respectively. We also define the WARN_LIST to store the non-expandable nodes in the multi-level prediction structure. The algorithm execution process differs from the original D-star algorithm in Step 2. The complete process is as follows:

(1) Create two lists, OPEN list and CLOSE list, and add the goal node to the OPEN list.

(2) Check if the OPEN list is empty. If there are no nodes in the OPEN list, the algorithm terminates, and path planning fails. Otherwise, check if the node with the minimum cost estimate in the OPEN list is the starting node. If true, path planning is successful, and proceed to Step 3. If false, add this node to the CLOSE list and check if there are obstacles in the first-level neighborhood of this node. If there are obstacle nodes, add the non-expandable nodes from the second-level neighborhood to the WARN_LIST. Add the nearby expandable nodes that are not in the OPEN and WARN_LIST lists to the OPEN list, and continue with this step.

(3) Traverse back to the goal node according to the backtracking pointers. If changes in the environment affect the nodes along the path, perform local replanning by adding the previous node of the changed node to the OPEN list, and return to Step 2. Otherwise, proceed to Step 4.

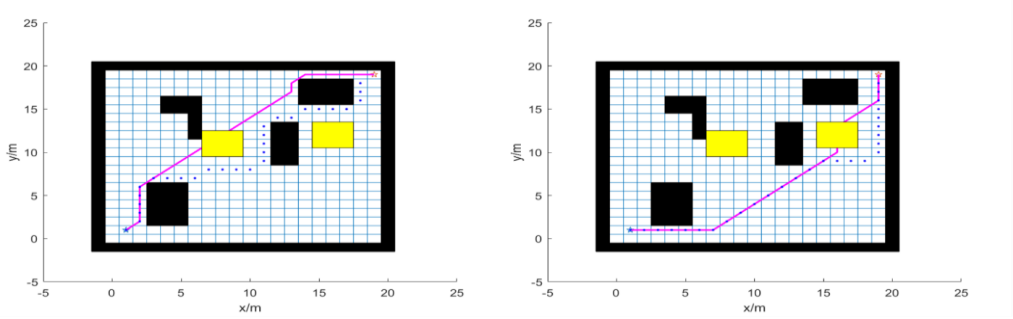
(4) Path planning is complete.

4. Algorithm Simulation and Analysis

To validate the feasibility and effectiveness of the proposed MLP D-star algorithm, this chapter conducts simulation experiments. Two environments are created: a conventional obstacle environment and an extreme obstacle environment, both with a size of 20x20. The starting point coordinates are set as $(1,1)$, and the goal point coordinates are set as $(19,19)$. The original D-star algorithm and the MLP D-star algorithm are compared as controls for studying dynamic path planning in local unknown environments. In this simulation, black grid cells represent static obstacles, yellow grid cells represent sudden obstacles. The magenta solid line represents the path planning result in the static environment, while the path represented by blue dots indicates the result after replanning due to the inclusion of sudden obstacles.

4.1. Path Planning in Conventional Obstacle Environments

The path planning result of the original D-star algorithm in the conventional obstacle environment is shown in Figure 4(a), while the path planning result of the MLP D-star algorithm is shown in Figure 4(b).



(a) D_star (b) MLP D_star

Figure 4. Path Planning in Conventional Environment

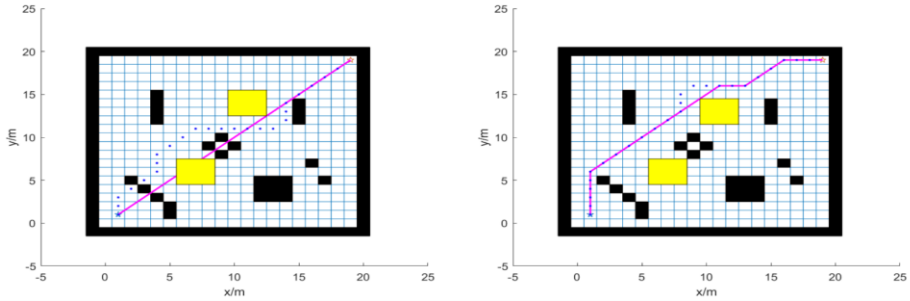
By comparing Figure 4(a) and Figure 4(b), it can be observed that in a conventional environment, where obstacles are scattered and mostly large-sized, the path planning result of the original D-star algorithm contains many waypoints and closely follows the edges and corners of the obstacles, which may lead to collision incidents in practical applications. The improved MLP D-star algorithm consistently maintains a safe distance from the obstacle edges and corners, reducing the probability of collision. Moreover, compared to the result of the original D-star algorithm, it has fewer waypoints and reduces unnecessary turns. Table 1 presents a comparison of path replanning results between the original D-star algorithm and the MLP D-star algorithm in a conventional obstacle environment.

Table 1. Comparison of D-star Path Planning Results

Comparison Parameters	Figure 4 (a)	Figure 4(b)
Number of Turns	14	4
Path Length	31.3137	30.7279
Time	0.3748	0.2792

4.2. Path Planning in Extreme Obstacle Environments

The path planning results of the original D-star algorithm in the extreme obstacle environment are shown in Figure 5(a), and the path planning results of the MLP D-star algorithm are shown in Figure 5(b).



(a) D_star (b) MLP D_star

Figure 5. Path planning in extreme environments.

By comparing Figure 5(a) and Figure 5(b), we can observe the following: In the extreme environment, the obstacles are small. The path generated by the original D-star algorithm hugs the edges of the obstacles and even exhibits the phenomenon of crossing obstacles as shown in Figure 5(a), which is not feasible in practical applications. The path generated by the MLP D-star algorithm, on the other hand, is significantly different from that of the original D-star algorithm. It consistently maintains a safe distance from the obstacles and avoids the phenomenon of crossing obstacles seen in the original D-star algorithm, resulting in a more practical and feasible path planning outcome. Table 2 presents a comparison of the path replanning results between the original D-star algorithm and the MLP D-star algorithm in extreme obstacle environments.

Table 2. Comparison of MLP D-star Path Planning Results

Comparison Parameters	Figure 5 (a)	Figure 5(b)
Number of Turns	7	6
Path Length	28.9706	29.5563
Time	0.4501	0.3025

By comparing Table 1 and Table 2, we can see that the path lengths generated by the MLP D-star algorithm are similar to those of the original D-star algorithm. However, the MLP D-star algorithm shows a reduction in the number of turning points in the path planning results for both obstacle environments, and there is also a noticeable decrease in runtime. Table 3 provides a comparative analysis of the performance between the MLP D-star algorithm and the original D-star algorithm.

Table 3. Comparison of the Two Algorithms

Comparison Parameters	D_star	MLP D_star
Number of Turns	No	Yes
Real-time Replanning	No	Yes
Considering obstacle threat cost	No	Yes
the practicality of the planned path	No	Yes

The simulation results demonstrate that the paths obtained by the MLP D-star algorithm are safer compared to the original D-star algorithm. Additionally, the algorithm exhibits higher computational efficiency. In practical applications, the MLP D-star algorithm proves to be more feasible and effective than the original D-star algorithm.

5. Conclusion

In this paper, based on the neighborhood concept and the original D-star algorithm, we propose a Threat-Cost-Based Multi-Level Predictive D-star algorithm (MLP D-star) for path planning of mobile robots in locally unknown environments. This approach improves the node expansion in the D-star algorithm by employing a multi-level predictive structure, thereby reducing excessive turns in the planned path. Additionally, we incorporate threat-cost and heuristic functions into the cost function of the D-star algorithm to prevent paths from crossing obstacles. Furthermore, we conduct comparative simulation experiments in both conventional and extreme environments to evaluate the

proposed algorithm. The results demonstrate that the MLP D-star algorithm can effectively handle path planning in locally unknown environments. Compared to the original D-star algorithm, the improved MLP D-star algorithm exhibits advantages in terms of real-time performance, practicality of path results, safety, and computational efficiency.

Funding

This paper is supported by Projects of major scientific and technological research of Ningbo City(2020Z065, 2021Z059,2022Z056,2023Z050(the second batch)), Major instrument special projects of the ministry of science and technology of China(2018YFF01013200), Projects of major scientific and technological research of Beilun District, Ningbo City(2021BLG002, 2022G009), Projects of engineering research center of Ningbo City (Yinzhou District Development and Reform Bureau [2022] 23), Projects of scientific and technological research of colleges student's of China(202213001008).

Data Availability

Data sharing is not applicable to this article as no new data were created or analysed in this study.

Conflict of Interest

The author states that this article has no conflict of interest.

References

[1] Zhang H-Y, Lin W-M, Chen A-X. Path planning for the mobile robot: A review . Symmetry, 2018, 10(10): 450.

[2] Gasparetto A, Boscariol P, Lanzutti A, et al. Path planning and trajectory planning algorithms: A general overview . Motion and Operation Planning of Robotic Systems: Background and Practical Approaches, 2015: 3-27.

[3] Dijkstra E W. A note on two problems in connexion with graphs. Edsger Wybe Dijkstra: His Life, Work, and Legacy. 2022: 287-290.

[4] Hart P E, Nilsson N J, Raphael B. A formal basis for the heuristic determination of minimum cost paths . IEEE transactions on Systems Science and Cybernetics, 1968, 4(2): 100-107.

[5] Stentz A. The focussed D algorithm for real-time replanning . Proceedings of the IJCAI. 1995: 1652-1659.*

*[6] Liu Jun, Feng Shuo, Ren Jianhua. Directed D * algorithm for dynamic path planning of mobile robots. Journal of Zhejiang University (Engineering Edition), 2020, 54 (2) : 291-300.*

[7] Rekleitis I, Bedwani J-L, Dupuis E, et al. Path planning for planetary exploration . Proceedings of the 2008 Canadian Conference on Computer and Robot Vision. IEEE, 2008: 61-68.

[8] Koenig S, Likhachev M. D^* lite . Aaai/iaai, 2002, 15: 476-483.

[9] Lavalle S M. Rapidly-exploring random trees: A new tool for path planning . 1998.

[10] Kuffner J J, Lavalle S M. RRT-connect: An efficient approach to single-query path planning . Proceedings of the Proceedings 2000 ICRA Millennium Conference IEEE International Conference on Robotics and Automation Symposia Proceedings (Cat No 00CH37065). IEEE, 2000: 995-1001.

[11] Karaman S, Walter M R, Perez A, et al. Anytime motion planning using the RRT . Proceedings of the 2011 IEEE international conference on robotics and automation. IEEE, 2011: 1478-1483.

[12] Khatib O. Real-time obstacle avoidance for manipulators and mobile robots . The international journal of robotics research, 1986, 5(1): 90-98.

[13] Fox D, Burgard W, Thrun S. The dynamic window approach to collision avoidance . IEEE Robotics & Automation Magazine, 1997, 4(1): 23-33.

[14] Lee D H, Lee S S, Ahn C K, et al. Finite distribution estimation-based dynamic window approach to reliable obstacle avoidance of mobile robot . IEEE Transactions on Industrial Electronics, 2020, 68(10): 9998-10006.

[15] Xiaoran Z, Hehua J. Path planning of mobile robot in local unknown environment . Proceedings of the The 2nd International Conference on Information Science and Engineering. IEEE, 2010: 5231-5234.