INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 10, Issue 5 - V10I5-1315)
Available online at: https://www.ijariit.com

Obstacle Avoidance in Mobile Robots: Algorithms, Optimization Techniques, and Analysis

Thinalisha M
lishaml2005@gmail.com
PSG College of Technology, Coimbatore

Yogesh Vk
yogeshvk2005@gmail.com
PSG College of Technology, Coimbatore

Subanesh V
subanesh543@gmail.com
PSG College of Technology, Coimbatore

ABSTRACT

A path-planning technique for robots with obstacles is proposed. The formulation of the issue is in Cartesian space. A map of possible robot configurations is obtained by the application of nonlinear programming techniques. Next, a weighted graph is linked to the map, and a search algorithm is employed to identify a series of robot configurations that avoid collisions between two pre-selected points. Spatial robotic manipulators and redundant planar robotic manipulators with prismatic and revolute joints have both benefited from the application of this approach.

KEYWORDS: *Collision Avoidance, Cartesian Path Planning, Robotics, Real-Time Performance, Optimization Techniques, Sampling-Based Planners, Machine Learning Models.*

INTRODUCTION

Robots or other devices that can function autonomously and make judgments based on their surroundings are known as autonomous robots. These robots have embedded systems and sensors to collect data about their environment, including obstacle recognition, mapping, and navigation. For autonomous robots to function and be able to safely and effectively navigate their surroundings, obstacle avoidance is essential. Robocops can safely reach their destination and complete their tasks by avoiding obstacles and limiting collisions with them thanks to obstacle avoidance algorithms. Thus, for autonomous robots to operate effectively and dependably, obstacle avoidance is a crucial component.

This study examines a survey of the literature on mobile robot navigation techniques and alternative route planning. The next sections go over the primary algorithms it takes into account.

In global path planning, a robot navigates by using previously collected environmental data that is fed into its planning system to calculate a path from its starting point to its goal. This approach effectively optimizes the route progressively by creating the entire path before the robot sets out on its journey [1]. The intentional process of determining how to move a robot from one location to another is called global path planning. In global route planning, the robot is released into the assigned environment after making its way from the starting point to the destination [2].

On the other hand, local path planning entails steering a robot through an unfamiliar or dynamic terrain while the algorithm adapts to barriers and changes quickly. The main goal of this approach is to provide safe navigation by employing sensor-based data for real-time obstacle avoidance [3]. Until it comes into an impediment, the robot usually travels the shortest, straightest route between the start and the destination. When it detects an impediment, it veers off course while updating crucial information like the updated target distance and the obstacle bypass location [2].

COLLISION IN PATH PLANNING

One of the main problems in robotics and autonomous navigation is avoiding collisions. Enabling cars and robots to navigate through an environment without running into obstacles is the aim. The foundation of this challenge is the idea of C-space, which is all potential robot orientations and positions. Movement-restricting obstacles are present in C-space, providing planners with an overview of possible routes and regions to avoid. Moving autos and pedestrians are examples of barriers that are in motion, whereas fixed obstacles like walls and furniture are not. Technologies for detecting collisions are essential for safe navigation.

These systems rapidly detect whether a pathway meets with any obstacles by using bounding volumes and spatial partitioning. In addition to recognizing obstacles and designing courses, safety margins are needed to allow for uncertainties such as sensor faults and tiny deviations produced by the robot's movement. Real-time path adjustments are necessary in dynamic environments to adapt to sudden changes, for example, using rapid recalibration approaches.

MATHEMATICAL FRAMEWORK FOR PATH PLANNING

This section describes a path planning scenario using a straight-line approach in a 2D environment containing an obstacle. The focus is on defining the start and goal positions, the representation of the obstacle, and the computation of the path.

A starting point S to a goal point G in a 2D space while considering an obstacle O.

$$S = (x_s, y_s) = (0,0) \quad (1)$$
$$G = (x_g, y_g) = (10,10) \quad (2)$$

The obstacle is defined as a rectangle with specified corner coordinates.

$$O = \{(x_1, y_1), (x_2, y_1), (x_2, y_2), (x_1, y_2)\} \quad (3)$$

Where

$$x_1 = 4, \quad x_2 = 6, \quad y_1 = 4, \quad y_2 = 6$$

The path from the starting point S to the goal point G can be computed using linear interpolation. The equations for the parametric representation of the path are given by:

$$x(t) = (1 - t).x_s + t.x_g \quad (4)$$
$$y(t) = (1 - t).y_s + t.y_g \quad (5)$$

Where *t* varies from 0 to 1. The resulting coordinates represent points along the line segment between S and G.

$$N = 100$$

The path points can be expressed as:

$$P(t) = \{(x(t), y(t)) \mid t \in [0,1] \quad (6)$$

Specifically, for N points:

$$t_i = \frac{i}{N-1}, i = 0,1,\ldots,N-1 \quad (7)$$

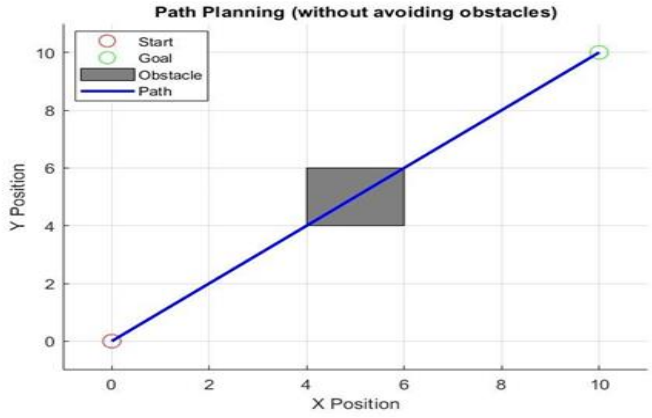

Figure 1- Path Planning (Without Avoiding Obstacles)

PATH PLANNING TO AVOID OBSTACLES

A. Robots or autonomous vehicles must plan a safe and effective path from a starting point to a goal location while dodging obstacles. This process is known as path planning to avoid obstacles. In order to avoid collisions, this process involves assessing the spatial arrangement of the surrounding area and making sure the selected trajectory does not cross any barriers.

Several algorithms have been developed to tackle this challenge, including Dijkstra's Algorithm, which finds the shortest path in a weighted graph; the Bellman-Ford Algorithm, which can handle graphs with negative weights; Bug Algorithms, which guide robots around obstacles by navigating their perimeters; the A* Algorithm, which combines pathfinding with heuristics for efficiency; and Fuzzy Logic Algorithms, which use fuzzy sets and rules to make decisions under uncertainty. These methods all have special benefits for avoiding obstacles in various situations.

A. DIJKSTRA Algorithm

Dijkstra's Algorithm is a fundamental method used to determine the shortest path from a starting node to all other nodes in a weighted graph. It is especially effective for graphs with non-negative edge weights and is commonly applied in various fields such as networking, robotics, and transportation[5].
The algorithm begins by initializing the distance d from the starting node s to all other nodes as follows:

$$d(v) = d(v) = \{0 \quad if\ v = s\ \infty \quad if\ v \neq s \quad (8)$$

Here, d(v) represents the tentative distance to node v. The algorithm uses a priority queue to select the unvisited node with the smallest tentative distance.
The core process of Dijkstra's Algorithm involves updating the distances for each neighbour u of the current node v:

$$d(u) = min\ (d(u), d(v) + w(v,u)) \qquad (9)$$

In this equation, w(v, u) denotes the weight of the edge connecting nodes v and u. The algorithm relaxes the edge by checking if the new computed distance d(v)+w(v, u) is shorter than the current known distance d(u).

The algorithm iterates through this process until all nodes have been visited or the shortest path to the desired target node has been found [5].

B. BELLMAN-FORD Algorithm

The Bellman-Ford Algorithm is an essential algorithm for finding the shortest paths from a single source node to all other nodes in a weighted graph, particularly when the graph may contain negative edge weights.

The algorithm initializes the distance d(v) from the source node S to every other node V and checks if the distance to node V can be shortened by going through node u as already done in the previous method (Relaxation process)

The relaxation process is performed for V−1 iterations, where V is the total number of vertices in the graph. This is necessary because the longest path in any graph can have at most V−1 edges[6]

After completing the relaxation process, the algorithm performs an additional iteration to check for negative weight cycles. If a shorter path is found during this final check, it indicates the presence of a negative weight cycle, which can be mathematically represented as:

$$d(v) > d(u) + w(u,v) \qquad (10)$$

If this condition holds for any edge after V−1 iterations, it signifies that the graph contains a negative weight cycle. The overall time complexity of the Bellman-Ford Algorithm is given by

$$O(V.E) \qquad (11)$$

where E is the number of edges in the graph. This complexity reflects the algorithm's efficiency in finding the shortest paths, even in the presence of negative weights, making it a valuable tool in various applications[6]

C. BUG Algorithm

Despite the presence of more efficient algorithms, Bug algorithms are still significant in robotics. These were the earliest navigation and obstacle avoidance algorithms that achieved relatively reliable results with speedy computation times. The algorithms are designed to work assuming that the robot is a single point in 2D space and that its movement is between each point[7].
The distance d from the robot's current position (xr, yr) to the goal position (xg,yg) can be calculated using the Euclidean distance formula:

$$d = \sqrt{(x_g - x_r)^2 + (y_g - y_r)^2} \qquad (12)$$

To compute the direction (angle) θ in which the robot should move towards the goal, the following equation can be used:

$$\theta = \left(\frac{y_g - y_r}{x_g - x_r}\right) \qquad (13)$$

When the robot encounters an obstacle, it switches to a wall-following mode. The angle ϕ for the wall following can be determined based on the geometry of the obstacle. If do is the distance from the robot to the nearest obstacle and dw is the desired distance from the obstacle, the adjustment can be given as:

$$\emptyset = \left(\frac{d_w}{d_o}\right) \qquad (14)$$

The new position of the robot after a small movement can be updated based on the current direction. If the robot moves a distance Δs in the direction θ

$$x_r^{'} = x_r + \Delta s.cos\ cos\ (\theta) \qquad (15)$$

$$y_r^{'} = y_r + \Delta s.sin\ sin\ (\theta) \qquad (16)$$

These equations provide the new coordinates (xr′, yr′) after the robot has moved a step toward the goal or along the wall.
To decide whether to continue moving towards the goal or to switch to wall-following, the robot checks the distance to the nearest obstacle dobs. It d_{obs} is less than a certain $d_{threshold}$, the robot switches to wall-following:

$$\text{If } d_{obs} < d_{threshold}, \text{ switch to wall-following mode}$$

Conversely, if dobs exceed the $d_{thershold}$, the robot resumes direct navigation towards the goal.

D. A Algorithm*

The A* algorithm is a graph search algorithm similar to Dijkstra's algorithm. to speed up the search process of Dijkstra's algorithm. To do this, they introduced a heuristic cost function, which is the distance between the current point and the target point. Like the Dijkstra algorithm, the A* algorithm needs an environment model, e.g., a grid map. In the A* algorithm, the search area is usually divided into small squares, where each square represents a node. The algorithm can solve various routing problems with superior performance and accuracy compared with Dijkstra's algorithm. Algorithm A* solves problems by finding the path with the lowest cost (e.g., the shortest time) among all possible paths to the solution. Of these paths, it first considers those that appear to lead the fastest to the solution. The A* algorithm uses an evaluation function.

$$f(n) = g(n) + h(n) \qquad (17)$$

The function $f(n)$ represents the cumulative cost from the starting point to the current point, extending to the target point. Meanwhile, $g(n)$ denotes the shortest cost from the initial point to the current position *n*, and $h(n)$ predicts the optimal path cost from the current point n to the destination, often calculated as the Manhattan distance [8]. Initially applied in port areas, Casalino used the A* algorithm [9] for local pathfinding. Guan proposed an improved version of the A* algorithm [10], which helps Unmanned Surface Vessels (USVs) avoid static obstacles at sea and reach their destination smoothly while avoiding local minima. In addition, a collision-free trajectory planning method for space robots based on the A* algorithm has been developed in [11]. The geometric A* presented in [12] is designed for route planning of automated guided vehicles (AGVs) operating in port environments. Despite its advantages, traditional A* does not always provide an optimal solution, as it does not take into account all feasible routes. In each iteration, A* evaluates the nodes based on their *f* values, which is a computationally expensive process, especially in large map search areas. Consequently, this approach can significantly slow down the speed of route planning.

ANALYSIS OF RESULT

The simulation results of the control techniques and their relative performance are presented below:

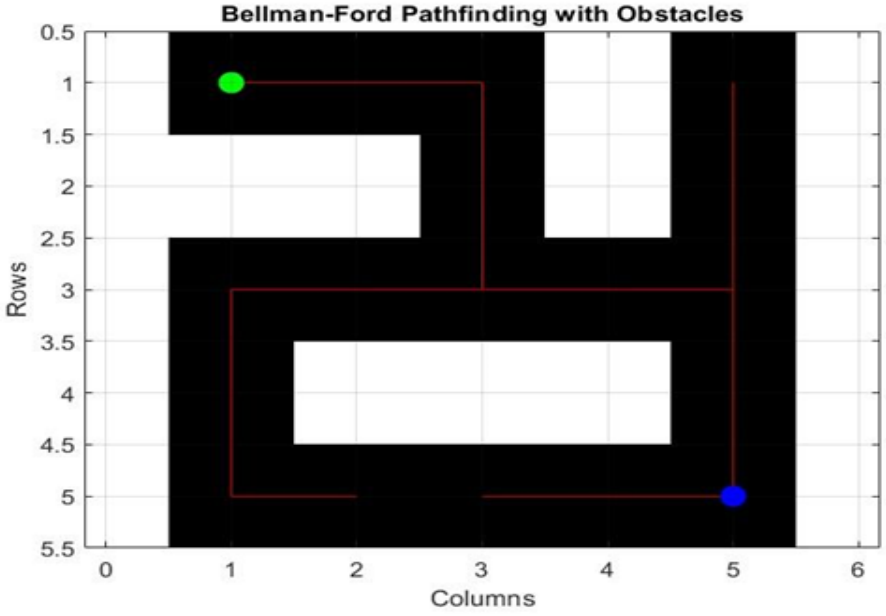

Figure 2- Dijkstra Path planning with obstacle

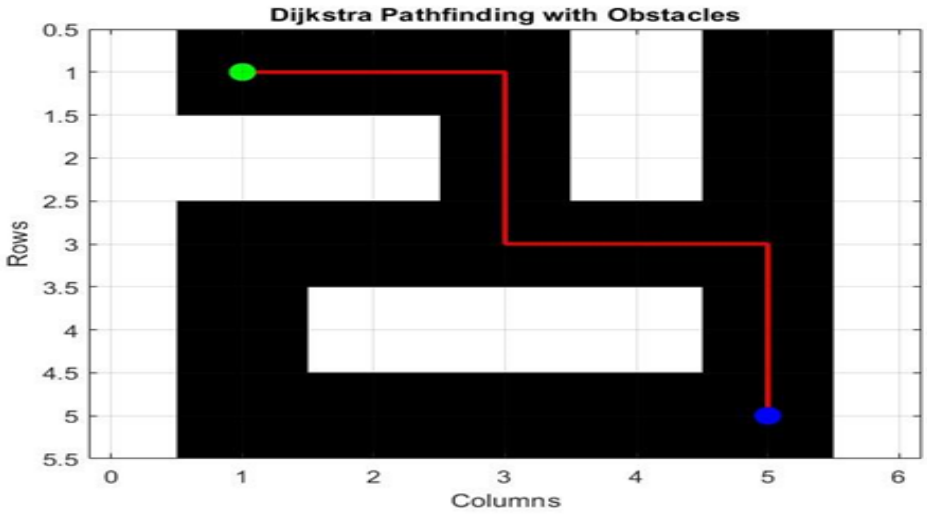

Figure 3-Bellman-Ford path planning

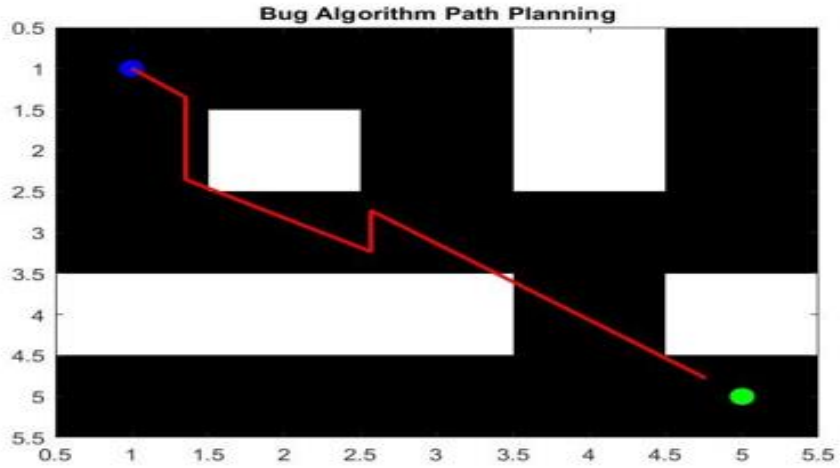

Figure 4- Bug Algorithm Path Planning

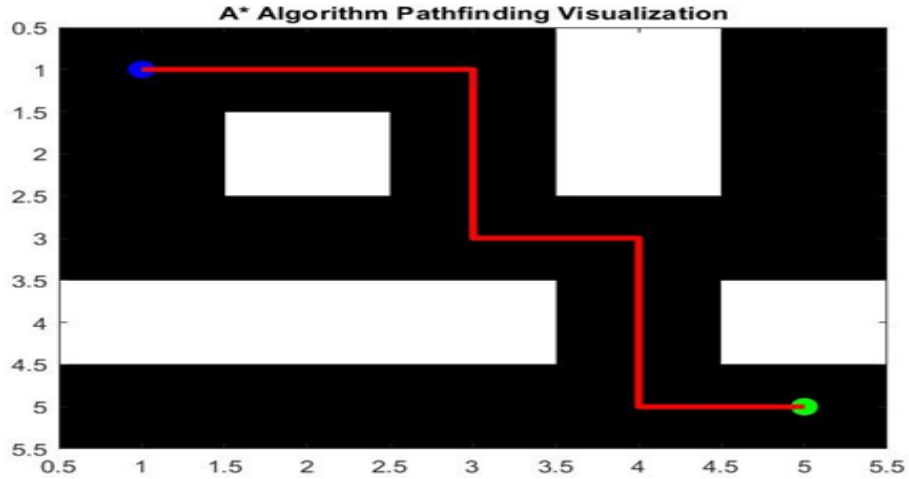

Figure 5- A Algorithm Path Planning*

By comparing the performance of the three methods Bellman-Ford, the Bug algorithm, and Dijkstra's algorithm it becomes evident that the A* algorithm is the most efficient and practical choice for pathfinding in a structured, known environment.

While Bellman-Ford guarantees the shortest path, it is computationally expensive. Though simple and adaptable to unknown environments, the Bug algorithm often results in suboptimal and longer paths. Dijkstra's algorithm is efficient but lacks the heuristic-driven optimization that A* offers. A* combines Dijkstra's reliability with a heuristic approach, making it faster and more efficient in finding the shortest path, particularly in complex environments with obstacles.

CONCLUSION

Overall, A* stands out as the most ideal approach due to its superior efficiency and efficacy among the four strategies presented. While Dijkstra's technique can be relied upon to compute shortest paths for static graphs, it may not perform as quickly in complicated situations when searching exhaustively. Bellman-Ford is capable of determining the best pathways in graphs with negative weights, but its greater computational complexity makes it difficult to apply to graphs of greater size. While the aforementioned code offers a significant level of simplicity in dynamic contexts, its reactive design frequently leads to lengthier, less-than-ideal pathways.

When Dijkstra employs efficiency to identify a heuristic shortcut to get around obstacles, A* quickly determines the shortest path—though not always the best one—around them. So, A* is particularly well suited for robotics, game, or navigation applications where speed and optima come to bear. A* is considerably better than the other option for pathfinding applications in real-world scenarios. All things considered, the way A* functions demonstrates the true superiority of this method when applied to difficult issues in real-world situations where precision and speed must be carefully balanced and nothing is flawless. As a result, A* may effectively direct its search toward the target using heuristics while avoiding pointless and unnecessary computations.

REFERENCES

[1] Fox, D., Burgard, W., & Thrun, S. (1997). The Dynamic Window Approach to Collision Avoidance. IEEE Robotics and Automation Magazine.
[2] Latombe, J.-C. (1991). Robot Motion Planning. Kluwer Academic Publishers.
[3] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. In The International Journal of Robotics Research, 5(1), 90-98.
[4] Quinlan, S., & Khatib, O. (1993). Elastic bands: Connecting path planning and control. In Proceedings of IEEE International Conference on Robotics and Automation (ICRA), 802-807.
[5] Dijkstra, E. W. (1959). A note on two problems in connection with graphs. In Numerische Mathematik, 1, 269-271.
[6] Bellman, R. (1958). On a routing problem. In Quarterly of Applied Mathematics, 16(1), 87-90.
[7] Lumelsky, V., & Stepanov, A. (1987). Path planning strategies for a point mobile automaton moving amidst unknown obstacles of arbitrary shape. In Algorithmica, 2(1), 403-430.
[8] Yao, J.; Lin, C.; Xie, X.; Wang, A.J.; Hung, C.C. Path Planning for Virtual Human Motion Using Improved A* Star Algorithm. In Proceedings of the 2010 Seventh International Conference on Information Technology: New Generations, Las Vegas, NV, USA, 12–14 April 2010; pp. 1154–1158.
[9] Casalino, G.; Turetta, A.; Simetti, E. A three-layered architecture for real-time path planning and obstacle avoidance for surveillance USVs operating in harbor fields. In Proceedings of the OCEANS 2009-EUROPE, Bremen, Germany, 11–14 May 2009; pp. 1–8.
[10] Guan, W.; Wang, K. Autonomous collision avoidance of unmanned surface vehicles based on improved A-star and dynamic window approach algorithms. IEEE Intell. Transp. Syst. Mag. 2023, 113,
[11] Gao, X.; Jia, Q.; Sun, H.; Chen, G. Research on path planning for 7-DOF space manipulator to avoid obstacle based on A* algorithm. Sens. Lett. 2011, 9, 1515–1519.
[12] 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.
[13] A., & Saldaña, J. (2018). "A Hybrid Approach to Robot Motion Planning with Collision Avoidance." Journal of Automation, Mobile Robotics & Intelligent Systems, 12(1), 18-27.
[14] Examines hybrid strategies combining traditional and modern techniques for effective collision avoidance.
[15] Gonzalez, J., & Rojas, R. (2021). "Real-Time Collision Avoidance for Mobile Robots Using Neural Networks." Robotics and Autonomous Systems, 137, 1-14.
[16] Discusses the application of neural networks for real-time collision avoidance in mobile robotic systems.
[17] Kuffner, J. J., & LaValle, S. M. (2000). "RRT-Connect: An Efficient Approach to Single-Query Path Planning." Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 3232-3237.
[18] Introduces the RRT-Connect algorithm, emphasizing efficient path planning and collision avoidance.
[19] Kumar, V., & Mavridis, P. (2018). "Multi-Objective Path Planning with Collision Avoidance Using Genetic Algorithms." Robotics and Autonomous Systems, 104, 90-100.
[20] Discusses a genetic algorithm approach for multi-objective path planning that incorporates collision avoidance.
[21] Li, M., & Wang, Y. (2019). "A Survey of Collision Avoidance Algorithms for Autonomous Mobile Robots." Journal of Control Science and Engineering, 2019, 1-17.
[22] A comprehensive review of various collision avoidance algorithms applicable to mobile robots.
[23] Santos, P., & Lima, P. (2020). "Towards Safe and Efficient Navigation: A Novel Collision Avoidance Approach for Autonomous Vehicles." IEEE Transactions on Intelligent Transportation Systems, 21(2), 870-879.
[24] Explores new strategies for collision avoidance in the context of autonomous vehicles, with implications for robotics.
[25] Huang, H., & Zhang, Y. (2021). "Deep Reinforcement Learning for Dynamic Collision Avoidance in Robotic Manipulators." International Journal of Robotics Research, 40(7), 897-912.
[26] Investigates the use of deep reinforcement learning techniques for real-time collision
[27] avoidance in robotic manipulators.
[28] Sedighi, K.H.; Ashenayi, K.; Manikas, T.W.; Wainwright, R.L.; Tai, H.M. Autonomous local path planning for a mobile robot using a genetic algorithm. In Proceedings of the 2004 Congress on Evolutionary Computation (IEEE Cat. No. 04TH8753), Portland, OR, USA, 19–23 June 2004; Volume 2, pp. 1338–1345. [Google Scholar]
[29] Yan, K.; Ma, B. Mapless navigation based on 2D LIDAR in complex unknown environments. Sensors 2020, 20, 5802. [Google Scholar] [CrossRef] [PubMed]

[30] Vckay, E.; Aneja, M.; Deodhare, D. Solving a Path Planning Problem in a Partially Known Environment using a Swarm Algorithm. arXiv 2017, arXiv:1705.03176. [Google Scholar]
[31] Kamil, F.; Tang, S.; Khaksar, W.; Zulkifli, N.; Ahmad, S. A review on motion planning and obstacle avoidance approaches in dynamic environments. Adv. Robot. Autom. 2015, 4, 134–142. [Google Scholar]
[32] Dijkstra, E.W. A note on two problems in connection with graphs. In Edsger Wybe Dijkstra: His Life, Work, and Legacy; Association for Computing Machinery: New York, NY, USA, 2022; pp. 287–290. [Google Scholar]
[33] Sabo, C.; Cohen, K. Fuzzy logic unmanned air vehicle motion planning. Adv. Fuzzy Syst. 2012, 2012, 989051. [Google Scholar] [CrossRef]
[34] Gonzalez, R.; Kloetzer, M.; Mahulea, C. Comparative study of trajectories resulted from cell decomposition path planning approaches. In Proceedings of the 2017 21st International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 19–21 October 2017; pp. 49–54. [Google Scholar]
[35] Liu, L.s.; Lin, J.f.; Yao, J.x.; He, D.w.; Zheng, J.s.; Huang, J.; Shi, P. Path planning for a smart car based on Dijkstra algorithm and dynamic window approach. Wirel. Commun. Mob. Comput. 2021, 2021, 8881684. [Google Scholar] [CrossRef]
[36] Kirono, S.; Arifianto, M.I.; Putra, R.E.; Musoleh, A.; Setiadi, R. Graph-based modeling, and Dijkstra algorithm for searching vehicle routes on highways. Int. J. Mech. Eng. Technol. (IJMET) 2018, 9, 1273–1280. [Google Scholar]
[37] Wang, C.; Cheng, C.; Yang, D.; Pan, G.; Zhang, F. Path planning in localization uncertain environment based on Dijkstra method. Front. Neurorobot. 2022, 16, 821991. [Google Scholar] [CrossRef] [PubMed]