Evaluating Dijkstra and A* Pathfinding Algorithms for Mobile Robots in Warehouse Environments Using CoppeliaSim

Submitted: 26th August 2025; accepted: 27th August 2025

Prabin Kumar Jha, Shambo Roy Chowdhury

DOI: 10.14313/jamris-2026-012

Abstract:
In modern warehouse automation, mobile robots are essential for enhancing operational efficiency by autonomously navigating to pick and transport items. Effective path planning is crucial for these robots to move through complex environments, avoid obstacles, and minimize travel time. This study evaluates two prominent path-planning algorithms, Dijkstra and A, implemented on a mobile robot within a simulated 3D warehouse environment using CoppeliaSim. Three distinct rack locations were analyzed to assess the performance of both algorithms concerning path optimality, computational efficiency, and real-time applicability. Simulation results indicate that while both algorithms successfully generated safe and accurate paths, A* outperformed Dijkstra in terms of speed and path efficiency. A*' 's heuristic-driven approach resulted in lower computational load and faster execution time, making it more suitable for real-time warehouse operations, where responsiveness is critical. The insights gained provide valuable guidance for robotics engineers and developers in selecting appropriate path-planning strategies for autonomous navigation in industrial settings.*

Keywords: *Dijkstra Algorithm; A* Algorithm; Warehouse; Mobile Robot; CoppeliaSim.*

1. Introduction

Path planning of mobile robot involves determining a route from a starting position to a destination while avoiding obstacles. This process is guided by optimization criteria such as minimizing distance, time, or energy consumption.

To begin path planning, the robot's operating environment must be modeled, a process known as map construction. There are two primary methods for this: the road marking method and the grid method. The road marking method creates a feasible path by connecting specific markers, using techniques like the visibility graph and the tangent method. However, this approach is complex, has low accuracy, and is not well suited for large-scale environments with many obstacles. The grid method divides the space into small, uniform cells, making map construction more straightforward.

Despite its advantages, this method struggles with real-time and high-precision path planning when dealing with large grid sizes and densely packed obstacles [1].

Using grid-based environment modeling, various techniques have been developed for mobile robot path planning. Traditional graph search algorithms, such as Dijkstra's algorithm and the A* algorithm, are widely used.

These algorithms provide accurate results and can find the optimal path. However, as the grid size increases, their computational efficiency decreases, making them less practical for large-scale environments. The A* algorithm is a more modern version of Dijkstra's algorithm, designed to improve efficiency by introducing a heuristic function to prioritize nodes that are closer to the goal. This heuristic makes A* more efficient than Dijkstra's, especially in large or complex search spaces.

The A* algorithm, integrated with a greedy approach, is utilized for multiobjective point planning to navigate five target nodes within a warehouse setting, allowing for a comparison of path length, turning angles, and additional metrics. Simulation results indicate that the proposed strategy produces a smoother path [2]. An improved version of Dijkstra's algorithm, incorporating a multilayer dictionary approach, was utilized to enable multiple robots to autonomously and simultaneously navigate across an indoor environment [3].

A pheromone-based enhancement to the traditional Dijkstra algorithm has been introduced to reduce redundant inflection points and improve path efficiency in mobile robot navigation [4]. The proximal policy-Dijkstra (PP-D) method integrates proximal policy optimization (PPO) with Dijkstra's algorithm to enable efficient strategy learning and real-time decision-making in complex warehouse environments [5]. An integration of Dijkstra's algorithm with genetic algorithms to develop a path-planning strategy for robots operating in fire scenarios, focusing on minimizing traversal time and thermal exposure [6]. An improved path-planning approach was developed by combining a segmented path-planning method with Dijkstra's algorithm to enhance efficiency in dynamic environments [7].

Design and implementation of an optimized, collision-free pathfinding algorithm based on an enhanced Dijkstra's algorithm, tailored for practical applications in obstacle-rich environments is reported in [8]. An enhanced Dijkstra algorithm was integrated with particle swarm optimization for global path planning in mobile robots, aiming to reduce computational time and improve efficiency in [9]. A novel method for autonomous mobile robot path planning that refines routes generated by both conventional and sampling-based algorithms through Bezier curve optimization, designed for environments where all obstacles are fully known and accounted for has been shown in [10].

An intelligent path-planning framework for assistive-care robots, integrating Dijkstra's algorithm with probabilistic model checking to dynamically navigate environments based on predicted human movement is shown in [11].

A hybrid of an improved A* algorithm for optimal global path planning and an enhanced DWA algorithm for local path planning was proposed for the forklift automated guided vehicle (FAGV). The improved A* ensures a global path better suited to FAGV navigation, while the modified DWA evaluation function guides the local path to align more closely with the global trajectory [12].

Local path planning introduced an environment-aware strategy for dynamically adjusting parameters, integrating evaluation factors for deviation and dynamic obstacle avoidance. This approach helps overcome local optima and ensures timely responses to moving obstacles [13].

A comparative analysis of Dijkstra and A* for mobile robots navigating through urban environments with obstacles has been studied in extensively. Both algorithms are capable of producing optimal paths. A* outperforms Dijkstra in terms of computational efficiency due to its heuristic-based approach, especially in environments where the goal was far from the start. However, the study concluded that A* performance degraded in dynamic environments where the heuristic function was not updated quickly enough [14]. Another comparison was done between Dijkstra's and A* algorithms for multirobot systems in a crowded environment. It was noted that A* provides faster pathfinding for individual robots in nondynamic environments but became less efficient when used in real-time multirobot scenarios where robots have to avoid each other. Dijkstra's algorithm, on the other hand, was more predictable and consistent in multirobot pathfinding but required significantly higher computation time [15].

An analysis was done for both Dijkstra and A* for real-time robot navigation, particularly in the context of indoor navigation and obstacle avoidance. It was found that A* was more suited for real-time applications, as it could produce paths more quickly by using a suitable heuristic. However, in more complex indoor environments where obstacles are nonstatic, Dijkstra's algorithm, when combined with obstacle detection and dynamic planning, showed better reliability in ensuring optimal paths [16].

A hybrid algorithm first uses A* to find a quick path in a static environment and switches to Dijkstra when the environment becomes dynamic or when the path needs to be adjusted frequently. A test was performed on mobile robots in dynamic and partially observable environments; results showed that hybridization improved efficiency and robustness [17]. A combined Dijkstra's algorithm with A* for solving multi-objective pathfinding problems in mobile robots considered multiple factors such as safety, time, and energy consumption. A conclusion was drawn that the hybrid approach could more effectively address complex objectives while still ensuring optimal paths, combining the guaranteed optimality of Dijkstra with the efficiency of A* algorithm [18].

Dijkstra demonstrated marginally superior path quality at the cost of increased computation time while tested on TurtleBot 3 across both symmetrical and asymmetrical test setups [19]. A detailed case study has been discussed for A* algorithm advantage in handling dynamic scenarios while emphasizing Dijkstra's robustness in obstacle-dense static settings [20]. The experiments, done under nonholonomic constraints, reinforced A*'s strengths in speed and adaptability [21]. Random-sampling-based robot path planning is reported in [22] for autonomous fruit harvesting robots. Among Dijkstra, A*, ant colony optimization (ACO), and rapidly exploring random tree (RRT) algorithms evaluated in grid maze environments, A* proved to be the most efficient, delivering the fastest search times and the most accurate paths under the given test conditions [23]. A self-directed telepresence robot (TR) capable of following a human subject has been introduced. It leverages a single-lens (monocular) camera and the you-only-look-once (YOLO) deep-learning framework for identifying individuals and estimating their proximity [24].

In summary, path planning of mobile robots begins with environment modeling, commonly done using either the road-marking or grid-based methods. While road marking offers precise geometric paths, it lacks scalability and accuracy in complex environments. The grid method is simpler and more adaptable but can face performance issues in dense or large-scale scenarios.

Recent advancements include combining Dijkstra or A* with algorithms like RRT, DWA, and Bezier curve optimization to generate smoother, more efficient, and dynamically feasible paths. Comparative studies show A* generally outperforms Dijkstra in speed, especially with real-time and heuristic-guided planning, while Dijkstra is more consistent in static and multirobot settings. Hybrid strategies leveraging both algorithms have shown promising results in adapting to dynamic, uncertain, or densely obstructed environments. Other AI methods like considering constraint satisfaction problems have also been used for solving robot path planning [25]. Traditional algorithms like Dijkstra's and A* are widely used for grid-based path planning.

Dijkstra ensures optimal paths but is computationally intensive, while A* improves efficiency using heuristics. Both have been enhanced through hybrid models, integration with machine learning (e.g., PPO), swarm intelligence (e.g., PSO, ACO), and multiobjective optimization (e.g., genetic algorithms), to improve navigation in dynamic and obstacle-rich environments.

This study presents a comparative analysis of Dijkstra and A* algorithms implemented in autonomous mobile robots for warehouse navigation using the CoppeliaSim simulation in 3D environment. Both algorithms are evaluated based on path optimality and computational efficiency to the warehouse layout.

2. Modeling of Grid

The grid method, introduced by W. E. Howden in 1968, is based on representing a mobile robot's environment as a binary matrix. Each cell holds either a 0 or a 1 , where 0 indicates a free space, and 1 indicates an obstacle. This approach simplifies environmental modeling for robotic path planning [1].

Algorithm 1: Make Graph (Grid)

Objective:
To generate an undirected graph G = (V, E) from a 2D occupancy grid, where each free cell is treated as a vertex, and edges are added between adjacent free cells in the four-connected neighborhood.

Description:

1. Initialize an empty vertex set V and an empty edge set E.
2. Iterate over each cell (i, j) in the grid: If the cell is marked as free (i.e., Grid [i][j] = 0), do the following:
   a) Create an entry in Graph with key (i, j) and initialize its value to an empty list.
   b) For each of the four neighbors $(i + 1, j), (i - 1, j), (i, j + 1), (i, j - 1)$:

   If the neighbor is within bounds and Grid $[i'][j'] = 0$: Append (i', j') to Graph[(i, j)].
3. Return the constructed Graph.

In the CoppeliaSim, this grid-based representation offers a clearer visual understanding of the workspace. ' S ' denotes the starting location and ' G ' indicates the destination. The environment is discretized into a 2D grid. Each cell in the grid is considered as a node by the search algorithm.

To find a collision-free path from point S to point G, specific search rules must be followed. The algorithm used in this study initially applies a fourneighborhood search pattern, where the movement cost from the starting point S in any direction is set to 1. The algorithm for grid is described in Algorithm 1 and is implemented with a Python API. To enable efficient path planning, we use an adjacency list representation, where the graph is stored as a dictionary, mapping each node to a list of its directly reachable neighbors.

3. Implementation of Dijkstra Algorithm in a Mobile Robot

Dijkstra's algorithm is used to search the shortest path between two points by minimizing the total cost or effort required to travel from the starting point to the destination. It works by evaluating nodes in a graph and keeping track of the ones with the lowest travel cost. Starting from the source, the algorithm explores all possible routes, continually updating the shortest known distances. Routes with higher costs are disregarded in favor of more efficient paths, ensuring the final result is the most costeffective path to the goal. The Dijkstra's algorithm is explained in Algorithm 2.

Algorithm 2: Dijkstra's Algorithm for GridBased Path Planning

Objective:
Given a graph in the format Graph[node] = [list of neighbor nodes], compute the shortest path from the Start node to the Goal node using Dijkstra's algorithm.
Description:
Let:

- Graph be a dictionary representing the adjacency list,
- Start and Goal be nodes of the form (x, y)
- dist be a dictionary mapping each node to its current shortest distance estimate,
- prev be a dictionary for backtracking the optimal path.

Steps:

1. Initialize dist [node] = ∞ for all nodes in the graph; set dist[Start] = 0.
2. Initialize prev [node] = None for all nodes.
3. Create a priority queue Q and insert Start with priority 0.
4. While Q is not empty: Extract the node u with the lowest dist[u]. If u == Goal, terminate the loop.
   For each neighbor v in Graph[u]:
   Compute alternative path cost: alt = dist[u] + 1.
   If alt<dist[v], then:
   Update dist[v] = alt
   Update prev[v] = u
   Insert or update v in Q with priority alt
5. Reconstruct the path from Goal to Start by following prev[v] links.
6. Return the path and its total cost (dist[Goal]). If Goal is unreachable, return failure.

4. Implementation of A* Algorithm in a Mobile Robot

The A* algorithm, introduced in 1968, is an informed search technique that combines elements of uniform-cost search and greedy best-first search. It evaluates paths based on both the cost incurred and an estimated cost to reach the goal from that node. This estimation is typically derived from a heuristic function, often representing the remaining distance to the goal. By integrating this heuristic, A* efficiently directs its search toward the most promising paths, aiming to search the optimal solution by exploring a fewer number of nodes. The A* algorithm, with similar 2D occupancy Grid as input, is expressed in Algorithm 3.

An evaluation function $F(n) = G(n) + H(n)$, which combines two key elements: G(n), the known cost from the starting point to the current node, and H(n), an estimated cost from the current node to the goal. These components work together and need to be properly balanced to ensure optimal performance [2]. The H(n) is a heuristic function that helps steer the search in the right direction.

Algorithm 3: A* Algorithm

Step 1: Initialization

1. Create an open set (priority queue) to store nodes to be evaluated, sorted by lowest f(n).
2. Create a closed set (visited nodes).
3. Initialize g(start) = 0.
4. Calculate f(start) = g(start) + h(start) using Manhattan distance.
5. Push the start node into the open set.

Step 2: Main Loop

6. While the open set is not empty:
   a. Pop the node current with the lowest f(n) from the open set.
   b. If current is the goal node, reconstruct and return the path.
   c. Add current to the closed set.
   d. For each valid neighbor of current (up, down, left, right):
      i. Skip if the neighbor is an obstacle or already in the closed set.
      ii. Calculate:

tentative = g (current) 1
h =| x_goal - x_neighbor| + |y_goal - y_neighbor|
f = tentative_g +h
If neighbor is not in open set or has a better g value:

   i. Update g(neighbor), f(neighbor)
   ii. Set cameFrom[neighbor] = current
   iii. Add neighbor to open set (or update priority)

Step 3: Path Reconstruction

1. Once the goal is reached, backtrack from the goal using came from [] map to build the shortest path.
2. Return the path as a sequence of grid coordinates.

However, if H(n) is too low or underestimated, the algorithm relies more heavily on G(n), causing it to behave more like Dijkstra's algorithm. This results in examining more nodes and increases computational time, ultimately reducing the efficiency of the A* search. Thus, selecting an appropriate and accurate heuristic is essential to maintain A*'s effectiveness and speed. For our case, we had considered the Manhattan distance between the current node and the goal node as the heuristics.

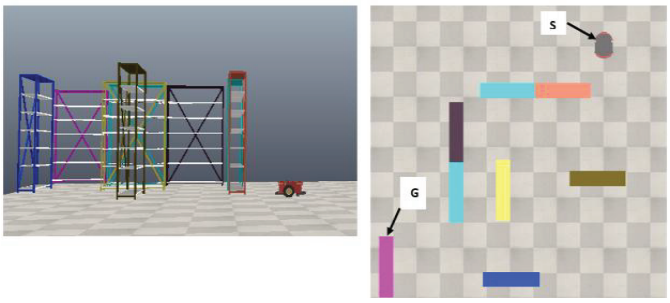

Figure 1. Simulation setup in CoppeliaSim

Algorithm 4: Heuristic Function

Step 1: Initialization

1. h(n)=|x. goal-xn|+|y. goal-yn| = abs (x. goal current) +abs (y. goal - current)
2. Pseudo-code
3. function A_Star (start, goal, grid):
4. openSet ← priority queue with (start, f(start))
5. cameFrom ←empty map
6. gScore[start] ←0
7. fScore[start] ← h (start, goal)
8. while openSet is not empty:
9. current ←node with lowest fScore in openSe
10. if current == goal: return reconstruct path (cameFrom, current) remove current from openSet for each neighbor in get_4_neighbors (current, grid): if neighbor is obstacle: continue tentative_gScore ← gScore[current] + 1 // move cost is 1 if neighbor not in gScore or tentative_gScore < gScore[neighbor]: cameFrom[neighbor] ←current gScore[neighbor] ←tentative_gScore fScore[neighbor] ←gScore[neighbor] + h(neighbor, goal) if neighbor not in openSet: add neighbor to openSet return failure // no path found
11. function h (node, goal): return abs (node.x - goal.x) + abs (node. y goal. y)
12. function get_4_neighbors (node, grid): neighbors ← [] for direction in [(0, 1), (1, 0), (0, −1), (−1, 0)] : nx ← node. x+ direction. x ny ←node. y+ direction. y if within grid (nx, ny) and not grid[nx][ny].is obstacle: neighbors. Append ((nx, ny)) return neighbors
13. function reconstruct path (cameFrom, current): path ←[current]

5. Simulation Environment in CoppeliaSim

The Pioneer P3DX mobile robot model is taken from the library of CoppeliaSim. It is a differential drive robot having 16 sonars (ultrasonic sensors). It is 485 mm long, 381 mm wide, and 217 mm high [25,26].

Pioneer P3DX is simulated for the navigating from a start point(S) to a goal point (G) using Dijkstra's algorithm and A* algorithm in CoppeliaSim version 4.7.0 [24]. It is free software for academic research purposes. The software offers a remote API interface with Python. The ODE physics engine is selected for the simulation of robot with a 0.001 s time step.

The work frame of CoppeliaSim is designed as a warehouse layout, as shown in Figure 1. Eight racks are imported in the simulation environment, and the mobile robot reaches to the target rack.

The complete control algorithms are developed in a Python API. The dimensions of the rack is 6 m long, 0.6 m wide, and 4 m high and has six racks. The grid size for both algorithms (Dijkstra and A*) is taken as 10×10. Each cell is 1×1 m^2.

Nodes correspond to key locations like intersections and storage areas, while edges denote the connecting routes between them. Edges are given weights according to either the distance between nodes or the estimated travel time along the path.

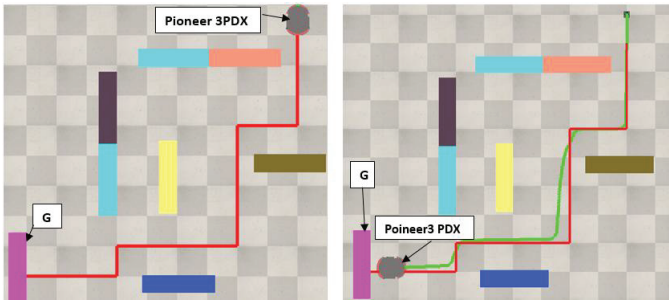

Figure 2. Simulation of Dijkstra algorithm in CoppeliaSim

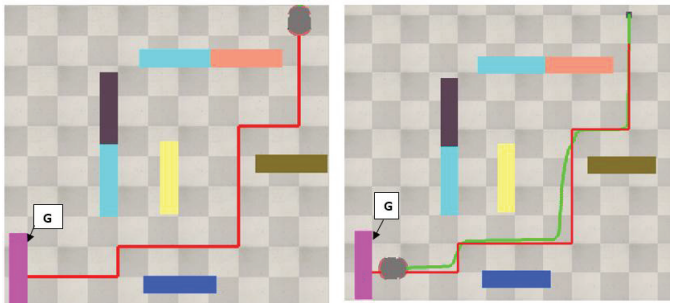

Figure 3. Simulation of A^* algorithm in CoppeliaSim

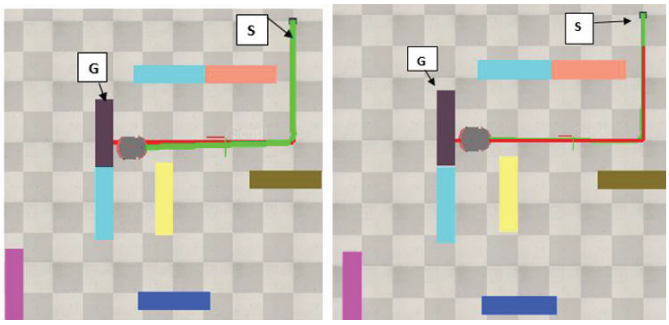

Figure 4. Implementation of algorithms in mobile robot for target rack $(6, 4)$

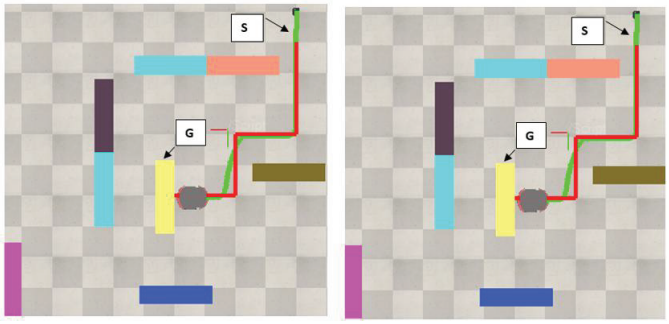

Figure 5. Implementation of algorithms in mobile robot for target rack $(4, 6)$

For our simulation, to maintain uniformity, we have kept the weight as 1 for each neighboring grid. The start position is the initial position of the Pioneer P3DX. The start position and the goal position are $(0, 0)$ and $(9, 9)$ for initial simulations. In addition, two more studies have been done for target racks at $(6, 4)$ and (4,6), respectively.

The mobile robot is placed at the origin of the frame layout. The path is determined by following the ordered list of nodes. Further, the control logic is designed in Python to follow the path to reach the goal rack. A separate simulation is run for the implementation of Dijkstra algorithm and A* algorithm in mobile robotics to search the shortest path.

6. Result

Path planning is a crucial aspect of robot navigation. It ensures that a robot can find the most efficient route from a start position to a goal while avoiding obstacles.

The Dijkstra algorithm was successfully implemented in a mobile robotic system within a simulated environment using CoppeliaSim. The red line in Figure 2 shows the shortest path to reach the goal rack. The path was achieved through both algorithms is: [(0,0),(0,1),(0,2),(0,3),(0,4),(1,4),(2,4),(2,5),(2,6),(2,7),(2,8),(3,8),(4,8),(5,8),(6,8),(6,9),(7,9),(8,9),(9,9)]. The obstacles were: = [(5,9),(3,9),(1,5),(7,5),(4,2),(8,2),(4,7),(0,0),(0,6),(2,3)]. The robot was able to follow the shortest path from the defined start location to the goal while avoiding obstacles placed in the environment. The Pioneer 3PDX has ultrasonic sensors to detect the obstacles. Threshold value for the sensors are written in the control logic to move the robot properly.

The environment was discretized into an uniform 2D grid, where each cell represented a possible robot location. Nodes were generated for all traversable grid cells, and edges were created based on adjacency (four-connected neighbors) for the search graph.

The trajectory of the mobile robot (green) followed the shortest path. It was noted that there is a variation in the trajectory of the mobile robot. The distance between the start point to the goal point is 24 m , which was determined by the Dijkstra algorithm. The robot traveled the same distance in 58 s. A separate simulation was performed with the A* algorithm in the same environment.

The simulation environment included predefined start and goal positions, along with a grid-based representation of the workspace, where each grid cell was assigned a cost value based on its distance from the goal and any obstacle present, as shown in Figure 4.

The simulation results validate the effectiveness of the A* algorithm for grid-based path planning in structured environments. The heuristic function played a significant role in optimizing performance and ensuring goal-oriented search behavior. The computational time for the mobile robot to cover the same distance was 47 s to reach the target. A* algorithm with an admissible heuristic guarantees optimality.

Target rack has been placed in two different nodes (6,4) and (4,6), respectively. The simulations are run to study the performance of both algorithms. Figure 3 shows the result of both algorithms (Dijkstra at left and A* at right). The path for this purpose is [(0,0),(0,1),(0,2),(0,3),(0,4),(1,4),(2,4),(3,4),(4,4),(5,4),(6,4)].

The shortest distance covered by the mobile robot to reach the target rack in the warehouse is 15 m in 43 s with the Dijkstra algorithm, whereas in the A*star algorithm, the same distance was covered in 39 s.

The result of both algorithms (Dijkstra on the left and A* ion the right) at the target node (4, 6), as shown in Figure 5. The path for this study is [(0,0),(0,1),(0,2),(0,3),(0,4),(1,4),(2,4),(2,5),(2,6),(3,6),(4,6)]. The shortest distance traveled by the mobile robot to reach the target rack in the warehouse is 19 m in 48 s with the Dijkstra algorithm and 41 s with the A* algorithm.

Dijkstra's algorithm is a classic approach for finding the shortest path in a weighted graph with non-negative weights. A*, on the other hand, enhances Dijkstra by using heuristics to prioritize which paths to explore, making it more efficient in many cases.

7. Discussion

The comparative implementation of the Dijkstra and A* algorithms for mobile robot path planning in a warehouse environment using CoppeliaSim revealed significant differences in performance, computational efficiency, and practical suitability for structured spaces like warehouses.

Three different cases have been studied in the warehouse environment. The target rack has been placed in nodes (9,9), (6,4), and (4,6), respectively. It was noted from Table 1 that the computation time for A* star algorithm is lower in all these cases for the robot to arrive at the target point while avoiding the obstacles.

Table 1. Computational time to reach the target rack

Target Node	Computational Times (s)		
	Dijkstra algorithm	A* algorithm	% difference in time
(9, 9)	58	47	23
(6, 4)	43	39	10
(4, 6)	48	41	17

Table 1 also shows that the difference in computational time is proportional to the distance between the start node and the goal node. Both algorithms successfully generated collision-free paths from the start to the goal point, avoiding static obstacles.

Dijkstra's algorithm, while reliable, showed increased latency in larger maps, as it computes the shortest path to all nodes before identifying the optimal route to the goal. This makes A* more suitable for real-time applications, where quick response and adaptability are crucial.

Both algorithms were effectively implemented using embedded scripts in CoppeliaSim, leveraging integration with external Python APIs via the remote API. The simulation environment allowed visualization of robot paths, node exploration, and obstacle avoidance. A* required the addition of a heuristic function, which added minor complexity but significantly improved performance.

Table 2. Summary of research work

Criteria	Distance Traveled (m)	
	Dijkstra algorithm	A* algorithm
Path optimality	Good	Better
Speed	Slower	Moderate
Heuristic use	No	Yes
Computational cost	Higher	Lower
Implementation ease	Easy	Moderate

While both algorithms functioned well in simulation, A* proves more practical for real-world warehouse scenarios. It offers better adaptability for static environments, especially when combined with real-time replanning or sensor feedback. Dijkstra, although deterministic and optimal, lacks the responsiveness required for such conditions unless heavily modified, as summarized in Table 2.

8. Conclusion

In the realm of autonomous mobile-robot navigation within warehouse environments, the A* The algorithm has garnered significant attention due to its efficiency and optimality in path planning. Recent literature underscores A*'s superiority over traditional algorithms like Dijkstra, particularly in scenarios demanding realtime decision-making and energy efficiency. For instance, studies have demonstrated that A* not only reduces computation time but also maintains path optimality, making it a preferred choice for mobile robots operating in complex warehouse settings.

Traditionally, many researchers have focused on 2D representations of warehouse layouts for path planning. However, the dynamic nature of modern warehouses necessitates a more comprehensive approach. Addressing this, recent research has introduced 3D environmental modeling, where racks and other obstacles are considered in three dimensions, providing a more realistic simulation of warehouse conditions. Simulations conducted using CoppeliaSim have highlighted the effectiveness of graph-based pathplanning algorithms, particularly A* and Dijkstra, in navigating the Pioneer P3DX robot through complex warehouse environments. By selecting three distinct target nodes within the warehouse racks, researchers have been able to assess the accuracy and efficiency of both algorithms. While both A* and Dijkstra are capable of finding optimal paths, A^* has demonstrated a significant advantage in terms of computational efficiency, primarily due to its heuristic-driven approach. This efficiency is crucial for real-time robotic applications where rapid decision-making is essential.

The implementation of these simulations underscores the utility of platforms like CoppeliaSim in prototyping and validating robotics algorithms before deploying them in realworld scenarios. Such simulation environments allow for thorough testing and refinement, ensuring that the algorithms perform reliably under various conditions. Moreover, the use of 3D modeling in these simulations provides a more

accurate representation of actual warehouse environments, leading to more robust and applicable results.

Looking ahead, future research can explore the development of hybrid algorithms that combine the strengths of various path-planning methods to enhance performance further. Additionally, integrating dynamic replanning capabilities and incorporating simultaneous localization and mapping techniques, along with real-time sensor data, can significantly improve the autonomy and adaptability of warehouse robots. Such advancements will enable robots to navigate more efficiently in ever-changing warehouse environments, ultimately leading to increased productivity and operational efficiency.

AUTHORS

Prabin Kumar Jha* – Department of Electronics and Commincation Engineering, ASET, Amity University, Bengaluru, India, e-mail: pkjha@blr.amity.edu.

Shambo Roy Chowdhury* – Department of Robotics and Automation Engineering, The Neotia University, Kolkata, India, e-mail: shambo.roychowdhury@tnu.in.

*Corresponding author

References

[1] S. Alshammrei, S. Boubaker, L. Kolsi "Improved Dijkstra Algorithm for Mobile Robot Path Planning and Obstacle Avoidance," *Computers, Materials and Continua*," vol. 3, no. 72, 2022, pp. 5939-5954; https://doi.org/10.32604/cmc.2022.028165.

[2] D. Xiang, H. Lin, J. Ouyang, J. et al., "Combined Improved A* and Greedy Algorithm for Path Planning of Multi-objective Mobile Robot," *Scientific Reports*, vol. 12, 2022, 13273; https://doi.org/10.1038/s41598-022-17684-0.

[3] S.A.M. Abgenah and A.J. Azrul, "Multi-Robot Path Planning Using Dijkstra's Algorithm with Multilayer Dictionaries," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 10, 2021.

[4] Q. Ma, et al., "Path Planning for Mobile Robots Based on Improving Dijkstra Algorithm," *Proc. SPIE 12712, International Conference on Cloud Computing, Performance Computing, and Deep Learning* (CCPCDL 2023), 127121F, 2023; https://doi.org/10.1117/12.2678830.

[5] K. Li, et al., "Research on Reinforcement Learning Based Warehouse Robot Navigation Algorithm in Complex Warehouse Layout," arXiv, 2024; https://arxiv.org/abs/2411.06128.

[6] X. Xing, et al., "Research on Global Path Planning Algorithm for Mobile Robots Based on Improved A*," *Expert Systems with Applications*, vol. 243, 2024; https://doi.org/10.1016/j.eswa.2023.122922.

[7] S. Vaish, Shreyam, and S. Singhal, "Segmented Approach to Path Planning," *2020 Sixth International Conference on Parallel, Distributed and Grid Computing*" (PDGC), 2020, pp. 266-271; https://doi:10.1109/PDGC50313.2020.9315788.

[8] A. Ubaidillahand H. Sukri, „Application of Odometry and Dijkstra Algorithm as Navigation and Shortest Path Determination System of Warehouse Mobile Robot," *Journal of Robotics and Control*, vol. 4, no. 3, 2023, pp. 413-423; https://doi.org/10.18196/jrc.v4i3.18489.

[9] Y. Tang, M.A. Zakaria, and M. Younas, „Path Planning Trends for Autonomous Mobile Robot Navigation: "A Review," *Sensors*, vol. 25, no. 4, 2025, 1206; https://doi.org/10.3390/s25041206.

[10] Z. Duraklı and V. Nabiyev, "A New Approach Based on Bezier Curves to Solve Path Planning Problems for Mobile Robots," *Journal of Computational Science*, vol. 58, 2022, 101540; https://doi.org/10.1016/j.jocs.2021.101540.

[11] J. Li et al., "Fusion of Improved A* Algorithm and Dynamic Window Approach for Path Planning," *2022 China Automation Congress* (CAC), 2022, pp. 390-394; doi: 10.1109/CAC57257.2022.10056090.

[12] B. Wu et al., "Dynamic Path Planning for Forklift AGV Based on Smoothing A* and Improved DWA Hybrid Algorithm,", *Sensors*, vol. 22, 2022, 7079; https://doi.org/10.3390/s22187079.

[13] Y. Shi, S. Huangand M. Li, "An Improved Global and Local Fusion Path-Planning Algorithm for Mobile Robots," *Sensors*, vol. 24, 2024, 7950; https://doi.org/10.3390/s24247950.

[14] Y. Zhou and N. Huang, "Airport AGV Path Optimization Model Based on Ant Colony Algorithm to Optimize Dijkstra Algorithm in Urban Systems," *Sustainable Computing, Informatics and Systems*, vol. 35, 2022, 100716; https://doi.org/10.1016/j.suscom.2022.100716.

[15] R. Kumar et al., "Performance Comparison of A* Search Algorithm and Hill-Climb Search Algorithm: A Case Study," *Multifaceted approaches for Data Acquisition, Processing & Communication*, 1st ed. CRC Press, 2024, pp. 185194,; https://doi.org/10.1201/9781003470939.

[16] C. Cheng et al., „Path Planning and Obstacle Avoidance for AUV: A Review," *Ocean Engineering*, vol. 235, 2021, 109355; https://doi.org/10.1016/j.oceaneng.2021.10935.

[17] H. Du et al., "Multi-Objective Loosely Synchronized Search for Multi-Objective MultiAgent Path Finding with Asynchronous Actions," *Journal of Shanghai Jiaotong University (Science)*, vol. 29, 2024, pp. 667-677; https://doi.org/10.1007/s12204-024-2744-x.

[18] L. Ait Ben Mouh et al., (2023). "A Hybrid Approach of Dijkstra's Algorithm and A* Search, with an Optional Adaptive Threshold Heuristic,"

Business Intelligence, R. El Ayachi, M. Fakir, and M. Baslam, eds. Springer, 2023; https://doi.org/10.1007/978-3-031-37872-0_9.

[19] P.M. de A. Brasil et al., "Dijkstra and A* Algorithms for Global Trajectory Planning in the TurtleBot 3 Mobile Robot," *Advances in Intelligent Systems and Computing*, Springer, 2020; https://doi.org/10.1007/978-3-030-711870_32.

[20] J. S.S. Pavithra et al., "Contrastive Analysis of Path Planning Algorithms for Mobile Robots: A Case Study," *2023 International Conference on Energy, Materials and Communication Engineering* (ICEMCE), 2023, pp. 1-6; doi: 10.1109/ICEMCE57940.2023.10434168.

[21] A.R. Warrier et al., "Implementation of Classical Path Planning Algorithms for Mobile Robot Navigation: A Comprehensive Comparison," *2022 International Conference on Electrical, Computer, Communications and Mechatronics Engineering* (ICECCME), 2022; https://doi.org/10.1109/iceccme55909.2022.99.88092.

[22] T. Yoshida et al. „Automated Harvesting by a Dual-arm Fruit Harvesting Robot," *Robomech Journal*, vol. 9, no. 19, 2022; https://doi.org/10.1186/s40648-022-00233-9.

[23] H.S. Yang, "Analysis and Study on Path Planning Algorithms in the Further Mobile Action," *Journal of Physics*, vol. 2824, no. 1, 2024, 012006; https://doi.org/10.1088/17426596/2824/1/012006.

[24] A.A.F. Sakri et al. „Autonomous Person-Following Telepresence Robot Using Monocular Camera and Deep Learning YOLO," *Applications of Modelling and Simulation*, Vol. 8, 2024, pp. 101109.

[25] *Robot Simulator CoppeliaSim: Create, Compose, Simulate, Any Robot*, Coppelia Robotics.

[26] Webots Documentation: Adept's Pioneer 3-DX.