International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

Comparative Analysis of Path Planning Algorithms in a ROS-based Navigation Framework for Industrial 4.0 Environments

Ankur Bhargava[1st] ,Chirag Khanna[2nd], Daksh Jain[3rd], Ajay K.S. Singholi[4th]
University School of Automation & Robotics, Guru Gobind Singh Indraprastha University, Delhi, India

Abstract—Although waypoint traversal is essential for autonomous mobile robots to navigate, grid-based algorithms, such as Dijkstra's and A*, produce jagged, suboptimal paths when limited to only a handful of movement directions. Thus, this work is the first to address the epistemological gap regarding empirical performance studies of pathfinding algorithms operating within modern, complex robotic middleware. Within this work, we present an extensive empirical comparison of Dijkstra's, A*, and the any-angle Theta* algorithm, which we developed as a custom plugin integrated into the ROS Navigation 2 (Nav2) stack. Our contribution was to develop a standardized, open-source C++ framework in which we conduct simulation experiments on the TurtleBot3 in the Gazebo simulator, using realistic 2D SLAM-derived costmap 2D and costmap 2D ROS messages, primarily for robot path planning. A sophisticated quantitative benchmarking was conducted, measuring algorithm performance in terms of operational efficiency and path planning quality. These findings showcase the indispensable value of any-angle planning in dynamic settings. Among the algorithms, A* demonstrated the highest efficiency, operating at 8.7 times the speed of Dijkstra's algorithm. Some comparisons of smoothest theta* paths to grid-based alternatives indicate that any-angle optimization can result in paths that are 6.7% shorter and may include drastically fewer waypoints, for instance, as opposed to indicating any-angle optimization has benefits for smooth and kinematically favorable trajectories. Out of the A* and Dijkstra comparisons, Theta* was most efficient in the search, exploring far fewer nodes in the process. Real-time autonomous navigation tasks present a new challenge for robotics and beyond. Theta* offers a compelling solution for the robotics community by providing the fastest response time to all alternative algorithms while maintaining a high level of path quality. The theta* algorithm provides all users in the robotics community with a reusable benchmarking framework and informed guidelines to select the most efficient pathfinding algorithm.

Keywords: Mobile Robot Navigation, Path Planning, Dijkstra's Algorithm, A* Algorithm, Theta* Algorithm.

1. Introduction

One of the most fundamental challenges in robotics is the autonomous navigation of mobile robots [1]. The ability for a mobile robot to operate autonomously over complex environments and to develop a collision-free path in real time is an advanced algorithm challenge. Autonomous navigation has a significant impact on warehouse automation and service robotics [2-3] and can be fundamental for self-driving cars and space exploration [4]. The fundamental pathfinding problem is at the heart of these navigation systems: a start position, a goal position, and a map of the environment. Determine an optimal or near-optimal route that avoids obstacles while minimizing travel cost [5]. The trajectory of most classical pathfinding algorithms has its roots in considering search techniques in graphs. Dijkstra's algorithm is a classic example of being able to optimally compute a path or numerous paths in a particular segment of discrete space [6]. The introduction of heuristics in guiding a search theoretically revolutionized how focused a search could be, although it drew the search more towards the goal and, hence, is often tied to an A* search [7], whereby comparably fewer nodes are evaluated in the search compared to the other nodes evaluated in the in-depth search. Both methods, however, are bound to use discretized grids; hence, they are limited in controlling motion to specified and often preferred angular orientations, generally bound to be 8-directional, and may include more non-optimal traversals. This gives rise to a disconnect or a zigzag path, which is suboptimal and inhibits the possibility of perhaps taking more direct routes. Furthermore, the suboptimality intensifies in environments where spatial configuration, such as an obstacle, influences the algorithm's performance. Such step-constrained paths produce unnatural motion in a mobile robot, revealing the shortcomings of pure grid-based planning [8]. This inconsistency between theoretical optimality in continuous space and the results at hand in discrete representations drove the creation of any-angle pathfinding algorithms that try to bridge this gap by providing direct line-of-sight connections between non-adjacent grid points. The figure below shows the layered architecture of an autonomous mobile robot.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

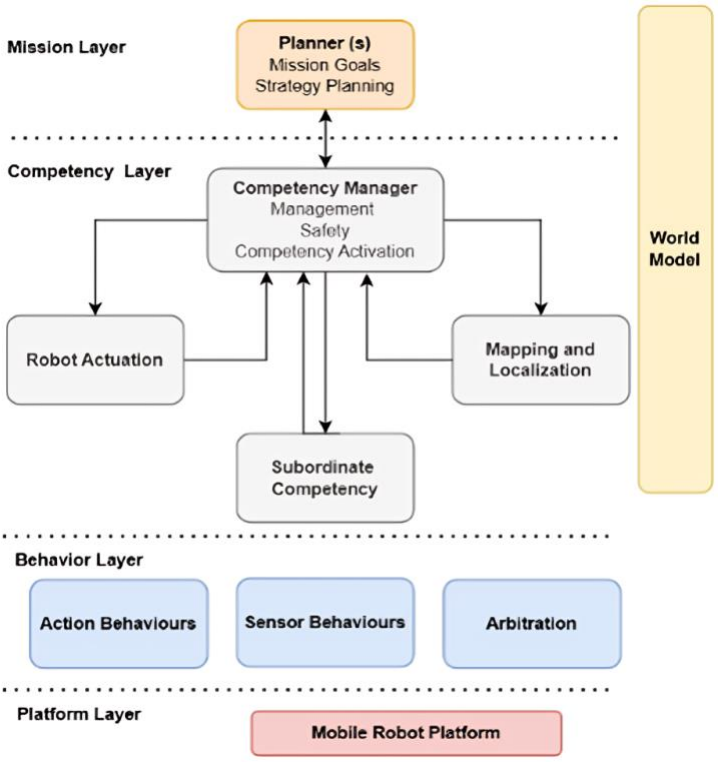

Figure 1: Layered Architecture of Autonomous Mobile Robot *(Source: Author's own work)*.

While there is a significant amount of theoretical analysis on specific pathfinding algorithms, a comprehensive empirical evaluation of the different approaches under modern robotic navigation frameworks is still lacking. Many existing studies focus on theoretical complexity or use highly simplified grid environments, which do not reflect the challenges of real-world navigation. Only a few studies have compared the practical performance of grid-based and any-angle pathfinding algorithms integrated with modern middleware (such as ROS2) and realistic costmap representations. Integration of pathfinding algorithms with robotic navigation systems introduces many complexities not usually considered in theoretical studies, including coordinate system transformations, real-time costmap updates, dynamic obstacle avoidance, and the overhead of middleware communication. Additionally, performance characteristics of different algorithms may change significantly with environmental parameters (e.g., obstacle density, map resolution, required path length). This necessitates a more generalized empirical study to cover a wide range of scenarios [9]. Fully developed robotic navigation frameworks, specifically the Nav2 stack of ROS2, already provide advanced costmap formations based on the available sensor data and active inflation and dynamic updates on obstacles. However, planners in such systems are usually off-the-shelf general-purpose implementations, not tailored to a particular application to maximize optimization. Understanding the basic pathfinding algorithms used in these established frameworks is essential for designing and optimizing navigation systems. The next section discusses the objective and scope of the research.

2. Literature Review

This is the first algorithm introduced by Edsger W. Dijkstra in 1959 and is the first and most important algorithm for optimal pathfinding in weighted graphs. It has remained fundamental to robotic navigation for over 60 years. It performs a systematic exploration that finds and guarantees a solution to the problem of pathfinding between any source and target nodes in a directed graph. This is an absolute necessity for safety-critical applications, as it guarantees an optimal solution to the path. Dijkstra's algorithm is based on the principle of optimal substructure. This principle states that the shortest path to any node is the one that goes through the shortest intermediate node and is also the shortest path to the target node. The algorithm processes nodes in the graph depending on their distance from the source node. For grid-based path planning, Dijkstra maintains a cumulative cost function:

$$f(n) = g(n)$$

where g(n) represents the cumulative cost from the start node to node n. For grid environments, g(n) typically represents the path length traveled.

$$g(n) = \sum_{i=0}^{n-1} d(i, i+1)$$

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

where d(i, i+1) is the distance between consecutive nodes. For grid cells, d equals 1 for orthogonal movement (four-connectivity) or √2 for diagonal movement (eight-connectivity). The algorithm functions through its ability to identify optimal solutions throughout discrete search spaces, which makes it suitable for tasks that need best-path solutions more than computational speed. The basic implementation of the algorithm requires $O(|V|^2)$ computational resources, yet the priority queue optimization reduces this requirement to $O(|V|\log|V| + |E|)$ (where |V| is the number of nodes and |E| is the number of edges), which still develops obstacles when trying to use the algorithm in real-time situations [4]. The large number of vertices in typical robotics costmaps, which reach hundreds of thousands, makes exhaustive exploration impossible for time-sensitive navigation tasks. Dijkstra's algorithm produces optimal results in theory, but users must evaluate system limitations when using it for real-time environmental adjustments. The researchers demonstrated through their warehouse environment mobile robot navigation study that the algorithm conducts an exhaustive search, which results in 40-60% of the search area being explored, thus producing excessive computation when quick, approximate solutions would suffice [12]. The A* algorithm emerged from Hart, Nilsson, and Raphael, who developed this method in 1968 to improve pathfinding through heuristic guidance, which leads searches toward their objectives while achieving better computational efficiency and optimal results under particular circumstances [13]. The algorithm breakthrough is obtained by the evaluation function $f(n) = g(n) + h(n)$. g(n) is the actual cost incurred to reach node n from the start, while h(n) is the estimated cost to reach the goal from n. A* maintains its theoretical best position as a result of both being admissible and consistent. A* search is admissible due to the heuristics, as it provides estimates of costs that are lower than the actual costs to the goal. The algorithm works with consistency, as the nodes revisited are not out of focus. The combined features of A* provide the best optimality and efficiency in robotic system operations. A* heuristic function selection for robotic navigation results in 60-80% fewer node explorations than Dijkstra's algorithm while achieving optimal solutions [14]. The performance of a system depends on the heuristic function selection because Manhattan distance works best for grid-based navigation with only straight movements, and Euclidean distance functions better for spaces that allow diagonal movements. Recent studies have shown that researchers have analyzed multiple A* optimization methods to enhance their performance in robotics applications. Koenig and Likhachev developed D* Lite in 2002, which operates as an incremental A* variant to handle changing environments through the preservation of previous search results [15]. Their research demonstrated that Incremental methods achieve 10-50x speedup when working with changing environments, which makes them useful for robot navigation through changing spaces. The A* algorithm faces fundamental problems when it operates with grid-based representations. The requirement to follow specific movement directions (usually 8-directional) leads to paths that stick to grid boundaries, which produces suboptimal routes that differ substantially from the actual shortest paths in continuous environments. The problem worsens in open areas because discrete grid systems fail to generate direct paths, which represent the most efficient travel routes [16].

The algorithm represents a new method for pathfinding at any angle, which has become popular in robotics research because of its straightforward design and successful results. Theta* develops any-angle paths by forming direct line-of-sight connections between non-adjacent grid points. This results in shortcuts that skip intermediate nodes, unless obstacles are in the way. The primary innovation of Theta* comes from optimizing parent pointers. In the expanding process of a node, most search algorithms only consider pathways through the current node. Theta*, however, takes into account direct line-of-sight pathways from the current node's parent to the successor nodes. If a direct line-of-sight path has no obstacles and offers a lower-cost alternative to grid-based pathways, the algorithm changes a successor's parent pointer to the earlier ancestor instead of the current adjacent one. The advantages of the line-of-sight approach are particularly significant compared to the grid-based methods. Theta* produces paths 5-10% shorter than A* while using similar resources. In almost every real-life situation, Theta* performs much better than A* in terms of quality of paths, and it maintains the same worst-case complexity. The core computational requirement for Theta* to optimally function is the line of sight checking process. Bresenham's line algorithm and similar rasterization techniques serve as standard methods for identifying which cells fall on a straight line between any two given points. Although checking costs more than a basic grid-based expansion, the method is more efficient as it decreases the number of cells that need to be traversed due to line-of-sight connections. [18]. Recently, there have been many attempts to improve the basic version of the Theta* algorithm. The researchers found Lazy Theta* to be effective in reducing computation time by 20-40% while keeping path quality the same, which is relevant for applications in dynamic environments for robotics [19].

The use of standardized communication protocols and plug-and-play software components, DIY software for robotics, has become more accessible due to the ROS (Robot Operating System). Since the introduction of ROS 1 in 2010, the navigation stack has become the framework for the navigation of autonomous mobile robots, providing a complete solution for research and industrial standards. The original ROS navigation stack architecture established essential robotic navigation concepts, which include costmaps, global and local planners, and recovery behaviors that continue to form the basis of present-day robotic navigation systems. The modular structure of their system

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

allows researchers and developers to add new pathfinding algorithms while using existing infrastructure for sensor processing, localization, and control. The transition to ROS 2 and the development of Navigation 2 (Nav2) have addressed many limitations of the original navigation stack while maintaining backward compatibility for core concepts. Nav2's system improvements enhance its suitability for research requirements that necessitate the implementation of specific pathfinding algorithms. The plugin-based structure of Nav2 allows users to add custom pathfinding methods through plugins without needing to alter the fundamental navigation system. The architectural decision has enabled multiple research studies to develop new pathfinding methods that operate within current robotic navigation systems. The framework presents difficulties for researchers who want to use custom algorithms because of its complex nature, which requires them to focus on interface requirements and data format specifications [20]. Costmap 2D is a Nav2 module that builds layered representations of a given environment by using multiple costmaps, integrating sensor information to track and fuse dynamic obstacles, and performing sensor data-driven obstacle inflation. Furthermore, this module combines obstacle inflation with sensor data. Each individual layer of a costmap is instrumental in determining the pathfinding process, factoring in the robot's dimensions, sensor precision, and a safety buffer. The Nav2 framework has four costmap layers, including a social navigation layer and a terrain assessment layer. The other layers comprise obstacle layers that utilize real-time sensor data, inflation layers that develop safe zones around obstacles, and layers that incorporate static maps made from SLAM occupancy grids. Integrating the data layer enhances the representation of the environment, providing the pathfinding algorithms with more informed choices. A crucial evaluation measure for the pathfinding algorithms involves the inflation process used by them. The inflation process determines the free space around connected obstacles and the optimality of the paths with regard to the robot footprint and safety buffer. It calculates the boundaries of obstacles and determines the connected space along the robot's safety buffer. According to the research [21], the frequency of operations of Nav2 global cost maps is relatively low, while the levels of their coverage are high. Consequently, they are ideal for implementing global path planning algorithms, as investigated in this research. Local cost maps, in contrast, operate with high frequency and within small coverage areas in order to assist with obstacle detection and immediate path planning. The correct interpretation of these costmap types allows for successful integration of algorithms and performance evaluations [20]. The figure below shows the architecture of environments.

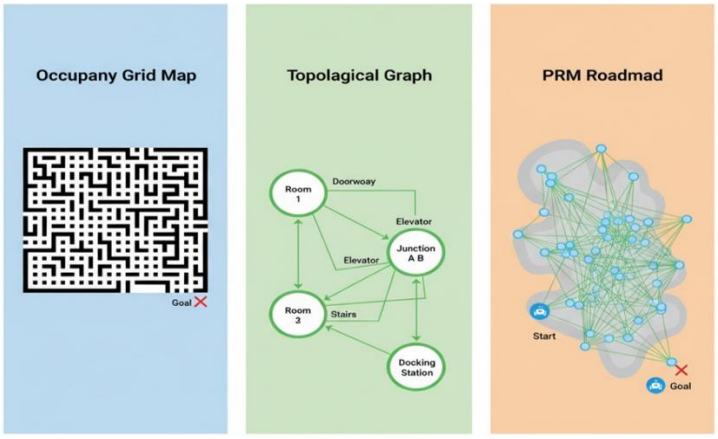

Figure 2: Architecture of environments *(Source: Author's own work)*.

Pathfinding techniques require simultaneous localization and mapping (SLAM) to navigate unknown or partially known environments where there is no prior knowledge of the surroundings. Integrating SLAM technology with pathfinding techniques is both positive and negative, impacting the navigation system's operational performance [22]. The SLAM toolbox is the default 2D SLAM implementation for ROS 2 and possesses excellent SLAM capabilities for evaluating pathfinding algorithms. The Karto SLAM algorithm, which includes loop closure detection, enhances the SLAM toolbox's ability to construct occupancy grid maps that Nav2's costmap can use for pathfinding algorithms requiring consistent environmental updates. Pathfinding algorithm performance is contingent on the three cardinal qualities of a map: resolution, accuracy, and completeness. Scholars have documented relationships between the quality of a map and the performance of navigation, with a major component of the map's quality being its resolution, which is a critical factor for computational performance versus path planning accuracy [22]. The scholars illustrated the effect of resolution on pathfinding by altering the connections of free space and increasing the size of obstacles on the map. The

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

implementation of dynamic map updates develops new challenges for pathfinding algorithm developers. SLAM systems improve their maps through sensor data, yet pathfinding algorithms operate with static environmental models during planning. Research studies on dynamic pathfinding techniques have investigated methods to handle map changes during planning, yet most operational systems rely on snapshot-based methods, which perform planning on short-term static map copies.

Pathfinding algorithm evaluation requires particular performance metrics that assess both computational efficiency and solution quality. As a result of a variety of operational constraints and application requirements, the research community has set several benchmarks, which carry varying degrees of significance. The analysis focuses on three primary considerations for computational efficiency: planning time, memory utilization, and the nodes that a specific search algorithm must traverse. For a given system, planning time is the most critical of the metrics, as there are real-time constraints. In the case of a resource-constrained robot, memory utilized is a crucial factor. In general, unexplored nodes are a metric that can be used regardless of the framework. Path quality is typically assessed in terms of three primary parameters: the length of the path, the smoothness of the path, and the safety features of the path. A route that is shorter in length will facilitate faster travel and conserve energy, but it also needs to be analyzed to ensure it is not too simple to meet the smoothness and robot kinematic constraints [23]. The evaluation of pathfinding algorithm performance needs to compare simulated results with actual real-world operational results. The controlled experimental conditions and reproducible results of simulation become difficult to replicate when moving from simulation to physical robotic systems due to new system complexities. The robotics research field employs Gazebo simulation to model real-world robotic navigation through its advanced physics modeling and sensor simulation features [24]. The simulation environment lacks complete replication of physical robot operation because it fails to accurately model sensor noise behavior, actuator constraints, and environmental factors that affect pathfinding results. The research framework demonstrates that simulations produce useful performance data for algorithms; however, physical robot testing is still necessary to validate these results for real-world applications [25]. Scientists have performed various studies to determine how simulated navigation systems differ from actual navigation systems in real-world environments. Researchers performed a comparative study analyzing pathfinding performance between simulated and real-world settings and discovered that simulation results show 10-20% higher performance than actual results due to perfect sensor models and ideal actuator responses. The algorithms maintain their relative performance order when tested in simulations and real-world environments. The scientific community backs ongoing robotic navigation benchmarking research through multiple programs, which develop shared evaluation criteria and testing resources. These solutions work toward improving the reproducibility of research, allowing scientists to assess research findings across several research groups. The series of RoboCup competitions has developed several challenges focused on navigation and offers benchmarked scenarios for autonomous mobile robots. The competitions assess complete navigation systems, but they do not cover every aspect, as pathfinding is also necessary. They allow researchers to assess the operational needs and constraints of systems for real-world use. Researchers have developed new benchmarks for pathfinding problems in robotic navigation systems. Bhargava et al. developed a comprehensive benchmarking framework for the analysis of pathfinding strategies in navigation systems based on ROS. It narrows the gap between the theoretical algorithm analysis and practical robotic implementation by offering integration and evaluation standards [26]. In response to the robotics research reproducibility crisis, researchers focused on open-source implementations and the development of standards for evaluation. The IEEE Robotics and Automation Society has developed guidelines on reproducible robotics research that state researchers are to provide a complete description of implementations and details of datasets, methodology, and evaluation used. These guidelines are crucial components for pathfinding algorithm research. This research examines the gap by reviewing three basic pathfinding algorithms—Dijkstra's algorithm, A* algorithm, and Theta* algorithm—in detail. This research has the following main objectives, which have been systematized:

Implementation and Integration: Construct an implementation procedural framework to benchmark pathfinding algorithms in the ROS2 navigation system using costmap_2d to model the environment consistently. This requires the construction of a software framework that is modular in design, enabling benchmarking to be performed in a comparative and yet standardized way for different robotic navigation systems.

Performance Profiling: Evaluation of different algorithms based on performance metrics such as planning time, memory usage, computational efficiency, path smoothness, optimality and length in the quality of path, and the behaviors of the algorithms based on the nodes that have been explored, success, and failure rate. This evaluation is conducted across different complex environments with varying lengths of the path to develop a comprehensive profile of the performance of the algorithm.

Real-world Uses: Finding the balance between the computational cost and the quality of the path in realistic robotic simulation environments and providing evidence-based guidelines on algorithm selection for different

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

operational contexts. This includes analyzing real-time performance constraints and resource utilization patterns typical of autonomous navigation applications.

Contribution to Framework: Establish an open-source, extensible framework for researching pathfinding algorithms in the ROS2 environment, enabling future comparison studies and educational uses in robotics research and development.

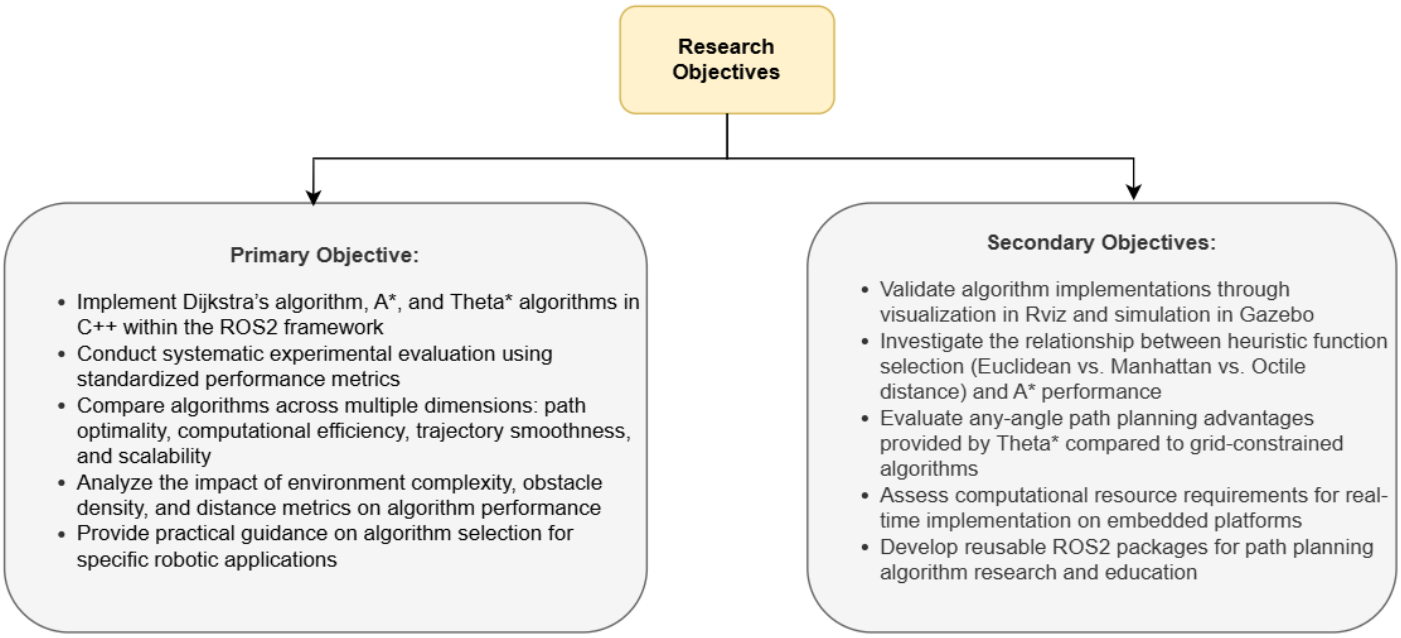

Figure 3: Research Objectives *(Source: Author's own work)*.

These constraints help guide our research as we analyze pathfinding and where we consider our self-navigation systems operationally limited to 2D pathfinding and resolving static occupancy grid situations. While the ability to circumvent dynamic obstructions and to solve pathfinding in 3D environments would be valuable, they lay outside the scope of this research. Excluding these variables allows for simpler studies on temporal dynamics and 3D complexity. Fairness of the experiments, coupled with the ability to replicate the constructs and methodologies to be used to assess the algorithms, will be established based on the Gazebo simulator and the Turtlebot3 as a sample platform. This research will be using the ROS 2 Humble middleware version as a default middleware, as this version supports the flexible and more dynamic seamless integration of communication and navigation functionalities in the Nav2 stack [10]. The 2D simulator included in the Gazebo environment is coupled with the SLAM toolbox, of which the grid resolution (0.05 m) is suitable for pathfinding and occupancy grid analysis [11]. The algorithmic modules of the pathfinding systems are built to allow the core algorithms of pathfinding to function with no dependencies on the ROS2 integration components so that performance comparisons reflect algorithmic differences rather than implementation variations. Each algorithm operates on identical costmap representations and utilizes consistent coordinate transformation procedures; this prevents bias in comparative analysis. A fully automated logging system records data collection through the logging of different data for performance metrics from the system in JSON format so that the data can be analyzed statistically and so that the results can be reproduced. Diverse operational conditions from multiple test cases are analyzed, considering different path lengths, varying complexities of environments, and distinct goals for robustness in performance characterization. The figure below shows the autonomous navigation pipeline overview.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

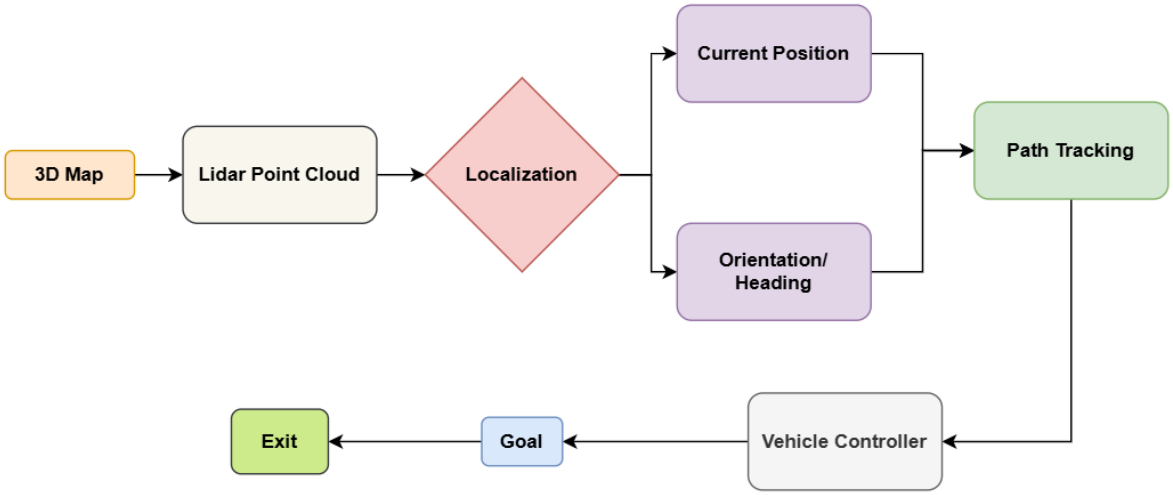

Figure 4: Autonomous Navigation Pipeline Overview *(Source: Author's own work)*.

This work, as an analytical contribution, serves autonomous sector robot navigation and pathfinding algorithms in several ways:

- Development of a comprehensive and fully integrated ROS 2-based framework to allow, for the first time, seamless and equitable comparison of pathfinding algorithms within the context of real robotic navigation. This work was designed to address the absence of evaluation benchmarks in the robotics community and has established a foundation for the extraction and further assessment of algorithms and more advanced robotics.
- Quantitative evidence to demonstrate the comprehensive analytical performance metrics of a range of algorithms across multiple dimensions of operational data. Crucially, it was found that A* was dominant in computational performance and efficiency, with a substantial 8.7x speed increase over Dijkstra's, and also that Theta* had the best results in integrative performance efficiency.
- This is the first step that has demarcated the process of integrating custom pathfinding algorithms with validated robotic navigation frameworks, which is likely to have an invaluable impact on the work of researchers and practitioners who wish to incorporate novel navigation approaches into their systems.
- Performance and empirical characterization of algorithm behavior across different scenarios and conditions are done in diverse environmental settings, and issue complexities provide analytical reasoning for algorithm selection for real-world applications. This encompasses determining performance crossover points where varying algorithms are most efficient according to operational demands.
- The systematic study of algorithm failure patterns and their limitations elucidates the robustness characteristics and the operational frontiers of each approach. This type of study is fundamental to building fault-tolerant navigation systems. An educational open-access resource that illustrates the real-world deployment of foundational pathfinding algorithms within contemporary robotics software, which is of value for research and teaching purposes in the robotics education curriculum.

3. Methodology

This system architecture utilized ROS 2 modular design principles to facilitate communication between the perception, localization, path planning, and control subsystems. The modular design of ROS 2 allows individual functional blocks to operate independently and still communicate with one another through topics, services, and actions. The design also allows developers to implement new modules with no changes to the current system architecture, facilitating new functionality for the system while maintaining its scalability and flexibility.

The system is built on the ROS 2 Humble distribution, which is the middleware layer communicating between nodes. ROS 2 depends on Data Distribution Service (DDS) as its base communication protocol to provide reliable and real-time data exchange between nodes. The design of each system component makes it possible for them to function independently, promoting distributed and simultaneous system operations. The navigation process involves five critical components: The map server, which distributes either a static or dynamic occupancy grid map; the localization node (AMCL), which finds and tracks the robot's position in the global map; the planning server, which provides the path to the goal; the controller server, which converts the path into local motion commands; and the behavior tree navigator, which orders and prioritizes tasks, makes decisions, and manages system failures. The nodes interact with each other to provide adaptive navigation using ROS 2 actions and topics. The TurtleBot 3 Burger model was chosen to serve as the base mobile robot platform for the execution and evaluation of the navigation algorithms. Its design with a small size, low cost, and open-source design makes it ideal for research settings. The platform features a differential drive system with two independently controlled wheels. It is powered by an onboard single-board computer, which is usually a Raspberry Pi 4 that runs Ubuntu 22.04 and

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

ROS 2 Humble. Two primary sensors are on the robot: a LIDAR for environmental mapping, along with obstacle detection.

The development of a custom planner plugin for the Nav2 framework allows users to perform assessments of flexible path planning strategies. The plugin adds the nav2_core Planner interface extension to support algorithm selection and configuration through YAML parameter files and dynamic runtime input. The planner uses three main algorithms: Dijkstra's algorithm for finding the best paths through all possible routes with equal costs, and the A* and Theta* algorithms for smart and efficient path planning. The planner operates using occupancy grid data provided by the /map topic, and the resulting path is published to the /plan topic for execution. The modular structure of this design allows researchers to include new algorithms and even modify existing ones with some coding. This eases the segue through the development and testing of the Nav2 framework [28]. Before the custom algorithms for navigation control are deployed on actual hardware, performance in navigation needs to be validated. For this purpose, RViz, a 3D visualization tool, is used to provide a real-time overview of the robot's current state, map data, sensor readings, and planned trajectories. The system allows researchers to understand how user interactions are managed and assess the degree of seamlessness and proficiency with which user interactions are processed. Gazebo is a physics engine that simulates real-life scenarios with real-life mechanical dealings like friction, inertia, and sensor noise. The TurtleBot3 Gazebo simulation environment allows users to validate the heuristics of the navigation algorithm via several experimental settings, like navigating a corridor, navigating a constantly changing path, or navigating a path full of obstacles. The different components of the system worked together to reduce the time taken to develop, debug, and validate navigation stacks and, with the ability to run the same experiment multiple times in the same environment, allowed researchers to gain repeatable results in controlled experiments [29].

Good perception and the ability to know one's location enhance reliable autonomy. The system utilizes multiple ROS 2 topic subscribers that stream different sensor data, such as /odom, which provides odometry data from the wheel encoders, and /scan, which offers 2D LIDAR data used for obstacle detection and map matching. The data from these topics is also used by the Adaptive Monte Carlo Localization (AMCL) node to determine the position of the robot in the provided map. To achieve this, AMCL uses one of the simplest methods of Monte Carlo localization, the use of a particle filter, where a set of particles, with different weights, is used to represent different estimates of the robot's position. New sensor data is incorporated as the robot progresses, and particle weights are modified using the functions through probability. Resampling ensures convergence to the most likely pose. This method of localization shows excellent performance in the presence of sensor noise and partial occlusions and continues to track the motion without losing the localization accuracy [30-31].

The core packages include:

- nav2_bringup: This is the launch package, which starts the Nav2 system. It does not do navigation but assists in starting the various essential elements.

Command: ros2 launch nav2_bringup bringup_launch.py use_sim_time:=True map:=/path/to/map.yaml

- nav2mapserver: This command allows access to the occupancy grid map representing the robot's surroundings. The occupancy grid map is a 2D map used for mapping and localization.

Command: ros2 run nav2_map_server map_server --ros-args -p yaml_filename:=/path/to/map.yaml

- nav2_planner: Provides the robot position to the targeted final destination and offers a global route using a chosen path planning algorithm.

Command: ros2 run nav2_planner planner_server

- nav2controller: Issues the robot's motion and speed commands based on the global path obtained from the command nav2planner.

Command: ros2 run nav2_controller controller_server

- nav2btnavigator: This is the decision-making layer that orchestrates all the navigation behaviors through a behavior tree.

Command: ros2 run nav2_bt_navigator bt_navigator

For its compact structure, straightforward differential-drive kinematics, and popularity in academia and industry, the TurtleBot3 Burger model was chosen as the robot platform. It is also open-source and budget-friendly and integrates into the ROS 2 ecosystem, which makes it suitable for testing navigation along with path-planning algorithms. The robot can be efficiently constructed, and modular hardware can be precisely manufactured to allow for the simulation of movement dynamics and sensor feedback in the laboratory. Using the turtlebot3_gazebo package, the simulation of the TurtleBot3 Burger was completed, which provided an accurate and detailed virtual representation of the robot. The package contains appropriate sensor models, plugins, and features for LIDAR, odometry, and IMU integration, along with fundamental physics simulation. The robot's behavior in motion was simulated, and the acceleration, turning radius, and wheel slip of the robot were measured and logged so that the algorithms in question could be evaluated, and the simulation setup was used for results that were attainable in the real world. The algorithms were embedded in the same hardware and used the same software to ensure that the results obtained in simulation were real and not a fabrication of the scenario. A 2D occupancy grid (OG) map was generated using the SLAM Toolbox provided by ROS 2. The toolbox performs SLAM by using the LIDAR scans of the environment to classify

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

areas of the environment into free, occupied, and unknown areas. This toolbox had to be used to provide a static view of the simple environment the robot had to navigate. The robot had the test space as it generated an exact spatial representation that corresponded to its physical surroundings. The output generated two files, which showed a ".pgm" image of occupancy probabilities and a ".yaml" configuration file that stored information about map resolution, origin, and threshold values. This map was later used as a static environment reference for all navigation experiments. The map was loaded into the navigation system through the map_server node, which publishes the occupancy grid on the /map topic as a nav_msgs/OccupancyGrid message. This allowed all other nodes in the navigation stack, such as the planner and localization modules, to reference the same environmental model. The pre-generated map provided a controlled environment for testing path planning algorithms because it maintained consistent conditions throughout all trials.

The navigation process relies on costmap configuration because it determines the spatial navigation costs that exist in the environment. The Nav2 stack utilizes the costmap_2d package to develop both global and local costmaps, which perform separate tasks. The global costmap considers all obstacles in the environment when planning a route, but the local costmap only takes a smaller, robot-centric area into consideration and focuses on avoiding and controlling the robot in real time. The costmap parameters must be realistic and safe for navigation, and they should accurately profile robot footprints and inflate obstacles based on pressure. Modifications to the inflation radius attempted to develop a safe gap between the robot and the environment to prevent collisions while optimizing paths. The system incorporated range and ray tracing parameters to allow for rapid obstacle detection and smooth system operation. The robot's footprint in the system was a polygon that accurately represented the TurtleBot3 Burger's geometry. The system setting allowed for accurate detection of all obstacles for the entire navigation task. The adjustment of these parameters became essential for the system to achieve collision-free motion, as it dictated the working relationship these blocks of functionality needed to have with the environmental boundaries, considering the goals of the collision planner and controller nodes [30-31]. To improve the ease of use and allow for repeatable results, a set of custom launch and configuration files has to be developed. The launch files serve to start all the necessary nodes to allow navigation through the robot model in Gazebo, map_server for map loading, the AMCL node for localization, the Nav2 stack for planning and control, and RViz for visualization. Automating the startup sequence minimizes human error, saves time on setup, and ensures that all nodes are launched on the same configuration template. The system used launch files along with ".yaml" parameter files to set system defaults for localization and cost maps and planning algorithms. The system had a feature that allowed the user to choose between three path planning algorithms: Dijkstra, A*, and Theta*. The Nav2 parameter file modifications allowed the system to choose which algorithm to use at runtime by adjusting the planner plugin without any code changes. The system required fine-tunable parameters to perform tests that would compare different planners under specific settings. The proposed path-planning algorithm was to be tested for scalability and practical performance within a path-planning algorithm in a large-scale simulated environment. The environment was a grid map of 100x100 cells or larger filled with a random, sparse distribution of small obstacles to simulate realistic navigation challenges [32]. The setup was done in a way that allowed the evaluators to achieve various objectives in the same configuration. The setup of the algorithm reviews its ability to balance obstacle avoidance with spatial complexity, scalability, and overall search efficiency with respect to extensive search areas. Additionally, the large search area enables smooth navigation and the ability to analyze the algorithm's capacity to circumvent several obstacles, avoiding collisions. As with other large-scale testing, overhead is exposed and monitored as the map size and obstacle density increase [33-34]. Evaluating the overall algorithm performance has been classified into three categories: optimal path, efficiency, and quality.

4. Implementation

Implementation of the pathfinding algorithm comparison system as a unitary ROS2 package, which is to be called all_algo, that will interface with the Nav2 navigation framework and conduct Dijkstra's, A*, and Theta* evaluations. Work and research on the implementation architecture, focusing on the modularity, extensibility, and strict metrics that will allow reliable experimental results. The foundational implementation is made up of four principal Python modules, i.e., the implementation of each of the algorithms *(dijkstra.py, astar.py, and theta_star.py);* a unified interface for handling the costmap, costmaphandler.py; and a benchmark orchestrator for path planning and for path-planning algorithms, pathplannercomparison. The foundational implementation includes a visualizer for real-time visualization of the algorithms, named visualizer.py. This modularization allows for the testing of each algorithm in isolation while maintaining unified workflows for their integration and subsequent comparisons. All algorithms use and share the same costmap. These costmaps are derived from Nav2 costmap_2d, enabling the performance articulations of the algorithms to be reflective of their architecture and not their interpretations of the environment. A significant attribute of the implementation is the full-spectrum entry and logging of performance metrics in real time and in portable JSON format to facilitate statistical analysis after experiments have been conducted. This feature is to encourage and facilitate statistical analytic evaluation after analytics. The CostmapHandler class implements facilities for cost map management, including transformations, cost map details that are Nav2-specific, and providing consistent interfaces to all pathfinding algorithms. This layer lets one compare

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

algorithms fairly without overlaps in how data is retrieved from the environment. It implements a transformation of coordinates from Nav2's continuous world coordinates to discrete grid indices, the expected interface of most pathfinding algorithms. This transformation accounts for map origin offset, resolution scaling, and coordinate system conventions to ensure an accurate fit. Procedures for classifying Nav2 costmap values as binary (free or occupied), required by pathfinding algorithms, are implemented for obstacle detection. One such implementation that is provided does this (with the default threshold: cost > 50 so as to hinder unknown areas). All grid-based algorithms, i.e., Dijkstra and A*, inherit a common search component for managing shared data structures and utilities of a search. This implementation includes a priority queue, a neighbor generator, and a path retracer to ensure consistent behavior across the algorithms, as they strive for efficient routing while reducing unnecessary code repetitions. The algorithm for path reconstruction employs a method tracing the parent via pointers from the destination to the origin. The algorithms do path smoothing and convert coordinates to make end products that Nav2 can understand and utilize for routing and visualization. Applied Dijkstra's demonstrates the implementation of the classical shortest-path algorithm, customized for pathfinding on a grid and optimized for that grid. With a cumulative cost of the path maintained, each node is added (to its cost), and the node is then incorporated into the other nodes in the unexplored priority queue, always in a fit-to-scan path order. And thus, that node path is then stored to only ever be scanned once. Numerous optimizations increase accuracy and computational efficiency and thus overall enhance the implementation of Dijkstra, such as terminating early once the goal is in sight, devising less computational structures, and finding neighbors smarter. The core of the implementation is the priority queue, which uses a section of the heap from a Python library, allowing for efficient minimum extractions and guaranteeing that insertions and deletions operate with log(N) complexity. The algorithm, while in the priority queue, guarantees handling duplicate entries by checking if the node has been scanned and thus exported from the queue, preventing unnecessary re-expansion of the node. Using minimal data structures while searching the grid, such as tuples and sets to store the grid coordinates, as they are immutable and hashable, also provides some memory optimization. The set containing visited nodes prevents the re-expansion of nodes that have already been processed, while the dictionary containing distances allows for efficient updates and retrievals of data related to the shortest paths. The A* implementation below employs Euclidean distance, which, as a heuristic, is both admissible and consistent for all 8 directions on the grid. The heuristic can provide such estimates and offer guidance on the optimally correct path toward the goal. The heuristic of Euclidean distance satisfies the admissibility requirements because it never overestimates the true cost to the goal. In 8-directional grids, the shortest possible path between any two points involves diagonal movement with a cost of $\sqrt{2}$ per step or cardinal movement for 1 per step, both of which exceed or equal the Euclidean distance estimate. Consistency, or monotonicity, follows from the triangle inequality property of Euclidean distance, i.e., heuristic estimates decrease appropriately along optimal paths. This property indeed avoids reopening closed nodes; it simplifies the implementation while guaranteeing the optimality of the results. The Theta* algorithm extends A* with line-of-sight optimization, enabling any-angle path generation where the parent connections can skip intermediate grid cells if the paths are directly unobstructed. Obstacle checking is carried out along possible line-of-sight connections using Bresenham's line algorithm. Of the potential optimizations to Theta*, the use of parent pointers is the most well-known. With each successor node, two parent assignments are possible: the conventional grid-based parent (current node) and the line-of-sight parent (current node's parent). This optimization allows the former to skip through nodes in the path when used as a boundary and is able to develop shortcuts, erasing the need to use other intermediate waypoints. The PathPlannerComparison class handles the comparison of multiple algorithms automatically. This covers the reception of goals, the running of algorithms, the gauging of performance, and the dissemination of the outcomes. This structure guarantees an even comparison of the executed algorithms against the same problem instances while capturing performance metrics. The detailed performance evaluation analyzes the metrics of computational efficiency, path quality, and algorithmic behavior for every single run of an algorithm. This evaluation is done in an open manner to reduce the impact on performance data collection while executing the algorithms. The framework supports the live visualization of the output of the algorithms in RViz, which can be used to evaluate the quality of the path taken and the behavior of the algorithm. It sends path messages for RViz that are of the Path type and uses different colors for each algorithm to ease comparison. The implementation allows for a comprehensive comparison of algorithms and facilitates performance analysis while adhering to the interoperability and frameworks of robotics and ROS2 navigation. The modularity means that other algorithms can be easily added to the suite for other experiments while balancing the algorithms under com parative evaluation through the same design and implementation principles and the same components of the system.

5. Results and Discussion

To explore robot navigation, experiments were conducted in an 8 m x 8 m simulated environment consisting of 0.05 m x 0.05 m grid cells. This developed an environment consisting of 11,536 grid cells containing rectangular navigation obstacles. The robot had to navigate a 3.7 m long diagonal from the lower-left block to the upper-right obstacle in the map. The robot's simulated navigation experiences were conducted in ROS2, Gazebo, and RViz, which enable the management of robot simulations and the

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

generation of visualizations. The figure below shows the experimental result in RViz.

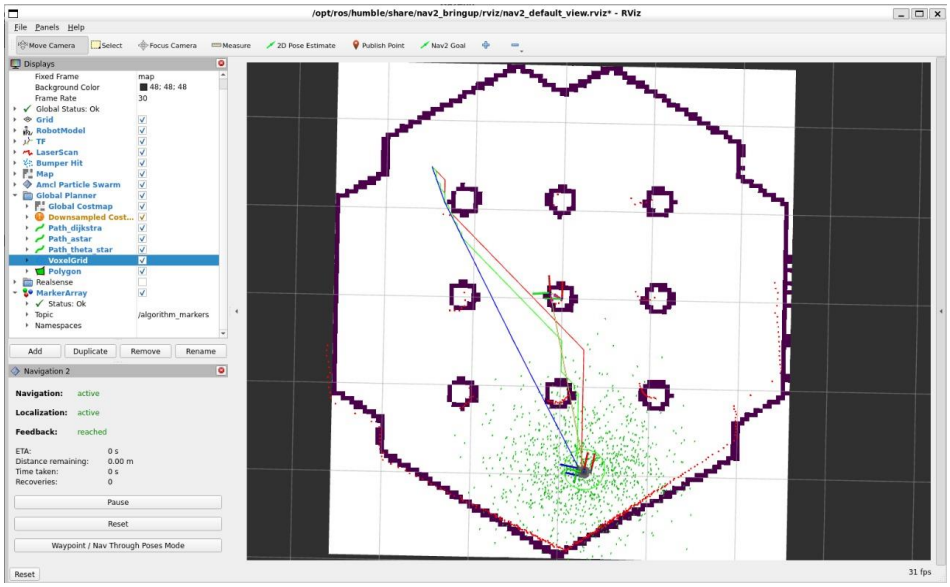

Figure 5: Experimental result in Rviz *(Source: Author's own work)*.

From the experiments, notable trends in the navigation performance of the applied algorithms were documented. The shortest-path algorithms, Dijkstra's and A*, developed paths that were 4.17 m in length. Each of these paths contained 70 waypoints, aligning them with the grid cells of the environment. On the other hand, an approximately 3.87 m long path with only four waypoints was developed by the other applied algorithm, Theta*, which uses any-angle planning. The shorter Theta* path likely reflects a trajectory that is more efficient and easier for a real robot to navigate, as the path would need fewer directional changes. Further review of the computation times illustrates even more of the practical benefits of the use of heuristics within planning systems. Dijkstra's algorithm is not goal-directed; hence, it took 302.56 ms to compute the path. A* demonstrates the usefulness of heuristics in planning systems with a further improvement of 70.25 ms. Theta* with 69.25 ms does a bit more processing due to some line-of-sight checks, though the difference is negligible. There is a clear tradeoff in this case; more planning time will yield a path that is smoother and shorter. The details of the various algorithms in use also show significant differences. While Dijkstra's algorithm searched 7,744 distinct nodes in the grid, A* with its heuristic searched only 1,404. Remarkably, Theta* was efficient with its search; it expanded only 188 nodes. The more advanced planning techniques show their benefits as the complexity of the environment increases. The distinction of A* and Theta* over Dijkstra's algorithm also exhibited a similar pattern as the size of the workspace increased. The environment's complexity increased, the more Dijkstra's algorithm benefited from the decrease in search time made possible by heuristics and the more sophisticated planning techniques; this speaks of the increasingly reliable use of all the algorithms in tactical, real-world robotics. The table below shows the parameter comparison of algorithms.

Table 1: Parameter comparison of algorithms.

Algorithm	Path Length (m)	Computation Time (ms)	Waypoints	Nodes Explored
Dijkstra	4.17	302.56	70	7744
A*	4.17	70.25	70	1404
Theta*	3.87	69.25	4	188

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

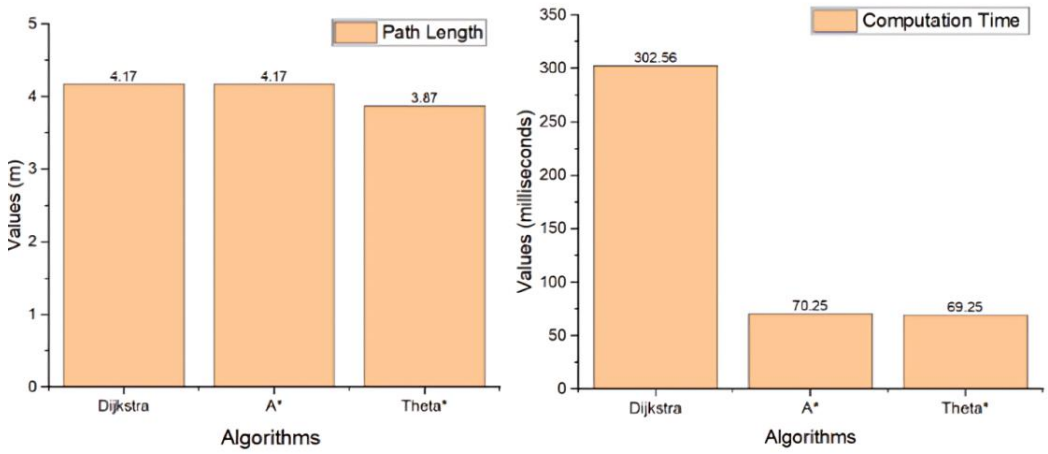

Figure 6: Graphs comparing the path length and computation time by different algorithms *(Source: Author's own work)*.

Every algorithm has specific trade-offs for specific types of problems. For example, Dijkstra's algorithm is better for theoretical benchmarks or small-scale problems because of its predictable simplicity. The A* algorithm is a better fit for real-time applications like service or warehouse robots in structured environments due to its favorable trade-offs between optimality and speed. In situations where a smooth and optimal path is a necessity (e.g., search and rescue or autonomous driving in open and cluttered environments), the Theta* algorithm is better than A*. The combination of ROS2, Gazebo, and RViz provides a seamless integration for testing and improves the time to test a given hypothesis. This method allows for the study of a scenario with real-world physics, provides visual feedback in real time, allows the changing of algorithms mid-simulation, and offers the ability to study trade-offs in a given algorithm while controlling for other variables to produce repeatable results.

6. Conclusions

This paper has performed a thorough comparative study of Dijkstra, A*, and Theta* in the context of modern robotics. The study concluded A* has a positive trade-off in time complexity for planning compared to Dijkstra. Additionally, Theta* offers more optimal paths and higher route smoothness with only a marginal increase in compute time. The study gains confidence from the implementation's open-source nature, thorough benchmarking, and built-in visualization. While significant efforts to provide a real-world test of dynamic 2D environments were limited to a single robot and did not tackle the challenges of kinematic constraints, dynamic obstacles, or multi-robot deployments. Future work must deploy these methods into 3D environments and tackle the real-time challenges of integrating machine learning for heuristic modifications and an adaptive framework, alongside other real-time challenges such as dynamic obstacles, kinematic feasibility, and multi-objective optimization. A vital next step is the deployment of these methods in real-world scenarios after extensive testing in simulation environments. A well-defined foundation has been developed for more research to be accomplished in this area, considering the experimental setup and open-source software strategies developed in this research to aid in providing autonomous robots with dependable and efficient navigation. This research consolidates and improves the currently available evidence. It demonstrates the different available classical and modern algorithms and the software engines pertaining to autonomous robot path planning.

References

[1] Singholi, A.K.S., Mittal, M., Bhargava, A. (2021). A Review on IoT-Based Hybrid Navigation System for Mid-sized Autonomous Vehicles. In: Pandey, V.C., Pandey, P.M., Garg, S.K. (eds) Advances in Electromechanical Technologies. Lecture Notes in Mechanical Engineering. Springer, Singapore. https://doi.org/10.1007/978-981-15-5463-6_65.

[2] Fragapane, G., de Koster, R.B.M., Sgarbossa, F., Strandhagen, K.O.: Planning and control of autonomous mobile robots for intralogistics: Literature review and research agenda. European Journal of Operational Research 294(2), 405–426 (2021).

[3] Cebollada, S., Payá, L., Flores, M., Peidró, A., Reinoso, O.: A state-of-the-art review on mobile robotics tasks using artificial intelligence and visual data. Expert Systems with Applications 167, 114195 (2021).

[4] Miao, Q., Wei, G.: A comprehensive review of path-planning algorithms for planetary rover exploration. Remote Sensing 17(11), 1924 (2025).

[5] A. Bhargava and A. Kumar, "Arduino controlled robotic arm," 2017 International conference of Electronics, Communication and Aerospace Technology (ICECA), Coimbatore, India, 2017, pp. 376-380, doi: 10.1109/ICECA.2017.8212837.

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

[6] Bhargava, A., Suhaib, M. & Singholi, A.S. A review of recent advances, techniques, and control algorithms for automated guided vehicle systems. J Braz. Soc. Mech. Sci. Eng. 46, 419 (2024). https://doi.org/10.1007/s40430-024-04896-w.

[7] Bhargava, A., Suhaib, M. & Singholi, A.K.S. Kinematic and dynamic modeling of a mecanum wheeled vehicle with simulation and experimental validation. J Braz. Soc. Mech. Sci. Eng. 48, 570 (2026). https://doi.org/10.1007/s40430-026-06505-4.

[8] Nash, A., Daniel, K., Koenig, S., Feiner, A.B.: Theta*: Any-angle path planning on grids. In: Proceedings of the AAAI Conference on Artificial Intelligence, pp. 1177–1183 (2007).

[9] Pittner, M. et al.: Systematic analysis of global and local planners for optimal trajectory planning. In: International Symposium on Robotics (ISR), Munich (2018).

[10] Quigley, M. et al.: ROS: An open-source robot operating system. In: ICRA Workshop on Open Source Software (2009).

[11] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W.: Robot navigation in the era of ROS 2: Design, architecture, and approaches. IEEE Robotics & Automation Magazine 27(2), 110–124 (2020).

[12] Bhargava A, Suhaib M, Singholi AK (2026;), "Omnidirectional autonomous mobile robot navigation in cluttered environments using a neuro-fuzzy control framework". Industrial Robot, Vol. ahead-of-print No. ahead-of-print. https://doi.org/10.1108/IR-10-2025-0387.

[13] 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 4(2), 100–107 (1968).

[14] Stentz, A.: Optimal and efficient path planning for partially-known environments. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 3310–3317 (1995).

[15] Koenig, S., Likhachev, M.: D* Lite. In: AAAI Conference on Artificial Intelligence, pp. 476–483 (2002).

[16] Bhargava A, Suhaib M, Singholi AKS. A hybrid optimization algorithm combining A* and the dynamic window approach for automated guided vehicle control in static and dynamic environments. Robotica. 2026;44(2):408-470. doi:10.1017/S0263574725102944.

[17] Silva, M., Ventura, R., Lima, P.: Energy-aware navigation using smooth trajectories for mobile robots. Robotics and Autonomous Systems 132, 103593 (2020).

[18] Lu, D.V., Hershberger, D., Smart, W.D.: Layered costmaps for context-sensitive navigation. In: IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 709–715 (2014).

[19] Marder-Eppstein, E. et al.: The office marathon: Robust navigation in an indoor office environment. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 300–307 (2010).

[20] Macenski, S., Foote, T., Gerkey, B., Lalancette, C., Woodall, W.: Robot navigation in the era of ROS 2: Design, architecture, and approaches. IEEE Robotics & Automation Magazine 27(2), 110–124 (2020).

[21] Wang, Y., Lü, P., Sun, Z.: Improved inflation layer modeling for safe robot navigation in dynamic environments. International Journal of Advanced Robotic Systems 18(2) (2021).

[22] A. Bhargava, A. S. Singholi and M. Suhaib, "Design and Development of a Visual - SLAM based Automated Guided Vehicle," 2023 IEEE Fifth International Conference on Advances in Electronics, Computers and Communications (ICAECC), Bengaluru, India, 2023, pp. 1-7, doi: 10.1109/ICAECC59324.2023.10560331.

[23] Dolgov, D., Thrun, S., Montemerlo, M., Diebel, J.: Path planning for autonomous vehicles in unknown semi-structured environments. International Journal of Robotics Research (2010).

[24] Sturtevant, N.R.: Benchmarks for grid-based pathfinding. IEEE Transactions on Computational Intelligence and AI in Games (2012).

[25] Koenig, S., Likhachev, M.: D* Lite. In: AAAI Conference on Artificial Intelligence, pp. 476–483 (2002).

[26] Bhargava A, Suhaib M, Singholi AKS. An omnidirectional mecanum wheel automated guided vehicle control using hybrid modified A* algorithm. Robotica. 2025;43(2):449-498. doi:10.1017/S0263574724001954.

[27] Wei, Z. et al.: ROS-based navigation and obstacle avoidance. PMC - National Institutes of Health (2025).

[28] Yu, J. et al.: Global path planning algorithm for mobile robots: A review. Journal of Networking and Information Technology 9(2), 649–668 (2024).

[29] Ghazal, M.T. et al.: Simulation of autonomous navigation of TurtleBot robot using ROS. Bulletin of Electrical Engineering and Informatics 13(2), 1–10 (2024).

[30] Fu, X. et al.: Research on path planning of mobile robots based on an improved A* algorithm. Frontiers in Robotics and AI 12, 1652251 (2025).

[31] Puente, P. de la et al.: Combining vision and range sensors for AMCL localization in corridor environments with rectangular signs. Frontiers in Robotics and AI 12, 1652251 (2025).

[32] Gatesichapakorn, S. et al.: ROS-based autonomous mobile robot navigation using LIDAR and RGB-D camera. IEEE Transactions on Robotics 35(4), 879–891 (2019).

[33] A. Bhargava, M. Suhaib and A. K. S. Singholi, "Design and Development of an AGV with Multi-Sensor Fusion for Autonomous Navigation," 2025 IEEE DELCON - International Conference on Recent Smart Technologies in Engineering for Sustainable Development, New Delhi, India, 2025, pp. 1-7, doi: 10.1109/DELCON68055.2025.11399958.

[34] A. Bhargava, A. S. Singholi and M. Suhaib, "A study on design and control of Omni-directional mecanum wheels based

International Journal of Engineering Research & Technology (IJERT)
ISSN: 2278-0181
Vol. 15 Issue 08 , August - 2026

AGV system," 2023 14th International Conference on Computing Communication and Networking Technologies (ICCCNT), Delhi, India, 2023, pp. 1-6, doi: 10.1109/ICCCNT56998.2023.10306665.

[35] A. Bhargava, A.K. S. Singholi and D. Chhabra, "Design And Development Of A Machine Learning-Based Gesture-Controlled 3D-Printed Robotic Hand Using Computer Vision," Global E-Journal of Social Scientific Research, Delhi, India, doi: 10.64706/jecmn681.