JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Comparative Performance Analysis of Pathfinding Algorithms for Autonomous Navigation Systems

Sarang Joshi
Department of Electronics and Telecommunications Engineering
AISSMS Institute of Information Technology
Pune, Maharashtra, India
sarangjoshi.business@gmail.com

Jaykumar Girase
Department of Electronics and Telecommunications Engineering
AISSMS Institute of Information Technology
Pune, Maharashtra, India
jaykumargirase62@gmail.com

Anish Bhujbal
Department of Electronics and Telecommunications Engineering
AISSMS Institute of Information Technology
Pune, Maharashtra, India
anishbhujbal88@gmail.com

Prof.M.P.Gajare
Department of Electronics and Telecommunications Engineering
AISSMS Institute of Information Technology
Pune, Maharashtra, India
milind.gajare@aissmsioit.org

Abstract*— Pathfinding is a fundamental challenge in robotics, particularly for autonomous navigation in structured and unstructured environments. This paper presents a comparative analysis of five widely used pathfinding algorithms: Flood Fill, A*, Dijkstra's Algorithm, Greedy Best-First Search (GBFS), and Wall Following, implemented on an ATmega328P-based autonomous robot to navigate mazes of varying complexities. The study evaluates each algorithm based on execution time, memory consumption, and scalability across different maze sizes (7×7, 9×9, 11×11, and 13×13). The performance metrics are analyzed to determine the trade-offs between computation time, optimal path efficiency, and resource utilization. The findings reveal that informed search algorithms such as A* and Dijkstra's Algorithm outperform others in terms of path optimality and reliability, while uninformed search methods like Flood Fill and BFS ensure completeness but at the cost of increased computational overhead. The Wall-Following algorithm, though simplistic, demonstrates guaranteed path discovery in connected mazes but lacks efficiency in minimizing traversal length. The experimental results provide insight into the suitability of these algorithms for real-world applications, emphasizing their applicability in constrained environments such as robotics, artificial intelligence, and emergency response systems.

Keywords: Automation, navigation, Heuristic algorithm, Path planning, Energy efficient

I. INTRODUCTION

Pathfinding is a fundamental problem in robotics and artificial intelligence, playing a critical role in enabling autonomous systems to navigate complex environments efficiently and accurately. The ability of a robot to determine an optimal path from a designated starting position to a goal while avoiding obstacles is essential for a variety of real-world applications, including robotics, artificial intelligence, video game development, network routing, and search-and-rescue operations. Pathfinding algorithms provide the computational framework necessary for autonomous agents to traverse environments that contain branching pathways, dead ends, and obstacles, ensuring effective and reliable navigation. In robotics, the efficiency of navigation is heavily dependent on the algorithm employed for pathfinding. An optimal algorithm must balance computational efficiency, memory consumption, and execution speed while ensuring accurate path discovery. Various classical algorithms have been extensively studied and implemented across multiple domains to facilitate effective pathfinding. These include uninformed search methods such as Breadth-First Search (BFS) and Flood Fill, which explore the search space systematically without heuristic guidance, as well as informed search techniques such as A*, Dijkstra's Algorithm, and Greedy Best-First Search (GBFS), which leverage heuristics to improve search efficiency. Additionally, reactive navigation strategies like Wall Following offer a simple yet effective means of path discovery in constrained environments where heuristic or complete search methods may not be practical. The objective of this study is to conduct a comparative analysis of five widely used pathfinding algorithms—*Flood Fill, A, Dijkstra's Algorithm, Greedy Best-First Search (GBFS), and Wall Following**—in the context of guiding an autonomous robot through a structured maze environment. The maze represents a grid-based environment where each cell is classified as either open (passable) or blocked (impassable), requiring the robot to identify a valid route to reach the goal. By evaluating these algorithms on key performance metrics such as execution time, memory consumption, and scalability, this research aims to provide insights into their relative advantages and limitations, thereby aiding in the selection of the most suitable algorithm for different robotic navigation scenarios. The significance of this study lies in its relevance to both academic research and practical applications. In domains where autonomous robots must navigate intricate and dynamic environments, selecting an appropriate pathfinding strategy can greatly influence operational efficiency. For instance, in disaster response scenarios where robots are deployed for search-and-rescue missions, an algorithm with minimal execution time is preferred. Conversely, in resource-constrained environments such as embedded robotic systems, memory efficiency becomes a critical factor. Understanding the trade-offs between execution speed, computational requirements, and scalability is crucial for optimizing robotic navigation in diverse real-world conditions.

II. LITERATURE REVIEW:

This study proposes enhancements to the traditional A-star algorithm[12]. This paper provides a theoretical probabilistic framework for visual perception, influencing algorithms that rely on environmental understanding[18]. This review traces the evolution of SLAM techniques, for robust navigation[14]. Emphasizing the integration of visual data with pathfinding algorithms for improved environmental perception has been presented [2]. A broad spectrum of autonomous navigation techniques have been reviewed [5]. This comprehensive review discusses localization techniques, critical for accurate pathfinding in autonomous systems [11]. A foundational paper on SLAM, detailing methodologies that support effective pathfinding in mobile robots [17]. Highlights key challenges in social robot navigation, including dynamic obstacle management, human-robot interaction [1]. Focused on maze-solving strategies, this study compares algorithmic approaches to finding efficient paths in constrained environments [19]. Focusing on small unmanned aerial systems, review of intelligent navigation advancements, discussing algorithm adaptations for aerial environments have been discussed [6]. The study presents current vision-based localization techniques, essential for enhancing pathfinding algorithms through precise environmental mapping[7]. This study evaluates navigation technologies in surface vehicles, providing performance benchmarks relevant to pathfinding algorithms [9]. exploration of machine learning approaches in motion planning and control [3]. The paper focuses on algorithms designed to find the shortest path in maze environments, offering comparative insights into algorithmic efficiency and accuracy [10]. The paper compares boundary fill and flood fill algorithms, providing performance metrics applicable to pathfinding scenarios [13]. The study demonstrates EKF-SLAM applications [15]. This paper introduces FastSLAM with SIFT features [16].

III. PROPOSED SYSTEM

System Overview

This study evaluates and compares the performance of the five distinct pathfinding algorithms—Flood Fill, A*, Dijkstra's, Greedy Best-First Search, and Wall Following which will then be applied to a robot which will navigate its way out of that maze. The maze is represented by a 7x7, 9x9, 11x11 and 13x13 grid, where each cell is either an open path (0) or blocked by a wall (1). The robot's task is to determine a viable path from a predefined start position to an end position.

Hardware Configuration:

It consists of three major parts which are the Arduino Uno r3 which is equipped with ATmega328P, DC motors, L298N motor drive and a power supply. Arduino Uno R3 is being used as the primary microcontroller used for executing the algorithms, controlling the robot's movement. The Arduino handles algorithmic logic and motor control based on the calculated path. Motors drive the robot according to the path computed by the algorithm, while a motor driver regulates motor speed and direction. A battery pack provides 7.4v to operate the motor drive, Arduino and the motors.

Software Configuration:

The environment is structured as a grid with fixed dimensions, where the starting and target locations are predefined for simplicity. Algorithms are programmed in Arduino Specific C++ and executed within the Arduino IDE. The following sections detail the implementation and behavior of the pathfinding algorithms used in this study: Dijkstra's, Greedy Best First Search, A*, Flood Fill, and Wall Following.

Dijkstra's algorithm employs a queue-based approach to systematically explore nodes in a structured search process, adhering to a First-In-First-Out (FIFO) strategy.To enable path reconstruction, a separate structure records the predecessor of each node. A validation function ensures that only permissible moves,those within boundaries and along accessible paths are considered.Following a shortest-path approach, the algorithm iterates through potential movement directions, updating distances when a shorter route is found. This process continues until the target location is identified. Once reached, the recorded predecessor information is used to reconstruct the optimal path, which is subsequently outputted.

Greedy Best First Search algorithm progresses by selecting the most promising move at each step.To ensure efficiency, it maintains a record of visited locations, preventing redundant exploration. A validation mechanism ensures that only feasible moves are considered, adhering to boundary constraints and path availability. The search expands directionally, evaluating each possible step based on the heuristic, continuously refining its path selection.If the destination is reached, the sequence of moves leading to the goal is reconstructed by retracing steps. If an impasse is encountered, alternative routes are explored, adjusting the approach dynamically.

A* algorithm uses a priority-based approach to determine the next location to explore, favoring positions that minimize overall travel effort. A function ensures that movement remains within defined boundaries and avoids restricted zones. The search process iterates through potential movements in multiple directions, maintaining both the cumulative travel effort and an estimate of remaining distance. Each explored location retains a reference to its predecessor, enabling the reconstruction of the traversed route upon reaching the objective. Once the destination is identified, the path is traced back to its origin. The method ensures that no location is revisited unnecessarily, and a tracking mechanism prevents redundant exploration.

Flood Fill algorithm, a depth-first search-inspired strategy, explores the environment. Movement occurs in steps, subject to constraints. A validation function ensures movement stays within boundaries, avoids obstacles, and prevents revisiting locations. The traversal path is recorded for backtracking. The search expands recursively until the target is reached or no further progress is possible. Impasses trigger backtracking to explore alternative paths.

Wall following algorithm starts from a defined entry point and aims to reach a predetermined goal while adhering to a predefined rule that ensures continuous movement along a reference boundary. Movement decisions prioritize an initial directional adjustment before proceeding forward, resorting to alternative directions when obstacles are encountered. If no viable path is available, a history mechanism enables backtracking to explore alternate routes. This mechanism maintains a record of previous states, allowing the entity to systematically reverse its course when necessary.

IV. METHODOLOGY

The work presented in this study focuses on a ATmega328P based robot which autonomously navigates a maze which is a complex and branched structure in which the algorithm must navigate from a specific start point to an end point. It is a grid in which each cell in the maze is classified as either open (passable) or blocked (impassable), requiring the robot to determine a valid path through the maze. In the domain of robotics, pathfinding algorithms are crucial for enabling autonomous agents to traverse intricate environments with both efficiency and precision. These algorithms compute the optimal route for the agent, facilitating intelligent decision-making. Pathfinding algorithms are integral to fields such as robotics, artificial intelligence, video game development, network routing, hazardous environment navigation, evacuation in disaster ridden areas and maze-solving robots, where ensuring safe and efficient navigation is important. This paper presents a comparative analysis of five widely used pathfinding algorithms: Flood Fill, A*, Dijkstra's Algorithm, Greedy Best-First Search (GBFS), and Wall Following, focusing on their application to guiding an autonomous robot through a maze. Each algorithm possesses unique characteristics in its exploration strategy, particularly regarding how it balances distance minimization, heuristic guidance, and computational efficiency. Without effective pathfinding algorithms, autonomous

robots risk inefficiency, suboptimal navigation, or unsafe behavior, potentially undermining the utility of these systems in real-world applications. Consequently, understanding the strengths and limitations of various pathfinding approaches is critical for enhancing the performance of autonomous robots across diverse domains and future use. The algorithms chosen for this particular study are all classical algorithms which have proven to be robust and have been implemented in various industrial and information technological applications.

The Breadth-First Search (BFS) algorithm is a graph traversal method that explores all nodes at a given depth level before proceeding to the next. The Flood Fill algorithm is a brute-force pathfinding technique that explores all possible paths from the starting point by expanding outward in a wave-like manner until it reaches the goal. Greedy Best-First Search selects nodes for expansion based solely on their estimated proximity to the goal, determined by a heuristic function h(n); this algorithm may not give optimal solutions in certain cases. The A* algorithm combines the actual traversal cost g(n) and the heuristic cost h(n) to compute a total cost function f(n)=g(n)+h(n), it balances exploration and heuristic guidance, making it a reliable algorithm in case of robotics and gaming. Dijkstra's Algorithm is a specific case of UCS (uniform cost search) that identifies the shortest path from a single source to all nodes in a graph with non-negative weights.

Algorithms such as BFS and Floodfill fall under the category of uninformed search algorithms which operate without prior knowledge of the search space's structure or characteristics, while algorithms; A* Dijkstra's Algorithm, Greedy Best-First Search (GBFS) falls under the category of informed search algorithms which use heuristic functions to estimate the cost from the current node to the goal. Wall-following on the other hand is a simple, reactive approach where the robot continuously follows a wall until it finds the goal, this method is less efficient but guarantees that the robot will find a path as long as one exists. Each algorithm offers a different approach to solving a maze, making them suitable for comparative analysis. Complete algorithms like BFS, A*, and Dijkstra's Algorithm guarantees finding a solution if one exists, even in large or complex graphs. In contrast, incomplete algorithms such as Greedy Best-First Search may fail due to infinite loops or over-reliance on heuristics. Therefore depending on the application, one can determine which algorithm fits best.

Evaluation Metrics:

The performance of the pathfinding algorithms will be evaluated on three parameters: running time (execution time), memory consumption and scalability. These metrics provide a comprehensive understanding of each algorithm's efficiency, resource consumption, and suitability for different maze sizes.

Running Time (Execution Time): This metric measures the total time taken by each algorithm to find a solution, from the start of the algorithm to the identification of the goal or the conclusion that no solution exists. Running time is crucial for assessing the efficiency of an algorithm in real-time applications, where rapid decision-making is necessary. It will be quantified in terms of milliseconds or seconds, depending on the complexity of the maze and the algorithm.

Memory Usage: Memory Usage refers to the amount of memory consumed by the algorithm during execution. This includes the storage required for the maze itself, visited cell tracking, open lists (in algorithms like A* and Dijkstra), and the storage of the final path. Algorithms with high space complexity may encounter performance limitations on memory-constrained platforms, such as microcontrollers like Arduino. Therefore, a detailed analysis of memory consumption will help determine the feasibility of each algorithm for resource-limited environments.

Scalability: Scalability evaluates how well each algorithm performs as the size of the maze increases. This will be tested by running the algorithms on grids of varying dimensions, such as 7x7, 9x9, 11x11, and 13x13. To quantify scalability, the percentage increase in the time taken by the algorithm as the maze size grows will be calculated. Specifically, the scalability percentage will be determined by comparing the execution time for larger grid sizes to that of the smaller grid sizes, as expressed by the formula:

Scalability Percentage = ((Time for larger grid – Time for smaller grid)/Time for smaller grid) *100

This metric will help assess the algorithm's efficiency in handling increasingly large and intricate maze structures. A low scalability percentage indicates that the algorithm can manage larger mazes with minimal performance degradation, which is crucial for its practical application in more complex environments. Additionally, both time and space aspects will be considered to evaluate the overall performance as the problem size grows.

V. SYSTEM ARCHITECTURE AND PROCESS FLOW

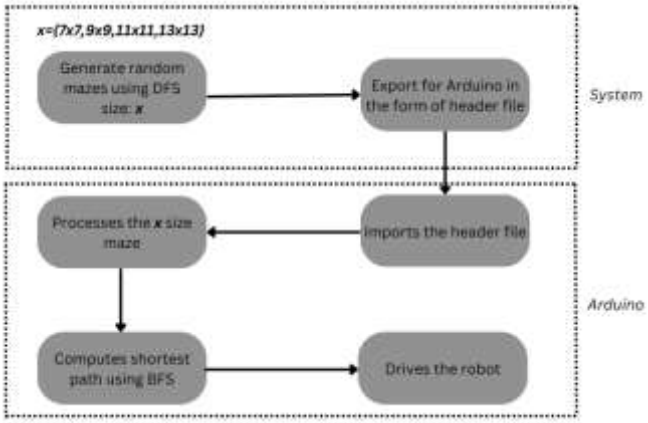

Figure 1 Process Flow

The process flow of the autonomous maze-solving robot is structured into two key components: the System and Arduino, each with distinct yet interconnected responsibilities that collectively enable efficient pathfinding and autonomous navigation. This workflow is designed to generate randomized mazes, process them in a structured manner, compute the shortest path using graph search algorithms, and execute navigation commands to drive the robot through the maze successfully.

The first stage of the process occurs within the System, where mazes of varying dimensions—7×7, 9×9, 11×11, and 13×13—are randomly generated using the Depth-First Search (DFS) algorithm. DFS is a recursive, backtracking-based algorithm widely used for maze generation due to its ability to create a complex, branching structure with a single connected path between any two points in the grid. By employing DFS, the system ensures that the generated mazes provide a diverse range of challenges for the pathfinding algorithms, allowing for a robust comparative analysis across different levels of complexity. The variation in maze sizes further enables an investigation into the scalability of the algorithms, evaluating how their computational efficiency and memory requirements fluctuate as the problem size increases.

Once the mazes are generated, they must be formatted into a structure that the Arduino microcontroller can interpret and process. To achieve this, the system exports the maze data in the form of a header file, a standardized format that encodes the maze structure into a form that can be readily accessed by the Arduino's firmware. This step is crucial as it ensures seamless integration between the computational capabilities of the system and the execution environment of the microcontroller, allowing for real-time path planning and robot movement.

In the next phase, the Arduino takes over the processing of the maze. It begins by importing the header file, which contains the maze structure, ensuring that all the information regarding open pathways and obstacles is accurately transferred. Once the maze data is successfully imported, the pathfinding algorithm is applied to determine the optimal route from the start point to the goal. In this particular implementation, the Breadth-First Search (BFS) algorithm is utilized to compute the shortest path. BFS is a graph traversal algorithm that systematically explores all nodes at a given depth level before proceeding to the next level, ensuring that the shortest path is always identified in an unweighted graph scenario. The use of BFS is particularly advantageous in this application due to its completeness and optimality, meaning that if a solution exists, BFS will always find the shortest possible path in terms of the number of moves.

After BFS computes the shortest path, the robot executes movement commands based on the computed path. The path is translated into a series of movement instructions—such as moving forward, turning left, or turning right—which are then sent to the robot's motor driver for execution. The robot follows the planned trajectory while continuously validating its position within the maze, ensuring accurate navigation from the starting point to the designated goal. This real-world execution of the computed path serves as the final validation step, demonstrating the effectiveness of BFS in guiding the robot through dynamically generated mazes.

This structured process flow highlights the integration of algorithmic decision-making with physical execution in autonomous robotics, emphasizing how computational techniques like DFS for maze generation and BFS for pathfinding are leveraged to enable real-world navigation. The inclusion of varying maze sizes allows for an in-depth evaluation of the performance metrics, including execution time, memory consumption, and scalability, providing insights into the suitability of different algorithms for robotic navigation in constrained environments. By following this approach, the study systematically assesses the feasibility of BFS as a pathfinding solution for autonomous robots while laying the groundwork for future explorations into more advanced, heuristic-driven algorithms.

VI. IMPLEMENTATION

The images presented below illustrate the complete process of maze generation, path computation, and real-time execution by the Arduino-based autonomous robot. The first set of images showcases how the system generates 7×7, 9×9, 11×11, and 13×13 mazes using the Depth-First Search (DFS) algorithm, ensuring a structured yet randomized environment for the pathfinding algorithms to operate.

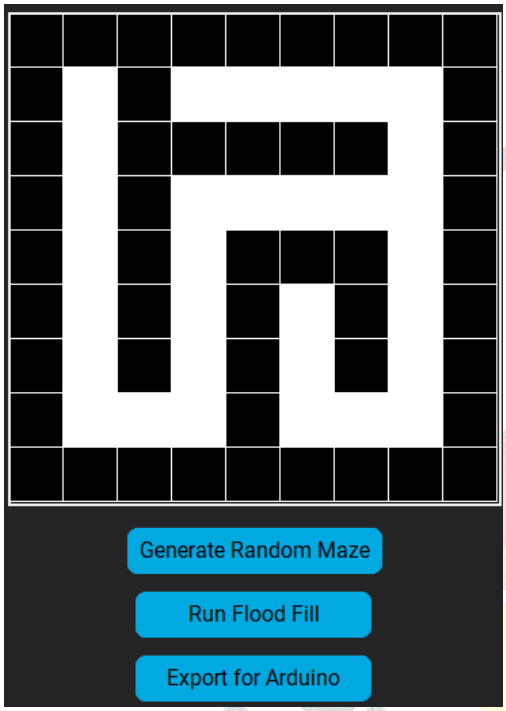

Figure 2 random maze of size 7x7

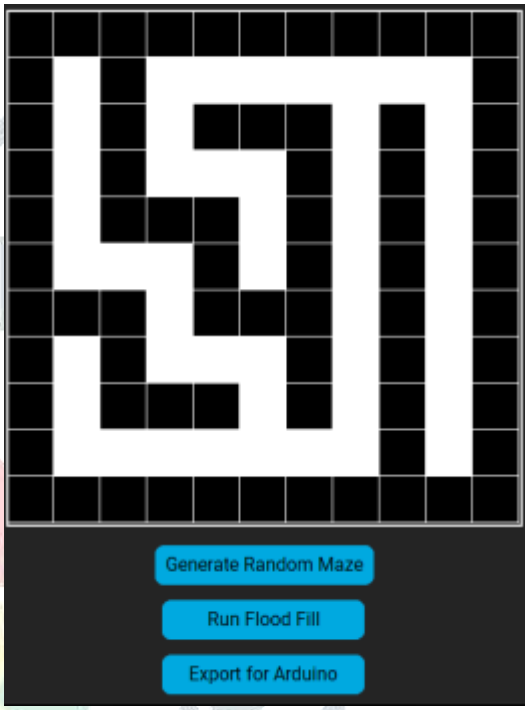

Figure 3 random maze of size 9x9

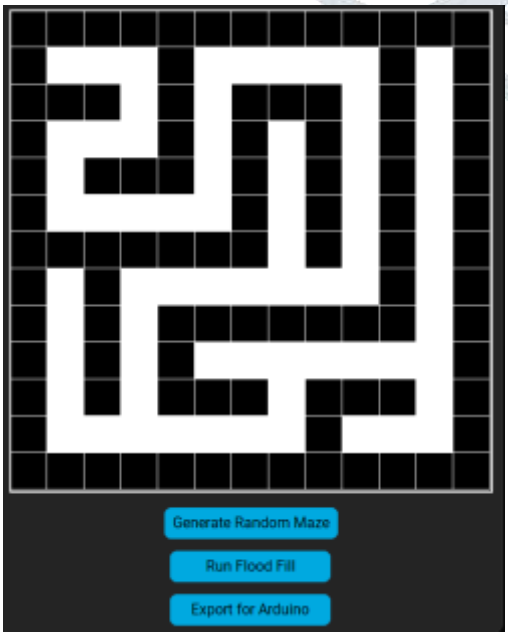

Figure 4 random maze of size 11x11

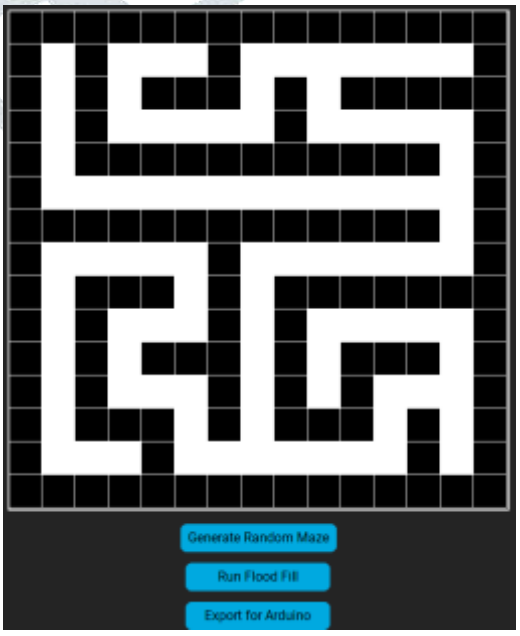

Figure 5 random maze of size 13x13

Additional images provide insights into the real-time execution of the algorithm on the Arduino, displaying the serial monitor output in the Arduino IDE, which logs the robot's decision-making process as it navigates through the maze. The serial monitor outputs key information such as path traversal steps, offering a clear view of how the algorithm is executed in a resource-constrained microcontroller environment. By visualizing the entire process—from maze generation to real-world navigation—these images serve as crucial evidence of the system's effectiveness in solving mazes of increasing complexity, validating the feasibility of BFS for autonomous robotic applications.

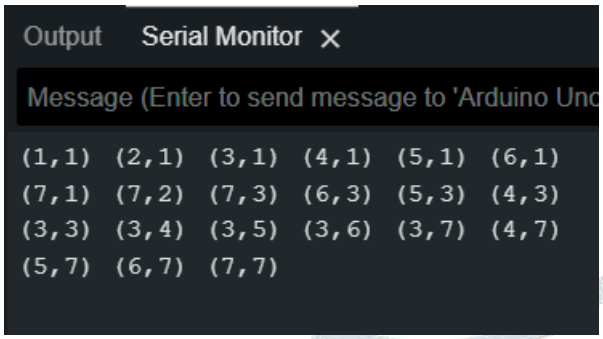

Figure 6 optimum path for 7x7 on-chip

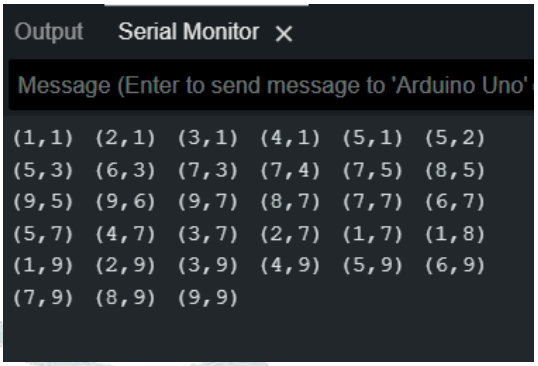

Figure 7 optimum path for 9x9 on-chip

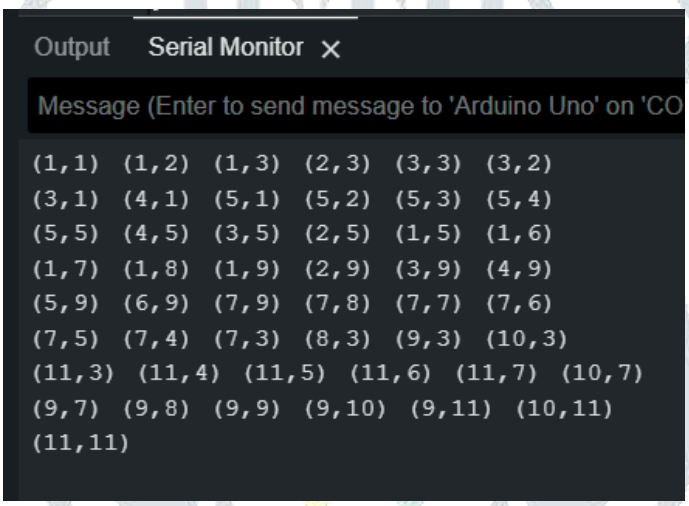

Figure 8 optimum path for 11x11 on-chip

VII. RESULTS

By analysing the metrics in section 4, the study aims to provide a nuanced comparison of each algorithm's performance in terms of both computational efficiency and its suitability for real-world applications in robotics.

Algorithms	Execution Time (ms)	Memory consumed (Bytes)	Scalability wrt time	Scalability wrt memory	Maze size
FloodFill	64	619	0	0	7X7
	97	843	34.02%	26.57%	9X9
	129	1123	24.80%	24.90%	11X11
	375	1459	65.60%	23.02%	13X13
Dijkstra	62	1094	0	0	7X7
	96	1286	35.41%	14.93%	9X9
	129	1766	25.50%	27.18%	11X11
	NA	NA(memory exceeded)	NA	NA	13X13
Greedy best first fit	64	619	0	0	7X7
	97	843	34.02%	26.57%	9X9
	129	1123	24.80%	24.93%	11X11
	NA	NA(memory exceeded)	NA	NA	13X13
Wall Following	10005	749	NA	NA	7X7
	30017	1037	NA	NA	9X9
	16508	1397	NA	NA	11X11
	24513	1829	NA	NA	13X13
*A**	69	929	0	0	7X7
	101	1153	31.68%	19.42%	9X9
	135	1433	25.18%	19.53%	11X11
	380	1769	64.47%	18.99%	13X13

Table 1 Comparative performace analysis

Table compares the performance of five pathfinding algorithms (FloodFill, Dijkstra, Greedy Best-First, Wall Following, and A*) in solving mazes of varying sizes (7x7, 9x9, 11x11, and 13x13). The comparison is done on the following metrics, execution time in ms is the time taken by each algorithm with increasing maze sizes, wall following algorithm requires a lot more time than others as it is a blind and local search algorithm. The memory consumed increases with increasing maze size. Scalability wrt time and memory represents the relative time and memory increment calculated as percentage increase.

In addition to the comparative theoretical and software-based evaluations, the algorithms and system framework discussed in this study were successfully deployed on a physical autonomous robot platform. The robot was programmed with the maze-solving logic and physically tested in a controlled environment using printed mazes of size 7×7. The mazes were structured using a plywood grid over white foam sheets and were carefully mapped to correspond with the encoded maze data fed to the robot. The robot demonstrated real-time decision-making capabilities and accurate traversal using movement instructions derived from the pathfinding algorithm. The microcontroller interpreted the maze structure, computed the path, and executed motion commands precisely to complete the maze.

To validate the implementation, several test runs were conducted using each algorithm across different maze variants. Among the tested methods, Breadth-First Search (BFS) was utilized for the final hardware implementation due to its completeness, robustness in unweighted graphs, and ease of path reconstruction. The robot exhibited successful traversal in all maze configurations, with minimal deviations or errors, highlighting the practical feasibility of the proposed approach. The performance matched the software-based simulation metrics closely in terms of path length and decision time, considering the inherent latency introduced by hardware-level actuation.

To further reinforce the credibility of the physical implementation, snapshots from the execution have been included, showing the robot mid-operation within various maze setups. The photos are captured amid traversal. The robot followed computed paths with high reliability. The visual validation serves as empirical evidence that the algorithms and system pipeline are not only theoretically sound but also practically deployable.

This real-world validation bridges the gap between simulation and physical deployment, demonstrating that the performance metrics observed in the comparative analysis effectively translate into tangible robotic behaviour. The hardware realization reinforces the study's emphasis on selecting appropriate algorithms based on computational constraints and real-world applicability.

Figure 9 traversal of the robot in a 7x7 maze

VIII. CONCLUSION

This study presents a comparative analysis of various pathfinding algorithms for autonomous robot navigation in maze environments, evaluating their execution time, memory consumption, and scalability. The results indicate that heuristic-based algorithms, particularly A* and FloodFill, offer a balanced trade-off between computational efficiency and memory usage, making them suitable for real-time applications. In contrast, Dijkstra and Greedy Best-First Search struggle with memory scalability, limiting their effectiveness for larger mazes. Wall Following, while memory-efficient, exhibits significantly higher execution times due to its lack of heuristic guidance, the scalability analysis revealed that as the difficulty of the maze increases, the execution time grows substantially. These findings emphasize the importance of selecting an algorithm based on application constraints, such as available computational resources and real-time processing needs. Future work could explore hybrid approaches that combine heuristic-based search with memory-efficient strategies to further optimize pathfinding for autonomous navigation systems.

X. FUTURE SCOPE

The research lays a foundation for further advancements in pathfinding optimization and real-time robotic navigation. Implementing reinforcement learning-based heuristics can enhance algorithm adaptability, allowing robots to learn optimal movement strategies dynamically rather than relying solely on pre-defined heuristics. Expanding the study to real-world navigation problems such as urban transportation, warehouse logistics, and disaster rescue operations where environments are dynamic and obstacles may change over time. Further studies can focus on hardware acceleration using FPGA-based implementations or parallel processing on microcontrollers to improve execution speed. Integration with SLAM (Simultaneous Localization and Mapping) techniques to enhance real-world navigation by enabling autonomous robots to map and adapt to unknown environments.

XI. REFERENCES

[1] Christoforos Mavrogiannis, Francesca Baldini, Allan Wang, Dapeng Zhao, Pete Trautman, Aaron Steinfeld, and Jean Oh. 2023. Core Challenges of Social Robot Navigation: A Survey. J. Hum.-Robot Interact. 12, 3, Article 36 (September 2023), 39 pages. https://doi.org/10.1145/3583741

[2] Yuri D. V. Yasuda, Luiz Eduardo G. Martins, and Fabio A. M. Cappabianco. 2020. Autonomous Visual Navigation for Mobile Robots: A Systematic Literature Review. ACM Comput. Surv. 53, 1, Article 13 (January 2021), 34 pages. https://doi.org/10.1145/3368961

[3] Xiao, X., Liu, B., Warnell, G. et al. Motion planning and control for mobile robot navigation using machine learning: a survey. Auton Robot 46, 569–597 (2022). https://doi.org/10.1007/s10514-022-10039-8

[4] Crespo J, Castillo JC, Mozos OM, Barber R. Semantic Information for Robot Navigation: A Survey. Applied Sciences. 2020; 10(2):497. https://doi.org/10.3390/app10020497

[5] Saeid Nahavandi, Roohallah Alizadehsani, Darius Nahavandi, Shady Mohamed, Navid Mohajer, Mohammad Rokonuzzaman, Ibrahim Hossain. A Comprehensive Review on Autonomous Navigation. 2022 https://doi.org/10.48550/arXiv.2212.12808

[6] Suraj Bijjahalli, Roberto Sabatini, Alessandro Gardi, Advances in intelligent and autonomous navigation systems for small UAS, Progress in Aerospace Sciences, Volume 115, 2020, 100617, ISSN 0376-0421, https://doi.org/10.1016/j.paerosci.2020.100617.

[7] Y. Alkendi, L. Seneviratne and Y. Zweiri, "State of the Art in Vision-Based Localization Techniques for Autonomous Navigation Systems," in IEEE Access, vol. 9, pp. 76847-76874, 2021, doi: 10.1109/ACCESS.2021.3082778.

[8] Leonard Uhr, Charles Vossler; Recognition of Speech by a Computer Program that was Written to Simulate a Model for Human Visual Pattern Perception. J. Acoust. Soc. Am. 1 October 1961; 33 (10): 1426. https://doi.org/10.1121/1.1908460

[9] Choi, J., Park, J., Jung, J. et al. Development of an Autonomous Surface Vehicle and Performance Evaluation of Autonomous Navigation Technologies. Int. J. Control Autom. Syst. 18, 535–545 (2020). https://doi.org/10.1007/s12555-019-0686-0

[10] K. -C. Chang et al., "Shortest Distance Maze Solving Robot," 2020 IEEE International Conference on Artificial Intelligence and Information Systems (ICAIIS), Dalian, China, 2020, pp. 283-286, doi: 10.1109/ICAIIS49377.2020.9194913.

[11] Rathin Chandra Shit, Precise localization for achieving next-generation autonomous navigation: State-of-the-art, taxonomy and future prospects, Computer Communications, Volume 160, 2020, Pages 351-374, ISSN 0140-3664, https://doi.org/10.1016/j.comcom.2020.06.007.

[12] C. Ju, Q. Luo and X. Yan, "Path Planning Using an Improved A-star Algorithm," 2020 11th International Conference on Prognostics and System Health Management (PHM-2020 Jinan), Jinan, China, 2020, pp. 23-26, doi: 10.1109/PHM-Jinan48558.2020.00012.

[13] Bhawnesh Kumar, Umesh Kumar Tiwari, Santosh Kumar, Vikas Tomer, Jasmeet Kalra. International Journal of Innovative Technology and Exploring Engineering (IJITEE) ISSN: 2278-3075, Volume-8 Issue-12S3, October 2019. DOI:10.35940/ijitee.L1002.10812S319

[14] C. Cadena et al., "Past, Present, and Future of Simultaneous Localization and Mapping: Toward the Robust-Perception Age," in IEEE Transactions on Robotics, vol. 32, no. 6, pp. 1309-1332, Dec. 2016, doi: 10.1109/TRO.2016.2624754.

[15] Ahn, S., Choi, J., Doh, N.L. et al. A practical approach for EKF-SLAM in an indoor environment: fusing ultrasonic sensors and stereo camera. Auton Robot 24, 315–335 (2008). https://doi.org/10.1007/s10514-007-9083-2

[16] T. D. Barfoot, "Online visual motion estimation using FastSLAM with SIFT features," 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems, Edmonton, AB, Canada, 2005, pp. 579-585, doi: 10.1109/IROS.2005.1545444.

[17] Chatterjee, A., Rakshit, A., Singh, N.N. (2013). Simultaneous Localization and Mapping (SLAM) in Mobile Robots. In: Vision Based Autonomous Robot Navigation. Studies in Computational Intelligence, vol 455. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-33965-3_7

[18] O. Ciftcioglu, M. S. Bittermann and I. S. Sariyildiz, "Towards Computer-Based Perception by Modeling Visual Perception: A Probabilistic Theory," 2006 IEEE International Conference on Systems, Man and Cybernetics, Taipei, Taiwan, 2006, pp. 5152-5159, doi: 10.1109/ICSMC.2006.385126.

[19] R. Covaci, G. Harja and I. Nascu, "Autonomous Maze Solving Robot," 2020 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR), Cluj-Napoca, Romania, 2020, pp. 1-4, doi: 10.1109/AQTR49680.2020.9129943.