Path Planning on Jetson Platform for a Differential Robot

Osiel Gastélum Zazueta, William Arad Chávez Borrego, Camila Magdaleno López, Ulises Orozco-Rosas, Kenia Picos*

CETYS Universidad, Av. CETYS Universidad No. 4. El Lago, C.P. 22210, Tijuana B.C., México
[gastelum.osiel, william.chavez, camilal]@cetys.edu.mx, [ulises.orozco, kenia.picos]@cetys.mx

Abstract

This work presents the development and implementation of an autonomous navigation system for the TurtleBot 2 robot, leveraging the ROS (Robot Operating System) framework. The system integrates a discrete path-planning algorithm to achieve an efficient and adaptive trajectory. The proposed approach utilizes the A* algorithm for path planning. The TurtleBot 2 is a widely recognized mobile robot platform designed for research and education in robotics. At the same time, the Jetson TX2 is a compact, high-performance computing board that facilitates real-time processing and AI integration. The results validate the proposed system's performance, highlighting the robot's ability to follow predefined paths, maintain accurate positioning, and reach goals efficiently. This work demonstrates the feasibility of integrating discrete path planning techniques with robotics frameworks, providing valuable insights into autonomous navigation challenges and solutions in robotics research and development.

Key Words— Autonomous Navigation, Robot Operating System, Jetson TX2, Path Planning, Mobile Robots

Resumen

Este trabajo presenta el desarrollo e implementación de un sistema de navegación autónoma para el robot TurtleBot 2, aprovechando el marco de trabajo ROS (Robot Operating System). El sistema integra un algoritmo de planificación discreta para lograr una trayectoria eficiente y adaptable. El enfoque propuesto utiliza el algoritmo A para la planificación de trayectoria. El TurtleBot 2 es una plataforma de robot móvil ampliamente reconocida, diseñada tanto para investigación como para educación en robótica, mientras que la Jetson TX2 es una tarjeta de desarrollo compacta y de alto rendimiento que facilita el procesamiento en tiempo real e integración de inteligencia artificial. Al combinar estas tecnologías, el sistema garantiza una navegación confiable. Los resultados validan el rendimiento del sistema propuesto, destacando la capacidad del robot para seguir rutas predefinidas, mantener una posición precisa y alcanzar la meta de forma eficiente. Este trabajo demuestra la viabilidad de integrar técnicas de planificación discreta para robótica, proporcionando información valiosa sobre los desafíos y soluciones en la navegación autónoma en investigación y desarrollo en robótica.*

Palabras Clave— Navegación Autónoma, Robot Operating System, Jetson TX2, Planificación de Rutas, Robótica Móvil

1. INTRODUCTION

Navigating autonomously is critical in mobile robotics, with applications ranging from industrial automation to exploratory missions [1]. This research focuses on developing a robust control system for TurtleBot 2, to enhance trajectory tracking. These systems empower robots to make decisions independently by leveraging advanced sensing, data-driven computing, and control frameworks [2].

Despite significant progress, achieving full autonomy remains a challenge due to dynamic environmental factors, such as obstacles, changing weather, and noisy sensor inputs [3]. Current approaches often rely on modular frameworks (software frameworks include support programs, compilers, code libraries, tool sets, and application programming interfaces that bring together the different components to enable the development of a project or solution) encompassing perception, mapping, planning, and decision-making.

While effective for structured environments, these systems lack adaptability to the unstructured and unpredictable nature of real-world conditions [4]. Enhancing localization and environmental understanding is critical for achieving higher levels of autonomy. This can be addressed by combining robust sensing methods, machine learning algorithms, and real-time processing frameworks.

This work builds upon these principles by implementing an autonomous navigation system for the TurtleBot 2. By employing discrete path-planning techniques and using the Robot Operating System (ROS), the system integrates data to achieve precise trajectory tracking, such as Von Neumann and Moore neighborhood models, and incorporates sensor feedback. This study demonstrates the feasibility of bridging the gap between simulation and real-world performance. This work aims to contribute to the field of mobile robotics by improving the adaptability and efficiency of navigation in complex environments.

* Corresponding author.

2. THEORETICAL FRAMEWORK

This section presents the core concepts of trajectory planning in robotics, emphasizing discrete state spaces and path-planning algorithms. It explores models for representing system states and actions, as well as search algorithms like A*, which optimize navigation while considering obstacles and environmental constraints.

2.1 Discrete Planning in Robotics

Path planning is a problem that requires finding a continuous path between a given start point and the goal point. For a particular system, subject to a variety of constraints, discrete planning in robotics relies on the concept of state spaces, where every possible configuration of the world is represented as a state, and available actions transform the current state into a new one [5].

This model is fundamental for trajectory planning, especially in environments with obstacles. A precise definition of the state space is critical, as it encompasses all possible system configurations, enabling algorithms to efficiently find viable solutions [6].

Each action executed by the robot results in a transition from its current state x to a new state x', mathematically represented as: $x' = f(x, u)$, where u is the applied action. State spaces must be finite and countable for effective discrete planning [7].

The Von Neumann and Moore neighborhood models are essential in modeling transitions in grid-based systems, see Fig. 1. The Von Neumann model considers four orthogonally adjacent cells, while the Moore model includes all eight surrounding cells, enabling greater freedom of movement. These models are critical for structured robotic navigation and obstacle avoidance.

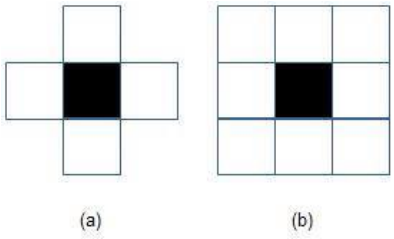

Fig. 1: a) Von Neumann neighborhood (b) Moore neighborhood

2.2 Path Planning with Obstacles

Path planning focuses on creating the shortest path from the source to the destination while considering data from the robot's environment. A critical aspect of this process is obstacle avoidance, which ensures the robot can navigate safely and efficiently in complex environments [8]. Among various approaches, the A* algorithm is a widely used method for finding optimal paths in environments with obstacles.

The A* algorithm operates on a graph-based representation of the environment and systematically evaluates potential routes from the source to the destination. It calculates a cost for each node based on two components: $g(n)$, the actual cost from the starting node to the current node, and $h(n)$, a heuristic estimate of the cost from the current node to the goal. The total cost, represented as $f(n) = g(n) + h(n)$, guides the algorithm to explore the most promising paths first.

The strength of the A* algorithm lies in its ability to combine accuracy with efficiency. By using heuristics tailored to the specific environment, such as Euclidean or Manhattan distance, the A* algorithm minimizes unnecessary exploration and reduces computational overhead. This makes it highly effective for navigating structured and semi-structured environments with static or moderately dynamic obstacles.

Robots utilizing the A* algorithm benefit from precise and adaptable trajectory planning, ensuring they can avoid obstacles and reach their goals optimally. Additionally, the A* algorithm is versatile and extensible, allowing for enhancements such as adaptive heuristics or dynamic replanning, enabling robots to handle more complex and rapidly changing scenarios [9].

The pseudocode shown in Fig. 2 outlines the core steps of the A* algorithm, which systematically evaluates potential paths while ensuring obstacle avoidance and cost optimization.

```
1: Initialize: OPEN list ← {} (empty list)
2: Initialize: CLOSED list ← {} (empty list)
3: Create goal node, call it node_goal
4: Create start node, call it node_start
5: Add node_start to the OPEN list
6: while OPEN list is not empty do
7:     Get node n from the OPEN list with the lowest f(n)
8:     Add n to the CLOSED list
9:     if n is the same as node_goal then
10:        return Solution(n)
11:    else
12:        Generate each successor node n' of n
13:        for each successor node n' of n do
14:            Set parent of n' to n
15:            Set h(n') as the heuristic distance to node_goal
16:            Set g(n') = g(n) + cost(n, n')
17:            Set f(n') = g(n') + h(n')
18:            if n' is in OPEN list and the existing n' is better or equal
   then
19:                continue
20:            end if
21:            if n' is in CLOSED list and the existing n' is better or equal
   then
22:                continue
23:            end if
24:            Remove occurrences of n' from OPEN and CLOSED
25:            Add n' to the OPEN list
26:        end for
27:    end if
28: end while
29: return failure: No solution exists
```

Fig. 2: The A* algorithm

3. PROPOSAL

This study examines autonomous navigation in mobile robots. Using TurtleBot 2, the focus is on path planning and an accurate trajectory tracking in both simulated and real-world environments.

In this work, the navigation is performed in static environments. In dynamic environments to complement the path-planning approach, the sensors enhance the robot's ability to maintain accurate navigation by correcting deviations caused by environmental changes or sensor noise.

In our case, the system is validated through a combination of simulation and physical experiments. In the simulation phase, the Gazebo environment is used to test TurtleBot 2's performance in navigating complex scenarios with varying obstacle configurations. The physical tests will extend these evaluations to real-world conditions, analyzing the system's trajectory accuracy, obstacle avoidance efficiency, and overall reliability.

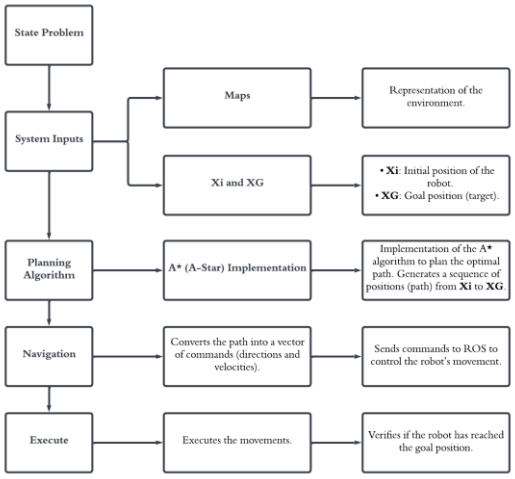

Fig. 3: System block diagram

Fig. 3 shows the proposed system architecture for autonomous navigation using the TurtleBot 2 robot. The process begins with defining the state problem, which establishes the robot's initial position **Xi**, and the target position **XG**. To achieve this goal, the system integrates key inputs such as environmental maps that represent the navigation space [10].

These maps allow the system to understand the surroundings and locate the robot's starting and goal positions within the environment. Once the state and inputs are defined, the focus shifts to the planning algorithm, where the A* algorithm is implemented. This algorithm computes the optimal path by generating a sequence of positions that efficiently lead the robot from the initial position to the goal.

With the optimal path determined, the navigation module translates this path into actionable commands. The positions calculated by the planning algorithm are converted into a series of directions. These commands are sent to the robot's control system through ROS, allowing TurtleBot 2 to follow the planned trajectory effectively.

The final stage is the execution phase, where the robot performs the movements according to the given commands. Throughout this process, the system verifies whether the robot has reached its goal position, ensuring accurate trajectory tracking. This step also accounts for potential deviations caused by environmental changes, allowing the system to correct the robot's path as needed.

Fig. 4 shows the flowchart that outlines the main structure of the code, detailing the sequence of operations for TurtleBot 2's autonomous navigation system. The program begins with the initialization of ROS. It then enters a continuous loop where the A* algorithm processes the map and computes the optimal path. This path is converted into motion commands and sent to the robot for execution.

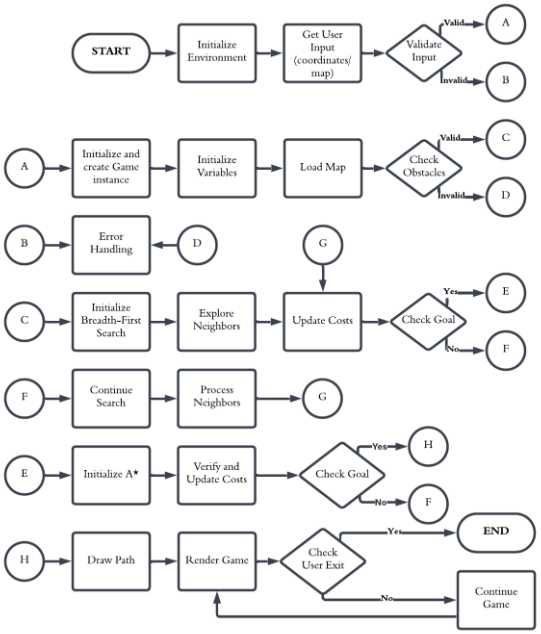

Fig. 4: Main program flow chart

4. TESTING AND VALIDATION

The primary objective of the testing phase was to validate the proposed path-planning solution within a simulated environment. These tests aimed to ensure the efficiency and accuracy of the navigation algorithm while evaluating the system's performance under different conditions.

Conversely, the validation process involved the execution of multiple scenarios within a 10×10 grid environment, incorporating static obstacles to assess the agent's ability to find an optimal path. The tests were conducted using a standardized set of initial and goal positions, with predefined obstacle placements. The experiments were run ten times each one to identify any inconsistencies and ensure repeatability.

During real-world operation, the TurtleBot 2 relies on its onboard sensors to navigate the environment. The primary sensors used include a LiDAR scanner for obstacle detection and wheel encoders for odometry-based position estimation.

However, since the planned paths were precomputed in the simulation, real-time obstacle detection was not required during the execution phase. The robot followed the predefined trajectory without actively re-evaluating its surroundings, assuming a static environment, meaning real-time obstacle detection was not necessary during execution. These maps were loaded into the simulation, and the algorithm selected the appropriate map based on the test scenario being executed, ensuring that all test runs followed the same obstacle distribution and making the results directly comparable. In the physical execution using the TurtleBot 2, the robot followed the precomputed paths without performing real-time obstacle detection, as the environment was assumed to be identical to the simulation

To further analyze the performance of the proposed navigation approach, execution times were recorded for both the simulated algorithm and the physical implementation using the TurtleBot 2 platform. In the simulation, execution time was measured as the total duration from the moment the algorithm started computing the path until it successfully reached the goal position. This included both the computational time required to generate the path and the simulated travel time based on the agent's movement model.

Test	Map 1	Map 2	Map 3	Map 4	Map 5
1	23	15.97	22.95	19.05	12.48
2	21.98	16.46	21.05	21.87	14.61
3	21.18	16.2	22.76	22.94	15.16
4	22.65	14.37	21.3	23.69	17.16
5	23.22	16.09	22.293	20.54	16.83
6	22.65	15.24	22.19	23	16.02
7	20.56	15.84	22.17	23.7	16.21
8	22.8	14.34	22.34	23.17	14.15
9	22.14	16.15	21.71	22.62	17.7
10	19.64	15.69	22.85	22.77	15.43
Avg	21.982	15.635	22.161	22.335	15.575
SdDev	1.1693	0.75	0.639	1.479	1.561

Table 1: Path-planning algorithm execution time (ms)

For the physical execution, the recorded execution time corresponded to the actual duration taken by the TurtleBot 2 to follow the planned path from start to goal, considering its real-world movement constraints. Table 1 presents the execution times of the algorithm in the simulated 10×10 grid environment, highlighting the computational efficiency across different test scenarios. Meanwhile, Table 2 summarizes the execution times observed when executing the planned paths on the TurtleBot 2, providing insights into the real-world feasibility of the solution. These results help assess the algorithm's scalability and practical applicability in dynamic environments.

In Table 1, the metric represents the time required by the algorithm to compute a path within the simulated 10×10 grid environment, considering predefined obstacle placements. The recorded values reflect the computational efficiency of the path-planning process, measuring how quickly the algorithm can generate a feasible trajectory before execution.

Test	Map 1	Map 2	Map 3	Map 4	Map 5
1	32.183	64.883	51.362	42.914	56.646
2	32.183	64.898	51.381	42.912	56.613
3	32.173	64.922	51.392	42.895	56.639
4	32.19	64.896	51.363	42.92	56.585
5	32.156	64.915	51.267	42.911	56.612
6	32.151	64.909	51.328	42.912	56.596
7	32.173	64.896	51.366	42.911	56.643
8	32.186	64.904	51.366	42.899	56.646
9	32.171	64.89	51.335	42.891	56.643
10	32.184	64.885	51.324	42.983	56.612
Avg	32.175	64.9	51.348	42.915	56.624
SdDev	0.0129	0.0127	0.0362	0.0256	0.0226

Table 2: TurtletBot 2's trajectory travel time (s)

In Table 2, the metric represents the time required by the TurtleBot 2 to physically traverse the planned path and reach its goal. The recorded values reflect the overall execution performance in a real-world setting, accounting for factors such as motor response, control accuracy, wheel slip, and environmental conditions

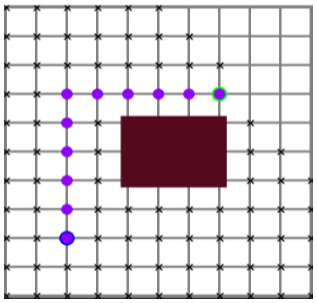

Fig. 5: Map 1

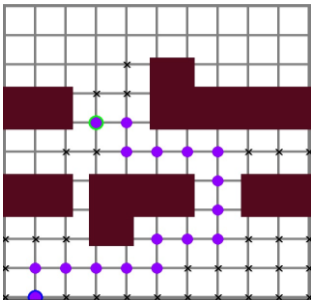

Fig. 6: Map 2

The first test scenario referred to as Map 1 involves navigating the agent from the starting position at coordinates (2,2) to the target at (7,7), see Fig. 5 where the origin is

located at the left-bottom corner (0,0). This map includes a series of static obstacles placed strategically to evaluate the algorithm. The system was evaluated based on the following parameters:

- Execution Time: Measured in milliseconds (ms) to capture minor variations.
- Path Accuracy: Ensuring the agent reaches the target while avoiding obstacles.
- Scalability: Testing the algorithm under varying complexity levels of ability to calculate an efficient and collision-free path.

The second test scenario, Map 2, assesses the agent's ability to navigate from the starting position at coordinates (1,0) to the target at (3,6), see Fig. 6. This map introduces a different obstacle arrangement, designed to challenge the algorithm's adaptability. The evaluation criteria remain consistent, determining the algorithm's efficiency in handling more complex navigation tasks. Consistent performance across multiple runs is crucial as it demonstrates the reliability and stability of the system under varying conditions. It ensures that the navigation algorithm can be trusted to perform effectively in real-world applications, reducing the likelihood of unexpected failures.

The physical implementation of the TurtleBot 2 involved translating the navigation algorithm from the simulation environment into a real-world robot system. One of the critical challenges in the physical environment was ensuring that the robot could maintain precise control over its movement despite external factors like wheel slippage, sensor inaccuracies, and environmental noise. The robot's performance was evaluated by running the algorithm multiple times across different obstacle configurations to ensure consistent results.

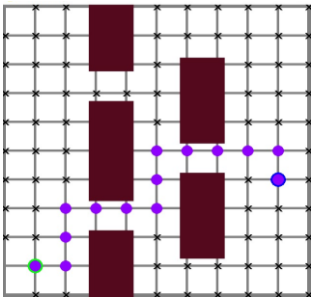

Fig. 7: Map 3

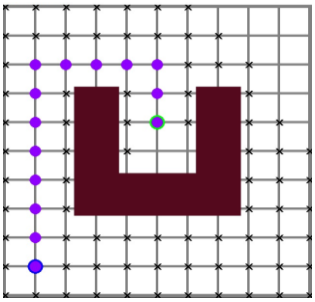

Fig. 8: Map 4

This step was crucial in validating the theoretical models and algorithms developed during the simulation phase, whereas the TurtleBot 2 successfully navigated through a complex environment. Additional testing scenarios were conducted to further validate the algorithm's performance. These scenarios include navigating the agent across different obstacle configurations with the following start and target coordinates: Map 3 from (9,4) to (1,1), see Fig. 7; Map 4 from (1,1) to (5,6), see Fig. 8; and Map 5 from (2,0) to (9,8), see Fig. 9.

The same evaluation criteria were applied to these scenarios, focusing on execution time, path accuracy, and obstacle avoidance. The tests were successful, with the agent consistently reaching the target positions without collisions, demonstrating the reliability and effectiveness of the path planning solution.

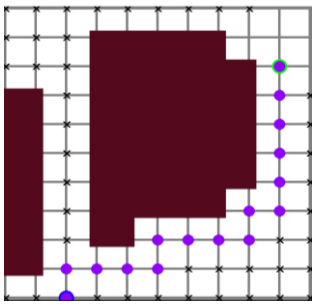

Fig. 9: Map 5

The navigation process on the physical TurtleBot 2 was built on the same core principles used in the simulation, but it included additional real-world considerations; based on the same maps, TurtleBot 2 conducted excellent physical tests. The robot successfully navigated from the starting position to the target coordinates without collisions, accurately following the planned path. Despite the challenges posed by the obstacle arrangements, the TurtleBot 2 demonstrated precise movement and responsiveness, validating the algorithm's effectiveness in real-world conditions. The tests confirmed that the system's execution was reliable, with consistent performance across multiple runs. This successful outcome demonstrates the robustness and accuracy of the navigation strategy implemented in this project.

5. RESULTS

The performance of the autonomous navigation system for the TurtleBot 2 robot was evaluated through extensive testing in both simulated and physical environments. The key evaluation criteria were execution time, path accuracy, and scalability, all tested across distinct map scenarios.

5.1 Simulation Results

In the simulation phase, multiple tests were conducted on a 10×10 grid with static obstacles placed strategically to challenge the algorithm's ability to find optimal paths. In the Map 1 scenario, where the robot started at coordinates (2,2) and targeted (7,7), the average execution time for calculating the optimal path was measured at 21.982 milliseconds. This demonstrated a slower response despite the simplicity of the obstacle arrangement. For Map 2, with a starting point of (1,0) and a target at (3,6), the execution time was slightly lower at an average of 15.635 milliseconds, attributed to the algorithm having to explore fewer spaces than Map 1.

Regarding path accuracy, the algorithm successfully guided the robot to the target positions in all maps without collisions. The robot consistently reached its destination

while avoiding obstacles, highlighting the high accuracy of the pathfinding. The A* algorithm was able to effectively handle obstacle avoidance and trajectory optimization, ensuring that the robot followed the planned path precisely.

When it came to scalability, the algorithm demonstrated its ability to maintain performance even as the map complexity increased in Map 2. Despite the addition of more obstacles, this benefits the speed of the process by not increasing the algorithm process values related to blocked cells present on the map.

5.2 Physical Implementation Results

The implementation was carried out using a TurtleBot 2 platform with a Nvidia Jetson TX2 as the main computer as shown in Fig. 10. For the physical implementation, tests were conducted under similar conditions to those of the simulation. The performance in the real world was evaluated by running the system across different obstacle configurations multiple times, see Fig. 10. The execution times in the physical tests ranged from 130 to 160 milliseconds, which were slightly higher than the simulation due to real-world factors like sensor noise and wheel slippage. In terms of path accuracy, the TurtleBot 2 was able to adjust its course in response to deviations caused by external factors such as surface irregularities or sensor inaccuracies, confirming the algorithm's capability to handle real-world challenges while maintaining precise navigation.

Fig. 10: Physical implementation

Lastly, the system's reliability was evident in its consistent performance across multiple runs. TurtleBot 2 was able to navigate without collisions in various environments, demonstrating its ability to adapt well to unexpected changes. This highlights the stability and reliability of the path-planning solution in real-world applications.

The results from both simulation and physical testing corroborate each other, confirming the effectiveness of the proposed navigation system. The algorithm showed adaptability to different obstacle arrangements, and the robot demonstrated reliable behavior in both simulated and physical settings. These results validate the use of discrete path-planning algorithms like A* for autonomous navigation, making the system suitable for real-world deployment in mobile robotics.

6. CONCLUSIONS

In conclusion, this work demonstrated the effectiveness and robustness of a navigation algorithm implemented on TurtleBot 2, both in simulation and physical tests. By utilizing a navigation system, the robot successfully navigated from a starting position to a target in a complex environment.

The system's performance in the physical environment validated the results obtained from simulations, confirming that the algorithm can adapt to various obstacle configurations while ensuring precise and collision-free navigation. The tests indicated consistent performance across multiple runs, highlighting the system's stability and reliability for real-world applications, such as autonomous delivery robots or self-driving vehicles. Future work may involve implementing dynamic path-planning algorithms and integrating additional sensors to further enhance the system's efficiency and versatility in different environments.

On this foundation, this work illustrates the capability of robotic systems to perform complex navigation tasks and provides a solid foundation for future developments in autonomous robotics, expanding the potential applications of such systems in real-world scenarios.

7. REFERENCES

[1] U. Orozco-Rosas, K. Picos and O. Montiel, "Hybrid Path Planning Algorithm Based on Membrane Pseudo-Bacterial Potential Field for Autonomous Mobile Robots," *IEEE Access,* vol. 7, pp. 156787 - 156803, 2019.

[2] R. C. Shit, "Precise localization for achieving next-generation autonomous navigation: State-of-the-art, taxonomy and future prospects," *Computer Communications,* vol. 160, pp. 351-374, 2020.

[3] V. Sezer and M. Gokasan, "A novel obstacle avoidance algorithm: "Follow the Gap Method"," *Robotics and Autonomous Systems,* vol. 60, no. 9, pp. 1123-1134, 2012.

[4] D. Serrano, "Middleware and Software Frameworks in Robotics - Applicability to Small Unmanned Vehicles," North Atlantic Treaty Organization, Brussels, 2015.

[5] M. A. Contreras-Cruz, V. Ayala-Ramirez and U. H. Hernandez-Belmonte, "Mobile robot path planning using artificial bee colony and evolutionary programming," *Applied Soft Computing,* vol. 30, no. 1, pp. 319-328, 2015.

[6] U. Orozco-Rosas, O. Montiel and R. Sepúlveda, "Parallel Evolutionary Artificial Potential Field for Path Planning—An Implementation on GPU," in *Design of Intelligent Systems Based on Fuzzy Logic, Neural Networks and Nature-Inspired Optimization*, Studies in Computational Intelligence, vol 601, Springer, 2015.

[7] S. M. Lavalle, Planning Algorithms, Cambridge University Press, 2006.

[8] U. Orozco-Rosas, O. Montiel and R. Sepúlveda, "Mobile robot path planning using membrane evolutionary artificial potential field," *Applied Soft Computing,* vol. 77, pp. 236-251, 2019.

[9] W. Kowalczyk, M. Przybyla and K. Kozlowski, "Set-point Control of Mobile Robot with Obstacle Detection and Avoidance Using Navigation Function - Experimental Verification," *J Intell Robot Syst,* vol. 85, p. 539–552, 2017.

[10] K. Katona, H. A. Neamah and P. Korondi, "Obstacle Avoidance and Path Planning Methods for Autonomous Navigation of Mobile Robot," *Sensors,* vol. 24, no. 11, p. 3573, 2024.