Strojnícky časopis – *Journal of* MECHANICAL ENGINEERING,
VOL 74 (2024), NO 1, 103 - 114

INCREASING THE EFFICIENCY OF MOBILE ROBOT NAVIGATION USING SLAM WITH SEVERAL ADVANCED ALGORITHMS AND FILTERS

PAJAZITI Arbnor[1], BAJRAMI Xhevahir[1*], FANDAJ Arjanit[1]

[1] *Faculty of Mechanical Engineering, University of Prishtina, Department of Mechatronics, Str. Kodra e Diellit, n.n., 10000 Prishtina, Kosovo, e-mail: xhevahir.bajrami@uni-pr.edu*

Abstract: Several algorithms such as A*, Dijkstra, SLAM (Simultaneous Localisation and Mapping) and APF (Artificial Potential Field) were used in this study to create local maps, plan the shortest path, and localize mobile robots. In fact, when compared to the SLAM/APF method, these algorithms achieved a reduction in road length by 1.18 meters. Nevertheless, the SLAM/APF method outperformed the Dijkstra algorithm by reducing navigation time by 7.62 seconds and surpassed the A* method by reducing navigation time by 5.76 seconds.

KEYWORDS: SAM, A*, Dijkstra, Algorithm, Turtlebot2i.

1 INTRODUCTION

Navigation is an important challenge for autonomous mobile robots. First, mobile autonomous navigation robots must localise and then determine the shortest path between two points in a short time and with high accuracy [1].

Simultaneous Localisation and Mapping (SLAM) aims to localise and create maps of movements in real time and can be implemented in many methods, so it is often considered as a concept or field of study rather than just an algorithm. Several algorithms are used for its implementation whereby SLAM perceives the mobile robot's surrounding environment by linking data, planning the next movement, achieving the desired movement, and finally updating the map of the environment or the movement, respectively [2-5]. SLAM has been the topic of much research and is important for the creation of maps that are used in autonomous robot navigation systems.

Visual SLAM is used in many mobile robots and other techniques are being evaluated to develop optimal methods for the localisation of robots and the creation of maps to facilitate movement [6]. Path planning is one of the main techniques for navigating robots in environments with static or dynamic obstacles.

Global path planning, also known as static planning, refers to the robot perceiving its environment through sensors and then planning the corresponding path based on this data [7]. The route planning in this study was achieved through several algorithms including A*, Dijkstra, and SLAM/APF (Artificial Potential Field). The APF method is prone to local optimisation and stagnation in the robot's coordinates, whereas neural networks have a much higher intensity of data generation and require many models for training [8].

In principle, SLAM is dependent on the surrounding environment, scanning and measuring the distance of the surrounding physical objects, and depends on detecting the obstacles through sensors. There are three main SLAM methods [9].

DOI: 10.2478/scjme-2024-0011, Print ISSN 0039-2472, On-line ISSN 2450-5471

Various vision-based SLAM strategies have been proposed utilising distinctive input sensors, such as monocular RGB-D SLAM [10], stereo SLAM [11], and Hammer [12], due to their abundance of data compared to other onboard sensors. There are also learning-based methods such as DynaSLAM [13], DS-SLAM [14] and the baseline RTAB-Map [15] that accomplish satisfactory execution when there are few dynamic obstacles. The YolactEdge technique has been used as it permits higher computational speed with moderately competitive exactness [16].

In this study, the Gmapping filter was used to create maps and AMCL for localization in static maps. Section 3 presents the integration of the SLAM process in ROS, and how the OGM maps were created and stored. After the modelling and simulation, it was tested with real robots and compared to the simulations.

Section 4 discusses the localisation of mobile robots with SLAM and the problems that arise during localisation. Then, the algorithms were applied for path planning algorithms such as A* or Dijkstra, and Convolutional Neural Networks based on Regions (CNN-R) during the navigation of the Turtlebot2i mobile robot. The conclusion and a summary of the proposed framework for future work are provided in Section 5.

2 MATERIALS AND METHODS

The main task of a mobile robot is navigation through static and dynamic obstacles, which must be as accurate and efficient as possible, and involves the movements of the robot towards a defined destination. It is also important that the robot is aware of its location and has a map of the given environment to determine the optimal route to avoid physical obstacles. The final essential feature of navigation is the calculation and travel of the optimal route to the destination, that is, route search and planning. Many algorithms provide safe navigation by using filters such as Gmapping, Extended Kalman Filter (EKF), potential field, and particle filter. The three main filters used for SLAM are explained in detail in the following sections.

2.1 Extended Kalman Filter (EKF)

EKF is a popular filter applied in the real world from autonomous cars to drones [18]. This filter is required when the different sensors do not transmit clean signals which reduces the accuracy of the measurement. The EKF corrects the sensor measurements to better calculate the position of the robot while it is moving. Furthermore, EKF is an algorithm that enables many parameters of the robotic motion i.e., the state variables model to make fine adjustments to the actual sensing measurements, to sense the amount of noise and as a result, to obtain information about the robot motion [17, 18]. The robot movements of the mobile robot in Fig. 1 were calculated using the following mathematical equations:

g – Rotation along the z-axis,
ω – Angular velocity along the z-axis,
v – Mobile robot velocity,
v_x – linear velocity along the x-axis $v_x = v \cdot \cos(g)$,
v_y – linear velocity along the y-axis $v_y = v \cdot \sin(g)$,

Assuming that the robot is at rest [19], the vector for the previous state is at time t-1 and that the oriented motion of the robot moves along the x-axis:

$$\hat{x}_{t-1|t-1} = \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \gamma_{t-1} \end{bmatrix} \quad (1)$$

In this case, if the robot moves for dt – time interval, then the distance is equal to the speed passed during the interval, so:

$$\begin{bmatrix} x_t \\ y_t \\ \gamma_t \end{bmatrix} = \begin{bmatrix} x_{t-1} + \vartheta_{t-1} \cdot cos(\gamma_{t-1}) \cdot \ dt \\ y_{t-1} + \vartheta_{t-1} \cdot \sin(\gamma_{t-1}) \cdot \ dt \\ \gamma_{t-1} + \omega_{t-1} * \ dt \end{bmatrix} = \begin{bmatrix} f_1 \\ f_2 \\ f_3 \end{bmatrix} \quad (2)$$

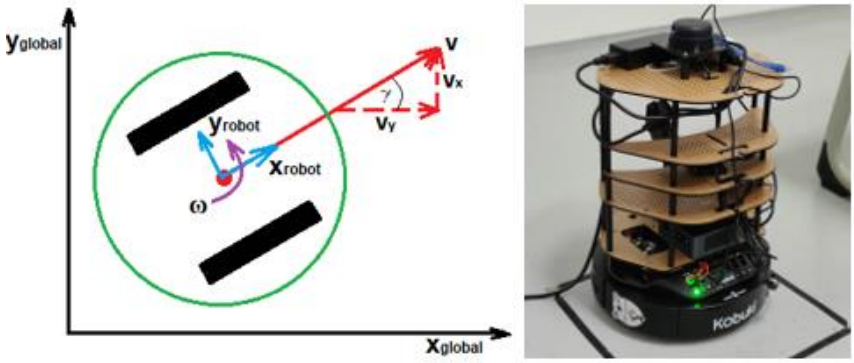


Fig. 1 Mobile robot coordinates left and real Turtlebot2i right.

Equation (2) describes the movement of the robot through mathematical equations, but this equation is non-linear and for the EKF [18, 19] calculations, it needs to convert the linear equation so the state variables of the mobile robot will be:

$$x_k = A_{k-1} \cdot x_{k-1} + B_{k-1} \cdot u_{k-1} \quad (3)$$

Where,

x_k – Current state vector $[x_k, y_k, \gamma_k]$,
x_{k-1} – Previous state of the mobile robot $[x_{k-1}, y_{k-1}, \gamma_{k-1}]$
u_{k-1} – Input vector of k-1 $[\vartheta_{k-1}, \omega_{k-1}]$ sample.

2.2 Gmapping

Gmapping is called FastSLAM algorithm and is a very efficient particle filter known as Rao-Blackwellised described in [6] which creates grid-like maps from information received from laser sensors. These filters are relatively effective in terms of SLAM since each particle contains an individual map of the environment and this filter performs the reduction to learn the map, as well as enables selective resampling operations to be performed. The main idea of this filter is to calculate the relation $\rho(x_{1:k}, m\,|z_{1:k,}u_{1:k-1})$ for the map m and the trajectory $x_{1:t} = x_1, \ldots\ldots, x_k$ of the mobile robot. This calculation is achieved by including the values obtained from the sensor $z_{1:k} = z_1, \ldots\ldots z_k$ and the odometric measurements $u_{1:k-1} = u_1, \ldots\ldots u_{k-1}$ of the mobile robot [18]:

$$\rho(x_{1:k}, m\,|z_{1:k,}u_{1:k-1}) = \rho(m|x_{1:k}, z_{1:k}) \times \rho(x_{1:k}|z_{1:k}, u_{1:k-1} \quad (4)$$

3 SLAM INTEGRATION IN ROS

This section clarifies how SLAM is integrated into ROS. Three main functions are needed to realise SLAM maps, position, and distance. In ROS, each of these is a function that is interconnected during the realisation of SLAM. These functions are usually ready for application in the form of packages and can be applied and purposefully coordinated to achieve more efficient results. Referring to the purpose of the paper, we will first clarify how maps are created in ROS using the Turtlebot mobile robot. Turtlebot2i is a robotic platform based on ROS, in addition to the redesigned chassis of Turtlebot2, the Pincher MK3 robotic arm with four degrees of freedom has been added as a function for the robot to interact with small objects in the real world, which makes this model a very advanced robot manipulator. The main challenge is to translate the map into the language that the robot understands since the robot

cannot read maps from paper. The definition of maps in robotics continues to be discussed and developed, so more recent maps include two-dimensional as well as three-dimensional dimensions. A two-dimensional OGM (Occupancy Grid Map) was used in this study and refers to a set of algorithms in the field of probabilistic robotics, whereby maps are generated based on noise and sensor measurements assuming that the position of the robot is known. The main purpose of this algorithm is to calculate the posterior probability for the maps considering the measured data:

$$\rho(m|z_{1:k}, x_{1:k}) \tag{5}$$

Where, m – represents the map,

z1: k – represents measurements from time 1 to t (input values measured by the sensor),

x1: k – represents the position of the robot from time 1 to t.

A simple SLAM manoeuvre is performed to clarify the OGM maps and save them. Thus, four terminals are opened in the operating system to execute the commands. Turtlebot2i navigation is performed using the keyboard to move the robot through the environment, where there are several corridors displayed and the environment is created in the Gazebo simulator.

Ready-made packages provided by ROS are used to simulate SLAM in ROS. There are three main packages for starting the robot turtlebot, for SLAM algorithm gmapping, for visual presentation view. Four terminals are opened in ROS to run the packages [7]. After navigating through the entire map, which took more than 2 hours, the map is saved and AMCL is opened for map navigation (Fig. 2).

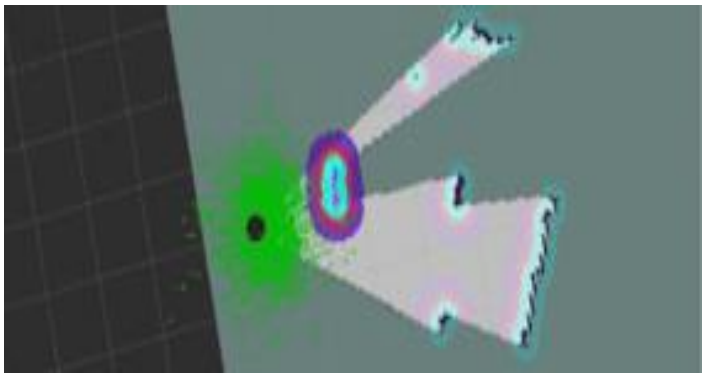


Fig. 2 SLAM simulation and mapping in ROS.

4 RESULTS AND DISCUSSION

Section 3 explained almost all elements of SLAM by creating the simulation of maps in a virtual environment, etc. In this section, SLAM is employed using the physical robot Turtlebot2i. First, the environment is configured, then the map is created using SLAM and finally, autonomous navigation through the environment. Since the map must be created in real-time as well as run SLAM and display the results, the communication between ROS must be configured, which is installed on the Personal Computer (PC) and referred to as the client and ROS installed on Turtlebot2i. The real robot will be the master node and the client will be ROS which is installed on the PC. The mobile robot is placed in the Mechatronics lab where we run SLAM, and the robot is navigated around the lab until a satisfactory map is generated. Two terminals are opened in the Turtlebot2i robot to start the robot and the Gmapping algorithm. For better results, the robot navigated through the entire laboratory as well as the entrance to the laboratory, stopping to perform a 360^oscan to create the most accurate position in relation

to static obstacles (Fig. 3). The scanning lasted about 30 min but the longer the navigation time, the more accurate the created map (Fig. 4).

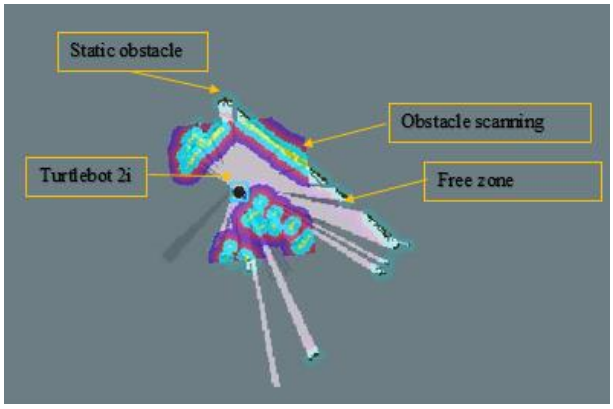


Fig. 3 Simulation results with SLAM after 30 seconds of navigation.

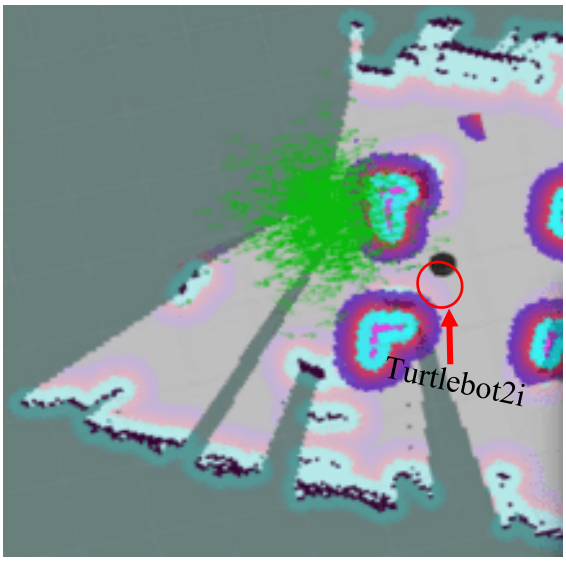


Fig 4. Map generation with SLAM.

4.1 Filter based CNN

In the current study, the researchers have recognized that significant image noise or occlusions can negatively affect the accuracy of predictions made by a CNN [20]. Image noise refers to any factors that make it challenging to accurately determine the level interface directly from the image. To overcome this challenge, the researchers have combined a Kalman filter with the CNN. This integration involves the CNN providing estimations not only for the level but also for the quality of the image. By combining the capabilities of the CNN with the predictions from the state, this approach allows for an optimal utilization of both the image data and the state prediction. Furthermore, the researchers propose an online update to enhance the algorithm's resilience to various types of occlusions that were not included in the training set Fig. 5.

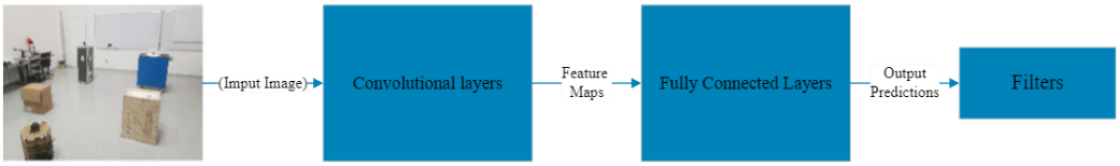


Fig. 5 Algorithm's architecture with filters and training methods.

Then, data is generated for planning the shortest route through the algorithms and navigating the robot through visual identifiers to avoid static or dynamic obstacles. Object detection is needed to obtain information in such a way that during navigation, the robot has information about the distance from the robot to the object. All this information is passed to the algorithms for calculating and creating the path reduction to the goal. In addition to static obstacles, there are also dynamic obstacles which present the most problems during autonomous navigation. The information on how far a dynamic object can be depends on the speed of the moving object as well as the speed of the robot. The robot can receive information through cameras, but an application is needed through which objects are detected.

Mathematical models are used to avoid this problem, for example, to calculate the average width and length of that object in the dataset or we can use distance sensors can be used to calculate the distance from the robot to the object for more accurate results, and they can get information about the object in front of the robot through the camera. Fig. 6 shows the view of the objects detected by the robot during navigation.

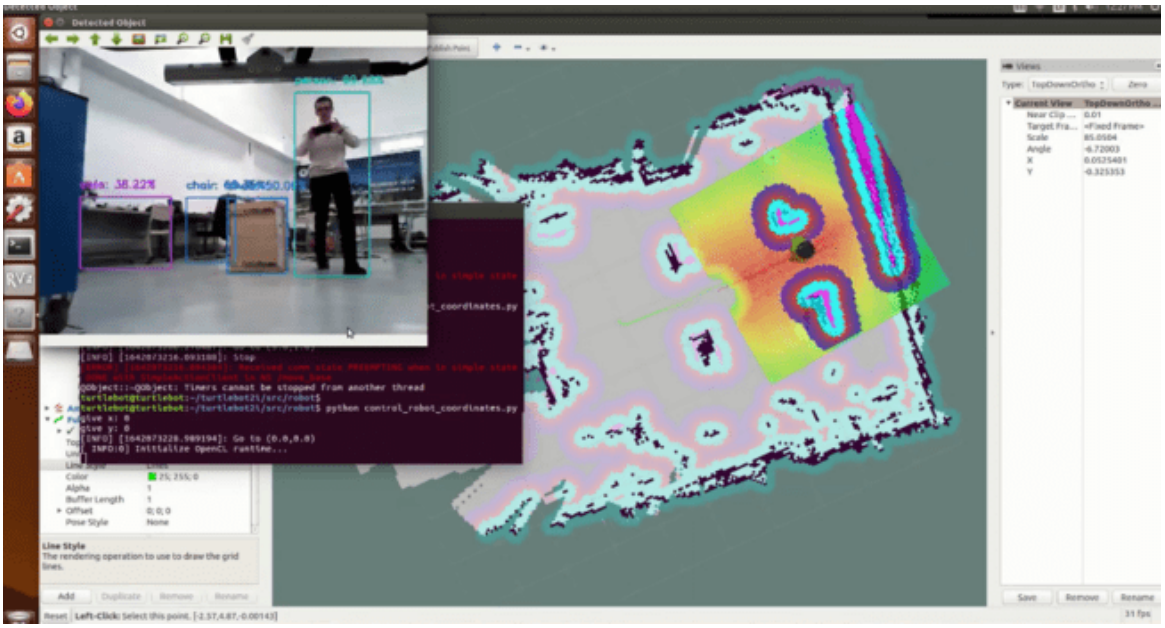


Fig. 6 Navigation through visual identifiers.

Dijkstra and SLAM/APF. These methods were used to generate different results but as SLAM was integrated by the robot to optimise the shortest path, an even shorter path was created than that created through SLAM/APF. Fig. 7 and 8 show the application for creating the shortest path through the A* and Dijkstra algorithms.

Algorithm A* starts from the green node and considers all cells around the starting point, then once the list of cells is filled, it filters out those that are inaccessible such as walls, obstacles, or even parts outside the boundaries that have been set. It then selects the cell with the lowest cost and this process is repeated recursively until the shortest path to the target is found.

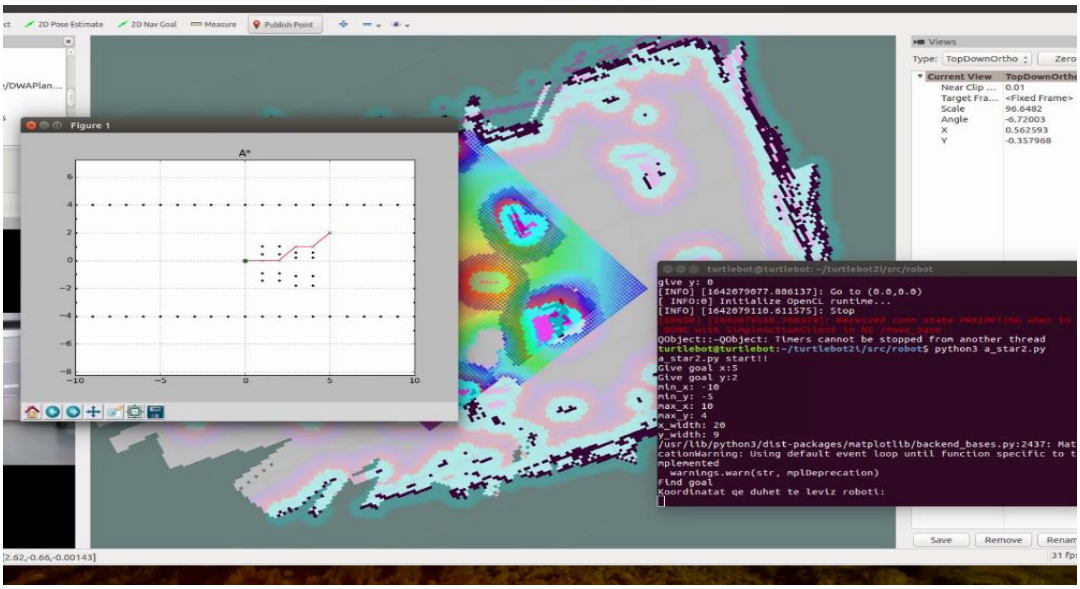


Fig. 7 Generation of coordinates by algorithm A* to point (5, 2).

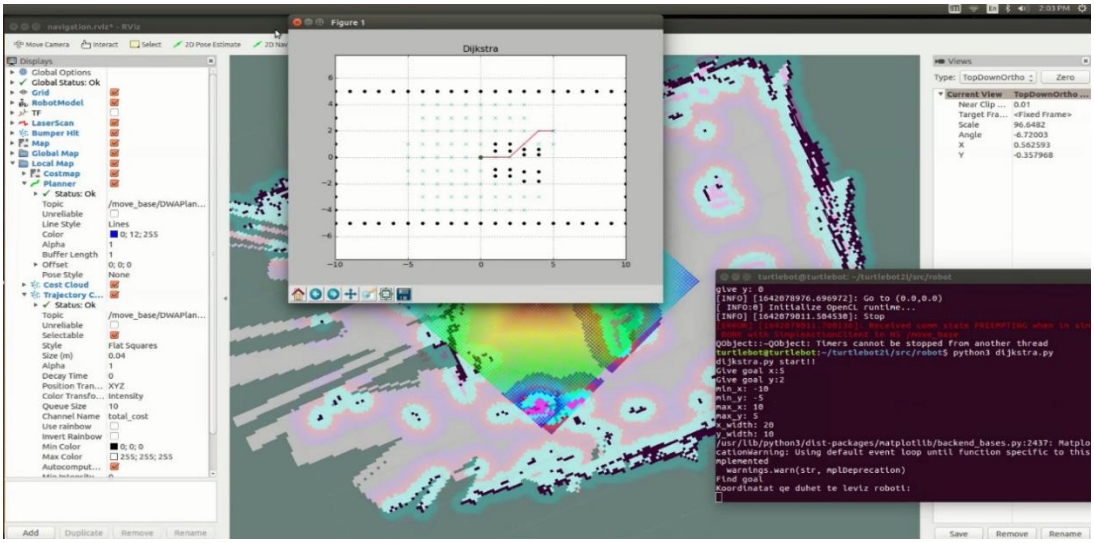
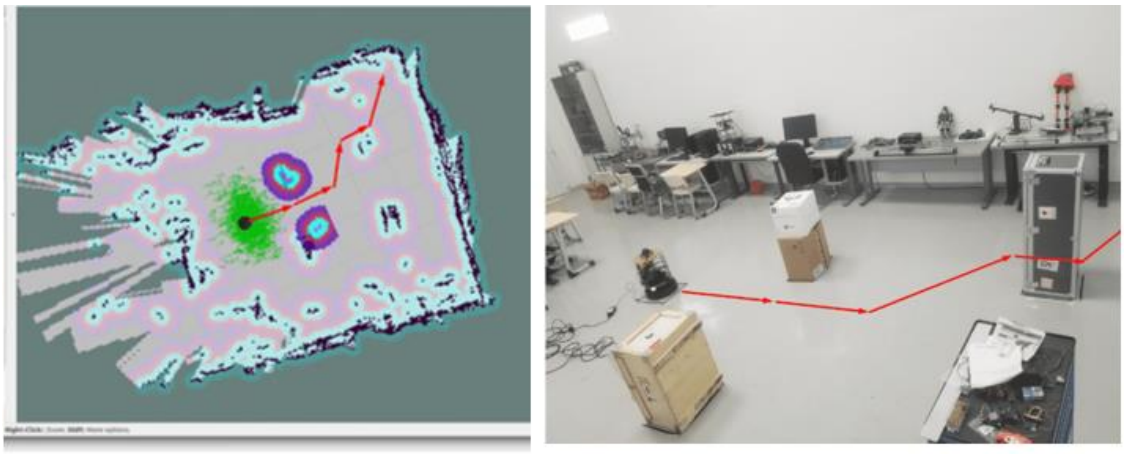


Fig. 8 Generation of the shortest path through the Dijkstra algorithm
from point (0,0) to point (5, 2).

The navigation time of the robot to the destination was shorter using SLAM/APF than with A* (Fig. 9 a&b) and Dijkstra (Fig. 10 a&b).

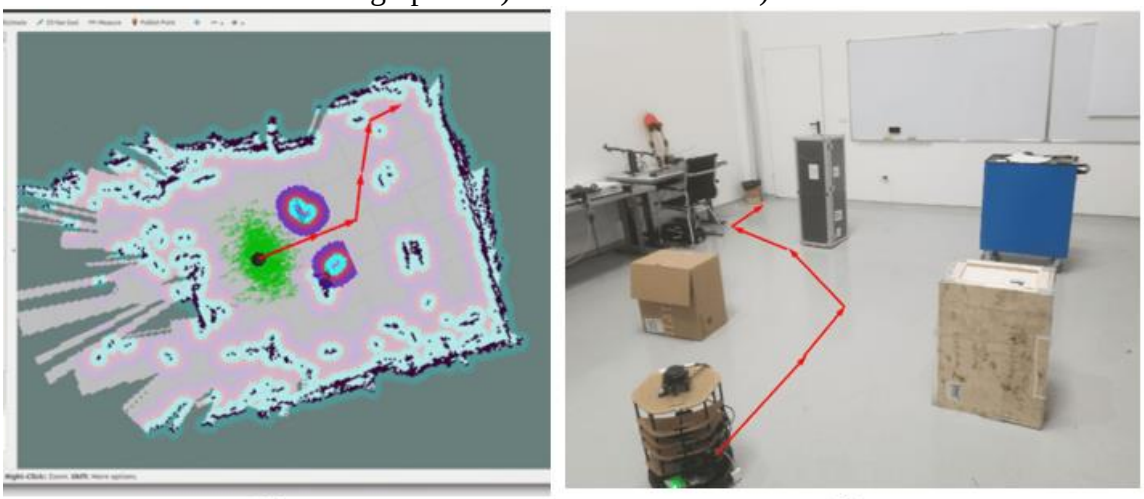
This happens because A* and Dijkstra at each point have to calculate the next step where it should stay during that coordinate, and whether that cell is free or not, whereas, in the SLAM, the map is created at the beginning and used as a reference for all static obstacles. However, in the case of dynamic obstacles, it will take the same time as Dijkstra or A*.

The robot navigation for map creation and shortest path generation shown in Fig. 11 a and b is achieved through SLAM/APF.



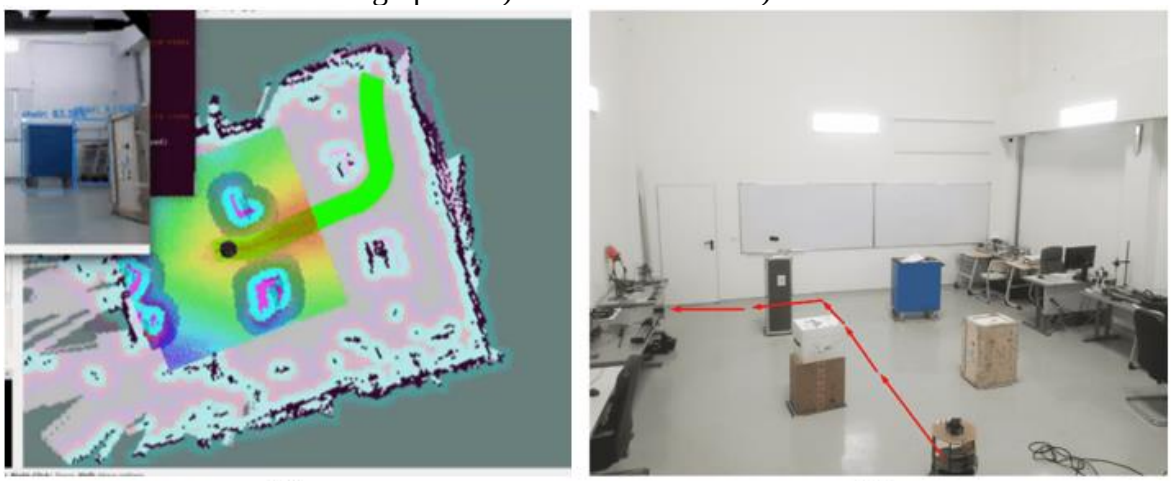
(a) (b)

Fig. 9 Turtlebot2i navigation through A* algorithm to point (5, 2): graphics a) and environment b).



(a) (b)

Fig. 10 Turtlebot2i navigation through the Dijkstra algorithm to point (5, 2): graphics a) and environment b).



(a) (b)

Fig. 11: Turtlebot2i navigation through SLAM/APF to point (5, 2): graphics a) and environment b).

The summary results presented in Table 1 show the time from when the robot passed to the destination and how many nodes were used to calculate the path.

Table 1: Comparison of the three algorithms A*, Dijkstra and SLAM/APF.

Algorithms	Time (s)	Distance (m)	Nodes
A*	23.57	5.82	5
Dijkstra	25.43	5.82	5
SLAM/APF	17.81	7	-

The A* and Dijkstra algorithms are much more efficient for creating a shorter path but during navigation, the time for calculating the values is longer than when using the SLAM/APF algorithm, since the robot must stop and review the obstacles and calculate the next coordinates.

In Table 1, we are presented with a comparison of three algorithms: A*, Dijkstra, and SLAM/APF. The table includes measures such as time (in seconds), distance (in meters), and the number of nodes explored.

- Time (s):

- A* algorithm: The A* algorithm took 23.57 seconds to complete its task.
- Dijkstra algorithm: The Dijkstra algorithm required 25.43 seconds.
- SLAM/APF algorithm: The SLAM/APF algorithm performed the task in 17.81 seconds.

From the time measurements, we can observe that the SLAM/APF algorithm was the fastest, followed by the A* algorithm, and finally the Dijkstra algorithm. This implies that the SLAM/APF algorithm was able to find a solution in the shortest amount of time, while Dijkstra took the longest.

- Distance (m):

- A* algorithm: The A* algorithm calculated 5.82 meters distance.
- Dijkstra algorithm: The Dijkstra algorithm reached 5.82 meters distance.
- SLAM/APF algorithm: The SLAM/APF algorithm also calculated 7 meters distance.

Both the A* algorithm and the Dijkstra SLAM/APF algorithm obtained the same distance, while the SLAM/APF algorithm achieved a slightly longer distance. This suggests that the A* and Dijkstra algorithms may have found more optimal paths to the destination compared to Dijkstra.

Based on the given information, it is difficult to draw conclusions about the node exploration of the SLAM/APF algorithm, as no value is provided. However, both A* and Dijkstra algorithms explored the same number of nodes, indicating a similar level of exploration.

In summary, the table provides a comparison of the three algorithms based on time, distance, and the number of nodes explored. The SLAM/APF algorithm demonstrated the shortest execution time, while both A* and Dijkstra achieved the same distance. The node exploration results are inconclusive for the SLAM/APF algorithm, but A* and Dijkstra explored an equal number of nodes.

CONCLUSION

The Conclusions section should clearly explain the main findings and implications of the work, highlighting its importance and relevance. The use of other algorithms such as shortest path generation, as well as mapping regularisation along SLAM-generated problems, have been equally successful in performing their tasks, such as the A* and Dijkstra algorithms, where the road has been reduced by 1.18 m compared to the SLAM/APF method. However, the navigation time of the SLAM/APF method was reduced by 7.62(s) compared to the Dijkstra algorithm and 5.76(s) compared to the A* method. The future works aim to advance the state-of-the-art on mobile robot navigation, improving efficiency, accuracy, and robustness. By leveraging advanced algorithms, filters, and incorporating additional information, the performance of control systems can be enhanced, enabling mobile robots to navigate more effectively and autonomously in various real-world scenarios.

Data Availability

The data used to support the findings of this study are available from the corresponding author upon request.
https://github.com/FIM-Mechatronics/AutonomosTurtlebot-A.Fandaj

Conflicts of Interest

The authors declare that they have no conflict of interest with respect to this paper.

Abbreviations:

ROS - Robot Operating System
RVIZ - ROS visualisation
SLAM - Simultaneous Localisation and Mapping
APF - Artificial Potential Field
GPS - Global Positioning Systems
EKF - Extended Kalman Filter
CNN-R - Convolutional Neural Networks based on Regions (CNN-R)
AMCL - Adaptive Monte Carlo Localisation
OGM - Occupancy Grid Map
MLP - Multi-Layer Perceptron
SVM - Support Vector Machine
LiDAR - Light Detection and Ranging or laser imaging detection and ranging.

REFERENCES

[1] Kamil Shehata, F K., A, H. H., El-Batsh, H. M. "Mobile robot obstacle avoidance based on neural network with a standardization technique", Journal of Robotics, pp. 1 – 14, **2021**.

[2] Gunaza, W., Zhongmin, W., Yu., Y. "Optimization of SLAM Gmapping based on simulation", International Journal of Engineering Research & Technology 9, pp. 74 – 81, **2020**.

[3] Tim, B., Durrant-Whyte, H. "Simultaneous localization and mapping (SLAM): Part II", IEEE robotics & automation magazine 13 (3), pp. 108 – 117, **2006**.

[4] Theodoros, T., Hu, H., McDonald-Maier, K., Gu. D. "Kinect enabled monte carlo localisation for a robotic wheelchair", Frontiers of Intelligent Autonomous Systems, pp. 17 – 27, **2013**.

[5] Marcell, M., Bennewitz, M. "Predictive collision avoidance for the dynamic window approach." International Conference on Robotics and Automation (ICRA), pp. 8620 – 8626, IEEE, **2019**.

[6] Vicky, K. Ioannidis, S, K., Sirakoulis, G. C., Kosmatopoulos, E. B. "Real-time active SLAM and obstacle avoidance for an autonomous robot based on stereo vision." Cybernetics and Systems 50 (3), pp. 239 – 260, **2019**.

[7] Li-sang, L., Lin, J., Yao, J., He, D., Zheng J., Jing, H., Peng, S. "Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach", Wireless Communications & Mobile Computing (Online) **2021**.

[8] Li, Fei-Fei, Yun Du, and Ke-Jin Jia. "Path planning and smoothing of mobile robot based on improved artificial fish swarm algorithm", Scientific Reports 12 (1), pp. 1 – 16, **2022**.

[9] Li-sang, L., Jia-feng, L., Jin-xin, Y., Dong-wei, H., Ji-shi, Z., Huang, J., Shi. P. "Path Planning for Smart Car Based on Dijkstra Algorithm and Dynamic Window Approach", Wireless Communications & Mobile Computing (Online), pp. 1 – 16, **2021**.

[10] Wen, W., Hsu, L., Zhang, G. "Performance Analysis of NDT-based Graph SLAM for Autonomous Vehicle in Diverse Typical Driving Scenarios of Hong Kong", MDPI, Sensors 18(11), 3928, **2018**.

[11] Mur-Artal, R., Tardos, J. D. "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras", IEEE Trans. Robot. 33 (5), pp. 1255 – 1262, **2017**.

[12] Gomez-Ojeda, R., Francisco-Angel, M., Zuniga-Noël, D., Scaramuzza, D., Gonzalez-Jimenez, J. "PL-SLAM: A stereo SLAM system through the combination of points and line segments", IEEE Transactions on Robotics 35 (3), pp. 734 – 746, **2019**.

[13] Yan, F., Copeland, R., Brittain, H. G. "LSD-SLAM: Large-Scale Direct Monocular SLAM", ECCV 72 (C), pp. 211 – 216, **2014**.

[14] Berta, B., Fácil, J. M., Civera, J., Neira, J. "DynaSLAM: Tracking, mapping, and inpainting in dynamic scenes", IEEE Robotics and Automation Letters 3 (4), pp. 4076 – 4083, **2018**.

[15] Chao, Y., Liu, Z., Liu, X.-J., Xie, F., Yang, Y., Wei, Q., Fei, Q. "DS-SLAM: A semantic visual SLAM towards dynamic environments", IEEE/RSJ international conference on intelligent robots and systems (IROS), pp. 1168 – 1174, IEEE, **2018**.

[16] Labbe, M., "RTAB-Map as an Open-Source Lidar and Visual SLAM Library for Large-Scale and Long-Term Online Operation", J. F. Robot. 36 (2), pp. 416 – 446, **2019**.

[17] Bajrami, X., Ahmet, S., Gezim, H., Rame, L. "Dynamic modelling and analyzing of a walking of humanoid robot", Strojnícky časopis – Journal of Mechanical Engineering 68 (3), pp. 59 – 76, **2018**. DOI: 10.2478/scjme-2018-0027

[18] Mathieu, L., Michaud, F. "RTAB-Map as an open-source lidar and visual simultaneous localization and mapping library for large-scale and long-term online operation", Journal of field robotics 36 (2), pp. 416 – 446, **2019**.

[19] Erjon, S., Bajrami, X., Likaj, R., Pajaziti, A. "Real Time Swinging Up and Stabilizing a Double Inverted Pendulum Using PID-LQR", Strojnícky časopis – Journal of Mechanical Engineering 73 (1), pp. 159 – 168, **2023**. DOI: 10.2478/scjme-2023-0013

[20] Pajaziti, A., Bajrami, X., Paliqi, A. "Path Control of Quadruped Robot through Convolutional Neural Networks", IFAC-PapersOnLine 51 (3), pp. 610 – 615, **2018**.