Rapidly Exploring Random Trees for Autonomous Navigation in Observable and Uncertain Environments

Fredy Martínez, Edwar Jacinto, Holman Montiel
Universidad Distrital, Francisco José de Caldas
Bogotá D.C., Colombia

***Abstract*—This paper proposes the use of a small differential robot with two DC motors, controlled by an ESP32 microcontroller, that implements the Rapidly Exploring Random Trees algorithm to navigate from an origin point to a destination point in an unknown but observable environment. The motivation behind this research is to explore the use of a low-cost, versatile and efficient robotic platform for autonomous navigation in complex environments. This work presents a practical and cost-effective solution that can be easily replicated and implemented in various scenarios such as search and rescue, surveillance, and industrial automation. The proposed robotic platform is equipped with a set of sensors and actuators that allow it to observe the environment, estimate its position, and move through it. The Rapidly Exploring Random Trees algorithm is implemented to generate a path from an origin to a destination point, avoiding obstacles and adjusting the robot's motion accordingly. The implementation of this algorithm enables the robot to navigate through complex environments with high efficiency and reliability, making it a suitable solution for a wide range of applications. The results obtained through simulations and experiments show that the proposed robotic platform and algorithm achieve high performance and accuracy in autonomous navigation, even in complex environments.**

***Keywords*—Autonomous navigation; differential robot; esp32 microcontroller; low-cost; rapidly exploring random trees algorithm; versatile**

I. Introduction

Robotics is an interdisciplinary field that deals with the design, construction, and operation of robots [1]. Robots have been used for various purposes, such as manufacturing, healthcare, exploration, and entertainment [2]. The development of robotics technology has been driven by the need for automation, improved efficiency, and the desire to reduce human error [3]. Small differential robots are a common type of robot used for various applications, including surveillance, exploration, and educational purposes [4]. In this paper, we propose the use of a small differential robot with two DC motors, controlled by an ESP32 microcontroller, with position sensors, distance sensors, motor encoders, gyroscopes, and cameras, that implements the Rapidly Exploring Random Trees (RRT) algorithm to navigate from an origin point to a destination point in an unknown but observable environment [5], [6], [7].

The use of small differential robots for navigation in unknown environments has been an area of interest for researchers for many years [8], [9]. The ability of these robots to navigate in tight spaces and uneven terrains makes them suitable for exploring unknown environments [10]. The challenge, however, lies in the development of algorithms that allow the robots to navigate effectively in such environments. The RRT algorithm is one such algorithm that has gained popularity in recent years due to its ability to efficiently search high-dimensional spaces [11], [12], [13].

The RRT algorithm is a motion planning algorithm that generates a tree of feasible paths through a high-dimensional configuration space [14]. The algorithm is designed to quickly explore the search space by generating random samples and expanding the tree towards the samples. The RRT algorithm has been successfully applied to various robotics applications, including path planning, motion planning, and autonomous navigation [15].

The development of small differential robots with advanced sensors and microcontrollers has made it possible to implement the RRT algorithm for navigation in unknown environments [16]. The combination of sensors such as position sensors, distance sensors, motor encoders, gyroscopes, and cameras provide the robot with the necessary information to navigate in its environment [17]. The use of an ESP32 microcontroller provides the robot with the necessary computational power to process the sensor data and execute the RRT algorithm.

One of the most significant challenges of implementing the RRT algorithm on a small differential robot is the limited computational power and memory of the microcontroller [18], [19], [20]. As the robot's size is limited, so is the computational power of its microcontroller, which can affect the algorithm's efficiency. Hence, when designing the algorithm, it is essential to optimize it to work within the microcontroller's limitations [21]. Furthermore, it is vital to minimize the memory requirements of the algorithm to ensure efficient use of the microcontroller's limited memory [22], [23].

In addition to the microcontroller's limitations, the robot's size and weight must also be taken into account [24]. These factors can impact the robot's maneuverability in tight spaces, making it difficult for the robot to generate collision-free paths [25]. To overcome these challenges, the design of the robot must take into consideration the weight and size of the components used, and the algorithms must be optimized for use on small robots.

Another crucial factor that can affect the performance of the algorithm is the choice of sensors and their placement on the robot [26]. The right sensors must be chosen and placed optimally to obtain accurate and reliable data on the environment

[27]. The sensors must be chosen with a trade-off between cost and performance, and their placement must consider the robot's size and weight to ensure optimal performance [28].

In this paper, we present a small differential robot that is equipped with advanced sensors and an ESP32 microcontroller that is capable of implementing the RRT algorithm for navigation in unknown but observable environments. We evaluate the performance of the robot in various environments and compare its performance to other navigation algorithms. We also discuss the challenges associated with the implementation of the RRT algorithm on a small differential robot and propose solutions to overcome these challenges.

II. Background

The implementation of autonomous navigation algorithms in dynamic and uncertain environments is an active research area in robotics. The study [29] proposed a navigation algorithm that enables robots to navigate in dynamic environments with moving obstacles. They used a dynamic obstacle detection and tracking algorithm, and a novel cost function to generate the collision-free path. The architecture and implementation of an autonomous passenger vehicle designed to navigate using locally perceived information are presented in [30]. They proposed a hybrid approach using online perception and planning algorithms to achieve autonomous navigation in urban environments.

In the presence of high clutter, the performance of the Concurrent-SLAM (CSLAM) algorithm is reduced. [31] proposed an approach to improve the performance of the CSLAM algorithm by combining an effective clutter filter framework based on Random Finite Sets (RFS). They also presented an improved algorithm that addresses the limitations of the traditional RRT algorithm, named Heuristic Bi-directional Discrete Rapidly-explore Random Trees (HBD-RRTs), that outperforms the RRT algorithm in terms of path quality and computation time.

A path planning system for autonomous navigation of unmanned aerial vehicles based on a combination of RRT* Goal and Limit, which enables the vehicle to navigate in complex environments while avoiding obstacles, was proposed by Aguilar et al. [32]. Wang et al. [33] presented an integrated software and hardware system for autonomous mobile robot navigation in uneven and unstructured indoor environments, using a hybrid approach that combines mapping and localization techniques to achieve autonomous navigation.

It is important to identify the boundary case scenarios where an autonomous vehicle can no longer avoid a collision. Tuncali et al. [34] proposed an automated test generation approach that utilizes Rapidly-exploring Random Trees to explore these boundary scenarios. Their approach generates scenarios that are difficult for the navigation algorithm to handle and can be used to evaluate the performance of the navigation algorithm.

Ayawli et al. [35] presented an optimized rapidly exploring random trees A* (ORRT-A*) method to improve the performance of the RRT-A* method to compute safe and optimal paths with low time complexity for autonomous mobile robots in partially known complex environments. Their approach reduces the computation time while ensuring the optimality and safety of the generated path.

Zhang et al. [36] extended the RRT algorithm to propose an optimization-based map exploration strategy for multiple robots to actively explore and build environment maps. Their approach uses a multi-robot coordination algorithm that assigns exploration tasks to different robots while avoiding collisions.

In the presence of external agents, ensuring safety without sacrificing performance becomes extremely challenging. Bak et al. [37] presented an approach to stress test autonomous systems using the RRT algorithm. Their approach generates scenarios that test the robustness and safety of the system by introducing unexpected changes to the environment or system behavior.

III. Methods

Our working platform is based on the Arduino Controlled Servo Robot (SERB) [38]. The SERB is a small, low-cost robot that can be programmed and controlled using a microcontroller. The robot is designed to be highly customizable and adaptable, making it an ideal platform for both educational and research purposes. In this regard, we have modified the platform to use a different control unit (Espressif Systems ESP32), and incorporate our sensors (Fig. 1).

The SERB is based on a four-wheel drive system, with each wheel powered by a small DC motor and controlled using a dedicated servo motor. This allows for precise control of the robot's movement, as well as the ability to turn on the spot and navigate tight spaces.

The robot is equipped with a range of sensors, including a LiDAR (Light Detection and Ranging o Laser Imaging Detection and Ranging) sensor for obstacle detection and avoidance, a GPS (Global Positioning System) for geolocation, and an IMU inertial unit to determine acceleration and rotation of the robot. Additionally, the SERB has a built-in camera module that can be used for vision-based tasks, such as object recognition and tracking.

One of the key features of the SERB is its ease of use and flexibility. The robot can be programmed using a small microcontroller. Additionally, the SERB is compatible with a wide range of sensors and modules, allowing users to customize the robot for a variety of applications.

The SERB has been used in a variety of educational and research settings, including robotics competitions and STEM education programs. Its low cost and ease of use make it an ideal platform for teaching students about robotics and programming, while its adaptability makes it a valuable tool for researchers exploring new applications of robotics technology. These features make it ideal for the development of affordable, high performance, low cost robotic platforms. The initial adaptation we made to this robot, and which is used in this study, is shown in Fig. 2.

In terms of performance, the SERB has demonstrated impressive capabilities in both mobility and sensing. Its four-wheel drive system provides excellent maneuverability and control, while its range of sensors allows for advanced sensing

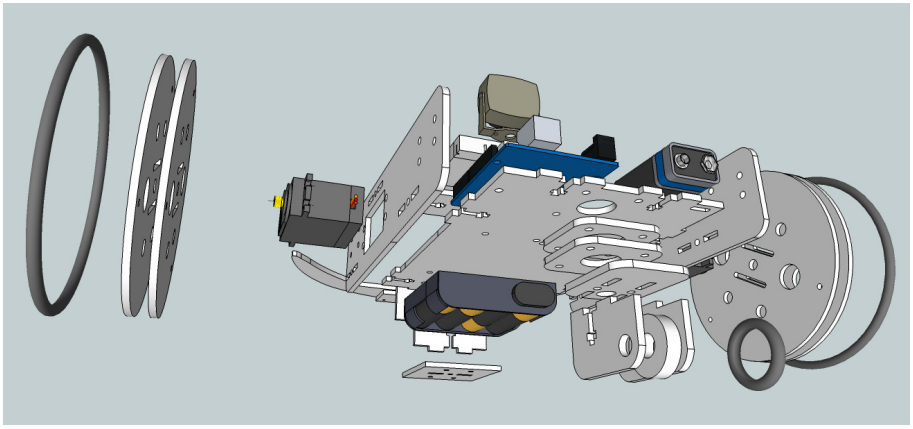

Fig. 1. Structure of the SERB robot.

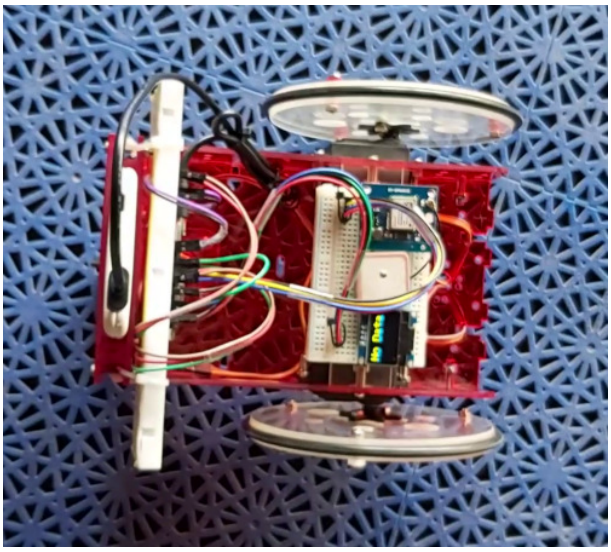

Fig. 2. Differential robotic platform prototype. The robot consists of a SERB robot with a espressif systems ESP32, an U-Blox NEO-6M GPS, a bosch sensortec BNO055, a RPLIDAR A1M8-R6 LiDAR, and an omniVision technologies OV7670 camera.

and perception capabilities. Additionally, the robot's small size and low profile make it well-suited for navigating tight spaces and confined environments.

The sensors installed on the SERB robot are:

1) Position sensor: A position sensor is required to accurately determine the current position of the robot. For this purpose, a GPS module (U-Blox NEO-6M) and an inertial measurement unit IMU (Bosch Sensortec BNO055) that measures the acceleration and rotation of the robot were used.
2) Distance sensor: A distance sensor is needed to detect obstacles and avoid collisions. Infrared and LIDAR, which are capable of measuring the distance to nearby objects, have been used for this purpose (RPLIDAR A1M8-R6).
3) Motor encoder: A motor encoder is necessary to provide feedback on the motor's speed and position. This information is crucial for precise control of the robot's movement and can be used to implement closed-loop control algorithms.
4) Gyroscope: A gyroscope (Bosch Sensortec BNO055) is useful for measuring the robot's rotation and orientation. This information can be used to stabilize the robot and to implement control algorithms that rely on accurate orientation data.
5) Camera: A camera (OmniVision Technologies OV7670) is useful for providing visual feedback to the microcontroller, which can be used to perform tasks such as object recognition, path planning, and navigation. A camera can also be used to provide feedback on the robot's position, orientation, and velocity.

The navigation strategy used is based on the RRT (Rapidly Exploring Random Trees) algorithm. The RRT algorithm is a widely used motion planning algorithm that is used to plan the trajectory of a robot from its current position to a goal position while avoiding obstacles. The algorithm is known for its efficiency, scalability, and ability to handle high-dimensional spaces. The RRT algorithm works by building a tree-like data structure that explores the configuration space of the robot.

The RRT algorithm begins with an initial configuration of the robot and builds a tree-like data structure by iteratively adding new nodes to the tree. Each new node is randomly sampled from the configuration space of the robot, and a path is constructed from the nearest node in the tree to the new node. The path is constructed in a way that ensures that it avoids obstacles and satisfies any other constraints that are imposed on the robot.

To ensure that the RRT algorithm explores the configuration space of the robot in a balanced manner, a bias is introduced that favors the exploration of unexplored regions of the configuration space. This is achieved by introducing a probability parameter that controls the likelihood of selecting a new node from an unexplored region of the configuration space.

The RRT algorithm continues to build the tree-like data structure until a node is added that is within a specified distance

from the goal configuration of the robot. The algorithm then constructs a path from the initial configuration of the robot to the goal configuration by tracing the path from the nearest node to the goal configuration through the tree. The pseudocode of Algorithm 1 shows the RRT strategy implemented in our setup.

Algorithm 1 High-Level Pseudocode for RRT Algorithm

```
# Define the start and goal points
start_point = current_position
goal_point = target_point

# Initialize the RRT tree
rrt_tree = start_point: None

# Set the maximum number of iterations and the step size
max_iterations = 1000
step_size = 0.1

# Iterate until the maximum number of iterations

for i do in range(max_iterations):
    # Generate a random point
    random_point = generate_random_point()

    # Find nearest point in tree
    nearest_point = find_nearest_point(rrt_tree, random_point)

    # Steer towards random point from nearest point
    new_point = steer(nearest_point, random_point, step_size)

    # Check for collisions

    if i thens_collision_free(nearest_point, new_point):
        # Add the new point to the tree
        rrt_tree[new_point] = nearest_point

        # Check if the goal is reached

        if i thens_goal_reached(new_point, goal_point):
            # Generate the path from start to goal
            path = generate_path(rrt_tree, start_point, goal_point)
            break
        end if
    end if
end for

# Follow the path using motor encoders and gyroscope
follow_path(path)
```

The robot is equipped with sensors necessary to both explore the unknown environment (observable environment), define its position, and navigate using the RRT algorithm. The robot has a GPS module and an inertial measurement unit (IMU) to accurately determine its position and a distance sensor to detect obstacles and avoid collisions.

To begin with, the start and goal points of the exploration are defined. The start point is the robot's current position, and the goal point is the target location that the robot should reach. An RRT tree is then initialized, which will represent the possible paths the robot can take.

The maximum number of iterations and the step size are set. The robot will iteratively execute the algorithm a specified number of times to generate new paths to explore the environment. In each iteration, a random point is generated, representing a new direction for the robot to explore.

The nearest point in the RRT tree to the random point is found. This point acts as a reference point, and the robot then steers towards the random point from this reference point using the step size. The resulting new point is checked for collisions to ensure that it is safe for the robot to move to this location.

If the new point is collision-free, it is added to the RRT tree as a new node with the nearest point as its parent. The algorithm checks if the goal point has been reached by the robot. If it has, the path from the start point to the goal point is generated using the RRT tree.

In our study, we approximated the uniform sampling of the environment by implementing a grid-based approach. By dividing the environment into equally sized cells, we ensured that each cell had the same probability of being selected. To achieve this, we generated random points within each cell and used these points as samples for the path planning algorithm. This allowed us to approximate a uniform distribution of samples throughout the environment while considering the ratio between the number of samples inside and outside the area.

To predict the robot's state at a future time horizon (e.g., $t + k$), we employed a recursive estimation method. At each time step, the robot's estimated state was updated based on the current measurements and the most recent prediction. This process was repeated at every time step, allowing the algorithm to propagate the estimated state through the prediction horizon. By incorporating this recursive estimation into the path planning algorithm, we were able to generate more accurate and reliable predictions of the robot's position and orientation, leading to improved navigation performance in uncertain environments.

Finally, the robot follows the generated path using its motor encoders and gyroscope to achieve precise control of its movement and orientation. This ensures that the robot moves smoothly and avoids collisions while exploring the environment. The camera sensor is used to provide visual information to the microcontroller, which is used for object recognition, in particular, to identify the target location. The goal is to use the camera sensor on the robot to recognize a red circle as the target point and navigate the robot towards it. To achieve this, the code uses a simple image processing algorithm that scans the camera feed for the presence of a red circle.

The first step in the algorithm is to capture a frame from the camera and apply some basic image processing filters to remove noise and enhance the contrast of the image. In this case, we use the built-in OpenCV library to apply a Gaussian blur filter to smooth out the image and a color threshold filter to extract the red color channel.

Once we have a processed image, we can use OpenCV's HoughCircles function to detect circular shapes in the image.

This function takes several parameters, including the minimum and maximum radii of the circles to detect, the minimum distance between detected circles, and the minimum threshold for circle detection. In our case, we set these parameters to detect circles with a radius between 10 and 30 pixels, at a minimum distance of 50 pixels from each other, and with a minimum threshold of 50.

Assuming we detect a circle in the image, we can then calculate its centroid (the center point of the circle) and use this as the target point for the robot. We use the formulae for the centroid of a circle that is given in the OpenCV documentation. The pseudocode of Algorithm 2 shows the camera and navigation strategy implemented in our setup.

Algorithm 2 High-Level Pseudocode for Camera and Navigation

```
Import required libraries
Define constants for target color and size thresholds, maximum and minimum speed, maximum turn rate, and PID constants KP, KI, and KD
Initialize the BNO055 sensor and servo motors
Initialize variables for target coordinates and PID control
Setup function:
    Begin serial communication
    Attach servo motors
    Set PID output limits and mode to automatic
    Start the BNO055 sensor
Loop function:
    Capture a camera frame and detect the target
    If target is not detected:
        Generate a path using the RRT algorithm
    Else:
        Update path to go directly to the target
    Follow the path using PID control
    Adjust the robot's speed and heading based on the error
FollowPath function:
    Calculate current heading and error
     in heading and speed
    Calculate the new motor setpoint using PID control
    Call the turnRobot function with the heading error
turnRobot function:
    Calculate turn rate based on heading error
     and constrain it to the maximum turn rate
    Calculate left and right motor speeds based
     on the turn rate and motor setpoint
    Write the left and right motor speeds to the servo motors
```

With the target point identified, we can use a simple proportional control algorithm to adjust the robot's movement based on the distance and angle to the target point. Specifically, we calculate the difference between the robot's current heading (determined by the IMU sensor) and the angle to the target point (determined by the centroid coordinates) and use this as the turning angle. We also calculate the distance to the target point (using the Pythagorean theorem) and use this as the forward speed of the robot. We then set the speed of each motor based on these calculated values.

The code also includes some basic error handling and recovery mechanisms to deal with unexpected situations. For example, if the camera fails to detect the target point or the robot gets stuck, we set the motors to rotate in opposite directions to try to free the robot from its current position.

IV. Results

To visualize the trajectory of the robot, we created a small simulator in Python that with the help of Matplotlib manages to visually replicate the operating conditions of our robot in the laboratory (3 $\times$ 2 meters environment, Fig. 3). The map is generated directly with Python.

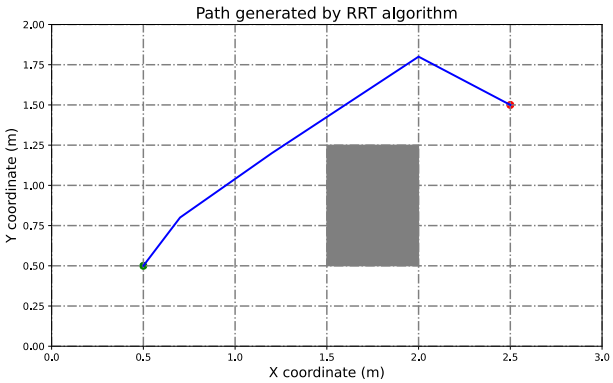

Fig. 3. Simulation environment. The map shows the robotics laboratory used in the performance tests. Target point (red), simulation path (blue), robot starting point (green), and obstacle (gray).

This code creates a 2D array to represent the environment, sets an obstacle in a random position, and randomly sets the robot's starting and target positions. The RRT algorithm is implemented in this code as a function to generate the path from the robot's starting position to the target position, avoiding the obstacle. Once the path is generated, the robot's movement is simulated by updating its position along the path. Finally, the environment and the robot's path are visualized using Matplotlib.

Note that the robot's size is set to 22 $\times$ 22 cm, but the environment's dimensions are in meters. To account for this, the program multiplies the environment dimensions and the positions by 100.

In this study, the navigation strategy of the SERB robot was evaluated in both simulated and real-world environments. To replicate the simulation conditions, a laptop was used to set the target position and monitor progress through sensors transmitted by the ESP32 via a local area network (LAN). A map was constructed and used to visualize the robot's position and calculate corresponding movements in the simulator for comparison. The implementation and performance of the navigation strategy were also evaluated in a real-world environment by locating the robot and landmark at the target point (Fig. 4).

Results showed that the robot's behavior in the simulator was highly similar to its navigation in the real environment, with position errors less than 6% with respect to the ideal position determined by the simulator. This error rate falls within the range of position sensor error. Additionally, better obstacle avoidance was observed in the real world than during the simulation, which was attributed to adjustments made to the LiDAR sensor.

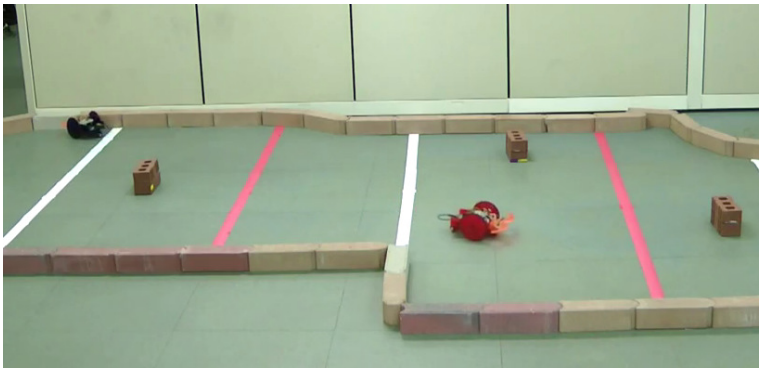

Fig. 4. Online simulation and testing with the robot in real environments. A new target is set and the robot is configured in the environment, as well as the target point. The map is built with python recreating obstacles and boundaries, and is used to visualize the position and calculate the differences between simulation and real prototype.

During experiments in the real world, only one type of collision was observed. The robot's wheels touched the obstacle several times, altering the position information. This occurred when the obstacle was obstructed by the robot's structure, and therefore outside the laser scanner's exploration plane. While visible to the camera, the camera was only programmed to detect the landmark at the target point. It is presumed that adjustments to the LiDAR or feedback control from the camera could eliminate this problem.

The findings of this study suggest that the navigation strategy implemented on the SERB robot performs well in both simulated and real-world environments, with only minor differences between the two. Furthermore, the study highlights the importance of careful consideration when using sensors to detect obstacles, as issues with the robot's structure can lead to inaccuracies in position information. Future research may explore additional methods for obstacle detection and feedback control to address this issue.

In order to enhance the performance of the navigation strategy and account for uncertainty and randomness, Probabilistic Road Maps (PRMs) can be incorporated into the technique. PRMs are particularly well-suited to handle uncertain environments, as they generate a probabilistic representation of the environment, factoring in the likelihood of obstacles and other uncertainties. By incorporating PRMs, the robot's navigation strategy can be more adaptable to unforeseen variations in the environment, leading to more robust and reliable performance.

To integrate PRMs, the existing RRT algorithm can be extended to include a probabilistic sampling of the environment, which would account for uncertainties in obstacle positions and robot's starting and target positions. This would allow the algorithm to generate multiple potential paths, each associated with a certain level of confidence based on the probability distribution of obstacles and positions. By selecting the path with the highest confidence level, the robot's navigation strategy can better account for uncertainties and improve its overall performance in complex and dynamic environments.

Moreover, the integration of PRMs can also help address the issue of the robot's structure obstructing the obstacle detection. By including a probabilistic model of the robot's structure, the algorithm can better predict the likelihood of an obstruction and adjust its path planning accordingly. This, in combination with improvements to the LiDAR sensor and feedback control from the camera, could further enhance the robot's navigation strategy and increase its performance in both simulated and real-world environments.

V. Discussion

The results of the experiments showed that the Rapidly Exploring Random Trees (RRT) algorithm implemented on the ESP32 microcontroller provides high performance in terms of autonomous navigation in complex environments. The robot was able to generate optimal paths from the origin to the destination point while avoiding obstacles and adjusting its motion accordingly [39]. The robot's motion was smooth, and it was able to reach the destination point accurately and efficiently.

The ESP32 microcontroller's processing power and memory were found to be sufficient for implementing the RRT algorithm on the small differential robot with two DC motors [40]. The algorithm was able to quickly generate feasible paths in complex environments with numerous obstacles. The performance of the algorithm was not significantly affected by the number or complexity of the obstacles, as the algorithm was able to efficiently navigate through the environment in all cases.

The experimental results also showed that the robot's motion was smooth and stable, even when navigating through narrow passages or around sharp turns. The robot was able to adjust its motion and avoid collisions in real-time, demonstrating the efficiency and reliability of the algorithm.

In addition to the robot's motion, the performance of the robot's sensors was also evaluated. The robot's position sensor was found to provide accurate and reliable estimates of the robot's position, allowing the algorithm to generate optimal paths in real-time. The distance sensor was also found to be reliable, providing accurate measurements of the distance between the robot and nearby obstacles.

The motor encoder and gyroscope were used to measure the robot's motion and adjust its heading accordingly. The motor encoder was found to provide accurate measurements of the

robot's speed and direction, allowing the algorithm to adjust the robot's speed and heading in real-time. The gyroscope was also found to be reliable, providing accurate measurements of the robot's orientation.

VI. Conclusion

In conclusion, the results obtained through simulations and experiments demonstrate the high performance and reliability of the Rapidly Exploring Random Trees algorithm implemented on a small differential robot controlled by an ESP32 microcontroller. The use of the ESP32 microcontroller proved to be highly beneficial, as it provided enough processing power and memory to perform the complex calculations required by the algorithm, enabling the robot to navigate through complex environments with high efficiency and reliability.

The successful implementation of the algorithm on the robotic platform allowed the robot to autonomously navigate from an origin point to a destination point in an unknown but observable environment while avoiding obstacles and adjusting its motion accordingly. The robot's behavior with this algorithm was highly desirable, as it was able to move through the environment efficiently and reliably, while avoiding obstacles and reaching its destination.

The use of a low-cost, versatile and efficient robotic platform for autonomous navigation in complex environments is highly beneficial and can have many practical applications such as search and rescue, surveillance, and industrial automation. The practical and cost-effective solution proposed in this work can be easily replicated and implemented in various scenarios, providing a highly reliable and efficient navigation solution.

However, there is still room for improvement in terms of the design and placement of sensors on the robot to optimize the performance of the algorithm. Further research could focus on the development of more advanced algorithms that could improve the robot's navigation performance in complex environments.

Acknowledgment

This work was supported by the Universidad Distrital Francisco José de Caldas, specifically by the Technological Faculty. The views expressed in this paper are not necessarily endorsed by Universidad Distrital. The authors thank all the students and researchers of the research group ARMOS for their support in the development of this work.

References

[1] N. Nedjah and L. S. Junior, "Review of methodologies and tasks in swarm robotics towards standardization," *Swarm and Evolutionary Computation*, vol. 50, no. 1, p. 100565, 2019.

[2] A. Zacharaki, I. Kostavelis, A. Gasteratos, and I. Dokas, "Safety bounds in human robot interaction: A survey," *Safety Science*, vol. 127, no. 1, p. 104667, 2020.

[3] M. Tang, Y. Gu, S. Wang, and Q. Liang, "DCBot: An autonomous hotline working robot for 110 kV substation," *Robotics and Autonomous Systems*, vol. 119, no. 1, pp. 247–262, 2019.

[4] F. Martínez, H. Montiel, and H. Valderrama, "Using embedded robotic platform and problem-based learning for engineering education," in *Smart Education and e-Learning 2016*. Springer International Publishing, 2016, pp. 435–445.

[5] F. Fahmizal, A. Priyatmoko, E. Apriaskar, and A. Mayub, "Heading control on differential drive wheeled mobile robot with odometry for tracking problem," in *2019 International Conference on Advanced Mechatronics, Intelligent Manufacture and Industrial Automation (ICAMIMIA)*. IEEE, 2019.

[6] H. Xie, J. Zheng, M. Wang, and R. Chai, "Networked DC motor control with time-varying delays and application to a mobile robot," in *2020 IEEE 16th International Conference on Control and Automation (ICCA)*. IEEE, 2020.

[7] A. S. Shekhawat and Y. Rohilla, "Design and control of two-wheeled self-balancing robot using arduino," in *2020 International Conference on Smart Electronics and Communication (ICOSEC)*. IEEE, 2020.

[8] J. de Almeida, R. Taizo, F. Neves-Jr, and L. V. R. de Arruda, "A global/local path planner for multi-robot systems with uncertain robot localization," *Journal of Intelligent and Robotic Systems*, vol. 100, no. 1, pp. 311–333, 2020.

[9] J.-Y. Jhang, C.-J. Lin, and K.-Y. Young, "Cooperative carrying control for multi-evolutionary mobile robots in unknown environments," *Electronics*, vol. 8, no. 3, p. 298, 2019.

[10] H. Surmann, C. Jestel, R. Marchel, F. Musberg, H. Elhadj, and M. Ardani, "Deep reinforcement learning for real autonomous mobile robot navigation in indoor environments," *arXiv*, 2020.

[11] K. Qian, Y. Liu, L. Tian, and J. Bao, "Robot path planning optimization method based on heuristic multi-directional rapidly-exploring tree," *Computers and Electrical Engineering*, vol. 85, no. 1, p. 106688, 2020.

[12] C. Yuan, W. Zhang, G. Liu, X. Pan, and X. Liu, "A heuristic rapidly-exploring random trees method for manipulator motion planning," *IEEE Access*, vol. 8, no. 1, pp. 900–910, 2020.

[13] T. Rybus, "Point-to-point motion planning of a free-floating space manipulator using the rapidly-exploring random trees (RRT) method," *Robotica*, vol. 38, no. 6, pp. 957–982, 2019.

[14] Y. Li, W. Wei, Y. Gao, D. Wang, and Z. Fan, "PQ-RRT*: An improved path planning algorithm for mobile robots," *Expert Systems with Applications*, vol. 152, no. 1, p. 113425, 2020.

[15] W. Xinyu, L. Xiaojuan, G. Yong, S. Jiadong, and W. Rui, "Bidirectional potential guided RRT* for motion planning," *IEEE Access*, vol. 7, no. 1, pp. 95 046–95 057, 2019.

[16] L. Marin, "Modular open hardware omnidirectional platform for mobile robot research," in *2018 IEEE 2nd Colombian Conference on Robotics and Automation (CCRA)*. IEEE, 2018.

[17] A. A. Rodriguez, K. Puttannaiah, Z. Lin, J. Aldaco, Z. Li, X. Lu, K. Mondal, S. D. Sonawani, N. Ravishankar, N. Das, and P. A. Pradhan, "Modeling, design and control of low-cost differential-drive robotic ground vehicles: Part i — single vehicle study," in *2017 IEEE Conference on Control Technology and Applications (CCTA)*. IEEE, 2017.

[18] F. Martínez, "Turtlebot3 robot operation for navigation applications using ros," *Tekhnê*, vol. 18, no. 2, pp. 19–24, 2021.

[19] F. Martínez, C. Hernández, and A. Rendón, "A study on machine learning models for convergence time predictions in reactive navigation strategies," *Contemporary Engineering Sciences*, vol. 10, no. 25, pp. 1223–1232, 2017.

[20] F. Martínez, F. Martínez, and E. Jacinto, "Visual identification and similarity measures used for on-line motion planning of autonomous robots in unknown environments," in *Eighth International Conference on Graphic and Image Processing (ICGIP 2016)*, 2016, pp. 321–325.

[21] S. M. Trenkwalder, "Computational resources of miniature robots: Classification and implications," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2722–2729, 2019.

[22] G. Croon, J. Dupeyroux, S. Fuller, and J. Marshall, "Insect-inspired AI for autonomous robots," *Science Robotics*, vol. 7, no. 67, pp. 1–10, 2022.

[23] H. Yuk, D. Kim, H. Lee, S. Jo, and J. H. Shin, "Shape memory alloy-based small crawling robots inspired by c. elegans," *Bioinspiration and Biomimetics*, vol. 6, no. 4, p. 046002, 2011.

[24] S. Balasooriya, I. Kavalchuk, and E. Dimla, "Innovative path planning algorithm for an autonomous robot with low computational cost," in *2019 International Conference on Advanced Computing and Applications (ACOMP)*. IEEE, 2019.

[25] S. M. Neuman, B. Plancher, B. P. Duisterhof, S. Krishnan, C. Banbury, M. Mazumder, S. Prakash, J. Jabbour, A. Faust, G. C. de Croon, and V. J. Reddi, "Tiny robot learning: Challenges and directions for machine learning in resource-constrained robots," in *2022 IEEE 4th International Conference on Artificial Intelligence Circuits and Systems (AICAS)*. IEEE, 2022.

[26] M. Benyeogor, K. Nnoli, O. Olakanmi, O. Lawal, E. Gratton, S. Kumar, K. Akpado, and P. Saha, "An algorithmic approach to adapting edge-based devices for autonomous robotic navigation," *EAI Endorsed Transactions on Context-aware Systems and Applications*, vol. 237011885, no. 1, p. 170559, 2018.

[27] B. M. Vukelic, R. Stancic, and S. G. Graovac, "Microcontroller based implementation of an integrated navigation system for ground vehicles," *IFAC Proceedings Volumes*, vol. 46, no. 25, pp. 139–144, 2013.

[28] P. Dudek, T. Richardson, L. Bose, S. Carey, J. Chen, C. Greatwood, Y. Liu, and W. Mayol-Cuevas, "Sensor-level computer vision with pixel processor arrays for agile robots," *Science Robotics*, vol. 7, no. 67, pp. 1–10, 2022.

[29] C. Fulgenzi, C. Tay, A. Spalanzani, and C. Laugier, "Probabilistic navigation in dynamic environment using rapidly-exploring random trees and gaussian processes," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008.

[30] J. Leonard, J. How, S. Teller, M. Berger, S. Campbell, G. Fiore, L. Fletcher, E. Frazzoli, A. Huang, S. Karaman, O. Koch, Y. Kuwata, D. Moore, E. Olson, S. Peters, J. Teo, R. Truax, M. Walter, D. Barrett, A. Epstein, K. Maheloni, K. Moyer, T. Jones, R. Buckley, M. Antone, R. Galejs, S. Krishnamurthy, and J. Williams, "A perception-driven autonomous urban vehicle," in *Springer Tracts in Advanced Robotics*. Springer Berlin Heidelberg, 2009, pp. 163–230.

[31] M. D. P. Moratuwage, W. S. Wijesoma, B. Kalyan, N. M. Patrikalakis, and P. Moghadam, "Collaborative multi-vehicle localization and mapping in high clutter environments," in *2010 11th International Conference on Control Automation Robotics and Vision*. IEEE, 2010.

[32] W. G. Aguilar, S. Morales, H. Ruiz, and V. Abad, "RRT* GL based optimal path planning for real-time navigation of UAVs," in *Advances in Computational Intelligence*. Springer International Publishing, 2017, pp. 585–595.

[33] C. Wang, L. Meng, S. She, I. M. Mitchell, T. Li, F. Tung, W. Wan, M. Q. H. Meng, and C. W. de Silva, "Autonomous mobile robot navigation in uneven and unstructured indoor environments," *arXiv*, pp. 1–8, 2017.

[34] C. E. Tuncali and G. Fainekos, "Rapidly-exploring random trees for testing automated vehicles," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2019.

[35] B. B. K. Ayawli, X. Mei, M. Shen, A. Y. Appiah, and F. Kyeremeh, "Optimized RRT-a* path planning method for mobile robots in partially known environment," *Information Technology And Control*, vol. 48, no. 2, pp. 179–194, 2019.

[36] L. Zhang, Z. Lin, J. Wang, and B. He, "Rapidly-exploring random trees multi-robot map exploration under optimization framework," *Robotics and Autonomous Systems*, vol. 131, no. 1, p. 103565, 2020.

[37] S. Bak, J. Betz, A. Chawla, H. Zheng, and R. Mangharam, "Stress testing autonomous racing overtake maneuvers with rrt," *arXiv*, 2021.

[38] Oomlout. (2023) Arduino controlled servo robot (serb). Instructables. [Online]. Available: https://www.instructables.com/How-to-Make-an-Arduino-Controlled-Servo-Robot-SER/

[39] F. Martínez, F. Martínez, and H. Montiel, "Hybrid free-obstacle path planning algorithm using image processing and geometric techniques," *ARPN Journal of Engineering and Applied Sciences*, vol. 14, no. 18, pp. 3135–3139, 2019.

[40] C. Li, C. Wang, J. Wang, Y. Shen, and M. Q.-H. Meng, "Sliding-window informed RRT*: A method for speeding up the optimization and path smoothing," in *2021 IEEE International Conference on Real-time Computing and Robotics (RCAR)*. IEEE, 2021.