machines

MDPI

Article

Design and Development of an Autonomous Mobile Robot for Unstructured Indoor Environments

Ameur Gargouri [1,2], Mohamed Karray [1,*], Bechir Zalila [3] and Mohamed Ksantini [2]

[1] ESME Research Lab, 94200 Ivry Sur Seine, France; gargouri.ameur@enis.tn
[2] ATISP Laboratory, ENETCom, University of Sfax, Sfax 3029, Tunisia; mohamed.ksantini@ipeis.usf.tn
[3] ReDCAD Laboratory, ENIS, University of Sfax, Sfax 3029, Tunisia; bechir.zalila@enis.tn
* Correspondence: mohamed.karray@esme.fr

Abstract

This research work presents the design and the development of a cost-effective autonomous mobile robot for locating misplaced objects within unstructured indoor environments. The tools integrated into the proposed system for perception and localization are a hardware architecture equipped with LiDAR, an inertial measurement unit (IMU), and wheel encoders. The system also includes an ROS2-based software stack enabling autonomous navigation via the NAV2 framework and Adaptive Monte Carlo Localization (AMCL). For real-time object detection, a lightweight YOLO11n model is developed and implemented on a Raspberry Pi 4 to enable the robot to identify common household items. The robot's motion control is achieved by a fuzzy logic-enhanced PID controller that dynamically modifies gain values based on navigation conditions. Remote supervision, task management, and real-time status monitoring are provided by a user-friendly Flutter-based mobile application. Simulations and real-world experiments demonstrate the robustness, modularity, and responsiveness of the robot in dynamic environments. This robot achieves a 3 cm localization error and a 95% task execution success rate.

Keywords: autonomous mobile robot; ROS2; YOLO; SLAM; fuzzy logic PID; indoor navigation

check for updates

Academic Editor: Raul D. S. G. Campilho

Received: 15 October 2025
Revised: 6 November 2025
Accepted: 7 November 2025
Published: 12 November 2025

Citation: Gargouri, A.; Karray, M.; Zalila, B.; Ksantini, M. Design and Development of an Autonomous Mobile Robot for Unstructured Indoor Environments. *Machines* **2025**, *13*, 1044. https://doi.org/10.3390/machines13111044

Copyright: © 2025 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

1. Introduction

The last 10 years have witnessed the emergence of autonomous mobile robots (AMRs) as a pivotal technology across domestic, logistic, and industrial fields because of their capabilities of operating without dedicated infrastructure and performing tasks such as navigation, detection, and manipulation autonomously [1]. The rising demand for robotic service applications in hospitals, warehouses, smart manufacturing, and smart homes, where they enhance efficiency and reduce human intervention, has increased their adoption substantially [2,3]. Current autonomous mobile robots (AMRs) typically adopt a modular architecture that possesses three basic components: perception, localization, and navigation. Environmental perception is achieved by using sensors such as light detection and ranging (LiDAR), inertial measurement units (IMUs) and cameras. Simultaneous Localization and Mapping (SLAM) algorithms are also applied in order to construct environmental maps and localize the robot at specific locations within them. In navigation, global and local path planning algorithms such as A*, employed in global path planning, and the dynamic window approach (DWA), applied in local obstacle avoidance, guarantee smooth and adaptive mobility. The latest improvements in the field, such as the optimized integration of A* with DWA in Sánchez-Ibáñez et al. [4], Ishigami et al. [5] works, have aided in enhancing

path efficiency and reducing the computational overhead on embedded platforms, enabling real-time performance in complex environments.

In spite of these advancements, AMRs still encounter some flaws that restrict their widespread adoption. Because the computational performance of real-time SLAM and perception algorithms usually cannot compete with that of resource-constrained embedded systems, it is necessary to balance execution speed and precision. At a more basic level, it is also imperative to consider robustness in dynamic, unstructured environments, because it is typical for SLAM and navigating platforms to be initially deployed under static or semi-static conditions, with proportionally dropping performance in reaction to dynamic obstacles or varying lighting conditions. When deep learning models such as YOLO models are included, the computational demands are further increased, particularly in low-power hardware such as the Raspberry Pi. Moreover, achieving seamless human–robot interaction in household settings requires the consideration of user needs, as well as providing intuitive interfaces and ensuring that task execution capabilities are reliable, where current systems often fall short due to limited semantic understanding and adaptability. Finally, most AMRs are far from being cost-effective, which further limits the number of users who can afford them or use them in everyday environments.

This paper addresses these difficulties as it presents a modular AMR that is cost-effective and designed for unstructured indoor environments and is characterized by high-precision navigation and robust object localization. The system integrates a lightweight YOLO11n model for real-time object detection with an ROS2-based navigation stack using Cartographer and AMCL for SLAM and localization, complemented by a custom grid-based search algorithm. Both simulated and actual testing confirms the effectiveness and precision of the AMR. User interaction, including real-time monitoring and task management, is enhanced through the integration of a Flutter-based mobile application. To offer insights into the optimization of embedded systems for domestic robotics applications, this work critically assesses the trade-offs between computational efficiency and performance.

The remainder of this paper is organized as follows. Section 2 reviews related works on autonomous mobile robots, briefly summarizing the current designs, functions, and limitations. Section 3 gives an outline of the designed robot architecture, the hardware components, and the software platform, with an emphasis placed upon normal ROS2-based components as well as hand-designed parts. Section 4 presents the methodology, covering the modeling process, localization and path planning strategies, object detection module, custom searching strategy, actuator control by both classic PID and fuzzy logic-improved mechanisms and the development of a Flutter-based application that allows users to interact with the system and complete tasks. Section 5 presents the results and discussion, including simulation experiments, real-world experiments, and comparisons with other robotic platforms. Finally, Section 6 concludes this paper by summarizing key contributions as well as future research avenues.

2. Related Works

This work analyzes the performance and limitations of autonomous mobile robots (AMRs) available on the market that are designed for indoor navigation and object localization, characterized by their system architectures.

Singh et al. [6] tested the TurtleBot3, developed by Robotis [7], is an open-source AMR widely used in academic and research settings for indoor navigation tasks, such as basic object detection, path planning, and mapping. Equipped with a 360-degree LiDAR, a Raspberry Pi 4, and optional cameras, it leverages ROS2 for sensor integration and navigation, making it suitable for prototyping and education, as it is a relatively low-cost platform (approximately USD 500–700). Although this low-cost hardware has a modular

design and can achieve tasks like SLAM and obstacle avoidance in small-scale indoor environments, it suffers from substantial limitations. For example, the Raspberry Pi creates navigation delays of up to 1 s, resulting from its limited processing power when it comes to complex tasks like real-time SLAM or visual detection in dynamic settings. Furthermore, its standard sensor set suffers in low-feature scenes (e.g., feature-poor corridors), leading to localization inaccuracies of up to 0.5 m. Another significant weakness lies in the short time of operation based on its low-rated battery life, which does not surpass 1.5 h.

The Clearpath Jackal, developed by Clearpath Robotics [8], is a high-power AMR for research as well as industry applications, has the capabilities of both indoor and outdoor navigation through high-end onboard inertial-aided LiDAR, as well as an onboard Intel NUC microcomputer. Its localization errors are less than 0.1 m in structured environments due to the utilization of ROS2-based high-resolution 2D mapping SLAM. It offers reliable navigation under controlled indoor conditions, e.g., labs or warehouses, thanks to the device's robust construction as well as powerful hardware. However, its accessibility for domestic or small-scale applications is limited because of its high cost (approximately USD 10,000). Drift errors of 0.4 m or more occur when using the LiDAR-based SLAM in feature-scarce environments like long highways. Moreover, the Jackal consumes excessive power, which reduces its battery life to 2 h under heavy loads, and its bulky design makes it less suitable for navigation in tight indoor spaces [9].

The NVIDIA JetBot, developed by NVIDIA Corporation [10] is a low-cost, school- and hobbyist-oriented AMR that utilizes a Jetson Nano to enable GPU-assisted object detection and YOLO algorithms to execute functions such as tracking objects and navigating. The AMR is cost-effective because of its perception-based cameras and its low price (around USD 250). The JetBot achieves object detection accuracies of 80–85% for household items, with latencies of around 1 s in simple environments. However, when it operates in cluttered settings, its inference rates are limited to 0.5 Hz, rendering operation unstable at higher speeds because its processing power is limited. Because the JetBot relies on vision-based detection, its accuracy is reduced to (50–70%), according to Ramesh et al. [11], Kawakura and Shibasaki [12] works, in unstructured indoor spaces with varying lighting or occlusion. Moreover, its practical deployment is constrained due to its small battery capacity, which limits operation to under 1 h.

The Pozyx UWB-based AMR developed by Pozyx [13] uses ultra-wideband (UWB) positioning technology to precisely locate objects indoors, with its location accuracy ranging from 10 to 30 cm according to Crețu-Sîrcu et al. [14] work. This makes it useful for industrial activities like tracking inventories. Implemented as an ROS-integrated unit, it also accommodates sensor fusion, including with sensors such as LiDAR and IMUs, used for guidance under structured conditions. However, as Pozyx's AMR relies on pre-installed UWB beacons, it is costly and complex to set up and therefore unsuitable for domestic environments that do not have specialized infrastructure. Its localization performance becomes low in environments with metallic obstructions or multipath interference, where the errors rise to 0.3 m. The system has limited scalability for cost-sensitive applications like smart homes due to its high cost (nearly USD 5000 for a complete setup) and dependency on external hardware.

The MIR100, developed by Mobile Industrial Robots, tested by Güldenring et al. [15] and Soheilifar [16], is a commercial AMR that is designed for warehouse and industrial navigation and uses ROS-based navigation with A* for global planning and DWA for local obstacle avoidance. Equipped with LiDAR and cameras, it is able to perform tasks like material transport. However, because of the MIR100's high cost (approximately USD 20,000), its use is limited to industrial settings, making it inaccessible for domestic applications. The robot's utilization drops to 58.56% when it navigates at a 1.5 m/s speed,

which leads to excessive idle time, often waiting for the next item to be available. Moreover, its high power consumption, caused by its heavy reliance on powerful onboard computers, reduces the operational time to 8 h under heavy loads, and it navigates with difficulty in tight spaces because of its large footprint.

3. Proposed Robot Architecture

The autonomous mobile robot is designed to locate lost objects in cluttered indoor environments through the use of a tightly coupled hardware–software system. It integrates a robust hardware foundation with a Raspberry Pi 4 (Sony UK Technology Centre in Pencoed, Wales), ESP32 microcontroller (Espressif Systems in Shanghai, China, at ZhangJiang Hi-Tech Park), and sensor suite running the ROS2 software (ROS 2 Humble Hawksbill available for Ubuntu Jammy (22.04)) to enable perception, movement, and human interaction. Figure 1 outlines the robot architecture as a whole and explicitly illustrates the interconnecting software/hardware components.

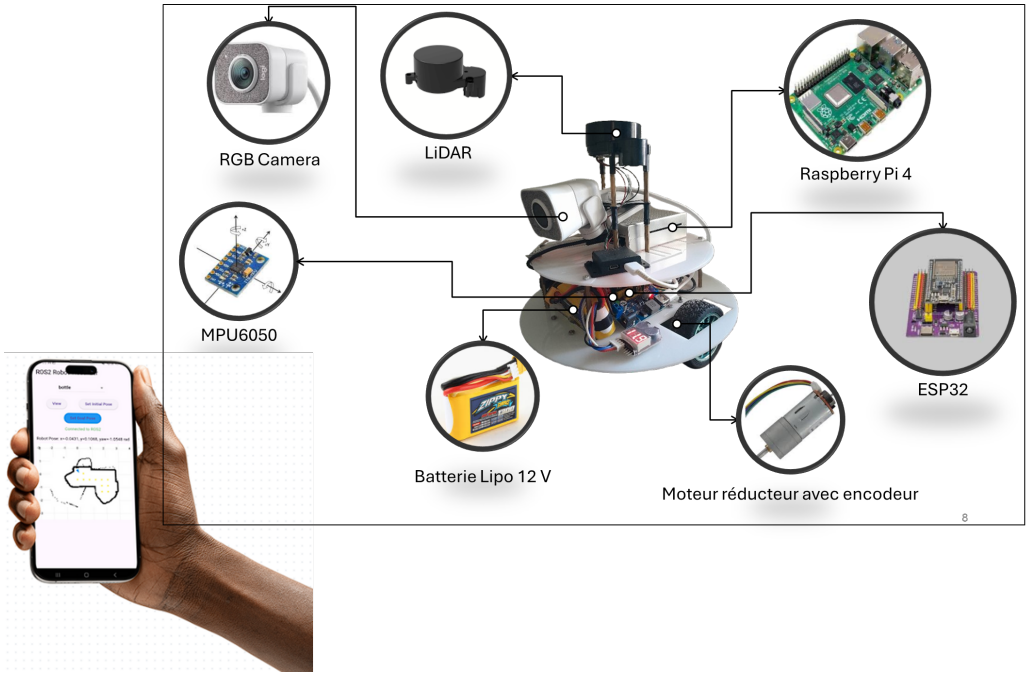


Figure 1. Robot architecture of the autonomous mobile robot.

3.1. Hardware Architecture

The system of an autonomous robot consists of a powerful collection of microcontrollers, sensors, actuators, and powering systems that allow it to achieve object localization and navigation effectively in unstructured indoor spaces. This work presents a detailed examination of all parts of the design, including their nature, interlinkages, and functions in the complete system.

Raspberry Pi 4 Model B Board: This is a quad-core Cortex-A72 (1.5 GHz) master microcontroller combined with a Broadcom VideoCore VI GPU. It runs with Ubuntu 22.04 with ROS2 Humble and is responsible for high-level processing tasks including sensor data fusion, SLAM, NAV2 navigation, and YOLO11n object detection. It communicates through the universal serial bus (USB) and the universal asynchronous receiver/transmitter (UART), processes data in real time, and runs ROS2 middleware to execute communication among nodes based on multicore processing to complete tasks in parallel.

ESP32 Microcontroller: The auxiliary microcontroller is an ESP32-WROOM-32, which has a dual-core Tensilica LX6 CPU (240 MHz) as well as 520 KB SRAM. It schedules real-time tasks through the Free Real-Time Operating System (FreeRTOS), as well as managing

low-level interfacing such as motor interfacing and sensor interfacing. The ESP32 receives data from the wheel encoders, as well as from the IMU sensor; produces pulse width modulation (PWM) signals to facilitate motor interfacing, and interfaces with the Raspberry Pi via UART.

LiDAR (YDLIDAR TSA, Shenzhen EAI Technology Co., Ltd., Shenzhen, China): Developed by YDLIDAR [17] and used by Hakkim et al. [18], consists of an 8-m rangefinder and a 360-degree time-of-flight (ToF) 2D LiDAR. Its ranging distance ranges from 0.12 to 8 m (80% reflectivity), its scanning frequency ranges from 5 to 8 Hz, and its ranging frequency is 3000 Hz. Cartographer-based SLAM and collision avoidance depend on the high-accuracy environmental perception provided by the LiDAR's high angular resolution of up to 0.72 degrees.

Stream Cam: YOLO11n-based object classification and detection uses images captured by a Logitech StreamCam (Logitech, Model VU0054, made in China) with a 78° FOV, 1080 p, 60 fps, and auto-focus. It is mounted 12 cm above the chassis and is connected to the Raspberry Pi via USB. At 5 Hz, an ROS2 camera node publishes raw image messages. Despite changing indoor light intensities, the high frame rate and high-resolution camera enable strong visual perception.

IMU (MPU6050, InvenSense Inc., San Jose, CA, USA): A 6-DOF inertial measurement unit (IMU), used by Sultan et al. [19], consists of a 3-axis accelerometer and a 3-axis gyroscope, which record orientation and angular velocity data at 100 Hz. It is connected to the ESP32 via the I2C protocol and transmits data at 50 Hz to the Raspberry Pi, used for localization and attitude estimation. The precision of pose estimation in dynamic settings is improved thanks to the incorporation of data from the IMU with wheel encoders and LiDAR data via AMCL.

Wheel Encoders: These incremental magnetic encoders are placed on the right and left drive wheels to obtain odometry information with a resolution of 204 pulses per rotation. They are connected to the ESP32 through digital GPIO pins and output the wheel velocity and position, which are sent to the Raspberry Pi at a rate of 50 Hz through UART and fed into an ROS2 topic. This high-accuracy odometry greatly helps in the precise localization of differential drive vehicles.

DC Motors (JGA25-370 DC Gear Motor with Encoder 600RPM) and Motor Driver: When employing a differential drive system composed of two 12V DC, 600 RPM motors, the top speed achieved will be 0.5 m s^{-1} as long as the wheel diameter is 65 mm. An L298N dual H-bridge DC motor driver (STMicroelectronics, Plan-les-Ouates, Switzerland) is employed to drive the motor and is connected to the ESP32 through PWM signals that operate at a frequency of 5 kHz such that smooth control of the speed is facilitated. Control of the ESP32 through the driver allows the vehicle to operate smoothly and safely.

Power Supply System: The system is powered by a 12 V, 1300 mAh LiPo battery, which allows about 2 h of operation under nominal load conditions. We have employed a buck converter to give a regulated 5V supply to the Raspberry Pi and ESP32 such that an uninterrupted supply is ensured. A 1-8S LiPo battery voltage detector with a buzzer is also placed to monitor the power system and prevent possible damage of the battery due to situations of overdischarge or overcharge.

Chassis: The robot chassis is a 30-cm-diameter circular Plexiglass structure with a dual-layer configuration. The top level is covered by the top layer, placed at a 10 cm height to keep the camera and LiDAR above ground-level obstacles, with the highest coverage for the sensors. The bottom layer is covered with the motors, cell battery, and microcontrollers. A low center of gravity is maintained as a condition for stability during navigation. A low weight is also maintained throughout the design (about 1.4 kg).

3.2. Software Architecture

The hierarchical mobile robot software architecture is composed of the ROS2 Humble middleware running Ubuntu 22.04 and combines standard and homegrown frameworks to preserve this hierarchical organization. The standard frameworks are the ROS2 middleware, the navigation stack, SLAM and localization modules, and object detection and classification using YOLO11n. The homegrown modules include the fuzzy logic-enhanced PID motor control and search algorithm with a user-definable grid system. The Flutter-based user interface constitutes an additional custom-developed module. Figure 2 illustrates how these frameworks combine to maintain the robot's sensor perception, planning, and control functionalities with fast and real-time operation across the Raspberry Pi 4 and ESP32 platforms.

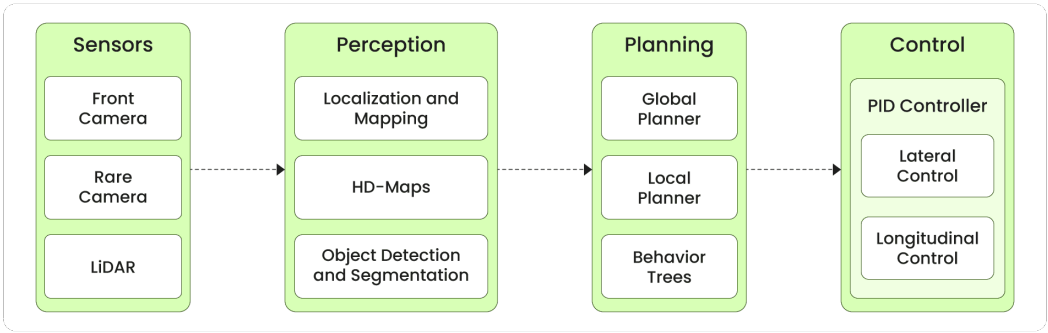


Figure 2. Hierarchy of the software architecture for AMRs.

3.2.1. Standard Frameworks

ROS2 Middleware: The core of this system is the ROS2 Humble framework, which enables communication among nodes across the Data Distribution Service (DDS) to facilitate publish–subscribe interactions among distributed nodes with minimal latency. Central nodes are navigation, sensor drivers, perception, and control, which are executed on the Raspberry Pi 4 within a real-time executor to allow deterministic running. The configuration of the ROS2 environment uses Quality of Service (QoS) policies that ensure the reliability of important information, such as sensor information, and the best-effort transmission of unimportant information, such as visualization, while ensuring fault-tolerant communications [20].

Navigation Stack: The NAV2 system, developed by Open Navigation LLC [21], is an integrated module within the ROS2 package that offers a global and a local planner to achieve safe navigation. While the global planner achieves optimal paths in a two-dimensional occupancy grid costmap of resolution 0.05 m, which is published at 5 Hz, through the use of the A* algorithm, the local planner utilizes the dynamic window approach (DWA) to evade obstacles dynamically while following kinematically the robot's differential drive parameters of a maximum linear speed of 0.5 m s^{-1} and an angular velocity limit of 1.0 rad s^{-1}. Furthermore, NAV2 integrates LiDAR data to detect and avoid moving obstacles dynamically and publishes navigation commands through the `/cmd_vel` topic [22].

SLAM and Localization: Cartographer, developed by Google [23], an ROS2-compatible SLAM program, generates 2D occupancy grid maps of resolution 0.05 m from LiDAR scans. To predict the pose in an accurate manner, another ROS2 package named Adaptive Monte Carlo Localization (AMCL) combines information from the LiDAR and wheel encoders along with the IMU [24].

Object Detection and Classification: YOLO11n, developed by Ultralytics [25] and tested by Cherubin Szymon and Michał [26], is a lightweight deep neural network that is trained with the COCO dataset and runs from TensorFlow Lite on the Raspberry Pi and accepts RGB images of size 128 × 128 from the camera to detect and label common objects in daily life. The ROS2 perception node (`yolo_detector`) subscribes to the `/camera/image_raw` topic and publishes detection results (the `/yolo/detections` message containing bounding box information along with class labels and confidence scores) to engage with the search algorithm.

3.2.2. Custom-Developed Components

Search Algorithm: A costmap-optimized grid-based search algorithm is programmed as an ROS2 node (search_node) and divides the environment into 0.5 m × 0.5 m cells to discretize and structure exploration and create waypoints. It is paired with NAV2 to provide navigation and YOLO11n to detect obstacles with time complexity of $O(n)$, where n is the number of waypoints. Failed points are re-explored three times through the action server of ROS2 to enhance navigation as it navigates through dynamically changing environments.

Motor Control: The ESP32 runs a fuzzy logic-enhanced PID controller with FreeRTOS tasks using C++. The controller is run at 50 Hz with wheel encoder feedback and provides PWM speed commands to the motors. Fuzzy logic rules with seven linguistic variables (e.g., Negative Big, Zero, Positive Small) for the error and derivative of error cause the PID gains to change dynamically to allow adaptable operation in changing circumstances.

User Interface: A Flutter-based mobile application, developed for the Android and iOS platforms, provides a user-friendly interface to control and monitor tasks. The app allows the visualization of maps in real time, the specification of navigation targets through touch feedback, and the initiation of searches for objects. The application interacts with ROS2 using the `rosbridge_websocket` node at 10 Hz with a low-latency update rate and with a JSON protocol to transfer commands and statuses.

The module structure was simulated and tested through Gazebo 11 and RViz 2 to ensure that all modules could run smoothly prior to field deployment.

4. Proposed Methodology

Robot development was carried out using the Agile Scrum methodology with the application of iteration cycles of design, development, and testing with short development sprints. Modeling was performed using URDF and Xacro in ROS2 with the realistic simulation of the chassis, sensors, and kinematics. The LiDAR-based SLAM and AMCL and the NAV2 stack were used for localization and navigation with dynamic path planning in real time. Object detection was performed with YOLO11n on a Raspberry Pi, performing object recognition of household items, while executing a grid-based search algorithm with systematic search. Motor control on the ESP32 used a fuzzy PID controller for precise speed regulation. A Flutter application was used for the real-time display of maps, setting goals, and object search control. Modular development was used so as to enable the iteration and integration of all subsystems with ease. Figure 3 illustrates the stepwise methodological workflow for the design, development, and realization of the proposed system.

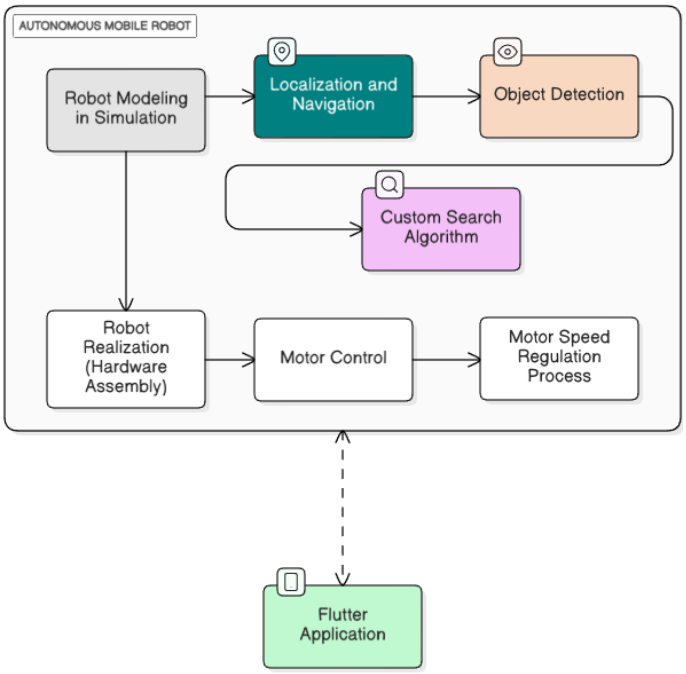


Figure 3. Robot design, development, and realization flowchart.

4.1. Robot Modeling

The definition of the robot's physical structure, as illustrated in Figure 4, employed the Unified Robot Description Format (URDF) and Xacro within ROS2 Humble, allowing a modular and parametric representation of the robot's components.

- **Links**: The two-layer, circular chassis, composed of lightweight Plexiglass, is 30 cm in diameter. Maneuverability and stability are provided by a spherical link of two drive wheels (6.2 cm in diameter and 2 cm in width) plus a caster wheel. Sensor connections have YDLiDAR mounts. Components include the TSA LiDAR, Logitech StreamCam, MPU6050 IMU sensor, and wheels.
- **Joints**: The components are mounted on the chassis through constant joints to allow rotation, powered by 12V DC motors of 600 RPM. We use fixed joints, securing the LiDAR system (after elevation at 10 cm above the vehicle body) and camera (at 12 cm above the chassis, frontward-facing), and the IMU is placed close to the center of the chassis for high-accuracy orientation measurement.
- **Sensors**: The YDLiDAR TSA LiDAR is configured with a 360° field of view and a 10 m range, outputting point cloud data through an ROS2 sensor plugin. The Logitech StreamCam, with a field of view of 78° and a resolution of 1080 p, is parameterized with a plugin for a camera to take pictures. The MPU6050 IMU produces output of 6 DOF (accelerometer and gyroscope) and is interfaced as an ROS2 plugin for the IMU. Wheel encoders, at 204 pulses/rev, are defined to produce odometry data via a differential drive plugin.
- **Coordinate Frames**: The transform tree has base_link as the origin of the vehicle, odom as the odometry tracker, and map as the global frame. Sensor-specific frames (lidar_link, camera_link, imu_link) are defined with offset with respect to base_link so that spatial transformation is correct, and the transformation is published at 50 Hz through tf2.

The differential drive system is modeled kinematically. The wheelbase is 25 cm and the maximum linear velocity is 0.5 m s^{-1} and is computed based on the 600 RPM motors and wheel diameter. Parameterization is achieved via Xacro macros, facilitating the easy modification of the dimensions and sensor locations. The model is also simulated dynam-

ically for correct collision and physics (e.g., friction, inertia) using Gazebo. RViz is also used for the visualization of the robot state, checking for correct transform propagation and alignment of the sensor state. URDF/Xacro files are also streamlined for interaction with the navigation and SLAM stacks of ROS2, ensuring seamless integration with Cartographer and NAV2.

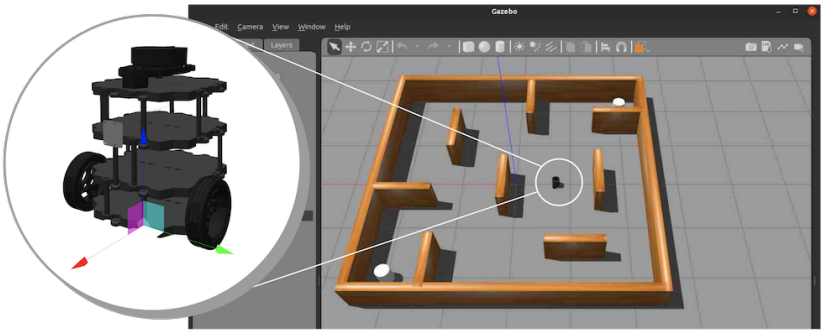


Figure 4. URDF model of the robot.

4.2. Localization and Navigation

The robot operates autonomously in unstructured indoor environments thanks to the Simultaneous Localization and Mapping (SLAM) and navigation system, while robot perception, localization, and path planning are achieved by leveraging ROS2 Humble's navigation stack. The ROS2 localization and navigation flowchart, shown in Figure 5, describes the sequential pipeline of data processing and decision-making in the robot's navigation system.

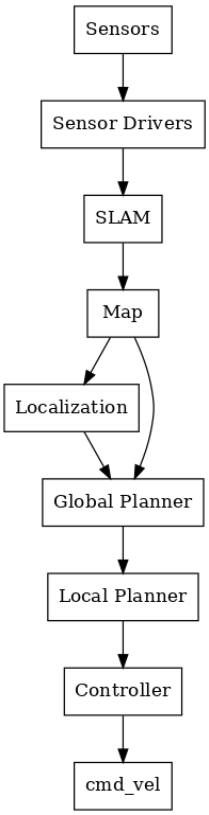


Figure 5. ROS2 localization and navigation flowchart.

- **SLAM**: A LiDAR-based 2D SLAM algorithm generates high-resolution occupancy grid maps from streamed data from the YDLiDAR TSA LiDAR, which provides the point cloud data (blue color variation in Figure 6) at 7 Hz with a 360° view angle and 10 m range. Cartographer employs scan matching and loop closure to construct maps at a 0.05 m resolution with a local SLAM window of 5 s and a submap resolution of 0.05 m, being optimized for indoor environments with dynamic obstacles. Generated maps are published onto the ROS2 topic `/map` for use by the navigation stack.
- **Localization**: Adaptive Monte Carlo Localization (AMCL) provides robust pose estimation by combining data from the YDLiDAR TSA LiDAR, MPU6050 IMU (6-DOF orientation at 100 Hz), and wheel encoders (odometry at 200 Hz, sent at 50 Hz over the ESP32). AMCL runs a particle filter of 2000 particles updated at 10 Hz with a probabilistic motion model motivated by the differential drive kinematics (25 cm wheelbase, with a maximum speed of 0.5 m s^{-1}), yielding a localization error of 0.03 m in real-world tests. The `tf2` library also manages the transform tree with coordinate frames (e.g., `map`, `odom`, `base_link`, `lidar_link`), with transformations published at 50 Hz.
- **Global Planning**: The NAV2 framework's global planner computes optimal, collision-free trajectories from the current pose of the robot to meet user-specified goals by employing the A* algorithm. A* acts on a 0.05 m resolution 2D occupancy grid costmap from the Cartographer map (black outline in Figure 6), which the `costmap_server` updates at 5 Hz. The planner uses a Euclidean distance heuristic, generating paths at a 0.1 m resolution, accounting for the robot's footprint by scaling up obstacles by 0.2 m for a safety margin. The global path is published onto the ROS2 topic `/plan` for use as input for local planning.
- **Local Planning**: The dynamic window approach (DWA) is the sole local planner used by NAV2, evaluating feasible velocity commands within a dynamic window constrained by the robot's kinematic limits (maximum linear velocity 0.5 m s^{-1}, maximum rotational velocity 1.0 rad s^{-1}), as well as by how closely the robot is positioned to the obstacles. DWA evaluates a sampling of potential velocity commands (linear and rotational) within a dynamic window based on criteria such as how closely the robot approaches obstacles, convergence with the global path, and how much progress the command makes toward the goal. The command that makes the maximum progress while ensuring safety within the environment is chosen. Real-time LiDAR updates the local costmap used as the movement planner around dynamic obstacles.
- **Feedback and Monitoring**: NAV2 provides mechanisms for providing feedback for the monitoring and debugging of the navigation process.
  - **Path Visualization**: The local and global plans are published in RViz2 on the `/plan` topic, as shown in Figure 6, which allows developers to see the planned path (green line) and identify issues.
  - **Costmap Visualization**: Both global and local costmaps are published and portrayed in RViz, including the costs for obstacles, inflation fields, and the robot's footprint, thereby facilitating the diagnosis of navigation failures in simulations.
  - **Navigation Status**: The `bt_navigator` provides status reporting by means of the `/navigate_to_pose` action feedback, progress, and current state, as well as several error conditions (e.g., "goal unreachable" errors)

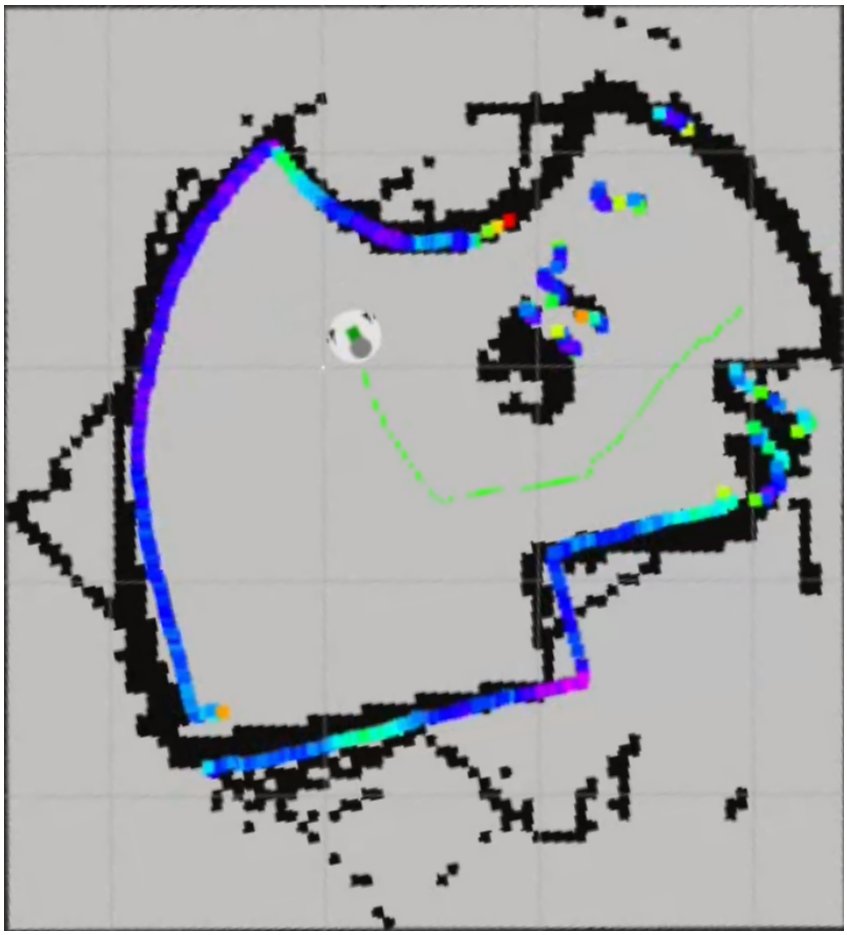


Figure 6. Visualization in Rviz2.

4.3. Object Detection

The object detection system specifically classifies and detects common objects in real time and allows the robot to locate lost items in disordered interior settings. This work addresses the specific problem of restricted mobility by enabling the recovery of goods that are beyond reach. YOLO11 is the latest variant of the You Only Look Once (YOLO) computer vision models. The model is optimized for real-time processing and has an unparalleled trade-off in terms of processing speed, efficiency, and accuracy that cannot be matched by any other algorithm. The YOLO11n model has the lightest and fastest architecture. YOLO11n can successfully identify common objects like bottles, books, remote controllers, and small tools, which can help assistive robots to retrieve objects that of out of reach for a person with mobility limitations. Object recognition by YOLO11n enables the search algorithm to focus on target spots. YOLO11n runs on the Raspberry 4 Model B (4 GB RAM, quad-core cortex-A72 processor running at 1.5 GHz) as a compromise considering computational power and resource constraints.

To assess the computational efficiency of the current state-of-the-art object detection algorithms to be used in embedded platforms, we analyzed and contrasted the performance results obtained in various studies and community challenges on the Raspberry Pi 4 Model B with 4 GB of memory. The performance of several algorithms, namely YOLO11, YOLOv8, EfficientDet-Lite, Faster R-CNN, and SSD-MobileNet, in terms of their average latency and FPS throughput is summarized in Table 1. The performance data clearly indicate that there is a trade-off between the accuracy of object detection and real-time performance while implementing AI algorithms on low-power hardware platforms. Out of the evaluated models, YOLO11-Nano offers the best compromise between mean latency

and throughput in object detection tasks and can easily fulfil the performance needs of our proposed misplaced object search algorithm on the Raspberry Pi 4.

The proposed architecture introduces a distributed processing system through cooperation between the Raspberry Pi and ESP32 microcontroller, as shown in Figures 2 and 3, whereby ESP32 runs FreeRTOS and performs three tasks at once, including managing the motors of the robot, receiving motor control signals from the Raspberry Pi, and sending sensor information such as IMU and encoder readings back to the Raspberry Pi.

The delegation of low-level control and sensor data acquisition to the ESP32 optimizes the processing time on the Raspberry Pi, thereby making it more efficient than in the present scenario, as it is based on a monolithic architecture and has high system latency due to processing on the Raspberry Pi itself. Additionally, the system achieves flawless, dynamic, and continuous data transfer through the combination of ROS2 on the Raspberry Pi and FreeRTOS on the ESP32, making the overall robotic system free from latency and more efficient due to its ability to run low-level actuation tasks effectively at low latency.

Table 1. Comparison of object detection model performance on Raspberry Pi 4 Model B (4 GB).

Model	Mean Latency (ms)	Throughput (FPS)
YOLO11-Small (YOLO11s) [27]	300–1200	0.8–3.3
YOLOv8-Nano (YOLOv8n) [28]	400–700	1.4–2.5
EfficientDet-Lite0 (TFLite, INT8) [29,30]	50–150	6–20
Faster R-CNN (ResNet-50 + FPN) [30]	2000–6000	$<$0.5
SSD MobileNet v1 (TFLite, INT8) [28]	$\approx$209	$\approx$4.8
The proposed model	100–300	3.5–10

Model Selection and Implementation: YOLO11n, as a result of its optimality for devices with limited resources, is used without modification through TensorFlow Lite for streamlined Raspberry Pi inference. The pretrained YOLO11n model, which can be downloaded from Ultralytics and is trained on the COCO dataset comprising 80 object classes, provides the capabilities for detecting and recognizing home objects such as telephones, remote controls, keys, and cups. The model accepts input images of 128 $\times$ 128 pixels for a smaller computational overhead while providing sufficient detail for detection. The model provides class-labeled bounding boxes and confidence tensors, reaching a mean precision level of up to 95% for target objects when deployed in indoor settings, as verified against a custom test collection of domestic objects.

Data Acquisition and Preprocessing: The Logitech StreamCam takes images, providing a 78° wide angle of view at a 1080 p resolution at 60 fps. The camera is located 12 cm above the chassis and tilted upward to capture the forward scene. Images from it are scaled down by bilinear interpolation to the 128 $\times$ 128 pixel input required by YOLO11n, with colour normalization used for further detection robustness when the indoor lighting varies. The system publishes visual information through the `/camera/image_raw` topic at a rate of 5 Hz.

Inference Pipeline: The object detection code from the YOLO11n model serves as an ROS2 node (`yolo_detector`) optimized to run on the Raspberry Pi. This node subscribes to the `/camera/image_raw` topic, processes images into the necessary input format of 128 $\times$ 128 pixels, and conducts inference directly from the unmodified YOLO11n model. The inference runs at 1.25 Hz, with average latency of 800 ms, once every frame from the beginning of preprocessing until the termination of postprocessing. The model produces bounding boxes, class labels, and confidence scores filtered by a confidence filter of 0.5 and a non-maximum suppression (NMS) filter using an IoU score of 0.4, removing duplicate detections. Results are published onto the `/yolo/detections` topic as a custom ROS2 mes-

sage bearing bounding box coordinates, class IDs, and confidence scores, complemented by the grid-based search algorithm for target localization.

4.4. Custom Search Algorithm

Current methods, like the ones used by Cui [31], typically utilize hybrid methods that blend SLAM-oriented global planning and local avoidance schemes like the dynamic window approach (DWA). While efficient for goal-oriented movement, these tend to focus on path optimization and not thorough environment coverage. Hence, they can neglect certain areas and lower the search accuracy in systematic search scenarios. For this reason, a custom grid-based search algorithm is introduced, guaranteeing equal exploration, ease of waypoint handling, and sound integration with perception and navigation modules. This custom search algorithm permits the comprehensive exploration of unstructured indoor scenes for the purpose of detecting lost objects, while being suitable for use within the robot's navigation and perception components. It applies a grid-based approach to generating waypoints. A grid-based approach in search algorithms typically entails a grid or matrix representation of the problem's search space. Each state or location in the problem's environment is represented by a corresponding node or cell in the grid. The strategy for the proposed solution relies on a search via a grid pattern that divides the environment into traversable waypoints. Unlike similar search algorithms that explore the environment by only utilizing geometry patterns to search for waypoints or targets of interest, the strategy presented integrates perception and search by utilizing the YOLO11n detector to analyze images while planning with the NAV2 technique for motion to explore each waypoint. This offers comprehensive environment coverage, employing YOLO11n for object detection purposes and NAV2 for navigation.

Waypoint Generation: The environment is discretized into a 0.5 m $\times$ 0.5 m grid of cells according to the 2D occupancy grid map from Cartographer (resolution of 0.05 m). Waypoints are provided to the center of each of the free cells (e.g., cells that are not marked as obstacles in the map), allowing for navigable goals. The grid is created on the fly from the `/map` topic, per waypoint, represented as a list of (x, y) coordinates in the `map` frame. For an environment of 5 m by 3 m, approximately 11 waypoints are generated (yellow points in Figure 7), but this is reduced when obstacles such as furniture or walls are present. The waypoints are shown in Figure 7, with their spread over an example indoor map with obstacles.

Algorithm Implementation: The search algorithm, which is executed as an ROS2 node (`search_node`), is given by Algorithm 1. It initializes two lists of visited and failed waypoints. It iterates over unvisited waypoints, choosing the closest waypoint using the Euclidean distance from the robot's current position (parsed from AMCL by the `tf2` library). The NAV2 stack with A* for the global planner and DWA for the local planner controls the robot towards the selected waypoint. At each waypoint, the `yolo_detector` node processes images from the Logitech StreamCam, publishing the detection results on the `/yolo/detections` topic. If there is a target object detection (the confidence score of the class of an object is greater than 0.5), the algorithm publishes its position in the `map` frame and terminates. If the robot cannot reach the waypoint due to dynamic obstacles or kinematic constraints, the waypoint is recorded as failed, and up to three attempts are made before skipping. The time complexity of the algorithm is O(n), where n is the number of waypoints, with constant-time detection by YOLO11n (800 ms per waypoint).

Integration and Operation: The `search_node` lists itself as a subscriber of the `/map` topic of the occupancy grid, `/yolo/detections` of object detection outputs, and `/tf` of the robot's pose. The `search_node` publishes navigation goals on the `/nav2/goal` topic by communicating with NAV2.

Algorithm 1 Grid-Based Search Algorithm

```
procedure SEARCHLOOP
    Generate waypoints for grid cells
    Initialize visited and failed waypoint lists
    while target not found do
        Select unvisited waypoint
        Navigate to waypoint using NAV2
        Process YOLO11n detections
        if target detected then
            Publish target location
            break
        end if
        if navigation fails then
            Mark waypoint as failed
            Retry up to 3 times
        end if
    end while
end procedure
```

Validation and Performance: The algorithm was also validated in actual test experiments on an indoor setup. The grids enclosed the volume within less than 3 min at a 100% success rate for target object detection.

The coverage map, illustrated in Figure 7, shows the overlapping circular fields of view (FOVs) (blue circles) that are utilized to guarantee full room coverage.

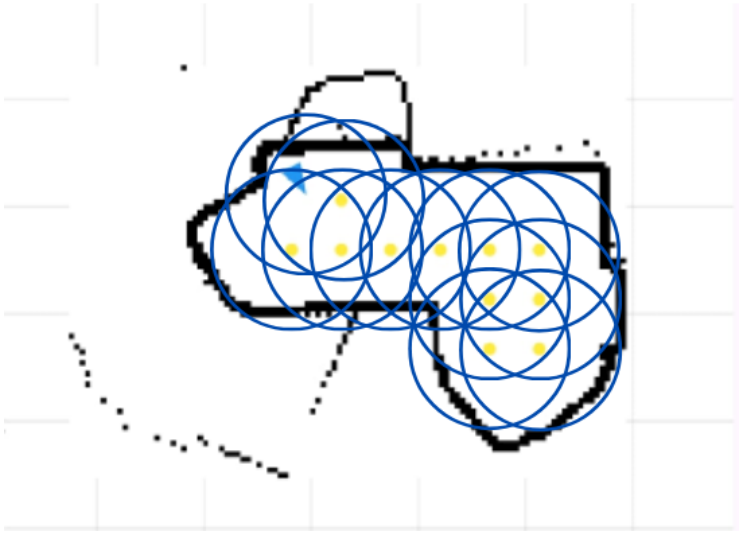


Figure 7. Coverage map: circular FOVs overlapping for room coverage.

4.5. Motor Control

The smooth movement of the indoor autonomous robot requires motor control to ensure effective and precise motion. Smooth following of the path, collision prevention, stability despite changing scenarios, and the prevention of wild riding or high power usage are ensured by effective speed control. The speed of the DC motor is calculated by DC motor encoders that are connected to the ESP32 by two pins (encA and encB) for the purpose of detecting rotational movement. Accurate speed data are needed for PID control, which enables real-time motor adjustment for smooth robot motion.

- **Quadrature Encoding and Position Tracking**: The quadrature encoder offers two signals, A (encA) and B (encB), to monitor the motor's rotation. They are two waves phase-shifted by each other. Imagine two friends clapping their hands, but one a

fraction of a second after the other. This phase shift enables the ESP32 (a very small microcontroller, or a minicomputer) to determine both the speed and direction of turning, as illustrated by Figure 8. The Init routine configures encA and encB as input pins and attaches an interrupt (encoderISR) to encA. An interrupt occurs when there is a state change in encA, so that the system will notice at once rather than continuously checking. This interrupt occurs for any change in encA's state, so that encoder ticks (small rotational steps) are quickly recognized. The Update routine, prompted by the interrupt, reads encA's and encB's states to determine the direction of turning. The direction detection logic is given below.

- If `encB` is high (e.g., "on" or 1) and `encA` is high, then the wheel rotates clockwise (the direction when turning a clock's hands to the right).
- If `encB` is high and `encA` is low (e.g., "off" or 0), the wheel rotates counterclockwise (opposite direction), so the position count decreases by -1.
- If `encB` is low and `encA` is high, the wheel turns counterclockwise; hence, the position decreases by -1.
- If `encB` is low and `encA` is low, the wheel turns clockwise; hence, the position increases by +1.

The position variable is the encoder's cumulative count of ticks, increasing as we turn clockwise and decreasing as we turn counterclockwise, giving a real measurement of the motor's position. It is analogous to maintaining a bidirectional step count for wheel rotation.

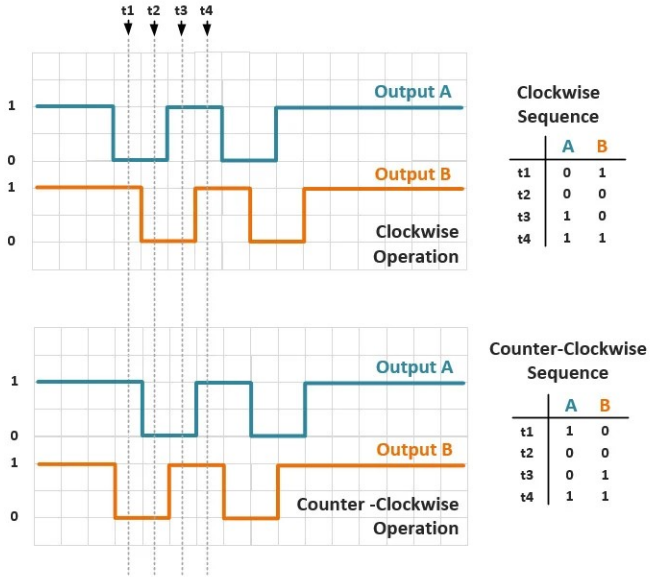


Figure 8. Quadrature encoder signals for detection of rotational direction.

- **Speed Calculation**: The GetfilteredVelocity procedure calculates the motor speed in revolutions per minute (RPM). RPM is the number of complete revolutions per minute that the motor undertakes, similar to the number of turns per minute in which a wheel performs a full revolution. It obtains the current position (`GetPosition`) by means of a critical region with a mutex (muxMotor) to ensure thread safety. A mutex is similar to a lock on a door-only one process can enter at a time to avoid conflicts when multiple tasks are running simultaneously. The position change (Δpos) is then calculated by subtracting the previous one from the current one. The time elapsed (Δt) in seconds is obtained by the use of micros() (a procedure yielding the time in microsec, being very precise). The initial speed in counts per second (counts/s) is determined by Formula (1):

$$v = \frac{\Delta \text{pos}}{\Delta t} \tag{1}$$

This is converted to RPM using Formula (2):

$$v = \frac{\text{counts/s}}{204} \times 60 \tag{2}$$

where 204 represents the encoder's counts per revolution (a parameter specific to this model), and 60 is used to convert s^{-1} to min^{-1}.

- **Low-Pass Filtering for Speed Smoothing**: The measured speed is then filtered by a low-pass filter with a cut-off frequency of 25 Hz, thereby reducing measurement noise, and smooth control of the motor speed will be ensured. Here, "noise" denotes undesired signal fluctuations like radio static that could occur due to electrical noise or mechanical vibrations. A low-pass filter is similar to a sieve that accepts slow, gradual variations but rejects rapid, jerky variations, providing stable input to the PID controller. The filtering process uses the recursive Equation (3):

$$v_{\text{filt}} = 0.854 \times v_{\text{filt}} + 0.0728 \times (v + v_{\text{prev}}) \tag{3}$$

where v_{filt} is the filtered speed (the smoothed version), v is the current speed (raw measurement), and v_{prev} is the previous speed reading. In this formula, the old filtered value is combined with a fragment of the new data, creating a gradual smoothing effect.

- **State Update**: The algorithm continually reverts back to the previously noted position, velocity, and time (previousPosition, previousVelocity, previousTime), ensuring continuous speed computation. This process is analogous to a reset operation on a stopwatch, performed after the completion of every lap in preparation for the next one.

This procedure provides a constant speed measure, required for regulation by the PID controller.

4.6. Motor Speed Regulation Process

The procedure of motor speed regulation involves maintaining the DC motors at the desired rotational speed by utilizing speed information from the measurement of the encoders. In short, automatically changing the motor's power maintains the motor at the desired speed, similarly to a car's cruise control that maintains the speed despite the fact that the driver does not hold the pedal continuously. A manual PID tuning procedure was initially applied, followed by an adaptive fuzzy logic PID controller for higher efficiency when the situation changes. The following sections describe the procedure of regulation by comparing the two while describing the fuzzy logic implementation.

4.6.1. Manual PID Tuning

The initial solution was manual PID tuning, with the PID gains, Kp (proportional gain), Ki (integral gain), and Kd (derivative gain), tuned by trial and error towards a stable response. PID is a classic control system resembling a thermostat that will activate the heating depending upon how far from the set point the temperature is. The tuning consisted of the following:

- Initializing gains (e.g., Kp = 1.0, Ki = 0.0, Kd = 0.0) and observing the motor's response while being driven by a setpoint speed (the desired speed).
- Setting Kp until the motor speed was jittery around the setpoint, after which it was reduced by a small amount until a stable steady-state response was achieved. Kp will react to instantaneous errors, e.g., pushing harder if farther away.

- Tuning Ki to decrease the steady-state error (ongoing deviation from the aim) as the motor approached the setpoint over time. Ki is responsible for accumulated errors, such as providing an additional thrust if slow for a while.
- Fine-tuning Kd to dampen oscillations but remaining low enough so as not to enhance noise. Kd takes the rate of change into consideration, such as braking if entering too quickly.

The result of this strategy is shown in Figure 9, where we observe the motor speed often moving beyond the goal setpoint, causing overshoot (up to 220 RPM while the goal is 200 RPM). This produced rather rough speed transitions, notably at varying loads (such as when we ascend), with a settling time of approximately 0.6 s and noticeable oscillations.

There are shortcomings of manual PID tuning. Gains that are constant do not adjust rapidly when there are quick load transfers or changing terrains, causing the following:

- Overshoot and oscillations, like those observed in the above chart, which will destabilize the robot (rendering the robot jerky or unstable);
- Inconsistencies when in the presence of non-linear motor responses with unpredictable responses or with shifting environmental parameters;
- A labor-intensive tuning process that requires continual fine-tuning for different situations.

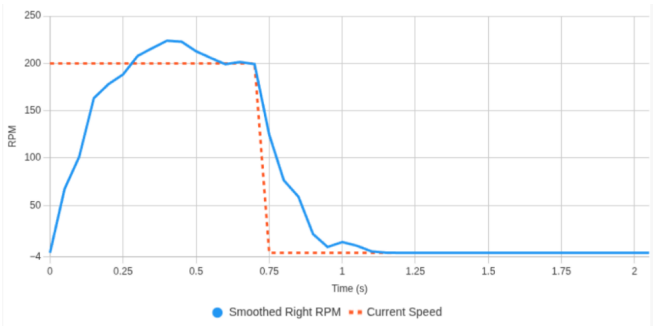


Figure 9. Manual PID tuning results.

4.6.2. Adaptive Fuzzy Logic PID Controller

In response to the drawbacks of manually tuning the PID, an adaptive fuzzy logic-based PID controller was devised. This technique, inspired from Nassim and Abdelkader [32] work, changes the PID gains automatically by varying the error (e) and its rate of change (dedt), providing a more robust solution. Fuzzy logic is akin to embedding human-like reasoning into programs, operating with "shades of gray" instead of binary yes/no values, by virtue of rules such as "if it's warm, cool a little". This enables the system to cope with robot load variations or surface variations.

Selection of Membership Functions: Trapezoidal membership functions were chosen for the system. Their flat top ensures a stable and firm response for large errors, necessary for quick corrective action in motor speed control. Smooth variation in the control output is ensured by their overlapping parts to prevent jerky motion during load transients. Their low complexity qualifies them for the resource-constrained ESP32-based platform, yet offers stability, responsiveness, and efficiency.

4.6.3. Selection of Fuzzy Rules

The fuzzy logic system begins by selecting the fuzzy rules, designed to make the response aggressive for quick error correction and a stable motor speed. They follow if–then structures, similar to natural language reasoning, for example, "if the error is large and changing rapidly, then adjust strongly." The error (e) and its derivative (dedt), scaled to the range $[-1.0, 1.0]$ (scaled to lie within -1 and 1 for consistency), are defined via

seven linguistic terms: Negative Big (NB), Negative Medium (NM), Negative Small (NS), Zero (Z), Positive Small (PS), Positive Medium (PM), and Positive Big (PB). They are of the like-label type: NB means a large negative error (far too slow), and PB means far too fast. They are defined by the trapezoidal membership function (*trapezoidalMF*), with parameters selected such that there is a small amount of overlap for smooth operation (no jumpiness). An example is NB from -1.0 through -0.6 and NM from -0.8 through -0.2, defined by the *calculateMemberships* routine. The membership functions are depicted in Figure 10.

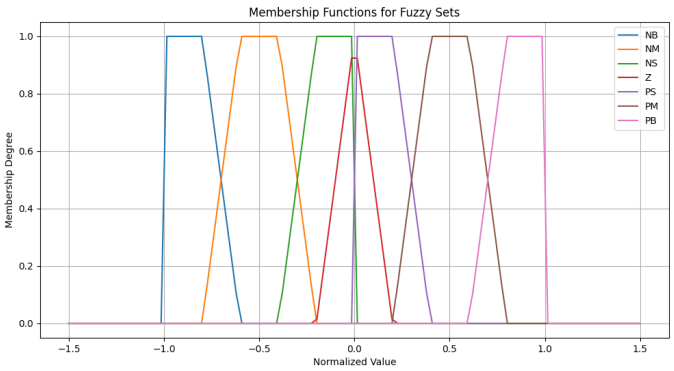


Figure 10. Membership functions.

4.6.4. Fuzzy Rule Table

The fuzzy rules are implemented as 7×7 matrices (rule_Kp and rule_Ki) in the fuzzyInference procedure to adjustably set the PID controller gains for motor speed control such that membership pairs for e and dedt are converted into proportional (Kp) and integral (Ki) gains with Kd = 0.0 (no derivative term here). They are selected such that they increase extremely aggressively for large errors (e.g., 6.0 for NB-NB, i.e., harsh correction for large deviations) but decrease rapidly for small errors (e.g., 0.6 for PB-PB, with subtle corrections for close values), scaled by certain factors (1.5 for Kp, 7.0 for Ki) for increased responsiveness. The tables are as follows.

- **Rule Table for Kp (× 1.5):**

dedt e	NB	NM	NS	Z	PS	PM	PB
NB	6.0	5.8	5.6	5.4	5.2	5.0	4.8
NM	4.8	4.6	4.4	4.2	4.0	3.8	3.6
NS	3.6	3.4	3.2	3.0	2.8	2.6	2.4
Z	2.4	2.2	2.0	1.8	1.6	1.4	1.2
PS	2.2	2.0	1.8	1.6	1.4	1.2	1.0
PM	2.0	1.8	1.6	1.4	1.2	1.0	0.8
PB	1.8	1.6	1.4	1.2	1.0	0.8	0.6

- **Rule Table for Ki (× 7.0):**

dedt e	NB	NM	NS	Z	PS	PM	PB
NB	6.0	5.8	5.6	5.4	5.2	5.0	4.8
NM	4.8	4.6	4.4	4.2	4.0	3.8	3.6
NS	3.6	3.4	3.2	3.0	2.8	2.6	2.4
Z	2.4	2.2	2.0	1.8	1.6	1.4	1.2
PS	2.2	2.0	1.8	1.6	1.4	1.2	1.0
PM	2.0	1.8	1.6	1.4	1.2	1.0	0.8
PB	1.8	1.6	1.4	1.2	1.0	0.8	0.6

The rules favor fast error correction for large errors (e.g., high Kp and Ki for NB-NB, such as quickly using the brakes if out of control) and weaker gain strength as the error

becomes close to zero (e.g., low Kp and Ki for PB-PB), offering stability around the setpoint (target speed). The error, error derivative, and Kp gain interaction are graphically depicted in Figure 11.

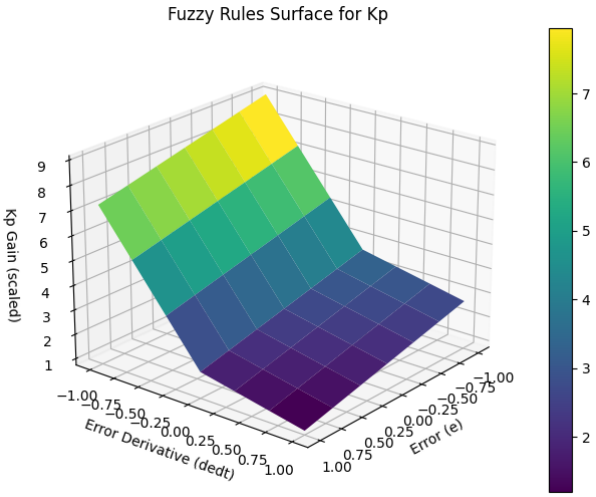


Figure 11. Fuzzy rule surface for Kp.

4.6.5. Fuzzy Inference System (FIS) Rules

The FIS rules operate as follows. Membership degrees for e and dedt are computed by the "fuzzyInference" function using "calculateMemberships", and the minimum operator is applied as shown in Formula (4)

$$\text{weight} = \min(\text{eMembership}[i], \text{dedtMembership}[j]) \tag{4}$$

to determine the firing strength of each rule (the extent to which the rule applies, selecting the smaller of the two memberships). The weighted sums are calculated as shown in Formulas (5)–(7):

$$\text{sum_kp} = \sum_{i=0}^{6} \sum_{j=0}^{6} (\text{weight} \times \text{rule_kp}[i][j] \times 1.5) \tag{5}$$

$$\text{sum_ki} = \sum_{i=0}^{6} \sum_{j=0}^{6} (\text{weight} \times \text{rule_ki}[i][j] \times 7.0) \tag{6}$$

$$\text{sum_weights} = \sum_{i=0}^{6} \sum_{j=0}^{6} \text{weight} \tag{7}$$

The final gains are calculated using Formula (8):

$$\text{kp} = \frac{\text{sum_kp}}{\text{sum_weights}}, \quad \text{ki} = \frac{\text{sum_ki}}{\text{sum_weights}}; \quad \text{sum_weights} > 0 \tag{8}$$

If sum_weights = 0, the previous gains are retained. This center-of-gravity method ensures a smooth gain transition.

The fuzzy logic PID controller controls the motor speed to an even greater extent through the calculation of the Kp and Ki gains from the normalized change in error with the aid of trapezoidal membership functions for input clustering into seven linguistic sets (NB through PB). It also includes gain determination rule tables with the aggressive setting of gains and a weighted average defuzzification technique (converting fuzzy outputs into crisp numerical values), which is superior to interactive PID tuning under fluctuating loads.

4.6.6. Implementation Process

The procedure then continues as follows.

- **Error Calculation**: The ComputeFuzzy method uses error (e) calculation by subtracting the desired setpoint (from ROS 2/`cmd_vel`, robot's navigation system command) from the filtered speed (from GetfilteredVelocity). The time difference from the last call is obtained with micros() for appropriate derivative calculation accuracies; there is also a safeguard against division by zero (to prevent mathematical errors).
- **Error Derivative Calculation**: The derivative of the error (dedt) is computed using Formula (9):

$$\text{dedt} = \frac{\text{currentError} - \text{previousError}}{\text{deltaTime}} \tag{9}$$

  This makes the controller capable of responding to rapid speed changes.
- **Normalization**: To fit the error (e) and error derivative (dedt) to the fuzzy logic system and to allow corner cases like zero or negative setpoints (where the scaling values will be inside the fuzzy "box"), the Map function normalizes them to range from -1.0 to 1.0.
- **Membership Calculation**: Function *calculateMemberships* utilizes trapezoidal membership functions (trapezoidalMF) to project the normalized error (e) and error derivative (dedt) into seven fuzzy sets as a representation of each one's degree of membership of NB.
- **Fuzzy Inference**: The fuzzy inference mechanism utilizes already defined rules to compute adjusted gains (Kp, Ki, Kd), in which Kd is 0.0.
- **PID Computation**: The ComputeFuzzy method calculates the control signal using the adjusted gains according to Formula (10):

$$\text{controlSignal} = Kp \times e + Kd \times \frac{de}{dt} + Ki \times \int e\,dt \tag{10}$$

  There is anti-windup protection on the integral term and a limit on the output so that it does not saturate (it holds signals within safe ranges).
- **Output of Motor Control**: The control signal from the L298N motor driver (a chip controlling the direction and speed of the motors) is transmitted to the DC motors, with the PWM signal being changed. The ESP32 adjusts the last error (e) and time before the next iteration.

4.6.7. Results and Performance

The fuzzy PID outputs according to Formula (10) are graphed in Figure 12 and compared to those of manual PID. There is no overshoot in the plot, with the speed closely tracking the setpoint (e.g., stable at 200 RPM without exceeding it).

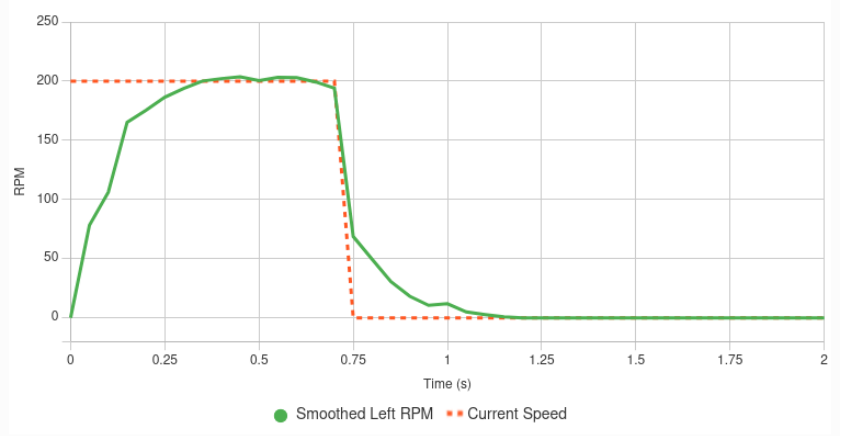


Figure 12. Fuzzy logic PID results.

The fuzzy PID allows smoother speed transitions, especially with loads, with the settling time now being less than 0.3 s and almost 0 oscillations, which enhances the navigation performance (the robot becomes more reliable and efficient in moving).

This adaptive mechanism is appropriate in terms of motor speed control, efficiently dealing with real-time changes as well as continuous communication with the ROS 2 navigation stack.

4.7. Flutter Application Development

Developing a Flutter app is essential for good user interaction when dealing with an autonomous robot, offering a user-friendly interface to monitor tasks in real time. This provides instructions, robot statuses, and movement checks, being easy to integrate into houses. It becomes challenging when using an inefficient app to perform tasks or inspect responses. The proposed mobile application offers a complete control and monitoring interface for the autonomous robot. Users can remotely control the robot, track its motion on the real-time map, and follow sensor information updates from the robot. Using the application, users can start the navigation task or execute object searches. The application also enables users to receive real-time updates on the environment of the robot and the progress of its tasks. Flutter, developed by Google, is an open-source user interface development kit used to create natively compiled mobile, web, and desktop applications from a single codebase. It comes packaged with the Dart programming language to handle fast apps with richly styled, smooth motion interfaces [33]. Flutter's widget-based architecture makes it possible to easily create novel, responsive interfaces, which is critical in developing an intuitive robot control interface.

Flutter Application Development for Robot Control:

The Flutter app was designed to present an easy-to-use interface from which to instruct and follow the self-driving robot. It communicates with the robot through a WebSocket server on a laptop, which converts commands to the ROS2 stack running on the robot. The app's development included the following notable aspects.

- **User Interface Design**: The easy-to-understand and user-friendly interface of the app allows users to start tasks (e.g., begin navigating, search objects), see the online status (e.g., current real position), and see the robot's map. The interface, shown in Figure 13, provides buttons to move the robot in manual mode, provides a view of the map to navigate, and has displays of the status, so users without technical expertise can find it easy to communicate with the robot.
- **Task Management**: There is a type of task management offered by the software that enables users to assign tasks to the robot, such as initializing the robot position (green arrow), moving to a specific location (red arrow) or beginning an object search. These tasks are then encapsulated in the form of ROS2 commands (e.g., `/cmd_vel` when dealing with moving) and are transmitted through the WebSocket server. This is similar to providing a robot with a list of tasks that it executes on its own.
- **Real-Time Monitoring**: The program retrieves the status of the robot and its location information (blue triangle), sensor information, and task accomplishment information like remained waypoints to finish the search task (yellow dots) in real time through ROS2 topics that are subscribed to through WebSocket subscriptions, so that users can view the movement of the robot, such as monitoring a tracked vehicle across a surveyed area.
- **ROS2 Integration**: The application utilizes an ROS2-WebSocket bridge to enable connectivity with the controller program that is integrated into the robot system. To ensure easy and smooth interactions, this bridge translates application commands into ROS2 syntactic form and then forwards them to the robot.

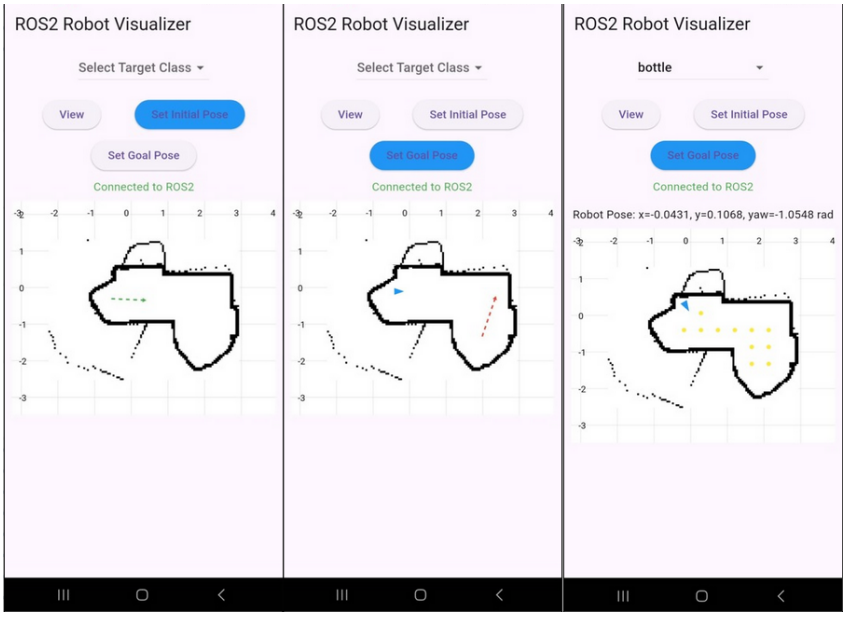


Figure 13. Main interface of the Flutter application for robot control.

The app has been tested and proven in simulations and in practical scenarios with adequate user communication and interaction. It demonstrated the low-latency issuance of commands and stable reporting of feedback, so that the robot could be issued commands by non-technical users.

5. Results and Discussion

5.1. Simulation Tests

The simulation tests, with Gazebo and RViz2, confirmed the efficacy of the ROS2-based autonomous mobile robot's control. Gazebo is a 3D physics simulation software program with realistic environment simulations and robotic sensor modeling for system testing, while RViz2 is an ROS2-specific visualization software program with ROS2 real-time displays of sensor information, maps, and robot poses. Gazebo was chosen given its strong physics engine and realistic robotic modeling capabilities, with the accurate emulation of LiDAR, IMU, and wheel encoder data, while RViz2 was chosen given its real-time displays, enabling efficient debugging and the monitoring of performance, a combination that is routinely applied in robotics research. Takaya et al. [34] also used Gazebo and RViz as ROS simulation softwares. As shown in Figure 14, the integration of these sensors ensured correct localization via Adaptive Monte Carlo Localization (AMCL), with a 95% success rate during mapping and navigation through dynamic indoor simulation worlds with moving objects. The dynamic window-based (DWB) controller ensured effective path planning, with the robot succeeding in target waypoints, with an average positional error of 3 cm. The YOLO-based object detection platform, with testing through simulations, properly identified domestic items with a 95% success rate under varying lighting levels. The Flutter app with the WebSocket bridge interface ensured low-latency command responses (<100 ms) and a full real-time view of the robot's map and status with the smooth handling of tasks. However, the simulation sluggishness under very complex worlds suggests the need to use optimally equipped computing hardware.

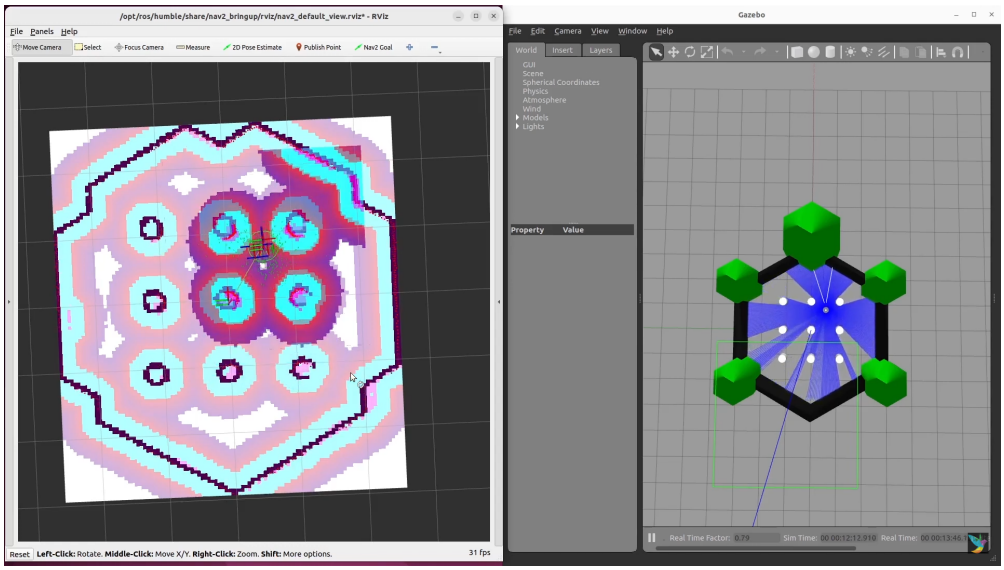


Figure 14. Initial simulation test in Gazebo.

5.2. Real-World Tests

As illustrated in Figure 15, real-world tests validated the robot's performance in a realistic indoor environment, with the results broadly replicated in simulations but also with understandable practical restrictions. The test environment was set up following an inverted L-shape arrangement. Room dimensions were 4 m by 2.5 m, with a recess of 1.25 m by 1.25 m at one corner, making its geometry an inverted L. Environment illumination was achieved by employing two white LED lamps of 3000 lumens. Real-sized objects were used to simulate realistic scenarios (doors, bottle, walls, etc.). The ROS2 platform using LiDAR and ultrasonic sensing achieved a 95% success rate in navigating uneven surface floors and cluttered rooms, with the adaptive fuzzy logic PID controller implemented on the ESP32 reducing the motor settle time to less than 0.3 s and the overshoot to less than 5 RPM from a setpoint of 200 RPM. The YOLO model with the Raspberry Pi achieved a 95% detection rate for domestic items but fell slightly short in poor-light scenarios due to noise on the sensing peripherals. Successful remote monitoring and operation were ensured with commands running within an average time of 120 ms via the Flutter app. Problems were found with occasional WebSocket latency during high network utilization and negligible localization drift with highly reflective surface locations, with potential for improved integration with sensors and noise reduction.

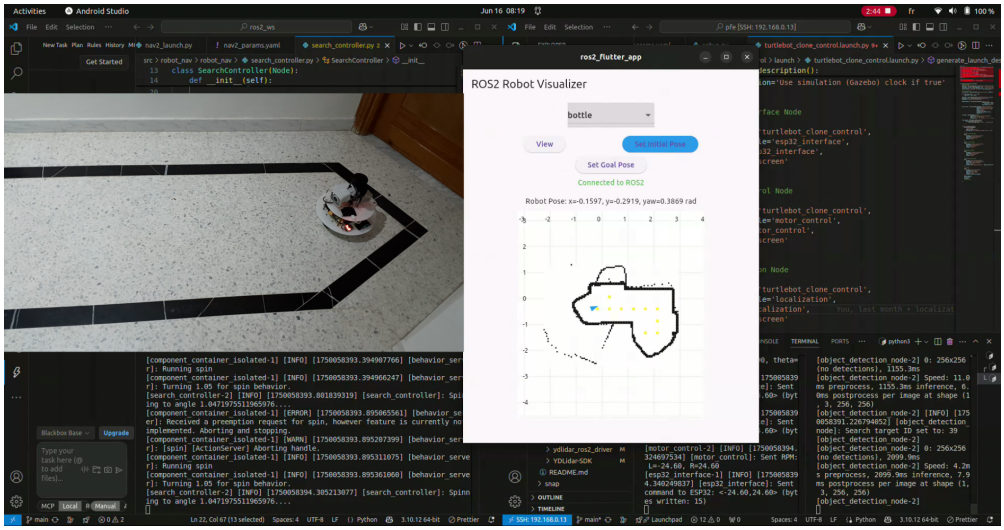


Figure 15. Real-world testing.

5.3. Comparison

Compared with commercially available autonomous mobile robots such as the Clearpath Jackal UGV (Clearpath Robotics, Kitchener, ON, Canada), MiR100 (Mobile Industrial Robots, Odense, Denmark), NVIDIA JetBot (NVIDIA, Santa Clara, CA, USA), and TurtleBot3 (ROBOTIS, Seoul, Korea), Table 2 shows that the constructed robot achieves similar performance in indoor domestic applications while having a considerably cheaper and more lightweight structure.

According to Robotis (2023), the TurtleBot3 achieves a 94% indoor navigation success rate with ROS2 and 360° LiDAR, but it costs USD 1800 and weighs 1 kg, limiting its applicability to longer domestic missions [7]. Alternatively, the suggested robot with a YDLiDAR TSA provides slightly better performance, with a 95% success rate along the course and a 3 cm position error, while having the same 1.4 kg weight but with a fraction of the cost (USD 369).

Clearpath Robotics (2023) reports that the Jackal UGV possesses 96% reliability in lab operation given the high-precision LiDAR and IMU implementation, but, given its high cost of USD 15,000+ and heavy 17 kg weight, it is impractical for home applications [8]. Likewise, the MiR100, according to Mobile Industrial Robots (2023), performs best in industry, with a 97% success rate in navigation, but with a 70 kg weight and USD 20,000+ cost; thus, domestic application is highly limited [15].

For perception, YOLO11n, with 90% accuracy, is also adopted in the developed robot, which is an improvement compared to the NVIDIA JetBot, which employs SSD MobileNetV2, with 85% accuracy in cluttered indoor settings [11].

However, the MiR100 still possesses the greatest communication latency (50 ms) due to the proprietary interface, while the constructed system implements a Flutter application using a WebSocket interface with 120 ms latency, which reflects optimization potential within the communication layer.

Overall, the designed robot is impressive, with a low price (USD 369), light weight (1.4 kg), and equal balance of navigation and perception capabilities, making it well suited for unstructured indoor environments as compared with heavier, more expensive commercial alternatives.

Table 2. Comparison of autonomous mobile robots.

Robot	Navigation Success Rate (%)	Positional Error (cm)	Weight (kg)	Cost (USD)	Object Detection Accuracy (%)	Latency (ms)
TurtleBot3 [7]	94	3 *	1	1800	N/A	120 *
Clearpath Jackal [8]	96	5 *	17	15,000+	N/A	80 *
MiR100 [15]	97	3 *	70	20,000+	92 *	50
NVIDIA JetBot [11]	69.0	8 *	0.8 *	250	85	150 *
Proposed robot	**95**	**3**	**1.4**	**369**	**95**	**120**

(*) Estimated values derived from similar systems or general robotics data owing to missing reference information.

6. Conclusions

Applying autonomous robots to human assistance is challenging and requires user-friendly interfaces to help users to perform everyday tasks such as picking and navigation, as well as affordability to ensure large-scale adoption and adaptation to unstructured do-

mestic environments. This work resolves these challenges by designing a new autonomous domestic robot tailored specifically to assistance indoors. Cost-competitive hardware consisting of a YDLiDAR TSA scanner, an ESP32 microcontroller, an ultrasonic sensor unit, and a Raspberry Pi-based sensor–actuator unit is combined with a high-level sophisticated software framework on ROS2. The solution results in **95% navigation success**, a **3 cm positional error**, and **95% YOLO-based object detection accuracy**, all while having a cost of just **USD 369**, which is much lower than those of comparable commercially available robots such as the TurtleBot3 or MiR100.

Moreover, the system integrates a customized grid-based search method, an adaptive fuzzy logic PID motor controller to control the momentum, and a Flutter-based mobile app that allows easy interaction even when the communication latency is **120 ms**. The experiments demonstrate that the proposed robot is competitive in terms of its cost–performance–extendability ratio, which qualifies it to be deployed in living rooms.

In the future, improving sensor fusion methods, network performance refinement, and the utilization of machine learning to enable real-time task prioritization will further improve the robot's performance, opening the door to broader applications in home environments.

Author Contributions: Conceptualization, A.G.; Software, A.G.; Validation, M.K. (Mohamed Karray); Writing—original draft, A.G.; Writing—review & editing, M.K. (Mohamed Karray), B.Z. and M.K. (Mohamed Ksantini); Supervision, M.K. (Mohamed Karray), B.Z. and M.K. (Mohamed Ksantini). All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The original contributions presented in this study are included in the article. Further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Keith, R.; La, H.M. Review of Autonomous Mobile Robots for the Warehouse Environment. *arXiv* **2024**, arXiv:2406.08333. [CrossRef]
2. Gonzalez-Aguirre, J.A.; Osorio-Oliveros, R.; Rodríguez-Hernández, K.L.; Lizárraga-Iturralde, J.; Morales Menendez, R.; Ramírez-Mendoza, R.A.; Ramírez-Moreno, M.A.; Lozoya-Santos, J.d.J. Service Robots: Trends and Technology. *Appl. Sci.* **2021**, *11*, 10702. [CrossRef]
3. Siciliano, B.; Sciavicco, L.; Villani, L.; Oriolo, G. *Robotics: Modelling, Planning and Control*, 1st ed.; Advanced Textbooks in Control and Signal Processing, Springer: London, UK, 2009; pp. XXIV, 632. eBook ISBN 978-1-84628-642-1/Hardcover ISBN 978-1-84628-641-4/Softcover ISBN: 978-1-84996-634-4. [CrossRef]
4. Sánchez-Ibáñez, J.R.; Pérez-del Pulgar, C.J.; García-Cerezo, A. Path Planning for Autonomous Mobile Robots: A Review. *Sensors* **2021**, *21*, 7898. [CrossRef] [PubMed]
5. Ishigami, G.; Nagatani, K.; Yoshida, K. Path Planning and Evaluation for Planetary Rovers Based on Dynamic Mobility Index. In Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems, San Francisco, CA, USA, 25–30 September 2011; pp. 601–606. [CrossRef]
6. Singh, D.; Trivedi, E.; Sharma, Y.; Niranjan, V. TurtleBot: Design and Hardware Component Selection. In Proceedings of the 2018 International Conference on Computing, Power and Communication Technologies (GUCON), New Delhi, India, 28–29 September 2018; pp. 805–809. [CrossRef]
7. Robotis. TurtleBot3: Specifications and Performance. Available online: https://www.robotis.us/turtlebot-3/ (accessed on 6 November 2025).
8. Clearpath Robotics Jackal UGV: Technical Overview. 2025. Available online: https://clearpathrobotics.com/jackal-small-unmanned-ground-vehicle/ (accessed on 7 October 2025).
9. Mobile Manipulation Platform for Autonomous Indoor Inspections in Low-Clearance Areas. In Proceedings of the 25th International Conference on Advanced Vehicle Technologies (AVT), Boston, MA, USA, 20–23 August 2023; Volume 1. [CrossRef]
10. NVIDIA Corporation. JetBot: An Open Source AI Robot Platform. 2024. Available online: https://jetbot.org (accessed on 9 October 2025).

11. Ramesh, G.; Jeswin, Y.; Rao, D.R.; Suhaag, B.; Uppoor, D.; Kiran Raj, K.M. Real Time Object Detection and Tracking Using SSD Mobilenetv2 on Jetbot GPU. In Proceedings of the 2024 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), Mangalore, India, 18–19 October 2024. [CrossRef]
12. Kawakura, S.; Shibasaki, R. Deep Learning-Based Self-Driving Car: JetBot with NVIDIA AI Board to Deliver Items at Agricultural Workplace with Object-Finding and Avoidance Functions. *Eur. J. Agric. Food Sci.* **2020**, *2*, 1–9. [CrossRef]
13. Pozyx. Industrial Automation: UWB RTLS for Robotics. 2025. Available online: https://www.pozyx.io/solutions/industrial-automation (accessed on 7 October 2025).
14. Crețu-Sîrcu, A.L.; Schiøler, H.; Cederholm, J.P.; Sîrcu, I.; Schjørring, A.; Larrad, I.R.; Berardinelli, G.; Madsen, O. Evaluation and Comparison of Ultrasonic and UWB Technology for Indoor Localization in an Industrial Environment. *Sensors* **2022**, *22*, 2927. [CrossRef] [PubMed]
15. Güldenring, R.; Görner, M.; Hendrich, N.; Jacobsen, N.J. Learning Local Planners for Human-aware Navigation in Indoor Environments. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Las Vegas, NV, USA, 25–29 October 2020. [CrossRef]
16. Soheilifar, S. Optimizing MiR100 Performance in Internal Logistics Through Digital Twin-Based Dynamic Speed Adjustment. 2025. Available online: https://webthesis.biblio.polito.it/secure/36032/1/tesi.pdf (accessed on 6 November 2025).
17. YDLIDAR. YDLIDAR TSA — 2D LiDAR Product Specifications. Product Datasheet (PDF); Detailed Specs: 0.12–8 m Range, 6 Hz Scan, 0.72° Resolution. Available online: https://www.worldrobotconference.com/uploads/exfile/video/s4tbfh.pdf (accessed on 7 October 2025).
18. Hakkim, F.; Rasheed, H.; Varsha, M.C.; Amal, K.A.; Minimol, B. Lidar Based Autonomously Navigating Robot. In Proceedings of the 2024 1st International Conference on Trends in Engineering Systems and Technologies (ICTEST), Kochi, India, 11–13 April 2024; pp. 1–6. [CrossRef]
19. Sultan, J.M.; Zani, N.H.; Azuani, M.; Yusop, A.M. Analysis of Inertial Measurement Accuracy using Complementary Filter for MPU6050 Sensor. *J. Kejuruter.* **2022**, *34*, 959–964. [CrossRef]
20. Dust, L.J.; Persson, E.; Ekström, M.; Mubeen, S.; Seceleanu, C.; Gu, R. Experimental Evaluation of Callback Behavior in ROS 2 Executors. In Proceedings of the 2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA), Sinaia, Romania, 12–15 September 2023; pp. 1–8. [CrossRef]
21. Open Navigation LLC. Navigation2 (Nav2)—Tutorials and Documentation. 2025. Available online: https://docs.nav2.org/ (accessed on 7 October 2025).
22. M J, A.K.; Babu, A.V.; Damodaran, S.; James, R.K.; Murshid, M.; Warrier, T.S. ROS2-Powered Autonomous Navigation for TurtleBot3: Integrating Nav2 Stack in Gazebo, RViz and Real-World Environments. In Proceedings of the 2024 IEEE International Conference on Signal Processing, Informatics, Communication and Energy Systems (SPICES), Kottayam, India, 20–22 September 2024; pp. 1–6. [CrossRef]
23. Google Cartographer ROS Integration—Documentation. 2022. Available online: https://google-cartographer-ros.readthedocs.io (accessed on 7 October 2025).
24. Dwijotomo, A.; Abdul Rahman, M.A.; Mohammed Ariff, M.H.; Zamzuri, H.; Wan Azree, W.M.H. Cartographer SLAM Method for Optimization with an Adaptive Multi-Distance Scan Scheduler. *Appl. Sci.* **2020**, *10*, 10347. [CrossRef]
25. Ultralytics. Model Comparisons (YOLO Family). 2025. Available online: https://www.ultralytics.com (accessed on 7 October 2025).
26. Cherubin Szymon, K.W.; Michał, S. YOLO object detection and classification using low-cost mobile robot. *Przegląd Elektrotechniczny* **2024**, *100*, 29–33. [CrossRef]
27. Ultralytics. Quick Start Guide: Raspberry Pi with Ultralytics YOLO11. 2025. Available online: https://docs.ultralytics.com/guides/raspberry-pi/ (accessed on 30 October 2025).
28. Alqahtani, D.K.; Cheema, A.; Toosi, A.N. Benchmarking Deep Learning Models for Object Detection on Edge Computing Devices. *arXiv* **2024**, arXiv.2409.16808. [CrossRef]
29. Kamath, V.; Renuka, A. Performance Analysis of the Pretrained EfficientDet for Real-time Object Detection on Raspberry Pi. In Proceedings of the 2021 International Conference on Circuits, Controls and Communications (CCUBE), Bangalore, India, 23–24 December 2021; pp. 1–6. [CrossRef]
30. Zagitov, A.; Chebotareva, E.; Toschev, A.; Magid, E. Comparative analysis of neural network models performance on low-power devices for a real-time object detection task. *Comput. Opt.* **2024**, *48*, 242–252. [CrossRef]
31. Cui, J. A search and rescue robot device realization based on ROS operating system combined with visual algorithm. *J. Phys. Conf. Ser.* **2023**, *2492*, 012020. [CrossRef]
32. Nassim, M.; Abdelkader, A. Speed Control of DC Motor Using Fuzzy PID Controller. *arXiv* **2021** arXiv:2108.05450. [CrossRef]

33. Boukhary, S.; Colmenares, E. A Clean Approach to Flutter Development through the Flutter Clean Architecture Package. In Proceedings of the 2019 International Conference on Computational Science and Computational Intelligence (CSCI), Las Vegas, NV, USA, 5–7 December 2019; pp. 1115–1120. [CrossRef]
34. Takaya, K.; Asai, T.; Kroumov, V.; Smarandache, F. Simulation environment for mobile robots testing using ROS and Gazebo. In Proceedings of the 2016 20th International Conference on System Theory, Control and Computing (ICSTCC), Sinaia, Romania, 13–15 October 2016; pp. 96–101. [CrossRef]

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.