P.A.R.C.E.L.: Payload Autonomous Robot for Coordination, Expedition, and Logistics

Riley Newport, Melyia Ince-Ingram, Brandon Zabawa, Ibrahim Khater

Dept. of Electrical Engineering and Computer Science, University of Central Florida, Orlando, Florida, 32816-2450

***Abstract* — P.A.R.C.E.L. (Payload Autonomous Robot for Coordination, Expedition, and Logistics), is an autonomous warehouse robot designed to transport payloads efficiently and safely within an indoor environment. The system integrates RFID technology for payload identification, LiDAR for obstacle detection and avoidance, and a Raspberry Pi 5 for navigation. Using a dynamic path-planning algorithm and SLAM mapping, P.A.R.C.E.L determines optimal routes in real time to minimize travel distance and avoid collisions. A companion mobile application provides users with an intuitive interface to assign payload destinations, monitor robot status, and view delivery progress. This project aims to demonstrate the potential of affordable, scalable automation in warehouse operations by combining hardware design, embedded systems, and intelligent control software into a cohesive, fully autonomous solution.**

***Index Terms* — Autonomous robots, Laser radar, RFID Tags, Robotics and automation, Simultaneous localization**

I. Introduction

With the rapid growth of shipping and delivery services, warehouses require more efficient ways to manage and transport products. To address this need, our team designed an autonomous delivery robot named P.A.R.C.E.L. (Payload Autonomous Robot for Coordination, Expedition, and Logistics). P.A.R.C.E.L. is developed to optimize payload transportation within warehouse environments while maintaining ease of use for operators.

The robot integrates RFID-based payload identification, obstacle avoidance, Visual SLAM navigation, and a mobile application for monitoring and control. Using RFID tags, P.A.R.C.E.L. identifies payloads and determines their destinations automatically. The Visual SLAM system allows the robot to map and localize itself in real time, dynamically avoiding obstacles and recalculating routes as needed. The mobile app communicates with the robot via Wi-Fi, providing users with delivery updates and control options.

P.A.R.C.E.L. demonstrates how autonomous robotics can enhance warehouse efficiency through automation, reliability, and user-friendly design, providing a practical foundation for future smart logistics systems.

II. System Overview

A. Hardware Block Diagram

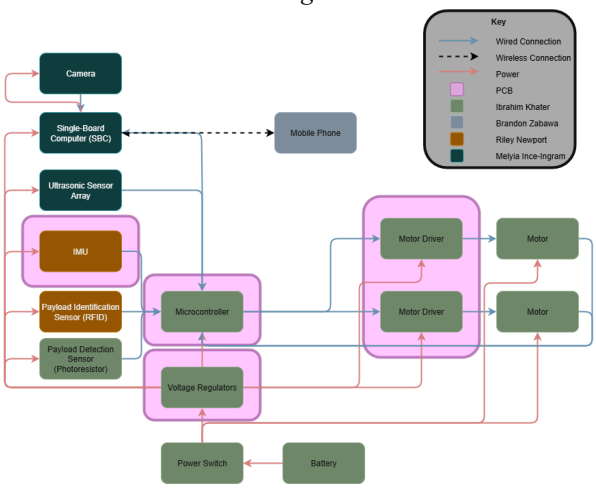

Fig. 1: Hardware Block Diagram

B. Software Block Diagram

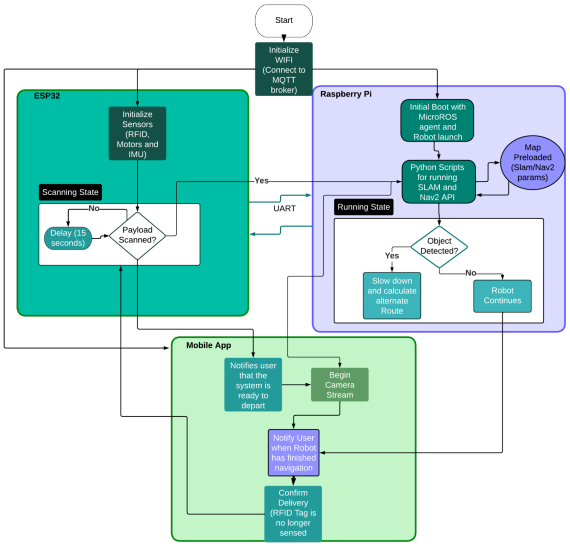

Fig. 2: Software Block Diagram

III. System Components

A. Payload Identification - RFID

The robot will include a mechanism to identify the name and location of each payload. There are several technologies that could help our robot achieve this including barcode scanning, quick response (QR) codes, or Radio Frequency Identification (RFID). RFID is a widely used technology that has been seamlessly integrated into our daily lives. It is used everywhere from preventing theft at retail stores to animal tracking to our government issued passports.

RFID will not require a person to scan the payload before placing it on the robot like a barcode would. Other advantages of RFID is the read time is much less than the barcodes, data can be updated, and payloads can be uniquely marked. It will also lay a strong foundation for future improvements aimed at achieving the stretch goal of detecting multiple payloads simultaneously. Although barcodes are less expensive than RFID, RFID is still an affordable option that will fit within our budget constraints.

For the purposes of this design, only minimal knowledge of RFID technology was required:

1) The RFID system is made up of two parts: the RFID reader module and one or more RFID tag(s).
2) The RFID module is powered but the type of RFID tags used in this project, high frequency passive tags, have no battery of its own and power only comes from the reader module through inductive coupling.
3) The RFID module then uses radio waves sent from the reader to the tag. As long as the tag is in range of the reader it will activate and send a signal back to the reader which interprets the data.
4) Readers have an RF signal generator, a microcontroller, and a receiver/signal detector. The RF signal generator transmits radio waves via the antenna. The receiver/signal detector is there to receive the signal from the RFID tag. The microcontroller is there to process this incoming data from the tag.
5) Tags consist of a transponder, rectifier circuit, controller, and memory. The transponder receives the signal from the reader and sends a signal back to the reader in response. The rectifier circuit allows for the power from the reader to be used as a supply for the controller and memory in the tag. The controller processes and manages the communication from reader to tag. And lastly, the memory stores the user-defined identification data.
6) RFID does not require the reader to be in line of sight of the tag and instead only needs to be within range making this technology less human resource intensive than barcodes.

The RFID Module chosen for this project is the Electhouse PN532 NFC RFID module V4. Using the PVC Mifare tag that was purchased alongside the module the RFID module will have a range of about 6-8cm. It has support for I2C, SPI and HSU (High Speed UART). For this project data is transmitted via HSU as ASCII with a 115200 baud rate, 8 bits, no parity, 1 stop configuration.

B. Distance Sensor - LiDAR

The LiDAR sensor is our main range sensor for creating and scanning our surroundings for SLAM. LiDAR has the characteristics of high accuracy, long range, strong anti-interference, independent of ambient light, etc. It can effectively sense the environmental situation both day and night[6]. Specifically, we will be using the RPlidar A1M8 2D sensor, designed to emit only a single beam onto the target object to collect important data on the X and Y axes. It is an intelligent sensor used for detecting objects in the radar surrounding area and completing the sensing and 2D positioning[6]. In order to properly use slam_toolbox, it needs to provide information from the scan topic to generate the necessary environmental variables for navigation.

C. Single-Board Computer

For our project the Single-Board Computer will be used to run Robotic Operating System 2 (ROS 2) and thus SLAM Toolbox, and NAV 2 that will allow our robot to achieve its autonomous mapping. These will be explained more in the Software Details section.

The Raspberry Pi 5 is the most advanced model for the Raspberry Pi. It provides various capabilities that make it the top choice for our project.. We thought this model would be best mainly for taking on the computational load of VSLAM without compromising on performance. The Raspberry Pi 5 has a quad-core processor at 2.4 GHz, offering a significant performance boost over prior models to the Pi, along with up to 8GB of RAM, a new VideoCore VII GPU, and faster USB 3.0 and microSD interfaces. This will allow us to completely leave the navigation processing and obstacle avoidance capabilities to our Raspberry Pi without straining our other components.

D. Camera

Raspberry Pi V2 Camera is the camera that will be used throughout the operation of our robot. It is a compact, and low-cost monocular camera that is specifically designed

for the Raspberry Pi. It contains an 8-megapixel Sony IMX219 image sensor capable of capturing high-resolution still images at 3280 x 2464 pixels and video at up to HD video and is suitable for low-end imaging applications. The camera connects via a MIPI CSI-2 interface, which is the main interface of the Video Core GPU (Raspberry Pi's graphics component). It uses a fixed-focus lens and has a rolling shutter [A.21], which is generally sufficient for static or low-motion areas which is ideal since SLAM would operate mostly in an indoor environment.

In the context of our project, the camera will be used as a guideline for the user to be able monitor the robot's movements via our mobile app's interface. The user will be able to use the images to monitor the robot's route and identify any objects or obstacles along the way.

E. Inertial Measurement Unit

An inertial measurement unit (IMU) is necessary to provide motion tracking data for our VSLAM algorithm. It consists of a sensor suite of a gyroscope, accelerometer, and magnetometer sensors. The IMU in this way is can provide info to the the algorithm with regards to reference to the rate of angle (gyroscope) and acceleration (accelerometer) on the x-, y-, and z-axes and, also, the magnetic field around the device (magnetometer) [A.65]. The reasoning for using an Inertial Measurement Unit for this project was mainly for estimating structure and the current positioning for the robot. This would allow us to get even more accurate readings when the robot is traversing the environment.

A 6-axis IMU consists of a 3-axis gyroscope and a 3-axis accelerometer. The gyroscope measures angular velocity, allowing the system to detect rotational movement, while the accelerometer measures linear acceleration, helping to track movement along the different axes. This combination enables the robot to estimate orientation changes and movement, which in turn aids in stabilizing the robot and detecting direction shifts. A 6-axis IMU would detect the robot's motion in all directions: forward, backward, left, and right. It also provides essential data for ORB-SLAM, ensuring smooth and accurate navigation.

For this project we are using the IIM-20670 because it is the only IMU available with pins on the side and not the bottom.

F. Motors

The motors provide the robot's driving force. For this project, brushed DC motors were selected for their simplicity, ease of implementation, and low cost. Specifically, the CQRobot 70:1 12V gear motors were selected for their balance between torque, speed, and efficiency. Each motor needs a 12V pulse width modulation (PWM) signal as an input, with the duty cycle controlling the speed of the motor.

The 70:1 gear ratio of the motors provides a high torque output capable of carrying the weight of the chassis and payload while maintaining a stable speed. Each motor has a built-in motor encoder that is used for feedback and monitoring the speed of each motor to ensure both motors are rotating at the same speed.

G. Motor Drivers

To drive the DC motors, a motor driver or H-bridge integrated circuit (IC) is required. Motor drivers act as the interface between the MCU and the motors and enable precise control over the speed and direction of the motors. For this project, the VNH5019 motor drivers were selected due to their high current capability and precise control features. The VNH5019 can control the speed and direction of the motors, via PWM and digital control signals from the MCU, respectively.

This motor driver takes in 12V directly and uses the PWM signal from the MCU to create a 12V PWM signal for the motors.

H. Microcontroller

The microcontroller (MCU) serves as the central processing and control unit of the robot, handling communication between sensors, motor drivers, and the Raspberry Pi 5. Unlike general-purpose processors, microcontrollers are made for real-time and low latency operation, allowing precise timing control of hardware components and real-time processing of various sensor inputs. In this project, the MCU is responsible for interpreting sensor data, controlling and adjusting the motors via PWM and digital I/Os through the motor driver boards, and maintaining reliable communication with the Raspberry Pi 5 and the mobile app to ensure proper autonomous navigation and monitoring.

After evaluating different MCUs, the ESP32-WROOM-32E was selected as the system's primary microcontroller. Its 240 MHz dual-core processor is ideal for real-time operations, while its built-in WiFi module allows wireless communication for the mobile device. The ESP32 also offers an abundant number of GPIO pins, PWM channels, and multiple serial interfaces.

I. Power Supply

The power supply is a critical component of the robot, responsible for providing the necessary energy for the motors, microcontroller, sensors, and other subsystems. A

stable and reliable power source ensures consistent performance from the robot. For this project, a 12V sealed lead-acid battery was selected as the main power source due to its safety, durability, and cost-effectiveness. Lead-acid batteries are robust and can handle the high currents the system may draw.

Table 1: Power Distribution Table

Component	Operating Voltage (V)	Max Current (A)	Max Power (W)
ESP32	3.3	0.5	1.65
Raspberry Pi 5	5.0	5.0	25.0
VNH5019 (x2)	3.3	0.016	0.0528
DC Motors (x2)	12	11	132
RFID Module	3.3	0.12	0.396
IMU	3.3	0.01	0.033
Total		**16.646**	**159.1318**

As shown in the power distribution shown in *Table 1*, the system will at most draw 16.646A of current. For our system, a battery with a capacity of 12Ah was selected. 12Ah would let the system run for 0.72 hours, or approximately 43 minutes under the greatest load. Under more realistic conditions, the system would not draw the maximum current and would have a higher run time.

J. Voltage Regulation

Stable voltage regulation is essential for ensuring reliable and stable operation of the different subsystems. As shown in *Table 1*, the components operate at different voltage levels, requiring efficient conversion from the 12V supply provided by the battery. Two dedicated DC/DC converters were used to regulate the voltage: the LM2576 buck regulator fixed at 3.3V, and the LM2679 adjustable buck regulator configured for a 5.2V output dedicated to the Raspberry Pi 5.

The LM2576 regulator was selected to supply 3.3V for the ESP32 microcontroller, sensors, motor driver logic, and peripheral modules. It can deliver up to 3A of continuous current, providing sufficient capacity for all components.

The LM2679 adjustable regulator was selected to provide a 5.2V output for the Raspberry Pi 5. Although the Raspberry Pi 5 nominally operates at 5V, it will shut off if the voltage drops below ~4.65V. The slightly higher 5.2V output prevents undervoltage and ensures the Raspberry Pi 5 does not shut off. It can deliver up to 5A of current, making it ideal for the Raspberry Pi 5.

IV. Hardware Detail

The electronic hardware design of the robot consists of multiple custom printed circuit boards (PCBs) that form the foundation of the system. Each PCB was developed to handle specific functions, ensuring efficient power management, reliable motor control, and stable communication between subsystems.

A. 3.3V Regulator Board

This board utilizes the LM2576-3.3V switching buck regulator to provide a stable 3.3V output for the low-voltage components, namely the ESP32 and various sensors. An LED is featured on the board as a visual indicator of operation.

B. 5.2V Regulator Board

The LM2679 adjustable switching buck regulator was used to supply a stable, regulated 5.2V output solely for the Raspberry Pi 5. An LED is featured on the board as a visual indicator of operation.

C. Motor Driver Boards

Two dedicated motor driver PCBs were designed around the VNH5019 full-bridge motor driver IC, each capable of driving a single one of the two 12V DC motors. These drivers support high current output and include integrated safety measures, such as thermal shutdown, undervoltage, and overcurrent protection. Each motor driver board routes the encoders from the motors to a connector to the ESP32, as well as the PWM and digital directional signals. A pair of LEDs are integrated on each of the boards to show the direction the motor is moving in, clockwise or counterclockwise.

D. Motor Driver Boards

The mainboard serves as the central hub of the system, holding the ESP32 and providing connections for all the other PCBs and sensors, interfacing everything together. It integrates communication lines for motor control, routes

the power input from the battery to the voltage regulator PCBs, data interfaces for the RFID reader and IMU, and communication between the Raspberry Pi 5 and the ESP32.

V. Communication

A. ESP32 and Raspberry Pi

The ESP32 and Raspberry Pi communicate over a wired Universal Asynchronous Receiver/Transmitter or UART connection using the microROS framework. UART is a two-wire serial hardware communication protocol that uses an asynchronous serial communication which means that no clock signal is used to synchronize the outgoing transmission to the receiving device. MicroROS is a set of software libraries that enables development of robotic applications to be deployed onto microcontrollers. It provides a bidirectional communication that is designed to work with the Robot Operating System (ROS 2). On the ESP32 there will be a MicroROS Node that connects to the MicroROS Agent on the Raspberry Pi over UART. The ESP32 node will publish the IMU and wheel encoder data to the Raspberry Pi's agent. The agent then properly subscribes and publishes to the ROS 2 topics. The ESP32 also publishes the location data so ROS 2 can know where to navigate to. ROS 2 will determine the directional commands that the robot will need to perform, publish this data to the command velocity topic and then the microROS agent will publish this to the ESP32 subscriber for the command velocity. From there the ESP32 will control the motors speed and direction.

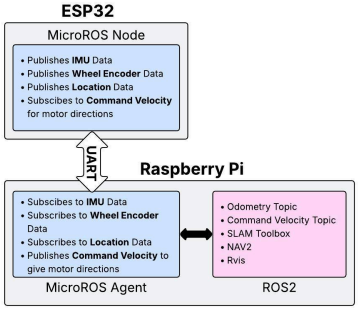

Fig. 3: Diagram of ESP32 and Raspberry Pi MicroROS Communication

B. ESP32 and IMU

The Inertial Measurement Unit being used for this project is the IIM-20670 which uses Serial Peripheral Interface (SPI). a synchronous communication protocol that is commonly used for sending data between microcontrollers and peripheral devices such as SD cards. SPI uses separate lines for data and a clock to keep both devices in sync. There can only be one controller device but there could be multiple peripheral devices. In this project only one peripheral device will be used.

C. ESP32 and RFID

The RFID module chosen for this project is the PN532, and we are using High Speed UART or (HSU) to communicate. The ESP32 serves as the interface between the PN532 and the main control system, receiving tag identification data through a serial connection configured at a baud rate of 115200. HSU was selected over I2C and SPI due to its reliable point-to-point communication and ease of implementation. Once a tag is detected, the ESP32 transmits the payload ID to the Raspberry Pi via the MicroROS UART connection.

D. App and ESP32

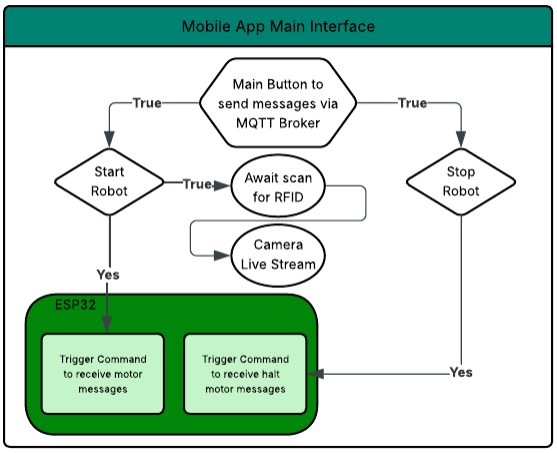

Fig. 4 Mobile App Interface

The App and ESP32 interact bi-directionally over the MQTT protocol. The communication is occurring over the same broker, enabling the ESP32 to send telemetry data to the App, and vice versa. The result is the ability to have bidirectional communication between both devices to perform critical operations such as starting and stopping autonomous operations with buttons. Ideally, we plan to transmit telemetry data in low-bandwidth packets. We believe this will prevent potential performance degradation between the Hardware and Software. The particular messaging format will occur asynchronously as mentioned by using two buttons: **Start Robot & Stop Robot.**

Start Robot will send a packet from the App to the ESP32 informing the Robot to begin autonomous operations. Afterwards, the ESP32 begins sending data from other Hardware components in packets to the app,

processing the information. Next we determine what operations to perform based on the received packet. For example, in our design we opted to use a notification system for the user, following traditional standards in mobile-app design. This information in our case is the RFID tags that were read and processed. After being successfully scanned, we publish a message (Tag-ID) over the network, back to the App being displayed as a notification message informing the user their package has been processed. This continues asynchronously until all the required packages are scanned and processed, or when the **Stop Robot** button is pressed.

Stop Robot does not terminate all robotic operations. It instead slows down the motors. This feature is designed for greater control over the robot's autonomous navigation. Allowing the user to stop it from moving if it is traveling off course, by pressing this button. Only **Start Robot** can undo this operation and resume further autonomous navigation.

E. App and Raspberry Pi

The mobile app to Raspberry Pi interactivity will not be using MQTT. It is assumed that once **Start Button** is pressed that our other features will run as anticipated. Furthermore, the Raspberry Pi only provides livestream service to the mobile app. Due to this, it is unnecessary to dedicate the same implementation of the MQTT protocol for the ESP32 to App interactivity with Raspberry Pi to App. Additionally, we want to prevent the Pi from taking on too many network features to prevent it from overworking itself. Thus, the only feature made will be a single-viewer livestream from the Pi-cam to the Apps side using third-party libraries such as WebRTC.

VI. Software Detail

A. System Firmware

The system firmware of our project, serves as the central control hub for the entire system. Running on an ESP32 microcontroller, this firmware acts as the robot's control center coordinating the communication, control, and decision-making processes that allow the robot to operate autonomously and safely. Its primary function is to manage sensor inputs, compute odometry for navigation, communicate with the App via Wi-Fi and communicate with the onboard Raspberry Pi via micro-ROS, ensuring the robot's movements and tasks are synchronized in real time.

At the core of the firmware is a state machine, which governs the overall flow of the robot's operation. The state machine organizes the robot's behavior into distinct operational states: idle, rfid, normalMove, obstacle, rerouting, and removePayload. We transitioned between them based on sensor data and received commands from NAV 2.

The setup() function is responsible for initializing all necessary hardware and communication components before the robot begins operation. This includes configuring GPIO pins for motor control and encoder inputs, setting up the IMU (Inertial Measurement Unit) for orientation data, and initializing the RFID reader for payload detection. The firmware also establishes Wi-Fi connectivity and connects to the micro-ROS agent acting as a communication bridge between the ESP32 and the Raspberry Pi. Within setup, the firmware initializes all publishers, subscribers, and timers required for the robot's real-time control loop.

The loop() function continuously cycles through the robot's operational logic. It runs the state machine to determine which actions to execute based on current conditions and received messages. Simultaneously, it runs the executor functions to allow micro-ROS to process incoming and outgoing data asynchronously. This ensures real-time responsiveness, enabling the robot to react quickly to new navigation commands or obstacle detections. Finally, the firmware performs odometry calculations using feedback from the wheel encoders. By measuring the change in encoder tick counts over time, it computes the robot's linear and angular displacement. These values are then integrated to estimate the robot's position and orientation within the environment. The calculated odometry data is published to the ROS network, where it contributes to localization and path planning. Overall, the ESP32 firmware serves as the intelligent control layer that harmonizes sensing, communication, and actuation for fully autonomous operation.

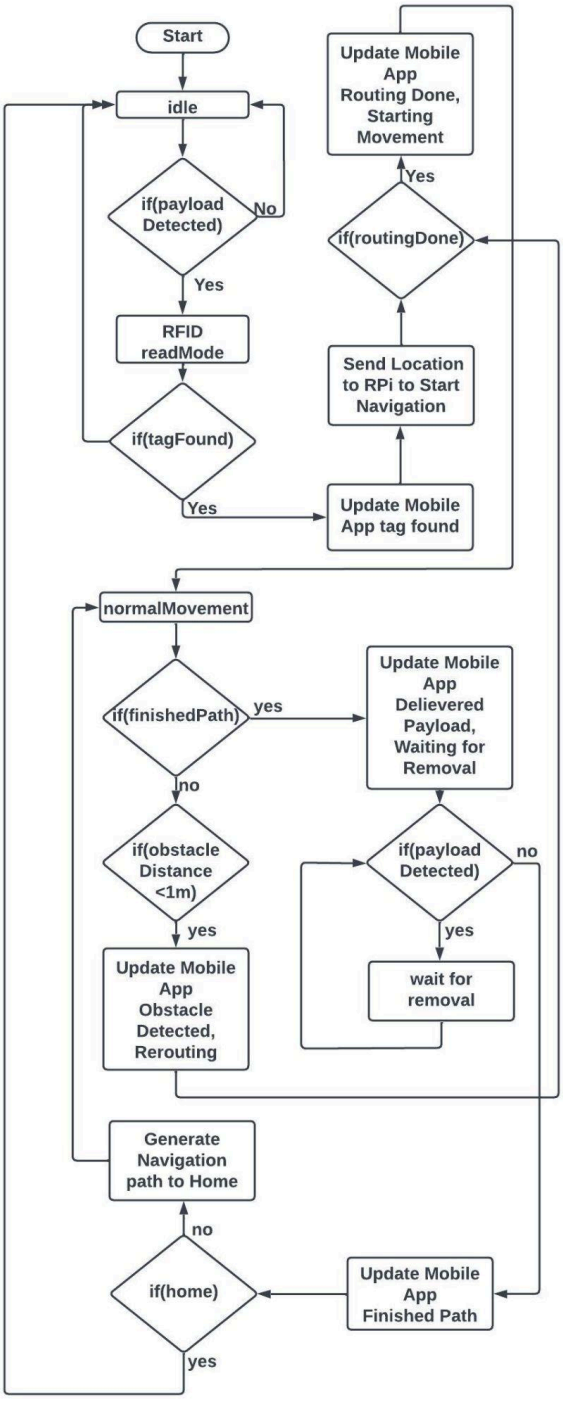

Fig. 4: Microcontroller State Diagram Flowchart Figure

B. Robot Operating System

Our Raspberry Pi 5 will be handling the autonomous navigation as well as interfacing with the robot via the ESP32. To start off any robotics system it will require the use of Linux operating system, specifically Ubuntu 24.04 which is supported with the distribution for our Robot Operating System (ROS). ROS is a framework that will contain the selection of software libraries that we need to create our robot. It provides a framework to create and connect components like actuators, sensors and control systems using ROS tools called topics and messages. This framework uses a distributed architecture, with the ROS computation graph.

There are two major versions, namely ROS and ROS 2. ROS is the older version, and it is still widely used. ROS 2 is a rewrite of ROS that is designed to be more reliable, scalable, and secure[4]. Since we are using the Raspberry Pi 5 it was ideal to utilize ROS2 as it was a more stable and scalable version than the opposing ROS. In addition to ROS there are different distributions that are supported with various versions of Ubuntu. Ubuntu 24.04 is used alongside ROS2 Jazzy Jalisco which comes with its most recent versioned libraries. [5]

When working with any ROS system when a library is in use it has the ability to publish from a ROS node with information (hardware or software) to a topic then it subscribes to retrieve the information for processing. Our ESP32 uses a MicroROS node that will be publishing the data from our motor encoders and IMU to the Raspberry Pi for working with SLAM and our path planning algorithm. The Raspberry Pi and ESP32 will then subscribe to these nodes and use this information to calibrate the robot's operating parameters. See Fig.3 for more information.

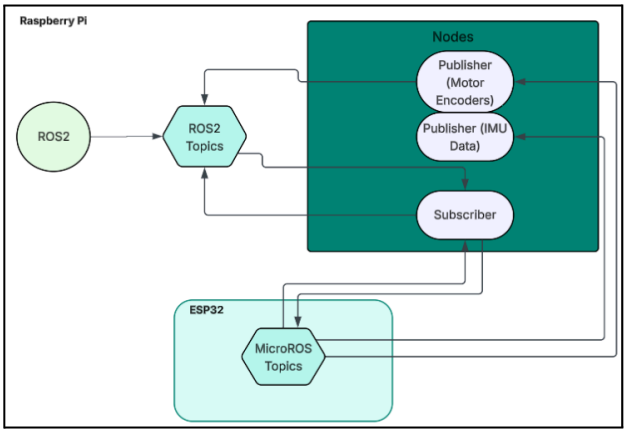

Fig. 4: Diagram of Overview ROS2 Interaction

C. Lidar SLAM

SLAM will be our main focus for navigating within an indoor environment using localization and pre-mapping techniques. Without the use of SLAM, it will just move randomly within a room and may not be able to detect certain obstacles along the path. In addition, this approach

uses excessive power, so the battery will run out more quickly.

Robots with a SLAM algorithm can use sensor information such as the number of wheel revolutions and data from cameras and other imaging sensors to determine the amount of movement needed.The foundation of SLAM includes the use of in using landmark correlations, data association, and loop-closure to reduce the uncertainties about its previously visited area and poses.[A.48] They primarily use Kalman Filter or particle filters for sequential state estimation and an optimal state estimation technique in a linear system with also calculating around the Gaussian noise. Bayes rule will be mainly used in the state estimation as SLAM is creating the map. It states that a posterior probability of an event given an event is equal to the likelihood of given , multiplied with α, a normalization factor as shown in the equation below.

For our project we will be using slam_toolbox which will implement our 2D Lidar sensor to gather distance data to generate the map itself. We will be using LiDAR to assess the environment. The value of intensity indicates the surface reflectance structure of barriers whereas the distance measurements are computed by time of flight [A.66].

D. Nav2 and Dynamic Window Approach

To implement the autonomous navigation aspect of our project, we will be using ROS2 Navigation Stack Nav2 provides perception, planning, control, localization, visualization and more in terms of autonomous navigation. It will compute an environmental model from raw or pre-processed laser sensor and semantic data, dynamically route a path through the environment, compute feasible motor commands, avoid obstacles, and structure higher-level robot behaviors[7]. The Dynamic Window Approach or specifically the default DWB controller within Nav2 allows the robot to successfully avoid obstacles along its route. In particular, DWA's infers on the best path from desired goal position, distance to obstacles, and robot speed all based on their fixed weight coefficient.

E. Mobile App

The mobile app is written in Flutter (Dart) for this project. Originally we wanted to have the ability to run the app inside iOS and Android environments. However, now we only deploy it inside an Android environment. This was because the barriers with iOS and requirements to properly execute a mobile app was too time consuming. We would be required to ensure the app supported both environments dependencies, and this was not practical for our design goals regarding time-constraints.

Moreover, the app is designed to be intuitive for people unfamiliar with PARCEL and its features. Buttons are color-coded to match interpreted messages like our use of Green and Red for **Start Robot & Stop Robot**. We can infer which button is colored based on the wording (**Start Robot** is Green and **Stop Robot** is Red). The livestream is positioned to the left-hand side of the Dashboard, and equally distant from the **Start/Stop** buttons. In addition, the use of other features to notify current connectivity being a **Robot-Connected** banner. These banners imply what operations should occur. **Start/Stop** should trigger **Robot-Connected**. If executed properly this color will resemble a calming color (ex. Gold, light green, etc). However, if an error occurs in the Network then **Robot-Connected** is highlighted to be Red. This allows us and the users to better understand the current states of the robot, and provides greater trust in the reliability of the robot.

Acknowledgement

The authors wish to acknowledge the assistance and support of Dr. Chan and Dr. Weeks as well as our review committee Dr. Mike Borowczak, Dr. Avra Kundu, Dr. Sonali Das.

References

[1]Kate, “Time of Flight Sensor vs. LiDAR: What Are the Differences?,” Precision Measurement Technologies, Mar. 06, 2024. https://pmt-fl.com/time-of-flight-sensor-vs-lidar-what-are-the-differences/ (accessed Feb. 18, 2025).

[2]X. Zhang, H. Zhang, C. Qian, B. Li, and Y. Cao, “A LiDAR-intensity SLAM and loop closure detection method using an intensity cylindrical-projection shape context descriptor,” International Journal of Applied Earth Observation and Geoinformation, vol. 122, p. 103419, Jul. 2023, doi: https://doi.org/10.1016/j.jag.2023.103419.

[3]Basheer Al-Tawil, T. Hempel, A. Abdelrahman, and Ayoub Al-Hamadi, “A review of visual SLAM for robotics: evolution, properties, and future applications,” Frontiers in robotics and AI, vol. 11, Apr. 2024, doi: https://doi.org/10.3389/frobt.2024.1347985.

[4] “Overview of Robot Operating System (ROS) - MATLAB & simulink,”MathWorks, https://www.mathworks.com/help/ros/gs/robot-operating-system-ros-basic-concepts_mw_1b4db38c-9199-4217-82cc-3680b8dda407.html

[5]“Distributions,”Distributions ROS 2 Documentation: Rolling documentation.:https://docs.ros.org/en/rolling/Releases.html

[6] “2D LIDAR,” Leishen Intelligent System, https://www.lslidar.com/products/2d-lidar/

[7] “NAV2 - NAV2 1.0.0 documentation,” Nav2 Navigation System, https://nav2.org/