University of Coimbra

Embedded Systems - Project Report

Search and Rescue Robot

Miguel Meireles Teixeira

José Pedro da Cunha Rodrigues

Ezequiel Juan Armando Alves Flores Rojas

Class: PL1
Head Teacher: Lino José Forte Marques

2023/2024

Contents

1 Definition of architecture and task allocation

1.1 System Engineering V Diagram

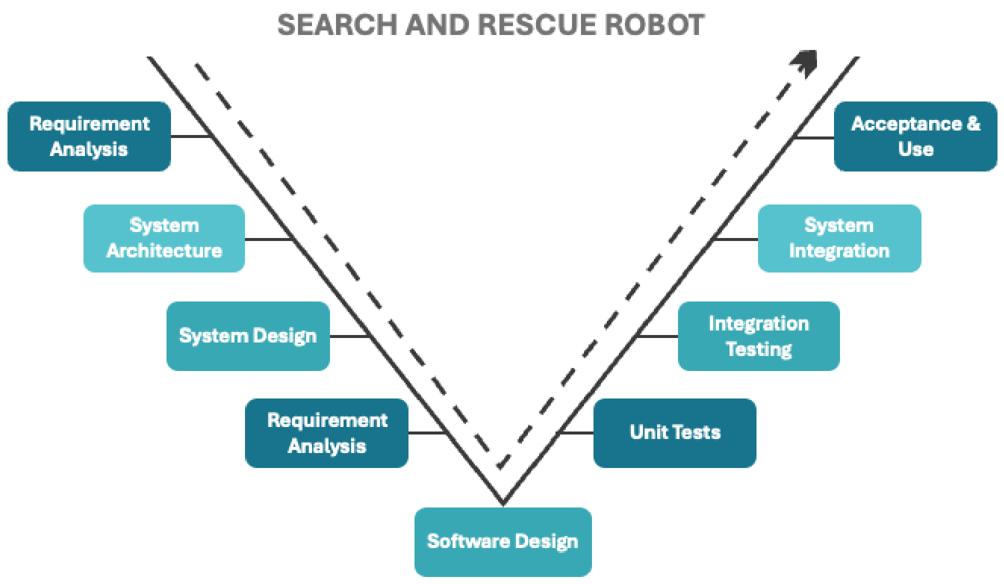


Figure 1: Search and Rescue Robot System Engineering V Diagram

1.2 Tasks

1.2.1 Requirements Analysis

- **Gathering Requirements:** Understand the mission and constraints of the search and rescue robot. Research existing solutions for insights.
- **Defining Functional Requirements:** Specify what the robot must do, including moving, detecting RFID tags, avoiding obstacles, and reporting positions.
- **Defining User Interface:** Identify the requisite functionalities for the User Interface, including real-time monitoring of robot status, visualization of RFID tag locations, and remote control command facilitation.

  This task was performed by: Miguel Teixeira, José Rodrigues and Ezequiel Rojas.

1.2.2 System Architecture

- **Selecting Hardware:** Choose sensors, motors, a microcontroller, communication devices that meet the functional requirements and a wireless camera module to live stream the robot rescue.

- **High-Level Design:** Create block diagrams to show how hardware components interact with STM32F411CEU6 and ESP32-WROOM-32D.

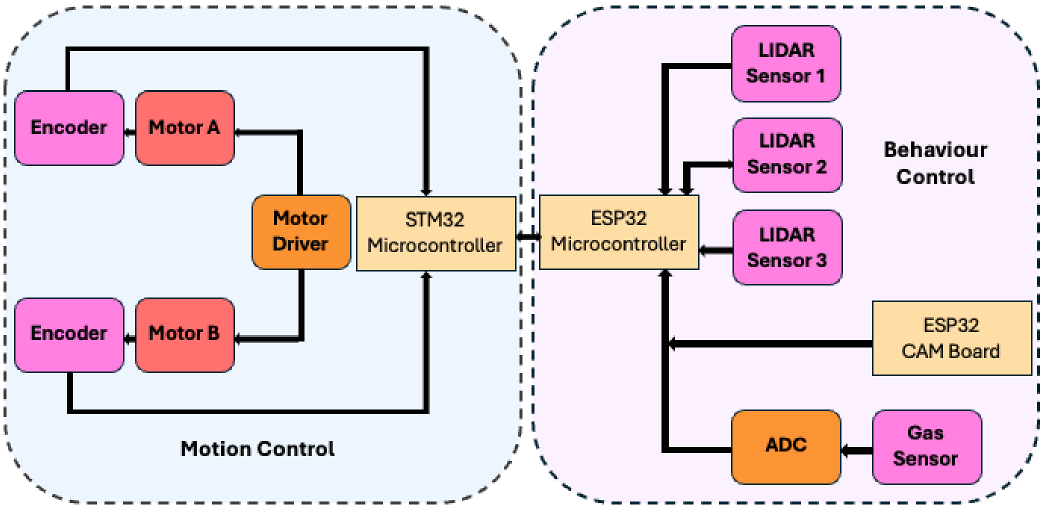


Figure 2: System Architecture

Table 1: Componentes

Categoria	Componente	Link de Referência
Motion Control	Micro-controlador	STM32F411CEU6
	Motor Driver	TB6612
	Motors	Micro Metal Gearmotors (210:1)
	Encoders	Quadrature Magnetic Encoder (7CPR)
Behaviour Control	Micro-controlador	ESP32-WROOM-32D
	Sensor 1	Sharp GP2Y0A21Y
	Sensor 2	VL53L0X-V2
	Sensor 3	Sharp GP2Y0A21Y
	Sensor 4	VL53L5CX (ToF 8x8)
	ADC	ADS1115

- **Defining Interfaces:** Specify the communication protocols between different hardware components and the microcontrollers.

  We will use System Peripheral Interface (SPI).

This task was performed by: Miguel Teixeira and José Rodrigues.

1.2.3 System Design

- **Detailed Component Layout:** Design the physical layout of all the components within the robot's chassis.

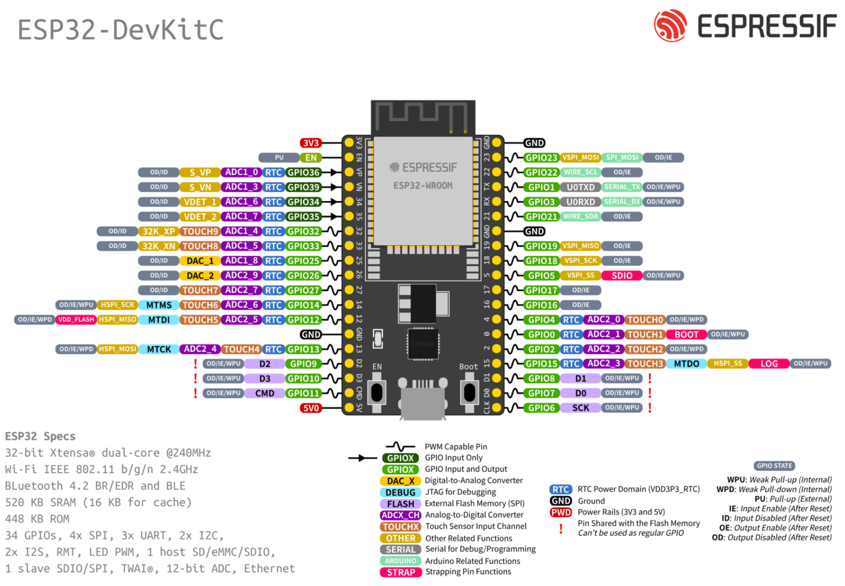

ESP32-DevKitC
ESPRESSIF
ESP32-WROOM
EN
Boot
ESP32 Specs
32-bit Xtensa® dual-core @240MHz
Wi-Fi IEEE 802.11 b/g/n 2.4GHz
BLuetooth 4.2 BR/EDR and BLE
520 KB SRAM (16 KB for cache)
448 KB ROM
34 GPIOs, 4x SPI, 3x UART, 2x I2C,
2x I2S, RMT, LED PWM, 1 host SD/eMMC/SDIO,
1 slave SDIO/SPI, TWAI®, 12-bit ADC, Ethernet
PWM Capable Pin
GPIOX GPIO Input Only
GPIOX GPIO Input and Output
DAC_X Digital-to-Analog Converter
DEBUG JTAG for Debugging
FLASH External Flash Memory (SPI)
ADCX_CH Analog-to-Digital Converter
TOUCHX Touch Sensor Input Channel
OTHER Other Related Functions
SERIAL Serial for Debug/Programming
ARDUINO Arduino Related Functions
STRAP Strapping Pin Functions
RTC RTC Power Domain (VDD3P3_RTC)
GND Ground
PWD Power Rails (3V3 and 5V)
! Pin Shared with the Flash Memory Can't be used as regular GPIO
GPIO STATE
WPU: Weak Pull-up (Internal)
WPD: Weak Pull-down (Internal)
PU: Pull-up (External)
IE: Input Enable (After Reset)
ID: Input Disabled (After Reset)
OE: Output Enable (After Reset)
OD: Output Disabled (After Reset)

Figure 3: ESP32-DevKitC Pin Layout

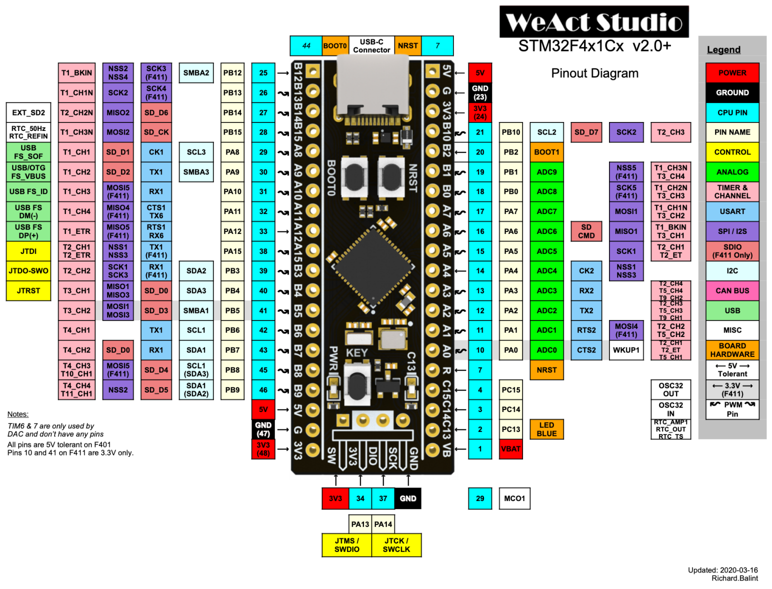


Figure 4: STM32F411CEU6 Pin Layout

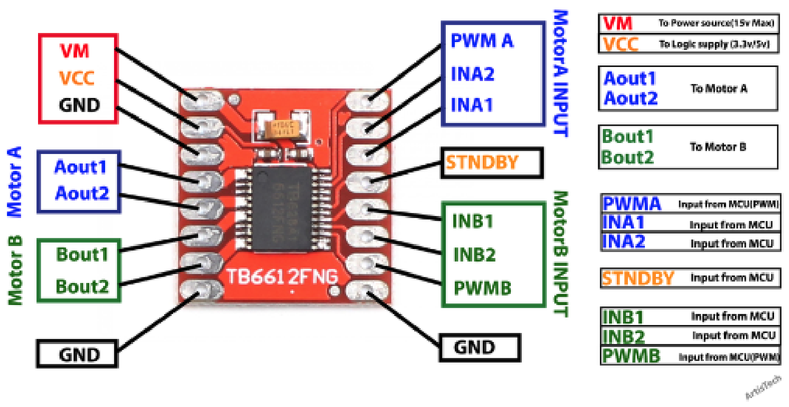


Figure 5: TB6612FNG Pin Layout

- **Wiring Schematics:** Create detailed wiring diagrams for electrical connections between components.

Table 2: Wiring Schematics

STM32 -> ESP32	
STM32	ESP32
PA7	23
PB4	19
PB3	18
PA4	5
PB0	4
Motor Driver -> STM32	
Motor Driver	STM32
PWMA	PB8
AIN1	PA15
AIN2	PB7
STBY	PB1
PWMB	PB9
BIN1	PB6
BIN2	PB5
Sensor1 -> ESP32	
Sensor1	ESP32
Data	VP
Sensor2 -> ESP32	
Sensor2	ESP32
SCL	33
SDA	32
XSHUT	17
Sensor3 -> ESP32	
Sensor3	ESP32
Data	34
ADC -> ESP32	
ADC	ESP32
SCL	22
SDA	21
Encoder Motor A -> STM32	
Encoder Motor A	STM32
C1	PA0
C2	PA1
Encoder Motor B -> STM32	
Encoder Motor B	STM32
C1	PA8
C2	PA9

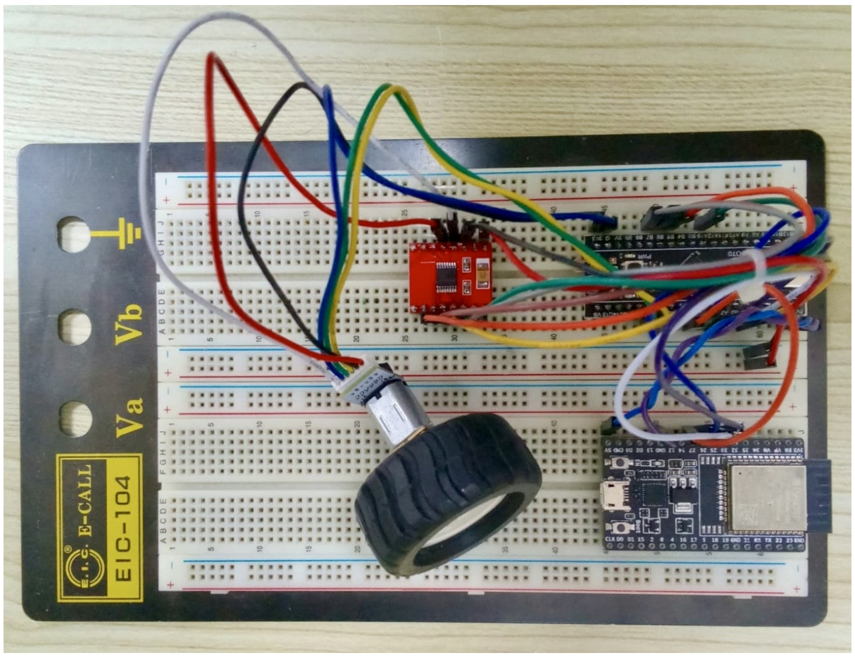


Figure 6: Initial Circuit Assemblage

This task will be performed by: Miguel Teixeira.

1.2.4 Software Architecture

- **Module Breakdown:** Identify distinct software modules, such as navigation, RFID detection, and communication.
- **Layered Architecture:** Define a layered software architecture.

  – We will use FreeRTOS as it allows the creation of independent tasks that can be executed simultaneously.
  – Semaphores to synchronize access to shared resources.
  – Interrupts tor handle real-time events and for managing input data from sensors.
  – SPI for communication between STM32 and ESP32.
  – Using Timers to count encoder pulses in order to measure the distance traveled with the Search and Rescue Robot.

This task was performed by: Miguel Teixeira and José Rodrigues.

1.2.5 Software Design

- **Algorithm Design:** Develop algorithms for autonomous navigation, obstacle avoidance, and efficient search patterns.
- **Interface Design:** Design interfaces for modules to interact with each other and with hardware components.
- **User Interface:** Design a user interface for monitoring and controlling the robot remotely.

This task was performed by: Ezequiel Rojas and José Rodrigues.

1.2.6 Unit Tests

- **Test Case Development:** Create specific test cases for each function within a module to validate its correct behavior.
- **Documentation:** Record all test results and anomalies for further investigation and fixing.

This task was performed by: Ezequiel Rojas.

1.2.7 System Integration

- **Physical Assembly:** Assemble the hardware components following the system design.
- **Functional Verification:** Verify that each component operates correctly within the system.
- **Subsystem Integration:** Integrate subsystems step-by-step, testing their interactions.

This task was performed by: Miguel Teixeira, José Rodrigues and Ezequiel Rojas.

1.2.8 Integration Testing

- **Integration Test Plan:** Develop an integration test plan with different scenarios.
- **Performance Testing:** Test the Search and Rescue Robot performance, including response times and accuracy, under varying conditions.
- **User Scenario Testing:** Simulate scenarios that the robot may encounter during a search and rescue mission.

This task was performed by: José Rodrigues

1.2.9 Acceptance & Use

- **User Acceptance Testing (UAT):** Conduct tests to ensure the robot meets expectations and requirements.
- **Field Testing:** Test the robot in an environment that closely resembles the final operating conditions.
- **Final Demonstration:** Prepare and conduct a final demonstration of the robot's capabilities.

This task was performed by: Miguel Teixeira, José Rodrigues and Ezequiel Rojas.

1.3 Additional Ideas

1.3.1 User Interface

The user interface will be designed to visually represent the movement and location of the robot within a predefined area using a two-dimensional XY graph. This graphical display will help users quickly understand the robot's current and past positions, making it easier to monitor and control the robot's path and activities.

- **Real-Time Position Tracking:** The UI will update in real-time as the robot moves, showing the robot's current position as a distinct point on the graph.
- **Historical Path Visualization:** Users can see the path that the robot has taken over time, which can be highlighted with lines or dots connecting the sequence of positions.

This task was performed by: Miguel Teixeira and Ezequiel Rojas.

1.3.2 Live Stream Camera

The robot will have an ESP32-CAM AI-Thinker to provide live video streaming capabilities. This feature enables users to receive real-time visual feedback from the robot's environment.

In addition to visual monitoring, users will be able to control the robot directly through keyboard inputs. This control mechanism will be integrated into the User Interface. The keyboard controls will allow users to command the robot by moving forward or backward, turning, and adjusting speed.

This task was performed by: Miguel Teixeira and José Rodrigues.

2 Final Project Review

2.1 Sensors

2.1.1 Left LIDAR Sensor

This sensor is positioned on the left side of the robot and is responsible for detecting obstacles on the left. It continuously scans the environment and provides distance measurements to the control unit.

2.1.2 Middle LIDAR Sensor

Placed at the front center of the robot, the middle LIDAR sensor is crucial for detecting obstacles directly ahead. Unfortunately, we were unable to establish communication with this sensor using I2C, rendering it non-functional in our setup. Consequently, the middle LIDAR sensor was not used in our obstacle detection system.

2.1.3 Right LIDAR Sensor

The right LIDAR sensor is mounted on the right side of the robot. It ensures that obstacles on the right side are detected, contributing to the robot's ability to navigate safely.

2.1.4 RFID Tag Reader

The RFID tag reader is used to detect and identify RFID tags scattered within the environment. These tags represent victims that the robot needs to find and report.

2.1.5 ADC Gas Sensor

The ADC gas sensor detects the presence of specific gases in the environment. This sensor's data can be crucial for identifying hazardous conditions in search and rescue missions.

2.2 Odometry

Odometry involves using data from motion sensors to estimate the robot's change in position over time. This is essential for navigation and mapping the environment, ensuring the robot can accurately track its movement and position. The following is a detailed explanation of the implementation of odometry in our project:

2.2.1 Key Components and Variables

- **Encoders:**
  - Two encoders are used, one for each wheel of the robot. These encoders provide the number of pulses generated as the wheels rotate.
  - *Right_ Wheel_Encoder_value* and *Left_ Wheel_Encoder_value* are used to store the current encoder values for the right and left wheels, respectively.
- **Distance Calculation:**
  - *Distance_Right* and *Distance_Left* store the distances traveled by the right and left wheels, respectively. These are calculated using the number of encoder pulses and the circumference of the wheels.
  - Constants like *WHEEL_DIAMETER*, *WHEEL_RADIUS*, *WHEEL_DISTANCE*, and *PULSE_PER_REVOLUTION* are defined to facilitate these calculations.
- **Position Variables:**
  - x, y - Current coordinates of the robot.
  - *theta* - Current orientation angle of the robot in radians.
  - *d_theta_degree* - Current orientation angle of the robot in degrees.

2.2.2 Odometry Calculation

1. **Encoder Reading:** The current values of the encoders are read and stored in *Right_ Wheel_Encoder_value* and *Left_ Wheel_Encoder_value*. The encoder counts are reset after each reading to measure the change in position during the next cycle.
2. **Distance Calculation:** The distances traveled by the right and left wheels are calculated using the formula:
$$\text{Distance} = \frac{\text{Encoder Count} \times \text{Wheel Circumference}}{\text{Pulses per Revolution}}$$
These distances are stored in *Distance_Right* and *Distance_Left*.
3. **Position Update:** The change in the robot's position and orientation is calculated as follows:
$$d = \frac{\text{Distance\_Right} + \text{Distance\_Left}}{2}$$
$$d\_theta = \frac{\text{Distance\_Left} - \text{Distance\_Right}}{\text{WHEEL\_DISTANCE}}$$
The new position (x, y) and orientation (*theta*) are updated using the following equations:
$$x+ = d \times \cos(\theta + \frac{d\_theta}{2})$$
$$y+ = d \times \sin(\theta + \frac{d\_theta}{2})$$
$$theta+ = d\_theta$$

4. **Velocity Calculation:** The angular velocities of the wheels are calculated:

$$\text{angular\_vel\_left} = \frac{60 \times \text{Left\_Wheel\_Encoder\_count}}{0.001 \times \text{PULSE\_PER\_REVOLUTION}}$$

$$\text{angular\_vel\_right} = \frac{60 \times \text{Right\_Wheel\_Encoder\_count}}{0.001 \times \text{PULSE\_PER\_REVOLUTION}}$$

5. **Error Correction:** The errors in the desired and actual velocities are calculated and used to adjust the PWM signals to the motors. Proportional control is used for this purpose.

In summary, the odometry system is vital for tracking the robot's position and orientation, which is essential for accurate navigation and task execution in dynamic environments.

2.3 SPI Communication

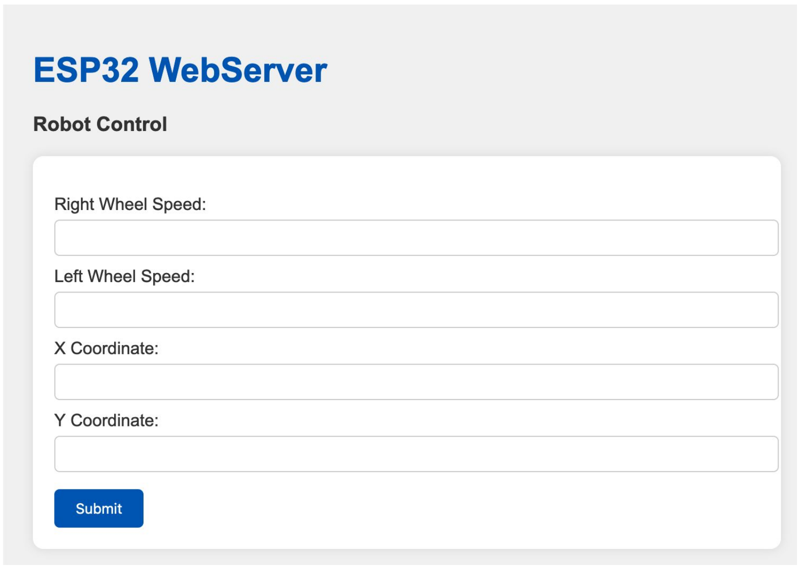
We utilized the Serial Peripheral Interface (SPI) technique for communication between the ESP32 and STM microcontrollers. This communication is crucial for the robot's operation. The following steps outline the implementation and functionality of the SPI communication in our project:

1. **Reading Sensor Values:** Functions were created on the ESP32 to read the values from the LIDAR sensors. These values are essential for detecting obstacles around the robot.
2. **Transmitting Data to STM:** The ESP32 sends the sensor data to the STM via SPI. This data transfer allows the STM to receive real-time information about the robot's surroundings.
3. **Processing Data on STM:** Upon receiving the sensor values, the STM processes this data to control the motors. This processing is vital for obstacle avoidance, as the STM adjusts the motor speeds and directions to navigate safely.
4. **Coordinate Transmission:** Additionally, we developed functions on the ESP32 to read the x and y coordinate values entered by a user on our website. These coordinates are sent to the STM, which uses them to navigate the robot to the desired location.
5. **Real-Time Tracking:** Functions were also implemented on the STM to send the robot's current coordinates back to our website. This feature ensures real-time tracking and updates, allowing users to monitor the robot's position continuously.

2.4 Web Server

The web server component is integral to the user interface and interaction with the robot. It allows users to input target coordinates, monitor the robot's status, and receive real-time updates. The following points highlight the key functionalities of the web server:

1. **User Interface:** The web server hosts a user-friendly interface where users can enter the target x and y coordinates for the robot. This interface is accessible from any device with internet connectivity.
2. **Coordinate Input:** Once the user inputs the desired coordinates, the web server sends these values to the ESP32. The ESP32 then transmits the coordinates to the STM, which navigates the robot to the specified location.


ESP32 WebServer

Robot Control

Right Wheel Speed:

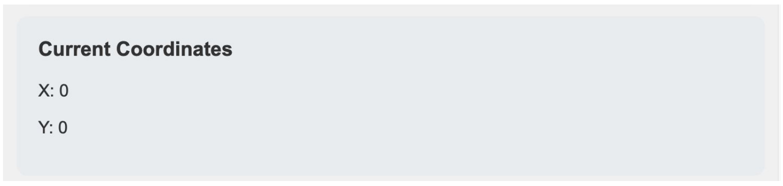
Left Wheel Speed:

X Coordinate:

Y Coordinate:

Submit

Figure 7: Input coordinates


Current Coordinates

X: 0

Y: 0

Figure 8: Coordinates

3. **Status Monitoring:** The web server continuously receives updates from the STM about the robot's current coordinates. This information is displayed on the interface, providing users with real-time tracking of the robot's position.

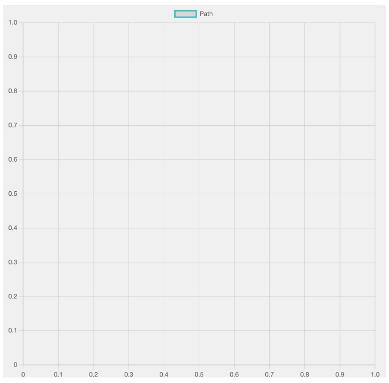


Figure 9: Tracking

4. **Real-Time Updates:** The implementation of real-time updates ensures that any changes in the robot's position are immediately reflected on the web interface. This feature is crucial for tasks requiring precise navigation and monitoring.

In summary, the SPI communication and web server components work together to provide seamless control and monitoring of the robot. The SPI interface enables efficient data transfer between the microcontrollers, while the web server offers an interactive platform for users to guide and track the robot's movements.

2.5 Collision Avoidance

In the collision avoidance system, the values read from the left and right sensors of the robot are transmitted via the ESP32 to the STM. The STM has a predefined safety distance. When the LIDAR sensors detect an obstacle at a distance shorter than the safety distance, the robot stops the right wheel and rotates the left wheel until it finds a spot where the values read by both the left and right LIDAR sensors are greater than the safety distance. Once this condition is met, the robot can continue moving towards the target.
Specifically, the process works as follows:

1. **Sensor Reading:** The left and right LIDAR sensors continuously monitor the distance to obstacles. These readings are sent from the ESP32 to the STM microcontroller.
2. **Safety Distance Check:** The STM compares the sensor readings to a predefined safety distance. This distance is a critical threshold to ensure the robot avoids collisions.
3. **Obstacle Detection:** If either the left or right LIDAR sensor detects an obstacle closer than the safety distance, the STM initiates an avoidance maneuver.
4. **Avoidance Maneuver:**
   - The robot stops the right wheel.
   - The robot rotates the left wheel to pivot away from the obstacle.
   - This rotation continues until both the left and right LIDAR sensors detect that the path is clear, i.e., the distance to the nearest obstacle is greater than the safety distance on both sides.
5. **Resume Movement:** Once the robot is in a clear area where the safety distance criteria are met, it resumes its path towards the target coordinates. The STM continuously monitors the sensors and adjusts the robot's path as necessary to avoid any new obstacles that appear.

This collision avoidance system ensures that the robot can navigate through its environment safely while avoiding obstacles and maintaining its course towards the specified target.

2.6 Move To Target

The "Move To Target" function is a crucial component of our robot, designed to navigate to specific coordinates entered by the user. This function integrates various aspects of control theory, odometry, and real-time computation to ensure accurate and efficient movement towards the target position. Below is a detailed explanation of its implementation:

2.6.1 Objective

The main objective of the "Move To Target" function is to allow the robot to navigate autonomously to a set of target coordinates (x, y) provided by the user. The robot uses real-time feedback from its sensors and encoders to adjust its path dynamically, ensuring it reaches the desired location accurately.

2.6.2 Key Components and Variables

- **Position Variables:**
  - x, y - Current coordinates of the robot.
  - $target\_pose\_x$, $target\_pose\_y$ - Target coordinates provided by the user.
  - $theta$ - Current orientation angle of the robot in radians.
  - d_theta_degree - Current orientation angle of the robot in degrees.
- **Velocity Variables:**
  - $right\_velocity$, $left\_velocity$ - Desired velocities for the right and left wheels.
  - $angular\_vel\_right$, $angular\_vel\_left$ - Measured angular velocities of the right and left wheels.
- **Control Gains:**
  - Kp_linear - Proportional gain for linear velocity control.
  - $Kp\_angular$ - Proportional gain for angular velocity control.
- **Distance Calculation:**
  - $Distance\_Right$, $Distance\_Left$ - Distance covered by the right and left wheels.
  - d, d_theta - Average distance and change in orientation computed from encoder readings.

2.6.3 Main Loop Functionality

1. **Read Current Position:** The current position (x, y, $theta$) is continuously updated based on encoder readings and distance calculations.
2. **Error Calculation:** Calculate the error in position ($error\_x$, $error\_y$) and orientation ($error\_theta$) with respect to the target coordinates.
3. **Distance and Angle to Target:** Compute the distance to the target and the required orientation angle to face the target ($target\_theta$).
4. **Control Law:** Use proportional control laws to compute the necessary linear ($linear\_velocity$) and angular ($angular\_velocity$) velocities:
   - $linear\_velocity = Kp\_linear \times distance$
   - $angular\_velocity =$ -$Kp\_angular \times error\_theta$

   These velocities are then converted to individual wheel velocities:
   - $left\_velocity = \frac{linear\_velocity}{WHEEL\_RADIUS}$ - $\frac{WHEEL\_DISTANCE}{2} \times \frac{angular\_velocity}{WHEEL\_RADIUS}$
   - $right\_velocity = \frac{linear\_velocity}{WHEEL\_RADIUS} + \frac{WHEEL\_DISTANCE}{2} \times \frac{angular\_velocity}{WHEEL\_RADIUS}$

5. **Motor Control:** Adjust the PWM signals to the motors based on the computed wheel velocities. The direction of each wheel is set using GPIO pins.
6. **Position Check:** Continuously check if the robot has reached the target within a specified tolerance. If the target is reached, the wheel velocities are set to zero.
7. **Real-Time Updates:** The control loop includes a delay (*HAL_Delay(100)*) to ensure it operates at a consistent rate, allowing for real-time adjustments based on sensor feedback.

In summary, the "Move To Target" function is a sophisticated control system that allows the robot to navigate to user-defined coordinates using real-time sensor feedback, proportional control algorithms, and precise motor control. This functionality is critical for the robot's autonomy and effectiveness in performing its tasks in dynamic environments.