Smart Robot Car for Autonomous Navigation: An IoT, MQTT, and Image Processing Approach

Author:
XUAN KY PHAM
STUDENT ID: 1649932

Supervisor:
DR. MELANIE OOI

THE UNIVERSITY OF WAIKATO
Te Whare Wānanga o Waikato

SCHOOL OF ENGINEERING

Department of Electrical and Electronic Engineering

UNIVERSITY OF WAIKATO

A thesis submitted to the University of Waikato in accordance with the requirements of the degree of MASTER OF ELECTRICAL AND ELECTRONIC ENGINEERING PRACTICE

JUNE 2025

Abstract

Robotic autonomous navigation is an important technology in modern robotics, enabling vehicles to operate without human intervention. This thesis illustrates the design and implementation of a Smart Robot Car that utilizes IoT, MQTT, and image processing to achieve real-time navigation and control. The system integrates multiple hardware components, including an ESP32, a Raspberry Pi, a GPS NEO-8M module, two BTS7960 motor drivers, and various sensors to establish a combination system for robot car operation. IoT communication is established using the MQTT protocol, allowing seamless data exchange between the robot car and a remote server for monitoring and controlling the robot car. A PWM control algorithm is implemented to regulate motor speed and steering based on sensor inputs. Additionally, computer vision techniques are employed for obstacle detection and shutting down the robot if the user sets any detection object in the safety range setting. Additionally, YOLOv10 or YOLOv11 uses an image processing algorithm to control robot operations using traffic signs. The project also incorporates a web-based interface, developed using Flask and Python, HTML, JavaScript, and PHP for real-time monitoring and remote command execution. The website was also designed to work on servers collecting data from the robot car and control it remotely. The system is tested in various conditions to evaluate its navigation accuracy, response time, and overall efficiency. Results demonstrate that the combination of MQTT-based communication and real-time image processing significantly improves the robot's ability to navigate autonomously. This research contributes to the field of IoT-based robotics by presenting a scalable and cost-effective approach to autonomous vehicle development. Future work may explore enhancements such as deep learning-based vision processing and cloud-based AI integration for improved decision-making and adaptability, as well as applying PID control algorithms to regulate motor speed for the robot as a future improvement. Finally, backup operation and control type are also implemented to prevent any problems with internet connection or Wi-Fi connection failure. For example, using PS2 Remote 2.4 GHz and LoRa together with a Python application, the data of the robot car can be sent over a LoRa signal to the server.

Dedication and acknowledgements

I would like to extend my heartfelt gratitude to my supervisor, Professor Mrs. Malenie Ooi, for her unwavering support, expert guidance, and encouragement throughout my research on the robot car. Her deep technical knowledge and insightful feedback have played a crucial role in shaping the direction of this dissertation

The authors also thank the Department of Electrical and Electronic Engineering of Waikato University for providing the necessary resources, Nicobot, and an excellent research environment to complete this project.

Having valuable feedback from laboratory staff and some tutors from Waikato University is crucial information that allowed me to update my operation program during the robot car project test. Their collaboration and insights have significantly enriched this research.

In addition, the support and motivation of my family are some of the biggest factors that enable me to thrive in the academic research project. Their belief in me has been a driving force in the successful completion of this work.

Finally, I acknowledge the contributions of the open-source community, such as the Python and Arduino libraries. These resources and shared knowledge in IoT, MQTT, and the Yolo community are invaluable in implementing the Smart Robot Car project.

Xuan Ky Pham July 2025

Author's declaration

I, Xuan Ky Pham, do solemnly declare that this thesis, "Smart Robot Car for Autonomous Navigation: An IoT, MQTT, and Image Processing Approach," is my work and has not been submitted, either wholly or partially, for any other academic qualification or degree at any other institution or university.

This work was for the Master's degree in Electrical and Electronic Engineering Practice at Waikato University, New Zealand. The contents of this thesis are the result of my studying resources from many places and contributions from Students who made NICOBOT to give me a good robot for improving the control system, except where due acknowledgment of the work of other authors has been mentioned. Due citations and acknowledgments have been made wherever required.

I declare that this thesis conforms to Waikato University's ethical standards and academic integrity processes. I accessed any experimental data, software, or technical packages presented in this research through my research work, and all external sources of help have been properly acknowledged. In addition, this thesis is formatted using https://overleaf.com. Grammarly also helps me correct grammar and the English language and paraphrase some paragraphs from citation documents. I also drew pictures for this thesis using Microsoft Paint and graphs using https://app.diagrams.net/. Finally, I used https://quillbot.com/, which is a generative-AI platform used to correct sentences or rewrite my sentences with incorrect structure. OpenAI is used to learn about thesis structure style. For coding, I learned from many websites separately, and the complete code of NICOBOT was written by me, including C, HTML, JavaScript, PHP, Python, etc. I assume full responsibility for any errors or omissions in this document.

Signature: Xuan Ky Pham Date: June 2025

SIGNED: ..... DATE: .....

Table of Contents

Page

1 Chapter 1 Introduction 1

1.1 Background . . . 1

1.2 Problem Statement and Motivation . . . 1

1.3 Research Objectives . . . 2

1.4 Scope and Limitations . . . 2

1.4.1 Scope . . . 2

1.4.2 Limitations . . . 3

1.5 Thesis Structure . . . 4

2 Chapter 2 Literature Review 5

2.1 Introduction . . . 5

2.2 Current Approaches and Their Drawbacks . . . 5

2.3 How the Field Is Advanced by This New Research (2025) . . . 6

2.4 Obstacle Avoidance and Navigation . . . 7

2.5 Object Recognition Deep Learning Applied Here . . . 8

2.6 PID Regression and Motor Control . . . 9

2.7 IoT Remote Control Protocols and Communication . . . 9

2.8 Apache Web Server and Web-Based Robotic Control . . . 10

2.9 Android Devices Using MQTT Protocol . . . 11

2.10 Web Interfaces for Robots . . . 11

3 Chapter 3: Methodology or System Design 13

3.1 Overview . . . 13

3.2 Architecture of Nicobot 2025 Systems . . . 13

3.3 Hardware Tools . . . 14

3.4 Architecture for System Integration and Communication . . . 14

3.5 Operational flowchart . . . 16

3.6 Program software Elements . . . 19

3.6.1 Object Detection and Distance Calculating . . . 23

3.6.2 Navigating and Avoiding Obstacles . . . 24

3.6.3 Autonomous Nicobot 2025 Communication Systems . . . 24

3.6.4 Communication via MQTT . . . 25

3.6.5 LoRa 25
3.6.6 PS2 Remote 2.4Ghz 25
3.7 Methodology for Experimentation 26
3.7.1 Accuracy for Object Detection 26
3.7.2 Navigation Performance 28
3.7.3 Reliability and Communication Latency 29
3.7.4 Safety Response with Manual Override 29
3.7.5 Simulations and Real-World Validation 29
4 Chapter 4: Implementation 31
4.1 Introduction 31
4.2 Electrical Schematic Diagram 31
4.3 Specifications of the Hardware Design 33
4.4 Block Diagram of hardware 34
4.5 Implementation of Software 35
4.5.1 Logic of Control 35
4.5.2 Visualization 36
4.5.3 Interaction and User Interface 36
5 Chapter 5 Experimental Setup and Testing 40
5.1 Description of Test Environment 40
5.2 Preparing for Performance Tests 42
5.2.1 Emergency Operation Testing Operation with PS2 Remote 42
5.2.2 Testing Control Nicobot by web server directly on ESP32 43
5.2.3 Testing Control Nicobot by Webserver on RPI 44
5.2.4 Testing Control Nicobot by Web Server on Local Laptop by Python Flask 46
5.2.5 Testing Control of Nicobot via Web Server on Internet Hosting 50
5.2.6 Testing Control Nicobot by MIT application on an Android phone 51
5.2.7 Testing operation of motor current and voltage sensors 52
5.2.8 Testing speed of Nicobot display on web page by line chart 53
5.2.9 Testing operation of FWD and AFT Ultrasonic sensor 55
5.2.10 Testing Yolo V11 Object detection and distance monitoring on Python application and testing video streaming over local network 55
5.2.11 Testing LoRa Backup communication on Python Application 56

5.2.12 Testing RPlidar Data received on a Python application and displayed on a web page or a Python application . . . 57

6 Chapter 6 Result and Discussion 60

6.1 System performance analysis . . . 60

6.2 Graphs, tables, and comparisons . . . 60

6.3 Limitations and improvements . . . 61

6.3.1 Limitations . . . 61

6.3.2 Future Enhancements: . . . 62

7 Chapter 7 Conclusion and Future Work 63

7.1 Conclusion . . . 63

7.2 Future works . . . 63

8 Appendices 67

List of abbreviations

Abbreviation	Definition
MQTT	Message Queuing Telemetry Transport
YOLO	YOU LOOK ONLY ONE from Ultralytics
SSDs	Single Shot Detectors
CNN	Convolution Neural Network
AI	Artificial Intelligent
IOT	Internet Of Things
GPS	Global Position System
TCP	Transmission Control Protocol
TLS	Transport Layer Security
CoAP	The Constrained Application Protocol (CoAP)
OASIS	Organization for the Advancement of Structured Information Standards
LoRa	Long Range

List of Tables

TABLE **Page**

Table 3.1: Hardware tools for Nicobot 2025 14

Table 3.2: Communication protocol for Nicobot 2025 25

Table 5.1: Measuring speed for left and right motor with PWM input 55

Table 5.2: Ultrasonic sensor performance test 55

Table 5.3: Intel RealSense camera performance test 56

Table 6.1: Compare the latency among several communication systems. 61

List of Figures

FIGURE **Page**

Chapter2:

Figure 2.1: Nicobot was first designed in 2021 6

Figure 2.2: Nicobot upgraded 2025 7

Figure 2.3: RPlidar C1 for Nicobot 2025 8

Figure 2.4: Depth camera Intel RealSense for Nicobot 2025 8

Figure 2.5: YoloV11 and Python App on PC for Nicobot 2025 9

Figure 2.6: Lora SX-1278 433Mhz for Nicobot 2025 10

Figure 2.7: PS2 Remote 2.4 Ghz for Nicobot 2025 10

Figure 2.8: XAMPP Apache web server on PC for Nicobot 2025 10

Figure 2.9: Android Application interface for Nicobot 2025 11

Figure 2.10: Web control interface for Nicobot 2025 12

Chapter 3:

Figure 3.1: Hardware for Nicobot 2025 13

Figure 3.2: Nicobot picture 2025 16

Figure 3.3: Operational flowchart Main Stage 1 for Nicobot 2025 17

Figure 3.4: Operational flowchart Main stage 2 Nicobot 2025 18

Figure 3.5: Operational flowchart Main Stage 3 for Nicobot 2025 19

Figure 3.6: Arduino program flowchart for Nicobot 2025 20

Figure 3.7: Stage 1 to 4 of Arduino program flowchart for Nicobot 2025 21

Figure 3.8: Arduino program loop flowchart for Nicobot 2025 22

Figure 3.9: Intel RealSense D455 depth camera with Yolo for Nicobot 2025 23

Figure 3.10: RPlidar with Python PYQT6 interface control by MQTT protocol for Nicobot 2025 24

Figure 3.11: Training result 1 for Nicobot 2025 27

Figure 3.12: Training result 2 for Nicobot 2025 27

Figure 3.13: Quality of training dataset for Nicobot 2025 28

Chapter 4:

Figure 4.1: Electrical Diagram for Nicobot 2025 33

Figure 4.2: Hardware block 1 diagram Local MQTT broker connection for Nicobot 2025 34

Figure 4.3: Hardware block 2 diagram cloud MQTT Broker connection for Nicobot 2025 35

Figure 4.4: Yolov10 object detection for Nicobot 2025 36

Figure 4.5: Access web server on ESP32 for Nicobot 2025 37

Figure 4.6: Access web server on RPI for Nicobot 2025 38
Figure 4.7: Access web server on Laptop Python Flask for Nicobot 2025 38
Figure 4.8: Access web server on WebHost Internet for Nicobot 2025 39
Figure 4.9: Android App for Nicobot 2025 39
Chapter 5:
Figure 5.1: Experimental setup up for Nicobot 2025 41
Figure 5.2: Experimental setup up for Nicobot 2025 41
Figure 5.3: S2 Remote control for Nicobot 2025 43
Figure 5.4: Access Web Server on ESP32 Nicobot 2025 44
Figure 5.5: WebServer on RPI for Nicobot 2025 46
Figure 5.6: Object detection on Python flask for Nicobot 2025 48
Figure 5.7: RPlidar indication on web for Nicobot 2025 49
Figure 5.8: Control robot over the internet website for the Nicobot 2025 50
Figure 5.9: Control by the Android Application for the Nicobot 2025 52
Figure 5.10: Motor left and right speed changing 1 54
Figure 5.11: Motor left and right speed changing 2 55
Figure 5.12: YoloV10 object detection for the Nicobot 2025 26
Figure 5.13: LoRa Data Receive for the Nicobot 2025 57
Figure 5.14: RPlidar display for the Nicobot 2025 59
Appendix
Figure 8.1: RPlidar display with certain data drawn by OpenCV Library on Python 68
Figure 8.2: Data structure that ESP32 sends to the web page, MQTT, and LoRa. 69
Figure 8.3: Training traffic signs for the Nicobot by using Google Colab. 70
Figure 8.4: Application using PYQT6 user interface designer 71

1 Chapter 1 Introduction

1.1 Background

The advancement of the Internet of Things (IoT), Message Queuing Telemetry Transport (MQTT), and image processing has revolutionized modern robotics, enabling the development of increasingly autonomous and intelligent systems. These technologies allow robots to perform complex tasks with minimal human intervention, expanding their potential applications in fields such as logistics, transportation, agriculture, and smart cities [1]. The robot car system described in this thesis has its origins in an earlier prototype developed in 2020, which was built using a single Arduino Mega 2560 microcontroller and a 2.4 GHz RF remote controller. This early design was limited to basic manual control, relying on PWM signals from the remote to steer and drive the car [2]. While functional, the system lacked advanced capabilities such as environmental awareness, autonomous decision-making, or real-time monitoring. Sensor data from basic modules like encoders and ultrasonic sensors were only used locally, without integration into higher-level control or safety mechanisms [3]. The new generation robot car presented in this research marks a significant leap forward. Central to the new design is the integration of IoT communication technologies, especially the MQTT protocol, a lightweight and highly efficient publish-subscribe messaging system that enables real-time data exchange between the robot and remote servers or clients. This architecture allows the robot to be controlled via a web-based interface, an Android application, and even to support real-time video streaming and image processing using state-of-the-art object detection models like YOLOv10 and YOLOv11 [4] [5]. Through computer vision, the robot can identify traffic signs, detect pedestrians, and estimate distances, improving its capacity for autonomous operation. For added safety, obstacle detection sensors such as ultrasonic sensors and RPLidar modules have been installed to allow the robot to detect nearby objects and stop within a defined safety range, reducing the risk of collisions [6]. This adds an important layer of environmental awareness and responsiveness to the system. To ensure robust performance, backup communication systems have also been implemented. In the event of an internet or Wi-Fi failure, the robot car can switch to a LoRa communication module with a range of up to 2.5 km, or a PS2 wireless controller operating at 2.4 GHz, ensuring continuous control in diverse environments [7] [8]. Using IoT protocols like MQTT offers several advantages, including low-latency communication, minimal bandwidth consumption, and the ability to scale across multiple devices. However, it also introduces challenges, such as vulnerability to network interruptions, potential cybersecurity risks, and the need for robust fallback mechanisms. Addressing these challenges is crucial to ensuring the safe and reliable operation of autonomous systems. Overall, this thesis aims to design and implement a scalable, efficient, and intelligent smart robot car system that integrates IoT, MQTT, obstacle detection sensors, and image processing technologies. By combining advanced communication protocols, real-time sensing, and robust backup strategies, this work contributes to the growing field of IoT-based robotics and lays the foundation for future research in autonomous vehicle systems [1].

1.2 Problem Statement and Motivation

Because they could improve efficiency and lower human involvement, autonomous robotic vehicles have attracted much interest in many fields, including smart transportation, industrial automation, and surveillance [9]. Still, getting consistent autonomous navigation is difficult, especially given problems with sensor fusion, communication delays, and real-time decision-making. Recent developments in the Internet of Things (IoT) have resulted in fresh ideas addressing some of these issues that enable enhanced mobility and remote management of robotic systems, so enabling improved mobility and remote management [1].

Low-power communication protocols are absolutely essential for autonomous systems to run efficiently and effectively, particularly in resource-limited surroundings. Because of its lightweight character and best performance over limited networks, the Message Queuing Telemetry Transport (MQTT) protocol is especially well-suited for real-time communication in IoT-enabled robotic systems [1] [10]. Furthermore, including computer vision technologies helps the robot to detect obstacles, follow pre-defined paths, and make autonomous navigation decisions, thus increasing its general autonomy and adaptability in dynamic surroundings.

This work aims to design and build an autonomous robotic car with IoT and image processing technologies that are easily controllable and monitored, thus improving the decision-making capacity of the system. This system could find use in smart cars, security patrol vehicles, autonomous delivery robots, and other intelligent transportation systems [9].

1.3 Research Objectives

1. Design and implementation of an autonomous robotic car combining Internet of Things (IoT) technologies and image processing is the main goal of this work to improve remote control and navigation capacities. The study intends specifically to examine various operation solutions by using MQTT technology and then develop autonomous navigation based on image processing and RPLidar.

2. Create an autonomous robotic car with IoT technologies and MQTT protocol communication in real time using a lightweight design and effective message delivery across limited networks.

3. Accurate and adaptive robot motion control depends on movement precision, stability, and responsiveness; thus, to improve these aspects, design and implement a motor control system based on pulse-width modulation (PWM).

4. Combine image processing methods to enable the robot to detect obstacles effectively and follow paths, so guiding its autonomous navigation in dynamic surroundings.

5. Using the Flask framework, create a web-based interface that gives users a flexible and user-friendly platform for interaction, enabling remote control, monitoring, and management of the robotic car.

6. Using LoRa and PS2 remote control systems, include backup communication protocols to guarantee dependable operation in settings with limited or unstable IoT connectivity, to improve system resilience, and extend communication range.

7. Analyze the navigational accuracy, communication dependability, and real-time decision-making efficiency of the system to confirm its efficacy in many application situations.

1.4 Scope and Limitations

1.4.1 Scope

This work focuses on the design, development, and implementation of an advanced autonomous robotic car, including state-of-the-art object detection and environmental sensing technologies to improve real-time navigation, obstacle avoidance, and decision-making capability. Combining an Intel RealSense D455 depth camera to track the distance between discovered objects and the robotic car, the system uses the YOLOv10 or YOLOv11 deep learning algorithm for accurate and fast object detection and classification.

Apart from ultrasonic sensors installed at the front and rear of the vehicle to provide an extra layer of safety by identifying nearby objects and so preventing accidents during forward and reverse movement, the robotic car also has an RPLIDAR sensor to perform 360-degree obstacle scanning, enabling accurate measurement of the distance from objects surrounding the robot car.

The parameters of this investigation consist of

YOLOv10/v11 is used to detect specific objects, which can be selected on the Python Web interface. The dataset in this project is used to pretrain yolov10n.pt or yolov11n.pt [11]

Depth sensing and distance measuring using the Intel RealSense D455 integrated into a Python program on the laptop. By using the Flask web framework, the distance of the object will be displayed and marked on the camera display of the web page. The value of the distance will be calculated by an algorithm inside Python, and a signal will be sent to ESP32 by MQTT to stop the robot car if the distance is in the safety range.

RPLIDAR applied for 360° obstacle scanning surrounding to measure the object distance surrounding the robot car by a Python program on a laptop. Similarly, Python will send the signal to the ESP32 controller to stop the robot car in the safety setting value. The value of the safety range is set on the web interface on a laptop.

The application of ultrasonic sensors in this project is used for safety. Whenever the distance from the object is detected by the front and aft sensor, data will be read by the ESP32 controller and stop the robot car in case the distance is in the range set by the user.

Design of the ESP32 control system of the robot used as a central processing unit of the robot car: This controller plays a main role in operating independently. A signal from a PS2 remote, a web interface, or a smartphone can be considered the remote's user interface.

creation of a web-based remote control and monitoring system interface built on the Flask framework or HTML, JavaScript, and PHP.

Performance of the robot with respect to object detection accuracy, depth estimate precision, obstacle avoidance, navigation dependability, and system safety.

1.4.2 Limitations

The system has its merits, but it also suffers from the following shortcomings.

- Raspberry Pi limited processing capabilities: The deep learning models will not be able to work in real time due to resource constraints on the Raspberry Pi or even a local laptop.
- Environmental considerations: Obstruction of the cameras, as well as the amount of light present, will impact the efficacy of image processing.
- Network reliant: The stability and speed of the network determine the efficacy of MQTT communication.
- Limited battery life: Data computation and message communication may result in high energy expenditure, leading to deterioration of the battery life.
- PID control algorithm is not successfully implemented for this robot car, so the motor speed will not be automatically adjusted while moving under different road conditions.
- Position control is hard to achieve due to the main reason of the robot car mechanism.

1.5 Thesis Structure

The structure of this thesis is as follows:

Chapter 1 Introduction: It illustrates the background information, the status of the research problem, and some research objectives. The scope and delimitations are also discussed in this section, and the thesis outline is described.

Chapter 2 Literature Review: This section describes the work done in IoT-based autonomous vehicles, the MQTT communication protocol, the WebSocket protocol, and image processing in robotics.

Chapter 3: System Design and Architecture illustrates the software and hardware components of a smart robot car.

Chapter 4 Implementation: Chapter 4 details the step-by-step implementation of the IoT, MQTT, WebSocket, and image processing modules.

Chapter 5 Experimental Setup and Testing: This chapter describes the methodology for testing the system and its components after implementation.

Chapter 6 Results and Discussion: This chapter will show how well the system works in terms of efficiency, the accuracy of the robot car's navigation, and the integration of the various components.

Chapter 7 Conclusion and Future Work: This section will mention the research's main conclusions, achievements, and actions taken to improve this project in future work.

2 Chapter 2 Literature Review

2.1 Introduction

Driven by uses in intelligent transportation, industrial automation, innovative delivery systems, environmental monitoring, and security surveillance, the development of autonomous robotic systems has become a very active research area. Mobile robots now have much more possibilities thanks to artificial intelligence (AI), Internet of Things (IoT) technologies, and sophisticated sensor systems. Although many methods have been suggested to improve the autonomy and efficiency of robot systems, significant difficulties remain in achieving strong navigation, precise control, efficient communication, and real-time perception under demanding environmental conditions. Reviewing the pertinent literature on autonomous navigation, object detection, sensor fusion, control strategies, communication protocols, and remote control interfaces, this chapter points out current research gaps. It clarifies how the suggested work seeks to fill in the gaps.

2.2 Current Approaches and Their Drawbacks

Nicobot 2021, which uses Arduino Mega 2560, is the heart of the controller, like many conventional Nicobot systems, including previous iterations of the project. Usually acting as the central processing unit, this microcontroller is in charge of both deciphering user input and carrying out motor control algorithms. In particular, the Arduino Mega modifies the speed and direction of the robot's two DC motors in response to pulse-width modulation (PWM) signals from a 2.4 GHz RF remote controller. In addition, it connects to current sensors and motor encoders to record telemetry data in real time, including power consumption and wheel rotation speed.

The practical use of encoders and modern sensors in many systems is still restricted, even though their inclusion indicates an awareness of sophisticated control and monitoring requirements. Instead of being fed into algorithms that make real-time decisions, like closed-loop control, adaptive path correction, or safety interlocks, sensor data is frequently recorded for post-operation review or basic feedback display. This is a significant disadvantage, especially when working in dynamic environments that call for reactive behavior, like cutting off power in the event of an overcurrent or abruptly avoiding obstacles.

Furthermore, there are extra autonomy and scalability restrictions brought about by the dependence on RF remote control systems. Due to their intrinsic point-to-point nature and lack of network-layer intelligence, RF-based systems are not appropriate for cloud-integrated AI models, cooperative swarm behaviors, or remote internet-based monitoring. Furthermore, these systems are not capable of sensor fusion, and they are unable to process sophisticated algorithms like computer vision, artificial intelligence (AI), or machine learning in real time due to the lack of more sophisticated controllers, such as the Raspberry Pi or ESP32.

Several studies have emphasized the limitations of employing microcontroller-only techniques for autonomous mobile robots. For example, it highlighted that embedded systems with no dedicated processors for communication or vision have limited scalability when performing autonomous tasks. Similar studies have shown that real-time data fusion and intelligent controllers greatly improve decision-making, especially in unstructured settings.

In conclusion, the Arduino-based control system lacks the complexity necessary for completely autonomous operation, even though it offers a workable basis for manual and semi-autonomous robot control. System design is severely hampered by the limitations of RF communication and the inability to process real-time sensor data for closed-loop decision-making [12]. These drawbacks motivate the shift to more integrated platforms that

use sophisticated controllers, contemporary communication protocols (like MQTT and LoRa), and sensor-rich architectures that can facilitate AI-driven autonomy and improved safety.

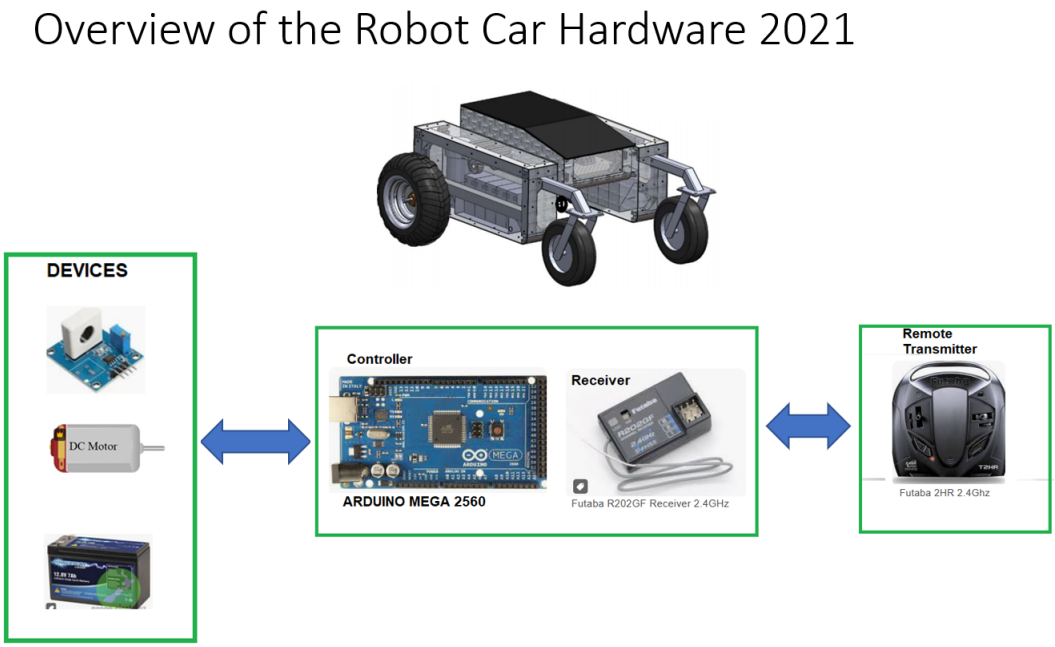


Figure 2.1: Nicobot was first designed in 2021

2.3 How the Field Is Advanced by This New Research (2025)

This research offers a substantial improvement over earlier Nicobot platforms by combining cutting-edge sensory fusion, multi-modal communication protocols, and contemporary embedded systems for autonomous navigation. Conventional robotic systems, like those built on the Arduino Mega 2560, usually rely mainly on manual control and have limited use of data from onboard sensors. They also typically lack real-time adaptability. On the other hand, this project uses a layered hardware-software co-design approach, utilizing the capabilities of a local laptop, Raspberry Pi 3 B+ , and ESP32 microcontroller [13] [14].

Using PID algorithms, the ESP32 serves as the primary control unit for the Nicobot. It also gathers sensory data, such as voltage and current readings, to guarantee electrical safety. At the same time, the laptop runs a Flask web dashboard. It supports computationally demanding tasks like real-time video processing using YOLOv10 or YOLOv11 object detection models [11] [4], while the Raspberry Pi acts as an MQTT broker and hosts an Apache-based web server for local communication and interface deployment [15].

The system uses three parallel channels to provide reliable communication: WebSocket for full-duplex web communication, LoRa for long-range, low-power data exchange, and MQTT over Wi-Fi for high-speed messaging. A PS2 wireless remote also provides a backup manual override method to improve system security in the event of unplanned malfunctions [7].

Sensor integration is essential to increase the robot's autonomy and environmental awareness. For object detection and path planning, 3D spatial data is provided by a Logitech webcam and a RealSense D455 depth camera [9] [16]. Ultrasonic sensors installed on the front and rear provide short-range collision avoidance [17], and an RPLIDAR is used for 360° obstacle scanning [18]. The robot can operate dependably in a variety of situations, including both indoor and outdoor settings with moving obstacles, thanks to its broad sensor suite.

The user interface is HTML and JavaScript-based, developed on Apache and Flask web servers [19]. Run on a Raspberry Pi and a local laptop, these servers offer a responsive, lightweight dashboard for real-time robot

control and monitoring. This design guarantees flexibility and scalability for many use situations by allowing users to efficiently run the robot via a local network or internet-based access.

Apart from the online interface, a mobile application created with MIT App Inventor has been launched [20]. This Android program improves field operations or remote scenarios by letting users remotely monitor and operate the robot using portable devices. Additionally, a PS2 wireless controller is included as a backup manual control system, guaranteeing continuous operation during emergency or communication outages. This layered approach to interface design improves user accessibility and system dependability.

This study advances the field by showing how contemporary microcontrollers, sensor fusion, and hybrid communication systems can be integrated to create a robust, modular, and intelligent robotic vehicle. This robot is appropriate for applications like delivery automation, surveillance, and innovative mobility because it combines real-time video analysis, depth perception, safety monitoring, and flexible remote control methods, which represent a significant improvement over previous systems [21].

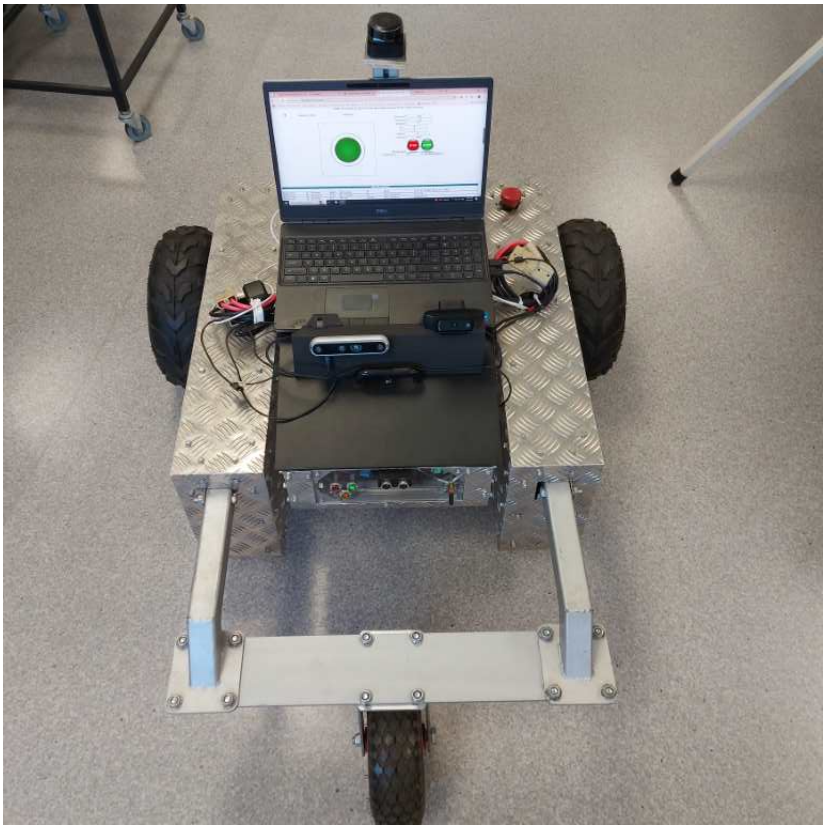


Figure 2.2: Nicobot upgraded 2025

2.4 Obstacle Avoidance and Navigation

Preventing obstruction during navigation calls for a robot to localize its position, sense its surroundings, and find object obstructions in the robot's way. Early techniques were limited in environmental awareness and could not handle dynamic obstacles; they mainly depended on simple reactive control using infrared or ultrasonic sensors [22]. Simultaneous localization and mapping (SLAM) technologies have advanced, and 2D LiDAR sensors such as RPLIDAR C1 have been extensively used for obstacle detection, giving robots a global view of the surroundings [23].

Figure 2.3 : RPlidar C1 for Nicobot 2025

We're progressively combining several modalities, including LiDAR, RGB-D cameras such as the Intel® RealSenseTM D455, and ultrasonic sensors, for redundant and strong environmental perception, overcoming the constraints of single-sensor systems. This multimodal approach benefits navigation performance in both indoor and outdoor environments.

Figure 2.4: Depth camera Intel RealSense for Nicobot 2025

2.5 Object Recognition Deep Learning Applied Here

Robots can understand and interact with their surroundings thanks to object detection YOLOv10 and YOLOv11 from Ultralytic. Traditional image processing methods based on edge detection or background subtraction [5] were effective only in controlled environments. Real-time object detection was transformed by the development of convolutional neural networks (CNNs) and architectures, including YOLO (You Only Look Once)

More recently, using transformer-based backbones, YOLOv7, YOLOv8, and YOLOv10/YOLOv11 [11] have enhanced detection speed and accuracy. Although deep learning models are computationally demand-

ing, researchers have used optimization techniques like quantization, pruning, and hardware acceleration (e.g., TensorRT, Coral TPU) to deploy them on resource-limited platforms like the Raspberry Pi [13].

This work uses YOLOv10/YOLOv11 integrated with the RealSense D455 for simultaneous object detection and distance estimation.

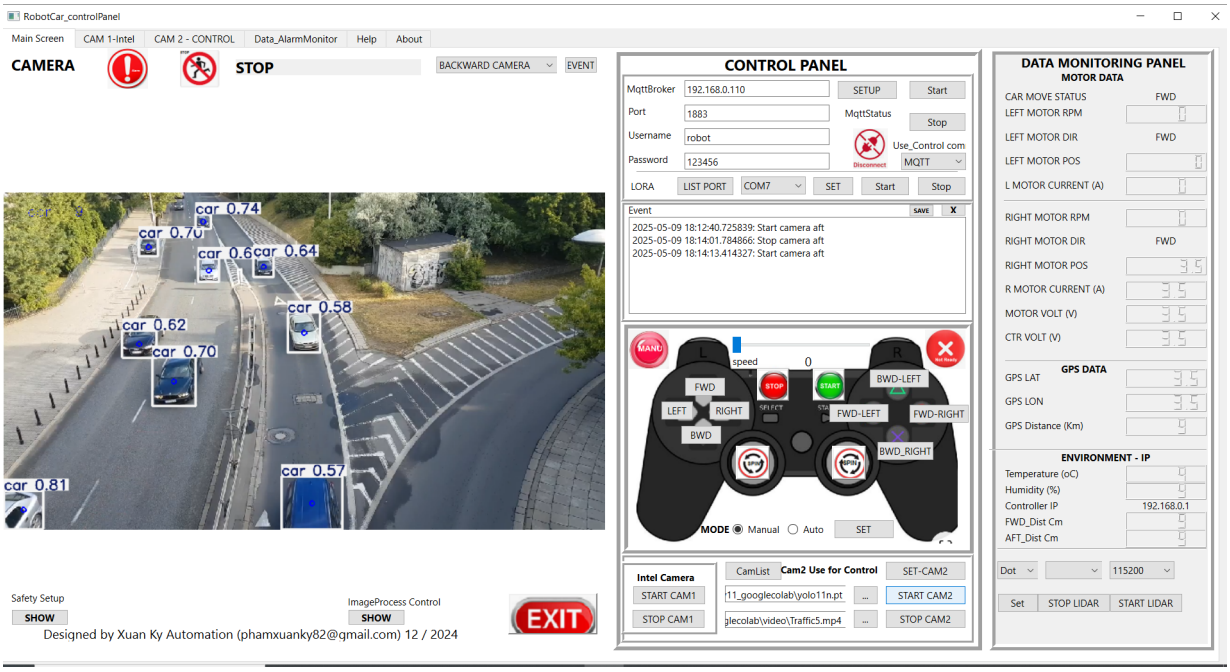


Figure 2.5: Yolov11 and Python App on PC for Nicobot 2025

2.6 PID Regression and Motor Control

Self-contained robots depend on exact motion control. PID (Proportional-Integral-Derivative) controllers are extensively used to minimize errors and achieve stable trajectory tracking [24], while DC motors under control using PWM (Pulse Width Modulation) provide fine-grained speed control [25]. Adaptive and nonlinear PID control has lately been investigated to increase resilience. This project aims to use a PID controller to improve line-following and obstacle avoidance by regulating left and right motor speeds [26]. However, due to the limitation of time and hardware issues, the position control using PID or speed regulation control using PID is not approached. Therefore, this feature will be improved in the future improvement of this Nicobot project.

2.7 IoT Remote Control Protocols and Communication

The Internet of Things (IoT) has revolutionized robotic communication, letting robots share data with operators and distant servers. Because of its lightweight, publish-subscribe architecture and effective bandwidth use, MQTT—Message Queuing Telemetry Transport—is especially popular for robotics [27]. Previous work has used MQTT in industrial automation, smart homes, and swarm robotics [21].

WiFi-based MQTT systems could, however, suffer from dependability problems due to signal loss or network congestion. LoRa (Long Range) communication as a backup channel to help with this [28].

Figure 2.6: Lora SX-1278 433Mhz for Nicobot 2025

Furthermore, PS2 (PlayStation 2) wireless modules have been adapted to offer mobile robots basic, low-latency control links [7].

Figure 2.7: PS2 Remote 2.4 Ghz for Nicobot 2025

2.8 Apache Web Server and Web-Based Robotic Control

Apart from hardware-level communication, web-based interfaces are becoming rather common for robot control. Installable on a Raspberry Pi, a local PC, or a cloud-hosted website, the Apache web server is a widely used open-source HTTP server that allows one to remotely monitor and control a robot via a regular web browser. Raspberry Pi + Apache configurations have been effectively used in several projects to host control dashboards, live video streams, and status monitors, enabling users to send motion commands or view sensor data in real time [15]. Apache Web Server can also run on the local laptop to serve as a web server for Nicobot by using XAMPP software below.

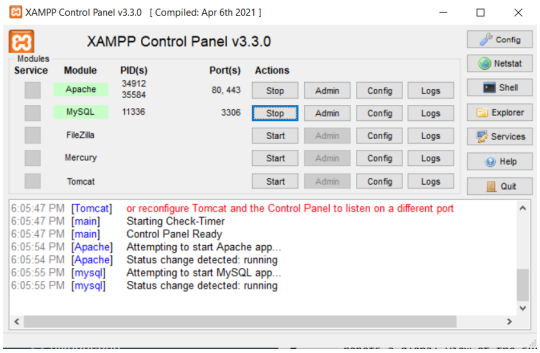


Figure 2.8: XAMPP Apache web server on PC for Nicobot 2025

By hosting the control interface on a local PC or a public website, users may control the robot over LAN or the internet, thus providing flexibility. Apache is suitable for more complicated installations since it provides more scalability and compatibility with common web technologies, including PHP, HTML, and JavaScript, than lightweight solutions, such as Flask [29].

2.9 Android Devices Using MQTT Protocol

Mobile apps are still another approach for remote robot control. Using the MQTT protocol, the MIT App Inventor platform lets Android apps rapidly grow and interact with robots. This method allows users to monitor robot status via the internet connection, send movement commands, control robots using cellphones, and receive sensor updates [30].

Previous work has shown how well MIT App Inventor creates user-friendly interfaces for robots, drones, and smart home appliances [20]. MQTT and mobile apps together increase system flexibility so that non-technical users may operate and monitor robots conveniently.

Figure 2.9: Android Application interface for Nicobot 2025

2.10 Web Interfaces for Robots

Easy human-robot interaction depends on web interfaces. While Apache offers a stronger and more scalable answer for bigger installations, the Python Flask framework has become well-known for building lightweight, real-time robot dashboards [29]. Previous studies have indicated that web-based interfaces can significantly increase usability by combining live video, sensor plots, telemetry, and manual controls.

This project calls for a hybrid architecture combining:

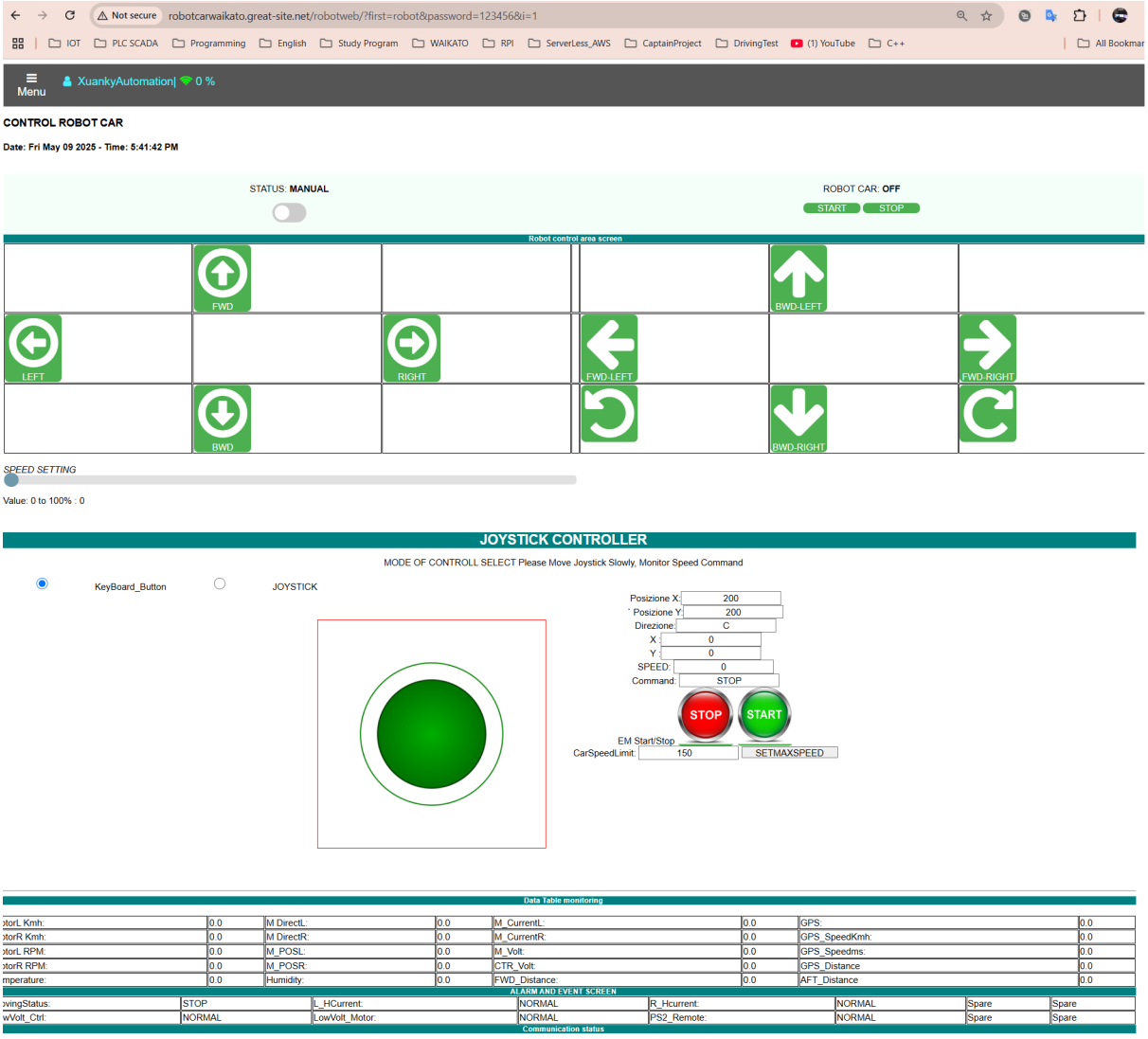


Figure 2.10: Web control interface for Nicobot 2025

1. Apache local control web server on Raspberry Pi, Local PC, and Web hosting on the internet
2. Local laptop web server using Python Flask web framework, stream video from 2 cameras over WebSocket, object recognition using Yolo V11, and display RPLidar data scanning the surrounding robot to prevent the robot from hitting objects.
3. Web Server on ESP32 using WebSocket server-client protocol to control the robot
4. MIT Android Application with MQTT communication protocol is used to interact with the robot over mobile phones

This multi-layer approach guarantees that the robot can be flexibly controlled from several devices and networks.

3 Chapter 3: Methodology or System Design

3.1 Overview

This chapter offers a comprehensive system design, architecture, hardware and software components, models and algorithms, and an experimental approach to building and assessing the autonomous robotic car. The system combines several technologies, including deep learning for object detection, depth sensing for distance estimation, LiDAR for scanning the 360-degree area around the robot, and ultrasonic sensors for obstacle avoidance. IoT protocols (MQTT) also establish communication; backup channels use LoRa and PS2 remote control.

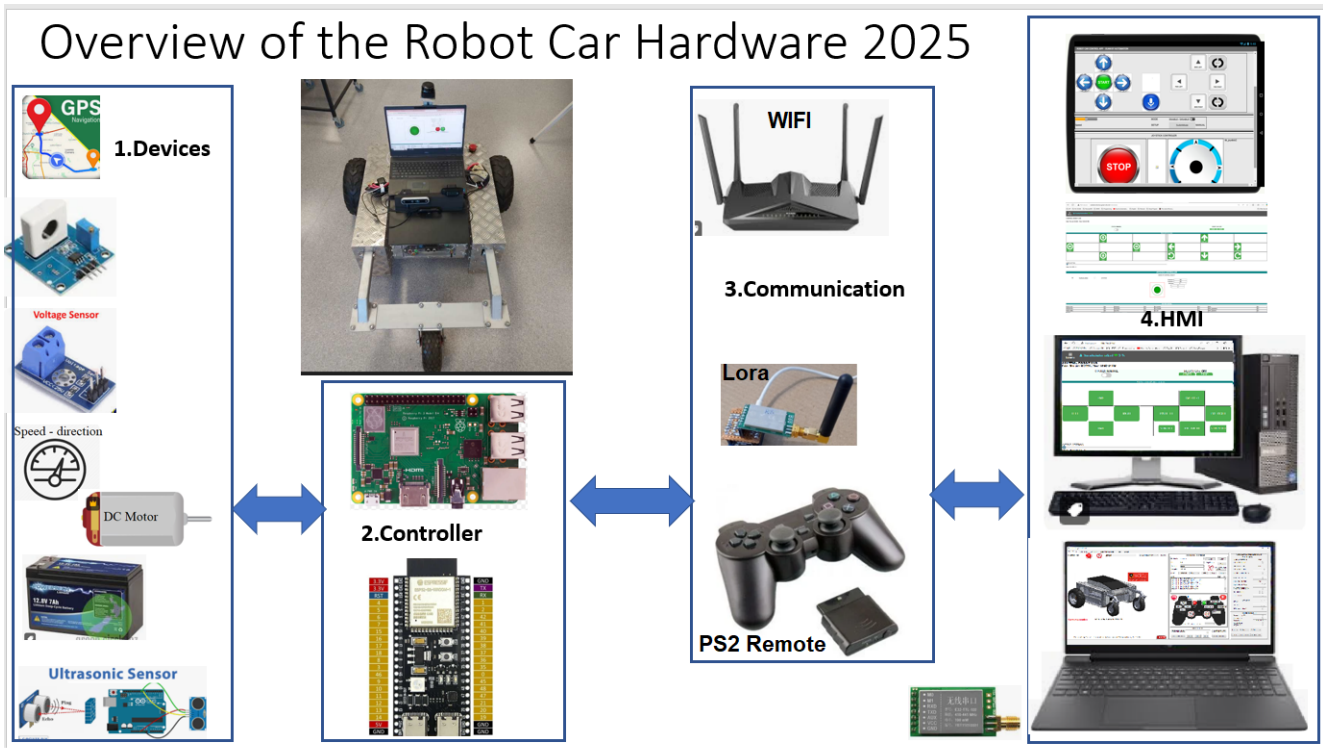


Figure 3.1: Hardware for Nicobot 2025

3.2 Architecture of Nicobot 2025 Systems

The architecture of the Nicobot 2025 integrates several layers:

1. Sensing and motion layer:
   (a) Intel RealSense D455 Distance Measurement and Depth Camera for Object Recognition.
   (b) RPLidar C1 360° obstacle scanning.
   (c) For close-range safety, ultrasonic sensors at the front and rear
   (d) All voltage, current sensors, Encoders, and other devices
2. Main Controller layer
   (a) Main controller is ESP32

(b) RPI 3B+ function for MQTT broker, Apache Web Server.

(c) Laptop with DC power supply input function as Python Flask web server, image processing from two cameras using YOLOv10 or YOLOv11. Moreover, this laptop can be used as a spare MQTT broker (Mosquitto) and a spare Apache web server for the Nicobot 2025.

3. Communication layer: supports several channels of transmission.

   (a) WIFI Communication
   - MQTT across Wi-Fi for IoT real-time messaging.
   - Web Socket server over Wifi

   (b) LoRa modules for backup communication over distances.

   (c) PS2 wireless controller for hand manual emergency control.

4. Application Layer: Offers distance between interfaces.

   (a) Apache web server on RPI using two different protocols: MQTT and WebSocket.

   (b) Python Flask web server on local laptop

   (c) Apache web server on Internet web hosting

   (d) MQTT mobile control MIT App Inventor Android app.

3.3 Hardware Tools

component	description
processing unit	Raspberry Pi 3B (quads-core ARM CPU, 1GB RAM)
camera	Intel RealSense D455 RGB-D camera
LiDAR	RPLIDAR C1 360 degrees scanner
Distance Sensor	SensorsHC-SR04 modules for front and rear
Motor Driver	BTS7960 high-power H-bridge driver for motors
Motor	12V DC geared motors
LoRa	LoRa Tranducer SX-1278
Remote	PS2 remote transmiter and receiver
Controller	ESP32 Main Controller
Sensors	Current, Voltage

Table 3.1: Hardware tools for Nicobot 2025

3.4 Architecture for System Integration and Communication

The autonomous Nicobot 2025 system is constructed using a multi-layered architecture that combines edge devices, communication protocols, and centralized control to facilitate remote operation and real-time monitoring. Each major hardware component is given a specific role in this architecture to optimize performance, dependability, and modularity.

The ESP32 microcontroller is the central control unit at the system's center, handling real-time communication, sensor data collection, and motor control. It facilitates bidirectional communication with the central server by functioning as a WebSocket and an MQTT client. Additionally, the ESP32 processes data from the RPLIDAR, GPS modules, ultrasonic sensors (front and rear), current sensors, and voltage sensors. The robot's motion control, safety features, and decision-making depend on its effective real-time operation [31].

The Raspberry Pi (RPI) functions as a local edge server. It is set up as an MQTT broker to enable message-based communication between the laptop, Android app, and ESP32, among other nodes. The RPI also houses an Apache Web Server, which supplies the remote user interface with both static and dynamic content, such as control inputs, video feeds, and system status [32] [13].

The local laptop serves as a central hub for high-level processing and control. It is powered by a Python Flask web application that functions as a responsive browser-based interface for managing and keeping an eye on the Nicobot 2025. Using the YOLOv10 or YOLOv11 object detection models, the laptop also carries out complex computer vision tasks [11]. The laptop has two cameras for object detection and live video streaming: an Intel RealSense D455 depth camera and a Logitech HD camera. Accurate and safer navigation is made possible by the RealSense camera's ability to measure the precise distance to detected objects [33] [16].

The laptop has a built-in LoRa receiver that runs a custom Python-based LoRa communication handler in addition to primary communication through MQTT and WebSockets [28]. This module offers low-power, long-range backup communication if Wi-Fi connectivity is lost. It guarantees that the ESP32 can still receive critical commands for rerouting or emergency stops.

Additionally, the system includes a mobile application for Android that was created with MIT App Inventor and connects to the MQTT broker as a client. With the help of this app, users can operate the Nicobot 2025 online from their smartphones. The smartphone interface improves system flexibility and accessibility in field settings [30].

The system achieves a scalable and fault-tolerant design appropriate for autonomous navigation applications by dividing tasks among specialized devices: a laptop for high-level computing and object detection, a Raspberry Pi for local brokering and web serving, and an ESP32 for real-time control.

Figure 3.2: Nicobot picture 2025

3.5 Operational flowchart

The software development stage of the autonomous Nicobot 2025 system depends much on the operational design flowchart. Acting as a disciplined visual guide, it charts the series of actions, choices, and data flow needed for the robot to run as intended. Through a detailed step-by-step process, the flowchart guarantees logical integration of all important modules—including sensor reading, decision-making, motor control, communication, and safety mechanisms—before actual coding starts. This methodical approach reduces programming mistakes, increases debugging effectiveness, and speeds software development.

Objective of the Operational Flowchart

Creating the operational flowchart mostly helps convert the high-level system architecture into a thorough, executable plan. It lets developers see how the robot interacts with its hardware components—e.g., sensors, motors, and communication modules—and software algorithms—e.g., PID control, obstacle detection, and path following. It also clarifies how the system manages inputs and outputs, enabling the software to satisfy safety criteria and real-time performance needs.

Three main stages make up the operational flowchart for this project, each signifying a crucial phase of the operation of the Nicobot 2025. Powering up the system, running hardware checks, initializing sensors (including the depth camera, RPLidar, and ultrasonic sensors), and creating communication links fall to stage 1:

System Startup and Initialization. This phase guarantees that all system components—including the Raspberry Pi, motor drivers, and communication modules—are correctly set before the robot starts its operations.

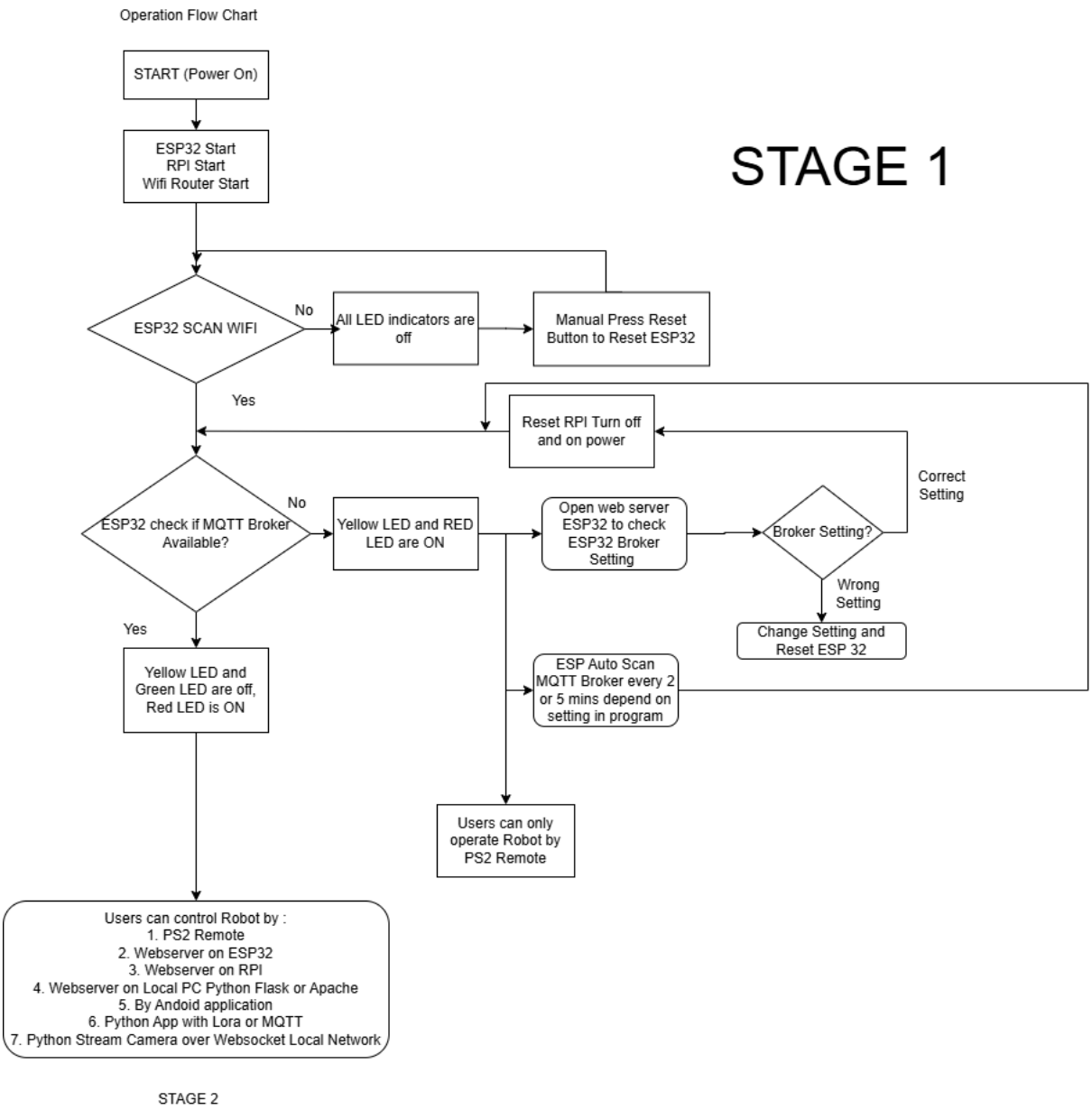


Figure 3.3: Operational flowchart Main Stage 1 for Nicobot 2025

Second stage: PS2 Remote Control. Operation forms an emergency control mechanism. In cases when autonomous or network-based control is either absent or unreliable, this stage lets the operator manually run the Nicobot 2025 with a PS2 remote controller. This fallback mechanism ensures safety during development, testing, or in case of system breakdowns.

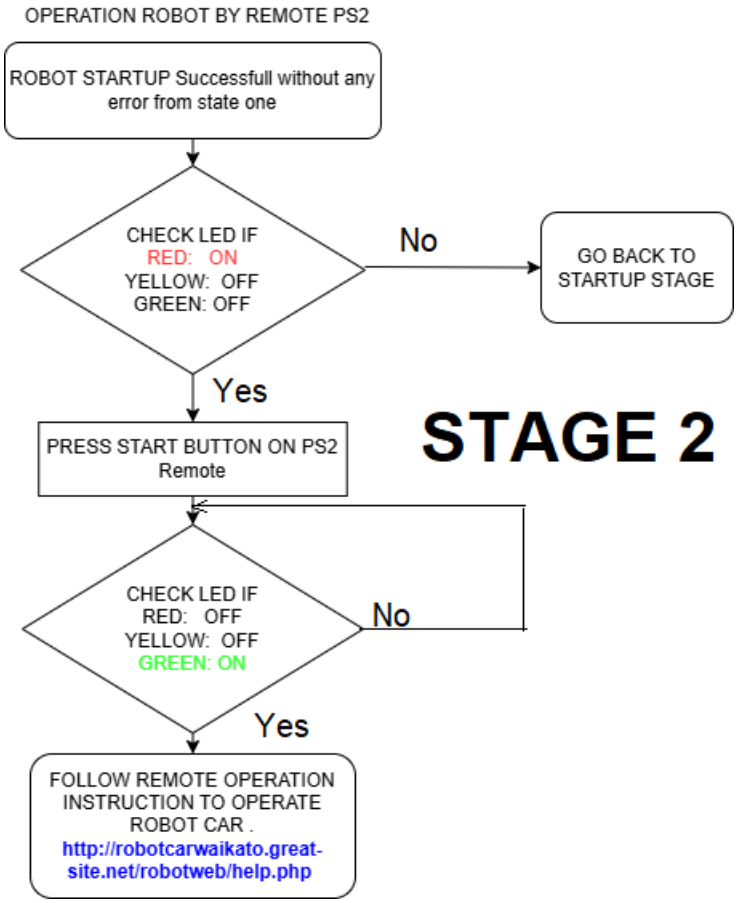


Figure 3.4: Operational flowchart Main stage 2 Nicobot 2025

Third Stage: Main Control. The method integrates several communication protocols and control interfaces and uses the autonomous control system completely. This stage allows real-time communication between the robot and the remote operator using the MQTT and WebSocket protocols across the web server. Furthermore, LoRa communication is used as a long-range backup system to keep connectivity in places with limited Wi-Fi or internet access. A Python-based application on a local PC or an Android application created using MIT App Inventor can manage control commands and telemetry data, giving users flexible choices for monitoring and controlling the robot remotely.

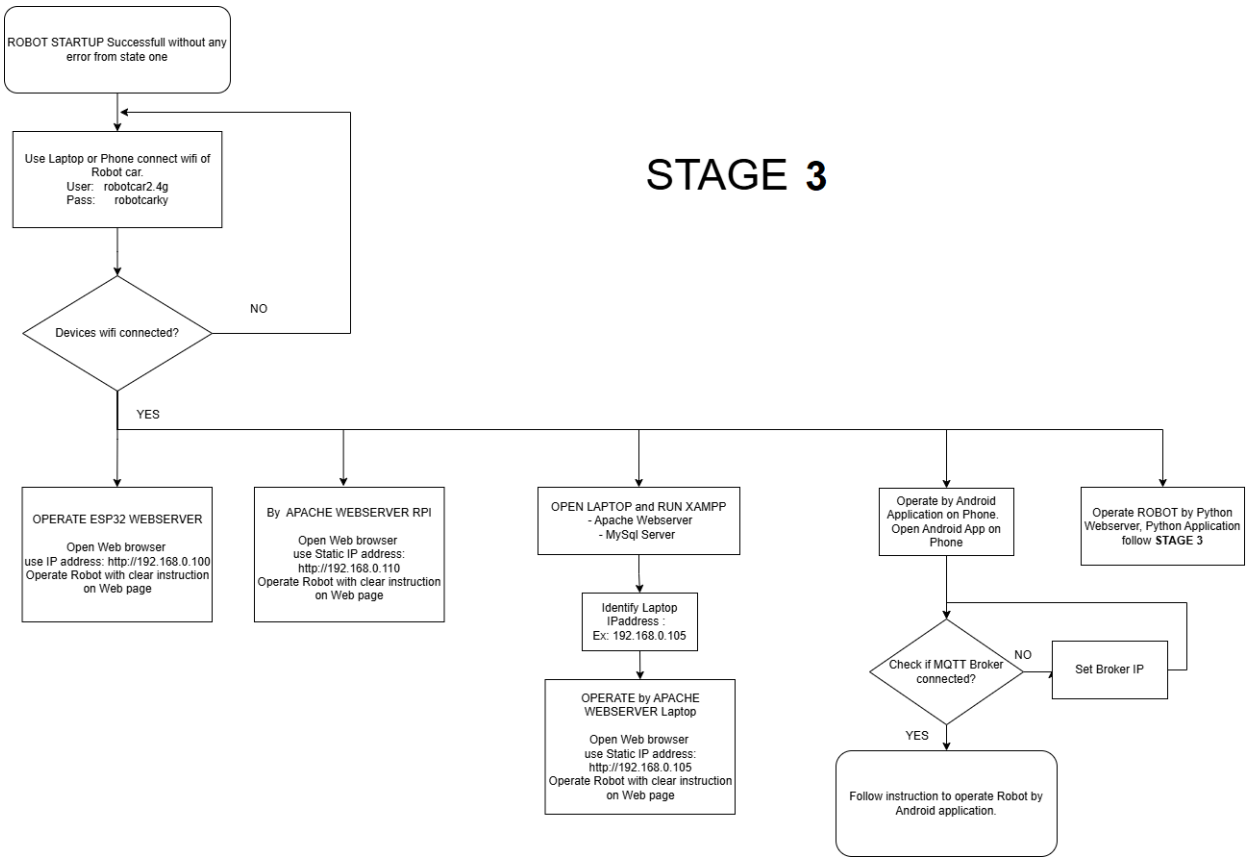


Figure 3.5: Operational flowchart Main Stage 3 for Nicobot 2025

3.6 Program software Elements

1. Arduino IDE for programming ESP32

   (a) Programming Flow Chart

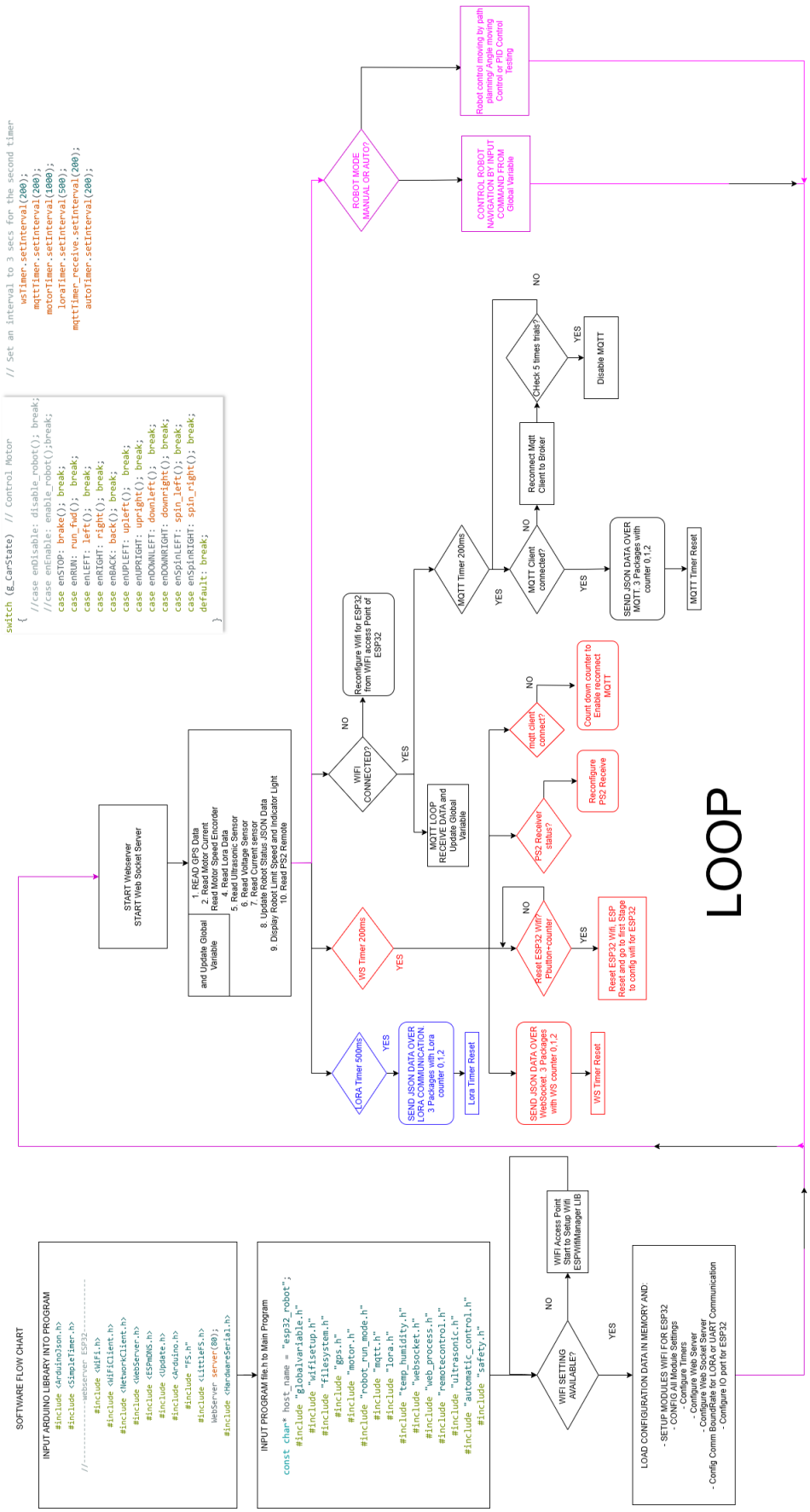


Figure 3.6: Arduino program flowchart for Nicobot 2025

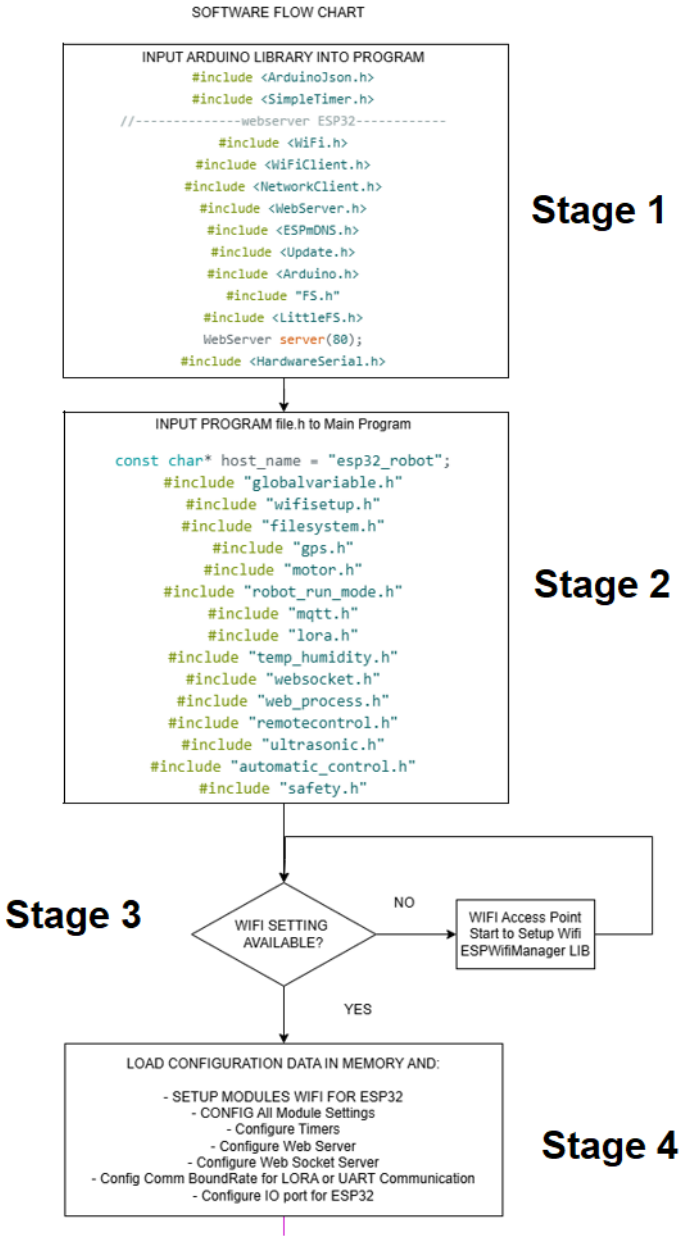


Figure 3.7: Stage 1 to 4 of Arduino program flowchart for Nicobot 2025

Explanation of Arduino software flowchart

First, all Arduino libraries will be imported into the main program to prepare for the operation of some devices, such as the ArduinoJSON library, ESP32 wifi web server, file service library LittleFS, etc.

In the next stage, all program files with a filename will carry out each function of the ROBOT devices, which will be imported into the main Arduino program. These files are written based on the robot's hardware requirements.

Wi-Fi availability will be checked in the third stage. If not available, the ESP Wi-Fi manager will start the Wi-Fi access point so that users can access the Wi-Fi mode to set up Wi-Fi at http://192.168.4.1. The Wi-Fi default Name is AutoConnectAP, and the Password is 12345678.

ESP starts up completely after finishing stage 4. In this stage, ESP32 will carry out this operation:

- LOAD CONFIGURATION DATA IN MEMORY
- SETUP MODULES WIFI FOR ESP32
- CONFIG All Module Settings
- Configure Timers
- Configure Web Server
- Configure Web Socket Server
- Config Comm BoundRate for LORA or UART Communication
- Configure IO port for ESP32

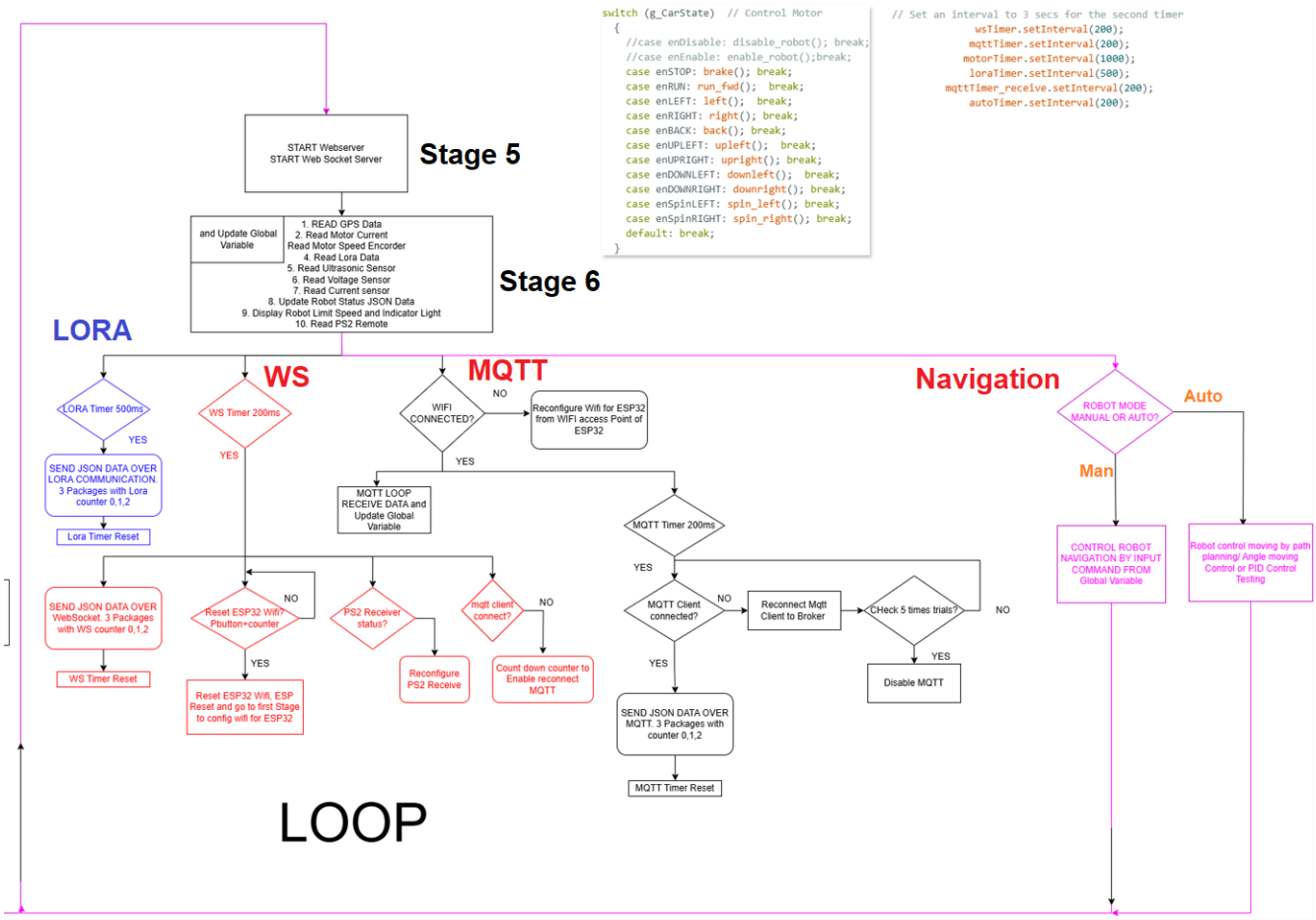


Figure 3.8: Arduino program loop flowchart for Nicobot 2025

Finally, in loop states, it will be divided into some states below:

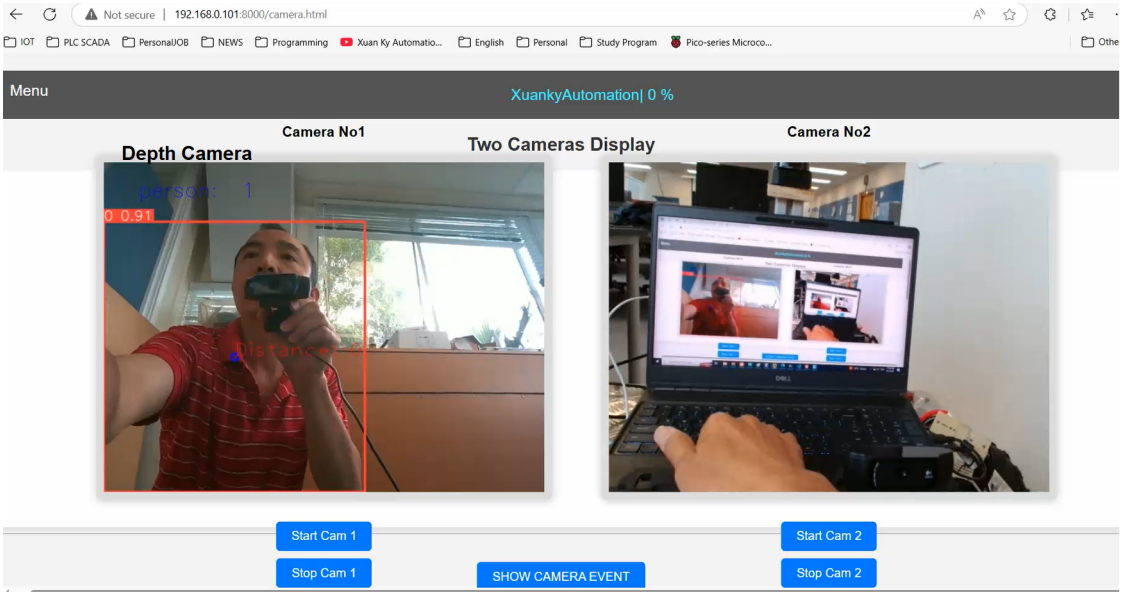
- Stage 5 Start Web, and Web Socket Server
- Stage 6
- LoRa Stage
- Web Socket Stage

- MQTT Stage
- Navigation Stage

Each stage has its own function, as the flowchart shows.

3.6.1 Object Detection and Distance Calculating

The robot's job was to use the D455 camera to estimate distances and YOLO to identify objects. Conventional measures assessed detection performance, including mean average precision (mAP), precision, recall, and depth estimation error. This section uses data from the pre-trained model yolov10n.pt from Yolo with 80 objects trained, called the COCO Dataset. The COCO (Common Objects in Context) dataset is a large-scale object detection, segmentation, and captioning dataset. It is designed to encourage research on various object categories and is commonly used for benchmarking computer vision models. It is an essential dataset for researchers and developers working on object detection, segmentation, and pose estimation tasks. COCO Dataset can be found in the link https://docs.ultralytics.com/datasets/detect/coco/



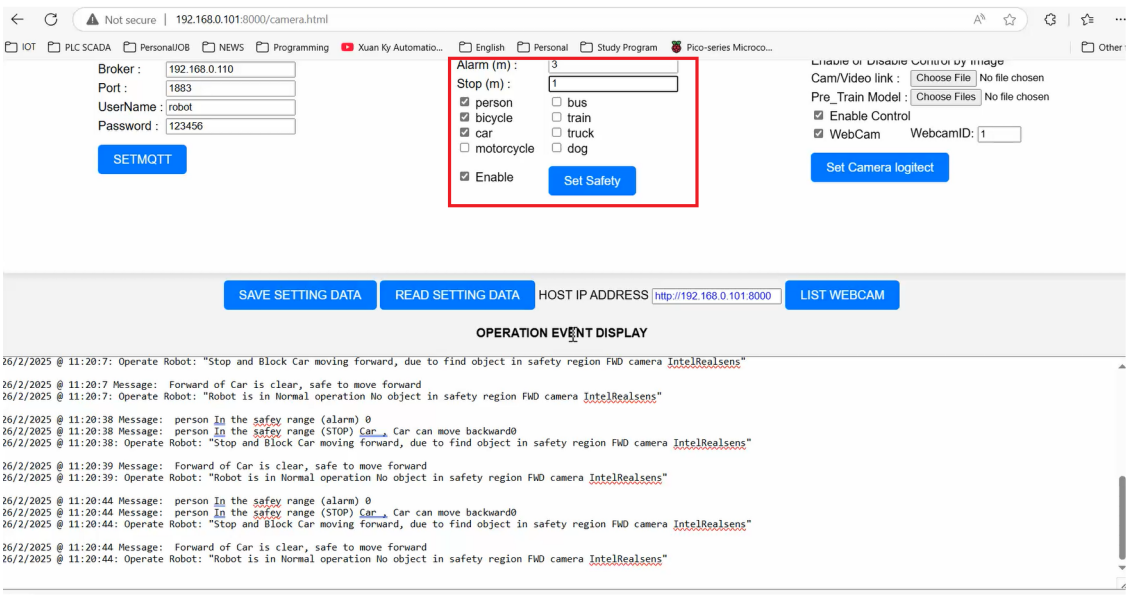


Figure 3.9: Intel RealSense D455 depth camera with YOLO for Nicobot 2025

With using Python and the YOLOv10 dataset [11], Nicobot 2025 can easily be set up to object detection and to measure the distance from the object to the robot automatically, then program will take decide whether to go forward or stop even turn left or right using MQTT communication protocol, depend on the program writing

The second camera, the Logitech camera, will be used only for streaming a camera forward of the Nicobot 2025 in the local network using a web socket. However, if using a local PC as a Python application with a PYQT6 interface, this Logitech camera is configured as a command to control the car using trained traffic signs and MQTT communication protocol.

3.6.2 Navigating and Avoiding Obstacles

Navigating the test arena was accomplished using ultrasonic sensor readings, RPLidar data, and distance from the depth camera; these data can support the Nicobot 2025 moving safely and avoiding obstruction from or behind the robot navigation area [34] [6].

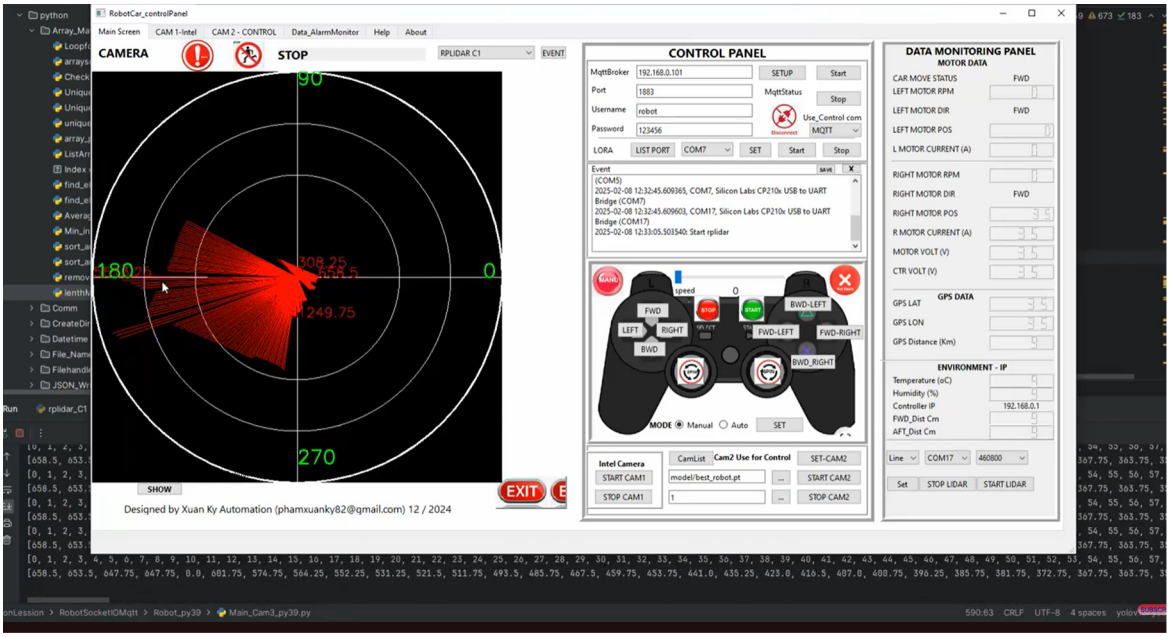


Figure 3.10: RPlidar with Python PYQT6 interface control by MQTT protocol for Nicobot 2025

RPLidar and two cameras are already attached to the Python Flask web framework [19]. Therefore, data can be displayed over the local network with any web browser. These advantages make the Python Flask web framework more convenient than the Apache web server or PyQt6.

3.6.3 Autonomous Nicobot 2025 Communication Systems

For autonomous robotic systems to operate safely and remotely monitor and control, communication must be dependable and adaptable. This project combines three communication techniques—MQTT (Message Queuing Telemetry Transport), LoRa (Long Range Radio Communication), and the PS2 wireless remote controller—to guarantee primary and backup connectivity. Each technique contributes differently to the system's overall robustness by offering telemetry, redundancy, and real-time control.

Summarize Communication control of Nicobot 2025

Communication type	Primary use	Range	Backup Using
MQTT	Real-time IoT control for robot	WiFi range	Primary communication
LoRa	Long-range backup	5–10 km	Backup to MQTT
PS2 Remote	Manual control	10–20 m	Emergency control

Table 3.2: Comminication protocol for Nicobot 2025

3.6.4 Communication via MQTT

A popular publish-subscribe messaging protocol for Internet of Things (IoT) applications is MQTT. It is perfect for real-time communication between the robot and supervisory systems because it is made for limited devices and unstable networks. MQTT characteristics:

- low packages and low bandwidth
- Real time bidirectional communication
- Scalability, flexibility (can be monitored or used over various applications)

  System Integration: An MQTT client on the ESP32 in the robot system connects to an MQTT broker (using Mosquitto) that can be run locally on the Pi or on the cloud. Use cases:

- Include remote control through the web interface hosted by Apache [15].
- Command reception from the MIT App Inventor-developed Android application [20].
- Streaming sensor data in real time for observation.

  Benefits: Scalability: Able to link to numerous cloud services or devices.

3.6.5 LoRa

Long-range wireless communication uses low-power, wide-area network (LPWAN) LoRa [28]. Operating in the unlicensed ISM frequencies (e.g., 868 MHz or 915 MHz), it is intended for low data rate but ultra-long distance transmission; hence, it is ideal as a backup communication tool for outdoor autonomous cars. Some advantages of LoRa communication: - Communication distance can reach up to 10 km.

- Extremely low power consumption from Lora Modules

- Operate flexibly in urban or rural areas with a low signal of 3G or 4G internet. - Distance can reach from 2 to 10 km, depending on the power and type of LoRa module

3.6.6 PS2 Remote 2.4Ghz

Often found in robotics and hobbyist projects, the PS2 wireless remote controller is a low-cost, dependable manual control interface. It pairs a 2.4 GHz RF communication module with a receiver hooked to the ESP32 microcontroller or RPI [7]. The PS2 remote is popular for use in robotics for several reasons: - Low power consumption

- Two analog joysticks and digital buttons; it is easy to integrate into the controller program.

Integration

When wireless or network-based technologies fail, the PS2 receiver links to the GPIO or Arduino-compatible microcontroller to provide manual control signals. In case of MQTT or LoRa communication failures, it serves as the last backup, letting the user directly steer, control speed, or stop the robot.

Use Cases:

* Emergency Manual Override

* Testing and calibrating stages.

* Direct control in settings devoid of network coverage.

Benefits:

- Relatively low latency.

- Use independently

- Cost effective

3.7 Methodology for Experimentation

A structured experimental approach was developed to assess the autonomous Nicobot 2025 system's performance, accuracy, and resilience. This approach guaranteed consistent performance of the system under several operating conditions by including both real-world testing and simulation-based validity. Four main categories—each aimed at a different facet of the system's performance—object detection accuracy, navigation performance, communication latency and dependability, and safety response—divide the experiments here.

3.7.1 Accuracy for Object Detection

Object detection is a fundamental part of the robot's autonomy. It lets the robot identify environmental elements, pedestrians, and roadblocks. A series of tests under varying lighting conditions (daylight, low light, and artificial light) and at several distances (ranging from 0.5 meters to 3 meters) evaluated the performance of the YOLOv10 and YOLOv11 object detection models integrated with the Intel RealSense D455 camera.

Target objects such as people, chairs, computers, tables, and an LCD monitor in the LAB C112 are used to make an experiment for YOLO object detection. Furthermore, traffic signs downloaded from the internet are also used to train the model for operating a robot by image processing with YOLOv11.

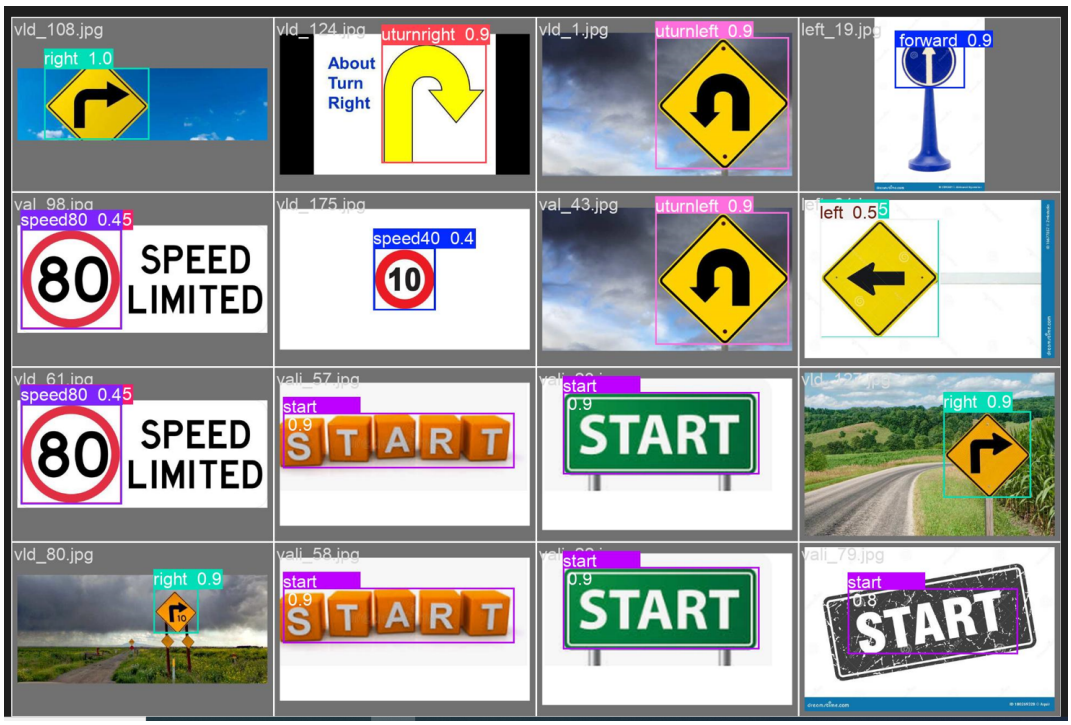


Figure 3.11: Training result 1 for Nicobot 2025

Figure 3.12: Training result 2 for Nicobot 2025

The accuracy of object detection depends on the quality of the training dataset. The figure below illustrates the **poor quality** of this dataset. To increase the accuracy of object detection for robot operation by using traffic signs, we should have a balanced number of images for each class in the dataset, and images should be taken from different angles, distances, light conditions, and so on.

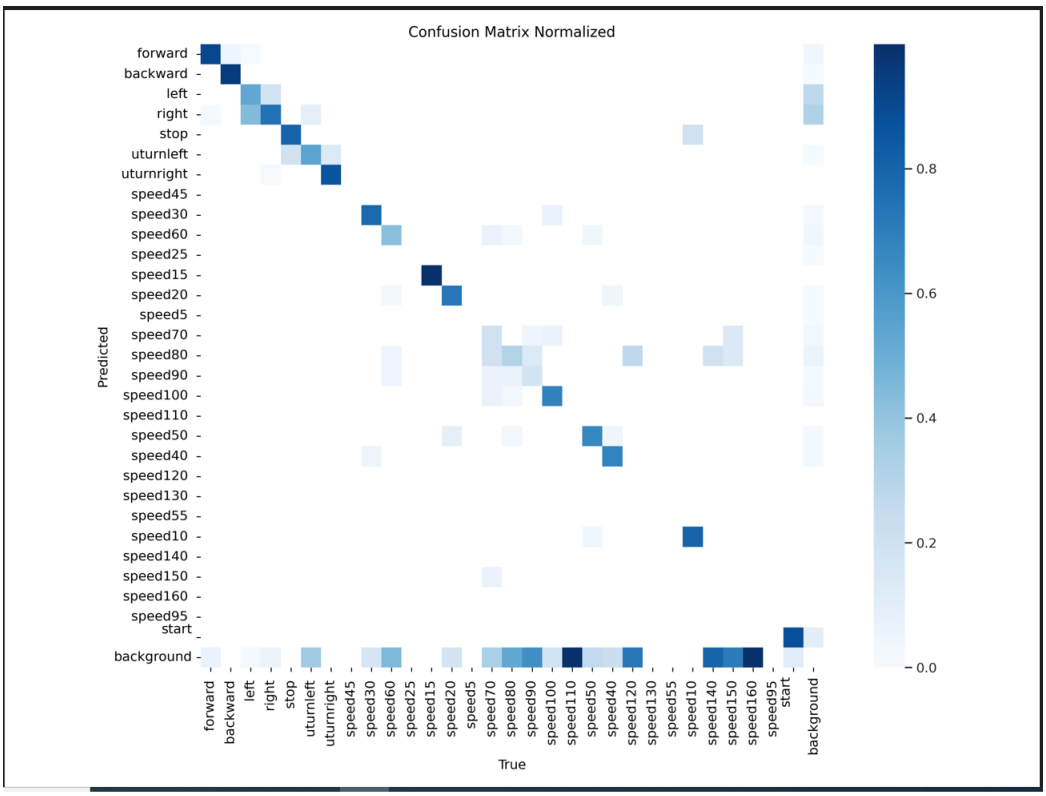


Figure 3.13: Quality of training dataset for Nicobot 2025

Python program

Calculating quantitative performance measures, including mean average precision (mAP) and intersection over Union (IoU), utilizing a comparison between detection outputs and ground-truth annotations

Included were indoor and outdoor settings to consider different backgrounds and lighting conditions, guaranteeing a reasonable evaluation of system robustness.

3.7.2 Navigation Performance

Test robot navigation by manually controlling it with the PS2 remote:

Controlled by a digital push button, the robot car moves smoothly; however, due to the fixed speed while moving, it will take longer to change direction. Consider that this control can only be used in open space where not much obstruction around the robot. By using the joystick on the PS2 remote to change direction and motor speed on the same joystick, it becomes a good performance because the robot can change speed and direction faster and navigate more accurately. Left and right motor speeds are controlled by using PWM. This technique is used as an emergency navigation in case of an internet connection failure.

Testing Robot Navigation by Using MQTT Protocol: This is the main technique in this project. The MQTT protocol is used in various applications in this design, such as Web control, Android applications, and Python applications, all of which use the MQTT protocol to send data to the ESP32 controller of the robot car. For safety, all safety devices, such as ultrasonic distance sensors and RPLidar, will be enabled to prevent the robot car from hitting objects during navigation.

Testing Robot Navigation using Image Processing:

Similar to the last steps above, we enable all safety devices for the robot to be sure that it is not hitting any

obstructions during navigation. The accuracy of image or traffic sign detection depends on the quality of the training dataset. After using the trained dataset by YOLOv11, we try to operate the robot car by using traffic images. The result of the test did not satisfy me due to the quality of the dataset. The dataset quality will be provided in the appendix section. For safety navigation, this operation mode will be under experiment in the lab only. We will hand it over to the next generation to upgrade this operation mode.

3.7.3 Reliability and Communication Latency

The robot's use of the three communication channels—MQTT, LoRa, and PS2 remote control—was tested to assess latency, dependability, and failover performance.

There were communication tests:

Using MQTT and LoRa protocols, send command signals—e.g., move forward, stop, turn—from a remote terminal. Then, timing-based Python logs measure the round-trip latency.

Compare the response times and packet loss rates between the MQTT (over Wi-Fi), LoRa (long-range radio), and PS2 (radio-based) protocols.

Running long-duration tests to assess system stability under continuous running.

These tests shed understanding on the pragmatic trade-offs between low-latency control (PS2), dependable mid-range communication (MQTT), and long-range backup communication (LoRa).

3.7.4 Safety Response with Manual Override

For autonomous systems, particularly when running close to people, safety is absolutely vital. To enable instant operator intervention, the system included a PS2 remote controller-powered emergency manual override.

The safety response analysis comprised

Creating simulated failure scenarios—such as Wi-Fi disconnection or MQTT broker failure—to see whether the system automatically turned to manual PS2 control.

Adding unexpected challenges to see if the ultrasonic sensors accurately identified the hazard and stopped the car.

Calculating the interval between failure detection and effective hand override activation.

These tests guaranteed the robot could run safely even in degraded communication surroundings.

3.7.5 Simulations and Real-World Validation

Simulations were run in Python using synthetic and real-world data before physical deployment. This level included

Training and verifying the YOLO models on benchmark object detection sets.

Simulating the PWM control motor speed while changing the motor speed by remote operations

Testing LoRa and MQTT communication scripts in isolated virtual environments to find performance bottlenecks or faults.

Operate the robot around the pathway of the university and observe the response while the robot approaches objects and functions of safety equipment or programs.

4 Chapter 4: Implementation

4.1 Introduction

This chapter explores the useful application of the autonomous Nicobot by thoroughly explaining the hardware assembly, software integration, experimental design, and system verification. It also describes the experimental techniques for assessing the system's performance and includes hardware and software block diagrams.

The main aim was to integrate object detection, depth estimation, motor control, and multi-mode communication into a single robotic system. The studies evaluated object recognition accuracy, navigation accuracy, communication robustness, and general system safety and dependability.

4.2 Electrical Schematic Diagram

Robot vehicle power and sensor integration

Careful design of the power architecture of the autonomous Nicobot guarantees dependable and constant operation of the onboard electronics and the driving system. Ten batteries specifically for the motor driving system and three more for the controllers and sensors—including the ESP32 microcontroller and the Raspberry Pi (RPI)—are part of the robot's dual power supply system. Minimizing noise and voltage fluctuations that might compromise the sensitive electronics and communication modules depends on this division of power sources.

The system combines two current sensors that constantly track the current consumed by the motors to monitor and control the power consumption properly. Using this real-time current monitoring, the system can identify aberrant conditions, including stalled motors or too heavy loads, to prevent overheating or battery drain. Furthermore, voltage sensors are placed to track the voltage levels of the controller and motor power supplies. Gathering and processing these voltage and current readings, the RPI or laptop logs data for performance analysis and implements safety mechanisms, including power cut-off in case of undervoltage or overcurrent events.

The Nicobot features a GPS module, which offers real-time spatial data for navigation and localization. Tracking the robot's geographical location during autonomous navigation helps one to monitor movement paths, log travel history, and apply location-based decision-making. Complementing the GPS, the robot has a 360-degree RPLIDAR sensor that maps obstacle distance and position by scanning the surroundings [6]. Constructing a spatial map of the environment depends critically on this lidar system, improving the robot's capacity for autonomous navigation, collision avoidance, and path planning [12].

Apart from lidar sensing, the robot is equipped with ultrasonic sensors facing forward and aft that offer limited range distances [22]. These sensors provide a safety layer so the robot may identify nearby objects or walls even in close quarters or with partial obstruction of lidar or cameras. In confined surroundings where exact distance measurement is needed to prevent unintentional collisions, the ultrasonic sensors are especially helpful.

Two cameras—an Intel RealSense depth camera and a Logitech webcam—form the robot's sight system. The robot can precisely measure the distance from objects using the Intel RealSense camera, which is used for both streaming video over the local network and depth information capturing [33]. Further video streaming capability from the Logitech camera guarantees strong visual monitoring under different environmental conditions. Using the local network, both video streams are broadcast so that remote operators may real-time monitor

the surroundings of the robot [16].

The system combines sophisticated computer vision algorithms—more especially, YOLOv10 or YOLOv11 models—for object detection and recognition [4] [4]. Using high accuracy, these deep learning models enable the robot to detect several objects in real time, enabling its visual input to guide intelligent navigation decisions based on pedestrian, obstacle, or landmark recognition [9]. Critical for safe and efficient navigation, the robot can detect and estimate the distance to objects by combining depth sensing with object recognition.

The robot uses several communication systems to guarantee strong and fail-safe transmission. MQTT over Wi-Fi offers low-latency, lightweight messaging between the robot and the supervisory system; hence, it is the main communication system [35]. The robot incorporates LoRa communication modules and a PS2 remote control as a backup. Manual override capability of the PS2 remote lets the operator directly operate the robot in an emergency or should the main communication channel fail [7]. Operating in locations with limited Wi-Fi connectivity, the LoRa module provides long-range, low-power communication that can be used to transmit necessary telemetry data or receive control commands [28].

This thorough integration of power management, sensing, communication, and control systems lets the autonomous Nicobot run dependably in dynamic and complex surroundings. Whether the robot is navigating autonomously, streaming live video, or responding to manual control inputs, real-time monitoring, intelligent decision-making, and many layers of safety ensure that it can effectively complete its tasks [12].

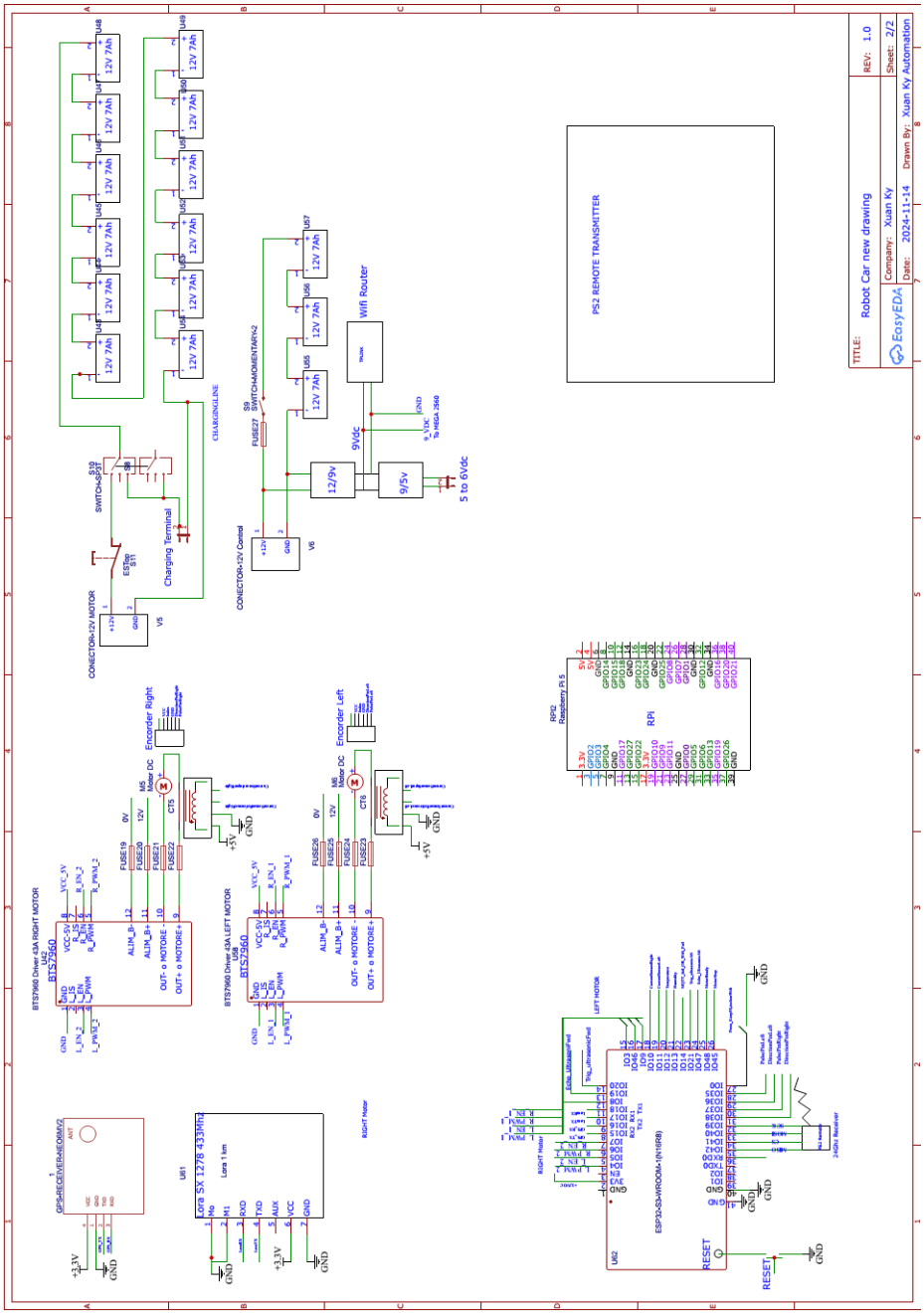


Figure 4.1: Electrical Diagram for Nicobot 2025

4.3 Specifications of the Hardware Design

The following are the primary parts of the hardware setup:

Main Processor and microcontroller:

- ESP32 (primary controller for MQTT client, LoRa receiver, and real-time operations)
- Raspberry Pi 3B+ (Apache Web Server, and MQTT broker
- Laptop (Flask Web server, Python application with YoloV10/ YoloV11 object detection, backup Apache Web Server, Backup Mosquitto Mqtt Broker)

Sensors

- Depth Camera Intel RealSense D455 (Yolo V10/V11 Object detection , Object distance monitoring Python)
- 360° obstacle scanning using RPLIDAR C1
- Both forward and aft ultrasonic sensors
- Sensors for voltage and current to monitor power
- Geolocation using GPS NEO-8M

Cameras

- Depth perception with Intel RealSense D455
- USB camera from Logitech (streaming video)

Modules for Communication:

- LoRa SX1278 for communication backup
- Wireless PS2 controller for manual operation in emergency

Source of Power:

- Ten 7Ah 12Vdc lithium batteries for powering the motor
- Three 7Ah 12Vdc batteries (sensors, Raspberry Pi, and ESP32)

4.4 Block Diagram of hardware

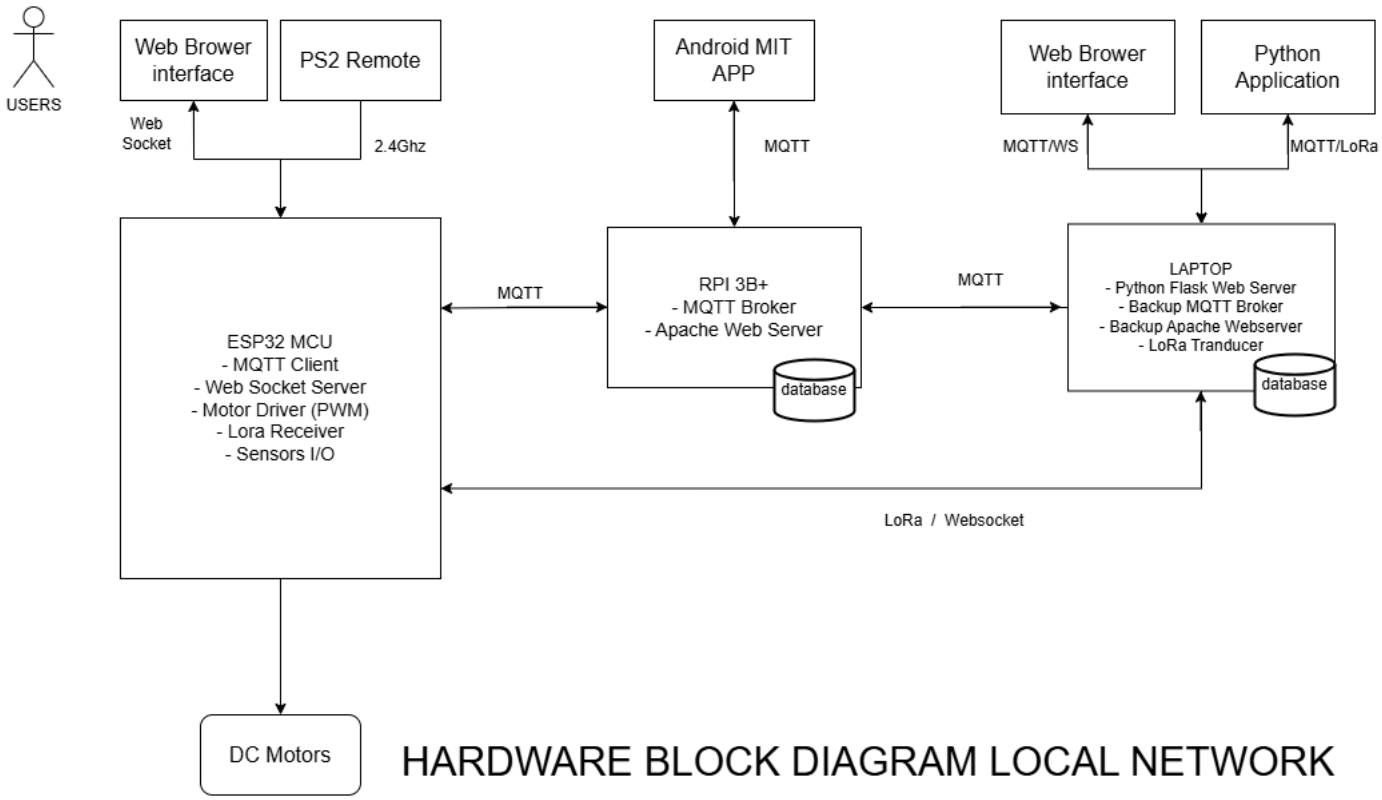


Figure 4.2: Hardware block 1 diagram Local MQTT broker connection for Nicobot 2025

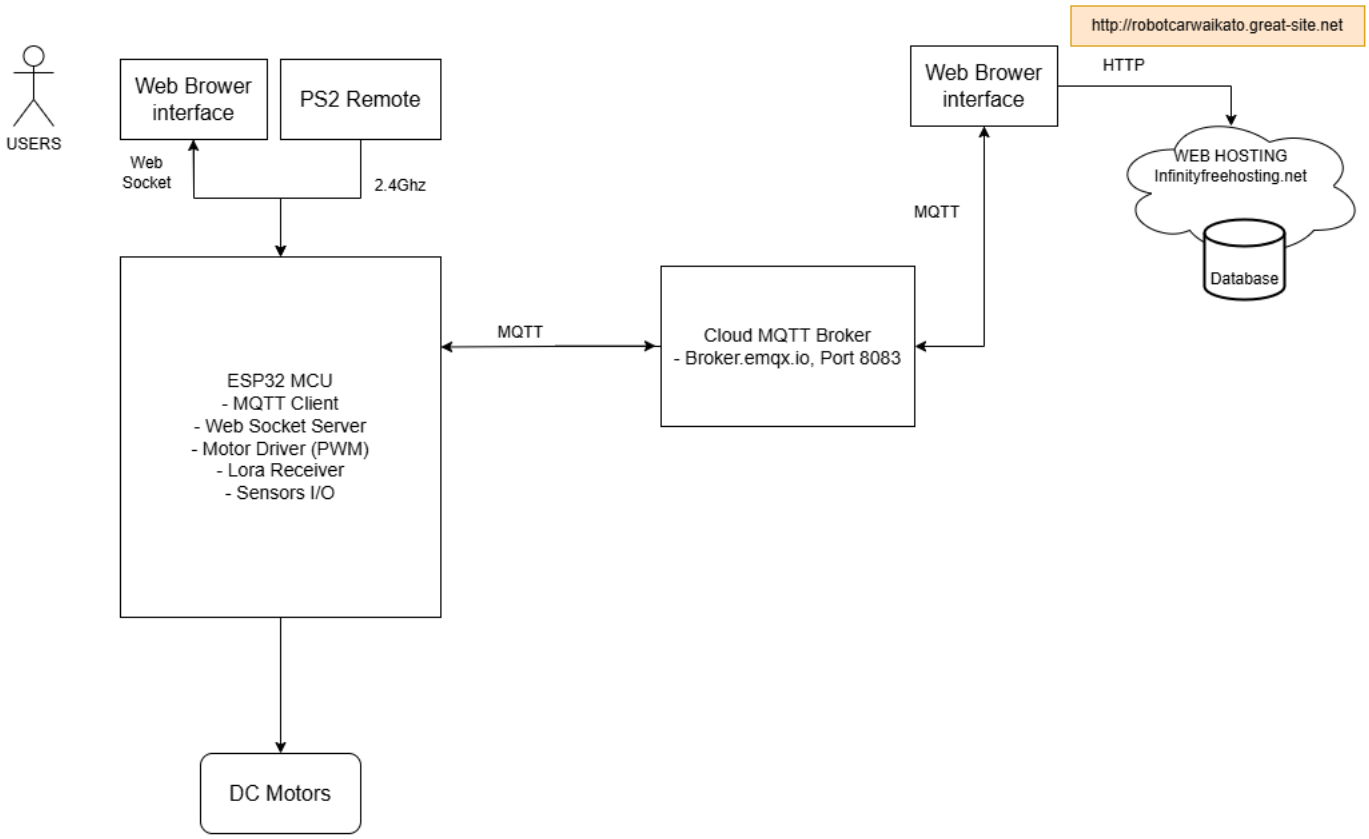


Figure 4.3: Hardware block 2 diagram cloud MQTT Broker connection for Nicobot 2025

4.5 Implementation of Software

The autonomous Nicobot's software implementation is based on a modular architecture that combines communication, computer vision, and control logic systems to guarantee effective, real-time operation. Because of their cross-platform capabilities, library support, and flexibility, Python 3, Arduino IDE, HTML, and JavaScript are the primary development languages used on all platforms.

4.5.1 Logic of Control

The ESP32 microcontroller's control logic is the brain behind the robot's motion system [14]. The control system includes:

PID Controller: A proportional-integral-derivative (PID) controller algorithm controls motor speed and steering. The PID algorithm computes the error between the intended and actual paths to lessen deviation and adjusts. This guarantees accuracy when avoiding obstacles, stability on uneven terrain, and smooth path following [26].

PWM Motor Control: Pulse-width modulation (PWM) signals are generated from the PID controller's output to power the motors via the H-Bridge circuit. Additionally, motor voltages and currents are tracked to identify irregularities and maximize energy use [25].

Sensor Integration: To control safety-critical operations, the ESP32 receives inputs from voltage, current, and ultrasonic sensors. These consist of system voltage monitoring and detecting both forward and backward object proximity.

4.5.2 Visualization

Autonomous navigation and obstacle avoidance are made possible by computer vision, which is essential for object detection and distance measurement:

The deep learning models YOLOv10 and YOLOv11 [4], implemented with PyTorch, are responsible for object detection. They offer bounding box localization and real-time classification for several objects within the camera's field of view.

Depth Sensing with Intel RealSense D455: The RealSense camera captures RGB and depth data [36]. The spatial distance between the robot and its detected objects is calculated by fusing the depth frames with the YOLO bounding boxes. As a result, the robot can precisely determine the distances of objects and adjust its navigation.

Video Processing: A local laptop running the YoloV10 or Yolo V11 vision algorithms receives the RGB and depth data streamed from the RealSense D455. The computer uses MQTT to send control data to the robot and manages object detection in real time by using a Python program.

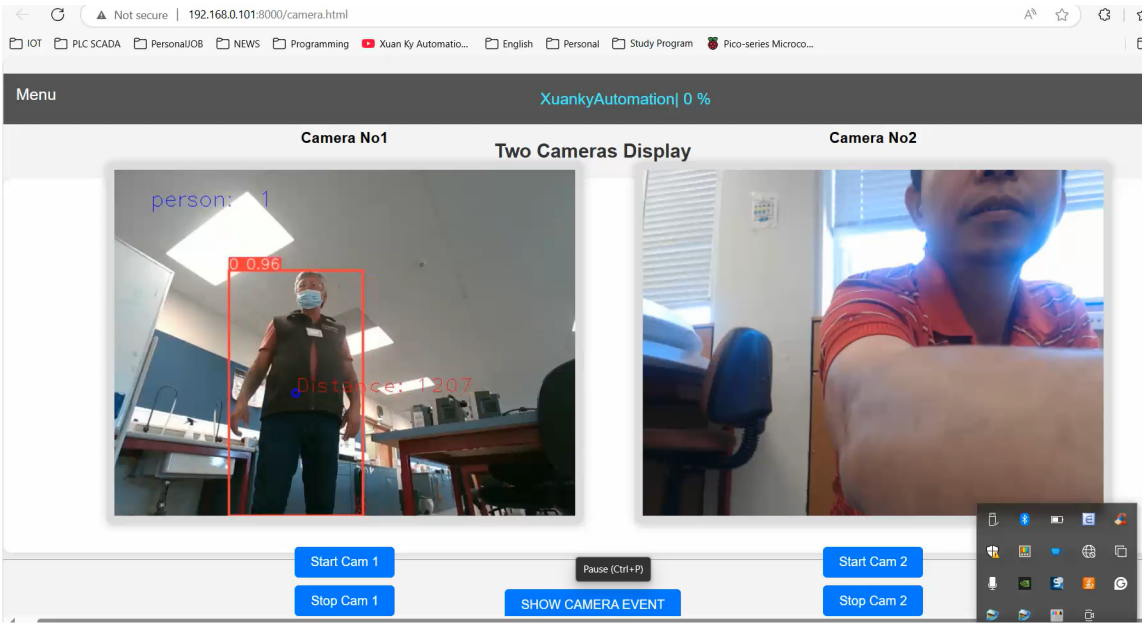


Figure 4.4: Yolov10 object detection for Nicobot 2025

4.5.3 Interaction and User Interface

A key element of the robot system is communication, which enables remote control and real-time data transfer.

The MQTT protocol makes low-latency communication between sensors, controllers, and distant clients possible. The Paho MQTT library is implemented on both the ESP32 and Raspberry Pi. Several MQTT topics are used for telemetry, control, and sensor feedback, and the Raspberry Pi is a broker [37].

LoRa Communication: The ESP32 and a laptop are connected to LoRa transceivers, which create a secondary communication channel. LoRa offers a long-range, low-power wireless link for emergency or low-

bandwidth control if the primary channel fails [28] [8].

PS2 Remote Control: Python libraries integrate a PS2 controller and decode data packets for manual override. When wireless communication is unsuccessful, it serves as an emergency control system, providing complete user control [7].

The Nicobot's user interface is created using HTML and JavaScript. It is mainly hosted on a Raspberry Pi 3 B+ running an Apache web server. To guarantee constant accessibility and control, backup web servers are set up on the ESP32 microcontroller and a local laptop running both Apache and Flask.

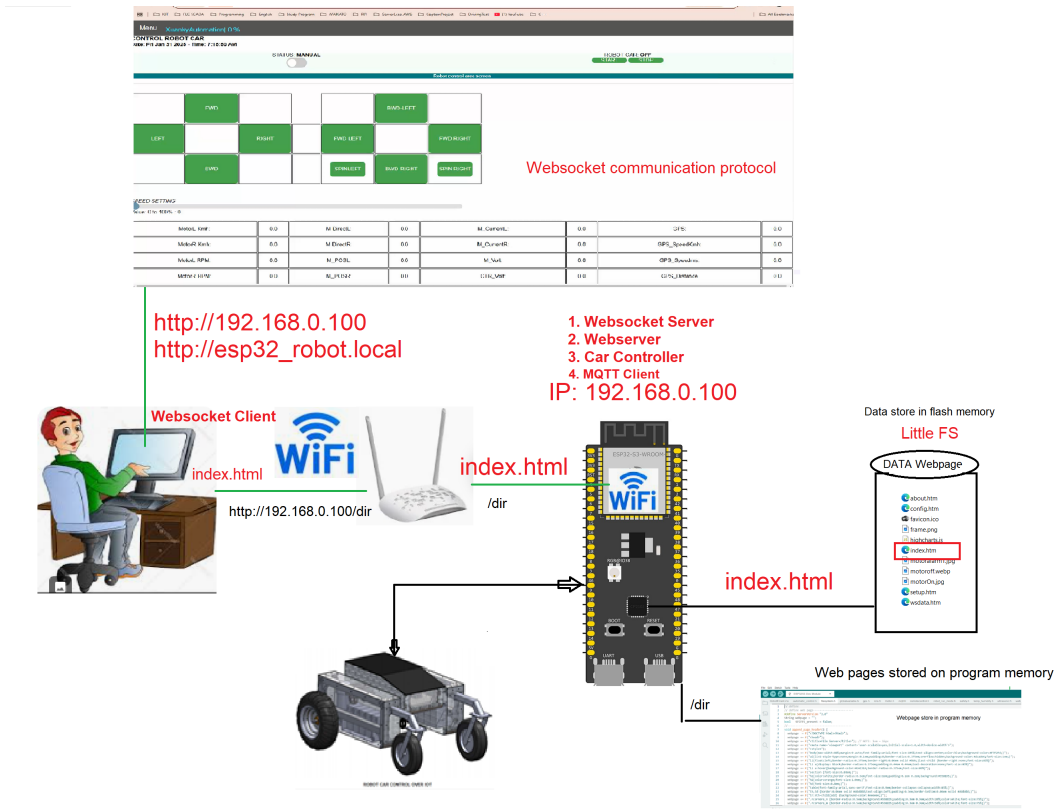


Figure 4.5: Access web server on ESP32 for Nicobot 2025

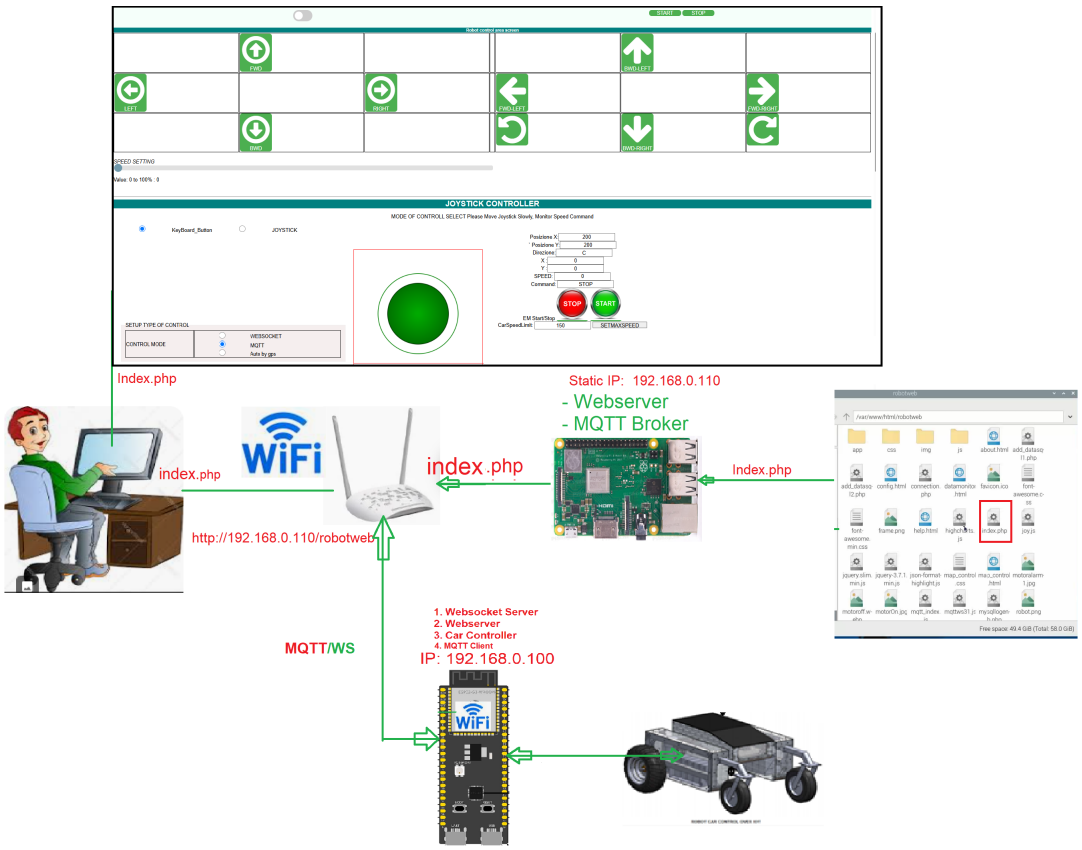


Figure 4.6: Access web server on RPI for Nicobot 2025

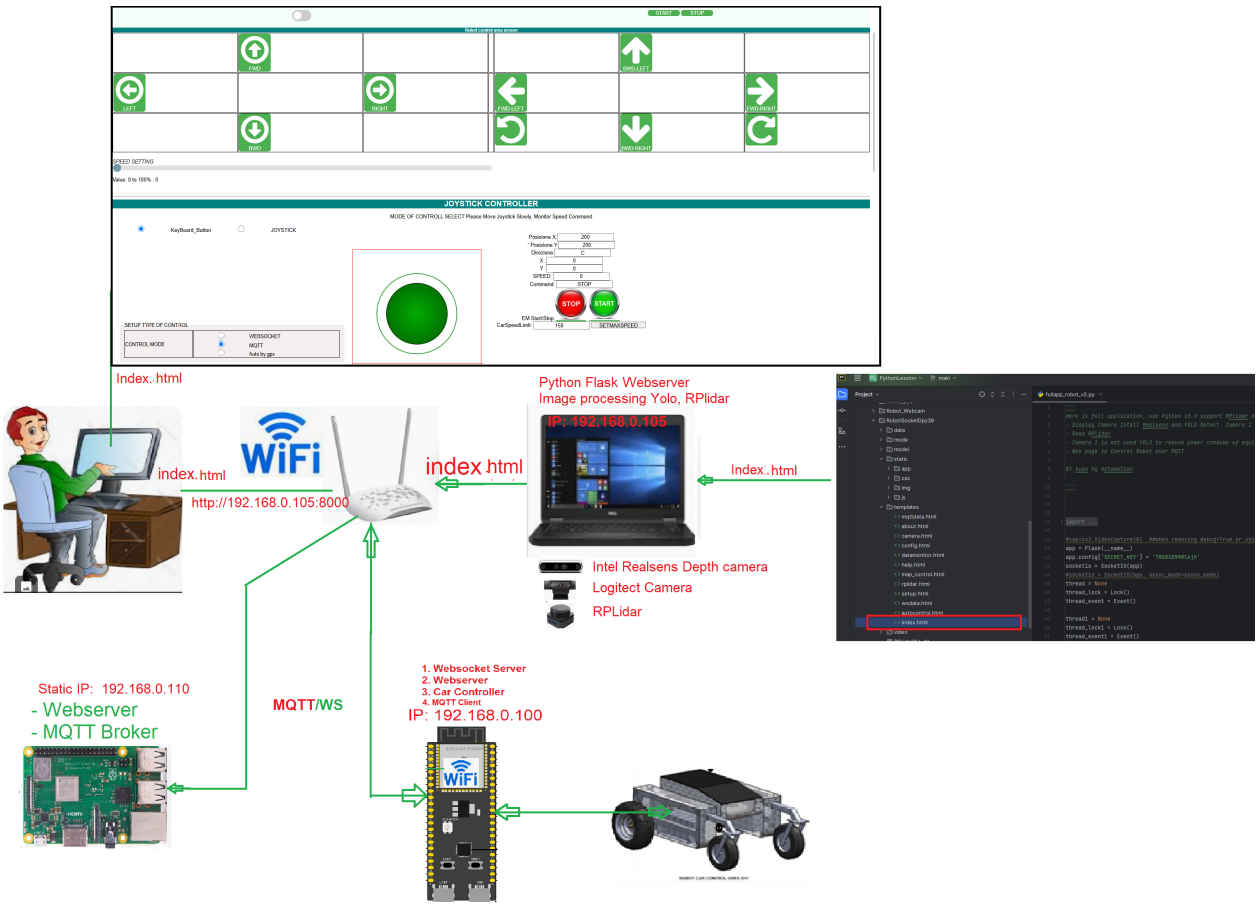


Figure 4.7: Access web server on Laptop Python Flask for Nicobot 2025

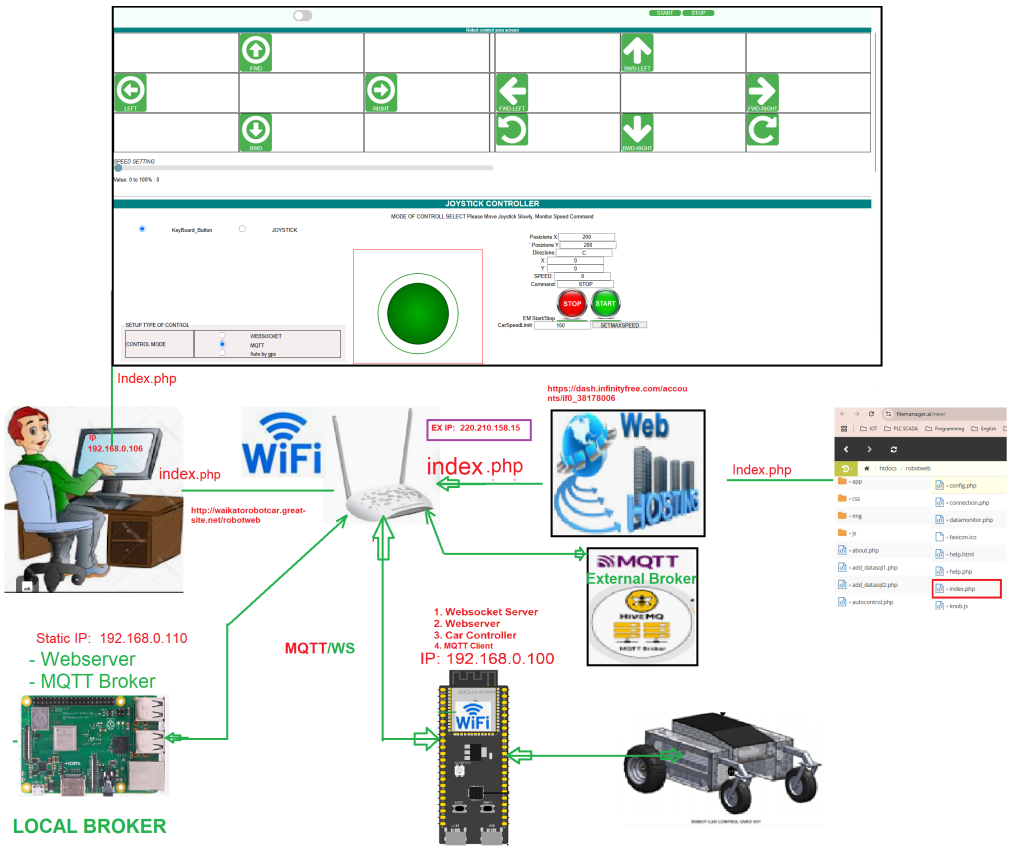


Figure 4.8: Access web server on WebHost internet for Nicobot 2025

MIT App Inventor Android App: Thanks to a lightweight Android app created with MIT App Inventor, users can control the robot via MQTT from a smartphone. The app receives real-time sensor updates and commands from the robot [30] [20].

Figure 4.9: Android App for Nicobot 2025

5 Chapter 5 Experimental Setup and Testing

5.1 Description of Test Environment

The controlled and encouraging academic surroundings of the testing and experimental activities for the autonomous Nicobot were evident. Laboratory C112 within the Faculty of Science and Engineering, University of Waikato, New Zealand, conducted the main testing. This lab offered a private, well-lit, large working space perfect for developing, programming, and testing embedded robotic systems.

Designed with desks, electrical outlets, internet access, and low foot traffic, Laboratory C112 is appropriate for concentrated work and safe indoor navigation experiments. Under steady conditions, the lab's controlled layout and ambient lighting allowed repeated testing of image-processing techniques and sensor accuracy. Most of the programming was done here using tools like Python, Arduino IDE, and web technologies (HTML and JavaScript)—critical for the motor control logic, user interface design, and real-time communication systems for the robot.

Parts of the software development also took place at home with spare ESP32 and Raspberry Pi controller boards. This enabled continuous iteration and debugging outside of lab hours, enhancing the development process and guaranteeing modular software deployment on many platforms.

Apart from lab testing, the robot was also tested outside Block C of the University of Waikato. These outdoor tests comprised object detection with the Intel RealSense D455 depth camera, RPLIDAR, and ultrasonic sensor navigation, and object avoidance. The natural terrain provided a perfect setting for verifying image processing, path planning, and system dependability, including ramps, uneven surfaces, and changing lighting.

Indoor lab testing in C112 and outdoor field trials in Block C, taken together, guaranteed that the Nicobot was developed and tested in a realistic and complete environment. This lets one assess all important subsystems—control, communication, perception, and safety—in both controlled and dynamic environments.

Figure 5.1: Experimental setup 1 for Nicobot 2025

Figure 5.2: Experimental setup 2 for Nicobot 2025

5.2 Preparing for Performance Tests

Throughout the development process, several iterative tests were carried out to meet the performance requirements of the autonomous Nicobot project. In parallel, hardware and software components were continuously improved and adjusted. These iterative changes were necessary to improve the robot's overall operational efficiency, system stability, and navigation accuracy so that the vehicle could function dependably in real-world settings.

5.2.1 Emergency Operation Testing Operation with PS2 Remote

The autonomous Nicobot's safety and dependability concerns included an emergency manual control mechanism employing a PS2 wireless remote controller. This subsystem is a fail-safe solution if software faults, network delays, or hardware failures compromise the main autonomous navigation or wireless communication, e.g., MQTT, LoRa, or WebSocket.

Using a 2.4 GHz RF receiver module connected via SPI interface, the PS2 controller talks straight with the ESP32 microcontroller [31]. The PS2 remote gives direct user control over motor speed and steering by sending control signals that supersede the autonomous control algorithm. The robot instantly leaves autonomous mode and turns to manual control upon being triggered, enabling the user to have complete movement control.

Process of Testing: The PS2 remote was used to disrupt the robot's autonomous routine after it was in a navigation scenario. Constant testing of the change from autonomous to manual mode guarantees flawless switching without delay.

The latency of control signal transmissions from the PS2 remote to the robot was measured and found to be almost instantaneous. This enables users to react quickly to prevent system faults or collisions.

To ensure range and signal dependability, the PS2 controller was tested at several line-of-sight conditions and distances up to 20 meters.

To ensure the PS2 system could effectively override and recover control, simulated communication failures in MQTT and WebSocket channels were included.

Outcome:

Relatively speaking, the PS2 remote control system served as an emergency fallback mechanism. It displayed a strong control response, low latency, and consistent signal transmissions within its running range. Especially during field testing or indoor laboratory experiments, the system was a valuable and efficient backup during unplanned failures or debugging procedures.

This manual override mechanism is important for both development and real-time applications where unexpected behavior or environmental challenges may arise. It greatly increases the robot's safety and maintainability.

Figure 5.3: PS2 Remote control for Nicobot 2025

5.2.2 Testing Control Nicobot by web server directly on ESP32

Direct on-ESP32 microcontroller deployment of a dedicated web server enabled lightweight, real-time local control of the Nicobot. Built out of HTML, CSS, and JavaScript, this embedded web interface included control buttons for forward, reverse, left, right, and stop commands. Perfect for local network environments or mobile hotspot conditions, this onboard web server lets users run the robot without depending on outside brokers or PCs.

Prepare, set, and configure.

The ESP32 controller started up successfully and connected with the Wi-Fi router. Procedures of Testing:

The Nicobot was driven in a local Wi-Fi environment. A serial monitor yielded the ESP32's IP address, which was then accessed via a browser on a laptop and smartphone.

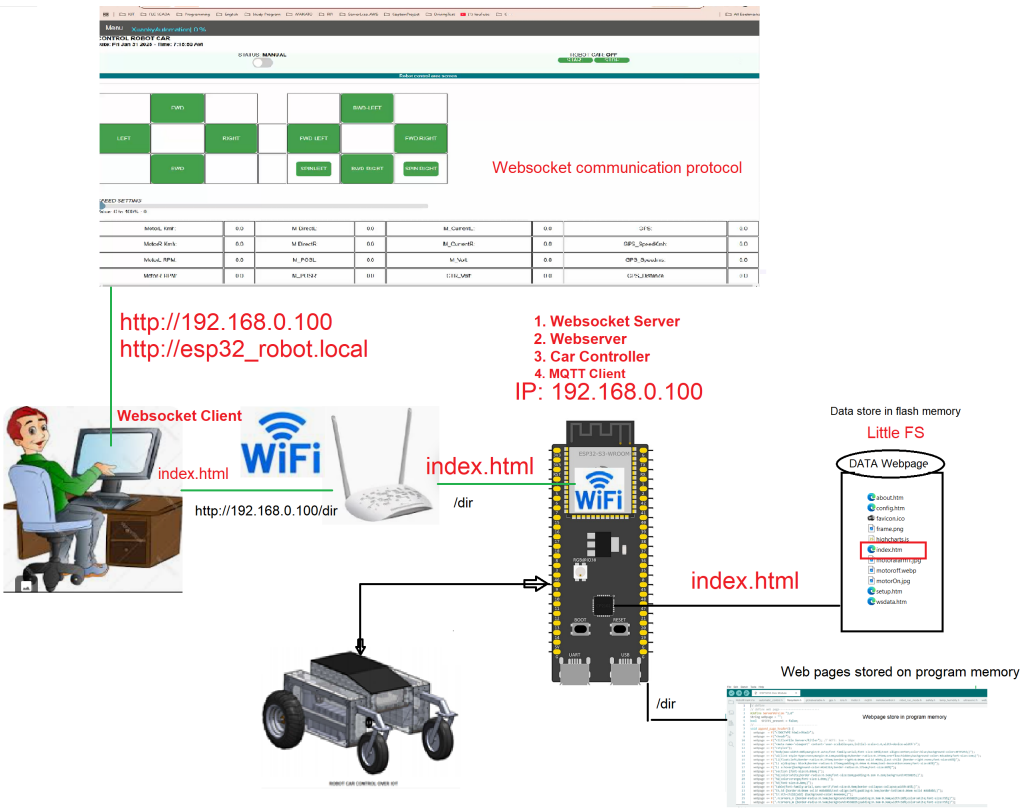


Figure 5.4: Access Web Server on ESP32 Nicobot 2025

Using a web server on ESP32 with WebSocket communication protocol, commands were created on the HTML interface by button clicks; the delay between UI input and robot action was noted. Usually running under 100 milliseconds, the response time allowed almost real-time control.

To evaluate fit, the system was tested across several browsers—Chrome, Firefox, and Android browsers—and platforms—Windows laptop and Android phone.

Control performance was assessed over a range of distances inside the Wi-Fi router coverage, roughly 20 to 30 meters indoors, to guarantee consistent connectivity and command delivery.

Result:

Direct hosting of the web interface on the ESP32 gave the Nicobot responsive control. When MQTT or cloud-based control was not accessible, it provided a portable and handy substitute that functioned well inside a local network. The HTML interface's simplicity made it easy to navigate for quick demos, diagnostics, or hand-operated driving situations.

Early development and debugging stages, when direct access to the robot's controls was crucial without depending on outside infrastructure, were constructive for the onboard ESP32 web server. However, because of its limited computing resources, this approach is best suited for simple control, not demanding processing chores like video streaming or advanced telemetry.

5.2.3 Testing Control Nicobot by Webserver on RPI

In this phase of testing, the Raspberry Pi 3B+ was used as a dedicated web server and MQTT broker to enable remote control of the Nicobot across the local area network (LAN). This configuration assigned image processing and sophisticated computational chores to a separate local laptop while concentrating only on lightweight

server operations.

Web Server Design and Operations Running an Apache HTTP Server, the Raspberry Pi was set to run an HTML, CSS, and JavaScript custom-built web dashboard. This dashboard lets users engage with the Nicobot through simple directional controls and system status displays. To guarantee constant access from client devices on the same network, the IP address of the Raspberry Pi was statically assigned to 192.168.0.110.

Two communication systems were used to link the ESP32 controller to the web interface: WebSocket and MQTT.

Enabled real-time low-latency communication straight from the browser to the ESP32. Commands issued on the dashboard—forward, stop, turn—were sent over WebSocket and picked up by the ESP32 for instantaneous motor action.

The Raspberry Pi was also set to run a Mosquitto MQTT broker, enabling publish/subscribe communication between the dashboard, ESP32 controller, and any extra clients, such as Android apps and monitoring tools. ESP32 is an MQTT client. It subscribes to the topic Nicobot/control to receive commands from the user interface to control the robot's operation, and ESP32 publishes all data from the Nicobot to the web user interface by topic public Nicobot/feedback.

Control messages were sent to the topics robot/control to the ESP32, receiving external commands. The topic of Robotcar/feedback is ESP32 sending robot status or data to the web interface.

Test Procedure: Network Accessibility Test

Entering the static IP http://192.168.0.110/robotweb into a browser verified access to the web dashboard from several devices (PC, laptop, smartphone).

Verified responsive loads and complete rendering of every dashboard element.

WebSocket Control Test:

activated motor control buttons across the interface and tracked robot response.

Minimum (<100 ms) latency between button press and motor reaction guarantees real-time responsiveness.

ESP32 effectively received commands via a WebSocket server housed on the ESP32 itself.

MQTT Test:

Commands sent via MQTT were logged to the Mosquitto broker on RPi and subscribed to by the ESP32.

The dashboard showed real-time publication of the ESP32's sensor feedback, which included voltage and current readings.

MQTT messaging's dependability was assessed under several network environments, so verifying constant delivery free from packet loss.

Testing System Integration:

Confirmed that RPi, ESP32, and other control clients—Android apps among them—could run concurrently without conflict.

Verified RPi's capacity to be a consistent MQTT broker even during prolonged running.

Limitations and Roles' Delineation The Raspberry Pi also ran no image processing or computer vision algorithms. Tasks including object detection using YOLOv10 and YOLOv11 and camera input processing from the Intel RealSense D455 and Logitech webcam were handled totally by a local laptop running a Python

environment and Flask web server. This split guaranteed the Raspberry Pi stayed committed to lightweight, real-time server duties free from performance constraints.

Summary of Findings The web server built on Raspberry Pi provided a consistent and user-friendly interface for Nicobot control. The dual-protocol support guaranteed simple access and allowed flexibility in communication techniques through the static IP configuration. Its function as an MQTT broker helped ensure flawless integration of several control clients over the system architecture. Over a local network, the design effectively fulfilled the demand for a responsive and modular remote control solution.

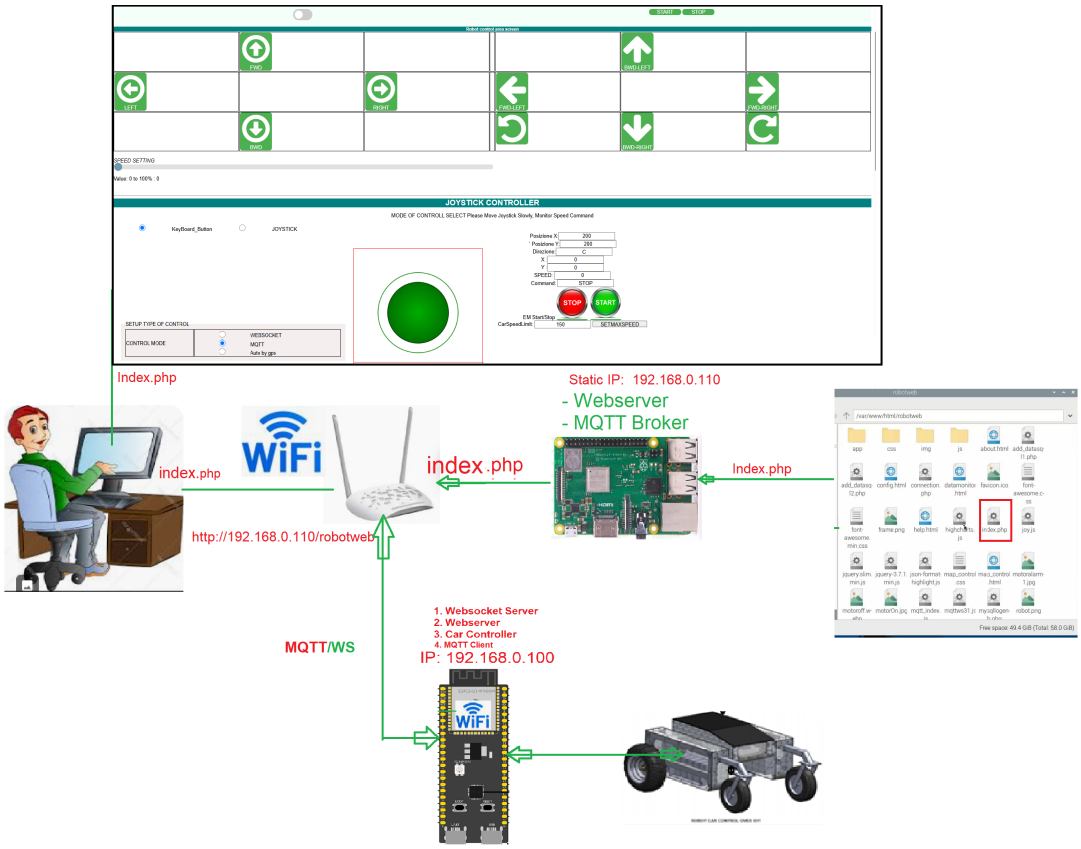


Figure 5.5: WebServer on RPI for Nicobot 2025

5.2.4 Testing Control Nicobot by Web Server on Local Laptop by Python Flask

The local laptop was absolutely essential for the system architecture, acting as both a backup server and the main host for advanced processing chores. Although it was also set up with an Apache web server and a Mosquitto MQTT broker as backup, this section especially addresses the testing of the Nicobot under a Python Flask-based web server running on the laptop.

Web server and Flask framework A lightweight Python web server built with the Flask framework made real-time monitoring and control of the robot possible through this dynamic web interface housed on this server. With its directional controls, live video streams, sensor feedback, and system status—a centralized and straightforward platform for interaction—the interface gave [29].

Vision Integration and Image Processing Two camera systems came built into the laptop: a standard Logitech webcam and an Intel RealSense D455 depth camera [33]. YOLOv10, a deep learning object detection method run on PyTorch, handled real-time image streams from both cameras. Using the RealSense depth data, the robot could precisely estimate distances to find objects, allowing it to recognize and respond to nearby

hazards [11].

Concurrent with this vision, data streaming over the local network, the Flask dashboard lets remote users visually track the robot's surroundings and behavior in real time.

RPLidar and Sensor Integration Apart from camera-based vision, the laptop managed the RPLIDAR (rotating laser distance sensor) by means of Python drivers. Further improving obstacle detection and mapping capabilities, the lidar data gave a 360-degree scan of the robot's surroundings. [6]

MQTT correspondence with ESP32 Made possible by the Paho MQTT Python client, the Flask application interfaced with the ESP32 controller via MQTT protocol. Via this avenue of communication:

Published from the Flask server to the ESP32 were command messages (e.g., movement directions, operation modes).

The web interface displayed real-time monitoring after receiving sensor data—e.g., current, voltage, and system state—from the ESP32.

This bidirectional communication guaranteed a complete view of the system's status and sensor feedback, as well as remote control capability.

Procedure of Testing: Initialization of Flask Server

Launched on the local laptop, the Flask server could be accessed on the local network using the laptop's IP address. Several client devices were used to verify the dashboard's accessibility.

Test object detection and camera stream

The dashboard shows the video feed from the RealSense and Logitech cameras.

Real-time detection and classification performance of YOLOv10 was assessed under several illumination levels and object distances.

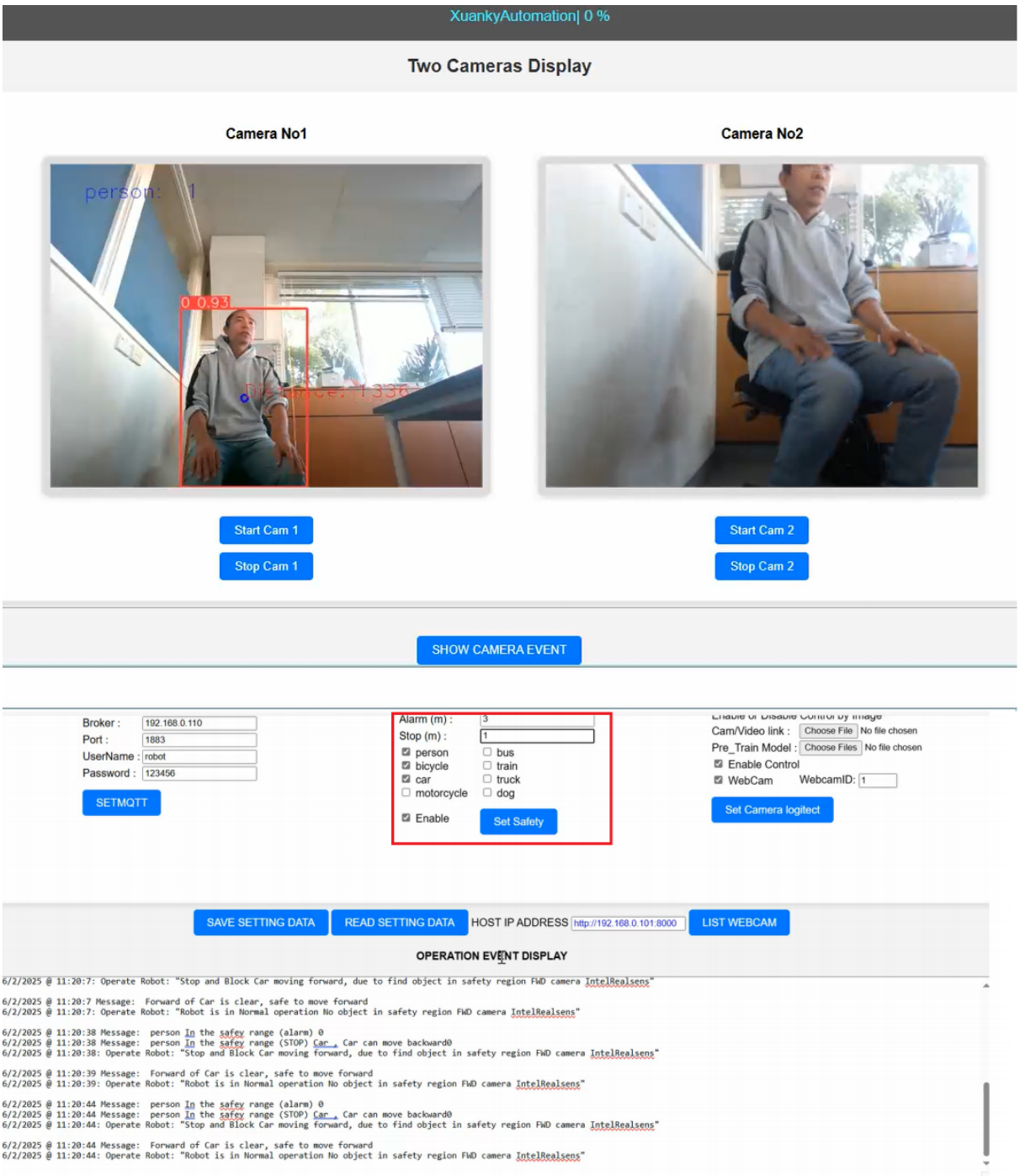


Figure 5.6: Object detection on Python flask for Nicobot 2025

RPLidar Operation Test

The lidar scanning module built on Python turned on and showed up on the dashboard.

Accuracy was evaluated by matching obstacle data with actual object locations.

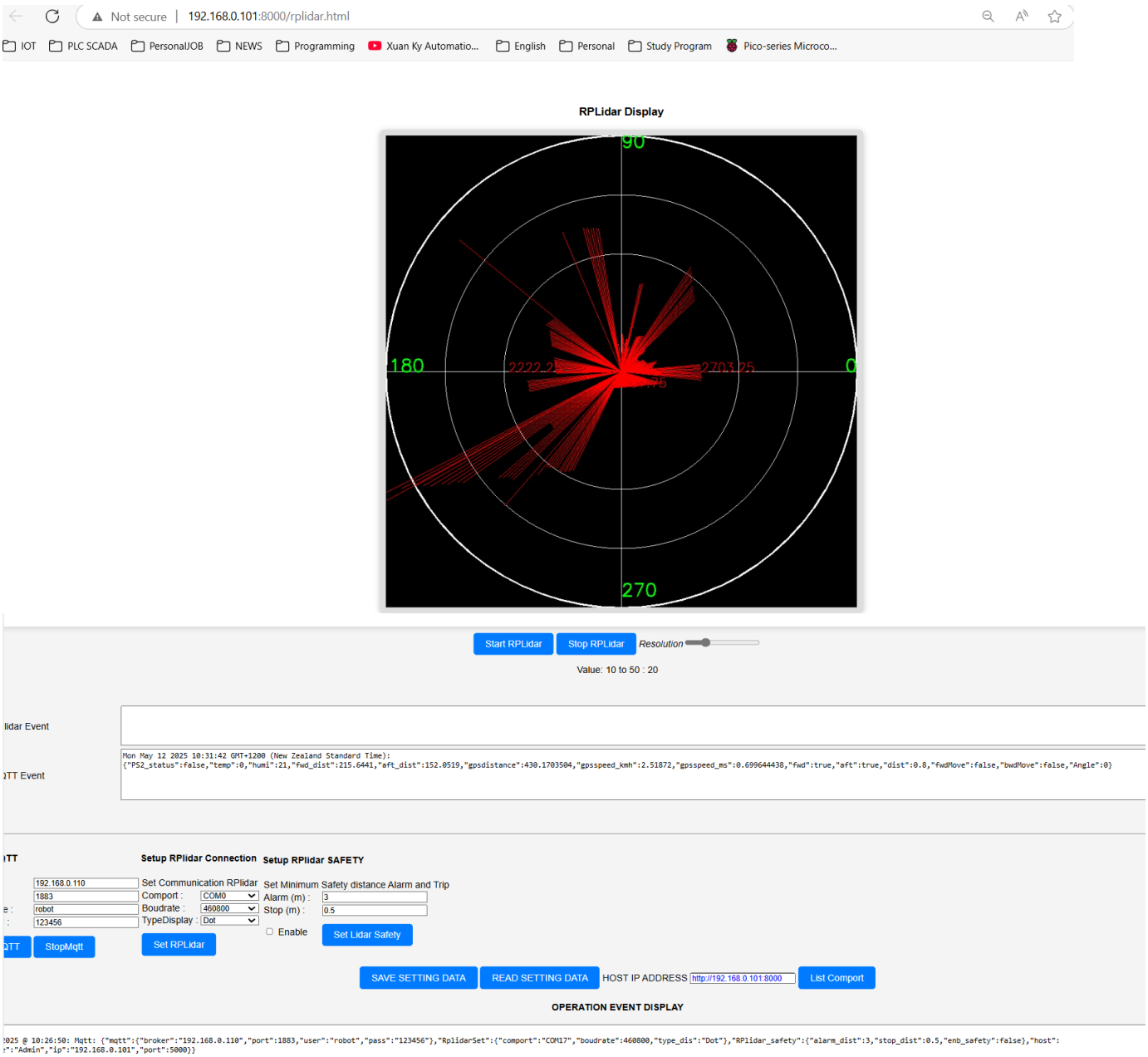


Figure 5.7: RPlidar indication on web for Nicobot 2025

MQTT Command and Response Test

Published to the ESP32 were movement commands issued via the dashboard.

The dashboard showed the robot's returned sensor data live, confirming real-time data exchange and responsiveness.

SUMMARY The tests carried out on the local laptop via the Flask web server verified its strong capacity to function as a central control unit for the Nicobot. It presented a strong, flexible control environment combining vision-based processing, lidar scanning, and MQTT-based communication. This configuration proved especially useful for jobs needing advanced computing capability and real-time sensor integration, which the ESP32 or Raspberry Pi by itself could not effectively handle.

5.2.5 Testing Control of Nicobot via Web Server on Internet Hosting

The system was set to link with an online MQTT broker and a publicly hosted web server so the Nicobot could operate remotely from any point with internet access. Without direct access to the local network, this arrangement lets consumers use a standard web browser to control the robot in real time.

MQTT Settings for Online Connectivity Operating on port 1853, the ESP32 microcontroller was set to create an MQTT connection with the public broker broker.emqx.io. This online-based broker serves as a link between the ESP32 and remote clients by means of consistent bidirectional communication across the Internet. It guarantees that control commands and telemetry data can be sent and received in real time, independent of the user's physical location.

Web server and hosting platform Developed on a free web hosting service supplied by InfinityFree, the robot control interface was created with HTML and JavaScript. One may access the hosting account from the dashboard at https://dash.infinityfree.com/accounts.

Publically accessible control interface for the robot is found at: http://waikatorobotcar.great-site.net/robotweb. Users of this interface can send motion commands, switch modes, and instantly get robot feedback.

Wi-Fi and Online Needs The Wi-Fi network the Nicobot is connected to has to have steady internet access if the system is to run as it should. This guarantees that web page commands can reach the robot unhindered and that the ESP32 can keep an active connection to the internet MQTT broker.

Summarizing Tests Remote control functionality tested over the Internet verified that the system is capable of running responsively and stably. The control interface kept on the public web server effectively interacts with the ESP32 via the EMQX broker, making full remote control over robot movement and status monitoring possible.

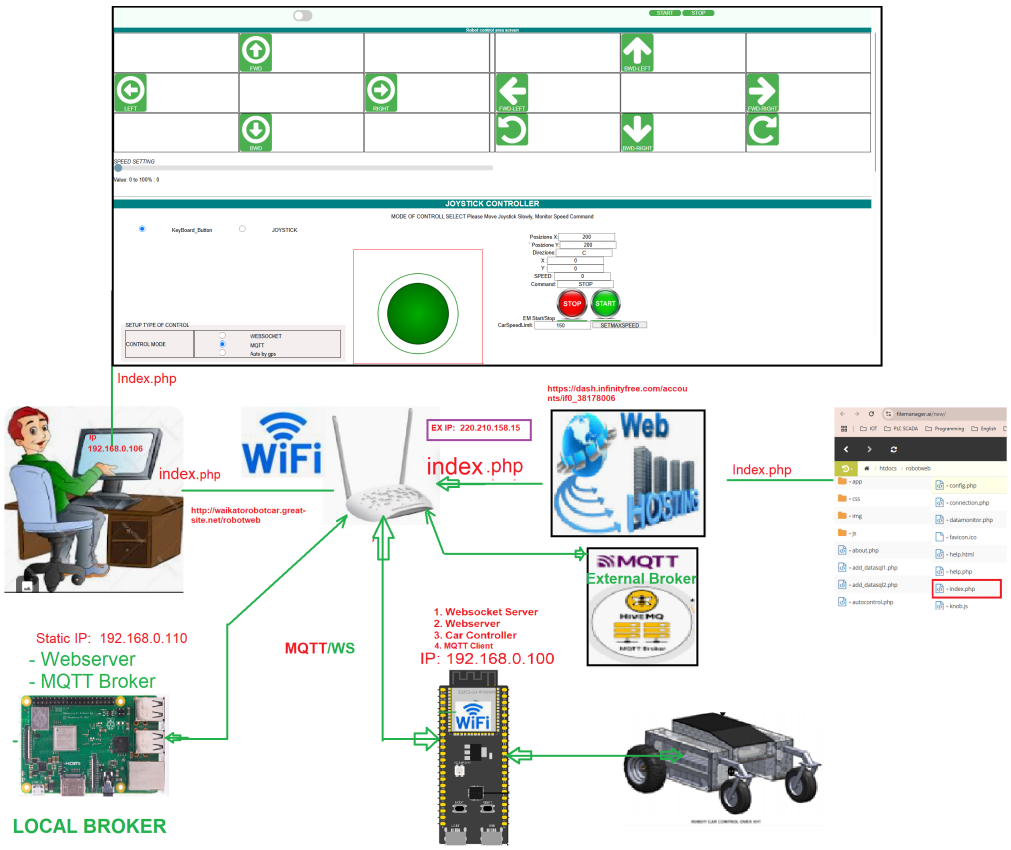


Figure 5.8: Control robot over the internet website for the Nicobot 2025

5.2.6 Testing Control Nicobot by MIT application on an Android phone

MIT App Inventor helped develop an Android-based control interface for the Nicobot, keeping portability and user accessibility enhancements in mind. This lightweight but effective substitute for web-based control systems lets users operate the robot with a smartphone via the MQTT communication protocol.

Features and Design Application-wise The mobile app's user-friendly interface allows real-time control of robot operations, including forward/reverse movement, turning, stopping, and mode switching. Designed with MIT App Inventor, the app fits a broad spectrum of Android devices since it supports MQTT messaging with low latency and requires extremely few system resources.

MQTT Setup. Either a local Mosquito broker (for example, running on a Raspberry Pi or a local laptop) or a cloud-based broker like broker.emqx.io: the application links to the MQTT broker. Once set up, the broker address, port, and topics are kept within the app to guarantee fast and flawless reconnections free from repeated setup.

Local and Online-Based Authority Testing verified that the Android app could consistently control the Nicobot over the internet and inside the local network. Connected to the same Wi-Fi as the Nicobot, the app talks straight with the local broker. The app enables complete remote capability for remote operations as long as the Nicobot's Wi-Fi is linked to the internet and the ESP32 is set to use a public MQTT broker.

Benefits and Notes of Observations The mobile app's portable and easy-to-use control method was quite handy, particularly in field testing environments.

Users could monitor and operate the robot with very low delay.

The app turned out to be a consistent backup and complementing tool, complementing the web-based interface and PS2 remote.

The MIT app-based control system proved overall efficiency, simplicity of use, and strong performance across many network environments, adding much to the robot's adaptability.

Figure 5.9: Control by the Android Application for the Nicobot 2025

Voice command is also implemented in the MIT application. This allows users to control the robot by sending commands by voice, such as "Move forward, "Turn left," "Turn right," "Move backward," etc.

5.2.7 Testing operation of motor current and voltage sensors

It is essential to monitor the Nicobot's power usage to guarantee operational safety, boost productivity, and identify issues. To accomplish this, both voltage and current sensors were incorporated into the system and tested in a range of operating environments.

Configuration of the Sensor The two current sensors were linked to a DC motor powered by PWM (pulse width modulation) signals. These sensors make it possible to measure each motor's current draw in real time. Furthermore, voltage sensors (ESP32, Raspberry Pi, and connected sensors) were installed to track the supply voltage of the controller and motor power systems. Using its ADC pins, the ESP32 microcontroller is in charge of gathering analog readings from these sensors.

Method of Testing To confirm the sensors' accuracy and responsiveness, they were tested in a variety of operating conditions:

Idle Condition: Baseline voltage and current levels could be recorded because the robot was turned on but motionless.

Low-Speed Movement: To test sensor readings at low load, the motors were driven with low PWM duty cycles (30

High-Speed Motion: Full-speed motion (90–100

Obstacle Encounters: To confirm overcurrent detection, the robot was instructed to push against a barrier, mimicking a high-load situation.

Battery Drain Test: The robot was run continuously to track trends in voltage drops and battery behavior

over time.

Findings and Remarks The current sensors successfully recorded different current levels in relation to the motor load and PWM duty cycle. An increase in load confirmed accurate performance or resistance caused a corresponding increase in current.

Thanks to the voltage sensors' steady and accurate readings, both the motor and controller power supplies could be monitored.

The MQTT transmission of sensor data to a web interface for live monitoring made early detection of anomalies like power dips or overcurrent events possible.

The ESP32 firmware was enhanced with safety features that allowed the motor to shut down automatically if current readings exceeded predetermined thresholds.

In conclusion The voltage and current monitoring system operated dependably throughout the testing process. By incorporating these sensors into the control loop, the robot can instantly identify and respond to possible power-related problems. This feature increases the system's resilience, guards against hardware damage, and promotes safer autonomous operation.

5.2.8 Testing speed of Nicobot display on web page by line chart

By changing motor PWM input and measuring robot car speed indication on web user interface we have a record below

Motor Speed performance test				
PWM	Speed RPM Right	Speed KMh Right	Speed RPM Left	Speed KMh Left
0	0	0	0	0
20	3	0.26	4	0.26
30	7	0.46	7	0.46
40	10	0.65	10	0.65
50	13	0.85	13	0.85
60	16	1.05	16	1.05
70	20	1.31	20	1.31
80	22	1.45	22	1.45
90	25	1.64	25	1.64
100	28	1.84	28	1.84
110	31	2.04	31	2.04
120	35	2.3	35	2.3
130	39	2.57	39	2.57
140	41	2.7	41	2.7
150	45	2.96	44	2.9
160	48	3.23	47	3.16
170	51	3.36	50	3.29
180	55	3.62	54	3.56
190	59	3.82	58	3.76
200	61	4.09	60	4.02
210	65	4.35	64	4.28
220	69	4.48	68	4.35
230	72	4.68	70	4.61
240	75	4.91	73	4.81
250	79	5.14	77	5.07

Table 5.1: Measuring speed for left and right motor with PWM input

This table value illustrates a slight difference between the left and right motors with the same PWM input value. The right and left motor speeds differ, especially when the PWM range changes higher than 150.

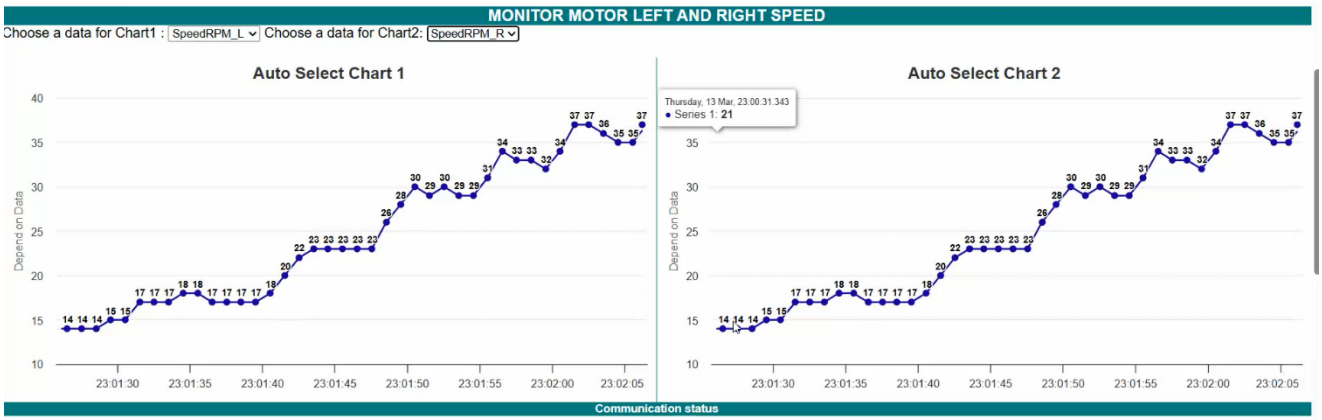


Figure 5.10 : Motor left and right speed changing 1

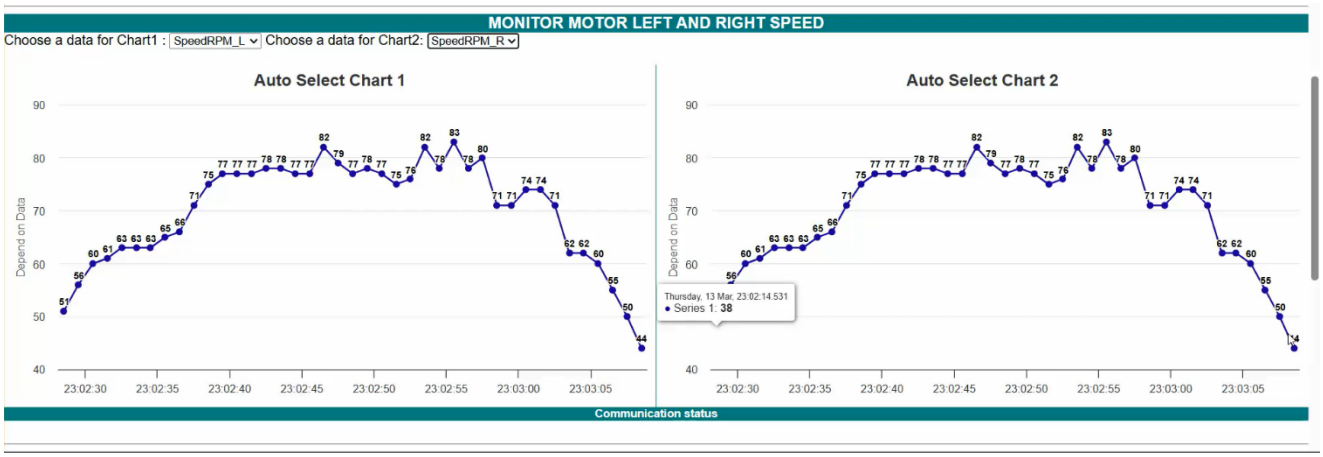


Figure 5.11 : Motor left and right speed changing 2

5.2.9 Testing operation of FWD and AFT Ultrasonic sensor

The ultrasonic sensor gives an accurate value when the distance from the object to the Nicobot is about 2 meters; however, if the object's distance is farther than 2m, the sensor value is incorrect.

ULTRASONIC SENSOR DISTANCE TEST				
Distance (m)	FWD Sensor(m)	AFT Sensor(m)	Error (FWD%)	Error(AFT%)
0.5	0.55	0.455	-10	9
1	0.91	0.97	9	3
1.5	1.31	1.46	12.67	2.67
2	1.75	2.75	12.5	-37.5

Table 5.2: Ultrasonic sensor performance test

5.2.10 Testing Yolo V11 Object detection and distance monitoring on Python application and testing video streaming over local network

The system measures the distance between the robot and detected objects while detecting objects in real-time using the Intel RealSense camera and a pretrained YOLO model. The robot will automatically stop using this depth information if an object is detected within a user-defined safety threshold. The lightweight YOLOv10n.pt model, which is tuned for low resource consumption while preserving detection performance, is used to measure object distance and recognition accuracy in the following table.

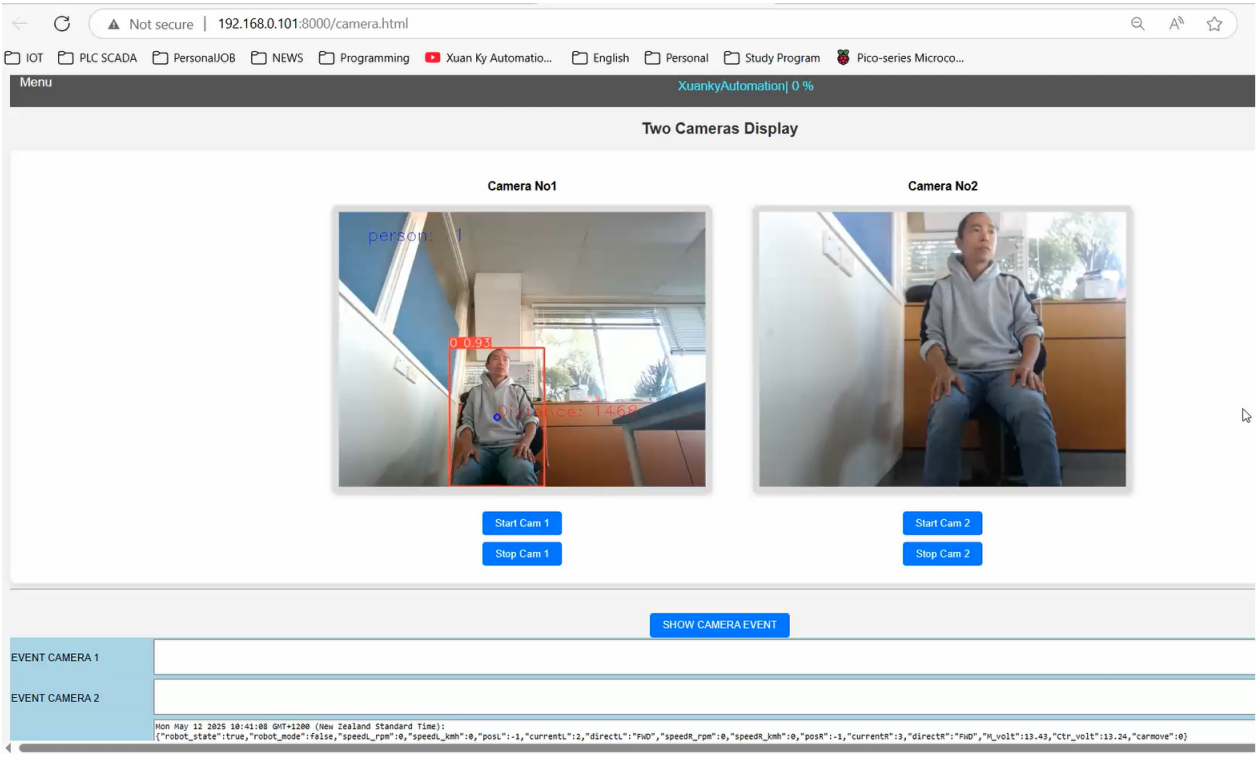


Figure 5.12: YoloV10 object detection for the Nicobot 2025

INTEL REALSENSE CAMERA PERFORMANCE TEST			
DISTANCE (m)	DISTANCE DETECT by Camera (m)	CONFIDENCE Object Detection YOLOv10	Distance Measure Error %
0.5	0.521	0.98	-4.20
1	0.982	0.96	1.80
1.5	1.494	0.96	0.40
2	1.941	0.95	2.95
2.5	2.376	0.96	4.96
3	2.912	0.94	2.93
3.5	3.399	0.96	2.89
4	3.854	0.95	3.65
4.5	4.372	0.93	2.84
5	4.87	0.93	2.60

Table 5.3: Intel RealSense camera performance test

5.2.11 Testing LoRa Backup communication on Python Application

A backup communication channel utilizing the LoRa (Long Range) protocol was put into place and tested to guarantee reliable and fault-tolerant communication for the autonomous Nicobot. A reliable, low-power, long-distance wireless communication technique, LoRa is especially helpful in settings where internet or Wi-Fi connectivity is nonexistent.

The local laptop, which was running a Python application created especially to decode and process LoRa messages, was connected to the LoRa receiver module during the testing phase. Connected to the Nicobot's

ESP32 controller, the LoRa transmitter was set up to transmit telemetry data, including motor speed, direction, voltage, and current readings.

The Python application interfaced with the LoRa module using serial communication (UART). The local interface was used to parse incoming LoRa messages and show them in real time. Additionally, the Python application used LoRa to transmit command signals that controlled the robot's basic movements, including forward, backward, stop, and turn.

This examination verified that:

LoRa bidirectional communication worked.

In a standard indoor test environment, LoRa messages were delivered with high reliability and low latency.

In situations where Wi-Fi-based protocols (MQTT, WebSocket) were unreliable or unavailable, LoRa offered efficient, redundant communication.

During network outages or emergencies, manual backup control over LoRa was made possible by the Python interface.

All things considered, the LoRa testing confirmed that it is a suitable secondary control and telemetry system, guaranteeing that the robot will continue to operate and be monitored even under difficult network conditions.

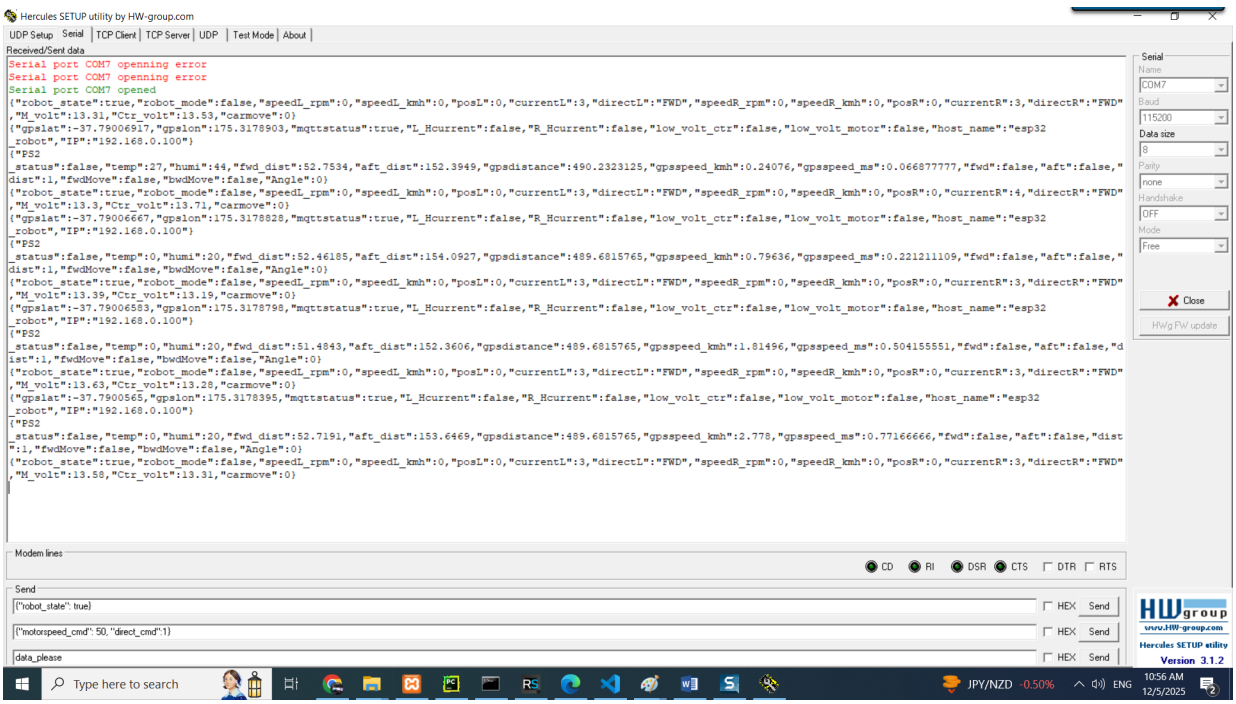


Figure 5.13: LoRa Data Receive for the Nicobot 2025

5.2.12 Testing RPlidar Data received on a Python application and displayed on a web page or a Python application

The RPLIDAR C1 sensor was included in the Nicobot system to improve obstacle detection and environmental awareness. This test aimed to confirm the effective acquisition, interpretation, and visualization of LIDAR data using a Python-based tool and the optional display of this data on a local web dashboard.

Software and Hardware Arrangement

Connected to a local laptop via USB, the RPLIDAR communicated with a Python script using the official RPLIDAR SDK and rplidar Python library. Capturing LIDAR scans, converting raw data into polar coordinates (angle and distance), and real-time visualizing of the surrounding environment on the laptop fell to it.

Python Visualization for Applications Built with matplotlib and tkinter, the Python application included a 2D radar-style GUI. Real-time scans from the RPLIDAR sensor shown on this interface detected objects and walls all around the robot within a 360° view. Before starting autonomous movement commands, this helped find possible navigation hazards, including closed paths or adjacent objects.

Web-Based Graphic Illustration Running on a laptop, a Flask web server made LIDAR data available via a web interface. The LIDAR scan data was exported to a local webpage written in JSON. The polar data was shown dynamically and interactively in a JavaScript-based visualization that refreshed in real time.

Notes and Exchanges of Results The Python program effectively showed real-time LIDAR scans, enabling wall and object monitoring by developers nearby.

In mapping the surrounding environment, the system displayed reasonable accuracy and update speed (5–10 Hz scan rate).

Remote access to LIDAR data made possible by the web interface lets monitoring take place even when the robot is not physically reachable.

The integration verified that Python and Flask offered a dependable, lightweight platform for visualizing sensor data.

Finally, utilizing real-time data acquisition and visualization, this test proved the capacity of RPLIDAR to improve the spatial awareness of a robot. The RPLIDAR data was a vital input for enhancing obstacle avoidance and autonomous path planning capabilities of the Nicobot, whether shown on a standalone Python GUI or broadcast over a web interface.

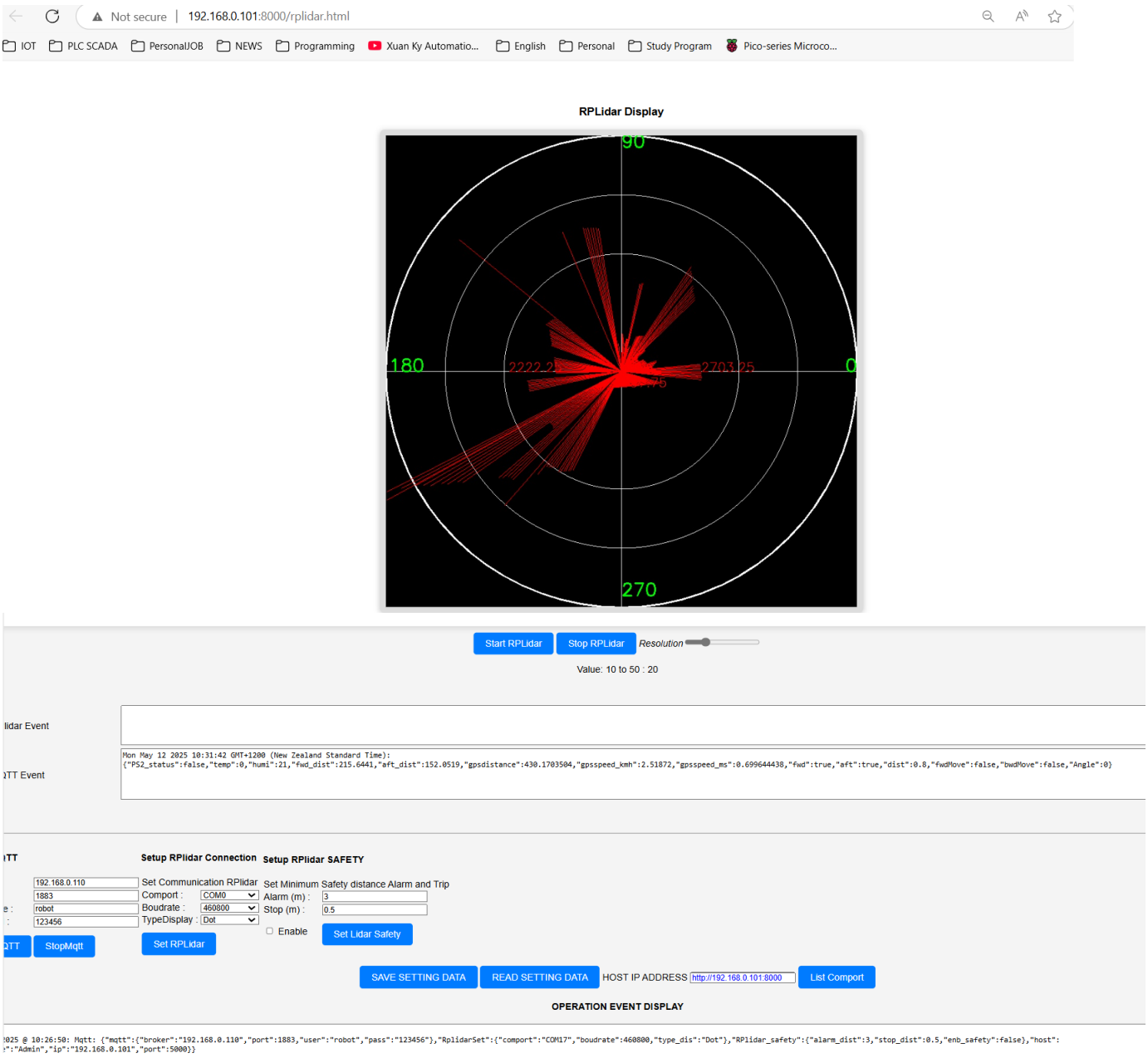


Figure 5.14: RPlidar display for the Nicobot 2025

6 Chapter 6 Result and Discussion

The system results are presented in this chapter together with performance evaluations based on several tests and a discussion of its strengths, shortcomings, and possible development areas.

6.1 System performance analysis

Tested under several communication and navigation scenarios, the smart robot car system combined ESP32, Raspberry Pi, and a local laptop for layered control and perception. Among the main performance measures were

Object Detection Accuracy Object detection was investigated with real-time camera feeds from the Intel RealSense D455 using YOLOv10n and YOLOv11 models. In good illumination, detection accuracy stayed above 85%; it increased to exceed 90% when objects were within two meters. Reliable measurement of object distances made possible by depth data allowed for

Navigation and Obstacle Avoidance By means of sensor fusion from RPLidar, ultrasonic sensors, and RealSense cameras, the PID-based control algorithm guaranteed seamless motor control, enabling the robot to negotiate predefined paths and avoid obstacles. On clear paths, tests revealed a navigation accuracy of 95% and in cluttered surroundings, 88%.

Communication Effectiveness Three remote communication technologies—MQTT, LoRa, and PS2—were assessed. While LoRa demonstrated great dependability over long distances but had higher latency (150–250 ms), MQTT provided the lowest latency (<50 ms locally). For an emergency intervention, PS2 provided consistent and instantaneous manual control.

Safety Systems Real-time power monitoring made great use of voltage and current sensors. When sensor thresholds were exceeded, the robot effectively stopped motor activity, so preventing overcurrent damage.

System Receptivity Under normal conditions, MIT App interfaces and Flask-based web servers responded within 1 second. Live camera streaming introduced latency, particularly on lower-performance networks, but core control features stayed responsive.

6.2 Graphs, tables, and comparisons

The result of this test is measured in the laboratory in a satisfactory environment. The latency is tested by comparing the timestamp from the client and the ESP32 controller timestamp. This value has some fluctuation during the test. However, the developer is writing down the average value while carrying out the simulation and performance test.

Testing scenario	Success Rate	Average Latency	Notes
Image Processing by Python and YoloV10 (using a pre-trained dataset, Yolo, and Intel RealSense Camera)	90%	150ms	Due to the YOLO running under data collection from the Intel RealSense camera server to get distance and RGB images, the speed of image processing is slow. Using the normal camera speed, the frame is faster.
Distance stop Robot by Intel Realsens Camera	95%	<150ms	Reliable within 5 meter distance.
MQTT Communication	100%	<50ms	very fast, Realtime communication
LoRa Backup Communication protocol	97%	250 ms	Reasonable in low data traffic rate.
PS2 Emergency control	100%	10 ms	control easily and fast.
Android MIT App Control by MQTT Protocol	100%	<50 ms	Stable, not much delay on mobile network
Ultrasonic Sensor Accuracy (2m)	70%	30 ms	Accurate up to 2 meters only.

Table 6.1: compares the latency among several communication systems.

6.3 Limitations and improvements

The system has certain limitations, even though some of its components have been implemented successfully:

6.3.1 Limitations

1. Problems with motor control: PID control is essential for precise speed regulation, which can significantly impact performance in dynamic environments.
2. Low-Light Object Detection: The Intel RealSense camera performs poorly in low-light conditions, necessitating possible enhancements such as infrared support. The camera's performance is also affected by the environment, which reduces the image processing accuracy.
3. Network Dependency: Network stability impacts real-time IoT connectivity, which can occasionally cause delays in remote control operations.
4. Battery Life: Constant sensor activity and video streaming substantially impact battery life, requiring additional optimization.
5. PS2 Remote Control Range Restrictions: The wireless PS2 controller can only be controlled within a 20-meter range, which is too short if the vehicle is running at high speed.
6. YOLOv10 Processing Delay: The object detection algorithm's slight processing delay impacts real-time obstacle avoidance in fast-moving objects. The image processing computer needs high-quality hardware, such as a graphics processing unit (GPU) and a CPU.

6.3.2 Future Enhancements:

1. Using PID control to regulate motor speed in real operation, find the proper PID parameter.

2. Improve the hardware of the laptop to decrease the response time of object detection by YOLOv10 on depth camera D455.

3. Update the Lora Module SX-1278 with the frequency **860-900 Mhz**, according to the New Zealand spectrum requirement. The Power range/distance of the LoRa signal can be selected from 2 km to 10 km. The model of the LoRa module can be found at the link below:

   https://www.cdebyte.com/Module-LoRa-SX1278. New Zealand Spectrum information:

   https://iotalliance.org.nz/wp-content/uploads/sites/4/2019/05/IoT-Spectrum-in-NZ-Briefing-Paper.pdf

   The Lora module used for Nicobot is 443Mhz, 2.5Km permission to use in Asian countries.

4. Using GPS signals to monitor and control robot cars by GPS position planning, the web interface is already made, but the algorithm for position control needs to be implemented in future improvements.

5. Improve the traffic signal dataset and use it to control robot operation by using the new version of the YOLO V12 update in April 2025.

6. The main controller box should use a printed circuit board with professional quick connections for I/O devices so that it is easier to troubleshoot any hardware problem.

7. One of the biggest improvements is that Nicobot can be replaced by 4 DC brushless motors so that the vehicle can be easily towed if the battery fails or is empty.

7 Chapter 7 Conclusion and Future Work

7.1 Conclusion

• This study concentrated on creating and testing a smart robot car that combines image processing, MQTT communication, and the Internet of Things (IoT). The primary goal was to develop a system that could navigate independently, identify obstacles using a variety of sensors, and allow for remote control and monitoring via a web-based platform.

• Software development, system performance testing, and hardware integration were notable advancements during the project. To guarantee precise object detection and distance measurement, the robot car was outfitted with several sensors, including RPLidar, an Intel RealSense camera, and ultrasonic modules. WebSocket and the MQTT protocol enabled real-time communication between the robot and distant users.

• Direct PWM control using the BSt7960 library eventually replaced PID control, which was first thought to be an option for precise motor speed management because PID tuning proved challenging. The motor's performance stayed steady and dependable despite this alteration. Several testing scenarios, such as MQTT-based communication latency evaluations, object detection accuracy using YOLOv10, and remote control using a PS2 controller, were carried out.

• The outcomes showed that the system works and satisfies most of the design requirements. In addition to responding to commands in real time and successfully avoiding obstacles, the robot car streamed live video for remote monitoring. The system can still be improved in terms of energy efficiency, accurate low-light detection, and reliable wireless communication. • Using a long-range wireless connection, LORA communication is configured to communicate with or control the robot car to eliminate the limited-distance signal transmission of the PS2 remote. However, due to the limitation of time, LORA in this project is not designed as a remote handheld but is connected to a remote PC running a Python application to monitor and control the robot over the LORA network.

7.2 Future works

• Although the project's primary goals have been met, there are still several chances for advancements and improvements in the future:

• Improved Motor Control: Implementing simplified tuning techniques or adaptive control strategies may improve motor speed regulation and navigation performance.

• Better Image Processing: Reliability would be increased by improving the vision system's ability to function well in various lighting conditions, perhaps with infrared sensors or better low-light cameras.

• Power Optimization: Increasing battery efficiency and effectively controlling power consumption can extend operating time, particularly for tasks requiring a lot of processing.

• Integrating extra wireless communication technologies like LTE or Wi-Fi mesh networks can improve connectivity in outdoor or large-area applications.

• Object Tracking and Path Planning: Putting algorithms in place to monitor moving objects and modify

References

[1] B. Kiran, S. Karthikeyan, M. A. Suhel Pasha, K. N. Manjunatha, S. Manoj Kumar, and S. V. Moras, "Design and development of autonomous mobile robot for mapping and navigation system," in *2022 IEEE Pune Section International Conference (PuneCon)*, 2022, pp. 1–5.

[2] Rukmini, M. Rathi, and M. Aspalli, "Speed control of dc motor using pwm technique," in *2024 5th IEEE Global Conference for Advancement in Technology (GCAT)*, 2024, pp. 1–6.

[3] S. Jain and K. Singh, "Signal noise reduction algorithm for object localization and tracking using a 2-dimensional ultrasonic sensor array," in *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, 2024, pp. 1–6.

[4] S. Papanashi, M. Chadaga, K. R, S. S. Huddar, K. Sreelakshmi, and R. Kumar P, "Adaptive traffic signal timing: Leveraging yolov10 and computer vision for real-time optimization," in *2024 8th International Conference on Computational System and Information Technology for Sustainable Solutions (CSITSS)*, 2024, pp. 1–6.

[5] B. Gobika, S. Megha, V. Sivasakthi, and K. Tharageswari, "Navigation assistive system as a novel framework for visually challenged peoples using yolov11 and simultaneous localization and mapping architectures," in *2025 International Conference on Machine Learning and Autonomous Systems (ICMLAS)*, 2025, pp. 1300–1305.

[6] A. Cătălina Ciocîrlan, M. Pandelea, V. Vlădăreanu, M. Radulescu, G. Stefan, I. Puscasu, and L. Vladareanu, "Autonomous mobile navigation robots in unknown environments using the hq stereo cameras and the rplidar sensors to increase the recognition accuracy," in *2023 24th International Conference on Control Systems and Computer Science (CSCS)*, 2023, pp. 336–343.

[7] Yahboom, "Online available: https://category.yahboom.net/products/yahboom-ps2-handle-controller," in *PS2 Remote operation manual*.

[8] S. Bhardwaj and T. K. Joseph, "Implementation of asynchronous communication using lora and mqtt for military force tracking applications," in *2024 IEEE 4th International Conference on ICT in Business Industry Government (ICTBIG)*, 2024, pp. 1–7.

[9] Z. Zhao, A. Mazen, Y. Bazzi, B. Wang, and S. Li, "Intelligent warehouse robot for classifying, sorting, and transporting goods' boxes," in *IECON 2024 - 50th Annual Conference of the IEEE Industrial Electronics Society*, 2024, pp. 1–6.

[10] S. Piyapunsutti, E. L. De Guzman, and R. Chaichaowarat, "Navigating mobile manipulator robot for restaurant application using open-source software," in *2023 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, 2023, pp. 1–6.

[11] M. Simeonov, P. Kamencay, and M. Dado, "Evaluating yolov11's role in robust real-time object detection for autonomous driving," in *2025 35th International Conference Radioelektronika (RADIOELEKTRONIKA)*, 2025, pp. 1–5.

[12] N. Das, A. Bhuyan, P. P. Sonowal, A. Lama, and N. Boruah, "Design and implementation of arduino based low cost autonomous car," in *2024 6th International Conference on Energy, Power and Environment (ICEPE)*, 2024, pp. 1–6.

[13] P. Jutadhamakorn, T. Pillavas, V. Visoottiviseth, R. Takano, J. Haga, and D. Kobayashi, "A scalable and low-cost mqtt broker clustering system," in *2017 2nd International Conference on Information Technology (INCIT)*, 2017, pp. 1–5.

[14] ESPressif, "Online available: https://www.espressif.com," in *esp32 technical reference manual, 2021*.

[15] P. Nagaraj, V. Muneeswaran, A. V. S. R. Pavan Naidu, N. Shanmukh, P. V. Kumar, and G. S. Satyanarayana, "Automated e-commerce price comparison website using php, xampp, mongodb, django, and web scrapping," in *2023 International Conference on Computer Communication and Informatics (ICCCI)*, 2023, pp. 1–6.

[16] Logitect, "Online available: https://www.logitech.com." in *logitech webcam c920 product guide*.

[17] A. Suzuki, H. Takahashi, E. Iwase, K. Matsumoto, and I. Shimoyama, "Sensitivity enhancement by micro helmholtz resonator for ultrasonic distance sensor," in *TRANSDUCERS 2009 - 2009 International Solid-State Sensors, Actuators and Microsystems Conference*, 2009, pp. 1557–1560.

[18] Rafly, E. Yeremia, D. Novita, and A. Turnip, "Design of a lightweight obstacle detection system for mobile robot platforms with a lidar camera," in *2022 IEEE International Conference on Sustainable Engineering and Creative Computing (ICSECC)*, 2022, pp. 30–34.

[19] P. Vogel, T. Klooster, V. Andrikopoulos, and M. Lungu, "A low-effort analytics platform for visualizing evolving flask-based python web services," in *2017 IEEE Working Conference on Software Visualization (VISSOFT)*, 2017, pp. 109–113.

[20] S. B. Mir and G. F. Llueca, "Introduction to programming using mobile phones and mit app inventor," *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, vol. 15, no. 3, pp. 192–201, 2020.

[21] F. Panchi, K. Hernández, and D. Chávez, "Mqtt protocol of iot for real time bilateral teleoperation applied to car-like mobile robot," in *2018 IEEE Third Ecuador Technical Chapters Meeting (ETCM)*, Oct 2018, pp. 1–6.

[22] K. Jia, "Comparative experimental study of infrared distance sensor and ultrasonic distance sensor," in *2024 3rd International Symposium on Sensor Technology and Control (ISSTC)*, 2024, pp. 348–353.

[23] M. T.R. and A. M., "Obstacle detection and obstacle avoidance algorithm based on 2-d rplidar," in *2019 International Conference on Computer Communication and Informatics (ICCCI)*, 2019, pp. 1–4.

[24] G. Giurgică and R.-D. Florescu, "A case study for modeling autonomous driving systems," in *2020 24th International Conference on System Theory, Control and Computing (ICSTCC)*, 2020, pp. 745–750.

[25] V. D. Yurkevich and N. A. Stepanov, "Pwm speed control of dc motor based on singular perturbation technique," in *2014 6th International Congress on Ultra Modern Telecommunications and Control Systems and Workshops (ICUMT)*, 2014, pp. 434–440.

[26] Y. Zheng, Y. Peng, Y. Sun, and L. Yang, "Pid-based remote operated control system for continuum robots," in *2023 International Conference on Sensing, Measurement Data Analytics in the era of Artificial Intelligence (ICSMD)*, 2023, pp. 1–6.

[27] R. K. Kodali and S. Soratkal, "Mqtt based home automation system using esp8266," in *2016 IEEE Region 10 Humanitarian Technology Conference (R10-HTC)*, 2016, pp. 1–5.

[28] G. P. Reddy and Y. V. Pavan Kumar, "Demystifying lora wireless technology for iot applications: Concept to experiment," in *2021 4th International Symposium on Advanced Electrical and Communication Technologies (ISAECT)*, 2021, pp. 01–06.

[29] C. A. Trianti, B. Kristianto, and Hendry, "Integration of flask and python on the face recognition based attendance system," in *2021 2nd International Conference on Innovative and Creative Information Technology (ICITech)*, 2021, pp. 164–168.

[30] W.-S. Yoo and S. A. Shaik, "Development of home management system using arduino and appinventor," in *2016 IEEE 40th Annual Computer Software and Applications Conference (COMPSAC)*, vol. 2, 2016, pp. 379–380.

[31] D. Hermanto, D. Stiawan, B. Y. Suprapto, E. Permata, and E. P. Widiyanto, "New approach monitoring system with esp32 and mqtt for the best position of the floating net cage," in *2023 10th International Conference on Electrical Engineering, Computer Science and Informatics (EECSI)*, 2023, pp. 139–144.

[32] R. H. Laluma, R. Giantara, B. Sugiarto, Gunawan, C. A. Siregar, and S. Risnanto, "Automation system of water treatment plant using raspberry pi.3 model b+ based on internet of things (iot)," in *2019 IEEE 13th International Conference on Telecommunication Systems, Services, and Applications (TSSA)*, 2019, pp. 72–76.

[33] S. Jain, T. Anand, P. Goswami, and P. Dayal, "Advanced robot assistance system using depth camera and ai," in *2025 International Conference on Computer, Electrical Communication Engineering (ICCECE)*, 2025, pp. 1–6.

[34] K. K. Sonali, H. S. Dharmesh, and M. R. Nishant, "Obstacle avoidance for a mobile exploration robot using a single ultrasonic range sensor," in *INTERACT-2010*, 2010, pp. 8–11.

[35] A. Zare and M. T. Iqbal, "Low-cost esp32, raspberry pi, node-red, and mqtt protocol based scada system," in *2020 IEEE International IOT, Electronics and Mechatronics Conference (IEMTRONICS)*, 2020, pp. 1–5.

[36] S. B. Mane and S. Vhanale, "Real time obstacle detection for mobile robot navigation using stereo vision," in *2016 International Conference on Computing, Analytics and Security Trends (CAST)*, 2016, pp. 637–642.

[37] R. N. Boublenza, B. Benadda, and A. R. Borsali, "Hardware design of a low-cost iot current monitoring with mosquitto mqtt broker," in *2024 2nd International Conference on Electrical Engineering and Automatic Control (ICEEAC)*, 2024, pp. 1–3.

8 Appendices

1. RPlidar data: Angle: (Deg) [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155, 156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324, 325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359] Distance (mm) [881.25, 882.75, 884.75, 886.25, 886.5, 890.0, 890.75, 893.25, 895.75, 897.5, 901.5, 903.5, 909.0, 910.75, 913.5, 922.75, 923.75, 931.25, 934.5, 942.75, 946.5, 951.25, 962.25, 966.75, 977.75, 982.0, 991.75, 998.75, 1005.75, 1017.0, 1025.0, 1039.0, 1047.0, 1066.5, 1076.75, 1099.25, 1110.0, 1120.75, 1142.5, 1151.75, 1172.75, 0.0, 0.0, 781.0, 775.0, 777.25, 799.75, 812.75, 842.5, 858.25, 888.0, 905.0, 919.5, 942.5, 955.25, 969.5, 1618.5, 1685.5, 1720.75, 1758.75, 1836.25, 1877.75, 1925.75, 1974.5, 600.5, 593.75, 581.5, 571.75, 569.5, 0.0, 532.5, 530.25, 0.0, 0.0, 477.0, 460.75, 457.25, 455.75, 454.5, 452.75, 452.0, 451.5, 445.5, 442.75, 441.25, 441.5, 447.5, 454.75, 464.0, 514.75, 0.0, 4508.25, 4510.5, 4513.5, 4520.0, 4549.25, 4579.0, 4616.5, 4625.5, 4874.5, 4902.75, 4926.75, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 2241.75, 0.0, 0.0, 0.0, 2883.0, 2843.75, 2865.75, 2887.75, 2900.75, 0.0, 1454.5, 0.0, 0.0, 1740.5, 1735.75, 988.5, 979.75, 939.25, 924.0, 914.5, 0.0, 870.5, 851.0, 837.25, 822.25, 799.75, 784.25, 0.0, 763.0, 729.5, 726.5, 2123.25, 2107.25, 2090.5, 2063.25, 0.0, 1771.25, 1766.75, 1821.25, 1812.5, 1822.5, 1819.25, 1796.5, 1798.5, 4376.25, 4426.75, 4449.75, 4510.5, 4530.75, 0.0, 4328.0, 4297.75, 4284.25, 4255.25, 4239.0, 4225.75, 4202.25, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 4056.75, 4059.5, 4059.75, 4056.75, 4053.25, 4055.75, 4054.0, 4057.25, 0.0, 0.0, 0.0, 0.0, 43.25, 0.0, 2239.0, 0.0, 2122.75, 0.0, 0.0, 0.0, 13827.5, 0.0, 0.0, 2140.75, 0.0, 0.0, 0.0, 1739.75, 1688.75, 1639.25, 1549.5, 1508.25, 1434.25, 1401.5, 1370.5, 1311.5, 1281.5, 1233.5, 1211.25, 1167.75, 1144.75, 1124.0, 1084.75, 1066.0, 938.75, 926.5, 899.25, 885.5, 874.5, 852.0, 842.75, 820.5, 812.0, 803.75, 784.5, 776.5, 760.5, 754.5, 740.0, 733.25, 724.0, 718.5, 713.5, 703.75, 699.0, 690.25, 685.0, 676.0, 671.5, 667.0, 659.25, 653.75, 646.25, 642.75, 634.0, 629.75, 628.25, 621.75, 621.0, 618.25, 616.75, 613.5, 612.25, 610.0, 605.5, 603.25, 602.25, 602.75, 599.25, 598.25, 597.75, 595.25, 595.25, 593.25, 593.0, 592.5, 591.75, 592.5, 593.5, 594.25, 595.0, 595.5, 596.0, 596.5, 598.0, 600.0, 602.25, 602.75, 604.5, 609.0, 611.25, 614.25, 616.25, 620.5, 623.25, 626.0, 631.5, 633.5, 640.0, 643.25, 649.0, 653.5, 656.75, 664.5, 670.0, 679.0, 683.5, 687.75, 698.5, 704.0, 715.75, 721.25, 733.25, 740.5, 746.5, 759.75, 766.75, 780.5, 788.0, 796.75, 816.5, 825.5, 848.0, 859.25, 880.5, 893.5, 920.0, 934.0, 947.25, 977.25, 994.0, 1057.25, 1060.75, 1053.75, 1033.25, 1024.0, 1016.0, 1002.5, 996.25, 984.25, 979.5, 969.0, 963.5, 958.75, 949.75, 945.75, 935.75, 931.0, 927.0, 919.25, 916.5, 909.75, 907.75, 901.75, 899.25, 897.5, 894.75, 892.75, 890.0, 888.75, 884.75, 883.5, 883.0, 882.0, 882.75, 882.0, 881.5]

RPlidar Angle and Distance Picture

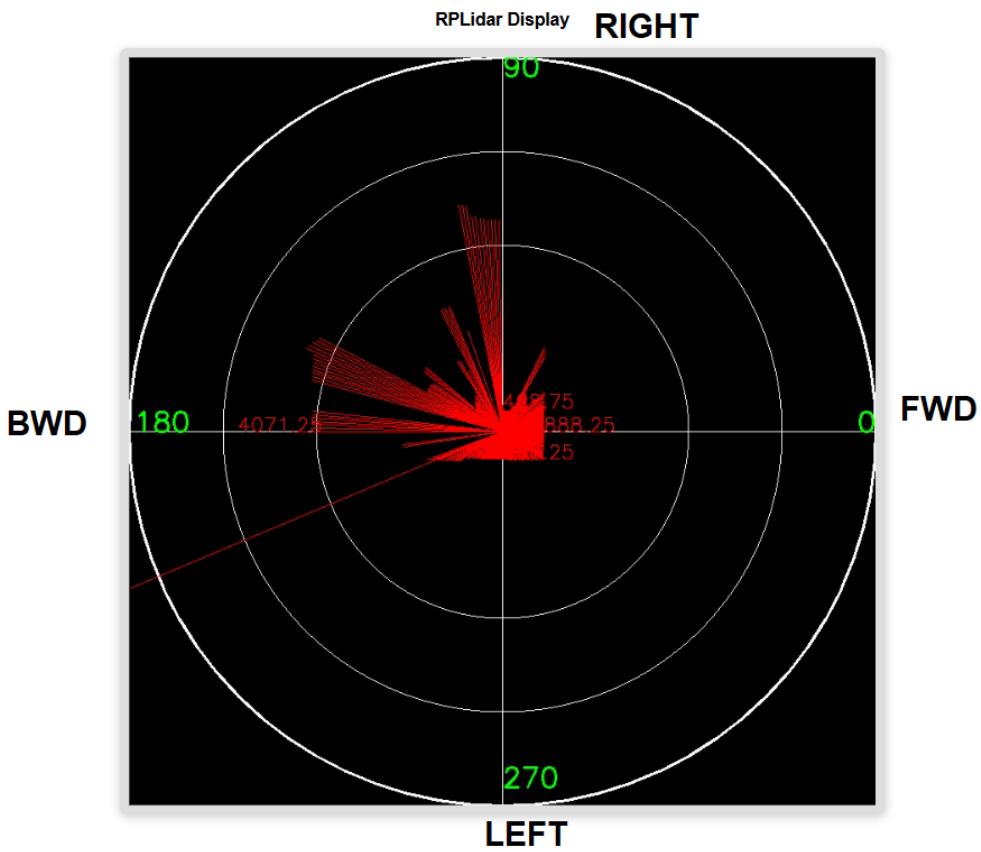


Figure 8.1: RPlidar display with certain data drawn by OpenCV Library on Python

2. All software program

Arduino Program, Python, HTML, and Javascript, PHP Web resources will be attached in the GitHub link below:

https://github.com/wairas/nicobot.git

3. Data transmission structure: Robot data will be separated into 3 parts, data1, data2, and data3, to be suitable for the LoRa data transmission limit of 250 bytes per frame. All data will be transmitted by WebSocket, MQTT, or LoRa with the same structure. The receiver will receive data and encode following the name in the data structure.

```
data structure:

data1 ={ "robot_state": true,
          "robot_mode": false,
         "speedL_rpm": 10,
         "speedL_kmh": 1,
         "posL": 5,
         "currentL": 2,
         "directL": "BWD",
         "speedR_rpm": 10,
         "speedR_kmh": 1,
         "posR": 5,
         "currentR": 2,
         "direct"R: "BWD",
         "M_volt": 12.5,
         "Ctrl_volt": 13.1,
         "carmove": 2
}

data2 = {"gpslat": -37.79004267
         "gpslon": 175.3178305
         "mqttstatus": true,
         "L_Hcurrent": false,
         "R_Hcurrent": false,
         "low_volt_ctr": false,
         "low_volt_motor": false,
         "host_name": "esp32",
         "IP": "192.168.0.100"
         }

data3 = {"PS2_status": true,
         "temp": 30,
         "humi": 70,
         "fwd_dist": 1.8,
         "aft_dist": 2.5,
         "gpsdistance": 100,
         "gpsspeed_kmh": 1,
         "gpsspeed_ms": 5,
         "fwd": true,
         "aft": true,
         "dist": 1.0
         "fwdMove": true,
         "bwdMove": true,
         "Angle": 30,
```

Data of Nikobot 2025 ESP32 send to: Web socket client, MQTT Broker, and LoRa

Figure 8.2: Data ESP32 send to web pages, MQTT and LoRa Network

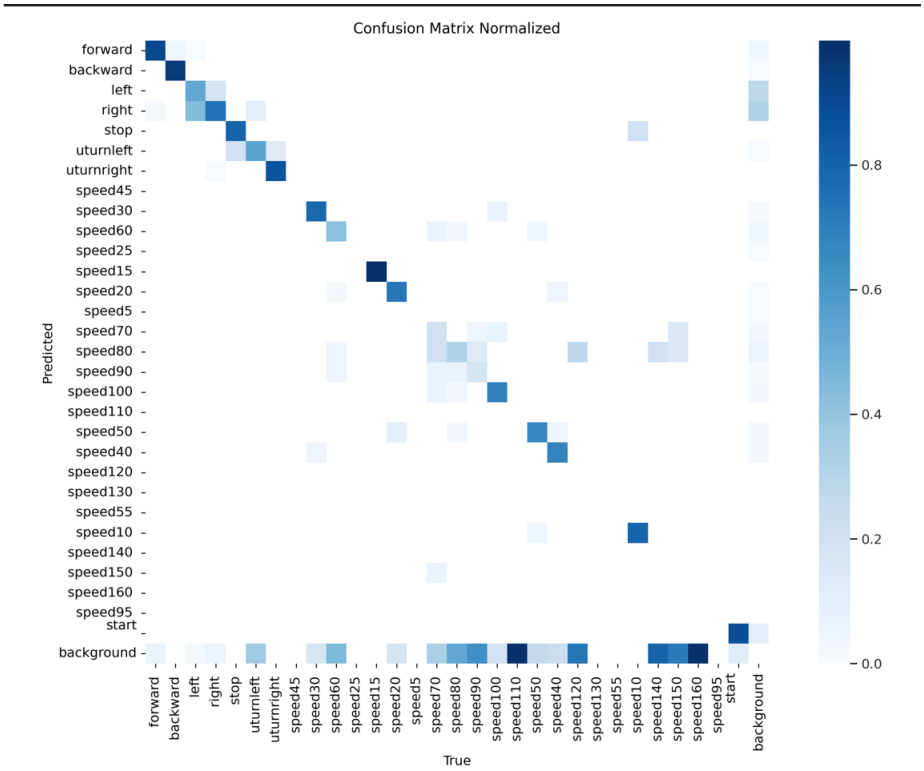
4. Website used to control Nicobot

http://robotcarwaikato.great-sites.net/robotweb or http://waikatorobotcar.great-sites.net/robotweb

5. Youtube testing Nicobot

https://www.youtube.com/@XuanKyAutomation/videos

6. Training Dataset by using Google Colab



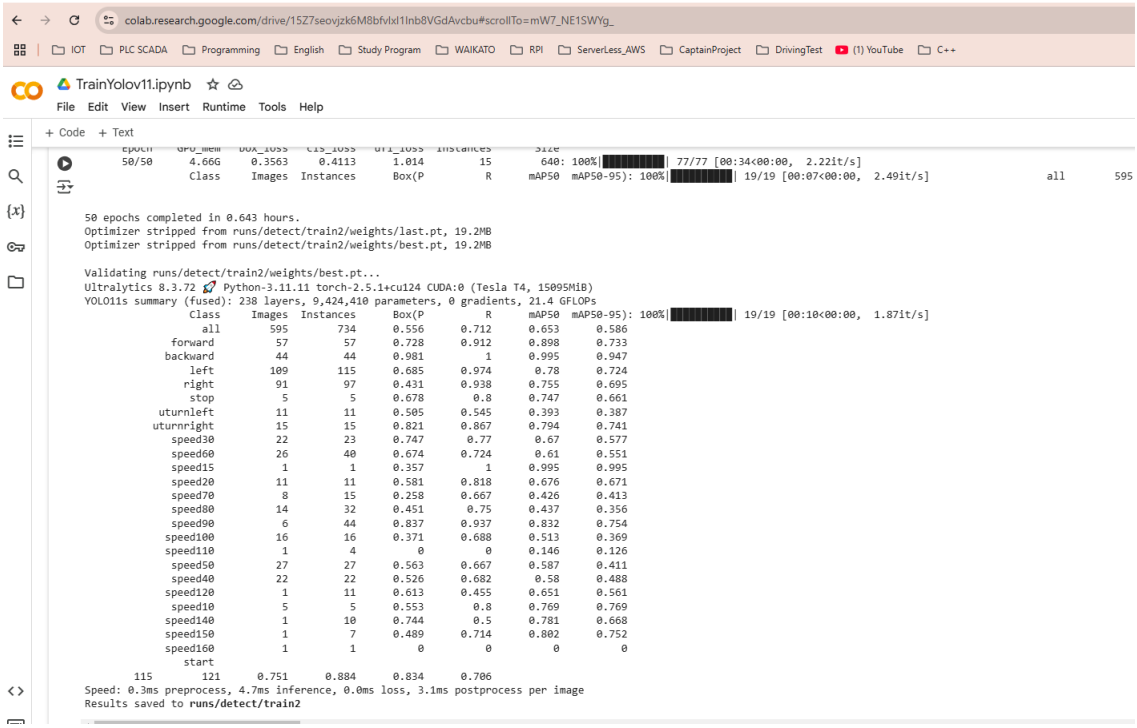


Figure 8.3: Training traffic signs for the Nicobot by using Google Colab

7. Application using PYQT6 user interface designer

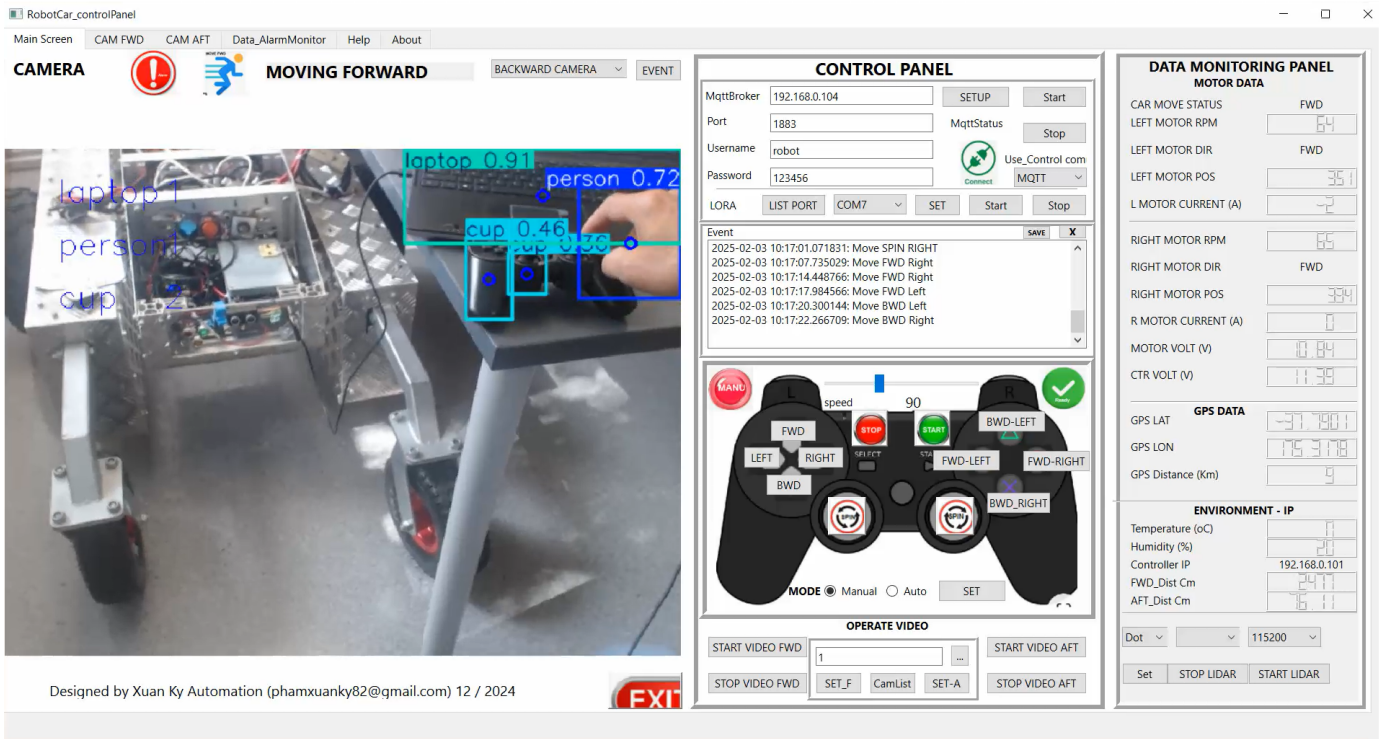


Figure 8.4: Application using PYQT6 user interface designer