PALESTINE POLYTECHNIC UNIVERSITY

College of Information Technology and Computer Engineering

Department of Computer Engineering

Cruise Master
"Autonomous Driving Car"

Project Team:

Momin Arafa 201016

Qasam Himoony 201119

Tamim Salhab 201120

Supervisor:

Dr. Elayan Abu Gharbieh

This project is submitted to fulfill the requirements for a Bachelor's degree in Computer Systems Engineering

Jan, 2025

Certification and Anti-Plagiarism Declaration

This is to certify that the graduation project titled 'Cruise Master' was prepared by the student(s) listed below under the supervision of Dr. Elayan Abu Gharbieh in partial fulfillment of the requirements for the Bachelor's degree in Engineering in Computer Systems Engineering.

We affirm that no part of this work has been reproduced illegally or constitutes plagiarism, including but not limited to direct copying or unacknowledged use of others' work. All referenced materials have been properly cited and used solely to support and substantiate the project's ideas.

By signing this declaration, we confirm our commitment to upholding academic integrity and certify that we have not and will not, engage in plagiarism, cheating, or any other violations of academic standards. We acknowledge that we bear full responsibility and accept any consequences should this declaration be found to have been violated. Date:

Graduation Project Group:

Momen Arafeh	Signature:
Qasam Himoony	Signature:
Tamim Salhab	Signature:

Abstract

The alarming rise in road accidents requires innovative solutions to prioritize driver and pedestrian safety. Cruise Master, an intelligent car system that uses object detection based on deep learning, represents a significant advancement in autonomous driving technology. This project aims to revolutionize travel by enabling safe and reliable journeys from one location to another.

Cruise Master is built on a compact car platform equipped with essential components for autonomous navigation. It features a DC motor to control movement, a servo motor for precise steering adjustments, and a camera to capture real-time environmental data. Additionally, an ultrasonic sensor enhances object detection by measuring distances to nearby obstacles. These components work together, enabling Cruise Master to analyze its surroundings, detect objects, and make informed driving decisions with accuracy.

When encountering obstacles, Cruise Master puts safety first by stopping the vehicle directly at a safe distance to avoid collision. This ensures a clear path before autonomously navigating to the destination. Cruise Master promotes a reliable and safe driving experience for everyone, contributing to a future with significantly reduced road accidents.

الارتفاع المقلق في حوادث الطرق يتطلب حلولًا مبتكرة تركز على سلامة السائقين والمشاة. نظام "كروز ماستر" ، وهو نظام ذكي للسيارات يعتمد على تقنية الكشف عن الأجسام باستخدام التعلم العميق، يمثل تقدمًا كبيرًا في تكنولوجيا القيادة الذاتية. يهدف هذا المشروع إلى إحداث ثورة في السفر من خلال تمكين رحلات آمنة وموثوقة من موقع إلى آخر.

تم بناء "كروز ماستر" على منصة سيارة مزودة بمكونات أساسية للملاحة الذاتية. فهو يحتوي على محرك تيار مستمر للتحكم في الحركة، ومحرك اخر لضبط التوجيه بدقة، بالإضافة إلى كاميرا لالتقاط البيانات البيئية في الوقت الفعلي. علاوة على ذلك، يعزز مستشعر الموجات فوق الصوتية عملية اكتشاف الأجسام من خلال قياس المسافات بالنسبة للعقبات القريبة. تعمل هذه المكونات معًا لتمكين "كروز ماستر" من تحليل بيئته المحيطة، واكتشاف الأجسام، واتخاذ قرارات القيادة بدقة عالية.

عند مواجهة المشاة أو المركبات القريبة، يعطي "كروز ماستر" الأولوية للسلامة من خلال إيقاف السيارة مباشرة على مسافة آمنة لتجنب الاصطدام. يضمن ذلك وجود مسار واضح قبل التنقل بشكل ذاتي إلى الوجهة. يسهم "كروز ماستر" في توفير تجربة قيادة آمنة وموثوقة للجميع، مما يساهم في مستقبل تقل فيه بشكل كبير حوادث الطرق.

Acknowledgment

In the name of "Allah", the most beneficent and merciful who gave us strength, and knowledge, and helped us to get through this project. To the people who have inspired and supported us into the people that we are today, our families, friends, and our supervisor. We would've never been able to reach this achievement without their support, care, and encouragement. We want to thank them all and we would like to express our gratitude to our graduation project supervisor Dr. Elayan Abugharbyeh for his guidance, support, and encouragement throughout the project.

Moreover, we owe an immense debt of gratitude to our families, whose unwavering encouragement and continuous support have been the cornerstone of our journey. Their generosity, both in spirit and action, has shaped the very fabric of who we are. Mom, Dad, and all our family members, your belief in us has been a guiding light, and for that, we are profoundly thankful.

At last, we acknowledge the collective effort that has propelled us forward. Each person who touched our lives, leaving an imprint of care and encouragement, has played a vital role in our story. As we celebrate this milestone, we do so with gratitude for the shared moments and the countless individuals who have left an indelible mark on our hearts.

Contents

1 Chapter 1: Introduction **1**

1.1 Overview . . . 1
1.2 Problem Statement . . . 1
1.3 Aims and objectives . . . 2
1.4 System Requirements . . . 2
1.4.1 Functional Requirements . . . 2
1.4.2 Non-Functional Requirements . . . 3
1.5 System Description . . . 3
1.6 Limitations and Constraints . . . 3
1.7 Schedule . . . 4
1.8 Report outline . . . 4

2 Chapter 2: Background **5**

2.1 Overview . . . 5
2.2 Theoretical background . . . 5
2.2.1 Computer Vision . . . 5
2.2.2 Convolutional Neural Networks (CNNs) . . . 5
2.2.3 Breadth First Search (BFS) . . . 6
2.2.4 Internet of Things (IoT) . . . 6
2.2.5 IoT Protocols . . . 6
2.3 Literature Review . . . 7
2.3.1 An Embedded System in Autonomous Vehicles for Pedestrian Detection using Deep Learning . . . 7
2.3.2 Autonomous RC-Car for Education Purpose in iSTEM Projects . . . 7
2.4 Summary . . . 8

3 Chapter 3: System Design **9**

3.1 Overview . . . 9
3.2 Hardware Components . . . 9
3.2.1 Controllers . . . 9
3.2.2 Camera . . . 10
3.2.3 Proximity detector . . . 11
3.2.4 Body and Structural Design . . . 11
3.2.5 Actuators . . . 13
3.2.6 Motor Driver . . . 14
3.3 Software Components . . . 14
3.3.1 Object detection and recognition using YOLO . . . 14
3.3.2 System Navigation . . . 15
3.4 Conceptual System Design . . . 17

3.5 Algorithms and methodologies 18
3.5.1 Sequence Diagram 21
3.6 Schematic Diagram 23
3.7 Summary 23

4 Chapter 4: Implementation and Testing 24
4.1 Overview 24
4.2 Hardware Implementation 24
4.2.1 Prototype Car Setup 24
4.2.2 Navigation System Setup 25
4.2.3 Road, road signs and obstacle recognition system 25
4.2.4 Motion System 26
4.2.4.1 Motors Setup 26
4.2.4.2 Controlling and interfacing circuits 27
4.3 Software implementation 27
4.3.1 Overall System 27
4.3.2 AI Models Integration 27
4.3.2.1 Steering Angle Prediction 27
4.3.2.2 Car Localization 28
4.3.2.3 Traffic Light Detection 29
4.3.2.4 Stop Sign Detection 31
4.3.3 Navigation System 31
4.3.3.1 Top-Down View Camera Integration 31
4.3.3.2 User Interaction for Target Selection 32
4.3.3.3 Path Planning 32
4.3.3.4 Message Communication via RabbitMQ 32
4.3.3.5 Car Movement and Feedback Loop 32
4.3.3.6 Arrival at Target Location 32
4.3.4 Driving System 33

4.4 Implementation Challenges and Issues . . . 33
4.4.1 Hardware Implementation Challenges . . . 33
4.4.2 Software Implementation Challenges . . . 33
4.4.2.1 AI Model Challenges and Solutions . . . 33
4.4.2.2 Decision-Making and Multi-Model Approach . . . 34
4.5 Summary . . . 34

5 Chapter 5: Testing and Results 35
5.1 Overview . . . 35
5.2 Hardware Components Testing . . . 35
5.2.1 Unit Testing . . . 35
5.2.1.1 Raspberry Pi 4 . . . 35
5.2.1.2 DC Motor . . . 35
5.2.1.3 Servo Motor . . . 35
5.2.1.4 Batteries . . . 35
5.2.1.5 Cameras . . . 35
5.2.1.6 Ultrasonic sensor . . . 36
5.3 Software Components Testing . . . 36
5.3.1 Unit Testing . . . 36
5.3.1.1 Raspberry Pi OS installation . . . 36
5.3.1.2 Steering Angle Prediction . . . 36
5.3.1.3 Traffic Light and Stop Sign Detection . . . 36
5.3.1.4 Car Location Detection . . . 38
5.3.1.5 Broker . . . 39
5.3.1.6 Response Time . . . 39
5.4 Integration Testing . . . 40

5.5 System Testing 40
5.5.1 Localization System Testing 40
5.5.2 Driving System Testing 41
5.6 Summary 41

6 Chapter 6: Conclusion **42**
6.1 Overview 42
6.2 Conclusion 42
6.3 Future Work 42

List of Figures

2.1 Convolutional Neural Network Architecture. 6
3.1 Logitech Webcam 10
3.2 Ultrasonic 11
3.3 Wheel car 12
3.4 chain car 12
3.5 YOLO algorithm 15
3.6 Map construction using OpenCV 16
3.7 System block diagram 17
3.8 System conceptual diagram 18
3.9 Navigation and Path planning Sequence diagram 21
3.10 Driving and Obstacle detection 22
3.11 Schematic Diagram 23
4.1 Prototype Components 24
4.2 Navigation setup 25
4.3 Recognition system setup 25
4.4 Motor Setup 26
4.5 Motor Driver interfacing 27
4.6 Location box loss, class loss, and object loss 28
4.7 Localization Model Examples 29
4.8 Traffic Light box loss, class loss, and object loss 30
4.9 Traffic light detection examples 30
4.10 Stop sign detection 31
4.11 Car Location Terminal 31
4.12 Calculated Path Terminal 32
4.13 Car Arrival at Target Destination Terminal 32
4.14 Environment Before and after 34
5.1 Trafiic Light Detection Example 37
5.2 Car Location Recognition Example 39

List of Tables

1.1 Project schedule in the first and the second semester 4
2.1 Comparison with other projects 8
3.1 Comparison of Raspberry Pi 4 and NVIDIA Jetson Nano 9
3.2 Comparison of Cameras 10
3.3 Comparison of Proximity Sensor 11
3.4 Types of Actuators used 13
3.5 Comparison of H-Bridge Motor Driver L298N and L298D 14

Abbreviations

Acronyms

AMQP Advanced Message Queuing Protocol. 6, 27, 32, 33

BFS Breadth First Search. 2, 6, 20, 32

CNN Convolutional Neural Network. 5, 8

CoAP Constrained Application Protocol. 6

GPU Graphics Processing Unit. 7, 9, 10

IoT Internet of Things. 6, 9

LoRaWAN Long Range Wide Area Network. 6

MQTT Message Queuing Telemetry Transport. 6

R-CNN Region-Based Convolutional Neural Networks. 14

SSD Single Shot MultiBox Detector. 15

YOLO You Only Look Once. iv, 7, 14–16, 20

1 Chapter 1: Introduction

1.1 Overview

According to the Palestinian Central Bureau of Statistics (PCBS), 16,508 road car accidents occurred in 2022 [1]. There are a lot of risks to being a car driver or passenger. And for other car drivers, pedestrians, and cyclists, it can be even more unpredictable and dangerous.

About 94% of accidents on the roads are caused by driver errors [2]. That could be caused by anything, turning the steering wheel too much, a pedestrian stepping out from behind a car, or suddenly trying to cross the road.

Modern cars are equipped with advanced safety features, such as object detection sensors. However, to truly prevent accidents, autonomous vehicles require more than just the ability to perceive their surroundings. They need the capability to analyze the situation, determine the most appropriate course of action, and execute that action independently, effectively replacing human decision-making in critical moments.

Leveraging advancements in computer vision and deep learning, autonomous vehicles are now capable of performing various tasks, including pedestrian detection. This project focuses on the development of a prototype autonomous car that utilizes these techniques to make informed decisions and execute appropriate actions while navigating a roadway environment.

1.2 Problem Statement

Although consumers often prioritize safety and comfort in new cars, the continued dependence on driver control remains a significant challenge. Human error remains a significant factor in car accidents. This translates to millions of injuries and fatalities every year, along with immense emotional and economic costs. Beyond safety concerns, traffic congestion plagues cities worldwide, leading to wasted time, increased fuel consumption, and air pollution.

This project addresses these issues by exploring the potential of autonomous vehicle technology. Our goal is to develop a system that leverages rapid response times and efficient decision-making to navigate complex situations. This includes the ability to safely stop the car in response to obstacles or changing road conditions, ultimately aiming to reduce accidents and improve overall traffic flow.

1.3 Aims and objectives

In this project, we propose a system that aims to provide the following features:

1. The Cruise Master system aims to develop a self-driving car capable of independently navigating from one point to another. To achieve this, the following objectives must be accomplished:

   (a) Develop a vision-based positioning system that accurately determines the car's location at the start, during, and end of each journey.

   (b) Train an AI model capable of identifying road boundaries to ensure the car remains on the correct path throughout the trip.

   (c) Implement a Breadth First Search (BFS) algorithm to calculate the shortest and most efficient route, considering the car's orientation, ensuring successful arrival at the destination.

2. The system must be capable of adapting to road changes by enabling the car to effectively respond to both expected and unexpected environmental conditions, ensuring safe and reliable movement. To achieve this, the following objectives must be met:

   (a) Identify and define the system's responses to various environmental scenarios, including obstacles and traffic variations, to ensure safe and reliable navigation.

   (b) Develop and train machine learning models using collected datasets to process real-time camera inputs, enabling accurate detection and prediction of obstacles and environmental changes.

   (c) Integrate object detection, classification, and interpretation capabilities into the system, enabling context-aware decision-making for adaptive and efficient navigation.

1.4 System Requirements

To specify the system requirements, the following functional and non-functional requirements can be considered:

1.4.1 Functional Requirements

- The system shall be able to capture, analyze and recognize road lanes, traffic lights, and stop signs using the camera.
- The system shall react to pedestrians, traffic lights, and stop signs on the street.

- The system shall track the road lanes and move between the lanes.
- The system shall move from one point to another using a navigation system.

1.4.2 Non-Functional Requirements

- Response time: The system must have a real-time response to events in the real world, and approximately less than 0.83 seconds or less as indicated in [3].
- Ease of use: The system should be user-friendly and intuitive for passengers. Passengers should be able to easily set destinations and understand the system's actions.
- Safety: The system must prioritize the well-being of passengers and pedestrians by employing robust perception, decision making, and fail-safe mechanisms.

1.5 System Description

Our system's primary objective is to create a comprehensive autonomous driving experience from one point to another and respond to road events using computer vision, through a three-stage process:
Stage1: The user inputs the target location and the system sets the starting location as its current location and finds a way to go from its location to the target location through the navigation system.
Stage2: The car starts moving between road lanes, and follows real-world traffic laws, if a traffic light is detected and it is red, the car stops and moves when it turns green. when an obstacle is detected on the road, the car stops. If it is a stop sign, the car will stop.
Stage3: when the car is now at the target address, it stops.

1.6 Limitations and Constraints

- Cost constraint: Our system model will simulate the real world. The proposed solution is fit for the real world. However, it will require more professional yet expensive components. So, due to the high cost of these components, we will design the prototype based on what's efficient to fit the designed model.
- Lighting Conditions: Lighting conditions in poor lights or at night can affect the system badly.

1.7 Schedule

The system implementation and operation tasks are distributed along the first and second semesters, summarised in Table 1.1.

Table 1.1: Project schedule in the first and the second semester

	The first semester			The second semester			
Week	1 - 4	5 - 10	11 - 15	1 - 4	5 - 10	11 - 14	15
Selection of project Idea							
Collecting the Data							
System Design							
System Implementation							
System testing							
system operation							
Documentation							

1.8 Report outline

This report is structured as follows: Chapter 1 provides an introduction, including an overview of the project, the problem statement, and the aims and objectives. It also outlines the system requirements, both functional and non-functional, and offers a detailed description of the system, along with its limitations and constraints. Chapter 2 presents the background of the project, covering the theoretical foundations of key technologies such as computer vision, CNNs, and IoT, followed by a literature review of similar systems. Chapter 3 describes the system design in detail, focusing on both hardware and software components, and includes a conceptual system design and relevant algorithms. Chapter 4 discusses the implementation and testing of the system, detailing the hardware and software setups, as well as integration challenges. Finally, Chapter 5 evaluates the testing procedures and results, while Chapter 6 concludes the report and suggests future work for further improvements.

2 Chapter 2: Background

2.1 Overview

This chapter introduces the theoretical foundation essential to our project. Following this, we'll dive into a literature review, comparing our project with what's been done before. This comparison helps us highlight the unique aspects and innovations our project brings. In essence, this chapter provides the background needed to understand our project's roots and place among previous field efforts.

2.2 Theoretical background

In this section, we'll explain the core components of our project. Computer vision, which involves processing inputs such as photos and videos. These inputs are fed into an algorithm that predicts the steering angle. Computer Vision and other software and hardware components are explained in our system.

2.2.1 Computer Vision

Computer vision is a field of Artificial Intelligence (AI) that deals with extracting information from digital images and videos [4]. By employing deep learning models, computer vision aims to understand and interpret the content of the visual data. Computer vision is widely used in applications requiring object or situation classification.

2.2.2 Convolutional Neural Networks (CNNs)

Convolutional Neural Network (CNN) are a powerful deep learning architecture that reigns supreme in image classification and object recognition tasks within computer vision. Unlike traditional neural networks, CNNs excel at processing grid-like data like images. Their secret lies in a specialized set of layers. Convolutional layers act like filters that scan the image, extracting significant features like edges, lines, and shapes. Pooling layers then down sample the data, keeping the most important information while reducing complexity. Finally, fully connected layers, similar to those in traditional neural networks, combine these extracted features to make a final prediction, such as classifying the image or recognizing objects within it [5], as shown in figure 2.1. Through this remarkable process of feature extraction, transformation, and classification, CNNs have become the backbone of many computer vision applications [6].

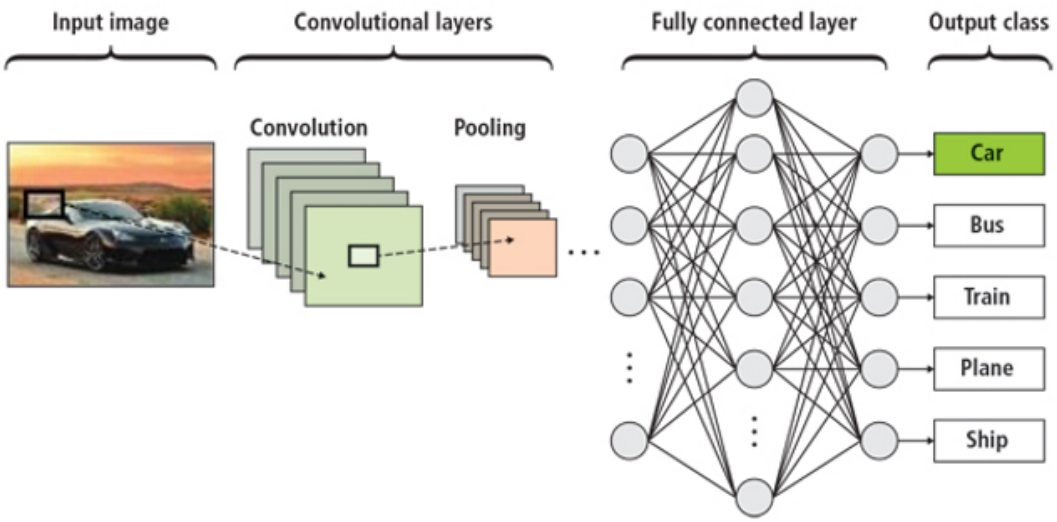


Figure 2.1: Convolutional Neural Network Architecture. [7]

2.2.3 Breadth First Search (BFS)

In the context of autonomous vehicles, BFS can be employed for path planning by systematically exploring all possible paths from the starting point to the destination. This approach ensures that the vehicle identifies the shortest path in an unweighted grid or graph, which is crucial for efficient navigation [8].

2.2.4 Internet of Things (IoT)

The Internet of Things refers to the physical items such as sensors, processing power, software, and other technologies that are linked to other systems and devices over the Internet or other communications networks and exchange data with them [9].

2.2.5 IoT Protocols

Within the Internet of Things (IoT), communication between devices and systems relies on established protocols. These protocols, acting as a set of rules and standards, define how data is transmitted, devices are discovered, connections are established, and security is ensured within the IoT network. Unlike traditional protocols, they are designed specifically for the resource-constrained nature of IoT devices, often with limited processing power, memory, and energy. The most important IoT protocols and standards include Message Queuing Telemetry Transport (MQTT), Constrained Application Protocol (CoAP), Advanced Message Queuing Protocol (AMQP), and Long Range Wide Area Network (LoRaWAN) [10].

2.3 Literature Review

Autonomous driving cars have been one of the important technologies to use and design. Many projects proposed different design suggestions to satisfy the requirements, the following are discussions of such works.

2.3.1 An Embedded System in Autonomous Vehicles for Pedestrian Detection using Deep Learning

This system is based on the installation of the system itself on autonomous driving cars. the system detects pedestrians on the road with the use of deep learning on a Nvidia jetson nano kit, The camera will be used to continuously capture images from the road that will be fed to enhance it before inserting it into the object detection algorithm You Only Look Once (YOLO) to detect a presence of a pedestrian if exist. If there are any pedestrians in the street in front of the car, the system will control the vehicle's velocity according to the distance from the pedestrian. Then the system distinguishes the pedestrian from the car, and this, in turn, is based on giving voice warnings to the driver and determining the speed of the car that must be driven or intervened to make a complete stop of the car. [11]

Unlike embedded pedestrian detection system for existing vehicles, our project focuses on a complete autonomous driving car prototype. Our system goes beyond pedestrian detection, aiming to recognize various objects on the road like traffic lights, signs, and other vehicles for a more comprehensive self-driving experience. Additionally, we incorporate a navigation system to plan and execute driving routes, making our prototype a more comprehensive self-driving solution. [11]

2.3.2 Autonomous RC-Car for Education Purpose in iSTEM Projects

RC-Car is a very simple car, based on a Raspberry Pi, a camera, and a servo shield board to interface with the RC-Car. Driving the vehicle around a lined track to capture images and vehicle headings (left, forward, and right), which trains a neural-network autopilot to drive itself around the track. While collecting training data, the car itself doesn't do all that much. It basically takes pictures sends them to a Graphics Processing Unit (GPU)-supported PC server and gets servo commands in return. The server is important for implementing autonomous driving functions. Firstly, it collects the images and driving information from the user manually driving the car around the track, by the default way delivered by the server. The server records data from a person driving the car, and then uses those images and heading variables to train a Keras/TensorFlow neural network model in software. This happens quickly if GPUs are used. Once trained, the model can be loaded on the car and the car should be able to drive like the manual driving style. [7]

This system stands apart from previous research by implementing all autonomous vehicle processes and functionalities on a real, physical car. While some studies may

explore similar functionalities like object detection and navigation, our project prioritizes real-world testing and development for a more practical and transferable approach to autonomous driving. [7]

Table 2.1: Comparison with other projects

	An Embedded System in Autonomous Vehicles for Pedestrian Detection Using Deep Learning	Autonomous RC-Car for Education Purposes in iSTEM Projects	Our Project
System type	Embedded System	Prototype for real system	Prototype for real system
Navigation system	There is no navigation system	There is no navigation system	We have our in-door navigation system
Object detection	Detects only pedestrians on the street	Road lane detection	Pedestrians, traffic lights and signs, and road lane detection
Action on detection	Change the velocity of the vehicle	Track the road and drive along it	Change the velocity, track, and drive on the road lane

2.4 Summary

Training AI models and using computer vision algorithms (i.e. CNNs) is a good solution in such systems.

3 Chapter 3: System Design

3.1 Overview

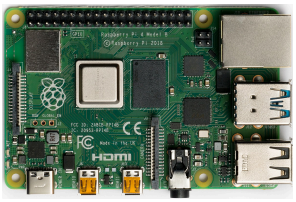
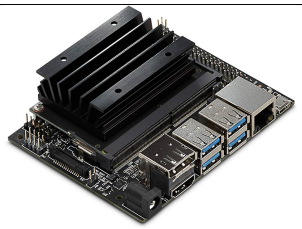
This chapter provides an examination of the key hardware and software elements necessary for our project. We explore various options for each component and offer a conceptual framework of the system's design. Additionally, we introduce a basic block diagram to illustrate the system's structure. Furthermore, we detail the system's algorithms and methodologies using straightforward flowcharts, while schematic diagrams depict the connections and interactions between components.

3.2 Hardware Components

3.2.1 Controllers

Microcomputers offer a wide range of options for different needs and projects, from basic DIY projects to advanced IoT solutions. Among these options, the Raspberry Pi 4 and the NVIDIA Jetson Nano stands out and are the most common. Table 3.1 provides a straightforward comparison between these options, detailing their features, performance, and applicability for various tasks.

Table 3.1: Comparison of Raspberry Pi 4 and NVIDIA Jetson Nano

Feature	**Raspberry Pi 4**	**NVIDIA Jetson Nano**
Image	 [12]	 [13]
Processor	Quad-core Cortex-A72	Quad-core Cortex-A57
GPU	Broadcom VideoCore VI	Nvidia Maxwell
RAM	2GB, 4GB, or 8GB LPDDR4	4GB LPDDR4
Storage	MicroSD, USB	MicroSD, USB, NVMe SSD
Camera Interface	CSI connector	MIPI CSI-2 connector
Number of Pins	40 GPIO	40 GPIO
I/O Ports	HDMI, USB, Ethernet, GPIO	HDMI, USB, Ethernet, GPIO
Power Supply	5V via USB-C	5V via barrel jack or micro-USB
Wireless Connectivity	Wi-Fi 802.11ac, Bluetooth 5.0	Wi-Fi 802.11ac, Bluetooth 4.2
Cost	120 Dollar	210 Dollar

The Raspberry Pi 4 stands out for its affordability, versatility, and extensive community support. With its budget-friendly price tag and broad range of GPIO pins, it's more accessible. Its user-friendly interface and vast community provide ample resources for our project. However, its compute power and GPU performance are limited compared to dedicated AI boards like the NVIDIA Jetson Nano. On the other hand, the Jetson Nano excels in AI and machine learning applications, boasting powerful GPU capabilities and hardware acceleration for tasks like TensorFlow and PyTorch. While it offers superior performance, its higher cost is overwhelming. Additionally, it's worth nothing that the Jetson Nano may encounter challenges with achieving high frame rates in certain applications, which could impact the suitability for real-time video processing in our project. Ultimately, for our project as it is prioritizing affordability and community support, the Raspberry Pi 4 is the preferred choice.

3.2.2 Camera

In our self-driving car project using computer vision, the camera is an essential part. It captures real-time visual data, allowing the system to recognize objects like stop signs, traffic lights, and road lanes. This information is crucial for safe navigation and adherence to traffic rules, making the camera the eyes of the autonomous vehicle.

Table 3.2: Comparison of Cameras

Feature	**Logitech Webcam**	**Raspberry Pi Camera Module**	**ESP32-CAM**
Resolution	1080p	8 MP	VGA
Interface	USB	CSI	Serial
Lens Type	Fixed focus	Fixed focus	Adjustable focus
Field of View	78°	62.2°	65°
Dimensions	Compact	Small	Small
Price	$30-$100	$25-$30	$5-$10

We have selected the Logitech webcam as shown in Figure 3.1 for our project due to its ease of use and compatibility. With its USB interface, high resolution, and compact design, it provides a simple and effective solution for our needs as shown in Table 3.3.

Figure 3.1: Logitech webcam [14]

3.2.3 Proximity detector

The ultrasonic sensor is a distance sensor as shown in Figure 3.2, so we will use it when an object is detected in front of the car to read the distance, and if the distance drops below a certain level, it is assumed that the object is close to the car and the car should be stopped if the object is stationary or slowed down if the object is moving in the same direction as the car.

Table 3.3: Comparison of Proximity Sensor

Feature	**HC-SR04**	**TOF10120**
Coverage	Exposed to environment	Sealed to environment
Size	Large Size	Very small
Response	Slow response	Quick response
Field of view	Wide	Narrow
Waves	Ultrasonic	Laser
Cost	Low cost (up to 3 Dollars)	High cost (up to 10 dollars)

Figure 3.2: Ultrasonic sensor (HC-SR04) [15]

3.2.4 Body and Structural Design

Robot wheel car (as shown in Figure 3.3) and robot tank car chassis bodies (as shown in Figure 3.4) offer distinct advantages for car robots, each suited to different environments. Wheeled designs prioritize high rotation speed, smooth linear motion, and agility, making them ideal for flat, even surfaces where speed and precision are essential. In contrast, chained robot bodies excel in flexibility and stability, enabling superior performance on rugged or uneven terrains by providing strong traction and adaptability. So a wheeled robot is better for our project because it drives on roads and we need rotation speed and precise rotation control.

Figure 3.3: Wheel car robot body [16]

Figure 3.4: Diy tracked robot body [17]

3.2.5 Actuators

We will use the following 2-types of motors in the car, see Table 3.4.

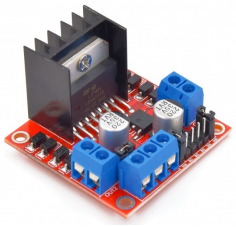
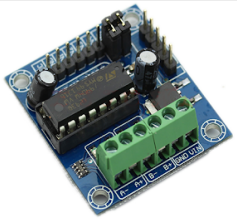
Table 3.4: Types of Actuators used

CORONA DS558HV	**12-volt DC-Motor**
To steer the car to the right and left, we chose the CORONA DS558HV Servo Motor, as it provides high torque, excellent precision, and control. This motor is ideal for the project, offering smooth movement and precise angle adjustments. The CORONA DS558HV operates within a voltage range of 6.0V to 8.4V and has an operating current that can vary from 0.1A to 3A depending on the load. At maximum load, the power consumption can reach up to 25.2W. Its high performance in this range makes it perfect for achieving the precise steering movements required in this application.	To enable precise and efficient movement of the self-driving car, we selected a high-performance 12-volt DC motor. This motor operates at 12V and typically draws a current ranging from 1A to 5A depending on the load, with power consumption between 12W to 60W. It provides the necessary power for smooth and controlled driving, and its durability and efficiency make it an ideal choice for autonomous vehicle applications that demand advanced motion control and high reliability.
 [18]	 [19]

3.2.6 Motor Driver

The following H-bridge motors allow bi-directional control of DC motors, allowing forward and reverse movement by changing the voltage polarity. These motors ensure precise and smooth movement of the intelligent interactive robot.

Table 3.5: Comparison of H-Bridge Motor Driver L298N and L298D

Feature	**L293N**	**L293D**
Image	 [20]	 [21]
Operating Voltage	5V	4.5 volt
Max Output Current (per channel)	2A	0.6 A
Number of Channels	Dual-channel	Dual-channel
Current Sense Pins	Available	Not available
Heat Sink	Included (for better heat dissipation)	Not included
Size	Slightly larger	Compact design
Protection Diodes	External diodes required for flyback	Internal flyback diodes included
Power Dissipation	Handles higher power dissipation	Handles lower power dissipation
Cost	8 Dollar	10 Dollar

The L293N motor driver is the best choice for our project because it provides a higher current capacity (up to 2A per channel), supports a wider operating voltage range as mentioned in Table 3.5, and includes a heat sink to improve thermal performance. These features make it the most suitable option.

3.3 Software Components

In this section, we will introduce the software components we're going to use.

3.3.1 Object detection and recognition using YOLO

YOLO is a widely recognized object detection algorithm implemented in Python and available on GitHub. It stands out among other object detection methods due to its unique approach to predicting both the class of an object and the bounding box that defines its location within an input image. Unlike traditional methods such as Region-Based Convolutional Neural Networks (R-CNN) or its variants like Fast R-CNN and Faster R-CNN, which involve a two-stage process (first generating region proposals and then classifying them), YOLO adopts a single-stage approach. This means YOLO simultaneously predicts multiple bounding boxes and class probabilities directly from full

images in a single evaluation. As a result, YOLO is significantly faster than these region-based methods, making it well-suited for real-time applications [22].

While other algorithms like Single Shot MultiBox Detector (SSD) also use a single-stage approach, YOLO differs in how it processes the image. YOLO divides the image into a grid and assigns each grid cell the responsibility of predicting a certain number of bounding boxes and class probabilities. This method leads to fewer background errors compared to SSD and typically offers a better trade-off between speed and accuracy [23].

Through extensive research, developers have found that YOLO can accurately distinguish between vehicles, people, and traffic signals based on the algorithm's confidence score as shown in Figure 3.5. The algorithm detects these objects as soon as they enter the camera's view on the road, analyzing their positions to make informed decisions about whether the vehicle should continue driving or stop.

Our choice of YOLO as the algorithm for object detection is driven by its reliability, efficiency, and ease of implementation. Compared to other options, YOLO offers a balanced combination of speed and accuracy, making it an ideal solution for real-time object detection tasks, such as distinguishing between vehicles, people, and traffic lights. To implement YOLO, we utilized the RoboFlow website, which simplified the model training process, allowing us to quickly generate and fine-tune the model for our specific needs.

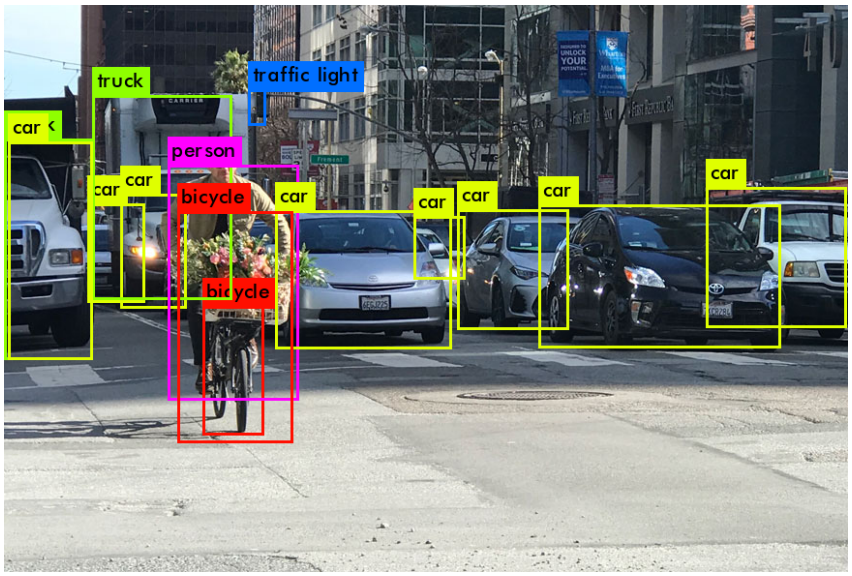


Figure 3.5: YOLO object detection and recognition [24]

3.3.2 System Navigation

In our indoor navigation system, the integration of YOLO and OpenCV offers a robust solution for real-time pathfinding and object detection. YOLO serves as the backbone for detecting and tracking the car's position on the map, ensuring accurate localization within the environment. By training YOLO to recognize the car and specific markers on the map, the system can effectively identify and classify these objects in real-time,

allowing for dynamic updates to the car's position as it navigates through the mapped area. Additionally, the user can specify the desired endpoint directly on the displayed map, enabling the system to calculate the optimal route based on the detected paths and guide the car toward that endpoint with precision. OpenCV complements YOLO by handling the analysis of the map to identify possible paths and navigable routes. With OpenCV's image processing techniques, such as edge detection and contour finding, the system can delineate potential paths on the map as shown in Figure 3.6, guiding the car along the optimal route determined by YOLO [25].

By combining YOLO's advanced object detection capabilities with OpenCV's versatile image processing tools, our project can develop a seamless and intelligent indoor navigation system. This integration enables real-time adjustments to the car's trajectory based on detected paths and objects, ensuring safe and efficient movement within the indoor environment. The ability for users to specify the endpoint on the map further enhances the system's usability, allowing for flexible and user-driven navigation in complex indoor settings.

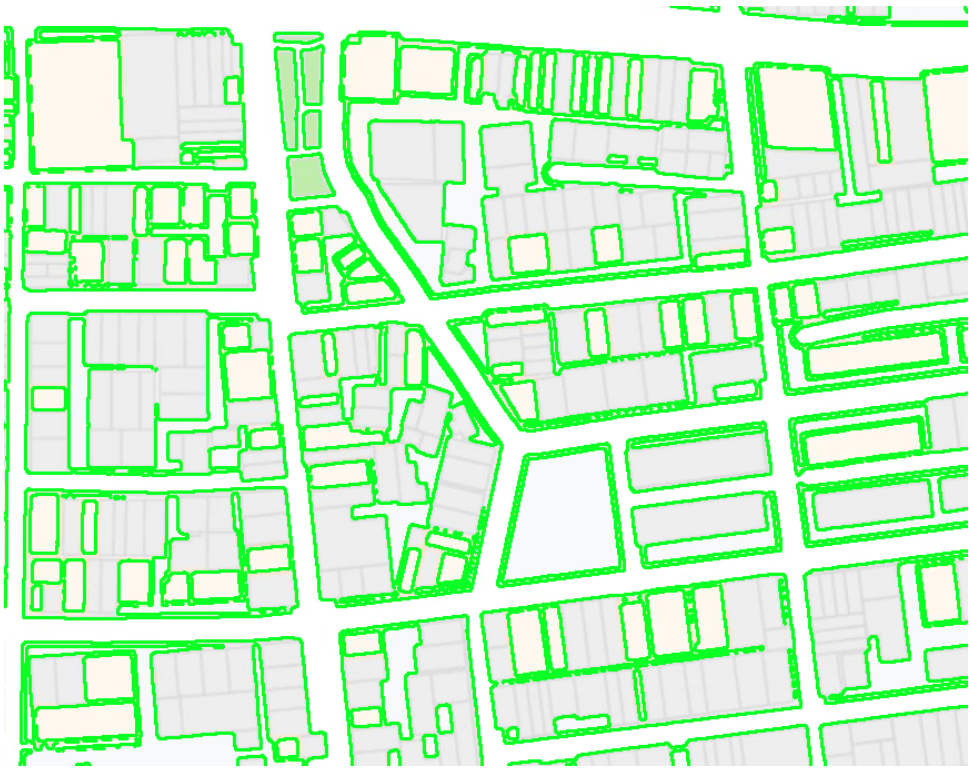


Figure 3.6: Map construction using OpenCV. [26]

3.4 Conceptual System Design

The conceptual system design for our project is modeled as a prototype of a self-driving car that leverages advanced computer vision technology to simulate real-world autonomous vehicle behavior. At the heart of this system is a Raspberry Pi microcontroller connected to a camera mounted on the car. This camera continuously captures images of the surrounding environment, identifying and classifying objects such as traffic lights, and stop signs. The system then reacts to these detections in a manner consistent with real-life driving scenarios, as shown in figure 3.7

The recognition system is responsible for constantly scanning for lane markings and instructing the motors to make lane changes as necessary. If an obstacle is detected in the car's path, the object detection system triggers the reaction system to stop the car, ensuring safety. In addition, the system can recognize the color of traffic lights, allowing it to stop or proceed based on whether the light is red or green. Road signs are also detected, read, and interpreted by the system, which adjusts the car's actions accordingly.

An alternative navigation method to GPS is integrated into the system, utilizing a camera mounted on a laptop placed above a map. This camera employs computer vision techniques to accurately determine the car's location on the map. The system then communicates these location data to the car, enabling it to calculate the optimal path from its current position to a user-specified destination. The car adheres to the recognized paths, using the available data to navigate effectively within the mapped environment. This integration of computer vision for both object detection and indoor navigation offers a comprehensive simulation of autonomous driving in controlled environments, as shown in the system conceptual diagram in figure 3.8.

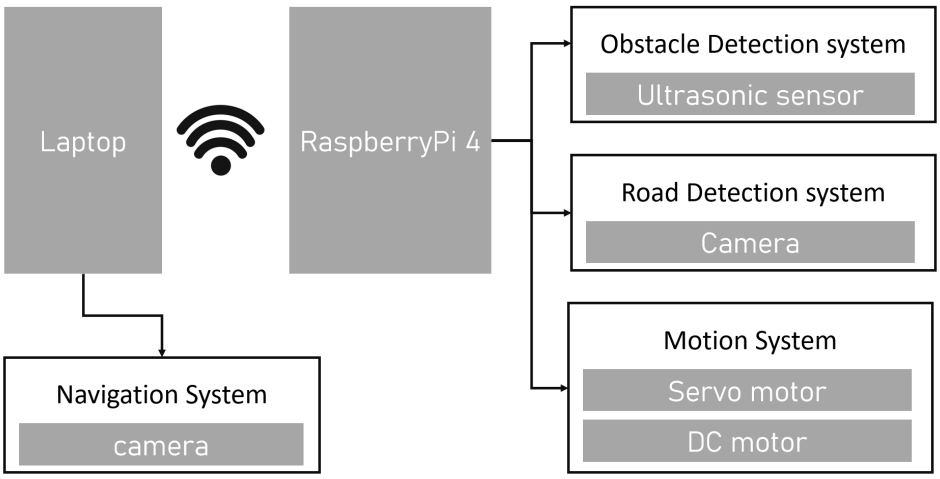


Figure 3.7: System block diagram

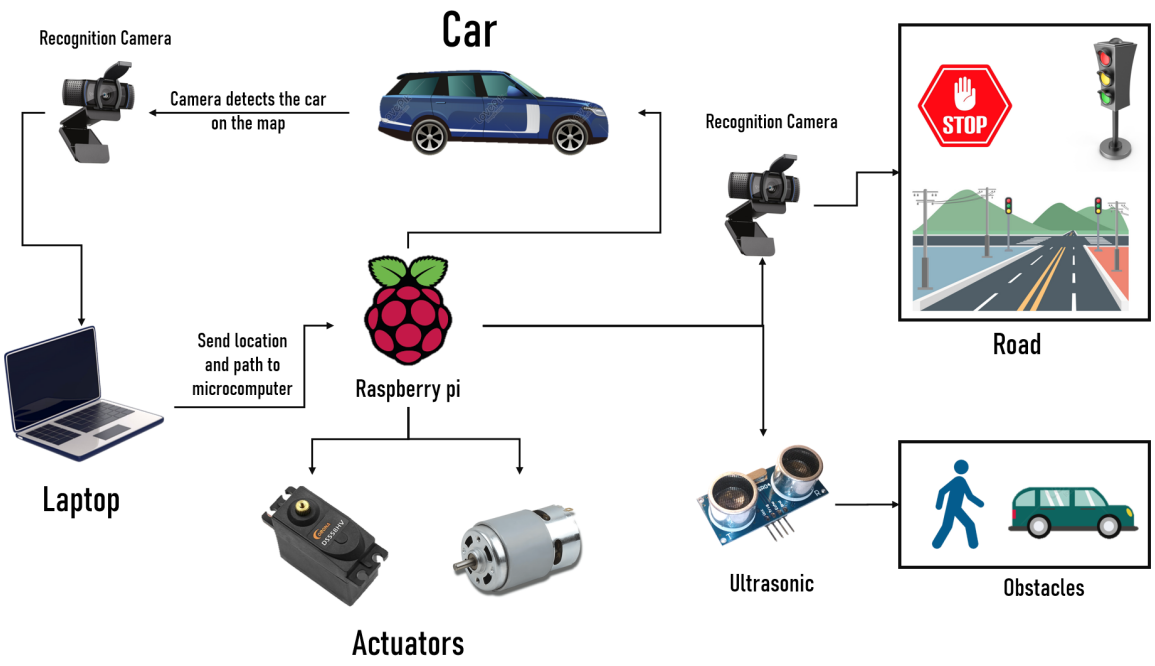


Figure 3.8: System conceptual diagram

3.5 Algorithms and methodologies

Algorithm 1 Car Algorithm - Idle Mode

```
1:  while Car is ON do
2:      if User specifies a destination using the laptop terminal then
3:          while Car has not reached the destination do
4:              if Next location is available from the navigation system then
5:                  Switch to Driving Mode
6:              else
7:                  Wait for the next location from the navigation system
8:              end if
9:          end while
10:         Turn off the car
11:     else
12:         Display "The car is stopped because no valid path is set"
13:         Stay in Idle Mode
14:     end if
15: end while
```

Algorithm 2 Car Algorithm - Driving Mode

```
1: while Car is Driving do
2:     if Obstacle is detected using Ultrasonic sensors then
3:         Stop the car
4:     else
5:         Capture images using the car's camera and analyze them
6:         if Stop sign is detected then
7:             Stop the car for 5 seconds
8:         else if Traffic light is detected then
9:             if Light is Red or Yellow then
10:                Stop the car
11:            end if
12:        else
13:            Predict the steering angle from the captured images
14:            Adjust the wheels to the predicted angle
15:            Run the motor
16:            if Car reaches the next location then
17:                Stop the motor
18:                Switch to Idle Mode
19:            end if
20:        end if
21:    end if
22: end while
```

Algorithm 3 Navigation System Algorithm

```
Build the map using OpenCV
while Car is not detected do
    Capture an image using the navigation system camera
    Attempt to find the car's initial position using YOLO from the image
    Print "Can't find the car's location on the map"
end while
Display the car's position and map to the user
if User specifies a destination then
    Calculate the best path for the car using the BFS algorithm
    Show the path to the user
    if User presses "Start" then
        Send the car's position and the next location to the car
        while Car has not reached its destination do
            Capture an image using the navigation system camera
            Update the car's position using YOLO
            Send car position to the car
            if Car's position matches the next location then
                if No further locations in the path then
                    Mark the car as having reached its destination
                    End the travel and stop the program
                else
                    Send the next location to the car
                end if
            else
                Wait until the car reaches the next location
            end if
        end while
    else
        Stop the program
    end if
else
    Stop the program
end if
```

3.5.1 Sequence Diagram

In this section we will review the sequence diagram for navigation and path planning as shown in Figure 3.9, in addition to the sequence diagram for driving and obstacles detection as shown in Figure 3.10.

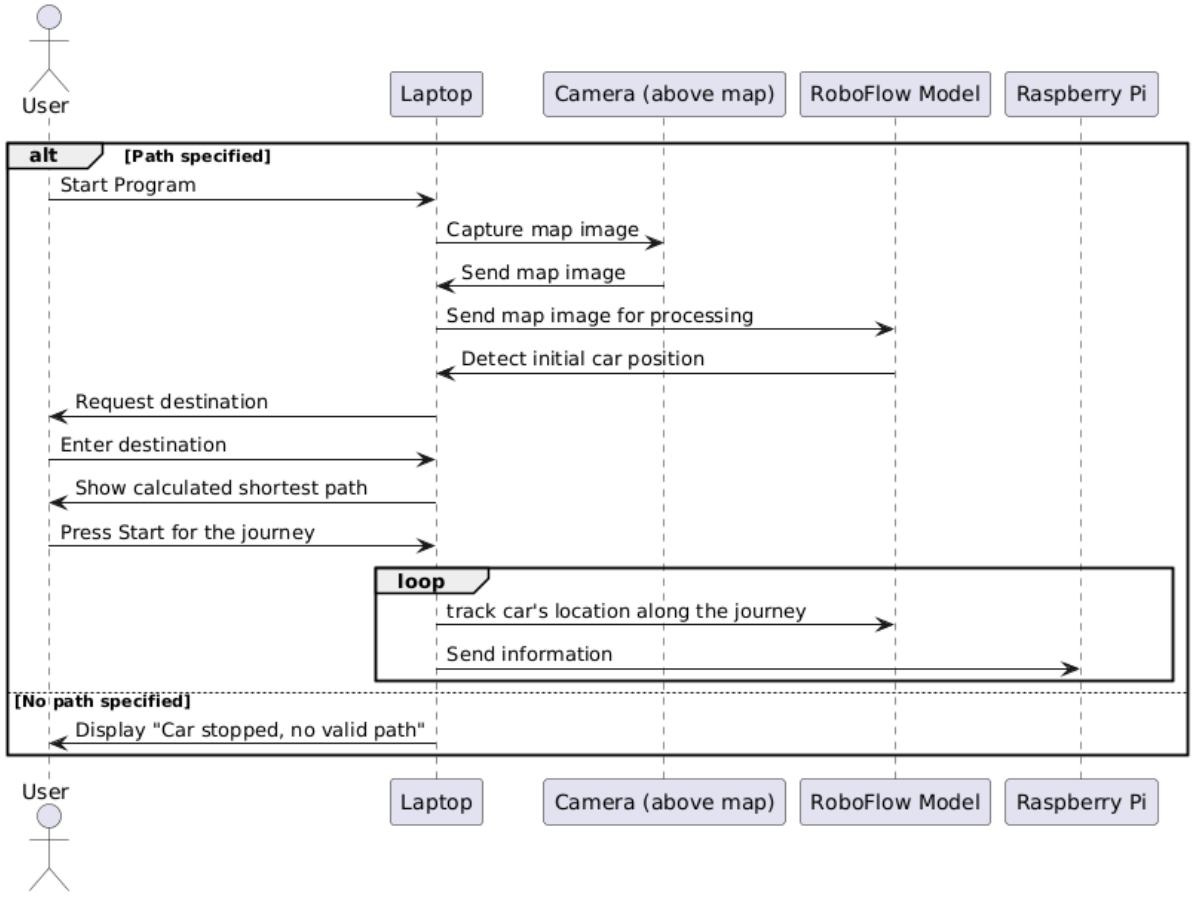


Figure 3.9: Navigation and Path planning Sequence diagram

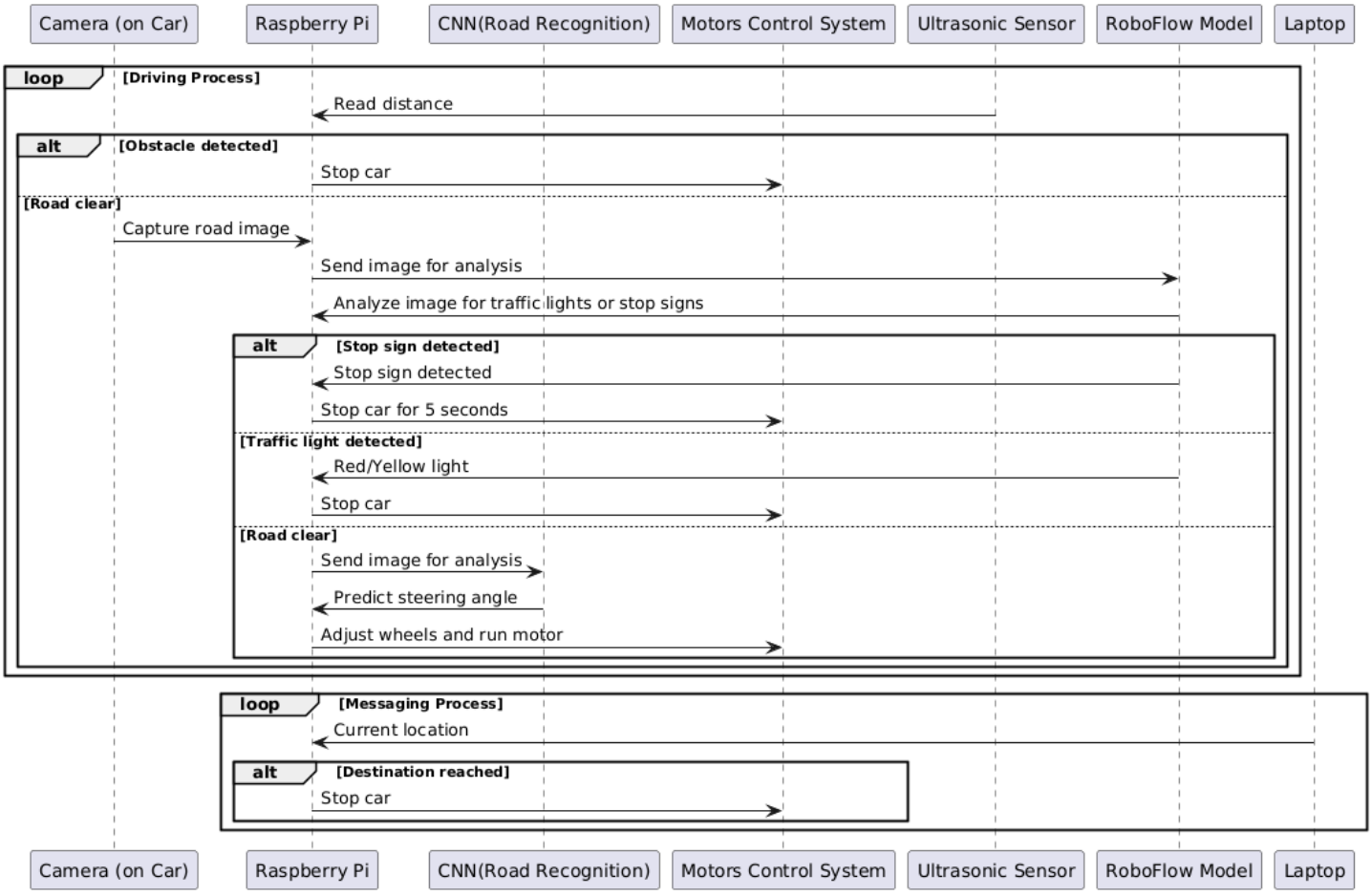


Figure 3.10: Driving and Obstacle detection

3.6 Schematic Diagram

Figure 3.11 shows the connection between the main components in the autonomous car.

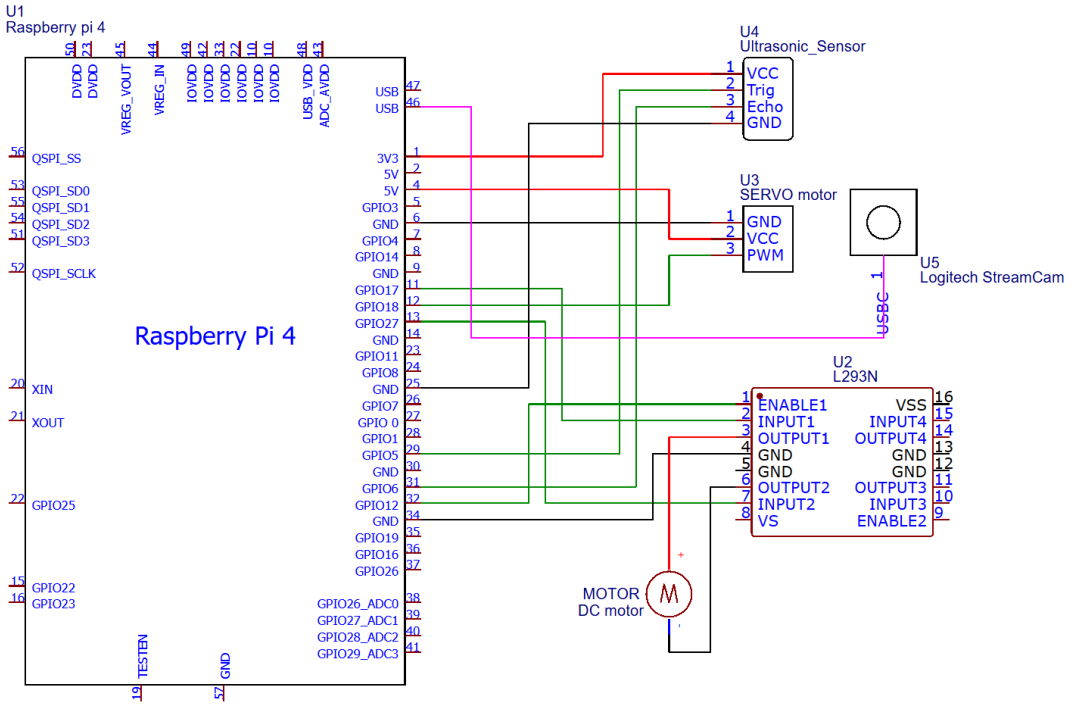


Figure 3.11: Schematic Diagram

3.7 Summary

In this chapter, we have examined the hardware and software components of the system, along with alternative options for each. We have provided a conceptual overview of the system and outlined its general flow, complemented by essential diagrams for clarity.

4 Chapter 4: Implementation and Testing

4.1 Overview

This chapter provides an overview of the hardware and software implementation. It outlines the integration of hardware components, the design, and implementation of software systems. The chapter aims to detail the systematic approach taken to ensure the successful realization of the project.

4.2 Hardware Implementation

In the previous chapter, we showed the components required to complete the project and how to connect them together. In this section of this chapter we will show how to assemble these components and how they will connect to the navigation system and the Raspberry Pi 4.

4.2.1 Prototype Car Setup

Our prototype self-driving car has a basic structure designed to hold a camera, a distance sensor, a controller, and the motors needed to move the car, as well as the rest of the components needed to drive the car, as shown in Figure 4.1.

Figure 4.1: Prototype Components

4.2.2 Navigation System Setup

The navigation system, as shown in Figure 4.2, consists of a stand with a camera mounted on it, connected to a laptop. The system also includes a lighting setup to enhance visibility. Together, these components work to detect the car's location on the map. Based on this information, the system calculates the shortest path from the starting point to the destination.

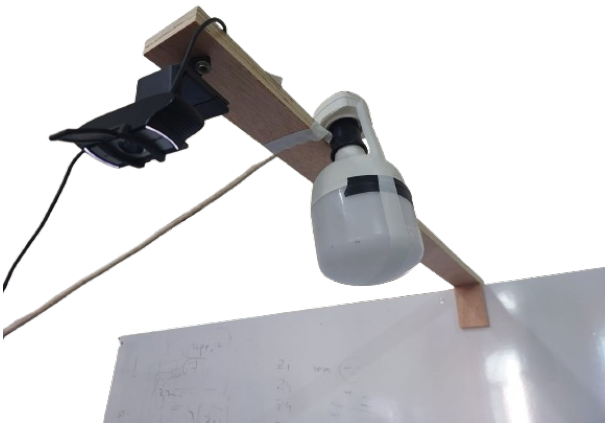


Figure 4.2: Navigation setup

4.2.3 Road, road signs and obstacle recognition system

We connected the ultrasonic sensor and the camera to the Raspberry Pi microcomputer. As shown in Figure 4.3, these components work together to detect and recognize the road, stop signs, traffic signs, and obstacles.

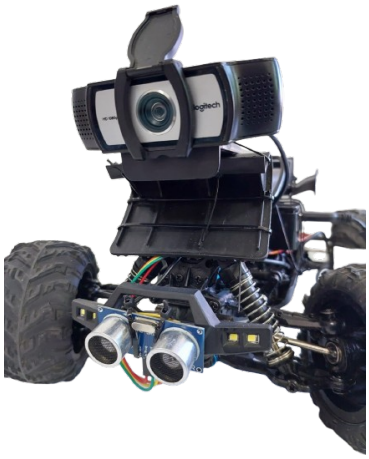


Figure 4.3: Recognition system setup

4.2.4 Motion System

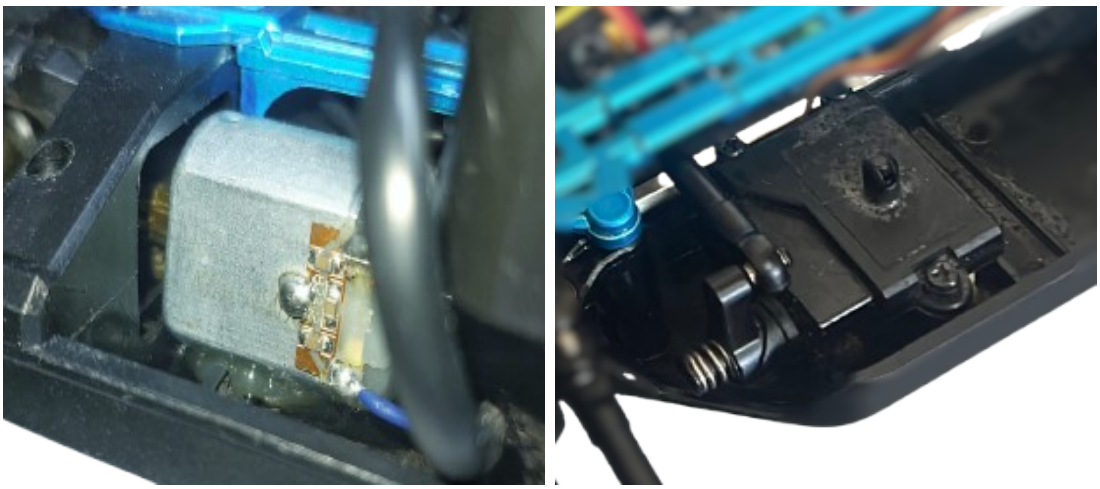
The motion system of our prototype car consists of two main components:

- **DC Motor:** The DC motor is responsible for moving the car forward and backward. It is connected to a motor driver and powered by a battery to ensure smooth and controlled motion.
- **Servo Motor:** The servo motor is primarily used to adjust the steering angle of the front wheels. It operates based on the predicted steering angle generated by the models we developed. The servo motor is directly connected to the Raspberry Pi 4 for precise control.

These components work together seamlessly, giving the prototype car the ability to mimic the driving behavior of real cars.

4.2.4.1 Motors Setup

We fixed the DC motor via a main axle connected to the front and rear wheels, and the servo motor was installed with a system that moves the front wheels to the right and left as shown in figure 4.4



(a) DC motor interfacing (b) Servo-motor interfacing

Figure 4.4: Motor Setup

4.2.4.2 Controlling and interfacing circuits

We connected a motor driver to a power supply and the Raspberry Pi to control the DC motor.as shown in figure 4.5

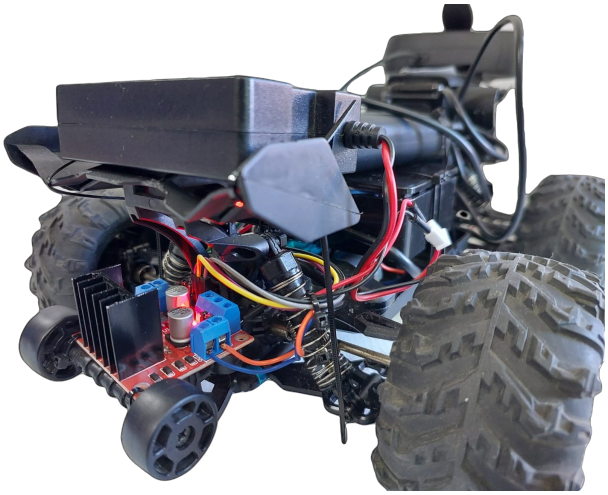


Figure 4.5: Motor Driver interfacing

4.3 Software implementation

4.3.1 Overall System

Our system is based on locating the car through the navigation system and determining the end point and then drawing the shortest possible path, then we send the information to the car using the AMQP protocol, the device hosts the Rabbit MQ broker, for this to happen we have to do some configuration.

4.3.2 AI Models Integration

Our system consists of four main AI models, each model has to be calibrated to suit our system requirements.

4.3.2.1 Steering Angle Prediction

Our steering angle prediction model employs image regression to determine the car's steering angle based on visual input. To train the model, we created a dataset of approximately 10,000 images captured from the car's perspective during a track drive. These images were evenly distributed across three driving scenarios: turning left, turning right, and driving straight, with around 3,000–3,200 images for each scenario. Each image was paired with its corresponding steering angle, recorded in a CSV file containing image paths and their respective steering angle values, ensuring precise and consistent labeling for training. This training approach follows the method described in Artificial Intelligence for Sustainable Applications [27].

We utilized three separate models to predict the steering angle in each specific state (left, right, straight). For feature extraction in each model, a 512-layer Convolutional Neural Network (CNN) was employed. This design allowed each model to specialize in learning nuanced features for its respective scenario, enhancing prediction accuracy.

4.3.2.2 Car Localization

This model utilizes the Roboflow 3.0 Object Detection framework to localize cars on a map and predict their directional orientation (East, West, South, or North). A dataset of over 850 images was collected and annotated to ensure accurate representation of car locations and directional classes. The data preparation and labeling process was conducted using Roboflow, ensuring consistency and precision for optimal model performance.

The model was trained using a dataset labeled with the correct information about objects, and Roboflow's tools were used for object detection. During training, we tracked important metrics like box loss (accuracy of object boundaries), class loss (accuracy of object labels), and object loss (accuracy of detecting objects). Figure 4.6 shows graphs that visualize how well the model performed and improved over time.

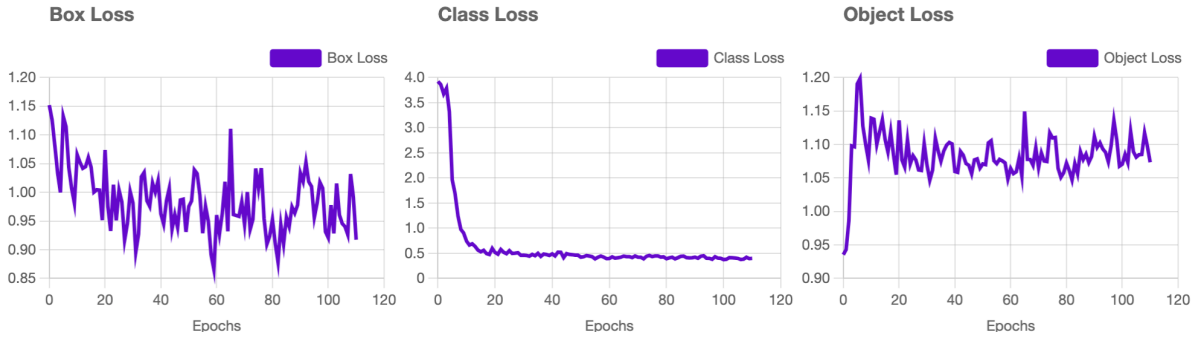
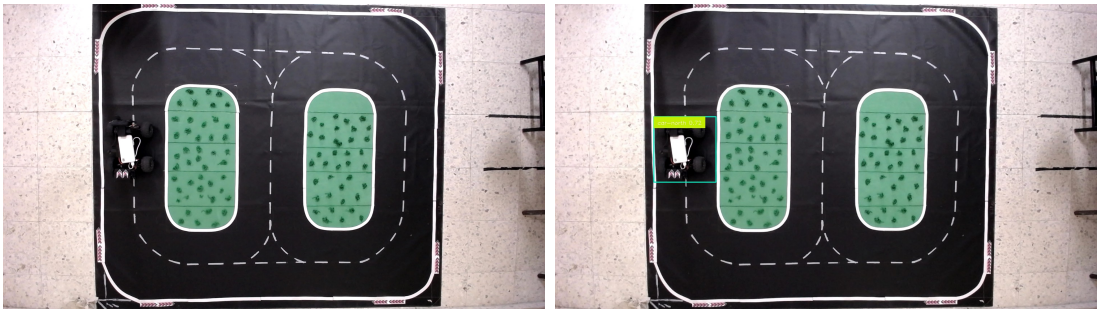
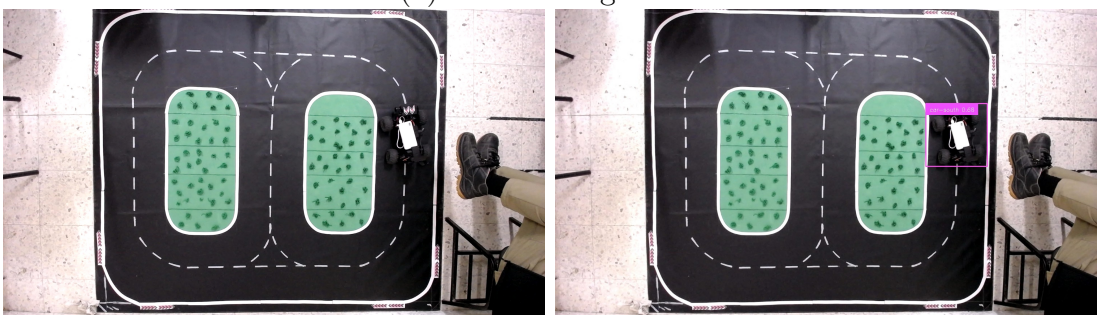


Figure 4.6: Location box loss, class loss, and object loss

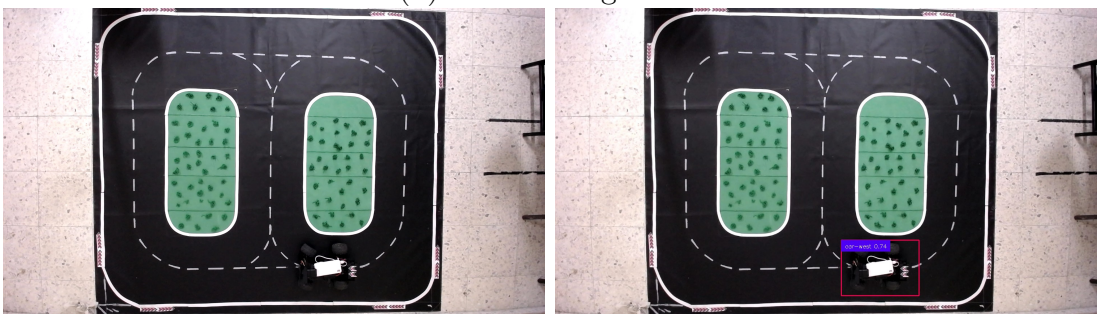
After training, the model was tested on unseen images to evaluate its accuracy and reliability. The results demonstrate its capability to precisely localize cars on a map and correctly predict their directions. Figure 4.7 showcases example outputs, highlighting the model's accurate detection of car positions and directional classifications, validating the robustness of this approach.



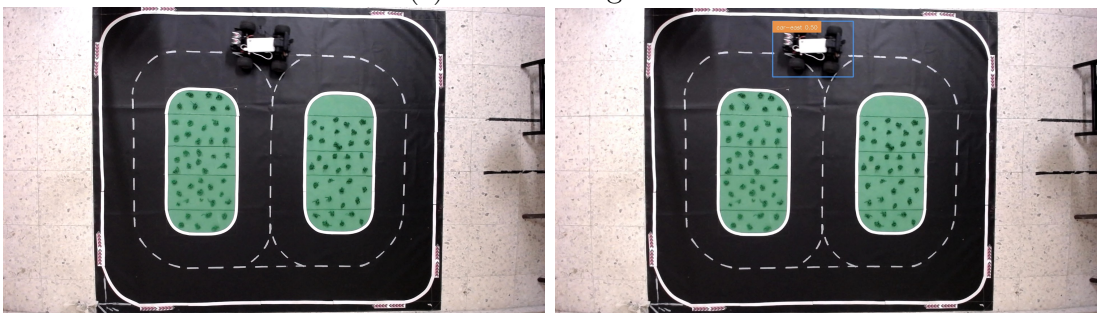
(a) Car heading north



(b) Car heading south



(c) Car heading west



(d) Car heading east

Figure 4.7: Localization Model Examples

4.3.2.3 Traffic Light Detection

The model uses a custom-trained Roboflow 3.0 Object Detection framework to predict traffic light states (green, red, or yellow) in real-time. It was developed using a dataset of over 750 images, annotated with precise bounding boxes and class labels, capturing various traffic light states across different environments.

The model was trained using the Roboflow 3.0 Object Detection framework, optimiz-

ing for box, class, and object losses. Training performance is shown in Figure 4.8. Tested on real-world data, the model accurately predicted traffic light states across various scenarios, as illustrated in Figure 4.9, highlighting its robustness in detecting green, red, and yellow lights.

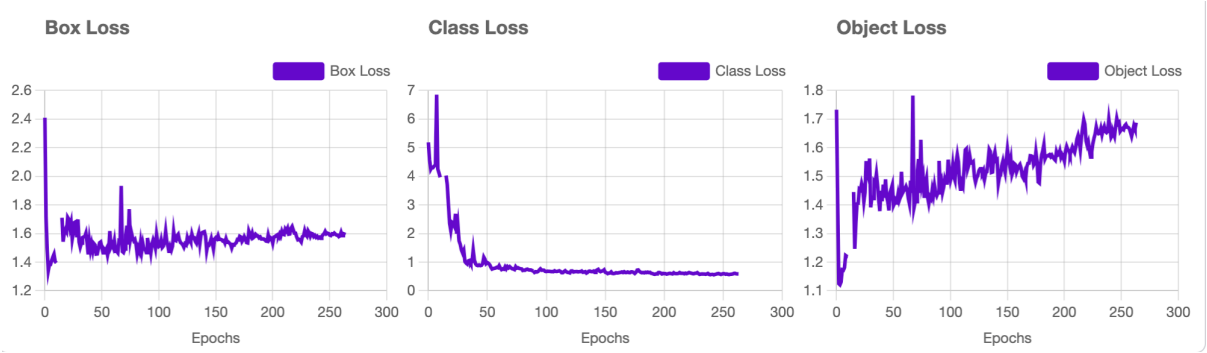
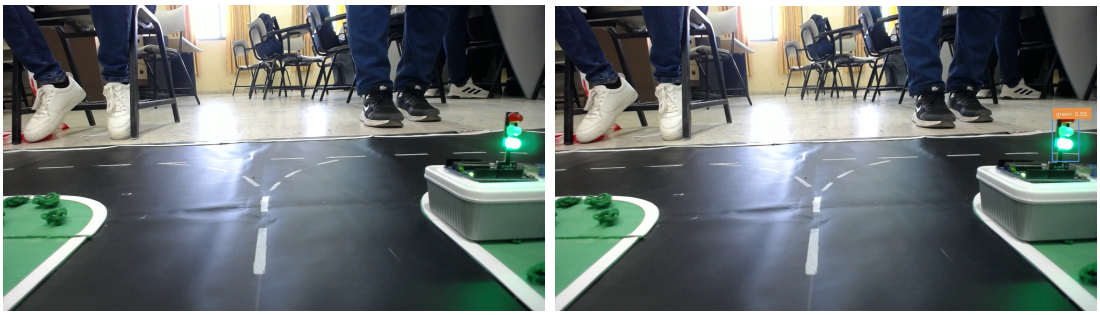
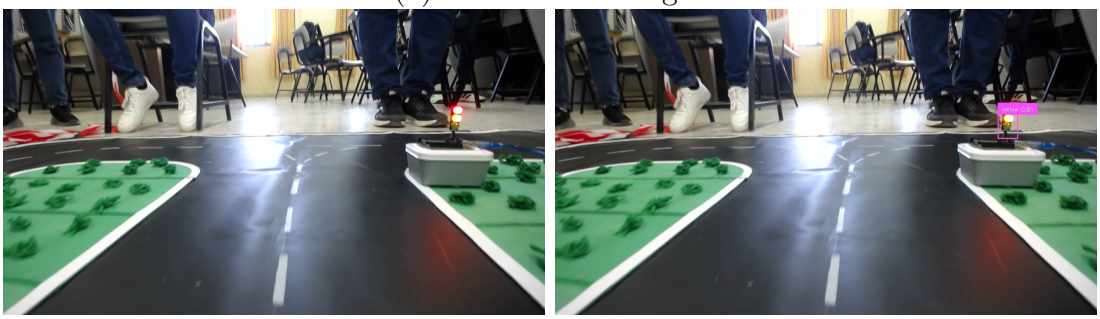


Figure 4.8: Traffic Light box loss, class loss, and object loss



(a) Green Traffic Light



(b) Yellow Traffic Light

(c) Red Traffic Light

Figure 4.9: Traffic light detection examples

4.3.2.4 Stop Sign Detection

This model utilizes Roboflow 3.0 Object Detection framework, a pre-trained deep learning model, to detect stop signs in real-time from a video feed. The script loads the Roboflow model , processes frames from the car camera, and applies the model to identify and localize stop signs within the captured frames. Figure 4.10 shows the output image of the model.

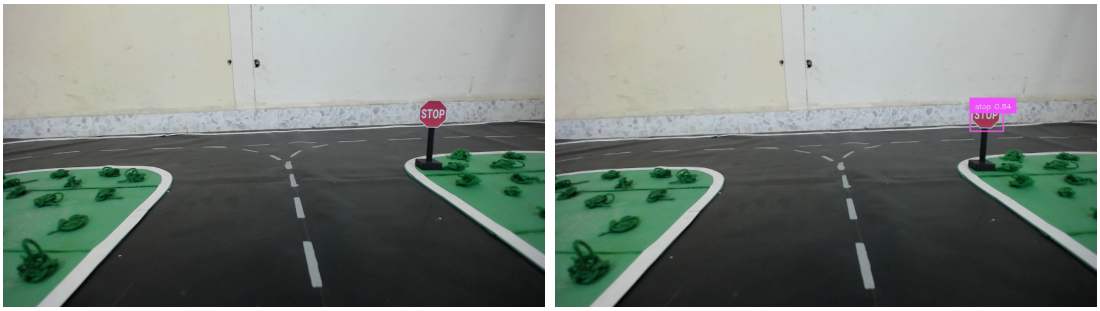


Figure 4.10: Stop sign detection

4.3.3 Navigation System

The navigation system in our autonomous driving car project integrates computer vision, machine learning, and message-based communication to provide a seamless user experience for controlling and monitoring the vehicle's movement. The terminal view, through which the user interacts with the system, is built using Python, allowing for good communication between the user and the vehicle.

4.3.3.1 Top-Down View Camera Integration

A top-down view camera is mounted to capture images of the environment. These images are analyzed in real-time using a pre-trained car detection model from Roboflow. This step identifies the car's current location and direction within the environment as shown in figure 4.11.

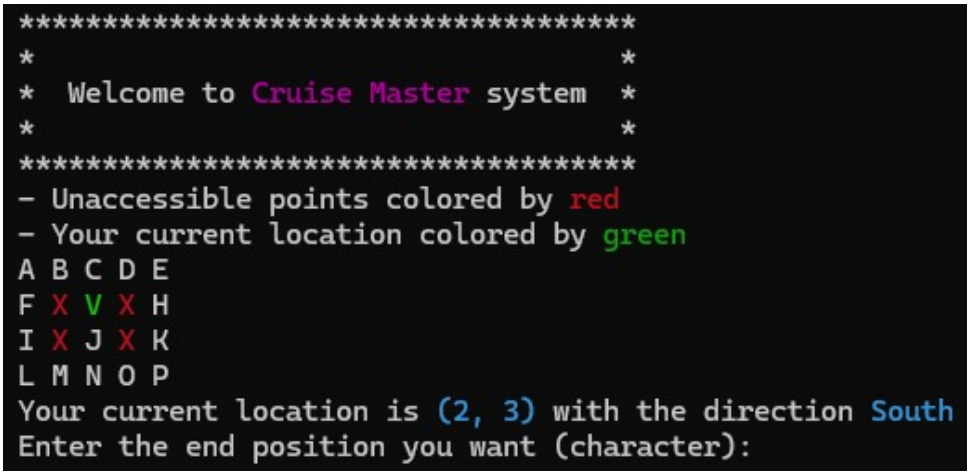


Figure 4.11: Car Location Terminal

4.3.3.2 User Interaction for Target Selection

After determining the car's position, the system prints its location to the user and prompts them to enter a target destination. The terminal interface, as illustrated in the figure 4.11, allows the user to choose one of the predefined possible target locations.

4.3.3.3 Path Planning

The system computes the optimal path using BFS algorithm to the selected destination. This path is translated into a series of directional commands (e.g., "straight," "left," "right") that guide the car as shown in the figure 4.12.

```
Your current location is (2, 3) with the direction South
Enter the end position you want (character): A
The destination you chose is (1, 1) at the character A
The expected path is:
 - ['V', 'J', 'N', 'M', 'L', 'I', 'F', 'A']
 - ['F','F','R','F','R','F','F']
Do you want to start the journey (Y / N)?
```

Figure 4.12: Calculated Path Terminal

4.3.3.4 Message Communication via RabbitMQ

The path directions are sent to the Raspberry Pi onboard the car using the AMQP protocol through RabbitMQ, a robust message broker running inside a Docker container. This ensures reliable communication between the path-planning system and the car's control system.

4.3.3.5 Car Movement and Feedback Loop

Once the car receives the next location in the path, it begins to follow the designated path. Simultaneously, a top-down view camera provides real-time feedback by continuously monitoring the car's position. This feedback is used to verify that the car remains on the correct path.

4.3.3.6 Arrival at Target Location

The car continues to follow the path and receive feedback until it reaches the specified target destination. Upon arrival, the system ensures the car stops at the correct location as shown in the figure 4.13

```
Your current location is (2, 3) with the direction South
Published: {"end": true, "start": true, "position": null}
You reached your destination successfully
```

Figure 4.13: Car Arrival at Target Destination Terminal

4.3.4 Driving System

The driving system integrates software and hardware components to control the car's movement and steering in real-time. Path directions calculated by the navigation system are transmitted to the car using the AMQP protocol via RabbitMQ. These commands include specific instructions for steering and motion, which are processed by the motion control system to ensure the car follows the designated path.

To execute steering adjustments, a servo motor receives precise angle predictions from the steering angle prediction model. The forward and backward movement is managed by a DC motor, with speed and direction controlled through regulated power signals. Together, these components allow the car to navigate seamlessly along the calculated path.

4.4 Implementation Challenges and Issues

In this section we will talk about the hardware and software challenges we faced in the project.

4.4.1 Hardware Implementation Challenges

Integrating the hardware components into a functioning system was a complex task. The car chassis required precise alignment of the DC motor, servo motor, and battery to ensure smooth operation. Ensuring that all components worked together seamlessly with the batteries was particularly challenging. We encountered difficulties in managing the power distribution, avoiding overloading, and maintaining consistent performance across the components.

4.4.2 Software Implementation Challenges

The software implementation of our system faced several challenges, primarily in synchronizing the driving and navigation subsystems. Achieving real-time communication between these components was complex, often leading to delays or inconsistencies that hindered smooth operation. Ensuring robust and reliable communication required extensive testing and optimization of the RabbitMQ message broker and control commands.

4.4.2.1 AI Model Challenges and Solutions

Developing AI models for autonomous driving introduced significant challenges, particularly when adapting existing pre-trained models to our environment. Initial attempts to use these models resulted in poor performance due to mismatches between the training conditions of the pre-trained models and the unique features of our project environment. Consequently, we opted to develop a custom AI model tailored to our specific requirements.

The dataset collection process posed additional obstacles. Our initial dataset of 8,500 images lacked sufficient features for precise steering angle predictions, and poor lighting conditions further degraded model accuracy. To address these issues, we enhanced the environment by improving lighting and incorporating more distinguishable features. As shown in Figure 4.14, these improvements significantly improved the quality of the training data, enabling more accurate predictions.

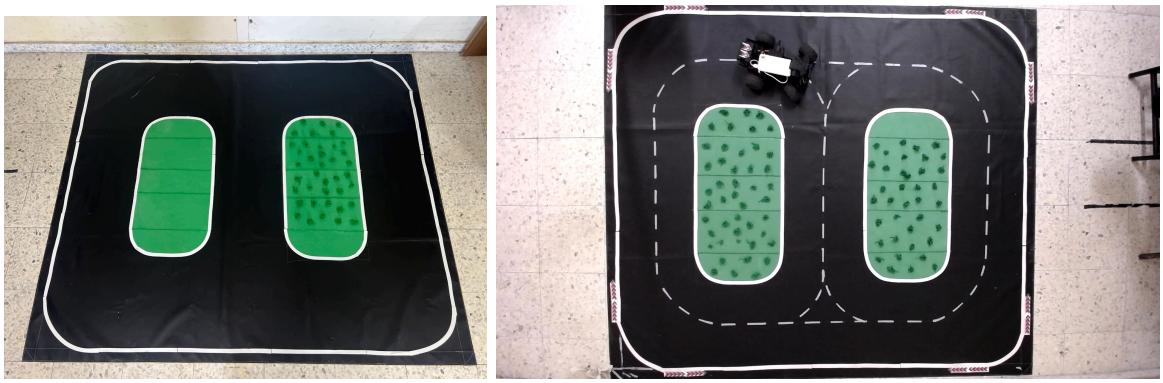


Figure 4.14: Environment Before and after

4.4.2.2 Decision-Making and Multi-Model Approach

A major challenge arose in ambiguous scenarios, such as intersections or corners, where the model struggled to decide between valid options (e.g., turning left or continuing straight). To resolve this, we implemented a multi-model strategy, training three specialized models for left turns, right turns, and straight movements. This approach required additional data collection for each scenario but proved highly effective. By dynamically selecting the appropriate model based on the car's location and destination, our system achieved reliable and accurate navigation. The final implementation demonstrated significant improvements in decision-making stability and overall system performance, ensuring the car could autonomously determine and execute the correct path.

4.5 Summary

In this chapter we reviewed the hardware and software implementation each component was fully explained and then we discussed the issues and challenges we faced during the implementation phase.

5 Chapter 5: Testing and Results

5.1 Overview

In this chapter we will present tests of all system components and their results.

5.2 Hardware Components Testing

5.2.1 Unit Testing

This section discusses the testing process of each of our hardware testing.

5.2.1.1 Raspberry Pi 4

We conducted comprehensive testing on the Raspberry Pi 4 to ensure its optimal functionality. This process included verifying the performance of all hardware components, such as the microSD card, power supply, USB ports, and network connectivity. The Raspberry Pi 4 successfully passed all tests, confirming its capability to meet the requirements of our project.

5.2.1.2 DC Motor

We conducted thorough testing of the DC motor and its motor driver to ensure proper functionality and integration. The testing involved controlling the motor's speed and direction through our system's control signals. The motor demonstrated smooth and responsive performance, meeting all expected operational standards.

5.2.1.3 Servo Motor

The servo motor was tested by executing a series of angular movements to assess its precision, responsiveness, and accuracy. It consistently achieved the desired positions without delays or errors, confirming its reliability for our application.

5.2.1.4 Batteries

The batteries were evaluated under various load conditions to verify their capacity and power output. The tests ensured stable and consistent power delivery throughout the operation, demonstrating their ability to support the system's energy requirements effectively.

5.2.1.5 Cameras

We connected one camera to the Raspberry Pi's USB port and the other to the laptop's USB port. Both cameras functioned as expected, successfully capturing their respective streams.

5.2.1.6 Ultrasonic sensor

We thoroughly tested the ultrasonic sensor to ensure its functionality within our system. This involved verifying the accuracy and reliability of its distance measurements under various environmental conditions. We assessed the sensor's response time and consistency by measuring known distances and comparing the results with expected values. Additionally, we tested its integration with our control system to confirm it reliably triggered the appropriate responses. The ultrasonic sensor passed all tests successfully, demonstrating its capability to deliver precise and dependable distance measurements for our application.

5.3 Software Components Testing

5.3.1 Unit Testing

This section discusses the testing process of each of our software testing.

5.3.1.1 Raspberry Pi OS installation

We evaluated and tested multiple versions of the "Raspbian OS" to identify the one that best met our requirements. Ultimately, we selected the Raspberry Pi OS (64-bit) for our project, as it provided the necessary features and functionality.

5.3.1.2 Steering Angle Prediction

To evaluate the performance of the steering angle prediction model, we conducted real-world tests by running the car and ensuring it successfully navigated to its destination. The testing process involved deploying the model on the vehicle and driving through various scenarios, including:

5.3.1.3 Traffic Light and Stop Sign Detection

We evaluated our traffic light color and stop sign detection model using a dataset of 664 images, split into three subsets:

- Training Set: 70% of the images, used for training the model and allow it to learn from the data.
- Validation Set: 20% of the images, used to fine-tune the model's parameters and prevent overfitting.
- Testing Set: 10% of the images, used to evaluate the model's performance and generalize its accuracy on unseen data.

The dataset was carefully curated to include diverse conditions, such as varying lighting, angles, and occlusions, to simulate real-world scenarios effectively.

The model's performance was measured using precision, recall, and mean Average Precision (mAP) metrics: Precision: 98.5%, Recall: 99.3%, mAP: 99.1% The mean Average Precision (mAP) is calculated as the average of the Average Precision (AP) values across all classes. In this case, the high mAP value indicates that the model consistently detects and classifies the colors of traffic lights with exceptional accuracy.

- Precision (98.5%): Indicates the model's ability to correctly identify traffic light colors without producing false positives.
- Recall (99.3%): Demonstrates the model's effectiveness in identifying all relevant traffic light colors and stop sign in the dataset, with minimal false negatives.
- mAP (99.1%): Confirms the model's overall high performance across all classes of traffic light colors and stop sign.

Figure 5.1 illustrates an example of the model's detections on the test image, showcasing its reliability in identifying red, yellow, and green lights under varied conditions.

Figure 5.1: Trafiic Light Detection Example

5.3.1.4 Car Location Detection

We evaluated our car location recognition model using a dataset of 836 images, classified into four distinct classes:

- Car East
- Car West
- Car North
- Car South

The dataset was divided into three subsets:

- Training Set: 70% of the images, used for training the model and allow it to learn from the data.
- Validation Set: 20% of the images, used to fine-tune the model's parameters and prevent overfitting.
- Testing Set: 10% of the images, used to evaluate the model's performance and generalize its accuracy on unseen data.

The dataset was designed to encompass a wide range of real-world scenarios, such as varying lighting conditions, angles, and environmental factors, ensuring the model's robustness and reliability.

The performance of the model was assessed using standard metrics, yielding the following results:

- Precision (94.4%): Indicates the model's ability to correctly identify car locations without producing false positives.
- Recall (96.2%): Demonstrates the model's effectiveness in identifying all relevant car locations with minimal false negatives.
- mAP (98.5%): Confirms the model's overall high performance across all classes, calculated as the mean of the Average Precision (AP) values for each class.

Figure 5.2 illustrates an example of the model's detections on a test image, showcasing its reliability in accurately recognizing cars' positions (east, west, north, and south) under varied conditions. This high level of performance underscores the model's potential for deployment in real-world car detection systems.

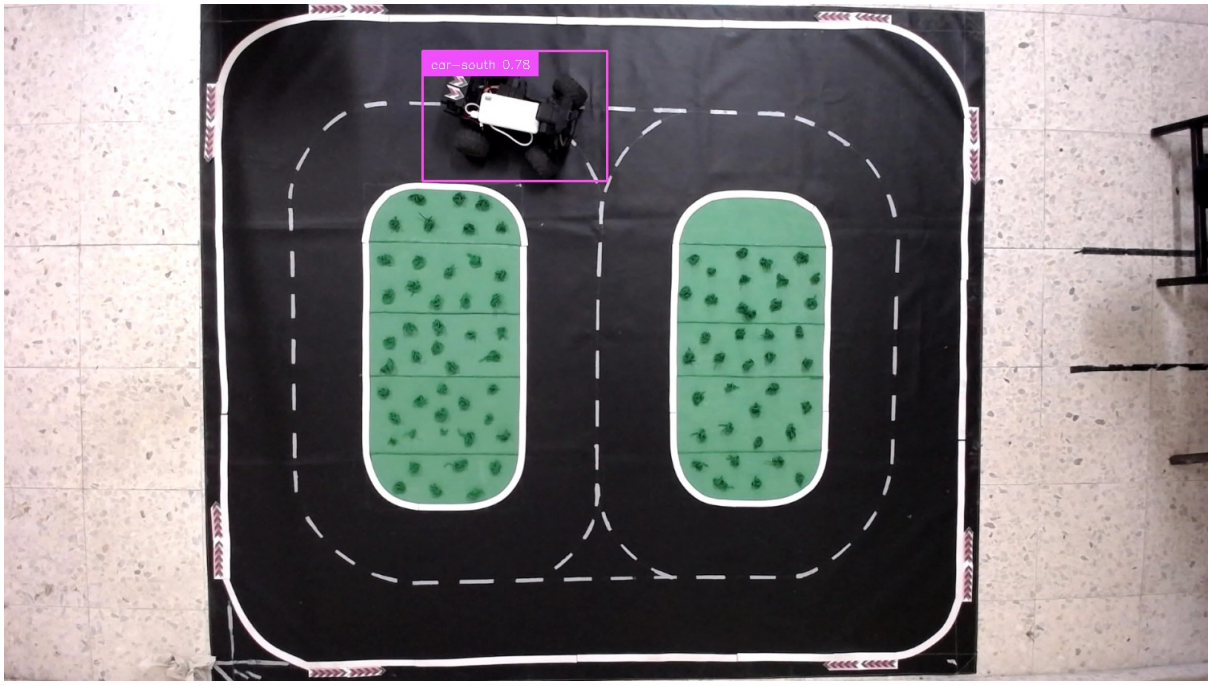


Figure 5.2: Car Location Recognition Example

5.3.1.5 Broker

We utilized RabbitMQ as our communication protocol, implemented through Docker, due to its scalability and reliability. After installing RabbitMQ, we conducted thorough testing to evaluate its reliability and performance. The tests demonstrated stable message transmission with low latency, averaging 5–10 milliseconds per message. RabbitMQ passed all tests successfully, confirming its capability to deliver robust and efficient communication for our system.

5.3.1.6 Response Time

Our autonomous driving car, continuously processes data from its sensors and cameras to make real-time decisions on the road. The system evaluates environmental conditions, traffic patterns, and potential hazards, ensuring the vehicle reacts instantly to changes. When a critical situation is detected, such as an obstacle or sudden traffic change, the system initiates an appropriate response, whether it's braking, steering adjustments, or route recalibration. The overall reaction time is optimized for safety and efficiency, relying on key factors such as sensor accuracy, processing power, network latency, and the performance of the onboard microcontroller. These elements work seamlessly to deliver a reliable and adaptive autonomous driving experience.

5.4 Integration Testing

To test the integration of all systems, we conducted end-to-end testing to ensure seamless communication and functionality across the localization, path calculation, and driving systems. The process began with the camera above the map capturing an image, which was sent to the PC for preprocessing and passed through the Roboflow model to detect the car's position. The localization system then calculated the shortest path to the user-defined target destination using the BFS algorithm and transmitted the path details to the RC car.

The RC car received these instructions and used its onboard camera to capture images for navigation. Based on the path details, the driving system interpreted the images to make real-time decisions—moving straight, turning left, or turning right—while simultaneously monitoring the road for traffic lights and stop signs. Comprehensive testing demonstrated that all components worked in harmony, with accurate localization, efficient path calculation, and reliable real-time navigation. The integrated system performed as expected, successfully guiding the car to its destination while adhering to traffic rules and environmental constraints.

5.5 System Testing

5.5.1 Localization System Testing

To test the localization system, we began by positioning a camera above the map to capture images of the environment. These images were sent to the PC, where preprocessing was applied to enhance the input data. The processed images were then passed through a Roboflow-trained model to detect the car and determine its exact location on the map. Using the x and y coordinates of the detected car, the system successfully identified and printed the car's location in the terminal. As the car started moving, the system continuously captured images, processed them, and updated the car's location in real-time. Extensive testing was conducted, and the entire system, including image capture, preprocessing, model inference, and location updates, performed as expected without any errors.

Once the car's position was established, the system calculated the shortest path to the user defined target destination using Breadth-First Search (BFS). The calculated path was sent to the RC car, which followed the path by navigating step-by-step based on the instructions provided by the BFS algorithm. As the car moved, the system continuously monitored its position, updated the localization in real-time, and ensured the car remained on track. Extensive testing confirmed that the entire process from localization and path calculation to real-time navigation operated seamlessly and achieved the desired outcomes.

5.5.2 Driving System Testing

To test the driving system, we began by providing the car with the path details calculated by the localization system. Based on these details, the car utilized its onboard camera to capture images and determine its next move. The onboard model processed the captured images to decide whether the car should proceed straight, turn left, or turn right, depending on the calculated path.

During its journey, the car continuously monitored the road environment to ensure it followed the designated path while adhering to traffic rules. The system successfully detected traffic lights and stop signs along the way, making appropriate decisions such as stopping when necessary. Extensive testing confirmed that the driving system accurately interpreted the path instructions, processed environmental inputs, and navigated the road safely and efficiently.

5.6 Summary

In this chapter, we discuss the hardware and software testing.

6 Chapter 6: Conclusion

6.1 Overview

The chapter introduces a summary of the project, the future directions, and future work.

6.2 Conclusion

The Cruise Master system presents an innovative approach to autonomous driving, seamlessly integrating localization, navigation, and driving systems to ensure a safe and intelligent self-driving experience. The system is designed to tackle road safety challenges by leveraging cutting-edge computer vision and deep learning technologies.

The Cruise Master achieves this through a three-stage process: first, the localization system uses a top-down camera to capture and analyze the car's position on a map, utilizing YOLO-based object detection and BFS for path planning. The system then calculates the shortest route to the user-defined destination and sends path details to the car. The driving system processes these path details, capturing real-time images with an onboard camera to determine the car's next move—be it moving straight, turning left, or turning right—while ensuring compliance with road signs, traffic lights, and obstacles.

This integration enables the car to make real-time decisions, adapt to dynamic environments, and autonomously navigate towards its destination. Key features include obstacle detection, adherence to traffic rules, and a user-friendly interface for endpoint selection. While the system currently operates in controlled environments, it lays a solid foundation for scalable autonomous driving solutions, showcasing significant advancements in the field of intelligent transportation systems.

6.3 Future Work

- **Enhanced AI Models:** Improve the accuracy and efficiency of the AI driving models (left, right, and straight) by collecting additional training data and conducting extended training sessions. This will enhance the driving precision and reliability of the system.
- **Integration of a Real GPS System:** Replace the custom GPS system with a real GPS module and test the system in real-world environments, such as actual roads and traffic scenarios.
- **Deployment in Real Vehicles:** Transition from a prototype setup to implementing the system in a full-scale vehicle to evaluate its performance under real driving conditions.
- **Comprehensive Road Sign Recognition:** Expand the system's capabilities to recognize a broader range of road signs, beyond just stop signs and traffic lights.

- **Advanced Driving Features:** Incorporate additional functionalities such as lane-changing techniques, autonomous parking, cruise control, and interaction with other vehicles.

References

[1] PCBS, "Main indicators related to road traffic accidents in palestine 2015- 2022," 2023, accessed on May. 10, 2024. [Online]. Available: https://www.pcbs.gov.ps/statisticsIndicatorsTables.aspx?lang=en&table_id=2032

[2] S. BLANCO, "2019 u.s. traffic deaths lowest since 2014, but 2020 numbers aren't looking good," 2020, accessed on May. 10, 2024. [Online]. Available: https://www.caranddriver.com/news/a34240145/2019-2020-traffic-deaths-coronavirus

[3] S. C. Vinayak V. Dixit and D. J. Nair, "Autonomous vehicles: Disengagements, accidents and reaction times," *PLOS ONE*, vol. 11, no. 12, p. 10/14, 2016.

[4] B. Marr, "7 amazing examples of computer and machine vision in practice," 2019, accessed on May. 15, 2024. [Online]. Available: https://www.forbes.com/sites/bernardmarr/2019/04/08/7-amazing-examples-of-computer-and-machine-vision-in-practice/?sh=4f71f2191018

[5] NVIDIA, "Convolution online photo," 2024, accessed on August. 21, 2024. [Online]. Available: https://www.nvidia.com/en-us/glossary/convolutional-neural-network

[6] N. Shahriar, "What is convolutional neural network — cnn (deep learning)," 2023, accessed on May. 15, 2024. [Online]. Available: https://nafizshahriar.medium.com/what-is-convolutional-neural-network-cnn-deep-learning-b3921bdd82d5

[7] B. H. W. H. X. G. F. Z. WeiHang Feng, Yun Pei, "Autonomous rc-car for education purpose in istem projects," *IEEE Intelligent Vehicles Symposium*, vol. 10, no. 1109, 2018.

[8] RROIJ, "Intelligent path planning of mobile robot/agent by using breadth first search algorithm," *RROIJ*, 2021, accessed: 2025-02-03. [Online]. Available: https://www.rroij.com/open-access/intelligent-path-planning-of-mobile-robotagent-by-using-breadth-first-search-algorithm.pdf?utm_source=chatgpt.com

[9] J. Schulze, "What is the internet of things (iot)? with examples," 2024, accessed on May. 15, 2024. [Online]. Available: https://www.coursera.org/articles/internet-of-things

[10] J. Haiston, "Iot protocols vs iot standards," 2023, accessed on May. 15, 2024. [Online]. Available: https://www.symmetryelectronics.com/blog/iot-protocols-vs-iot-standards/

[11] L. M. Abed Al-Fattah Hroub, Deema Al-Hafiz, "An embedded system in autonomous vehicles for pedestrian detection using deep learning," College of IT and Computer Engineering/Palestine Polytechnic University, 2021, graduation Project.

[12] A. singh, "Raspberry pi photo," 2023, accessed on August. 24, 2024. [Online]. Available: https://www.hackatronic.com/raspberry-pi-4-specifications-pin-diagram-and-description/

[13] D. Franklin, "Nvidia jetson nano photo," 2019, accessed on August. 24, 2024. [Online]. Available: https://developer.nvidia.com/blog/jetson-nano-ai-computing/

[14] "Logitech webcam photo." [Online]. Available: https://doctorhead.ru/product/logitech_webcam_c920e_black/

[15] "Ultrasonic sensor photo," accessed on August. 24, 2024. [Online]. Available: https://developer.nvidia.com/blog/jetson-nano-ai-computing/

[16] "Wheel car body," accessed on August. 24, 2024. [Online]. Available: https://www.amazon.es/Wireless-Off-Road-High-Speed-Shock-Absorbing-Children/dp/B0BF9LV1P6

[17] "Chain car body," accessed on August. 24, 2024. [Online]. Available: https://tinyurl.com/ayufrsx7

[18] "Servo motor photo," accessed on August. 24, 2024. [Online]. Available: https://www.corona-rc.com/product/60.html

[19] "Dc-motor photo," accessed on August. 24, 2024. [Online]. Available: https://tinyurl.com/bdza79pa

[20] S. Shahjahan, "L293n motor driver photo," 2022, accessed on Nov. 28, 2024. [Online]. Available: https://www.circuits-diy.com/h-bridge-motor-driver-circuit/

[21] F. Nawazi, "L293d motor driver photo," 2022, accessed on Nov. 28, 2024. [Online]. Available: https://www.circuits-diy.com/h-bridge-motor-driver-circuit-l293d-2/

[22] R. Kundu, "Yolo: Algorithm for object detection explained [+examples]," 2023, accessed on August. 21, 2024. [Online]. Available: https://www.v7labs.com/blog/yolo-object-detection

[23] A. Mirkhan, "Yolo algorithm: Real-time object detection from a to z," accessed on August. 21, 2024. [Online]. Available: https://kili-technology.com/data-labeling/machine-learning/yolo-algorithm-real-time-object-detection-from-a-to-z

[24] A. Aggarwal, "Yolo photo," 2020, accessed on May. 25, 2024. [Online]. Available: https://medium.com/analytics-vidhya/yolo-explained-5b6f4564f31

[25] R. Alake, "Opencv tutorial: Unlock the power of visual data processing," 2024, accessed on August. 24, 2024. [Online]. Available: https://www.datacamp.com/tutorial/opencv-tutorial

[26] nathancy, "Extract building edges from map image using python," 2019, accessed on may. 24, 2024. [Online]. Available: https://stackoverflow.com/questions/56736043/extract-building-edges-from-map-image-using-python

[27] K. U. S. K. Somasundaram, B. Vinoth Kumar, *Artificial Intelligence for Sustainable Applications*. John Wiley Sons, 2023.