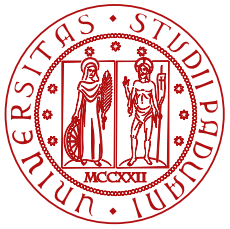


Università degli Studi di Padova

Dipartimento di Ingegneria dell'Informazione

Master Thesis in ICT for Internet & Multimedia

Embarking on the Autonomous Journey: A Strikingly Engineered Car Control System Design

Master Candidate

Kristel Çoçoli

Supervisor

Leonardo Badia

Date : 10 October

Academic Year

2022/2023

I dedicate this to my most inspirational professor, Leonardo,
to my love, Henri,
to my strength, my mommy,
to my family, Enxhi, Alisa, Oriana.
To the infinity and beyond.

Abstract

The evolution of autonomous cars represents a significant stride in modern transportation, combining innovation with sustainability. Central to this thesis is the development of an autonomous car control system with the Raspberry Pi as its primary processing unit. Integrated into this system are two predictive frameworks: a Convolutional Neural Network (CNN) for lane detection, and a traffic signs detector. Within this construct, the Raspberry Pi module acts as the center for a single car.

To cater to its immediate decision-making needs, a Proportional Integral Derivative controller and web server were instituted to process data through the trained neural networks. To enhance it visually, Unity and the Meta Quest 2 VR headset have been used as a testing method.

This study not only illuminates the engineering dynamics behind autonomous vehicles but also underscores the potential of augmented reality in automotive testing.

ITALIANO: L'evoluzione dei veicoli autonomi rappresenta un significativo passo avanti nel trasporto moderno, combinando l'innovazione con la sostenibilità. Al centro di questa tesi vi è lo sviluppo di un sistema di controllo di auto autonome con il Raspberry Pi come unità di elaborazione principale. In questo sistema sono integrati due framework predittivi: una CNN per il rilevamento delle corsie e un rilevatore di segnali stradali. All'interno di questa struttura, il modulo Raspberry Pi agisce come centro per un'unica auto.

Per soddisfare le sue immediate esigenze decisionali, è stato istituito un controllore Proportional Integral Derivative (PID) e un server web per elaborare i dati attraverso le reti neurali addestrate. Per migliorarne l'aspetto visivo, sono stati utilizzati Unity e il visore per la realtà virtuale Meta Quest 2, come strumenti per il testing.

Questo studio non solo esplora le dinamiche ingegneristiche delle auto autonome, ma sottolinea anche il potenziale della realtà aumentata nei test automobilistici.

Contents

List of Figures

List of Tables

List of Algorithms

List of Code Snippets

List of Acronyms

ML Machine Learning

CNN Convolutional Neural Network

IMU Internal Measurement Unit

VR Virtual Reality

AR Augmented Reality

SAE Society of Automotive Engineers

LiDAR Light Detection and Ranging

IoT Internet of Things

UUID Universal Unique Identifier

GPS Global Positioning System

ODD Operational Design Domains

I2C Inter-Integrated Circuit

YUV (Y) luma, or brightness, (U) blue projection, (V) red projection

HSV Hue, Saturation, Value

ROI Region of Interest

JSON JavaScript Object Notation

CSV Comma-Separated Values

TCP Transmission Control Protocol

PID Proportional Integral Derivative

ReLU Rectified Linear Unit

GPIO General-purpose Input/Output

CSI Camera Serial Interface

DC Direct Current

SoC System on a Chip

PoE Power-over-Ethernet

DSI Display Serial Interface

LAN Local Area Network

PCI Peripheral Component Interconnect

ELU Exponential Linear Unit

ReLU Rectified Linear Unit

API Application Programming Interface

AI Artificial Intelligence

1 Introduction

1.1 Frame of reference

1.1.1 Autonomous Vehicles

The 6 levels of Vehicle Autonomy

Autonomous cars are typically classified into different levels based on their capability to perform driving tasks without human intervention. These levels are defined by the Society of Automotive Engineers (SAE) in their J3016 standard.[1]

Level 0 (No Automation): At this level, the vehicle is entirely controlled by a human driver, and there is no automation involved. Some basic driver assistance systems may be present, such as warnings or momentary intervention, but the overall control remains with the human driver.

Level 1 (Driver Assistance): Level 1 vehicles have systems that can assist the driver with either steering or acceleration/deceleration tasks but not both simultaneously. Examples include adaptive cruise control or lane-keeping assistance.

Level 2 (Partial Automation): Level 2 vehicles have combined automated functions for both steering and acceleration/deceleration simultaneously. [2] However, the human driver must remain engaged and monitor the driving environment at all times. Tesla's Autopilot and some advanced driver assistance systems fall into this category.

Level 3 (Conditional Automation): At this level, the vehicle can handle all aspects of driving in specific conditions or scenarios, such as highway driving,

without human intervention. The human driver must be available to take over when the system encounters situations it cannot handle. [3] The handover time for the driver to regain control is critical.

Level 4 (High Automation): Level 4 vehicles are fully autonomous within specific Operational Design Domains (ODD). They can operate without human intervention in predefined conditions and locations, such as within a geofenced urban area. However, outside of these predefined conditions, human intervention may be required.

Level 5 (Full Automation): Level 5 vehicles are fully autonomous in all driving conditions and environments. They do not require human intervention and can operate at the same level as a human driver in any situation.

It is important to note that the development and deployment of fully autonomous Level 5 vehicles were still ongoing, and the technology's widespread adoption raised various legal, ethical, and technical challenges that required resolution. [4] Therefore, advancements in autonomous driving technology may have occurred since my last update, and the levels might have evolved.

1.1.2 Machine Learning Methods

Machine Learning (ML) is a critical subfield of artificial intelligence that focuses on the development of algorithms and statistical models enabling computer systems to learn from data and improve their performance on specific tasks. [5] Various ML paradigms exist, each catering to distinct learning scenarios and data types. This chapter provides an in-depth exploration of four prominent machine learning paradigms: supervised learning, unsupervised learning, semi-supervised learning, and reinforcement learning. [6]

Supervised Learning

Supervised learning is one of the most widely employed machine learning paradigms, characterized by the presence of labelled training data. In this approach, the algorithm learns to map input data to corresponding output labels by generalizing patterns from the training set. The primary objective is to build a predictive model that can accurately predict outputs for unseen input data. Supervised learning involves two key components: [7]

- Training data: Consist of input-output pairs, where the algorithm learns the mapping between inputs and their associated labels.

- Loss function: Measures the discrepancy between the predicted outputs and the actual labels, guiding the model to minimize prediction errors during training.

Unsupervised Learning

Unsupervised learning deals with datasets lacking labelled output data, requiring the algorithm to identify patterns, structures, and relationships solely based on the input data. [8] The objective is to discover hidden patterns and group similar data points into clusters without explicit guidance. Key characteristics of unsupervised learning:

- Lack of labels: Unsupervised learning operates on data without any predefined output labels, making it suitable for exploratory data analysis and pattern discovery.

- Clustering: A common unsupervised learning task, clustering aims to group similar data points together, revealing underlying structures within the data. [9]

Semi-Supervised Learning

Semi-supervised learning represents a hybrid approach that leverages both labelled and unlabeled data for model training. [10] In many real-world scenarios, obtaining fully labelled datasets can be expensive and time-consuming. Semi-supervised learning addresses this challenge by utilizing the abundance of unlabeled data, combined with a limited amount of labeled data, to improve the model's performance. Key aspects of semi-supervised learning:

- Utilizing unlabeled data: Unlabeled data aids in capturing more comprehensive data distributions and enhancing model generalization.

- Label propagation: Semi-supervised learning methods propagate [11]

Reinforcement Learning

Reinforcement learning differs significantly from the aforementioned paradigms, as it revolves around training agents to make sequential decisions in an environment to maximize a cumulative reward. [12] The agent interacts with the environment, learning by trial and error through exploration and exploitation. Key elements of reinforcement learning:

- Agent-Environment Interaction: The agent takes actions based on its policy, and the environment responds with rewards and new states.
- Policy: A strategy determining the agent's actions in specific states to maximize the expected cumulative reward.
- Exploration vs. Exploitation: The agent must balance between exploring new actions and exploiting its current knowledge to optimize long-term rewards. [13]

1.1.3 Convolutional Neural Networks for Self-Driving Car Systems

In recent years, the field of self-driving cars has seen tremendous advancements owing to breakthroughs in artificial intelligence and machine learning. One of the fundamental technologies driving these developments is Convolutional Neural Networks (CNN)s. [14] CNNs have revolutionized various computer vision tasks, including object recognition, image segmentation, and localization. In the context of self-driving cars, CNNs play a pivotal role in understanding the surrounding environment, detecting obstacles, recognizing traffic signs, and making critical decisions in real time.

Convolutional Neural Networks

Convolutional Neural Networks are a class of deep learning models inspired by the human visual system. They are designed to process and analyze visual data, such as images and videos, by effectively capturing hierarchical patterns and features. The core building blocks of a CNN are convolutional layers, pooling layers, and fully connected layers. The distinguishing feature of CNNs is their ability to learn spatial hierarchies of patterns, [15] making them highly suitable for tasks requiring local spatial understanding, like object detection and localization.

CNN Architecture for Self-Driving Car Systems

In the context of self-driving car systems, CNNs are typically employed as perception modules. The perception module is responsible for analyzing the data from various sensors, such as cameras, Light Detection and Ranging (LiDAR), and radar, to create a comprehensive understanding of the environment. [16] Convolutional Neural Networks process the visual input, i.e., camera images, and extract relevant information for decision-making processes.

Benefits of CNNs for Self-Driving Car Systems

- Feature Extraction: CNNs excel at automatically learning relevant features from raw pixel data, reducing the need for manual feature engineering, which can be a time-consuming process. [17]
- Spatial Understanding: CNNs inherently capture spatial hierarchies in data, making them suitable for tasks that require understanding local structures, such as detecting lane boundaries and other vehicles.
- Real-time Performance: CNNs can be optimized to achieve real-time performance, allowing self-driving car systems to make instantaneous decisions and react to dynamic driving environments.
- Adaptability: The flexibility of CNN architectures allows them to adapt to different driving scenarios, weather conditions, and road layouts. [18]

1.2 Our Proposal

In this research-driven project, we introduce an advanced autonomous vehicle system tailored to enhance road safety and navigation efficiency. The major components and methodologies of our system are enumerated below:

- **Machine Learning Integration:**
  - **Lane Detection Model:** A sophisticated ML model has been employed to accurately recognize lanes, ensuring that the car remains correctly aligned with the roadways.
  - **Traffic Sign Detection Model:** An additional ML model is incorporated to reliably detect and interpret traffic signs, allowing the vehicle to respond in real-time to changing road conditions.

- **Sensor Fusion:** By combining data from various onboard sensors, I utilize sensor fusion algorithms. [19] This not only refines the computation of the steering angle but also ensures a smoother vehicular motion, minimizing abrupt movements and turns.
- **GPS Integration:** The inclusion of a Global Positioning System (GPS) module serves a dual purpose. It facilitates real-time tracking of the vehicle's geolocation and assists in efficient route planning, positioning the car accurately on digital maps.
- **Augmented Reality Interface:** As an innovative interface solution, I have enabled visualization of the autonomous car's journey via the Oculus Quest headset. This provides a unique and immersive experience, allowing observers to gain a first-person perspective of the vehicle's behaviour.
- **Physical Prototype:** I have built a car prototype leveraging the computational capabilities of the Raspberry Pi. This project involved detailed planning and assembly of key components essential for the vehicle's operation. Central to these components are the IMU and the GPS modules. By effectively interfacing these modules with the Raspberry Pi, I have been able to achieve a functional prototype with integrated navigation and tracking capabilities.

The rest of this thesis is organized as follows: In Chapter 2, I have reviewed the existing literature and studies pertinent to this subject, setting the foundational context for subsequent discussions. Chapter 3 delves into the specific hardware components utilized and their specifications, configurations, and the logic behind their selection. In Chapter 4, it is presented the design and structure of the software components, illustrating the hierarchical design and how the software complements the hardware. In Chapter 5, I discuss the methodologies applied for data acquisition and preparation. Chapter 6 introduces the underlying deep learning frameworks, elucidating the architectural choices, training paradigms, and outcomes. Chapter 7 emphasizes the implementation of GPS for pinpoint vehicle localization and the challenges therein. In the following chapter, I have explained the system's control strategies, ensuring seamless operation and functionality. The last chapter focuses on the integration of a Virtual Reality (VR) headset, justifying its implementation and significance within the system.

2

Related Work

The development and advancement of autonomous car systems have been a pivotal focal point of modern engineering. This emphasis has been driven by the desire to transform our transportation systems into a safer, more efficient paradigm. As such, numerous research endeavours have taken centre stage, each exploring a number of techniques to enhance system safety, performance, and functionality. This chapter delves into prominent works that have addressed different aspects of this domain, from the intricacies of sensor integration to the nuanced mechanics of lane and object detection. One of the oldest papers regarding this topic, dating back to 1984 [20], presents a groundbreaking perspective. Here, a new modular driver information system is proposed, offering a glimpse into the early visions of autonomous systems. This design addressed the challenge of hefty initial investments in road and railway infrastructures by employing innovative two-way transmitters and receivers. As we trace the historical timeline, it's intriguing to observe how the thought paradigms from decades past align with contemporary perspectives, sometimes eerily predicting the trajectories we see unfolding today.

2.1 Sensor Integration in Autonomous Driving Systems

Ensuring the robustness and safety of autonomous vehicles poses challenges, primarily because of the amalgamation of various sensors and computational

components sourced from different manufacturers. An insightful approach to this complexity is discussed in previous research [21]. It introduces methodologies to determine the optimal blend of sensors, methods, and algorithms. The paper explores the intricate system of sensors used by autonomous vehicles to perceive their surroundings. Leveraging artificial intelligence, these vehicles analyze vast amounts of data from these sensors to make decisions mirroring human-like driving behaviour. Due to the data's sheer volume, advanced machine learning and data-driven approaches become pivotal. Handling such extensive data, especially onboard, poses challenges. The autonomous vehicle sector lacks standardized practices but employs a known set of sensors including cameras, radar, and LIDAR, which feed data to a central processing unit. Given the complexity and risks associated with full-scale vehicles, model-sized self-driving vehicles offer a cost-effective and safer avenue for development. Such models, though scaled-down, are crucial for testing and simulation, ensuring that learning is consistent across different platforms. The paper details the creation of a model autonomous vehicle equipped with various sensors, intending to enhance its autonomous capabilities through collected data.

The use of sensor fusion algorithms, explored in numerous papers [22] [23], improves output accuracy, ensuring safer and more precise driving. These works analyse experimentally, the impact on the speed of calculation and precision that Kalman, Madgwick and Mahony algorithms offer.

Another study [24] introduces a sensor fusion algorithm that combines data from Inertial Measurement Units (IMU) and GPS to predict the vehicle's next position and orientation more accurately. While traditional navigation methods predominantly rely on GPS alone, which doesn't comprehensively scan the environment, the fusion with IMU data addresses these shortcomings, striving for greater accuracy, reliability, and overall safety in autonomous navigation.

2.2 Object Detection and Behavioral Impacts

Recognizing on-road objects, whether they are pedestrians, other vehicles, or traffic signs, is imperative for safe and responsive autonomous driving. Several papers [25] [26] dive deep into this domain, elaborating on techniques and algorithms optimized for reliable object detection.

This study [27] underscores the criticality of ensuring safety, stability, and in-

telligence in high-speed autonomous driving scenarios. One pivotal aspect is the early detection and interpretation of pedestrian behaviours, especially in highway conditions, enabling the self-driving system to react correctly. However, there is a noted lack of datasets geared towards behaviour recognition in such scenarios. Addressing this gap, the paper introduces the THU-IntrudBehavior dataset, a comprehensive collection of pedestrian lane intrusion behaviours relevant to real-world highway situations. The dataset captures a broad spectrum of pedestrian and cyclist behaviours across various urban settings and weather conditions. Alongside a detailed annotation for each video entry, the paper also presents baseline experimental results using this dataset. Ultimately, the THU-IntrudBehavior dataset offers valuable resources for enhancing behaviour recognition in high-speed autonomous driving environments. The consequential behaviours of the vehicle, post object detection, are contingent on these research outcomes.

Another study [28], conducted on a Raspberry Pi, focuses on obstacle detection, which impacts the driving process. This study zeroes in on the implementation of obstacle detection and avoidance for autonomous cars using the Convolutional Neural Network (CNN) for real-time video/image analysis via an IoT device. A Raspberry Pi is utilized for vehicular control and on-the-fly inference through the CNN based on instantaneous inputs. The developed model showcases a commendable accuracy rate of 88.6 %, aligning well with the anticipated performance benchmarks.

Some other studies use the principles of game theory to describe the behaviour of the vehicle when encountering obstacles or moving objects. In a previous study [29], it is evaluated an emergency scenario where two autonomous vehicles encounter a randomly behaving road obstacle. Game theory is applied, initially considering complete information about the obstacle and then incorporating incomplete information to formulate a Bayesian game. The autonomous vehicles can either stay in their lane, swerve, or change lanes, leading to various outcomes like driving unobstructed, colliding with the obstacle, or hitting another vehicle. The research investigates the Nash equilibria and explores if one vehicle's knowledge about the obstacle benefits other road users, a crucial consideration for connected vehicles.

2.3 Visualization and AR Integration

Venturing into the realm of augmented reality integration for visualization purposes presents a unique challenge, especially when interfacing with computing modules like Raspberry Pi. While literature in this niche area is scant, this thesis showcases an innovative approach by bridging the Raspberry Pi car system with a VR headset. This integration compensates for the inability to execute real-car tests and provides a simulated environment for testing.

2.4 Challenges in Autonomous Driving Systems

Traffic safety concerns regarding interactions between pedestrians, cyclists, and human-driven vehicles have grown over time. The advent of autonomous vehicles introduces new challenges, questioning if they can prevent accidents while ensuring harmonious road-sharing with human-driven cars and pedestrians. A previous study [30] presents game-theoretical models that examine the strategic interplay between pedestrians and autonomous vehicles. Simulations support the theoretical findings, providing insights into potential urban traffic regulations and communication systems needed to enhance traffic management with the inclusion of autonomous vehicles.

Another previous research [31] provides a comprehensive understanding of the early challenges faced in the realm of autonomous driving. A notable emphasis is laid on the quality of sensor data and its implications on the reliability of self-driving algorithms, providing a comprehensive review of the advancements and challenges in autonomous car technology. It charts the journey of the automobile industry, emphasizing its impressive achievements over the last century and noting how modern computation and communication breakthroughs are propelling the shift towards autonomous vehicles. The paper acknowledges the significant testing milestones and vast investments made by tech giants and car manufacturers in this domain. Despite the advancements, there are recognized challenges on both the technical front, like software design and real-time data processing, and the nontechnical side, such as public acceptance and ethical considerations. The study discusses these issues in depth, emphasizing the need for solutions that meet the demands of consumers, the industry, and regulatory bodies. The paper also sheds light on potential applications of autonomous cars

beneficial to consumers and other sectors. The discussion concludes by presenting the challenges that need attention and offers constructive recommendations to various stakeholders involved in the development and deployment of autonomous cars.

Other studies [32] [33], which are conducted using a Raspberry Pi, focus on the calculation of the steering angle while driving in the lane. This work-in-progress paper [33] introduces "PyRoboCar", an affordable, neural network-driven autonomous car initiative. PyRoboCar, functioning on a smaller scale, mimics real self-driving car systems by utilizing a deep CNN that processes images from a front-facing fisheye camera to determine steering directions. The network structure parallels that of commercial autonomous vehicles. For real-time operation, it integrates a camera, an auxiliary Tensor Processing Unit, and a Raspberry Pi 4 platform.

Additional works [34] emphasize that a "zero-accident car" is one of the primary objectives of researching autonomous driving. As we progress towards the vision of a "zero-accident car", the focus shifts to autonomous driving, enabled by intricate networks of advanced sensors, possibly even smart tires, and sophisticated control algorithms processed on decentralized computing platforms. Central to this evolution is the ongoing development of the car's sensor component. The paper discusses the challenges in designing and manufacturing these sensors for autonomous vehicles. Despite technological advancements supporting this vision, considerable integration and algorithmic challenges still need to be addressed.

3

Hardware Setup

3.1 Introduction

The system I implemented consists of two main components: autonomous cars and a centralized processing unit, which operates as a cloud-based Internet of Things (IoT) system/server. The purpose of this system is to enable autonomous driving by sending commands from the server to the individual cars.

Figure 3.2 illustrates the high-level communication flow. The cars communicate with the server using an event bus, which requires an internet connection for data exchange. The event bus allows the cars and the server to communicate without directly knowing the exact number of cars or servers involved, promoting a flexible and scalable system.

To enable data collection, the cars are equipped with sensors that gather relevant information. These sensors need to be appropriately configured, including setting parameters such as unit of measurement, sensitivity, and power supply management. Communication with the sensors is established using the Inter-Integrated Circuit (I2C) protocol [35], and the collected data is then formatted in a readable manner, using Protobuf, before being published to the event bus.

On the server side, it receives the data from the event bus and interprets it to generate appropriate commands for each car. These commands are then sent back to the cars via the event bus to implement the control system and enable autonomous driving.

For the implementation of all these functionalities, I have chosen the Rasp-

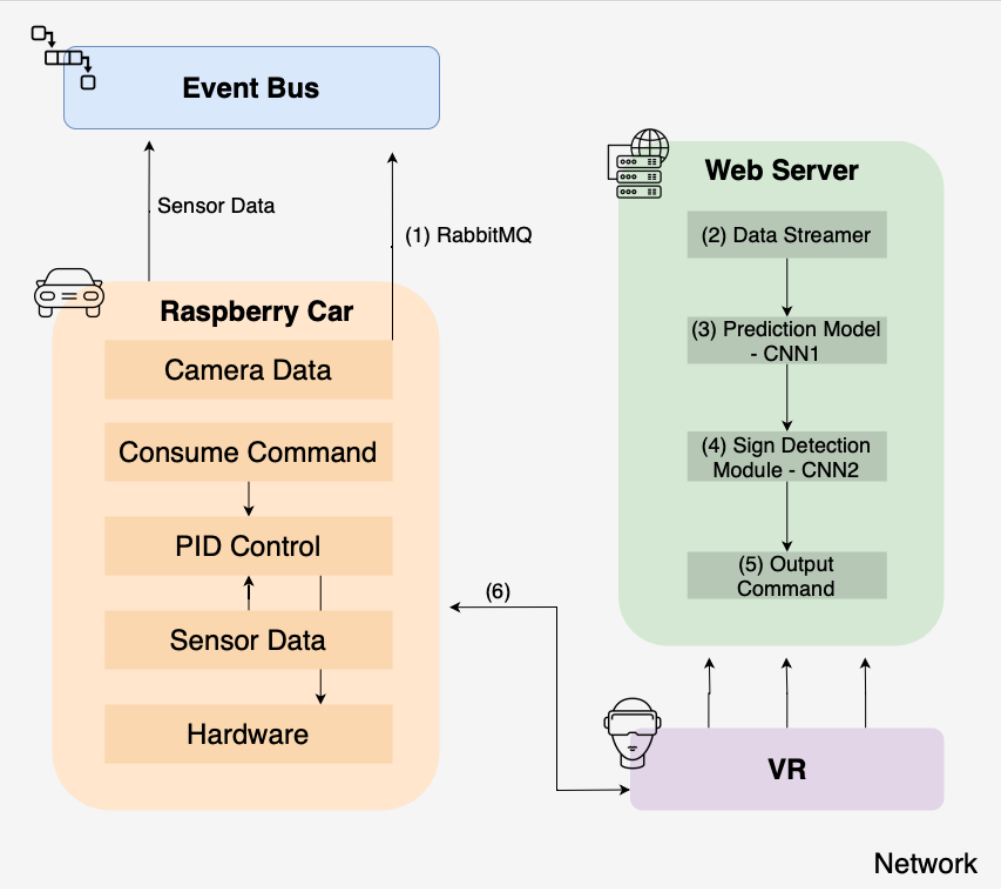


Figure 3.1: Car Control System on High Level

berry Pi 3B+ as the remote processing unit. [36] The decision to use Raspberry Pi 3B+ was motivated by its powerful processing capabilities, energy efficiency, and compact size, making it suitable for deployment in remote systems like the cars in this setup.

By employing this cloud-based IoT system and Raspberry Pi 3B+ as the processing unit, the goal is to achieve autonomous driving for cars while maintaining a scalable and adaptable architecture. I have created a car model, to test the system.

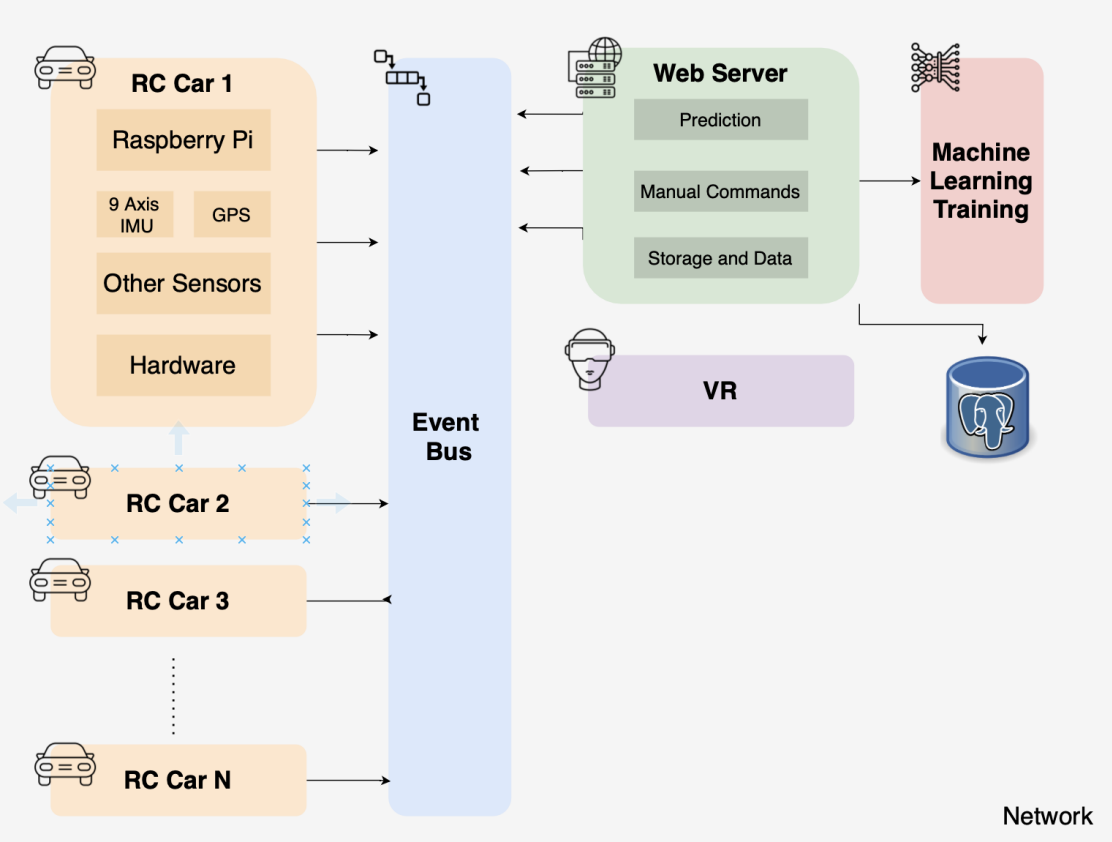


Figure 3.2: Car Control System on Low Level

3.2 The car components

3.2.1 Raspberry Pi

The Raspberry Pi 3 Model B+ represents the final version of the Raspberry Pi 3 series and is of particular relevance for this study. It features a Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit System on a Chip (System on a Chip (SoC)) running at 1.4GHz. The device is equipped with 1GB LPDDR2 SDRAM, offering sufficient memory capacity for processing tasks. For wireless communication, it supports both 2.4 GHz and 5 GHz IEEE 802.11.b/g/n/ac wireless Local Area Network (LAN), as well as Bluetooth 4.2 and BLE, providing versatile connectivity options. In terms of connectivity, the Raspberry Pi 3 Model B+ includes Gigabit Ethernet over USB 2.0, with a maximum throughput of 300 Mbps. It offers extensive input/output capabilities through an extended 40-pin General-purpose Input/Output (GPIO) header, 4 USB 2.0 ports, a Camera Serial Interface (CSI) camera port, and a Display Serial Interface (DSI) display

port. Moreover, it provides a full-size HDMI® port, a 4-pole stereo output, and a composite video port for multimedia purposes. To load the operating system and store data, the device features a Micro SD port. Power is supplied via a 5V/2.5A Direct Current (DC) power input, and Power-over-Ethernet (PoE) support is available. The Raspberry Pi 3 Model B+ offers a compact and capable platform for this study.

3.2.2 The motors

I have used four brushed motors, to drive the wheels and enable movement of the car. They have a straight-forward design, consisting of a rotating armature (the rotor) and a fixed set of magnets (the stator), with brushes that come into contact with the commutator to control the current direction in the motor

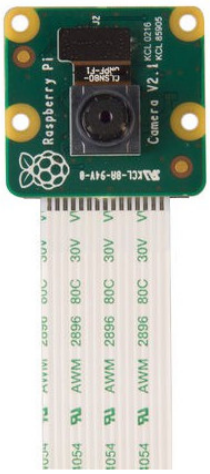
To power the motors, I have used lithium batteries with an external charging system. These batteries provide a 5V DC output with a capacity of 3000 mAh (battery capacitor) and can handle a discharge current of up to 20 A. This high discharge current is necessary to accommodate the initial step response, overshoot, and subsequent stabilization (around 1-2 A) that occurs when the entire system is initiated.

3.2.3 The camera module

Another reason behind the selection of the Raspberry Pi is its built-in Peripheral Component Interconnect (PCI) socket designed specifically for the camera module. The camera utilized in this setup, shown in figure 3.3a boasts a 30 frames per second capability and offers a resolution of 1080p. These camera parameters surpass the requirements for the machine learning algorithm since the photos' resolution will be downscaled regardless.[37]

3.2.4 The ultrasonic sensor

While the car primarily follows commands from a machine learning-based predictive model, in certain critical situations, hardcoded solutions take precedence. When reading sensor data, it is important to evaluate an efficient Age of Information and Value of Information management.[38] The ultrasonic sensor is the sensor of interest, and the information it provides should be prioritised. One such scenario is when the car detects an object within 5 centimetres of its



(a) Raspberry Camera

(b) Ultrasonic Sensor

Figure 3.3: Camera and Ultrasonic Sensor

proximity, triggering an immediate stop command. This safety measure relies on an ultrasonic sensor, which emits sound waves to measure distances from nearby objects. The sensor's real-time data ensures collision avoidance. Additionally, it enhances the accuracy of the predictive model by providing valuable feedback for diverse scenarios. The integration of the ultrasonic sensor ensures a balanced and adaptive autonomous driving experience.[39]

I have used a HC-SR04 ultrasonic sensor. It operates at a voltage of +5 V and is designed to measure distances theoretically ranging from 2 cm to 450 cm. In practical terms, its effective measuring distance spans from 2cm to 80cm, with an impressive accuracy level of 3 mm. The sensor covers a measuring angle of less than 15°, making it suitable for precise distance measurements in a specific direction. The sensor operates based on the formula: Distance = Speed × Time. It works by transmitting an ultrasonic wave, which travels through the air and reflects back when it encounters an object. The Ultrasonic receiver module detects this reflected wave. To calculate the distance, we need to know the speed and time. As we are using ultrasonic waves, the universal speed at room conditions is 330 m/s. The module's circuitry calculates the time taken for the wave to return and activates its echo pin accordingly. By utilizing the Raspberry Pi, the distance can be easily calculated.

3.2.5 The GPS module

For this setup, I have employed a GPS module from Adafruit, a provider of quality electronic components. To ensure reliable performance both indoors and outdoors, I equipped our board with a small antenna. However, for optimal indoor functionality, it became necessary to integrate a signal amplifier into the system.

Signal amplifiers come in two types: active and passive. After careful consideration, I decided to implement active amplifiers from Adafruit. Active amplifiers provide enhanced signal boosting capabilities, which is essential for maintaining a strong and stable GPS signal even in challenging indoor environments.[40] [41]

Now I will dive into a brief explanation of how a GPS module works:

GPS, or Global Positioning System, relies on a network of satellites orbiting the Earth. These satellites constantly transmit signals that are received by GPS receivers, like the one from Adafruit. The GPS receiver calculates its position based on the time it takes for the signals to travel from the satellites to the receiver. [42] By triangulating signals from multiple satellites, the GPS module determines the user's precise latitude, longitude, and altitude.[43]

In our case, the Adafruit GPS module (3.4) processes the signals received through the small antenna and amplified by the active signal amplifier. The amplified signals provide a stronger and more accurate data feed to the GPS module, enabling it to perform effectively both indoors and outdoors.

The integration of the Adafruit GPS module, along with the active signal amplifier, ensures that our setup can confidently provide precise and reliable location data, regardless of whether it operates inside buildings or in open outdoor spaces.

3.2.6 The IMU

To ensure a robust and stable control system, relying solely on commands from the server is insufficient; an IMU within the car is imperative to achieve a smooth and coherent response to these commands. If an IMU is not used, the car's response to commands from the server would be less precise and potentially erratic.

For effective control system implementation, real-world measurements of the

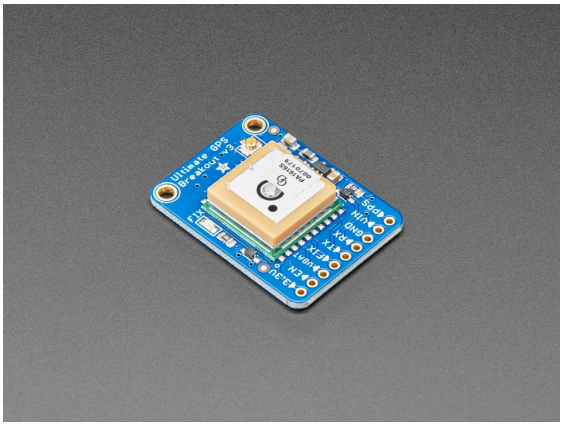


Figure 3.4: GPS Module
s

car's state become essential. For instance, when aiming to rotate the car by a specific angle, such as 30 degrees from its current orientation, a PID controller algorithm [44] (discussed in Chapter 8) is employed in the negative feedback loop. This controller necessitates two inputs: the command from the server and the car's current state. Without an IMU, these issues would probably occur:

- Loss of Orientation: The absence of IMU data would make it challenging for the system to accurately determine the object's orientation in three-dimensional space. As a result, the vehicle may struggle to understand its current heading, roll, and pitch angles, leading to difficulties in navigation and maintaining stable movement. [45]

- Reduced Stability:IMUs play a crucial role in providing continuous measurements of linear accelerations and angular velocities. Without this data, the control system may lack the necessary feedback to maintain stability during manoeuvres, resulting in erratic movements or difficulty in responding to dynamic changes in the environment. [46]

- Limited Motion Monitoring: IMUs continuously track an object's motion, providing valuable data on position, velocity, and acceleration. Without this input, the control system would have to rely on other sensors or estimation techniques, potentially leading to inaccuracies and reduced precision in motion monitoring. [45]

- Impaired Control: In autonomous vehicles or robotics applications, precise and real-time information about motion and orientation is essential for making informed decisions. Without an IMU, the control system may not be able to respond promptly to changing conditions, leading to suboptimal performance and potentially unsafe operation. [47]

- Compromised Navigation: IMUs are crucial for aiding in navigation, especially in GPS-denied environments or when GPS signals are temporarily

unavailable or inaccurate. Without an IMU, the vehicle may face challenges in accurately estimating its position and heading, affecting its ability to navigate effectively.

To acquire the current state of the car accurately, a 9-axis IMU is employed. This remarkable device measures the car's three-dimensional orientation, utilizing Euler angles or quaternions, through the readings of three distinct sensors: accelerometer, gyroscope, and magnetometer.

For this purpose, we have chosen the Adafruit 9-axis IMU 3.5, model TDK InvenSense ICM-20948 IMU[48], a reliable and high-performance unit capable of precisely determining the car's orientation in real-time. By incorporating this IMU into the control system, we can obtain the vital data required for seamless and responsive manoeuvring of the car by the server's commands, thus creating a harmonious and efficient overall response.

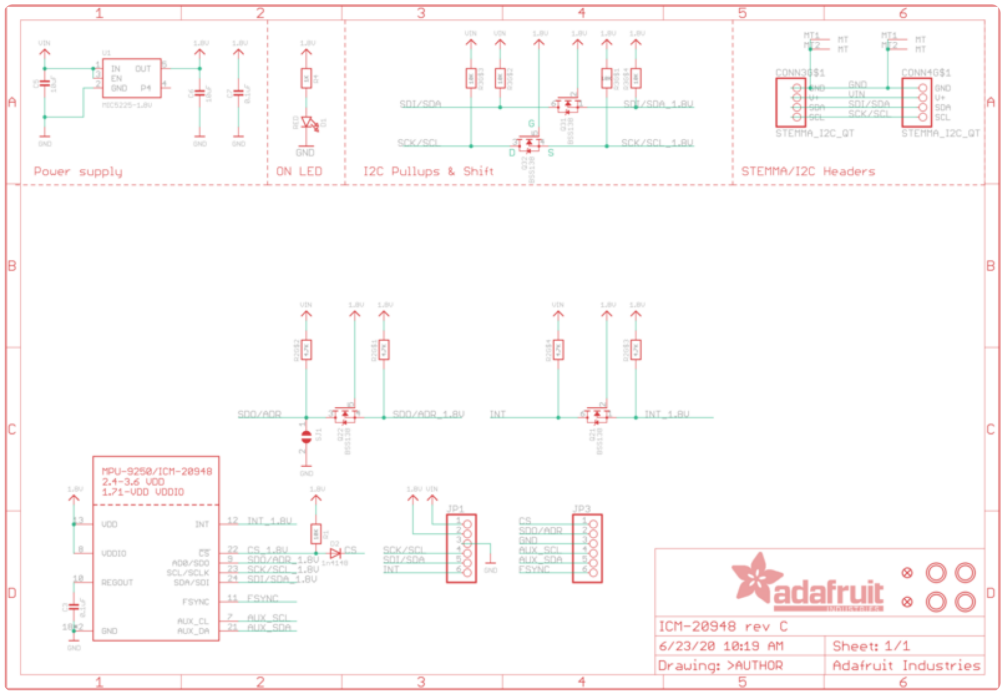


Figure 3.5: Adafruit IMU Schema

3.2.7 The assembled car

Including all the elements mentioned above, the assembled car is shown in figure 3.6. The goal is to create a system that will allow this, and every other car of this built, to drive autonomously and to be observed during this process.

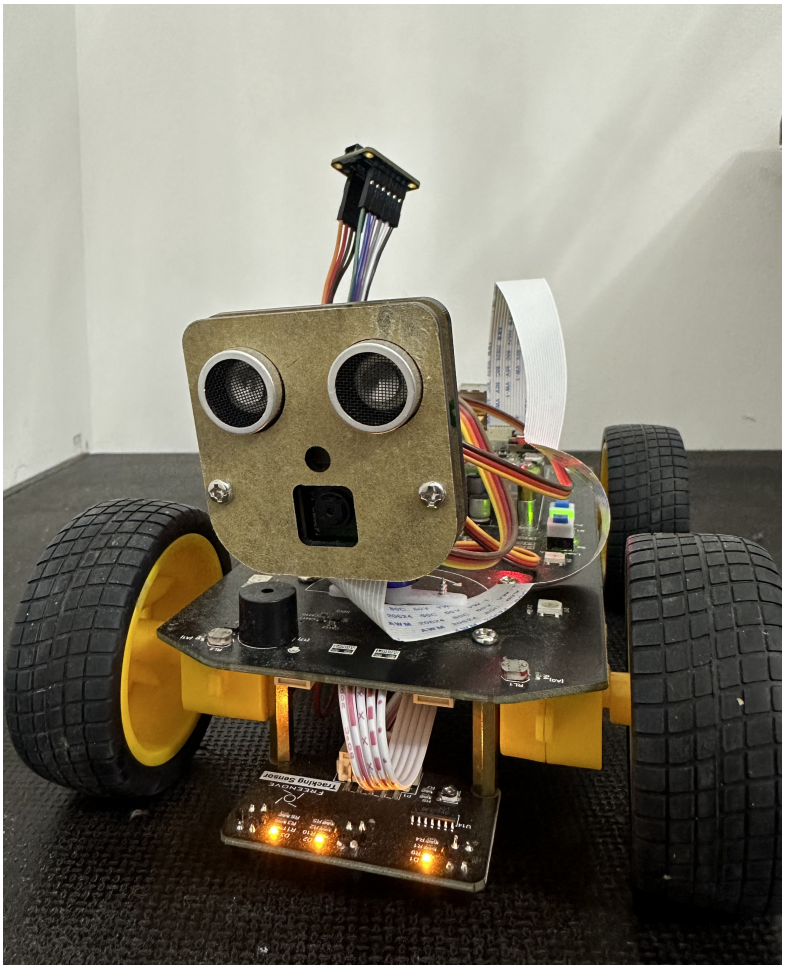


Figure 3.6: The Assembled Car

4

Software Architecture

Within the system architecture, shown in figure 4.1, the server does not know the source of incoming data and does not recognize the individual autonomous vehicles. It is a vehicular network [49] where the data transmission speed is a priority. All the communication is done via an Event Bus as mediator. Each autonomous vehicle is, theoretically, assigned a Universal Unique Identifier (UUID) during its software installation. The vehicle's onboard camera captures the frame, which is subsequently transmitted to the event bus as a byte array. [50] The server, connected to this event bus, then decodes the byte array in preparation for the processing steps that follow:

1. **Lane Detection:** The first processing stream pertains to lane detection. Based on the analysis, the system generates predictions related to the required acceleration and steering commands for the vehicle.[51]

2. **Traffic Sign Detection:** The second stream focuses on detecting traffic signs. The output of this processing is an array, detailing the identified traffic signs and their respective coordinates. Given the potential for multiple traffic signs within a single frame, this array format is essential. Notably, the predictions derived from traffic sign detection can supersede those from lane detection. For instance, if lane detection suggests a 20-degree turn at 40 km/h, but a "Stop" sign is detected, the vehicle is instructed to halt. [52]

These analyses leverage two distinct convolutional neural networks (CNNs) for processing.

The final system output consists of steering and acceleration directives along with the identification and coordinates of detected traffic objects.

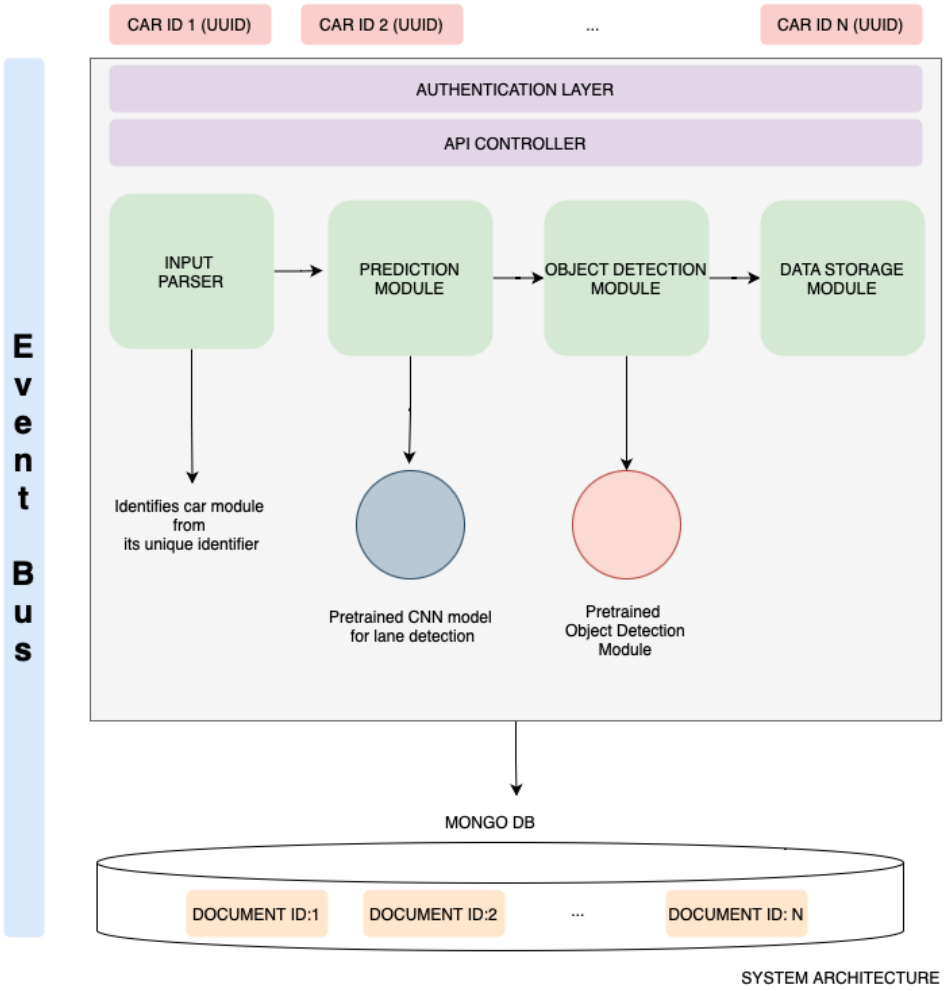


Figure 4.1: Software Architecture

4.1 System Design and Implementation

4.1.1 Technology Stack

For the server component of the system, I chose .NET 7[53] due to its enhanced performance features, which have been optimized with each new version. This makes it particularly suitable for high-throughput server applications integral to autonomous vehicle management. Additionally, the cross-platform compatibility that .NET 7 offers is indispensable, enabling deployment across diverse platforms such as Windows, Linux, and macOS. This flexibility ensures my server remains versatile across different deployment environments. Another key advantage of .NET 7 is its integrated asynchronous programming model.

The Task-based asynchronous pattern, integral to .NET, is adept at handling numerous simultaneous data streams from multiple vehicles without unduly taxing server resources.

For the data storage, I used MongoDB, primarily due to its document-centric schema which is flexible and highly performant. [54] This quality is important for my system since it means that as the structure of the vehicle data evolves, the database can readily adapt without extensive schema alterations.[55] MongoDB's design ensures each car is treated as a distinct document, facilitating rapid read operations and minimizing performance overhead when scaling to handle an influx of data entries.[56] Another significant aspect of MongoDB is its capability for horizontal scalability, made possible through sharding.[57] This ensures that the system remains adaptive to the growing demands of an expanding fleet of vehicles.

Lastly, for the event bus, RabbitMQ [58] was the technology of choice. Its reputation for memory efficiency in message queuing stands out, a quality that's crucial when handling data from a multitude of vehicles communicating in real time. Any system lag resulting from memory overheads can be detrimental, making RabbitMQ's efficiency a pivotal asset. Furthermore, RabbitMQ offers a robust framework for reliability, characterized by its message durability features. This ensures that even in the face of system disruptions, there's a minimal risk of losing crucial vehicle data. Designed with high availability in mind, RabbitMQ is equipped to handle a vast volume of simultaneous messages, a scenario likely in environments teeming with autonomous vehicles.

4.1.2 System Implementation

In the context of this autonomous car system, each vehicle is uniquely identified using a UUID. In figure 4.2, is depicted as an example, to illustrate the data flow. A car with the identifier "abc123" captures visual data from its camera. This data is then sequentially processed frame-by-frame, converting each frame into a JavaScript Object Notation (JSON) structure. Within this structure, the image data is represented as a byte array, and the desired prediction type is specified. I will be focusing on the CNN prediction.

Following the initial processing, the CNN model generates predictions detailing the steering angle and acceleration values for the vehicle. Subsequently, this frame undergoes analysis by a second model dedicated to traffic sign detec-

tion. This model identifies the traffic signs within the frame.

Based on the detected objects, the system formulates the final output, which includes directives for acceleration, steering angle, and information about the identified objects, their names and coordinates within the frame. It is worth noting that the presence of specific traffic signs can influence the final recommended acceleration and steering angle, adjusting them as per traffic regulations and safety considerations. After the output is formulated, it is sent back to the car through the RabbitMQ message broker, using its Pub/Sub communication. [59][60]

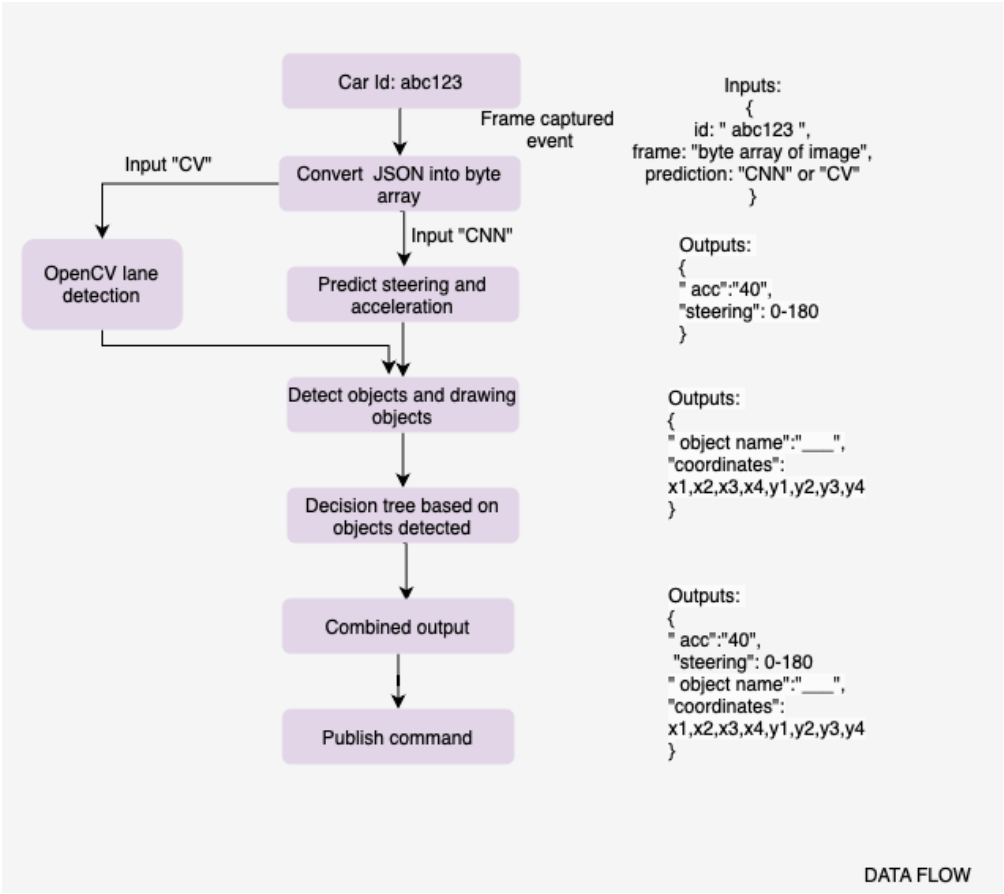


Figure 4.2: Data Flow

RabbitMQ Pub/Sub Communication

In my car control system, I leveraged RabbitMQ's publish-subscribe (pub-/sub) 4.3 pattern to enable real-time communication between the car and the control system. The pub/sub pattern in RabbitMQ works by sending messages

to exchanges, rather than specific queues. Subscribers express interest in specific types of messages, and RabbitMQ routes the messages to the appropriate queues for these subscribers.[58] The primary components involved are:

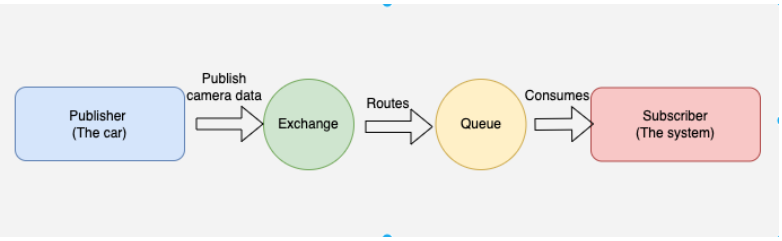
- Publisher: Send messages to an exchange.
- Exchange: Routes the messages to one or more queues based on the rules defined.
- Queues: Buffers that store messages until they are consumed.
- Consumers: Processes or tasks that consume messages from the queues.

Camera Frames Delivery

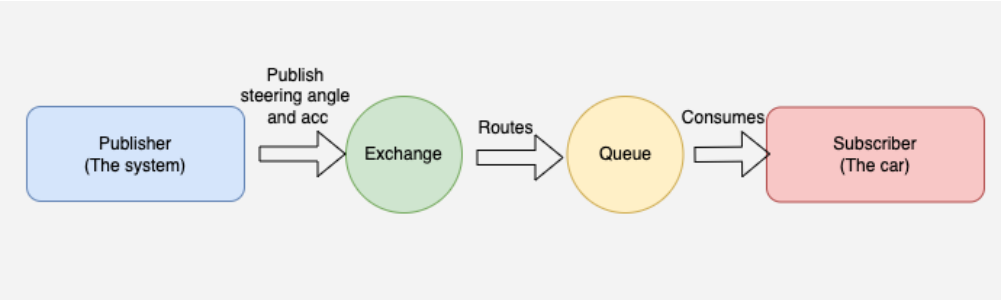
The autonomous car, acting as a publisher, captures frames from its onboard camera and publishes these frames to an exchange named "CameraFramesExchange". In this step, the system acts as a consumer since multiple subsystems (traffic signs detection and lane detection) are interested in these frames, they each subscribe to this exchange. RabbitMQ, based on the binding rules, routes these frames to relevant queues, ensuring that each subsystem gets the frame data for further processing.

Control Commands Delivery

The control system processes the received frames, determines the optimal steering angle and acceleration, and then publishes these commands to an exchange named "CarCommandsExchange". The car, acting as a consumer this time, has a dedicated queue where it waits for these command messages. Once the acceleration and steering angle arrive, it is consumed by the car's onboard computer to execute the respective driving action.



(a) Camera Frames Delivery



(b) Control Commands Delivery

Figure 4.3: RabbitMQ Pub/Sub Communication

5

Data Collection and Processing

5.1 Data Collection and Processing for Lane Detection model

5.1.1 Data Collection

To enable autonomous navigation of the self-driving vehicle, I need a relevant dataset. This dataset should ideally reflect the real-world conditions of the testing environment. To address this, I designed a basic manual driving setup using the WASD keys, allowing me to navigate along road lines while recording corresponding videos. Subsequently, I employed an OpenCV program to process these videos as input. The outcome of this process was the creation of a CSV file containing both the names of the video frames and their corresponding steering angles. This was achieved through the utilization of a lane detection algorithm[61] implemented in OpenCV.

The algorithm steps I have followed are:

1. **Preprocessing**: I converted the image to the Hue, Saturation, Value (HSV) space to focus on blue lanes. [62] This colour filtering ensures that the majority of the unwanted objects in the scene are excluded.
2. **Edge Detection**: I used the Canny edge detection method [63]. It identifies the boundaries of objects within images, crucial for detecting the outlines of lanes.
3. **Region of Interest (ROI)) Extraction**: To reduce computational load and improve accuracy, the focus is on the bottom half of the image.

4. **Line Detection**:The Hough transform identifies lines within the ROI. This method transforms image points into a Hough space, making line identification straightforward.[64]

5. **Line Averaging & Extrapolation**: Multiple detected lines are combined into two main lanes (left and right) by averaging their position and extrapolating their length.

6. **Steering Angle Calculation**: The deviation of the vehicle from the centre of the lane is used to compute a steering angle, ensuring the vehicle remains centred within its lane.

The results of one single frame are shown in figure 5.1. The green lines are the detected edges and the red line suggests the steering degree. I saved the frame names and steering degree in a Comma-Separated Values (CSV) file, and this data will be used to train the ML model.

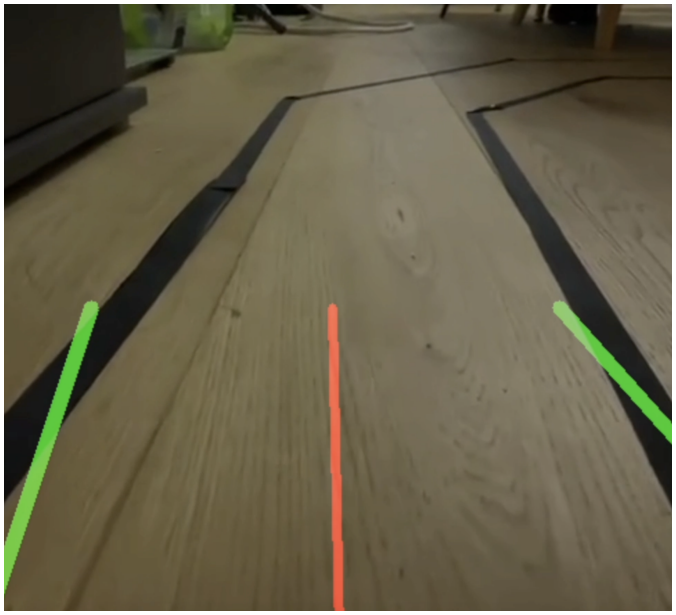


Figure 5.1: Lane Edge Detection

For the training, I used 224 images, and for the validation 65 images, as shown in figure 5.2. The angles are mostly smaller than 90 degrees because, in my training lane, the car turned mostly left. This data will be augmented with further processes like image flips, therefore it does not pose a problem in terms of training data quality.

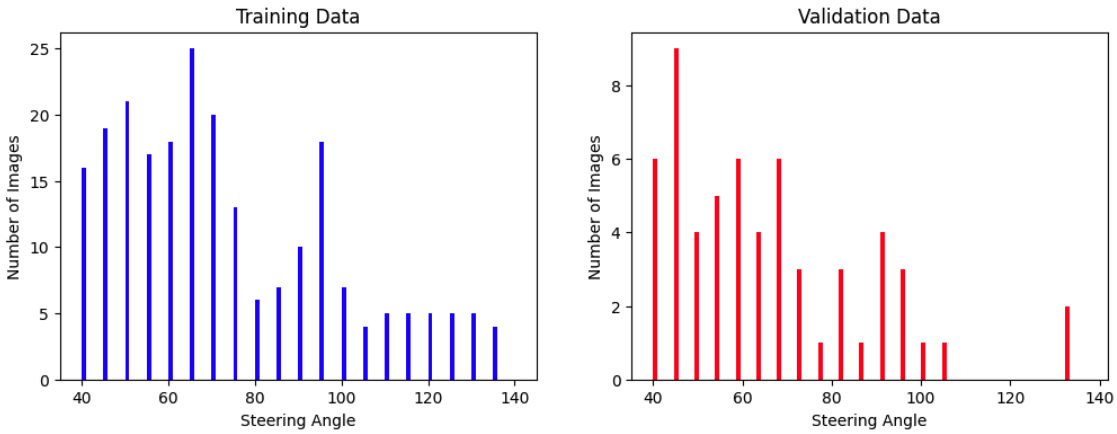


Figure 5.2: Training and Validation Dataset

5.1.2 Data Augmentation

Data processing is an integral part of machine learning and data analytics. To diversify and increase my training dataset, I tried augmenting my data. I have used multiple techniques:

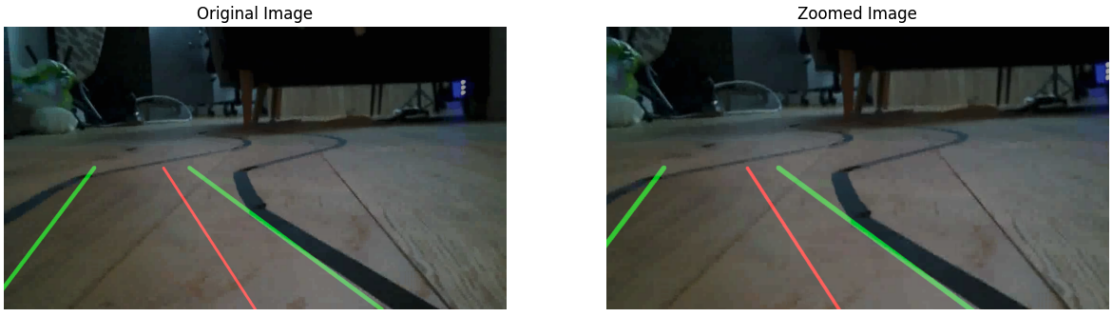
1. Zoom: It was performed by cropping a smaller portion from the centre of the original image. By doing so, I emphasized on the significant features of the image while discarding the peripheral content. 5.3a

2. Pan (Offset Cropping): Panning involves cropping parts of the image from varying sides, be it left, right, top, or bottom. Using this technique, I simulated the effect of capturing images from different positions and angles. Such augmentation helps my model understand scenarios where the subject of interest is not perfectly centred, mimicking real-world camera misalignments or dynamic scenes where the region of interest changes. 5.3b

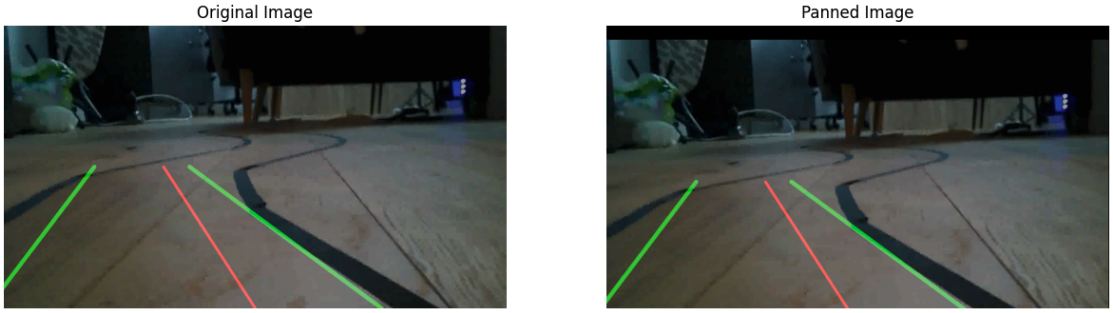
3. Brightness Adjustment: Modifying the brightness of images simulates different lighting conditions. This can range from underexposed (darker) to overexposed (brighter) images. For autonomous driving systems, this is crucial. Different times of the day (dawn, midday, dusk) or environmental conditions (tunnels, shaded areas) present varying lighting conditions. A robust model should recognize objects irrespective of these variations. 5.4a

4. Horizontal Flip: Flipping the image horizontally results in a mirror image of the original. When applying this to autonomous driving, the associated steering angles must be inverted too. This is useful to double the dataset size instantly and to train the model for scenarios where mirrored or opposite scenarios might arise.

5. Gaussian Blur: Applying a Gaussian blur smoothens the image by averaging pixel values within a local neighbourhood. [65] This can simulate



(a) Zoomed Image



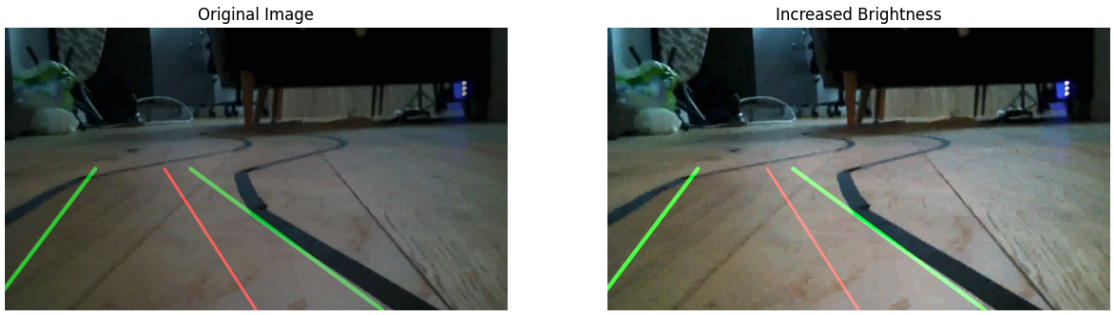
(b) Panned Image

Figure 5.3: Image processing

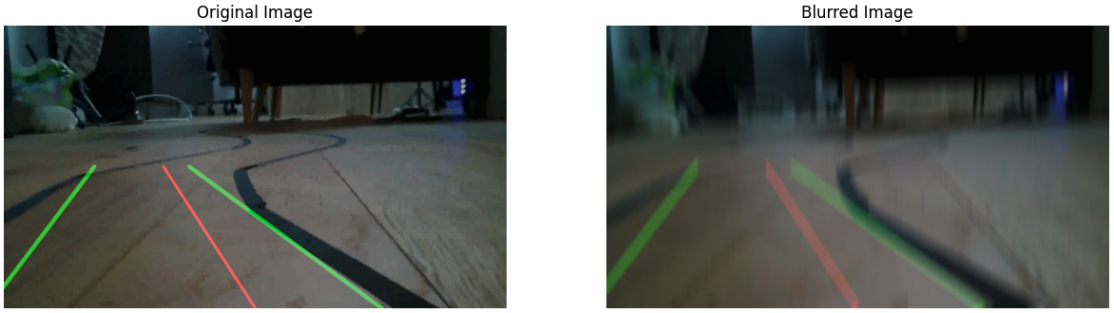
scenarios where the camera is out of focus or when environmental conditions cause visual obscurity. This is important, especially in scenarios like a dirty camera lens. Ensuring a model can decipher blurry images can be the difference between detecting an object or missing it entirely. 5.4b

5.1.3 Data Processing

I have based the lane detection model on the Nvidia Model Architecture [66], since it is a well-tested architecture. The model architecture is further explained in chapter 6. The input planes are 66x200. In the Nvidia paper, it is also suggested that (Y) luma, or brightness, (U) blue projection, (V) red projection (YUV) colour space is recommended, therefore I did some further processing shown in figure 5.5.



(a) Increased Brightness



(b) Gaussian Blur

Figure 5.4: Image processing

5.2 Data Collection and Processing for Traffic Signs Recognition model

The ability of autonomous vehicles to recognize and interpret street signs is paramount to their safe and lawful operation. In this chapter, we present a methodology for detecting and classifying four types of traffic signs: Stop, No Entry, 30 km/h, and 50 km/h. The method is implemented using Python libraries including Keras for evaluating the models [67], Pandas to analyse the big data quantity [68], Numpy [69] for the mathematical operations, especially on PID implementation, and OpenCV for the image detection algorithms. [70]

5.2.1 Dataset Collection

The dataset used to train the model is the DFG Traffic Sign Data Set. [71]. I used the photos of four traffic signs: "Stop", "No Entry", 30 km/h and 50 km/h. I labelled the data accordingly in a CSV file. To further enhance the dataset, I incorporated additional images of miniature traffic signs that I took and had in my possession.

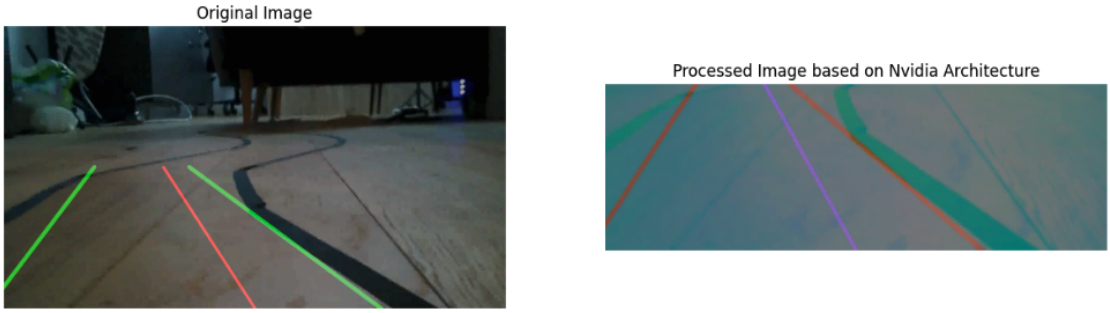


Figure 5.5: Processed Images for Nvidia architecture

5.2.2 Dataset Preprocessing

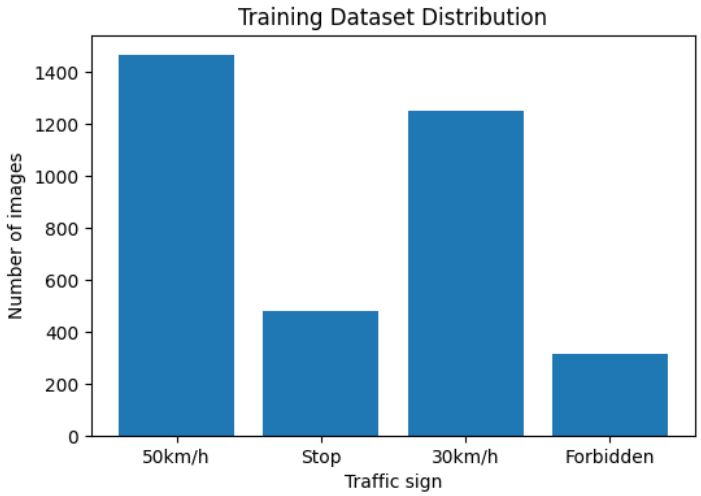


Figure 5.6: Training Set Distribution

- **Image Importation:** The images are imported and resized to 64x64 pixels. The count of images per class is determined, and the dataset is split into training (60%), validation (20%), and testing (20%) subsets.
- **Data Visualization:** The distribution of classes within the dataset is visualized using Matplotlib, and sample images are displayed for each class.
- **Image Preprocessing:** The images are preprocessed by converting them to grayscale and equalizing the histogram to improve contrast. These preprocessed images are then normalized by dividing by 255, scaling the pixel values between 0 and 1.
- **Data Augmentation:** Using the Keras image data generator, the training set is augmented with variations of the original images, including random shifts, zooms, shears, and rotations. This enhances the model's ability to generalize from the training data.

6 Deep Learning Model

6.1 Lane Detection model

6.1.1 Model Architecture

As I briefly mentioned in chapter 5, for the lane detection model, I have used the Nvidia network architecture. Figure 6.1 depicts the network architecture, comprising a total of 9 layers. These layers encompass a normalization layer, 5 convolutional layers, and 3 fully connected layers. The initial step involves dividing the input image into YUV planes, which are subsequently fed into the network.

Based on this model, I wrote the Python code to train my model. (see Appendix, Code 2).

I have used the Exponential Linear Unit (ELU) activation [72], Differing from Rectified Linear Unit (ReLU)s, ELUs incorporate negative values, enabling them to shift the mean activations of units closer to zero. This property resembles batch normalization, yet ELUs achieve it with reduced computational complexity.

6.1.2 Model Training

I chose the number of epochs, steps per epoch and validation sets empirically. Basically, after training my augmented dataset of around 50000 images, with various combinations, I concluded that the best combination for my study, which

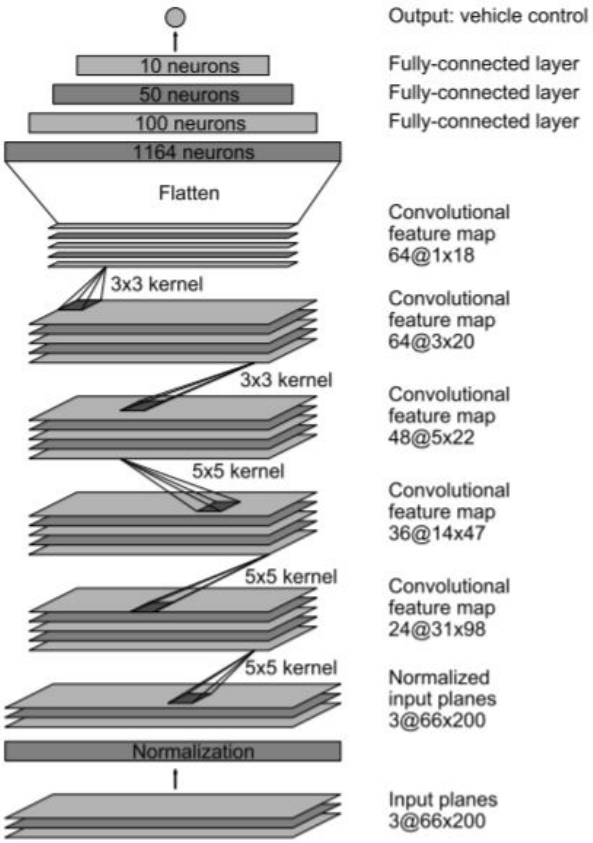


Figure 4: CNN architecture. The network has about 27 million connections and 250 thousand parameters.

Figure 6.1: Nvidia Model Architecture

performs well and avoids overfitting, is this combination: 150 steps per epoch, 10 epochs, a batch size of 200 and 100 validation steps:

Training Data Size: Each epoch consists of 150 steps, and for each step, a batch of 200 samples is processed, therefore the total training samples = steps per epoch x batch size = 150 x 200 = 30000 samples.

Validation Data Size: I have 100 validation steps, and for each step, I am using a batch size of 200. Total Validation Samples = validation steps x batch size = 100 x 200 = 20000 samples.

After the successful training of the model, I generated the graph in figure 6.2 with the training and validation loss. We see that after each epoch, the loss decreases, up until the last epoch, where we reach a training loss of 74 and a validation loss of 45.

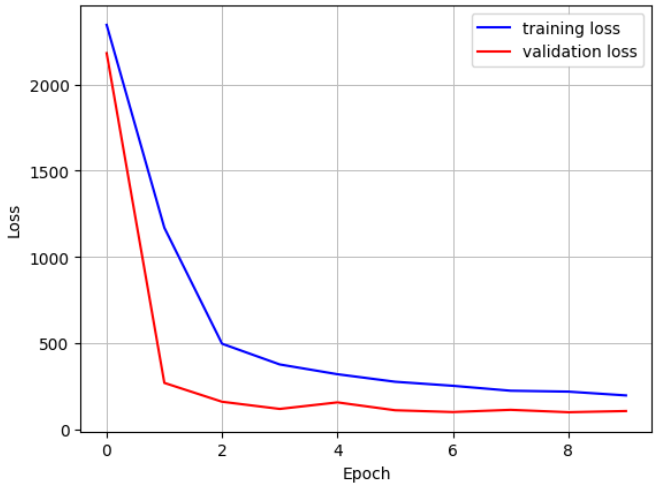


Figure 6.2: Training and Validation Loss

6.2 Traffic Signs Detection model

6.2.1 Model Architecture and Training

The model architecture, implemented with Keras[73] consists of several layers:

- **Convolutional Layers:** Two consecutive convolutional layers with 60 filters of size 5x5 followed by two convolutional layers with 30 filters of size 3x3. ReLU activation functions are used for these layers.
- **Max-Pooling Layers:** These layers follow each pair of convolutional layers and help reduce the spatial dimensions of the feature maps, making the detection of features invariant to scale and translation.
- **Dropout Layers:** Used after the convolutional layers to prevent overfitting, these layers randomly set a fraction of the input units to 0 during training, which helps to prevent overfitting.
- **Flatten Layer:** This layer is used to convert the 2D matrix data to a vector before building the fully connected layers.
- **Dense Layers:** Includes one hidden layer with 500 nodes and a ReLU activation function, followed by the output layer with a softmax activation function to classify the four types of traffic signs.

The model is compiled using the Adam optimizer [74] with a learning rate of 0.001 and a categorical cross-entropy loss function.

The training process uses 10 epochs, and the model's performance is evaluated using both the training and validation data. Training and validation loss and accuracy are plotted to visualize the model's performance over time.

6.2.2 Results and Model Evaluation

The trained model is evaluated on the test set, achieving a test accuracy of score 98.7.

The model, including its weights and architecture, is then saved as a Pickle object for future use in the traffic sign recognition task.

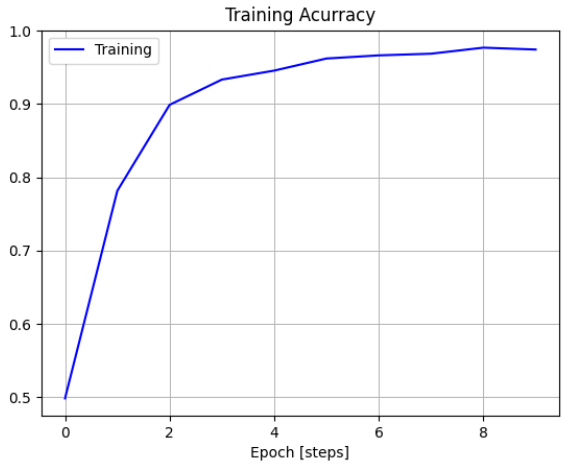
To get more accurate results, I increased the number of epochs [75]. The training and validation loss in 6.1 decreases from 10 to 15 epochs, signifying that the model is learning and minimizing the error in its predictions. This is expected behaviour during the training process as the model adjusts its weights to fit the data better. Correspondingly, the training and validation accuracy increased from 10 to 15 epochs. This indicates that the model is making correct predictions more often, not only on the training data but also on the unseen validation data. An increase in accuracy alongside a decrease in loss suggests that the model is becoming more generalized and is likely to perform well on new, unseen data.

Epochs	**Train loss**	**Train accuracy**	**Validation loss**	**Validation accuracy**
10	0.2500	0.8600	0.220	0.8550
15	0.0801	0.9870	0.0623	0.9882

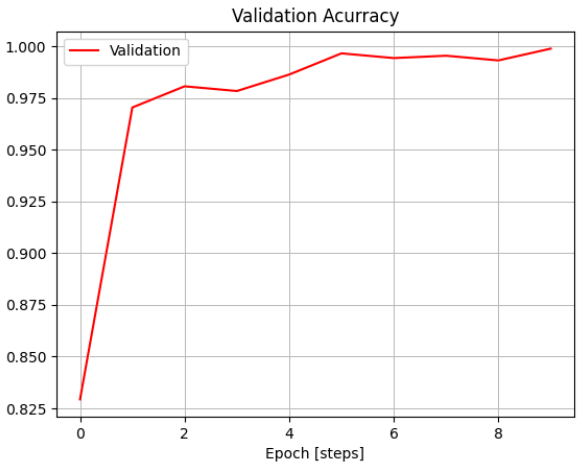
Table 6.1: Results with increased epochs

In Figure 6.3, the training and validation accuracy curves are depicted over the training epochs. A notable observation from the plot is the concurrent upward trend of both curves, which signifies a well-balanced learning process. The training accuracy showcases the model's proficiency on the training dataset, progressively enhancing as the model learns the underlying patterns. Simultaneously, the validation accuracy's consistent increase represents the model's ability to generalize the learned patterns to unseen data. The convergence of the two curves towards similar values emphasizes the model's robustness and indicates the absence of overfitting, reflecting a harmonious balance between bias and variance.

Figure 6.4 illustrates the trend of the training and validation loss throughout the training epochs. Both curves display a consistent downward trajectory, which is an affirmative indication of the model's performance. The decreasing training loss shows the model's refinement in minimizing the error in the training data. Concurrently, the decline in validation loss is indicative of the



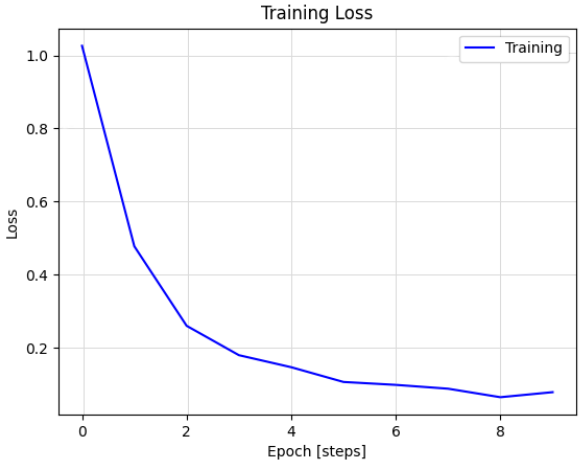
(a) Training Accuracy



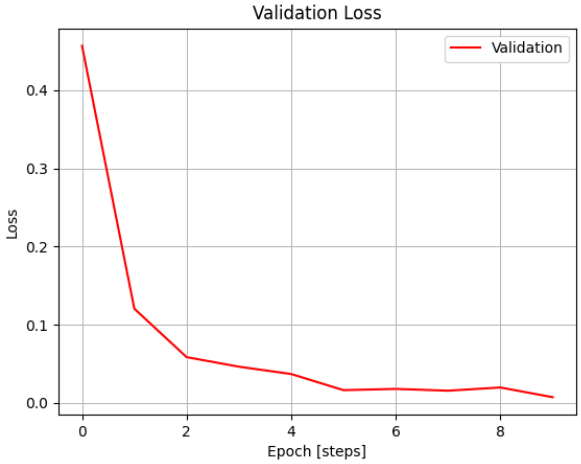
(b) Validation Accuracy

Figure 6.3: ML Model Accuracy Values

model's capacity to extend this refinement to unseen data, thus reducing prediction errors on the validation set. The synchronized decrease in both training and validation loss shows a well-tuned model.



(a) Training Loss



(b) Validation Loss

Figure 6.4: ML Model Loss Values

7 GPS Localization

To monitor the exact geographical location of autonomous vehicles on a global scale, a system needs to incorporate the Global Positioning System (GPS). [76] This can be considered an example of independent mobility [77], where each vehicle has a certain memory. To ensure real-world applicability, I integrated a GPS module into the system. Recognizing that a majority of my experimental procedures were executed within indoor environments—where satellite signals can be weak—I took the measure of testing the module with an active signal amplifier to amplify the signal strength. Additionally, to ensure user-friendliness and interoperability, I chose to implement the location visualization using the Google Maps platform, [78] considering its ubiquity and familiarity. This was facilitated through the Google Application Programming Interface (API), which I integrated into our system.

7.1 Implementation

7.1.1 Choosing the mapping and communication system

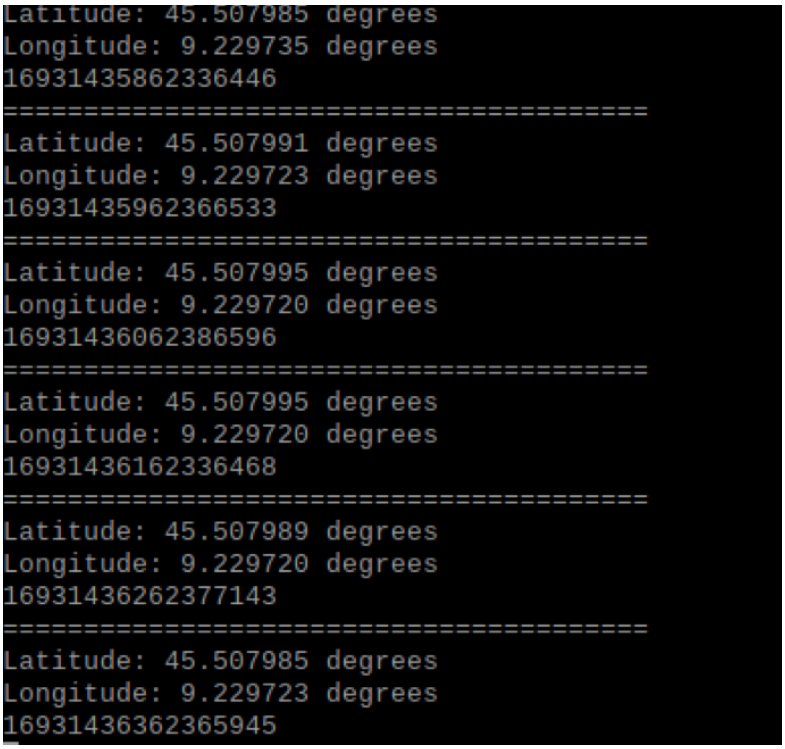
I first set up a developer account with Google and enabled the requisite permissions for Google Maps integration. By doing this, the system gained the capability to forward the latitude and longitude coordinates, as sourced from the GPS module, to the mapping API, in Fig. 7.1a. For this autonomous vehicle solution to be effective, especially in dynamic environments, it is crucial to have real-time data transmission. After evaluating several solutions, I settled on

Pubnub—a state-of-the-art real-time communication platform. Notably, during comparative testing, Pubnub displayed superior performance metrics compared to alternatives like RabbitMQ. The configuration of Pubnub is streamlined, involving a distinct publisher key and a subscriber key.[79] In my system's configuration, the autonomous car assumes the role of the publisher, broadcasting the latitude and longitude data. In contrast, the web application, integrated with the Google Maps API, acts as the subscriber, visualizing the vehicle's live location.

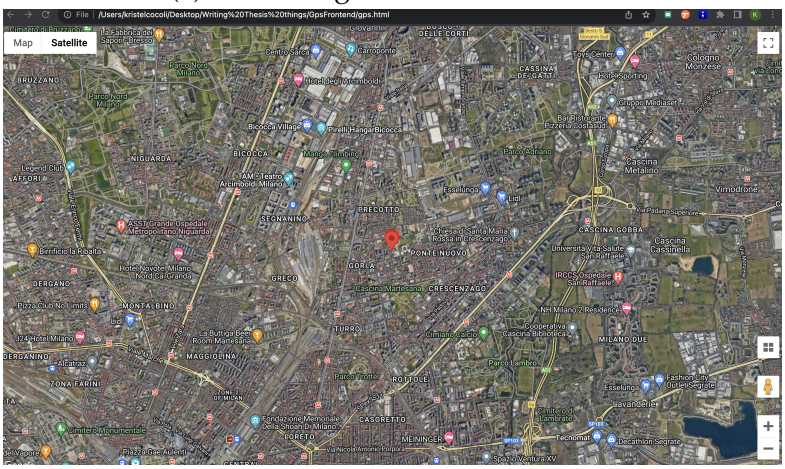
To ensure the reliability and integrity of data transmission, and to mitigate the potential transmission of redundant or duplicated data, I incorporated a system where each data transmission is appended with a unique token. This serves as a checksum and can be visualized as depicted in Figure 7.1a.

7.1.2 Results

In the operational configuration, the system is designed to refresh and update the location data at intervals of 1 second. This ensures a near real-time representation of the vehicle's movement and trajectory, providing users and system administrators with accurate and timely location data.



(a) Data being sent from the server



(b) Google Maps view of the location

Figure 7.1: RabbitMQ Pub/Sub Communication

8 System Control

Among the different control systems employed in the realm of robotics, the Proportional-Integral-Derivative (PID) controller is a popular choice due to its effectiveness and simplicity. [80] This chapter explains the formulation and implementation of the PID controller to dictate the movement of my autonomous vehicle.

8.1 The principle of PID control

A PID controller is an iterative control loop feedback mechanism (or controller) that adjusts a process by calculating the difference between a measured process variable and a desired setpoint.[81] It attempts to minimize the error by adjusting the process control inputs in three distinct manners: Proportional, Integral, and Derivative, hence the name PID.

1. **Proportional Control (P):** This component produces an output value that is proportional to the current error. The proportional response can be adjusted by multiplying the error by a constant known as K_p, the proportional gain constant.

2. **Integral Control (I):** This component is concerned with the accumulation of past errors. If the error has been present for an extended period, it will accumulate (integral of the error), and the controller will respond by changing the control input about a constant K_i, the integral gain.

3. **Derivative Control (D):** This predicts the future trend of the error by understanding its rate of change. It provides a control output to counteract

the rate of error change. The contribution of the derivative component to the overall control action is termed the derivative gain, K_d.

8.1.1 Challenges and Limitations

Though PID controllers are efficient, they may be subject to errors:

- **Noise in Derivative Term:** A major limitation is the amplification of noise in the derivative term, which can cause erratic movements.
- **Steady-State Error:** While the integral term ensures zero steady-state error for constant or slowly varying setpoints, rapid changes can introduce overshoot and oscillations.

8.2 Implementing PID on Raspberry Pi

8.2.1 Hardware Configuration: PID Control with IMU Integration

The implementation of a PID controller in an autonomous vehicle requires a feedback loop, where the actual state of the vehicle is consistently compared to the desired state (setpoint). The desired state is the steering angle dictated by the lane detection module, which the car will aim to achieve. The IMU provides valuable data about the vehicle's current state, such as orientation and acceleration. By incorporating this feedback into the PID controller, one can ensure precise control over the vehicle's movements and stability.

IMU Choice and Placement

For my dissertation, I have used a 9-axis IMU that includes a magnetometer. [82] I have positioned the IMU as close as possible to the centre of gravity of my autonomous car. This minimizes inaccuracies in readings due to external forces. The IMU is well-secured to avoid any vibrations, which can introduce noise into the data. The use of this IMU provided real-time data regarding the vehicle's position, orientation, and velocity:

- Orientation: The combination of an accelerometer (which provides tilt sensing) and gyroscope (which provides rotational speed) helps the system understand in which direction the vehicle is pointing and how it is oriented.
- Position and Navigation: Magnetometers assist in providing a reference to the Earth's magnetic north, which can be beneficial in navigation.
- Stabilization: The IMU's rapid feedback help in systems like traction control and stability control.

To get accurate angle results, it is advised to use sensor fusion algorithms. The most famous algorithms to perform this task are Kalman, Madgwick and Mahony. I experimented with these algorithms and measured their behaviour. The experiments were further analysed with the collaboration of Professor Leonardo Badia. Our work, presented in the next segment of this chapter, was finalised by a published paper in IEEE. [83].

8.2.2 A Comparative Analysis of Sensor Fusion Algorithms

Experiments and Methods

To ensure a fair assessment between the Kalman, Madgwick, and Mahony filters, identical input data and integration periods are provided for each filter. The integration period encompasses the duration of the dead time between sensor readings, during which a timer initiates at the initial data capture and concludes at the subsequent readout operation. This time span is used for processing and is considered in the filter computations. Despite the sequential operation of the filters, the same sensor readout data is maintained and sampled until all the filters have processed the input to guarantee a just comparison of their outputs.

Two thousand data recordings were captured for each filter, focusing on pitch, roll, and time intervals. I explored two situations: (i) with the sensor at rest and (ii) while it's in movement, shifting the sensor. By analyzing the results of the filters under the same input parameters and integration timeframe, the effectiveness of each filter can be gauged and contrasted.

I assessed three diverse sensor fusion methodologies to identify the optimal technique for our specific need. The primary approach we ventured into is the classic trigonometry procedure detailed in the earlier section. This involves consolidating the sensor readings without any extra filtration. Such a method establishes a reference point for comparisons with subsequent methods.

Kalman filter

The Kalman filter can be used to estimate the status of linear systems [84]. This filter algorithm consists of two stages: prediction and update, sometimes [85] also referred to as "propagation" and "correction." A synthetic version of Kalman Filter Algorithm is reported in Table 8.1.

Table 8.1: Steps of the Kalman Filter

Description	Equation
Kalman Gain	$K_k = \mathbf{P}'_k \mathbf{H}^T (\mathbf{H}\mathbf{P}'_k \mathbf{H}^T + \mathbf{R})^{-1}$
Update Estimate	$\hat{\mathbf{x}}_k = \hat{\mathbf{x}'}_k + K_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}'}_k)$
Update Covariance	$\mathbf{P}_k = (\mathbf{I} - K_k\mathbf{H})\mathbf{P}'_k$
Project into k+1	$\hat{\mathbf{x}'}_{k+1} = \mathbf{\Phi}\hat{\mathbf{x}}_k$ $\mathbf{P}_{k+1} = \mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T + \mathbf{Q}$

The first equation represents the expression for the Kalman gain in a Kalman filter, where K_k is the Kalman gain at time step k, $\mathbf{P}'_k$ is the predicted error covariance matrix, $\mathbf{H}$ is the measurement matrix, and $\mathbf{R}$ is the measurement noise covariance matrix [86]. The second equation represents the update step in a Kalman filter, where $\mathbf{x}_k$ is the updated state estimate at time step k, $\mathbf{x}'_k$ is the predicted state estimate, $\mathbf{z}_k$ is the measurement at time step k. The third equation represents the update step for the error covariance matrix in a Kalman filter, where $\mathbf{P}_k$ is the updated error covariance matrix at time step k. The matrix subtraction $(\mathbf{I} - K_k\mathbf{H})$ is often referred to as the Kalman gain update.

For the projection, the first equation, $\hat{\mathbf{x}'}_{k+1} = \mathbf{\Phi}\hat{\mathbf{x}}_k$, predicts the state of the system at the next time step based on the current state. Here, $\hat{\mathbf{x}}_k$ is the estimated state of the system at time step k, and $\mathbf{\Phi}$ is the state transition matrix, which describes how the state of the system evolves over time. Multiplying $\hat{\mathbf{x}}_k$ by $\mathbf{\Phi}$ gives the predicted state of the system at time step $k+1$, denoted by $\hat{\mathbf{x}'}_{k+1}$.

The second equation, $\mathbf{P}_{k+1} = \mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T + \mathbf{Q}$, predicts the error covariance matrix at the next step. Matrix $\mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T$ represents the propagation of the uncertainty in the estimated state to the next step. Matrix $\mathbf{Q}$ is the process noise covariance matrix, which represents the uncertainty in the dynamics of the system that is not accounted for by the state transition matrix. The sum of $\mathbf{\Phi}\mathbf{P}_k\mathbf{\Phi}^T$ and $\mathbf{Q}$ gives the predicted error covariance matrix at time step k+1, denoted by $\mathbf{P}_{k+1}$.

Mahony filter

The Mahony filter [87] tries to improve the estimates from low-quality measurements through a quaternion-based approach, to represent the orientation of the device in 3D space. It uses a proportional-integral-derivative (PID) control algorithm to adjust the estimated orientation based on the difference between the measured and estimated sensor readings. The algorithm is designed to minimize errors over time by adjusting the gain parameters of the filter.

This filter is known for its ability to provide accurate and stable estimates of orientation even in the presence of external disturbances such as vibration or magnetic interference. It is used in a variety of applications, including robotics, aerospace, and virtual reality [88, 89]. However, it requires cautious tuning of its gain parameters to achieve the best performance. A pseudocode of the Mahony filter, as was implemented in the sensor is reported in Algorithm 1.

Algorithm 1 Mahony Filter Algorithm for IMU (Accelerometer and Gyroscope Only)

1. Set algorithm coefficients K_i, K_p and initialize quaternion $q_1 = 1,\ q_2 = q_3 = q_4 = 0$
   while: sensor data is available
2. Read accelerometer measurements a_x, a_y, a_z and gyroscope measurements g_x, g_y, g_z
3. Compute orientation error from accelerometer data, where $e_{i,t}$ represents the integral error of the measurements at time t.
$$\mathbf{e}_{t+1} = \begin{bmatrix} a_{x,t} \\ a_{y,t} \\ a_{z,t} \end{bmatrix} \times \begin{bmatrix} 2(q_2 q_4 - q_1 q_3) \\ 2(q_1 q_2 + q_3 q_4) \\ (q_1^2 - q_2^2 - q_3^2 + q_4^2) \end{bmatrix} \tag{8.1}$$
$$\mathbf{e}_{i,t+1} = \mathbf{e}_{i,t} + \mathbf{e}_{t+1} \Delta t \tag{8.2}$$
4. Update angular velocity computed from gyroscope with the K_i and K_p terms using feedback (fusion)
$$\omega_{t+1} = \omega_t + K_p e_{t+1} + K_i e_{i,t+1} \tag{8.3}$$
5. Compute orientation increment from gyroscope measurements
$$\dot{q}_{\omega,t+1} = \frac{1}{2} \hat{q}_t \otimes [0, \omega_{t+1}]^T \tag{8.4}$$
6. Numerical integration
$$q_{t+1} = \hat{q}_t + \Delta t \dot{q}_{\omega,t+1} \tag{8.5}$$
   endwhile

Madgwick filter

The Madgwick filter used in this paper[90] is relatively simple and can be implemented in most programming languages. It involves calculating the error between the predicted and measured quaternion values and then using that error to update the quaternion estimate. The filter performance can be improved by fine-tuning the algorithm parameters and adjusting the sensor fusion weights to suit specific application requirements. Overall, the Madgwick filter is robust and efficient for attitude estimation and can provide accurate orientation estimates in real-time. The pseudocode of the Madgwick filter, as implemented in the experiments, is shown in Algorithm 2.

Algorithm 2 Madgwick Filter Algorithm for IMU (Accelerometer and Gyroscope Only)

1. Set algorithm gain β and initialize quaternion $q_1 = 1,\ q_2 = q_3 = q_4 = 0$
   while: sensor data is available
2. Read accelerometer measurements a_x, a_y, a_z and gyroscope measurements g_x, g_y, g_z
3. Normalize the accelerometer measurements
4. Calculate the Jacobian matrix and compute the gradient of the cost function

$$\nabla f = J^T f \tag{8.6}$$

$$f = \begin{bmatrix} 2\left(q_2 q_4 - q_1 q_3\right) - a_x \\ 2\left(q_1 q_2 + q_3 q_4\right) - a_y \\ 2\left(\frac{1}{2} - q_2^2 - q_3^2\right) - a_z \end{bmatrix} \tag{8.7}$$

$$J = \begin{bmatrix} -2q_3 & 2q_4 & -2q_1 & 2q_2 \\ 2q_2 & 2q_1 & 2q_4 & 2q_3 \\ 0 & -4q_2 & -4q_3 & 0 \end{bmatrix} \tag{8.8}$$

5. Update the quaternion using the gradient descent algorithm

$$q_{\nabla,t+1} = -\beta \frac{\nabla f}{||f||} \tag{8.9}$$

6. Compute orientation increment from gyroscope measurements

$$\dot{q}_{\omega,t+1} = \frac{1}{2} \hat{q}_t \otimes [0, \omega_{t+1}]^T \tag{8.10}$$

7. Fuse measurements to obtain the estimated attitude

$$q_{t+1} = \hat{q}_t + \Delta t (\dot{q}_{\omega,t+1} + q_{\nabla,t+1}) \tag{8.11}$$

   endwhile

8.3 Experimental results

We performed data collection in two scenarios, referred to as 'stationary' and 'in movement,' where the setup is static and corresponding to being in a moving vehicle, respectively [77]. In both setups, I compare the no-filter trigonometric computation (referred to as the "Matrix" method in the plots) and the three presented filtering methods. I evaluated pitch and roll angles, and the computational complexity of the procedure. In our setup, for the Madgwick filter implementation, I have chosen $\beta = 0.05$. A low value of β, such as 0.05, results in a smoother output by allowing slower convergence. This gives an advantage in this scenario where the measured data contains noise or rapid fluctuations, and a stable and less jittery output is desired. Mahony filter parameters K_p and K_i are 10 and 0, respectively. I noticed during the experiment that in our scenario, K_i did not significantly affect the quality of the measurement, therefore, we set it to 0 for simplicity. This also avoids issues associated with integration, such as overshoot, instability, or slow convergence, especially in scenarios with negligible steady-state errors and gyroscope bias [91].

Stationary results

Figs. 8.2 and 8.1 present the results for the four compared methods in the static scenario, where the sensor is in a resting state, showing the pitch and roll angles, respectively. They show the ability of all techniques to converge to a stable and consistent estimate of the state of the system over a period of time when the inputs to the system are not changing, yet with different overall performance. [92].

Since the device is static without any tilt, I expect the pitch and roll values to be near 0. In Fig. 8.2, Madgwick's behavior appears to have wider oscillations, possibly caused by the acquisition of more noise. This depends on the coefficient β, which tunes the memory from the previous state of the quaternion, as described in (8.9). This implies that β is implicitly related to the low-pass pole of the system, and there exists a trade-off. If we are looking to achieve a higher bandwidth of the filter, to respond to fast-changing angles, the cost is the presence of more noise, acquired by the filter itself. This value of β was chosen because it allowed the Madgwick filter to respond as fast as the other filters, but as we can see, we do acquire more noise compared to Kalman and Mahony

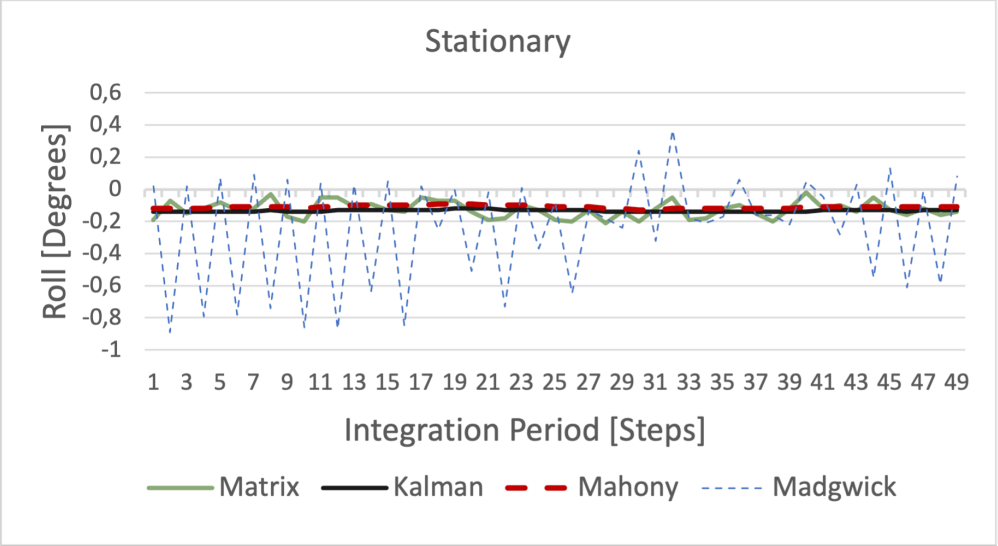


Figure 8.1: Roll angle comparison of the filters with stationary IMU.

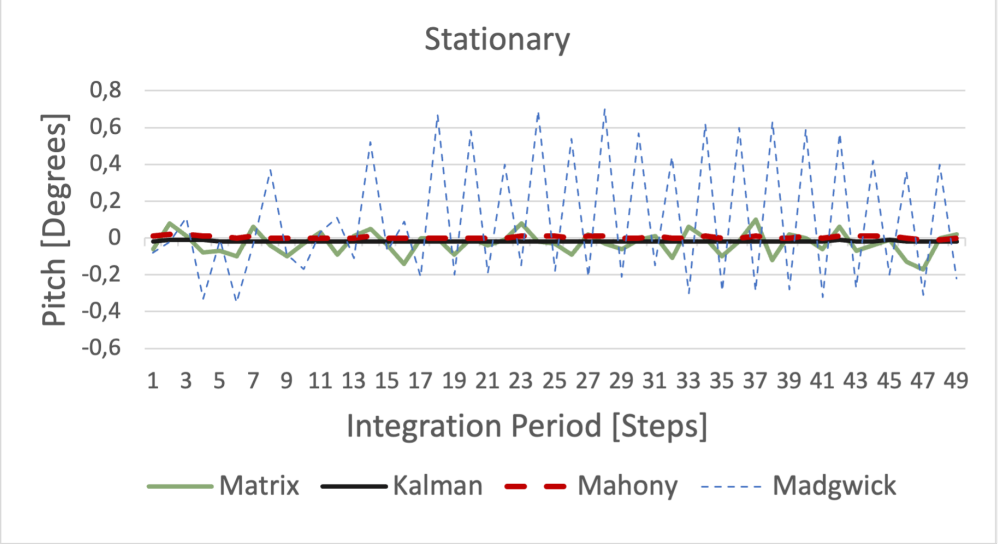


Figure 8.2: Pitch angle comparison of the filters for a stationary IMU.

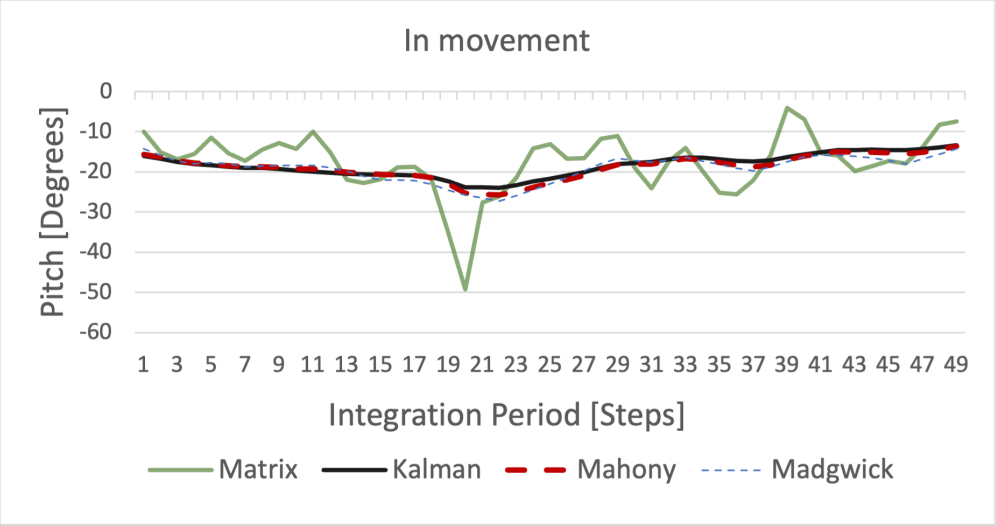


Figure 8.3: Pitch angle comparison of the filters with moving IMU.

filters.

In Fig. 8.1, we notice oscillations around -0.1 degrees. In an ideal scenario, in a stationary state, the roll value would be 0. This could be due to several factors. One possible reason is that the sensor has a small bias or offset in its measurements, which can cause it to read a non-zero value even at rest. Another possibility is that the sensor is mounted on a surface that is not perfectly level. In most cases, a small deviation such as -0.1 degrees of roll angle when the sensor is not moving is not a significant concern, as it falls within the expected range of error for many IMUs

Results in movement

It is also important to evaluate the ability of fusion filters to provide accurate and reliable estimates of the system state as it moves. I intend to utilize this IMU in self-driving vehicles [24], onboard of which the orientation of the device can change. However, unless the car is flipping over, the values of pitch and roll angles are expected to be less than 90 degrees, so I moved the device within that extent.

Figs. 8.3 and 8.4 show that the recorded pitch and roll angles in the no-filter scenario (Matrix) have more oscillations. It is interesting to note that the expected behaviour of the traditional method would be with the presence of drifts. In these graphs, I do not have drifts. This is possibly because we have graphed data from only 50 integration steps over 2000 acquired integration steps. By

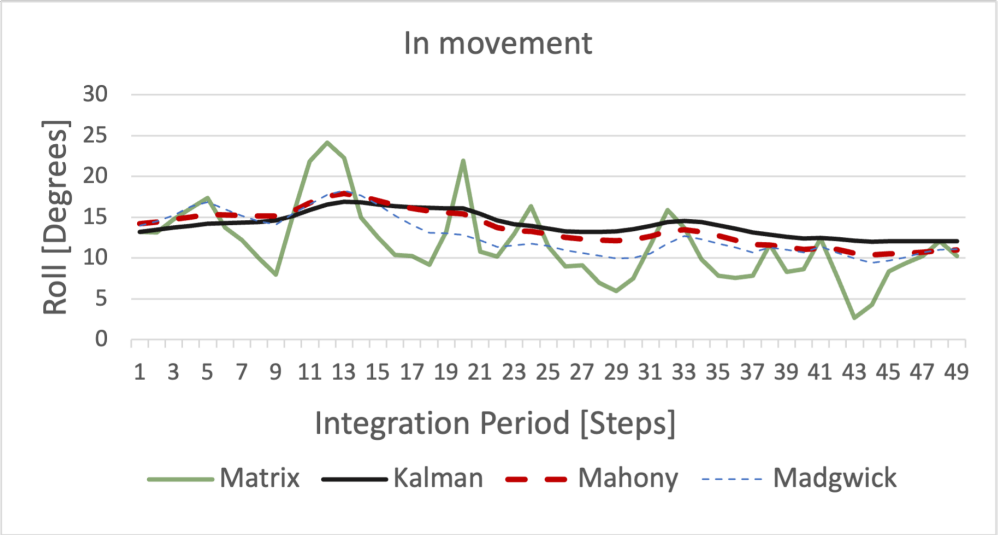


Figure 8.4: Roll angle comparison of the filters with moving IMU.

applying no-filtering action, it is expected that in the signal acquisition, the high frequency component of the noise is present. The filters' response is similar. It is seen a small difference in Kalman filter's response. Kalman filter exhibits a better behaviour, with less oscillations than Madgwick filter and Mahony filter.

In Fig. 8.3, the pitch measurement is around the value −15 degrees, which indicates that the sensor is tilted downwards, whereas in Fig. 8.4, the roll measurement is around the value +15 degrees, implying that the sensor is tilted to the right.

Complexity

In the moving state, when real-time analysis of the input data is valuable, I computed the processing speed of the filters. Fig. 8.5 shows that the trigonometry method is the slowest, while the time is significantly reduced with the filter implementations. Kalman and Madgwick exhibit similar behavior. Mahony is the fastest, with the lowest computation time per integration step. Table 8.2 reports the average values.

The selected Sensor Fusion Algorithm

It is strongly advised to use a filter, the choice of which depends on the specific requirements and limitations of the application, such as the sensor type, noise level, computation power, and desired accuracy and stability. Based on

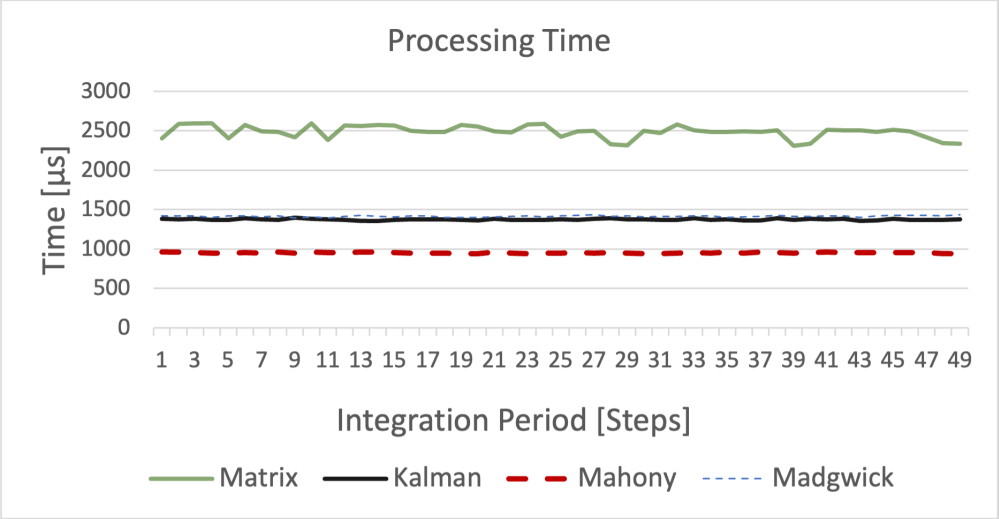


Figure 8.5: Comparing the processing time of the filters

Table 8.2: Average Processing Time

Method	Average processing time
Matrix	2477.4 µs
Kalman	1375.9 µs
Mahony	950.5 µs
Madgwick	1416.8 µs

our results, the Kalman filter would be the suggested option if accuracy is the primary concern and computational resources and memory are not constraints. If computational efficiency is a critical factor, the Mahony filter would be more suitable due to its computational speed. In the case of autonomous cars, a fast response is more important, therefore, I have used Mahony to conduct my further research and experiments on the autonomous journey.

8.4 Software

After taking the steering angle from the lane detection module, as well as the current car angle from the IMU, the difference of these data will be used as input to the PID controller. The goal of the PID is to decrease this difference, by changing the angle taken from the IMU (the real-time angle), until the difference reaches zero. When the difference equals zero, that means the car motors have moved, changing the IMU angle towards the angle it should be. (the angle dictated from the lane detection module).

8.4.1 PID Implementation

I have written the PID code implementation with the help of MathWork's technical documentation on PID [93].

The proportional coefficient (K_p) determines the car's oscillation speed, while the derivative coefficient (K_d) is employed to fine-tune the extent of deviation (or oscillation amplitude) from the desired trajectory. Over time, accumulated errors in the steering angle, stemming from systematic biases, can eventually take the car off its course. The integral component addresses this issue. Given its influence on error accumulation over time, the integral coefficient (K_i) must be adjusted since it greatly affects the system's overall behavior. Therefore,setting the optimal gains for P, I and D can be complex, therefore I chose them empirically, based on which combination gave the best result for my system, as shown on table 8.3

K_p	K_d	K_i	**Car Behaviour**
1.2	0.030	14	Too many ocillations
1.4	0.06	17	More oscillations than before
1.15	0.020	10	Response very fast and too many oscilations
1.15	0.020	9	Response very fast and too many oscilations
0.9	0.010	4	Stepped over the lane
0.6	0.005	3	Stepped over the lane
0.55	0.005	3	Stepped over the lane
0.3	0.0025	2.2	Satisfying result

Table 8.3: Empirical choice of K_p,K_d,K_i

The final values are $K_p = 0.3$, $K_d = 0.0025$ and $K_i = 2.2$. With this values, the autonomous car drove smoothly through the test lanes, with gradual angle changes and more stable reaction (see Appendix, code 1).

9

AR & VR Headset

Leading automobile brands like Tesla, Rivian, and Waymo have been testing their vehicles on roads across various terrains and conditions. [94] The meticulous and evolving nature of these tests emphasizes their vital importance to the safe and efficient functioning of autonomous vehicles. Parallelly, the world of Augmented Reality (AR) and VR has seen transformative advancements, with tools such as the Apple Vision Pro revolutionizing how users interact with and perceive digital content.

Given the confluence of these two realms, I was driven by an idea: Could there be a more immersive way for individuals to engage with the autonomous testing phase, transcending the conventional methods of screen observation or third-party witnessing?

To explore this intersection, I embarked on integrating the feed from a Raspberry Pi camera directly into the Oculus Quest 2 VR headset. While the experiment was interesting in its approach, it was not without its limitations. The hardware constraints of the Raspberry Pi camera constrained the immersive experience to a 2D visual landscape, rather than the more encompassing 3D, primarily due to the camera's inability to capture a 360-degree panorama. Despite these limitations, the underlying technology is sound and stands as a testament to the potential development of autonomous vehicular testing and immersive VR experiences.

Figure 9.1: Oculus Quest 2

9.1 Implementation

9.1.1 Unity Scene Setup

For interfacing with the VR headset, I used the Unity engine. The scene configuration is depicted in Figure 9.2. Given that the primary objective was to display the camera data, a solitary screen sufficed. The pink quadrilateral figure represents the display screen, and for this screen, the YUY2 texture format was utilized. This format uses the YUV colour space. [95]

YUV Color Space

- **Luminance and Chrominance Separation:** The Y component in YUV represents the luminance (brightness) of the colour, while the U and V components represent the chrominance (colour information). This separation can lead to efficient compression since the human eye is more sensitive to luminance than chrominance. By storing or transmitting chrominance at a reduced resolution, bandwidth can be saved without significant perceived loss in image quality. [96]

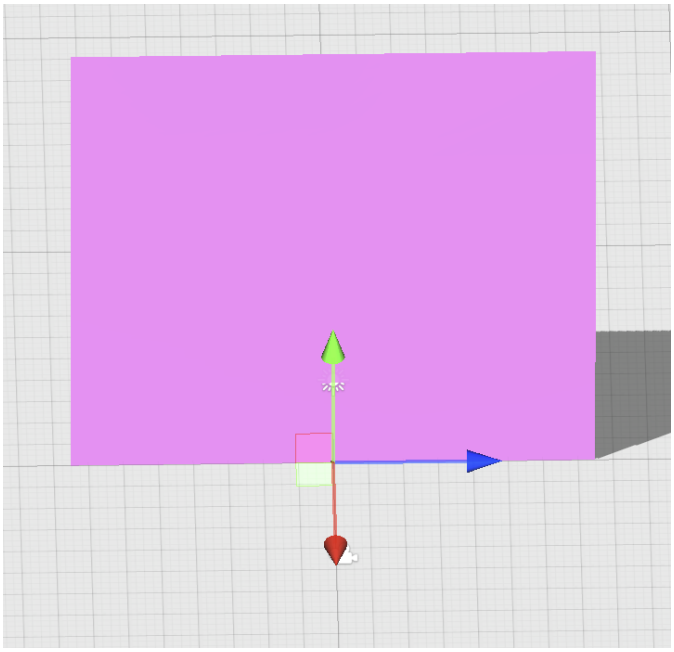


Figure 9.2: Unity Scene Setup

- **Compatibility with Black and White Displays:** The Y component alone can be used to represent black and white images, which ensures compatibility with monochrome displays and legacy systems.
- **Reduced Bandwidth for Transmission:** For broadcasting and streaming applications, YUV allows for efficient chroma subsampling. Popular formats like YUV 4:2:0 take advantage of this by storing fewer chrominance values than luminance, leading to reduced data rates.
- **Color Correction:** The separation of luminance and chrominance also facilitates colour correction without affecting the brightness of the image.
- **Reduced Artifacts:** When compressing video data, separating the colour from luminance can result in fewer visual artefacts. This means that even if there's a loss in colour detail, the overall structure and clarity of the image might remain intact.

9.1.2 Software Implementation

The simplest way to send data from the Raspberry Pi to a machine running Unity in real-time is to use sockets, specifically Transmission Control Protocol (TCP) Sockets [97]. They use the Transmission Control Protocol (TCP) for data transmission and are reliable because they ensure the delivery of each packet of data. If a data packet is lost during transmission, the protocol will retry until the

packet is correctly received or until the connection is dropped. On the Raspberry side, the 'sockets' library is used. In the Unity site, 'System.net.sockets' is used.

I noticed that the broadcast was not smooth enough. Especially in streaming or real-time communication, data might be produced at a rate faster or slower than it can be consumed or transmitted. This is caused by delivery delays. In the literature, there are many interesting approaches on error control techniques [98] when transmitting packets. My proposed solution to this specific problem is a buffer that can temporarily hold this data, ensuring smooth and steady data flow. Therefore I added a 4096 bytes buffer to my program.

The final results from the headset, taking into account the hardware limitations, are shown in figure 9.3:

Figure 9.3: View from Oculus Quest 2 headset

10

Conclusions and Future Works

10.1 Conclusive thoughts

This autonomous car journey represents an approach to develop an autonomous car system, focusing on real-time performance, efficiency, and safety. [99] Two core machine learning modules, the lane detection and the traffic signs detection through CNN, are the backbone of my solution.

The integration of an Inertial Measurement Unit to get the actual angle of the vehicle in real-time is a significant enhancement over traditional methods. The combination of lane detection, which provides the desired angle, and IMU, offering real-time angles, aids in ensuring that the car remains stable and on course. This is further refined by a PID controller, smoothing the vehicle's trajectory adjustments, and providing a more naturalistic driving experience.

Another achievement worth mentioning is the integration of the GPS module. By offering real-time geolocation data, the system not only gains awareness about its global position but also potentially supports functions like route planning and geofencing in the future.

The Virtual Reality (VR) headset linkage presents a novel approach to visualizing the autonomous car's perspective. This feature enhances the user's connection with the technology, potentially offering novel diagnostic, entertainment, or even remote-control applications.

The compact nature of the Raspberry Pi, combined with its cost-effectiveness, transforms our autonomous car system into a feasible option for scaled-down

models, educational purposes, and potential prototype development for larger vehicles. With further refinement, it could also be explored for commercial applications or even in sectors like agriculture for automated machinery.

The VR linkage might set the groundwork for teleoperated vehicles, where operators can remotely control vehicles in environments where human access might be dangerous or impractical.

10.2 Future Works

- Enhanced Machine Learning Models: As the field of Artificial Intelligence (AI) continues to evolve, newer and more efficient architectures could be explored. Transfer learning or fine-tuning with local datasets might boost detection accuracy.
- Integration of Additional Sensors: Incorporating LiDAR or Radar could aid in better obstacle detection and enhance overall system robustness. [100] [101]
- Advanced Navigation Features: With the GPS module in place, developing features like dynamic route planning, traffic congestion awareness, and obstacle avoidance could be the next steps.
- User Experience (UX) Enhancements with VR: The usage of a 360-degree camera [102] is compulsory to ensure immersiveness and to have a broader field of view. The Raspberry Pi camera offered only a 2D view of the screen. Improving the VR experience by integrating more sensory feedback, such as haptic responses [103] could be an interesting avenue as well, considering that tacticle simuli can significantly improve the visual feedback. [104] In addition, innovative approaches on the classification of grasping tasks [105] like holding the wheel, could lead to an even more realistic experience.

10.3 Closing Remarks

In conclusion, this thesis has marked a journey in creating an autonomous car system leveraging the capabilities of a Raspberry Pi. It signifies the promise and potential of affordable and accessible technology to shape the future of autonomous mobility. The innovations introduced, from machine learning-driven modules to VR integrations, form the basis upon which future research can build, enhancing the boundaries of what's possible in the realm of autonomous

driving. We anticipate that this work will not only inspire further advancements in the field but also play a pivotal role in making autonomous systems more accessible to the broader community.

References

[1] *Taxonomy and Definitions for Terms Related to Driving Automation Systems for On-Road Motor Vehicles*. Standard. United States: The Society of Automotive Engineers, 2021.

[2] Lee, M.S. et al. "Individual Stable Driving Pattern Analysis for Evaluating Driver Readiness at Autonomous Driving Levels 2 and 3". In: *2018 International Conference on Information and Communication Technology Convergence (ICTC)*. 2018, pp. 315–319.

[3] Min, K. et al. "SAE Level 3 Autonomous Driving Technology of the ETRI". In: *2019 International Conference on Information and Communication Technology Convergence (ICTC)*. 2019, pp. 464–466.

[4] Arfini, S., Spinelli, D., and Chiffi, D. "Ethics of Self-driving Cars: A Naturalistic Approach. Minds & Machines 32". In: (2022), pp. 717–734.

[5] Drummond, C. "Machine learning as an experimental science (revisited)". In: *AAAI workshop on evaluation methods for machine learning*. AAAI Press Menlo Park, CA, USA. 2006, pp. 1–5.

[6] H. Jhaveri, A.R. "A Review on Machine Learning Strategies for Real-World Engineering Applications". In: *Mobile Information Systems* 2022 (2022), p. 26.

[7] Wang, Q. et al. "A comprehensive survey of loss functions in machine learning". In: *Annals of Data Science* (2020), pp. 1–26.

[8] Dayan, P., Sahani, M., and Deback, G. "Unsupervised learning". In: *The MIT encyclopedia of the cognitive sciences* (1999), pp. 857–859.

[9] Omran, M.G., Engelbrecht, A.P., and Salman, A. "An overview of clustering methods". In: *Intelligent Data Analysis* 11.6 (2007), pp. 583–605.

[10] Hady, M.F.A. and Schwenker, F. "Semi-supervised learning". In: *Handbook on Neural Information Processing* (2013), pp. 215–239.

[11] Nukita, T. et al. "Damaged Lane Markings Detection Method with Label Propagation". In: *2018 IEEE 24th International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA)*. 2018, pp. 203–208.

[12] Sutton, R.S. and Barto, A.G. *Reinforcement learning: An introduction*. MIT press, 2018.

[13] Coggan, M. "Exploration and exploitation in reinforcement learning". In: *Research supervised by Prof. Doina Precup, CRA-W DMP Project at McGill University* (2004).

[14] Singh, M.K. and Singh, K.K. "A review of publicly available automatic brain segmentation methodologies, machine learning models, recent advancements, and their comparison". In: *Annals of Neurosciences* 28.1-2 (2021), pp. 82–93.

[15] Feng, F. et al. "Learning deep hierarchical spatial–spectral features for hyperspectral image classification based on residual 3D-2D CNN". In: *Sensors* 19.23 (2019), p. 5276.

[16] Jiang, J. et al. "MultiBSP: multi-branch and multi-scale perception object tracking framework based on siamese CNN". In: *Neural Computing and Applications* 34.21 (2022), pp. 18787–18803.

[17] Jogin, M. et al. "Feature extraction using convolution neural networks (CNN) and deep learning". In: *2018 3rd IEEE international conference on recent trends in electronics, information & communication technology (RTEICT)*. IEEE. 2018, pp. 2319–2323.

[18] Chandana, R. and Ramachandra, A. "Real time object detection system with YOLO and CNN models: A review". In: *arXiv preprint arXiv:2208.00773* (2022).

[19] Sasiadek, J.Z. "Sensor fusion". In: *Annual Reviews in Control* 26.2 (2002), pp. 203–228.

[20] Zimdahl, W. "Guidelines and some developments for a new modular driver information system". In: *34th IEEE Vehicular Technology Conference*. Vol. 34. 1984, pp. 178–182.

[21] Kovács, L. et al. "Sensor design and integration into small sized autonomous vehicle". In: *2022 IEEE 2nd Conference on Information Technology and Data Science (CITDS)*. 2022, pp. 171–176.

[22] Wang, Y., Liu, D., and Matson, E. "Accurate Perception for Autonomous Driving: Application of Kalman Filter for Sensor Fusion". In: *2020 IEEE Sensors Applications Symposium (SAS)*. 2020, pp. 1–6.

[23] Escamilla-Ambrosio, P. and Lieven, N. "Sensor fusion approaches to guideway and obstacle detection in the autotaxi system". In: *2005 7th International Conference on Information Fusion*. Vol. 2. 2005, 7 pp.-.

[24] Raveena, C. et al. "Sensor Fusion Module Using IMU and GPS Sensors For Autonomous Car". In: *2020 IEEE International Conference for Innovation in Technology (INOCON)*. 2020, pp. 1–6.

[25] Suwattanapunkul, T. and Wang, L.-J. "The Efficient Traffic Sign Detection and Recognition for Taiwan Road Using YOLO Model with Hybrid Dataset". In: *2023 9th International Conference on Applied System Innovation (ICASI)*. 2023, pp. 160–162.

[26] Fang, S., Xin, L., and Chen, Y. "Traffic sign detection based on co-training method". In: *Proceedings of the 33rd Chinese Control Conference*. 2014, pp. 4893–4898.

[27] Lu, H., Deng, Z., and Zhang, R. "Lane Intrusion Behaviors Dataset: Action Recognition in Real-world Highway Scenarios for Self-driving". In: *2021 International Joint Conference on Neural Networks (IJCNN)*. 2021, pp. 1–6.

[28] Sanil, N. et al. "Deep Learning Techniques for Obstacle Detection and Avoidance in Driverless Cars". In: *2020 International Conference on Artificial Intelligence and Signal Processing (AISP)*. 2020, pp. 1–4.

[29] Vicini, M. et al. "Decision Making via Game Theory for Autonomous Vehicles in the Presence of a Moving Obstacle". In: *Proc. IEEE COMNETSAT*. 2022, pp. 393–398.

[30] Michieli, U. and Badia, L. "Game theoretic analysis of road user safety scenarios involving autonomous vehicles". In: *Proc. IEEE PIMRC*. 2018, pp. 1377–1381.

[31] Hussain, R. and Zeadally, S. "Autonomous Cars: Research Results, Issues, and Future Challenges". In: *IEEE Communications Surveys and Tutorials* 21.2 (2019), pp. 1275–1313.

[32] Paponpen, K. et al. “The Implementation of Steering Angle Estimation on Miniature Raspberry Pi-based Autonomous Car”. In: *2022 IEEE 17th Conference on Industrial Electronics and Applications (ICIEA)*. 2022, pp. 1037–1042.

[33] Hendry, G. et al. “PyRoboCar: A Low-cost Deep Neural Network-based Autonomous Car”. In: *2021 IEEE Frontiers in Education Conference (FIE)*. 2021, pp. 1–5.

[34] Sangiovanni Vincentelli, A.L. and Vigna, B. “Autonomous vehicles: A playground for sensors”. In: *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)*. 2017, pp. 2–2.

[35] Karbab, E. et al. “Car park management with networked wireless sensors and active RFID”. In: *2015 IEEE International Conference on Electro/Information Technology (EIT)*. 2015, pp. 373–378.

[36] Richardson, M. and Wallace, S. *Getting started with raspberry PI*. " O’Reilly Media, Inc.", 2012.

[37] Rocha Rodrigues, E. et al. “DeepDownscale: A Deep Learning Strategy for High-Resolution Weather Forecast”. In: *2018 IEEE 14th International Conference on e-Science (e-Science)*. 2018, pp. 415–422.

[38] Zancanaro, A., Cisotto, G., and Badia, L. “Modeling value of information in remote sensing from correlated sources”. In: *Comp. Commun.* 203 (2023), pp. 289–297.

[39] Ismail, M., Dhamodharan, R., and Sekar, G. “Smart Obstacle Recognition System using Raspberry Pi”. In: *2021 5th International Conference on Computing Methodologies and Communication (ICCMC)*. 2021, pp. 672–675.

[40] Özsoy, K., Bozkurt, A., and Tekin, I. “2D Indoor positioning system using GPS signals”. In: *2010 International Conference on Indoor Positioning and Indoor Navigation*. 2010, pp. 1–6.

[41] Özsoy, K., Bozkurt, A., and Tekin, I. “Indoor positioning based on global positioning system signals”. In: *Microwave and Optical Technology Letters* 55.5 (2013), pp. 1091–1097.

[42] Thompson, R.B. “Global positioning system: the mathematics of GPS receivers”. In: *Mathematics magazine* 71.4 (1998), pp. 260–269.

[43] Bulusu, N., Heidemann, J., and Estrin, D. "GPS-less low-cost outdoor localization for very small devices". In: *IEEE Personal Communications* 7.5 (2000), pp. 28–34.

[44] Talebi Abatari, H. and Dehghani Tafti, A. "Using a fuzzy PID controller for the path following of a car-like mobile robot". In: *2013 First RSI/ISM International Conference on Robotics and Mechatronics (ICRoM)*. 2013, pp. 189–193.

[45] Scheiner, N. et al. "Automated ground truth estimation for automotive radar tracking applications with portable GNSS and IMU devices". In: *2019 20th International Radar Symposium (IRS)*. IEEE. 2019, pp. 1–10.

[46] Petrenko, D., Kryvenchuk, Y., and Yakovyna, V. "Enhancing Data Discretization for Smoother Drone Input Using GAN-Based IMU Data Augmentation". In: *Drones* 7.7 (2023), p. 463.

[47] Ahmad, N. et al. "Reviews on various inertial measurement unit (IMU) sensor applications". In: *International Journal of Signal Processing Systems* 1.2 (2013), pp. 256–262.

[48] Siepert, B. *Adafruit TDK InvenSense ICM-20948 9- DoF IMU*. TJA1043. Adafruit Industries. Nov. 2021.

[49] Badia, L. and Scalabrin, M. "Stochastic analysis of delay statistics for intermittently connected vehicular networks". In: *Proc. European Wireless*. 2014.

[50] Cui, J. and Wu, C. "Design and implementation of directory service network management system based on event bus pattern". In: *2010 2nd International Conference on Computer Engineering and Technology*. Vol. 4. IEEE. 2010, pp. V4–255.

[51] Viwatpinyo, S. and Phatchuay, S. "The Automatic Car to Implementation of Lane Detective using Raspberry Pi 3 Model B on OpenCV". In: *2022 International Conference on Cybernetics and Innovations (ICCI)*. 2022, pp. 1–5.

[52] Tabernik, D. and Skočaj, D. "Deep learning for large-scale traffic-sign detection and recognition". In: *IEEE transactions on intelligent transportation systems* 21.4 (2019), pp. 1427–1440.

[53] *.NET documentation*. 2023.

[54] Patil, M.M. et al. "A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retriewal operations using a web/android application to explore load balancing—Sharding in MongoDB and its advantages". In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)(I-SMAC)*. IEEE. 2017, pp. 325–330.

[55] Patil, M.M. et al. "A qualitative analysis of the performance of MongoDB vs MySQL database based on insertion and retriewal operations using a web/android application to explore load balancing — Sharding in MongoDB and its advantages". In: *2017 International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud) (I-SMAC)*. 2017, pp. 325–330.

[56] Jordaan, P.W. and Holm, J.E.W. "Reflection on MongoDB Database Logical and Physical Modeling". In: *2019 IEEE AFRICON*. 2019, pp. 1–8.

[57] *MongoDB Manual*. 2023.

[58] *RabbitMQ Server Documentation*. 2023.

[59] Estrada, N. and Astudillo, H. "Comparing scalability of message queue system: ZeroMQ vs RabbitMQ". In: *2015 Latin American Computing Conference (CLEI)*. 2015, pp. 1–6.

[60] Vandikas, K. and Tsiatsis, V. "Performance Evaluation of an IoT Platform". In: *2014 Eighth International Conference on Next Generation Mobile Apps, Services and Technologies*. 2014, pp. 141–146.

[61] Garg, M., Sehrawat, A., and Savaridassan., P. "Vehicle Lane Detection for Accident Prevention and Smart Autodrive Using OpenCV". In: *2023 International Conference on Computer Communication and Informatics (ICCCI)*. 2023, pp. 1–5.

[62] Kim, J.-H. et al. "Lane recognition algorithm using lane shape and color features for vehicle black box". In: *2018 International Conference on Electronics, Information, and Communication (ICEIC)*. IEEE. 2018, pp. 1–2.

[63] Rong, W. et al. "An improved Canny edge detection algorithm". In: *2014 IEEE International Conference on Mechatronics and Automation*. 2014, pp. 577–582.

[64] Sultana, S. et al. "Vision-Based Robust Lane Detection and Tracking in Challenging Conditions". In: *IEEE Access* 11 (2023), pp. 67938–67955.

[65] Gedraite, E.S. and Hadad, M. "Investigation on the effect of a Gaussian Blur in image filtering and segmentation". In: *Proceedings ELMAR-2011*. IEEE. 2011, pp. 393–396.

[66] Bojarski, M. et al. "End to End Learning for Self-Driving Cars". In: (Apr. 2016).

[67] Pritt, M. and Chern, G. "Satellite Image Classification with Deep Learning". In: *2017 IEEE Applied Imagery Pattern Recognition Workshop (AIPR)*. 2017, pp. 1–7.

[68] Stančin, I. and Jović, A. "An overview and comparison of free Python libraries for data mining and big data analysis". In: *2019 42nd International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*. 2019, pp. 977–982.

[69] Walt, S. van der, Colbert, S.C., and Varoquaux, G. "The NumPy Array: A Structure for Efficient Numerical Computation". In: *Computing in Science and Engineering* 13.2 (2011), pp. 22–30.

[70] Jain, H. et al. "DoodSearch - OpenCV with Image Recognition". In: *2022 IEEE Conference on Interdisciplinary Approaches in Technology and Management for Social Innovation (IATMSI)*. 2022, pp. 1–4.

[71] Tabernik, D. and Skočaj, D. "Deep Learning for Large-Scale Traffic-Sign Detection and Recognition". In: *IEEE Transactions on Intelligent Transportation Systems* (2019). ISSN: 1524-9050.

[72] Hao, W. et al. "The Role of Activation Function in CNN". In: *2020 2nd International Conference on Information Technology and Computer Application (ITCA)*. 2020, pp. 429–432.

[73] Dell'Aversana, P. *DEEP NEURAL NETWORKS WITH KERAS Testing different Deep Neural Network parameters for classification of rock data samples*. May 2019.

[74] Mehta, S., Paunwala, C., and Vaidya, B. "CNN based traffic sign classification using Adam optimizer". In: *2019 international conference on intelligent computing and control systems (ICCS)*. IEEE. 2019, pp. 1293–1298.

[75] Chi, J. et al. "Performance Analysis of Three kinds of Neural Networks in the Classification of Mask Images". In: *Journal of Physics: Conference Series* 2181.1 (Jan. 2022), p. 012032.

[76] Ceruzzi, P.E. *GPS*. MIT Press, 2018.

[77] Badia, L. and Bui, N. "A group mobility model based on nodes' attraction for next generation wireless networks". In: *Proc. ACM Mobility*. 2006.

[78] Zhu, Y. "Introducing Google Chart Tools and Google Maps API in Data Visualization Courses". In: *IEEE Computer Graphics and Applications* 32.6 (2012), pp. 6–9.

[79] Pubnub. *Pubnub Developers Documentation*. Mar. 2023.

[80] Abdulameer, A. et al. "Tuning methods of PID controller for DC motor speed control". In: *Indonesian Journal of Electrical Engineering and Computer Science* 3.2 (2016), pp. 343–349.

[81] Li, Z., Hu, J., and Huo, X. "PID control based on RBF neural network for ship steering". In: *2012 World Congress on Information and Communication Technologies*. 2012, pp. 1076–1080.

[82] Brunner, T. et al. "Magnetometer-augmented IMU simulator: In-depth elaboration". In: *Sensors* 15.3 (2015), pp. 5293–5310.

[83] Çoçoli, K. and Badia, L. "A Comparative Analysis of Sensor Fusion Algorithms for Miniature IMU Measurements". In: *2023 International Seminar on Intelligent Technology and Its Applications (ISITIA)*. 2023, pp. 239–244.

[84] Li, Q. et al. "Kalman Filter and Its Application". In: *Proc. ICINIS*. 2015, pp. 74–77.

[85] Govaers, F. *Introduction and Implementations of the Kalman Filter*. IntechOpen, May 2019. ISBN: 978-1-83880-537-1.

[86] Song, S.Y., Pei, Y., and Hsiao-Wecksler, E.T. "Estimating Relative Angles Using Two Inertial Measurement Units Without Magnetometers". In: 22.20 (2022), pp. 19688–19699.

[87] Mahony, R., Hamel, T., and Pflimlin, J.-M. "Nonlinear Complementary Filters on the Special Orthogonal Group". In: 53.5 (2008), pp. 1203–1218.

[88] Barshan, B. and Durrant-Whyte, H. "Inertial navigation systems for mobile robots". In: 11.3 (1995), pp. 328–342.

[89] Ludwig, S.A. and Burnham, K.D. "Comparison of Euler Estimate using Extended Kalman Filter, Madgwick and Mahony on Quadcopter Flight Data". In: *Proc. ICUAS*. 2018, pp. 1236–1241.

[90] Madgwick, S., Harrison, A.J.L., and Vaidyanathan, R. "Estimation of IMU and MARG orientation using a gradient descent algorithm". In: *Proc. ICRR* (2011).

[91] Ludwig, S.A. "Optimization of Control Parameter for Filter Algorithms for Attitude and Heading Reference Systems". In: *Proc. IEEE CEC*. 2018.

[92] Ricci, L. "On the Orientation Error of IMU: Investigating Static and Dynamic Accuracy Targeting Human Motion." In: *PLoS One* 11 (2016).

[93] Inc., T.M. *Optimization Toolbox version: 9.4 (R2022b)*. Natick, Massachusetts, United States, 2022.

[94] Tang, L. et al. "Performance test of autonomous vehicle lidar sensors under different weather conditions". In: *Transportation research record* 2674.1 (2020), pp. 319–329.

[95] Technologies, U. *TextureFormat YUY2*. Unity Technologies. 2023.

[96] Ho, Y.-H. et al. "Learned Video Compression for YUV 4:2:0 Content Using Flow-based Conditional Inter-frame Coding". In: *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*. 2022, pp. 829–833.

[97] Thanh, V.T. and Urano, Y. "Mobile TCP socket for secure applications". In: *2010 The 12th International Conference on Advanced Communication Technology (ICACT)*. Vol. 2. 2010, pp. 971–974.

[98] Badia, L. "On the effect of feedback errors in Markov models for SR ARQ packet delays". In: *Proc. IEEE Globecom*. 2009.

[99] Hussain, R. and Zeadally, S. "Autonomous cars: Research results, issues, and future challenges". In: *IEEE Communications Surveys & Tutorials* 21.2 (2018), pp. 1275–1313.

[100] Lee, H. et al. "AVM/LiDAR sensor based lane marking detection method for automated driving on complex urban roads". In: *2017 IEEE Intelligent Vehicles Symposium (IV)*. IEEE. 2017, pp. 1434–1439.

[101] Yoo, H.W. et al. "MEMS-based lidar for autonomous driving". In: *e & i Elektrotechnik und Informationstechnik* (2018).

[102] Huang, J. et al. "6-DOF VR videos with a single 360-camera". In: *2017 IEEE Virtual Reality (VR)*. IEEE. 2017, pp. 37–44.

[103] Kuchenbecker, K.J., Fiene, J., and Niemeyer, G. "Improving contact realism through event-based haptic feedback". In: *IEEE transactions on visualization and computer graphics* 12.2 (2006), pp. 219–230.

[104] Caporusso, N., Mkrtchyan, L., and Badia, L. "A multimodal interface device for online board games designed for sight-impaired people". In: *IEEE Trans. Inf. Tech. Biomed.* 14.2 (2010), pp. 248–254.

[105] Cisotto, G. et al. "Classification of grasping tasks based on EEG-EMG coherence". In: *Proc. IEEE Healthcom*. 2018.

Appendix

.1 Example codes developed for the project

```
def construct_autonomous_drive_model():
    auto_drive_net = Sequential(name='Autonomous_Drive_Net')

    auto_drive_net.add(Conv2D(24, (5, 5), strides=(2, 2), input_shape
    =(66, 200, 3), activation='elu'))
    auto_drive_net.add(Conv2D(36, (5, 5), strides=(2, 2), activation=
    'elu'))
    auto_drive_net.add(Conv2D(48, (5, 5), strides=(2, 2), activation=
    'elu'))
    auto_drive_net.add(Conv2D(64, (3, 3), activation='elu'))

    auto_drive_net.add(Conv2D(64, (3, 3), activation='elu'))
    auto_drive_net.add(Flatten())
    auto_drive_net.add(Dense(100, activation='elu'))
    auto_drive_net.add(Dense(50, activation='elu'))
    auto_drive_net.add(Dense(10, activation='elu'))

    auto_drive_net.add(Dense(1))
    return auto_drive_net
```

Code 1: Implemented code based on Nvidia Network Model

```
class ProportionalIntegralDerivativeController:
    def __init__(self, p_coeff=0.3, i_coeff=2.2, d_coeff=0.0025):
        self.proportional = p_coeff
        self.integral = i_coeff
        self.derivative = d_coeff

        self.measured_value = 0
        self.iteration = 1

```

```
        self.err_value = 0
        self.delta_err = 0

    def proportional_term(self):
        return round(self.proportional * self.err_value, 4)

    def integral_term(self):
        return round(self.integral * self.delta_err, 4)

    def derivative_term(self, prev_error):
        return round(self.derivative * prev_error, 4)

    def pid_update(self, desired_val):
        control_action = 0
        self.err_value = desired_val - self.measured_value
        control_action += self.proportional_term()

        self.delta_err += self.err_value
        control_action += self.integral_term()

        temp_err_rate = self.delta_err / self.iteration
        control_action += self.derivative_term(temp_err_rate)

        self.measured_value += control_action
        self.iteration += 1

    def get_measured_value(self):
        return self.measured_value

    def __str__(self):
        return "Measured Value = {}, Error Value = {}, Iteration = {}
    ".format(
            self.measured_value, self.err_value, self.iteration
            )
```

Code 2: ProportionalIntegralDerivative Controller