INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS CIUDAD DE MÉXICO

SCHOOL OF ENGINEERING AND SCIENCE

DESIGN OF A PROPRIETARY SELF DRIVING CAR PLATFORM AND DEVELOPMENT OF AUTONOMOUS DRIVING ALGORITHMS BASED ON COMPUTATIONAL VISION AND DEEP NEURAL NETWORKS.

A DISSERTATION PRESENTED BY

ALDO IVÁN AGUILAR ALDECOA

SUBMITTED TO THE
SCHOOL OF ENGINEERING AND SCIENCES
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

ENGINEERING SCIENCE

Mexico City, June 08 2021

Dedication

A mi madre María de Jesús Aldecoa Camacho, quien ha sido siempre mi mayor inspiración. A mi padre Juan Aguilar Hernandez, el pilar más importante en mi vida. A mi hermano Juan Aguilar Aldecoa, mi más grande motivación y el mejor compañero. Ustedes han sido el motor más importante para mi vida profesional y personal, sin su apoyo este trabajo no hubiera sido posible.

Los quiero.

Acknowledgments

Gracias a mis padres por el apoyo incondicional que me han brindado siempre. Por ser los más grandes modelos de éxito para mi y para mi hermano. Gracias por los valores que han sembrado en mi y que me han permitido lograr este y muchos exitos más. Siempre estaré eternamente agradecido.

Gracias a mis profesores y principalmente a mi asesor el Dr. Alejandro Aceves López, por su enorme apoyo, por compartir sus conocimientos, por su gran paciencia y por la siempre oportuna y enriquecedora guía y retroalimentación que me permitieron culimar este proyecto exitosamente.

A Alfonso, Fer y Ro que formaron parte fundamental durante el desarrollo de este proyecto. Gracias por su apoyo, comprensión y por el tiempo brindado, pero principalmente gracias por su amistad.

Gracias a Jorge, Sam y Lau por formar parte de un exito más en mi vida y por ser parte de mi familia. Por confiar siempre en mi y motivarme a alcanzar mis sueños y poder compartirlos con ustedes.

¡Muchas gracias!

Design of a Proprietary Self Driving Car Platform and Development of Autonomous Driving Algorithms based on Computational Vision and Deep Neural Networks.

by

Aldo Iván Aguilar Aldecoa

Abstract

This research project presents a detailed description of the design and development of the first small-sized self driving development platform at the ITESM Campus Estado de México. The implemented hardware and software is presented based on the state of the art research platforms. Additionally, the required sensor and instrumentation implementation is described as well as the platform's mechanic and electric design. Moreover, the dynamic identification of the vehicle actuators is presented for the linear velocity control of the platform however no clear evidence of a linear dynamic behavior could be identified, leading to the implementation of a herustically tuned PI velocity control system.

Specific Computer Vision (grayscale color thresholding) and Deep Neural Network (U-Net semantic segmentation) based road lane segmentation mechanisms were developed, tested and validated. These segmentation mechanisms served as the main input of the final autonomous driving system proposal based on a road lane following strategy. Finally, a well-defined autonomous driving performance evaluation methodology is described and implemented to compare the proposed systems response, identifying comparable performances between CV and DNN segmentation systems.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Software simulations are widely used for Autonomous Vehicle (AV) technology investigation, depending mainly on appropriate modeling of real world scenarios [3] [4] [5] [6]. However this strategy limits the application of self-driving technologies in real life experiments, limiting the validation, repeatability and robustness of the developed systems in real life [7]. Moreover, real-scale vehicles require particularly important space, security and high investments, limiting the accessibility for multiple researchers and students around the world.

Small-sized self driving cars have permitted to establish a realistic testing and validation environment offering a simple, trustworthy, secure and cheap alternative for complex autonomous systems deployment [8]. Furthermore, small-sized vehicles present important use cases for performance analysis of algorithms, validation of maneuver protocols and investigation of advantages and drawbacks of complex driving protocols at traffic.

With the emergence of smaller and more efficient processing units, highly capable development platforms have taken great relevance for research communities. This improved platforms have offered many applications for scaled vehicles such as trajectory planning, route mapping and security capabilities [9]. Moreover a great interest in the development of AV systems on small-sized vehicles has emerged, allowing to test, compare, validate and develop the state of the art AV technologies such as Computational Vision (CV) and Deep Neural Network (DNN) Algorithms.

Multiple small-sized self driving platform proposals have emerged in recent years product of the rapidly growing research interest on autonomous vehicles [10] [11] [12] [13] [14] [15] [16], being the design and implementation of these platforms a highly relevant research area itself. These AV systems depend on an adequate hardware and software development, aided mainly by the usage of specific sensors such as ultrasonic and infrared sensors, vision-based sensors, wheel encoders, monocular cameras, application boards (Debian-based Linux ARM board) and software/hardware interface boards (sensors and actuators management and interfacing) [8].

Some of the most relevant small-sized AV development platforms include open source proposals such as the F1TENTH autonomous racing platform [10], Berkeley Autonomous Race Car (BARC) [12], AutoRally [15], MIT RACECAR [13], the Multi-agent System for non-

Holonomic Racing (MuSHR) and other commercial alternatives such as the AWS DeepRacer [14] or the Donkey Car [17]. However, the implementation of these open source systems is limited in some manner, requiring the acquisition of specific hardware and software as well as the actual manufacturing of the platform, while commercial alternatives such as the AWS DeepRacer present an easier way to start yet high provider dependency and more expensive acquisition process by directly buying the platform.

On one hand, the aforementioned proposals motivated the design and deployment of a proprietary platform based on the state of the art system architectures and capabilities, and using the resources available at Campus. The goal was to define the first small-sized self driving development platform at the ITESM Campus Estado de México that could serve as a baseline for further AV research, allowing the testing and development of multiple autonomous driving mechanisms.

On the other hand, it was required to count with an adequate hardware and software architecture design for the correct vehicle instrumentation and efficient velocity and directional control. The project central scope was to develop a road lane follower capable of maintaining an autonomous performance while driving inside a delimited track. More importantly, the proposed platform would be utilized for the implementation of specific CV and DNN segmentation and road lane following algorithms as a manner of comparing the vehicle's autonomous capabilities and establishing an adequate performance evaluation methodology.

Some of the achievements and contributions provided by this research project include the implementation of a structured and well designed small-sized AV platform that could serve for future autonomous driving research validation and testing. Moreover, the system provides adequate hardware and software support for the implementation of relevant AV algorithms based on the state of the art development platform proposals as the aforementioned. Additionally, a well-defined design and implementation procedure is described for further integration of similar platforms as well as future required updates or modifications. Finally, the platform served as a baseline for the integration and comparison of specific CV and DNN strategies permitting to demonstrate the autonomous capabilities the system can provide at specific testing conditions and acting as the first AV research proprietary platform in the Campus.

This document firstly presents the design and construction of the developed platform in Chapter 2, including a deep description of the implemented hardware and software as well as the mechanical and electrical requirements of the system. The platform's linear velocity control system is also described in Chapter 3, including the dynamic identification of the vehicle actuators and the automatic actuators' control signal generation. Finally, Chapter 4 and Chapter 5 include the implemented road lane segmentation and identification systems description based on state of the art CV and DNN techniques, including a complete definition of the methodology used for their development, to finally evaluate the autonomous driving response achieved by comparing the performance of each system proposal.

Chapter 2

Platform Design and Implementation

2.1 Hardware and Software Description

There exist multiple projects focused on the development of autonomous driving in small-sized cars [9][18][19][8][20]. These approaches have impulsed the research for a rapidly growing field of autonomous vehicles (AVs). Small-sized AVs present affordable and high-performance systems capable of implementing the state of the art autonomous driving algorithms [7][21]. A variety of small-sized cars for AVs research have been developed, establishing well-defined state of the art hardware and software architectures [11] [22] [10][23], and that is also the case for this platform developed at ITESM Campus Estado de México, shown here in Figure 2.1.

Figure 2.1: Developed Small-Sized AV Platform.

The design of this small-sized car was restricted to use the available hardware resources at Campus displayed in Figure 2.2 including: Turnigy Trooper SCT-X4 1/10 4x4 Nitro Course Truck [24] , Turnigy RC ON/OFF Switch [25], Turnigy Radio Control Transceiver + Receiver [26], Arduino Nano microcontrollers [27], Odroid XU4 Development board [28], NVIDIA Jetson Nano Development Kit board [29], Intel RealSense SR300 BlasterX Senz3D Depth Camera [30], RPLidar A2 sensor [31], Turnigy 2200mAh 2S LiPo battery [32], Matek PDB-XT60 Power Distribution Board [33], Pololu D24V90F5 5V@9A Step-Down Voltage Regulators [34].

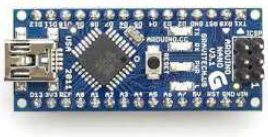
(a) Turnigy Trooper SCT-X4 1/10 4x4 Nitro Course Truck.

(b) Turnigy RC Transceiver + Receiver.

(c) Turnigy RC ON/OFF Switch.

(d) Odroid XU4 Development board.

(e) NVIDIA Jetson Nano Developer Kit board .



(f) Arduino Nano.

(g) Intel RealSense SR300 BlasterX Senz3D Depth Camera.

(h) RPLidar A2 sensor.

Figure 2.2: Platform's hardware baseline.

The Turnigy Trooper SCT-X4 1/10 4x4 Nitro Course Truck [24] served as the vehicle's primary mechanical baseline. It is a 1/10 high-speed small-sized vehicle counting with a PWM 60A Electronic Speed Controller (ESC), a 2080kV Sensor-Less Brushless DC Motor (BLDC) and a PWM high torque directional Servo Motor. The Turnigy Radio Control Transceiver + Receiver and Turnigy RC ON/OFF Switch modules [26][25] were required to select between autonomous or manual mode, enabling a human operator to override possible faulty computer control and manually stop the car. In fact, this hardware turned into a great advantage when conducting the experiments on identification and automation.

Both PWM signals to control speed and steering in the car are generated by a robust PWM signal generation built with an Arduino Nano board. The odometry system was built with another Arduino Nano [27] and four Infrared sensors (IR). These microcontrollers served as the low-level control units. The Odroid XU4 board [28] and the NVIDIA Jetson Nano [29] are responsible for the high-level control. Specifically, the NVIDIA Jetson Nano acquires incoming images from the camera and processes them, while the Odroid XU4 is responsible for the vehicle's linear velocity and steering control interfacing with the vehicle low level control units.

The environment perception was achieved with a Intel RealSense SR300 BlasterX Senz3D Depth Camera [30] and a RPLidar A2 sensor [31], allowing the acquisition of RGB images of the vehicle's front view and relevant distance and depth information. Figure 2.3 shows a block diagram of the overall architecture. The vehicle's software development was based on a distributed subsystem architecture. The implemented subsystems include (a) the vehicle's linear velocity and directional control, (b) low-level management of actuators, and (d) vehicle's environment perception. The Robot Operating System (ROS) [35] was used as the central software integration platform. ROS is one of the leading software frameworks used in the AVs research area.

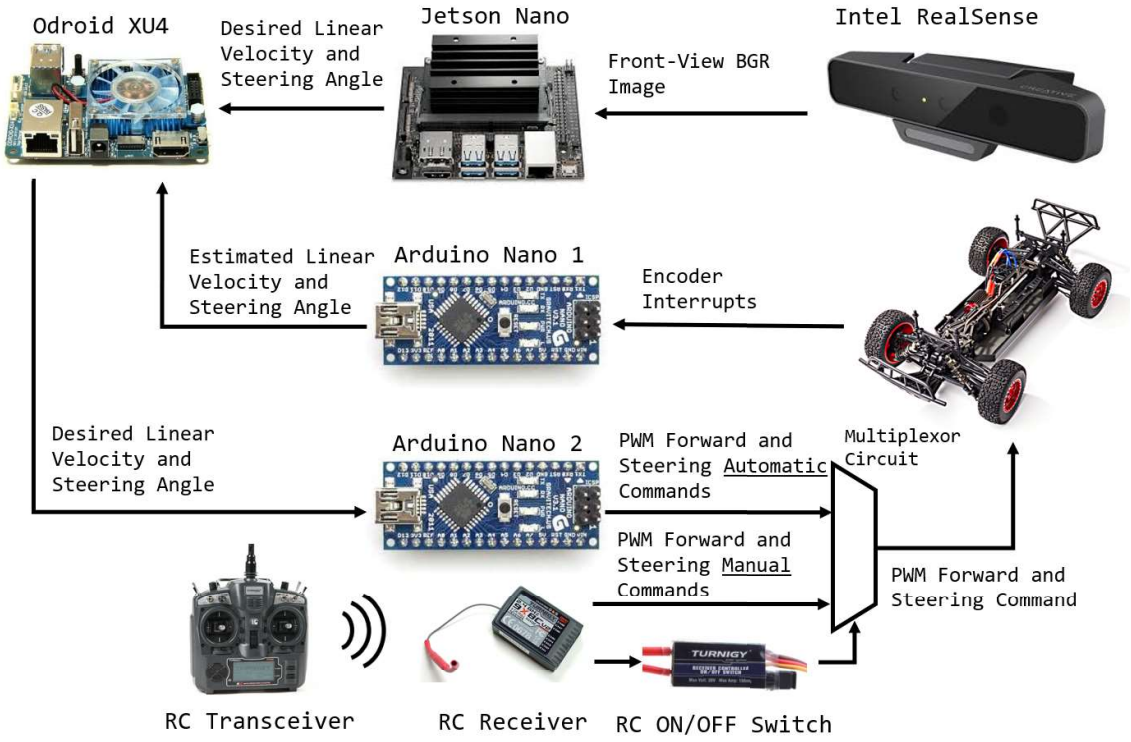


Figure 2.3: Block Diagram of the system architecture.

It was required to install the software for the Intel RealSense Depth Camera and RPLidar sensor. These sensors interact directly with the Jetson Nano board as part of the environment perception subsystem. The software installation procedure was based on various sources online [36] [37] [38]. Specific modifications on the NVIDIA Jetson Nano OS Kernel were needed for the IntelRealSense Camera. The overall installation procedure is described in the project's GitHub repository [39]. The Arduino Command Line Interface (CLI) was adopted to help low-level coding and optimize performance [40]. Some ROS scripts were developed to automatically upload and test the microcontrollers' code with the Arduino CLI tool. Furthermore, TensorFlow [41] and OpenCV [42] were used as the software development libraries for the implemented Computer Vision (CV) and Deep Neural Network (DNN) algorithms. In fact, TensorFlow and OpenCV are commonly used all around the world in the development and research of autonomous driving systems for small-sized AV.

2.2 Mechanical and Electrical Design

A mechanic montage for the small-sized car was designed and built, as displayed in Figures 2.4, consisting of three main parts (a) the 1/10 RC car chassis, (b) a laser-cut 3mm thick acrylic plate where all hardware was placed, and (c) a second small acrylic plate for the Intel Depth Camera. It was a requirement to mount all the platform components without altering the car chassis. This constraint was taken into account for the overall montage design. The plates' CADs, as displayed in Figures 2.9 and 2.10, considered the vehicle's shape and size and those of the implemented hardware. All the components were placed on the acrylic plates achieving an easy to mount structure. The small plate was attached to the main one with two metallic posts, assuring an adequate camera placement. While the large plate was mounted on the car using four easily removable plastic pillars directly attached to the chassis, allowing a simple overall montage.

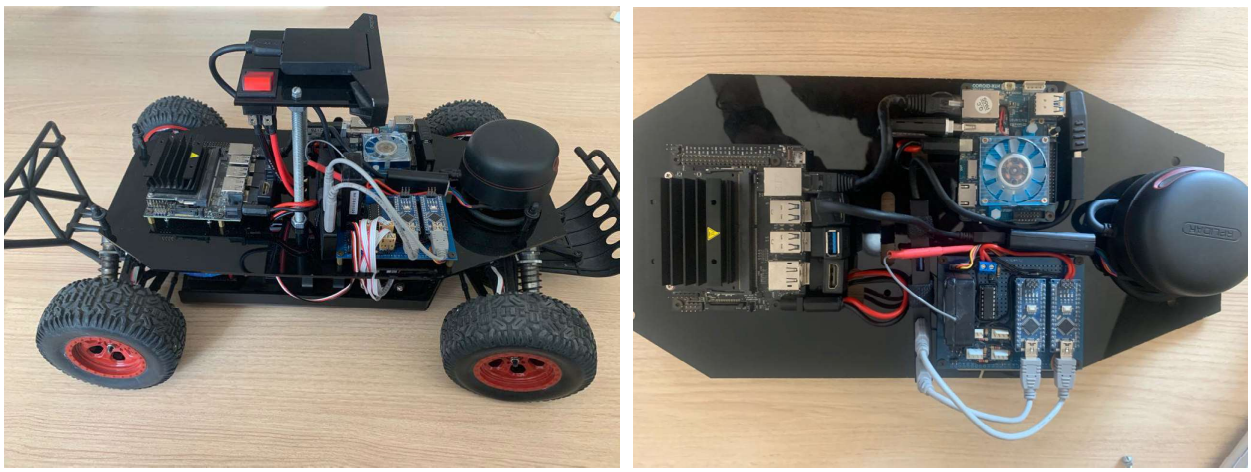


Figure 2.4: Platform mechanical montage.

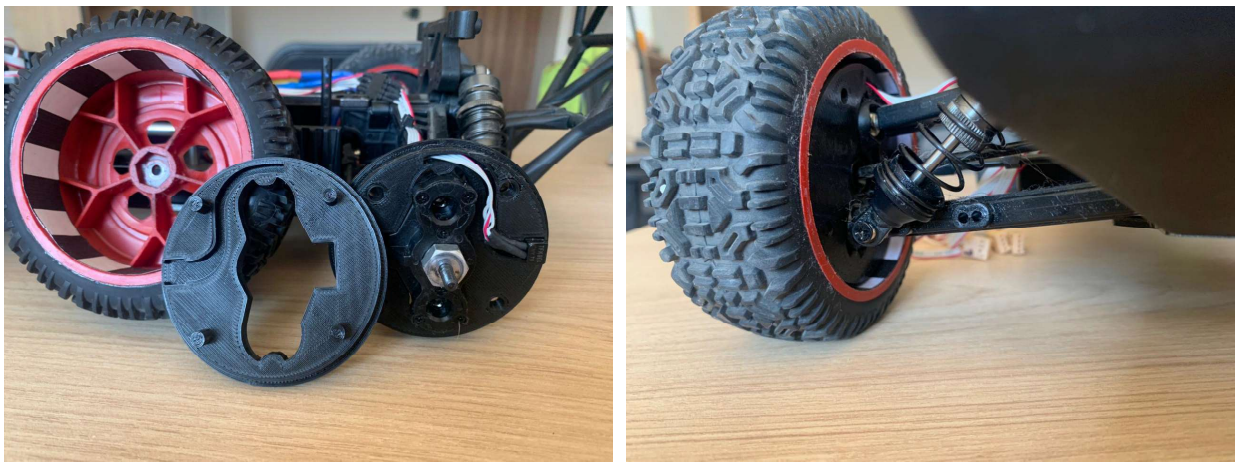


Figure 2.5: Implemented odometry system.

An adequate odometry system was primordial for the linear velocity control and autonomous driving of the AV platform. The hardware baseline did not count with a pre-existent odometry system. Therefore, an odometry system similar to those presented in [43] and [44] was designed and built. This system consists on four infrared sensors (IR) connected to an Arduino Nano board, each of them mounted on the wheel's shaft along with a dark-clear segmented film as it is shown in Figure 2.5. Accordingly, it was needed to attach the IR sensors to the car chassis mechanically without altering it. The mounts specifications included (a) acquiring a clean IR lecture by isolating the sensor to the light, (b) the mounts must be attached to the wheels axles without breaking or modifying them, and (c) they would be 3D printed. Figures 2.11 and 2.12, show the implemented disk-like mounting geometry and its dimensions. The overall design permitted a mechanically stable attachment of the sensors to car's wheels.

Besides, a PCB was designed for the integration of the developed electronics. The platform electronics consisted of (a) the vehicle's remote operation mode selection and driving override system aided by a Turnigy RC ON/OFF switch, (b) IR sensors power supply and output acquisition, (c) two Arduino Nano boards, one for actuator's control and the other for odometry system and (d) remote RC data retrieval via the Turnigy RC Receiver. The PCB implementation considered the space limitations of the platforms' mount, a clean montage of the electronic components, as well as the actuators' power and interfacing requirements.

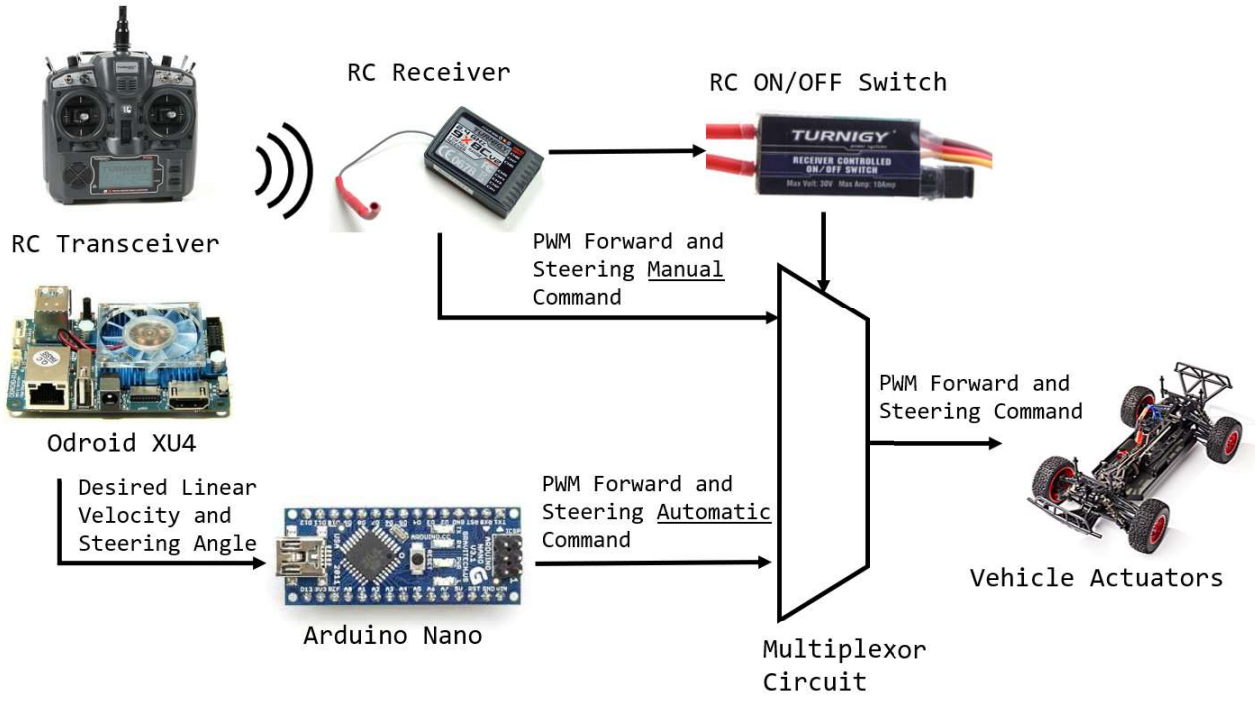
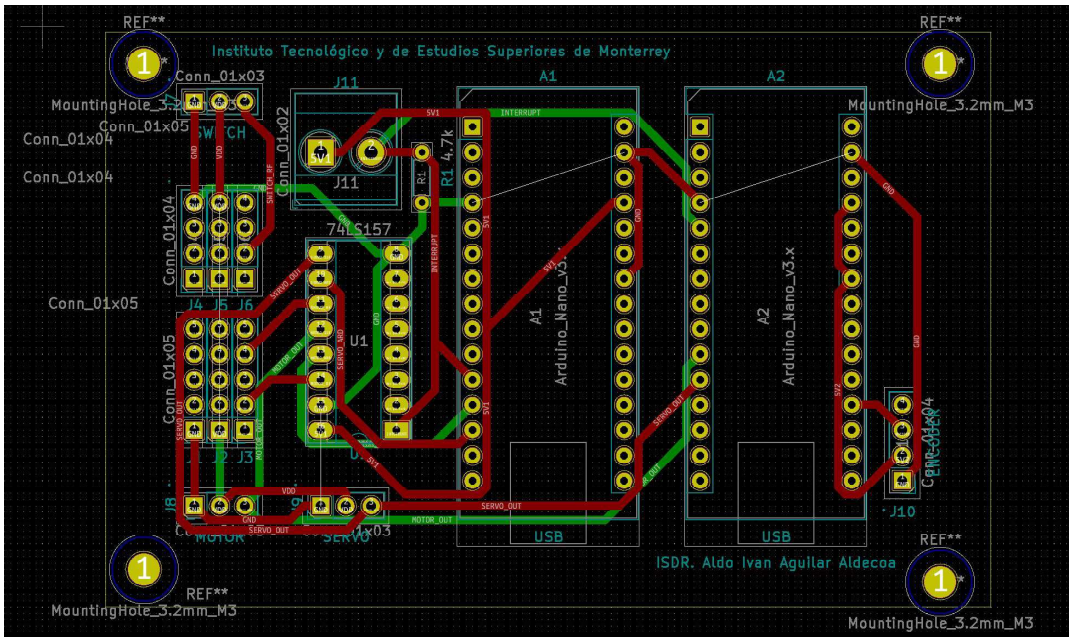
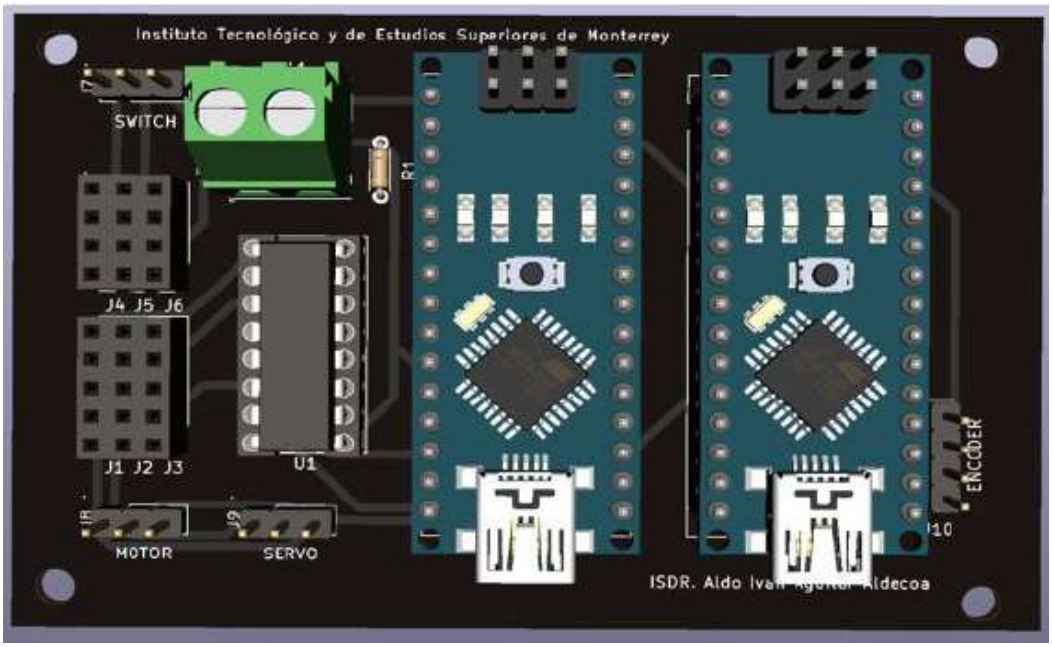


Figure 2.6: Block diagram of the override system.



(a) PCB Design Diagram



(b) Render of the implemented PCB

Figure 2.7: Implemented PCB design.

The vehicle's override system is important for a safe and controlled algorithms development, allowing a rapid alternation between the autonomous and manual driving operations, see Figure 2.6. This emergency system was essential for preventing any crash danger or non-desirable behavior of the platform. Specific electronics were designed for the selector circuit, multiplexing between the autonomously generated control signals and those obtained by the manual control from the RC Receiver. The override system allowed an efficient remote operation selection based on the Turnigy RC ON/OFF Switch [25] and the designed electronics.

The Turnigy Trooper SCT-X4 1/104x4 Nitro Course Truck [24] has to be powered with a LiPo battery of 2 (or 3) cells, accepting voltages from 7.4 to 12 V. Operational tests were developed with both type of batteries. Even though 3 cells batteries allowed higher speeds, reasonable low speeds could not be easily achieved. Whereas, for 2 cells batteries, the platform achives lower speeds, making easier the development of the autonomous driving algorithms. This behavior led to select a 2 cells 7.4 V Turnigy 2200mAh LiPo battery [32] as the vehicle's main power supply.

Furthermore, the implemented hardware (such as peripherals, single-board computers, and microcontrollers) required a 5 V power supply with a 4 A total current consumption. This electric power requirement was obtained from the mentioned 2 cell LiPo battery. Furthermore, a Y-connector was necessary followed by a Matek PDB-XT60 Power Distribution Board (PDB) [33]. Moreover, a pair of Pololu D24V90F5 5V@9A step-down voltage regulators [34] were used to assure correct power supply to main Odroid and Jetson boards. Figure 2.8 shows a block diagram of all those components interconnected.

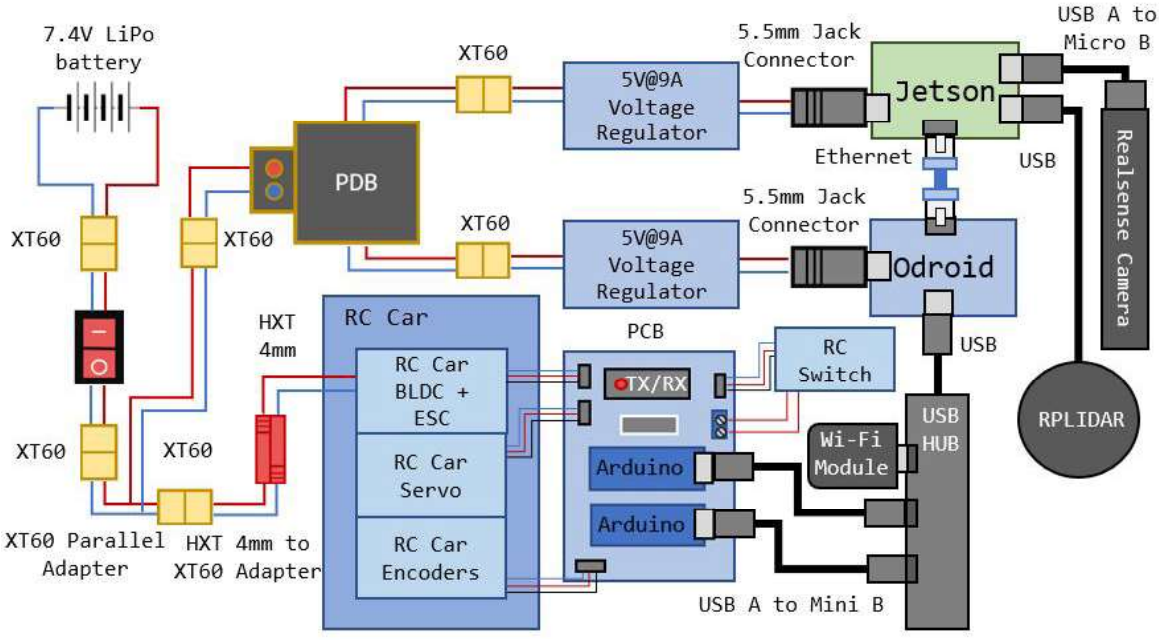


Figure 2.8: Block Diagram of the platform's electric system.

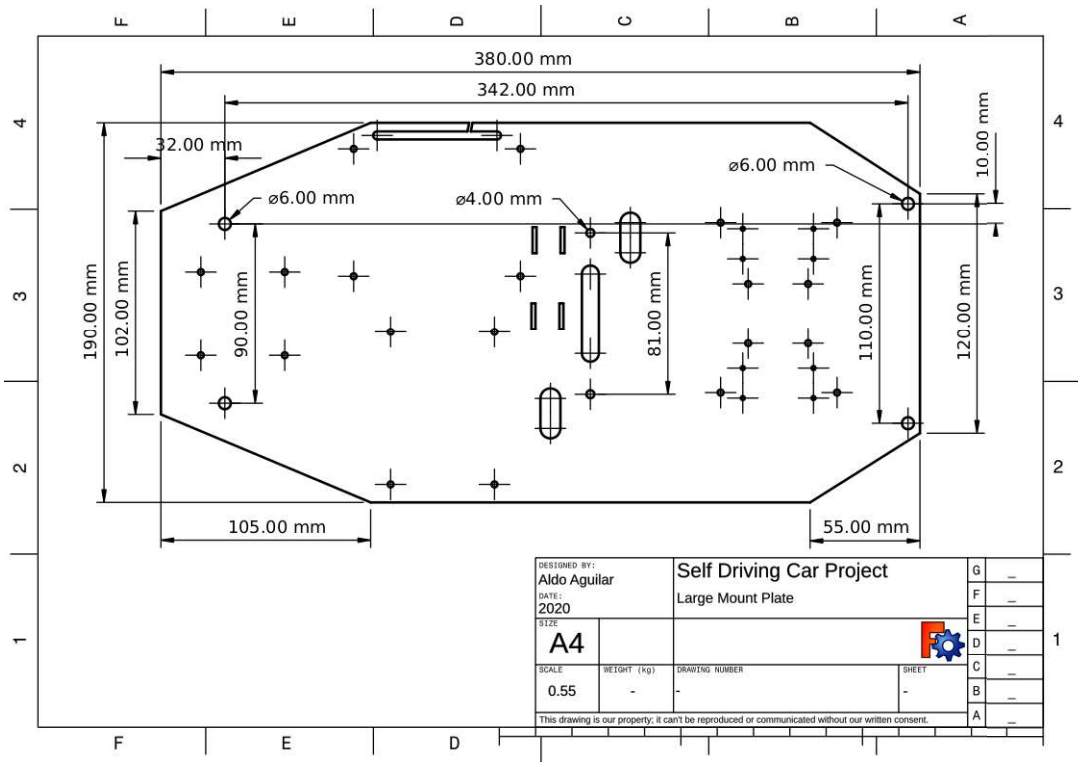


Figure 2.9: Large Mount plate design.

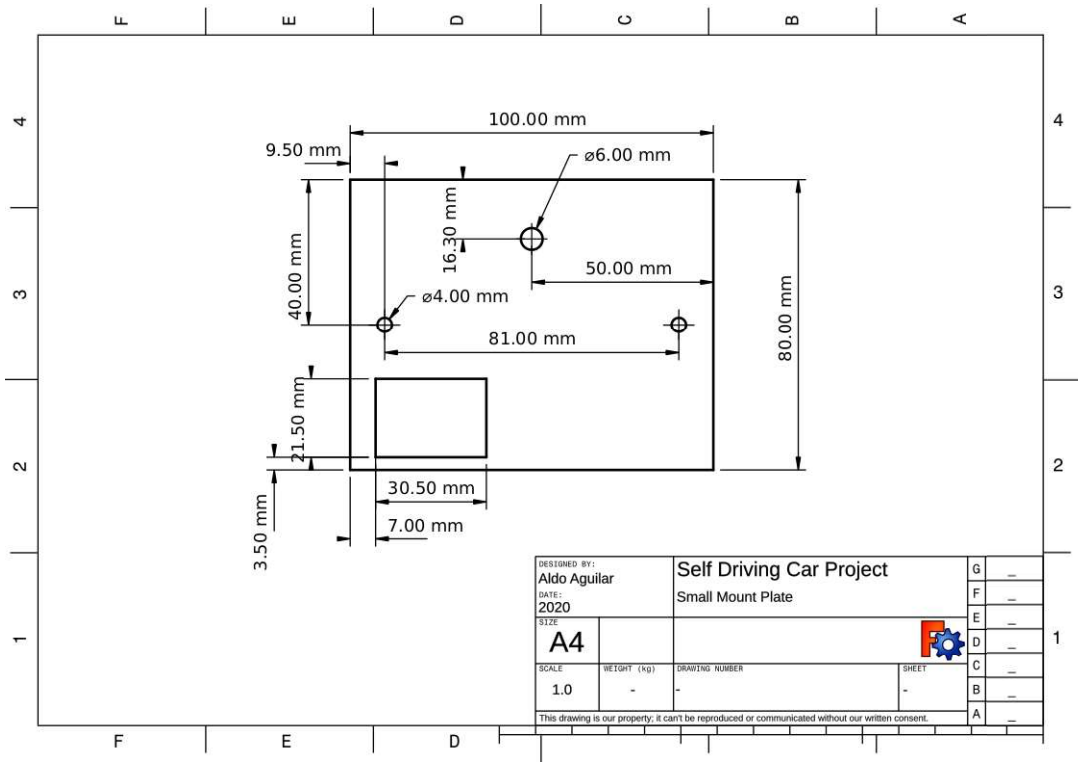


Figure 2.10: Small mount plate design.

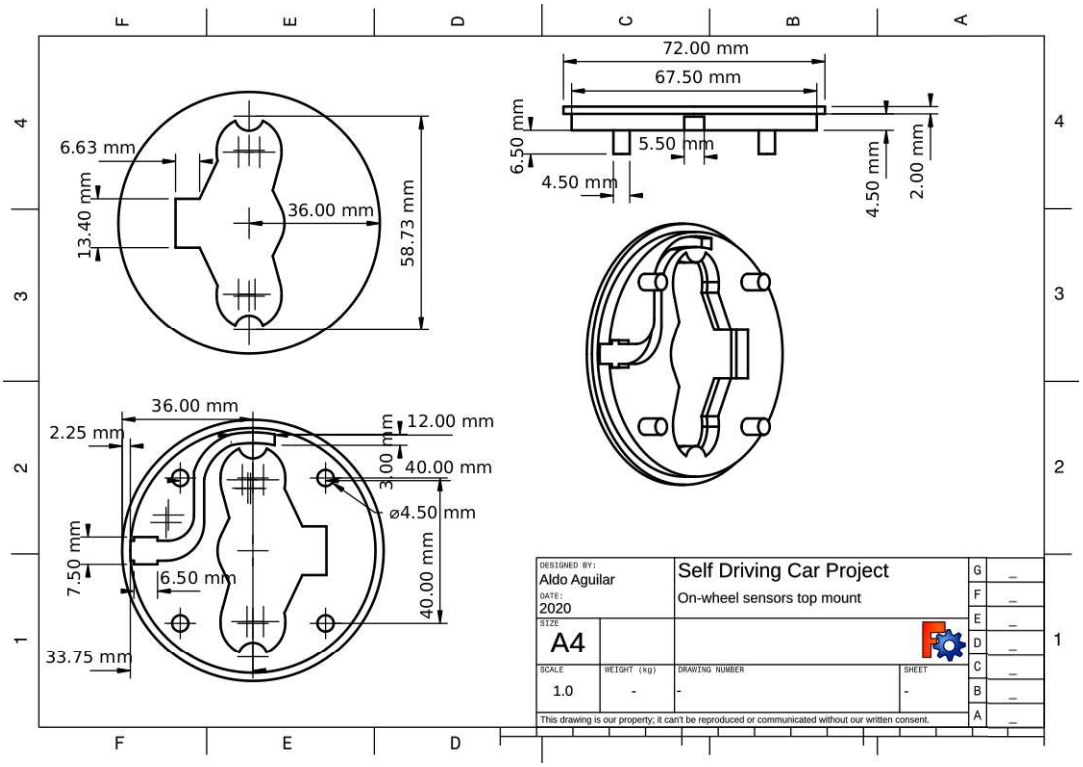


Figure 2.11: On-wheel IR sensor mount top part CAD.

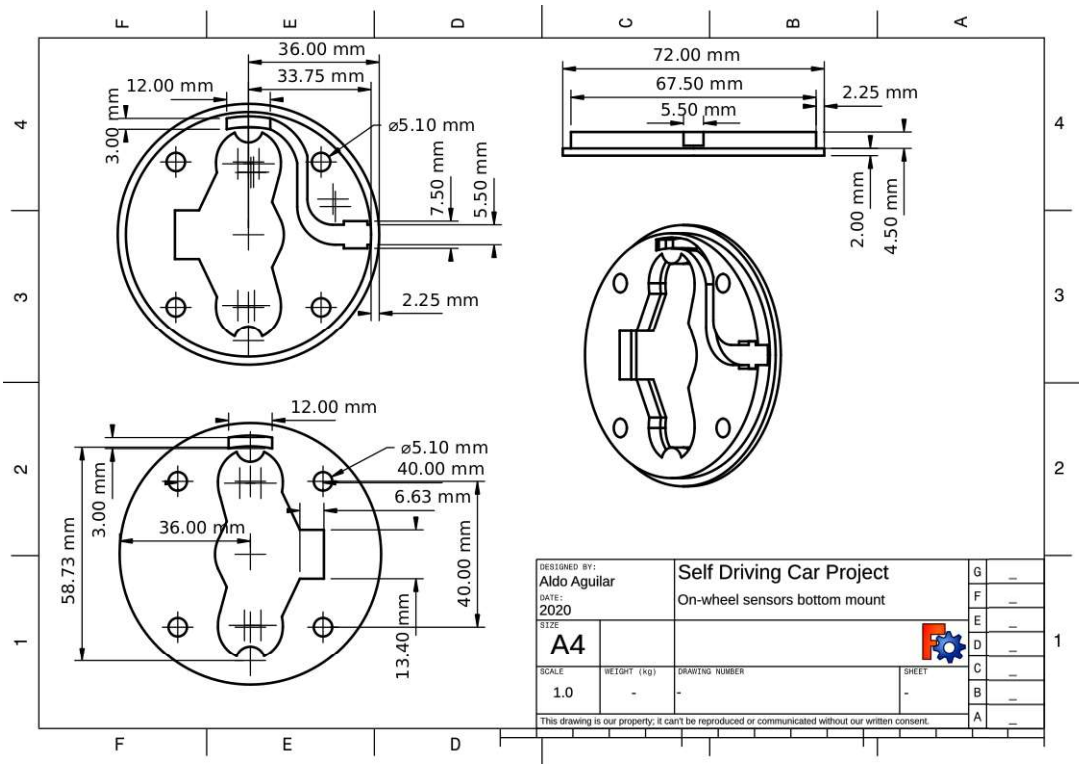


Figure 2.12: On-wheel IR sensor mount bottom part CAD.

2.3 Network Architecture

In order to achieve a distributed processing among Odroid, Jetson and operator computer (PC), a network design was necessary. ROS utilizes TCP socket based communication among its nodes, demanding the implementation of a well-established network architecture. Direct cable network connection between the Odoid XU4 and the NVIDIA Jetson Nano board was required to enhance communication speed. Moreover, to communicate with the vehicle at any time, a WiFi remote network was established. This WiFi network will allow the development, debugging and data transferring between the platform and a remote PC. The final architecture, as displayed in Figure 2.13, consists on a pair of networks, (a) an Ethernet-based LAN between the vehicle-single board computers and (b) a WLAN between the Odroid XU4 and a remote PC through a router.

In order to get direct access to Jetson from PC, it was necessary to configure the Odroid XU4 to mount a network sharing strategy by configuring it as a network routing device. A way to do that can be read in [45]. By configuring a routing device, all traffic could be forwarded from one network to another. This arrangement allows the remote PC to establish direct communication with the Jetson Nano board and vice-versa through the Odroid XU4. Though, specific configurations had to be set in the end devices to define how they could reach a different network by adding static routes, achieving successful communication among all computers. Finally, ROS Networking configurations [46] were set to deploy adequate bi-directional connectivity of the nodes used, by establishing the Odoid XU4 as the ROS master node in all the end-devices, assuring complete overall connectivity. An in-depth explanation of the network configuration can be reviewed in the project's GitHub repository [39].

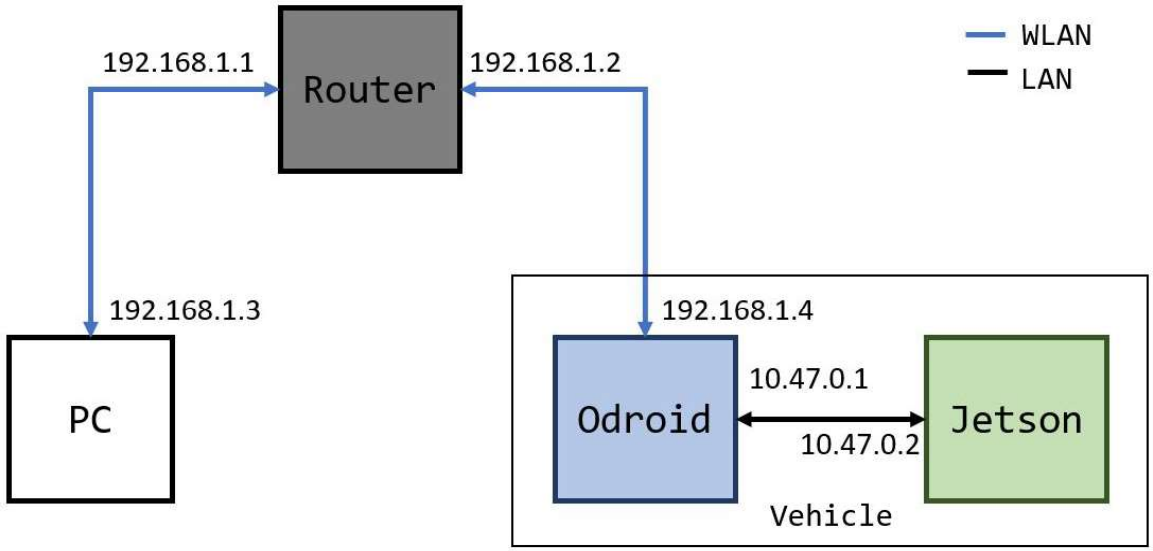


Figure 2.13: Implemented network architecture.

2.4 Software Architecture in ROS

As aforementioned, the vehicle utilizes a distributed software strategy. This structure is based on the interaction of dedicated subsystems. ROS was selected as the central software integration platform. The designed ROS architecture for this project is displayed in Figure 2.14, where all subsystems could interact adequately [47] [48]. The Odroid XU4 is responsible of low-level linear velocity and directional control, while the Jetson Nano board manages the vehicle's environment perception. Accordingly, each subsystem was defined as a specific ROS node, considering the network characteristics previously mentioned. The ROS nodes included the `self_driving_car_node` and the `environment_percept_node`.

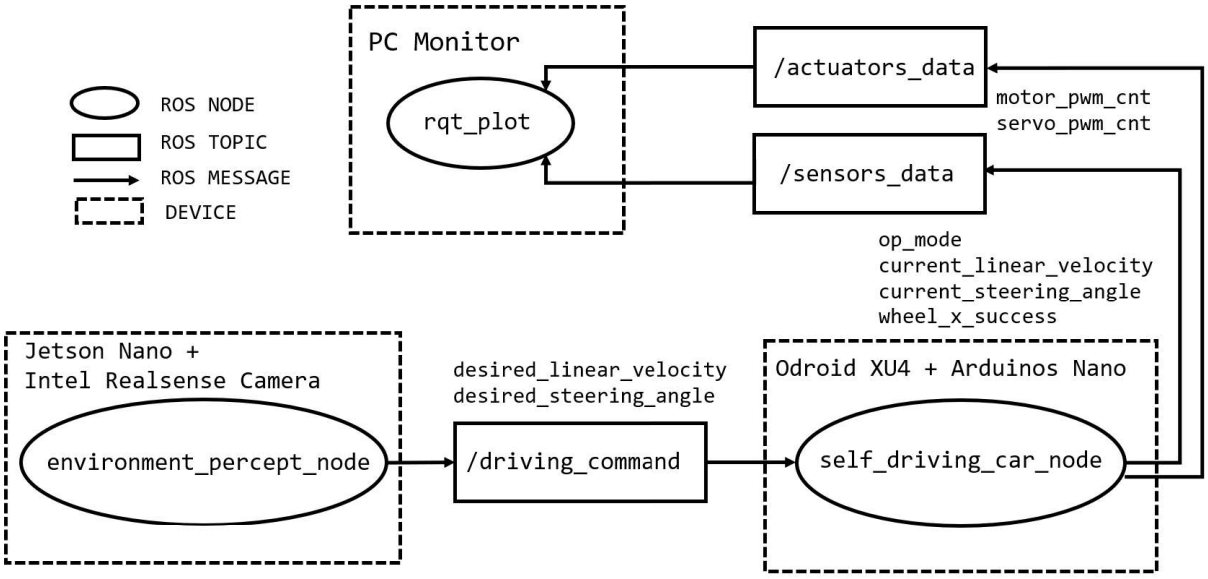


Figure 2.14: Implemented software architecture in ROS.

The `self_driving_car_node` is also responsible for the translation from low-level control to high-level supervision of the platform. It utilizes the `/actuators_data` and `/sensors_data` topics to publish current state information such as the vehicle's operational mode, current speed and directional angle, wheel encoder functional state, and current actuators' PWM High Time supplied. It also subscribes to the `/driving_command` topic to receive the platform's desired speed and directional angle to keep car inside the track.

The `self_driving_car_node` receives relevant data from a dedicated Arduino Nano board regarding the current actuators and odometry system states. Also, the `self_driving_car_node` transmits the desired platform speed and directional states to a second Arduino microcontroller responsable to achieve such desired autonomous driving behavior.

The `environment_percept_node` is responsible for the environment perception data acquisition and processing. It utilizes the acquired image from Intel RealSense Depth Camera, analyzes it, generates the desired speed and directional correction to keep car on track, and publish them to the `/driving_command` topic. This node executes the designed DNN and CV based algorithms internally to recognize the current road state and generates the desired platform's speed and directional angle.

It can be noticed that the implemented architecture allows having a well-defined remote monitoring system. This is achieved by the network configuration made and the `/actuators_data` and `/sensors_data` topics. A remote PC is subscribed to the mentioned topics to visually analyze relevant platform data using the ROS `/rqt_plot` node.

Chapter 3

Dynamic Model Identification and Linear Velocity Control

3.1 BLDC Motor and ESC Pairing Behavior

As aforementioned, the Turnigy Trooper SCT-X4 1/10 4x4 Nitro Course Truck is integrated with a 60A Electronic Speed Controller (ESC) and a 2080kV sensor-less Brushless DC (BLDC) Motor [24]. The ESC module is responsible for providing the necessary polarization sequence required to rotate the BLDC motor at a desired speed. In general, the ESC translates a PWM input into a tri-line voltage polarization sequence. This output is directly fed to the motor control lines to produce a rotational motion on the car's wheels.

To have adequate control of the speed, the ESC module must recognize the motor coils' position to produce the required polarization sequence. However, sensor-less motors do not count with a mechanism capable of providing this positional information directly. Nonetheless, the ESC can still recognize the motor's rotational condition aided by the motor's counter-electromotive voltage produced while it is rotting. This signal is produced by any motor's non-polarized coil when a rotational movement has been achieved by polarizing the rest of the coils. This characteristic of a sensor-less BLDC motor allows establishing a sufficient positional recognition mechanism for its further speed control [49].

Figure 3.1: Implemented BLDC Motor + ESC system. [1]

The induced output voltage facilitates synchronizing the tri-line polarization sequence and achieving specific vehicle speeds. However, this speed control is limited, as an initial motion must be achieved to have a confident positional detection. This constraint takes great relevance for the BLDC motor starting sequence when the vehicle is stopped. Starting from a still condition, no counter-electromotive voltage is induced, and the ESC ignores the motor's positional condition [49]. When this situation happens, the ESC output depends fully on the input PWM signal.

Therefore, the voltage polarization sequence frequency varies based on the provided PWM High-Time until a constant motion is produced and the ESC synchronization can be achieved. Finally, when BLDC motor has already started, its speed can be easily adjusted by varying the PWM High Time inputted to the ESC. It is then necessary to acknowledge the startup complications of the couple ESC-motor. As a result, the car can exhibit a slightly different behavior every time when it starts moving from zero-velocity. Moreover, constant car's forward speed is not guaranteed with this ESC controller at certain low speed conditions, requiring an external speed control.

3.2 Linear Velocity Estimation

An adequate linear velocity estimation is essential to achieve vehicle's linear velocity control. As mentioned, the car does not count with any direct speed measurement system. This limitation led to the design of a robust odometry system based on a set of four CNY70 IR sensors and segmented dark-clear films mounted on each wheel. The design allows detecting the rate of change of the perceived film pattern as the vehicle moves. While the wheels rotate, the IR sensors emit a pulse each time they are facing a film's dark-segment [50][51]. Each film counts with a total number of 20 dark-clear segments.

Every IR output is connected to an Arduino's External Interrupt pin [52]. Each interruption modifies an internal counter CNT_i which stores the number of pulses that have occurred in a fixed period of time s. Given that the total number of pulses per turn is known ENC_CNT, the actual rotational speed w_i of each wheel can be easily calculated. Finally, the vehicle's linear velocity V is estimated from the average wheels' rotational speeds and wheel radius r using the following equations:

$$V = \frac{\omega_1 + \omega_2 + \omega_3 + \omega_4}{4} * r \tag{3.1}$$

$$\omega_i = 2\pi * \frac{CNT_i}{ENC\_CNT * s} \tag{3.2}$$

Where:

V = Vehicle's linear velocity (cm / s)
ω_i = Wheel angular velocity (rads / s);
r = Wheel radius (cm) = **5.25 cm**;
CNT_i = Encoder Counts;
ENC_CNT = Total number of counts per turn = **20**;
s = Period of time in seconds (s).

As stated in Equations (3.1) and (3.2), the vehicles' linear velocity can be successfully estimated given the wheels' angular velocity and radius. However, the wheels' angular velocity resolution depends on the total number of dark-clear segments on the film. Considering a fixed sampling period of 200 ms, the wheels' angular velocity can be estimated as follows:

$$\omega = 2\pi \times \frac{CNT}{20 \times 0.2s} \tag{3.3}$$

$$\omega = \frac{\pi}{2} \times CNT \tag{3.4}$$

Substituting Equation (3.4) in (3.1) and considering a wheel radius of 5.25 cm:

$$V \approx 8.2467 \times CNT \tag{3.5}$$

As shown in (3.5), the minimum linear acceleration perceived by the implemented system is 8.2467 cm/s. Therefore, if a single encoder count is detected every 200 ms, that is 5 Hz sampling frequency, the estimated linear velocity will be equal to 8.2467 cm/s, meaning that no speed lower than this minimum value will be perceived. In conclusion, the linear velocity estimations would be limited in quantiles of Q = 8.2467 cm/s. This will turn in a highly relevant characteristic for the further linear velocity control implementation.

Furthermore, a full-speed experiment in a long-lane was executed to detect the maximum linear velocity that could be achieved with the car, that was 726 cm/s. Therefore the quantile Q=8.2467 cm/s represents the 1.136% of that maximum perceivable speed. This signified that the achieved quantile offers a just-enough resolution of 88 different possible levels of car's estimated speed.

As shown in Equation (3.3), the selected quantile depends inversely on the sampling period s. If smaller quantiles Q were desired, a smaller sampling frequency $1/s$ would be necessary that, in turn, produce a worse speed control. Conversely, if a higher sampling frequency $1/s$ were desired for better control (say 10 Hz), a worse speed quantile Q would be achived (16.4933 cm/s) and fewer perceivable speed levels (only 44). This analysis justified the first selection of a 200 ms sampling time, allowing a good-enough proportion between velocity resolution and sampling rate.

3.3 Actuators PWM Control Signal Generation

The vehicle actuators control depend on a proper PWM signal generation. In absence of any technical specifications of the vehicle actuators, an experimental identification became necessary. This identification served as a baseline to achieve robust linear velocity and directional control of the platform. The procedure followed permitted (a) to identify the actuators control PWM frequency and operation ranges, (b) acknowledge the PWM High Time resolution required to achieve adequate control of actuators, and (c) automate the PWM control signals generation, using an Arduino Nano microcontroller.

To roughly identify the PWM signal characteristics, a first off-road experiment was conducted. The car was placed on a platform with all wheels on air and manually controlled by the RC Transceiver. The PWM output from the Rx module to the vehicle ESC + BLDC motor system was measured using an oscilloscope. The procedure consisted on pairing the RC modules and slowly pushing the RC Transceiver left-stick from its center position

The PWM High Time readings were registered when the modules changed from "Not paired" to "Paired but still not-moving", and when the wheels' state changed to "Moving", starting the vehicle's acceleration. Finally the speed was adjusted to the maximum achievable value. This experiment was conducted 10 different times as the ESC-motor pairing condition was not always archived at the same PWM High Time. Average results are presented in Table 3.1.

Table 3.1: Identified PWM signals

PWM Frequency	50Hz
PWM High Time when RC is not paired	0 ms
PWM High Time when RC is paired but still not-moving	$\approx$ 1.46 ms
PWM High Time to start accelerating	$\approx$ 1.49 ms
PWM High Time to achieve maximum speed	$\approx$ 1.98 ms

The acquired data demonstrated that 50 Hz PWM frequency is always constant. Besides, the PWM High Time is different when the ESC-motor couple is paired or unpaired. As shown, a difference of around 30 microseconds is enough to change from "Paired but still not-moving" to "Moving". Furthermore, a PWM High Time operation interval of approximately 520 microseconds is required to obtain all achievable speed values. Accordingly, a microseconds resolution on PWM High Time is required to control the vehicle efficiently.

Arduino counts with a pre-existing PWM library (Analog IO) [53] in its IDE programming tool. However, this library has restricted configuration capabilities. In particular, the Arduino Nano board natively supports PWM frequencies of only 480 Hz or 980 Hz and permits a fixed PWM High Time variation of only 256 different values [54]. This pre-existing resource was not sufficient for the desired application.

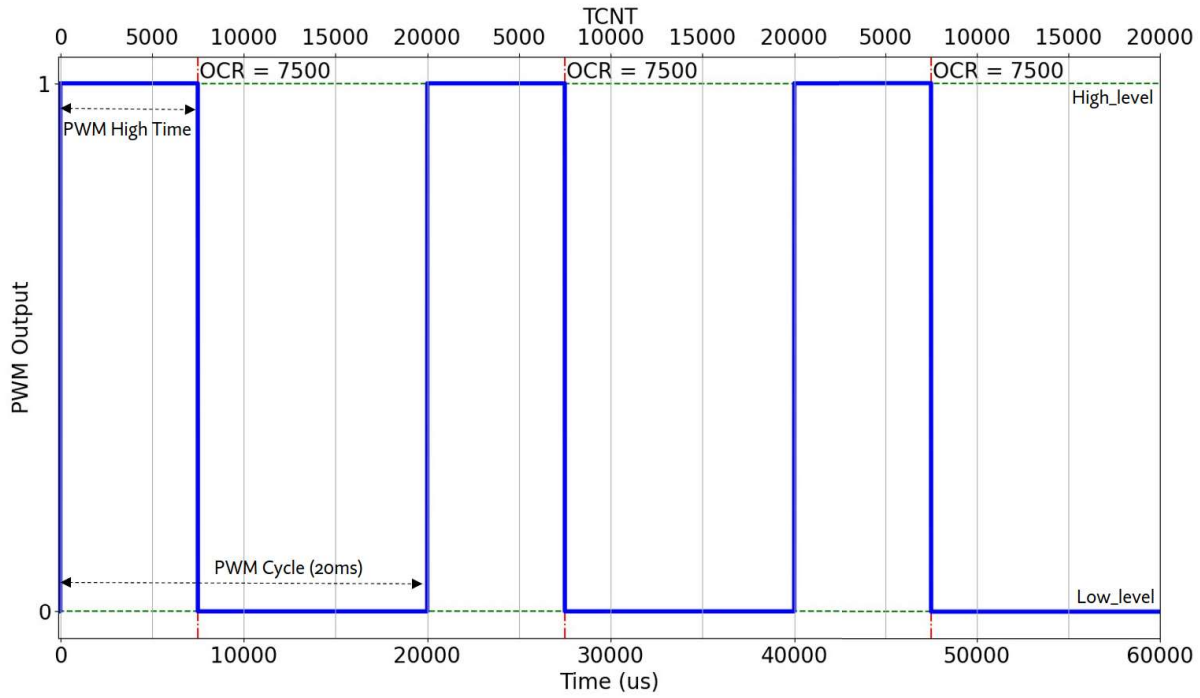


Figure 3.2: PWM characteristics.

The Arduino board has an Atmega 328P microcontroller chip [55]. This microcontroller can be appropriately configured to achieve a more robust PWM functionality. The microchip counts with a 16 bits PWM Timer that allows the adjustment of the operating frequency with an excellent PWM High Time resolution. The Arduino Command Line Interface (CLI) was adopted to optimize this low-level coding and overall performance [40]. The PWM frequency configuration [55] is determined by the equation:

$$f_{PWM} = \frac{f_{clk}}{2 * N * TOP} \tag{3.6}$$

Where:

f_{PWM} = PWM Operation Frequency = **50 Hz**.
f_{clk} = Atmega328P Internal Clock Frequency = **16 Mhz**;
N = PWM Operation Frequency Prescaler divisor (1, 8, 64, 256, 1024) = **8**;
TOP = PWM Timer Register Top Counting Value;

In order to achieve the desired 50 Hz PWM Operation Frequency, the Prescaler divisor was manually set to 8, allowing to easily determine the required TOP counting value. A 16-bit PWM Timer was used, allowing a maximum counting of 65535 values. However, as shown in Equation 3.6, the PWM Timer Register Top Counting Value must be configured with a TOP value of just 20000 to achieve the desired Frequency. The prescaler divisor factor of 8 produces the best counting resolution in comparison to others prescaler values. That is, if a 64 factor prescaler is selected, only 2500 counts would be available within the desired 50 Hz frequency, whereas a 1 factor prescaler would require 160000 counting values which cannot be configured given the 16-bit Timer resolution available.

When the Timer Register (TCNT) is set to zero, the PWM output is set to High until the Timer value is equal to a desired value as shwon in Figure 3.2. Such desired value can be configured using an specific Output Compare Register (OCR). Once the TCNT value reaches the adjustable OCR value, the PWM output is set to Low until the TCNT reaches the Top Counting value ($TOP = 20000$). This mean that every PWM cycle, the Timer Register (TCNT) reaches 20000 counts and then resets to zero again. Considering that each PWM cycle lasts 20 ms (at a frequency of 50 Hz), a single count takes 1 μs (that is 20 ms/20000). Therefore, an excellent PWM High Time resolution of 1 μs is achieved. Finally the PWM High Time can be easily configured by modifying the OCR value when necessary [55].

3.4 PWM Operational Ranges and Startup Behavior

As shown in Table 3.1, there is a clear gap between the vehicle's RC pairing and the motor's startup PWM High Times. However, such High Time value gap was hard to identify manually. This range of uncertain control signals values or *uncertain dead zone* correspond to a region where the vehicle's BLDC Motor and ESC pairing process has not been achieved yet, causing an unpredictable car's dynamic response at very slow speeds.

With the new and accurate PWM generator system based on Arduino, it is now possible to determine with more precision the best startup PWM High Time value and limit the mentioned *uncertain dead zone*. Therefore, a new set of experiments were conducted using both (a) the Linear Velocity Estimator, and (b) the PWM Control Signal Generator subsystems as displayed in Figure 3.3.

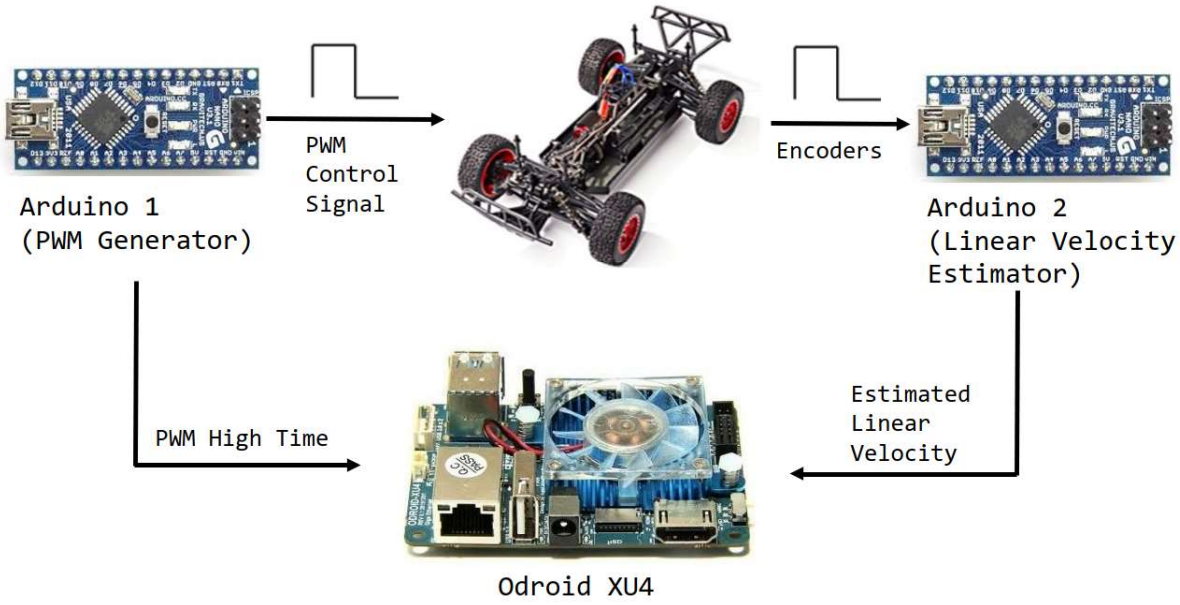


Figure 3.3: PWM High Time and estimated linear velocity monitoring system.

With the car's wheels on the air, a first off-road experiment was conducted, consisting on varying the PWM control signal with a defined High Time growing from 1.52 ms to 1.585 ms. That PWM signal was maintained constant for exact 4 seconds and then a 1 μs variation was added to the last outputted PWM High Time. Specific ROS topics, described in the Implemented software architecture in ROS subsection, were used to monitor: a) the generated PWM Control Signal and b) the estimated vehicle's linear velocity.

Figure 3.4 shows the typical car's startup behavior. The vehicle remained stopped when PWM High Time value is small but at about 1.53 ms the vehicle started moving exhibiting an overshoot that lasted no more than 1 or 2 sec. After that, a smooth and growing steady-state behavior appears, suggesting a possible linearity on the vehicle's dynamic response.

This first experiment confirmed the presence of an *uncertain dead zone* at the startup therefore, a slightly different experiment was executed to identify such startup behavior. All conditions of the previous experiment were kept the same, except that: a) the vehicle was placed on the floor and b) the input PWM High Time signal was changed a little bit. Previously, that signal grew up by 1 μs regardless the exhibited linear speed, but now it will vary continuously until the vehicle starts to move, registering the identified startup value.

A typical example of the developed experiment can be observed in Figure 3.6 where a High Time value of 1530 was just enough to provoke movement in the car. Unfortunately, when the experiment was repeated, the mentioned startup value was not always the same. Figure 3.5 shows the recorded performance of some experiments.

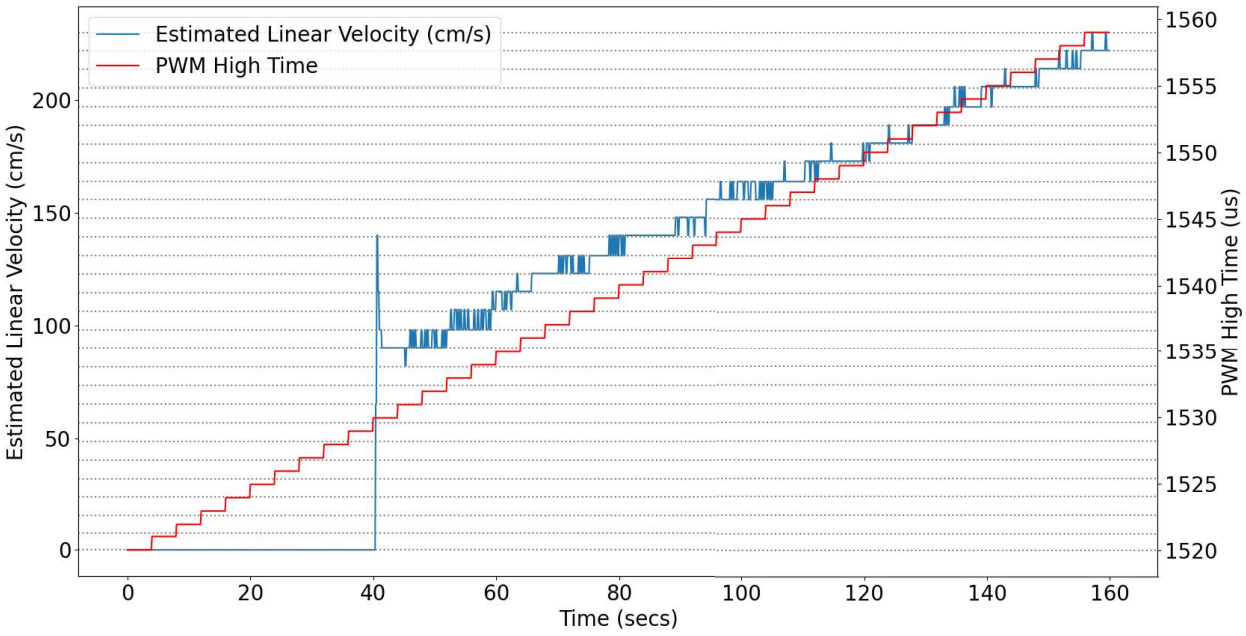


Figure 3.4: Vehicle Linear Velocity response to growing input PWM High Time.

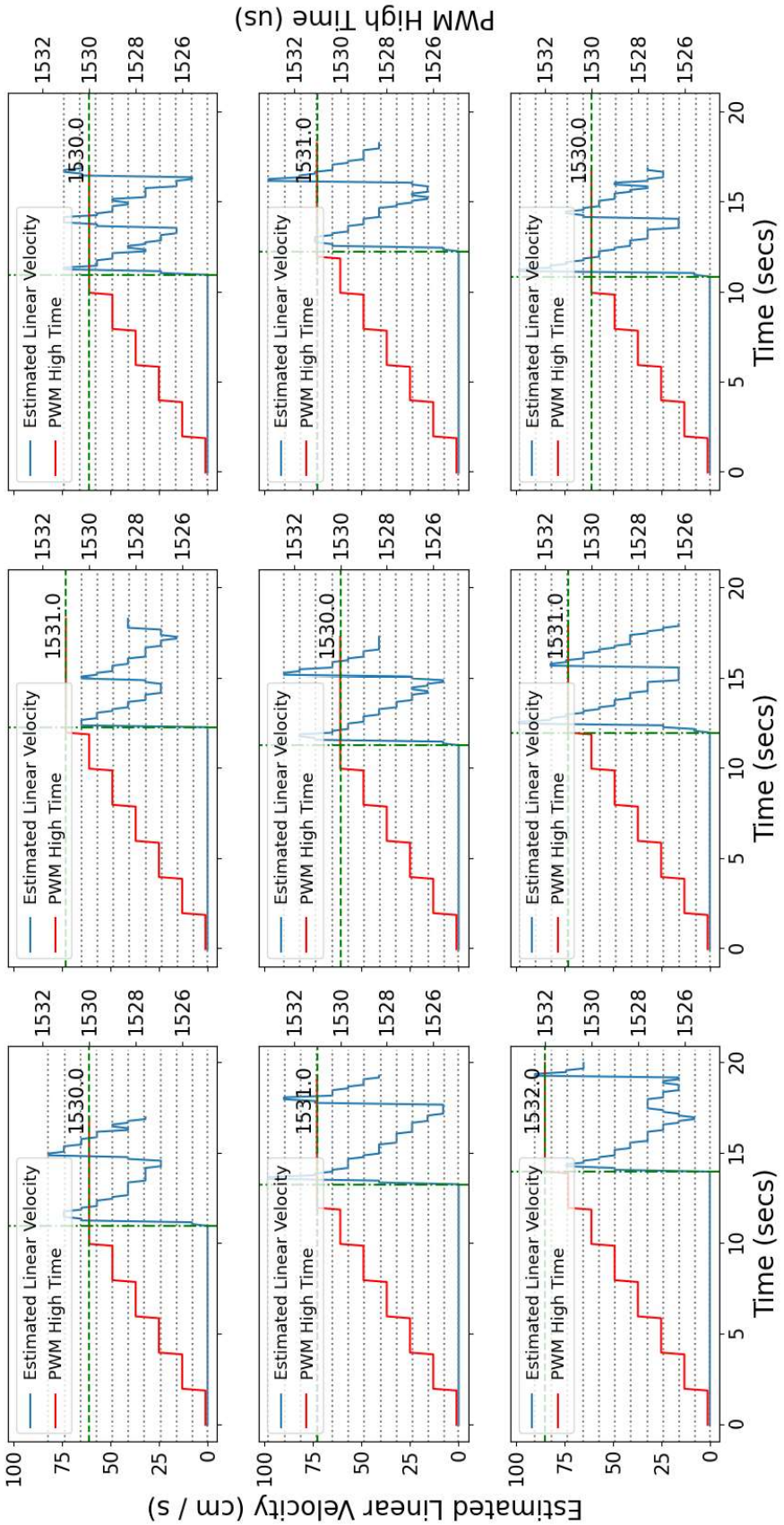


Figure 3.5: Examples of the developed acquisition experiments for the minimum PWM High Time required to begin the vehicle motion..

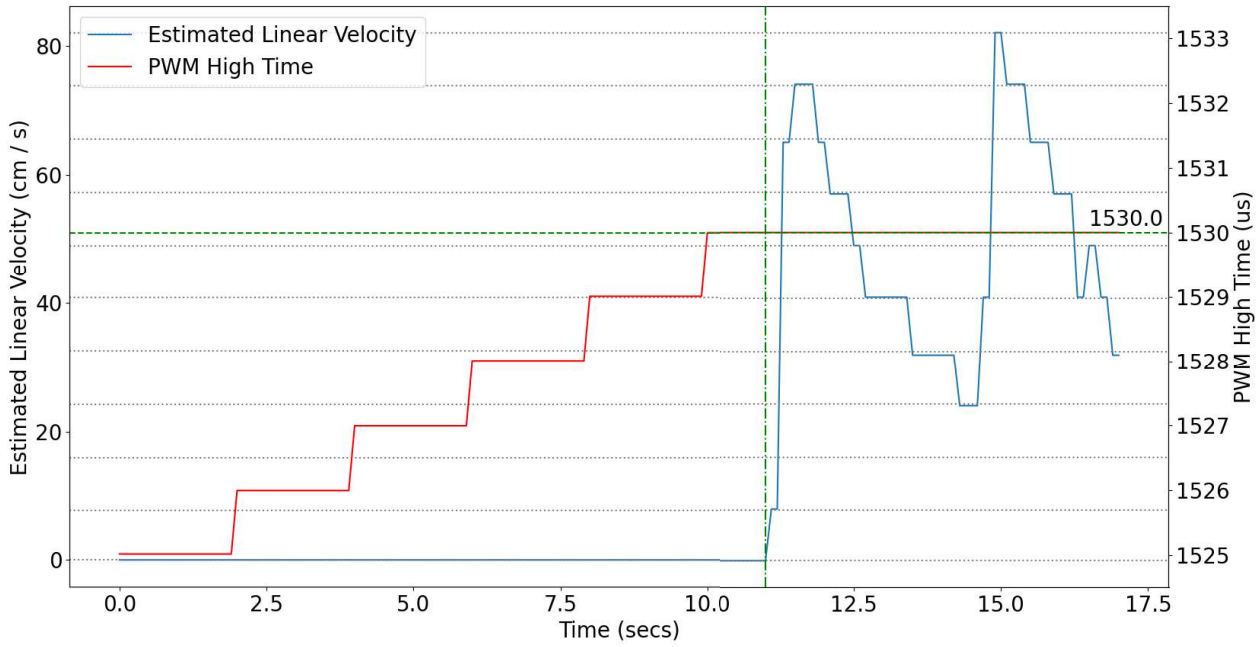


Figure 3.6: Single acquisition experiment for the minimum PWM High Time required to begin the vehicle motion.

The experiment was repeated 47 times. The sequence of each experiment started with the car static on the floor and a PWM High Time of 1.525 ms outputted during 2 seconds. If no motion was detected, an addition of 1 μs was applied to the PWM High Time. The new PWM Signal was then maintained during another 2 seconds period and the process was repeated until the vehicle began to move. The PWM High Time that caused the vehicle's motion was registered and the overall procedure was repeated.

Table 3.2: PWM Startup High Time Identification

PWM High Time	Repetitions	Probability
1.525 ms	7	14.89%
1.529 ms	5	10.63%
1.530 ms	12	25.53%
1.531 ms	22	46.80%
1.532 ms	1	2.13%

A histogram of all registered startup PWM High Times was created. Results are presented in Figure 3.7 and Table 3.2 showing four things: a) the distribution is not gaussian, b) the vehicle startup was achieved with PWM High Times ranging from 1.525 ms to 1.532 ms, c) the most frequent startup value was 1.531 ms, occurring in the 46.8% of the overall experiments and, d) in the 100% of the cases the car is guaranteed to be moving forward with PWM High Time of 1.532 ms.

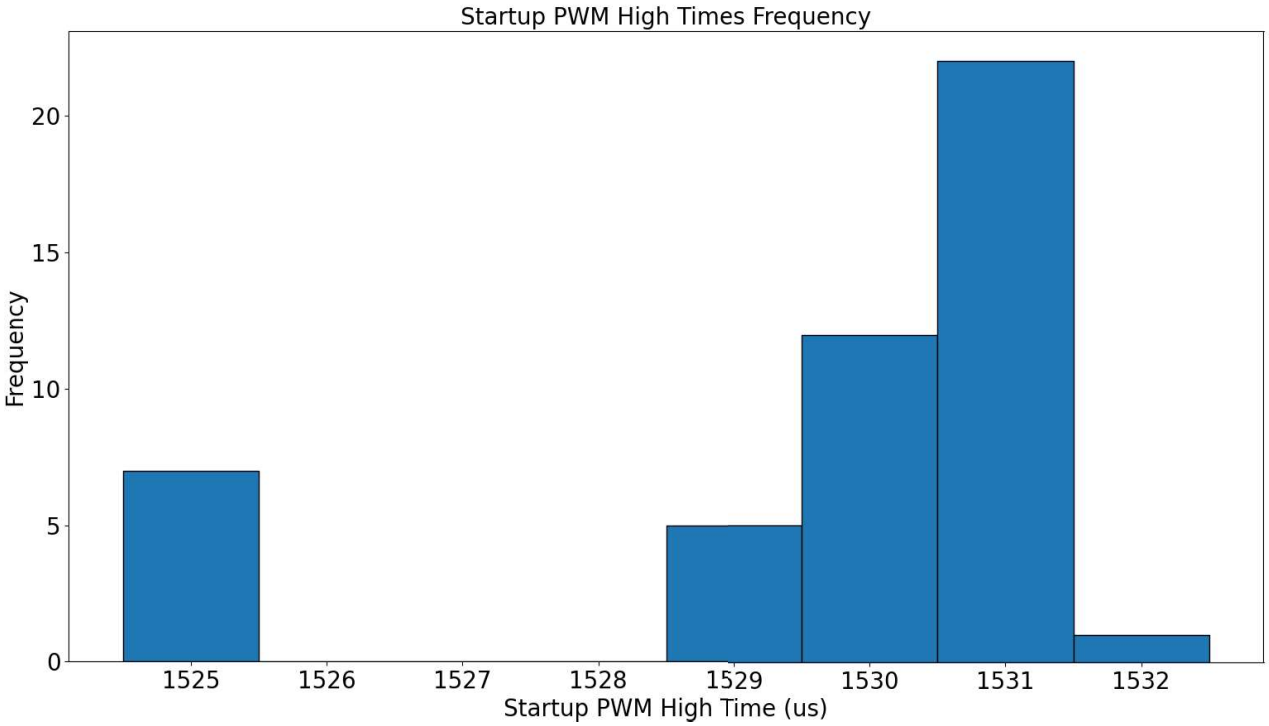


Figure 3.7: PWM startup High Times histogram for 47 different startup sequences.

3.5 Dynamic Step Response

To obtain an adequate model of the vehicle's forward speed for further control, a step response identification was performed. Therefore a new set of experiments were executed. With the car static on the floor and starting in one corner of the laboratory, the experiment consisted on inputting a PWM High Time step to the vehicle's ESC + BLCD motor and then record the outputted forward speed until the vehicle traversed a straight lane to the other side of the laboratory and then stopped before it collided into the wall. This procedure was repeated multiple times with different steps on the PWM High Time.

Figures 3.8, 3.9, 3.10, 3.11 exhibit the input-output dynamic of four different cases. For instance, Figure 3.8 shows two plots, the first one is the inputted PWM High Time signal in red, stepping from 1480 to 1530 μs at 5 sec, while the second plot in blue, is a horizontal line at 0 cm/s corresponding to the car's forward speed. In Figure 3.9 the input signal steps from 1480 to 1531 μs at 5 sec, while the output forward speed remained oscillating after a delay of about 0.5 sec.

Figure 3.10 shows the inputted control signal stepping from 1480 to 1541 μs at 5 sec, with an output signal overshooting after approximately 1 sec and reaching a steady-state velocity of around 41 cm/s. Finally, Figure 3.11 shows an input step of 1571 μs with an output speed with steady-state of about 131 cm/s. It must be noticed that all speed measurements are segmented in quantiles of $Q = 8.2467$ cm/s, as explained in the Linear Velocity Estimation, subsection 4.2.

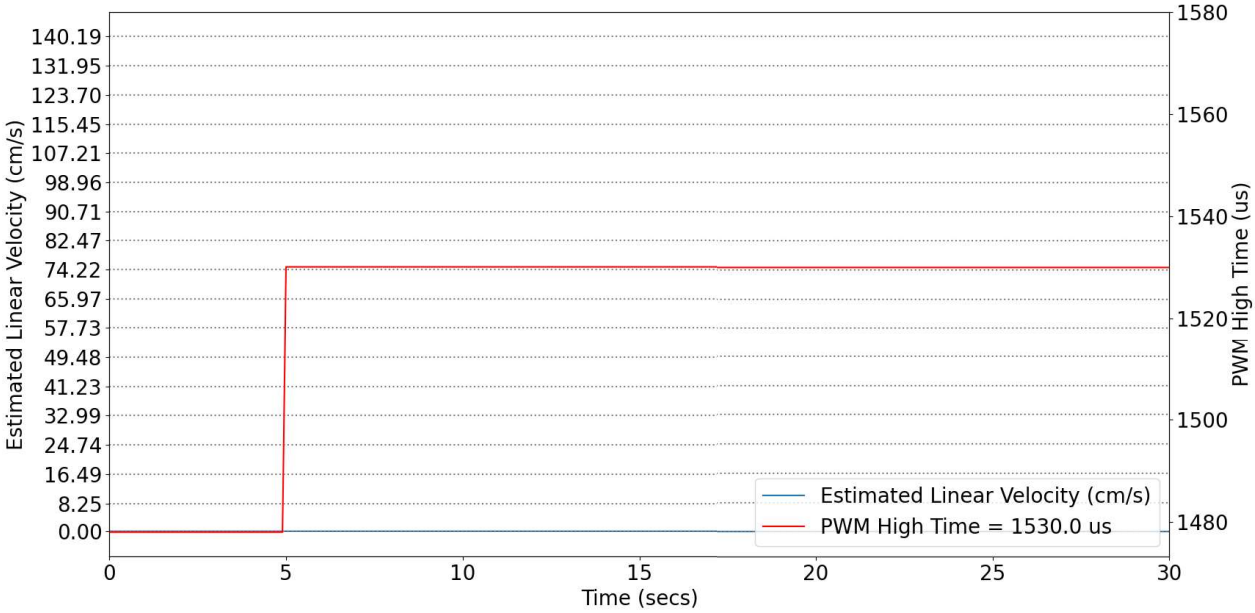


Figure 3.8: Vehicle's Speed Step Response for PWM High Time of 1.53 ms.

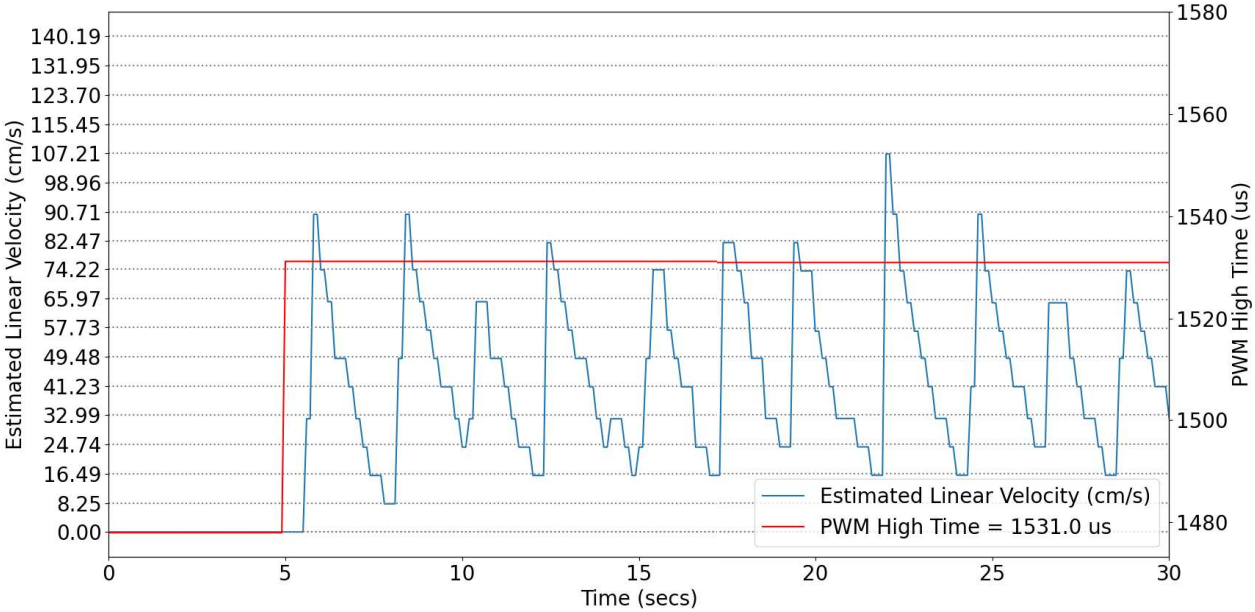


Figure 3.9: Vehicle's Speed Step Response for PWM High Time of 1.531 ms.

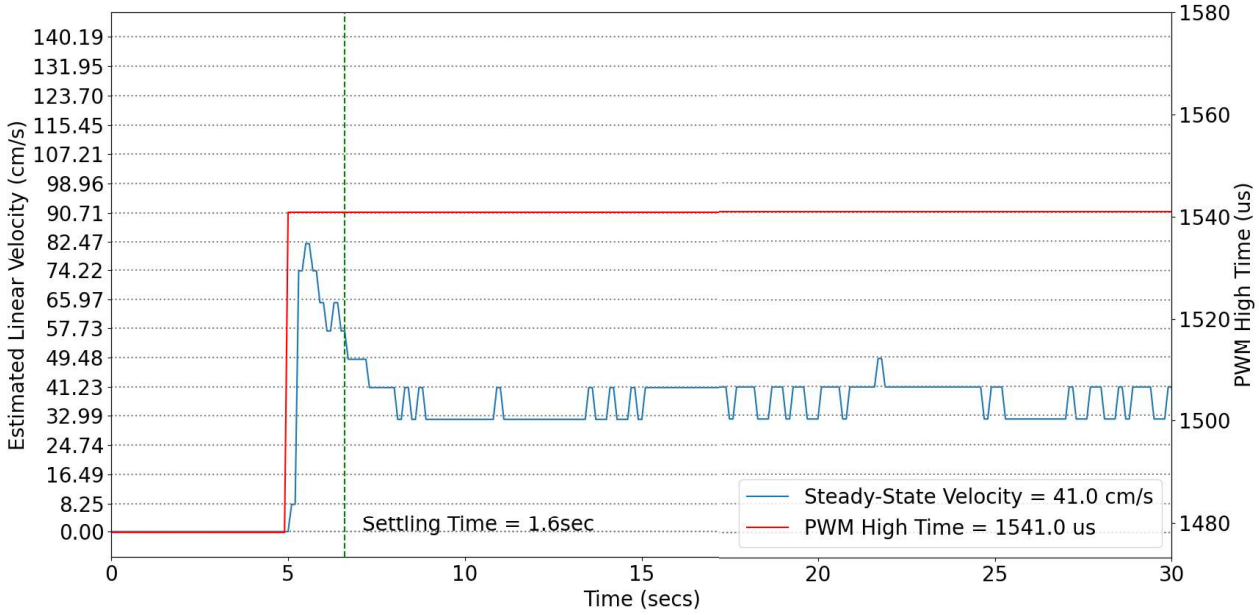


Figure 3.10: Vehicle's Speed Step Response for PWM High Time of 1.541 ms.

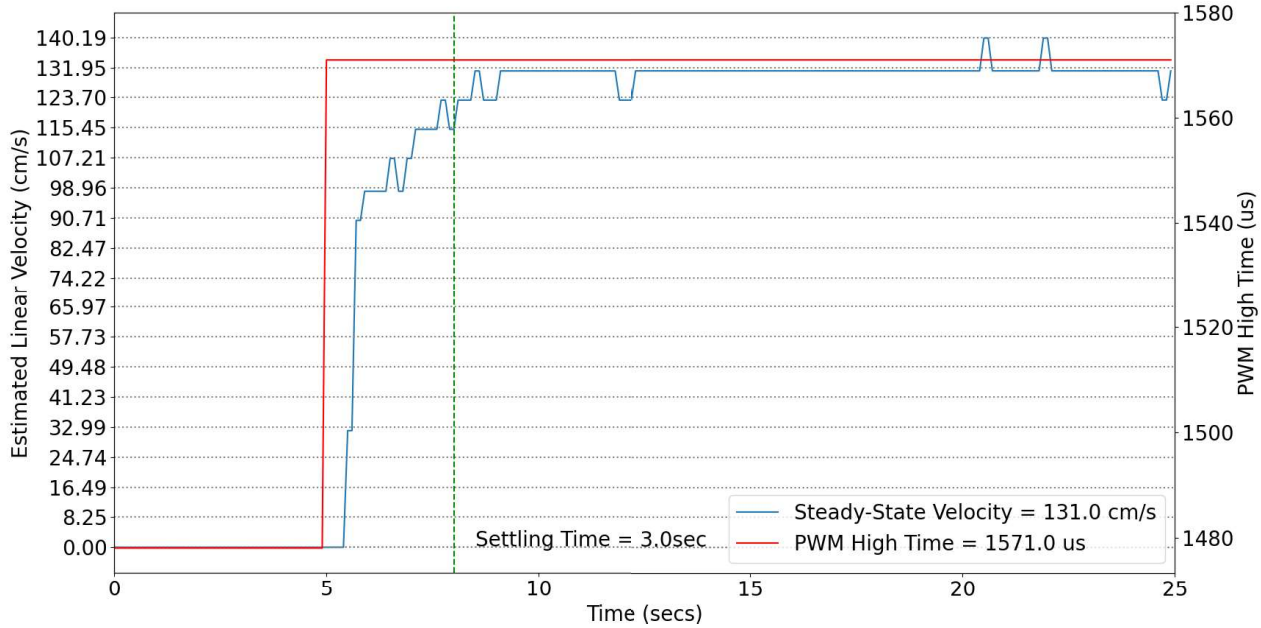


Figure 3.11: Vehicle's Speed Step Response for PWM High Time of 1.571 ms.

Table 3.3: PWM High Time Step Responses.

High Time (us)	Steady-state Velocity (cm / s)	Settling Time (sec)
1541.0	41.0	1.6
1542.0	41.0	1.5
1543.0	41.0	1.9
1544.0	41.0	2.3
1545.0	49.0	1.5
1546.0	49.0	1.3
1547.0	57.0	1.0
1548.0	57.0	0.9
1549.0	57.0	1.1
1550.0	65.0	1.1
1551.0	65.0	1.5
1552.0	74.0	0.2
1553.0	82.0	0.3
1554.0	74.0	0.9
1555.0	82.0	0.2
1556.0	90.0	0.5
1557.0	90.0	0.4
1558.0	82.0	0.6
1559.0	90.0	0.6
1560.0	90.0	0.6
1561.0	98.0	3.0
1562.0	98.0	1.8
1563.0	107.0	2.6
1564.0	107.0	1.9
1565.0	107.0	2.2
1566.0	115.0	1.8
1567.0	115.0	2.2
1568.0	123.0	2.2
1569.0	131.0	2.8
1570.0	123.0	2.0
1571.0	131.0	3.0
1572.0	131.0	2.4
1573.0	140.0	2.7
1574.0	131.0	2.5
1575.0	140.0	2.5
1576.0	148.0	2.7
1577.0	148.0	2.0
1578.0	148.0	2.8
1579.0	148.0	4.7
1580.0	148.0	2.7
1582.0	156.0	3.5
1583.0	164.0	2.8
1584.0	164.0	2.8
1585.0	173.0	3.1

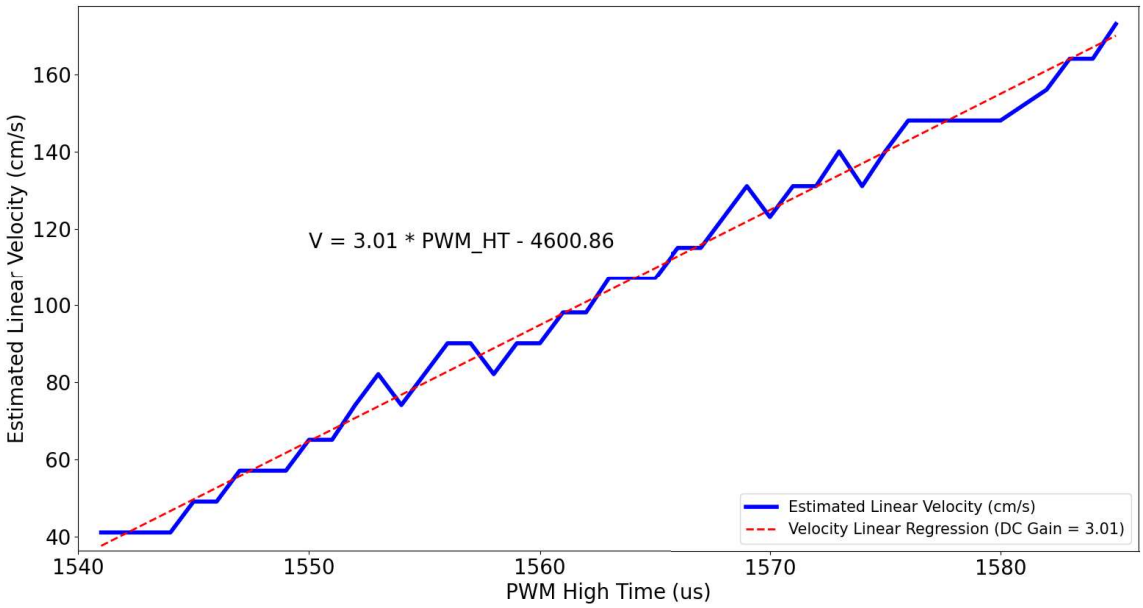


Figure 3.12: PWM High Time vs Steady-state Linear Velocity.

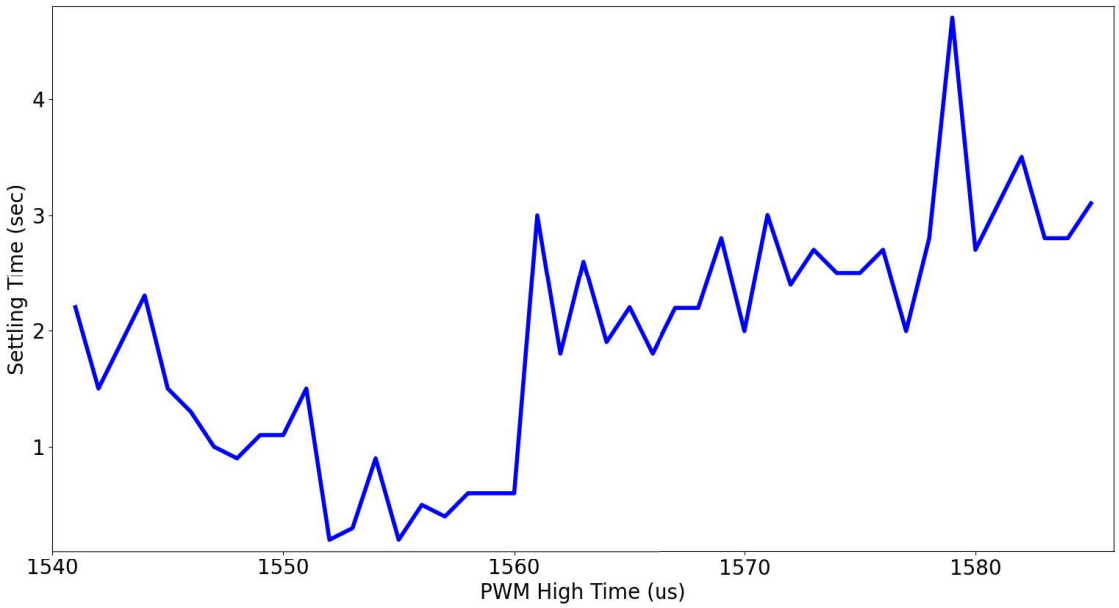


Figure 3.13: PWM High Time vs Settling Time.

Roughly speaking, four different type of dynamic responses were identified: (a) a zero velocity when PWM High Time inputs are too small, (b) an oscillatory response with PWM High Times close to the previously identified startup value of 1.531 ms, (c) a seemingly underdamped behavior for intermediate input values, and (d) an overdamped response for higher input values.

In previous experiments, the PWM High Time of 1.531 ms demonstrated to be a good startup value as shown in Figure 3.9. These oscillatory responses were also seen in multiple subsequent experiments with small enough High Times. However, PWM High Times smaller than this startup value were not sufficient to move the vehicle, see Figure 3.8. Therefore, it was ratified that a High Time value of 1.531 ms could be considered a good-enough startup value. In other words, this value can be used as an offset to avoid the so-called *uncertain dead zone*.

Finally, an analysis of the relationship between the input signals versus steady-state speed and settling-time was conducted. For this analysis, all experiments with input step higher than 1.541 ms were taken into account, where dynamics are overdamped. Steady-state velocity is defined as the last velocity reached at the end of the experiment, typically at 30 s. Settling time is defined as the first time the signal remained inside a tolerance range of $\pm 2Q$ of its final speed.

Table 3.3 shows the results. Moreover, Figure 3.12 shows a linear relationship between the inputted PWM High Time and the achieved steady-state linear velocity. As the PWM High Times increased so did the steady-state velocity estimations with an approximate DC Gain of 3.01, suggesting that it could be possible to estimate a linear dynamical model.

Nonetheless, the settling time did vary strongly throughout the tested conditions as shown in Figure 3.13. Furthermore, the setting time took considerably smaller values for medium speeds (say 1.552 ms to 1.56 ms PWM High Times) than those registered for other velocities. These variations limited the assumption of the stated linearity of the vehicle dynamics.

3.6 Dynamic Model Identification

A series of experiments were developed to identify an adequate dynamic linear model. The experiments consisted on the generation of multiple pseudo random PWM High Time sequences and registering the speed response of the vehicle while traversing a straight lane. The relation between the input pseudo random variations and the vehicle dynamic response was later analyzed to produce a dynamic model approximation.

The vehicle's straight trajectory was limited by the available space inside the laboratory where the experiments were developed. Therefore, multiple trajectories were executed in order to successfully capture the vehicle's overall dynamic response. Specifically a variety of 30 vehicle trajectories were produced.

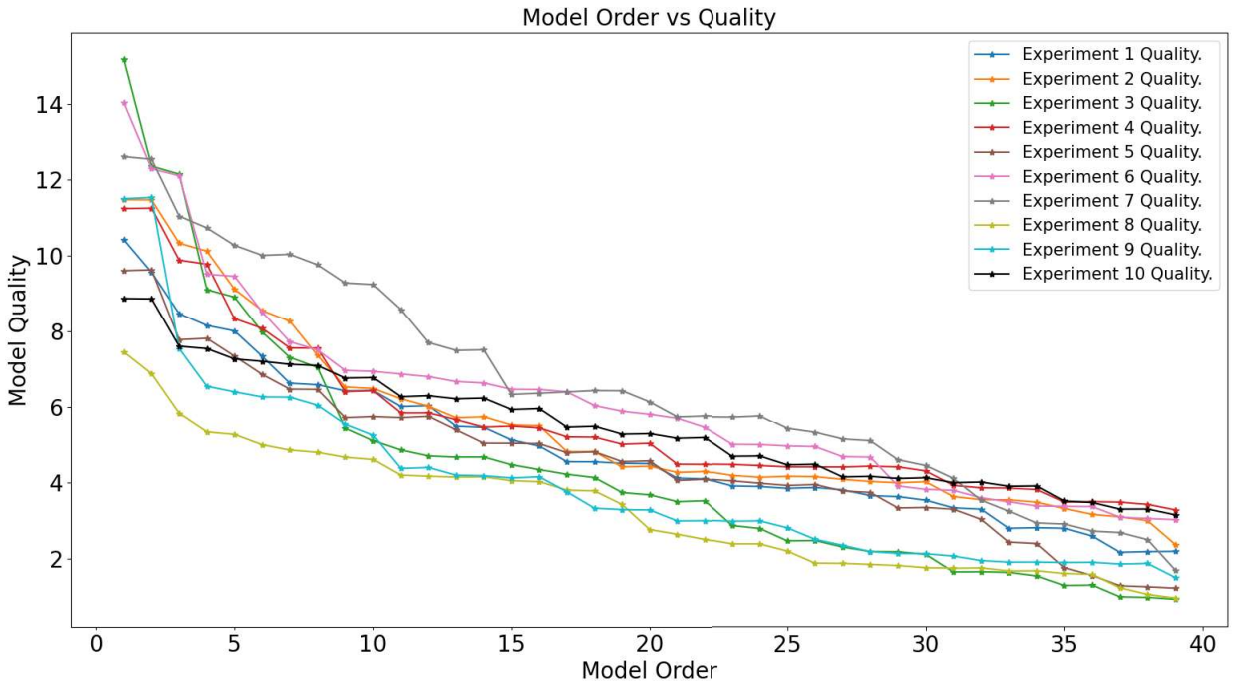


Figure 3.14: Model Order VS Quality plot for the first 10 Model Identification Experiments.

The registered response of each experiment was used to define a particular dynamic model. Such models were based on the vehicle's dynamic response registered once an adequate startup was achieved after exact 10 seconds, avoiding the aforementioned *uncertain dead zone*. This criterion was defined based on the non-linear response identified at startup as described in the PWM Operational Ranges and Startup Behavior subsection.

Given a set of input-output data $\{u(k),\ y(k)\}$, an ARMA model of a given order n can be derived using a Least Squares Identification [56]:

$$G(z) = \frac{Y_m(z)}{U(z)} = \frac{b_1 z^{n-1} + b_2 z^{n-2} + ... + b_n}{z^n + a_1 z^{n-1} + ... + a_n} \tag{3.7}$$

where model quality is defined as:

$$J = \sum e(k)^2 \tag{3.8}$$

$$e(k) = y(k) - y_m(k) \tag{3.9}$$

and:

$y(k)$ = Vehicle's registered linear velocity (cm/s);
$u(k)$ = PWM High Time input (μs);
$y_m(k)$ = Identified model linear velocity estimation output (cm/s);
$e(k)$ = Linear velocity estimation error (cm/s);
n = Model order.

Figure 3.14 shows the relationship between model quality and model order for 10 different experiments. It is clear that the models behaved similarly, with a continuous quality improvement as the model order increases. It is also remarkable that the quality kept on improving for bigger model orders instead of presenting an asymptotic behavior (typical of linear dynamics). This behaviour could be the result of an unidentified non-linearity of the vehicle's overall dynamics. Moreover, it is know that, bigger the model order bigger the chances of over-fitting and therefore getting a bad model.

Knowing that a hyper-high model order is useless, a strong assumption was made and a second order model was kept as the best fit for the vehicle's identified overdamped dynamic. General form of such model is show in equation 3.10.

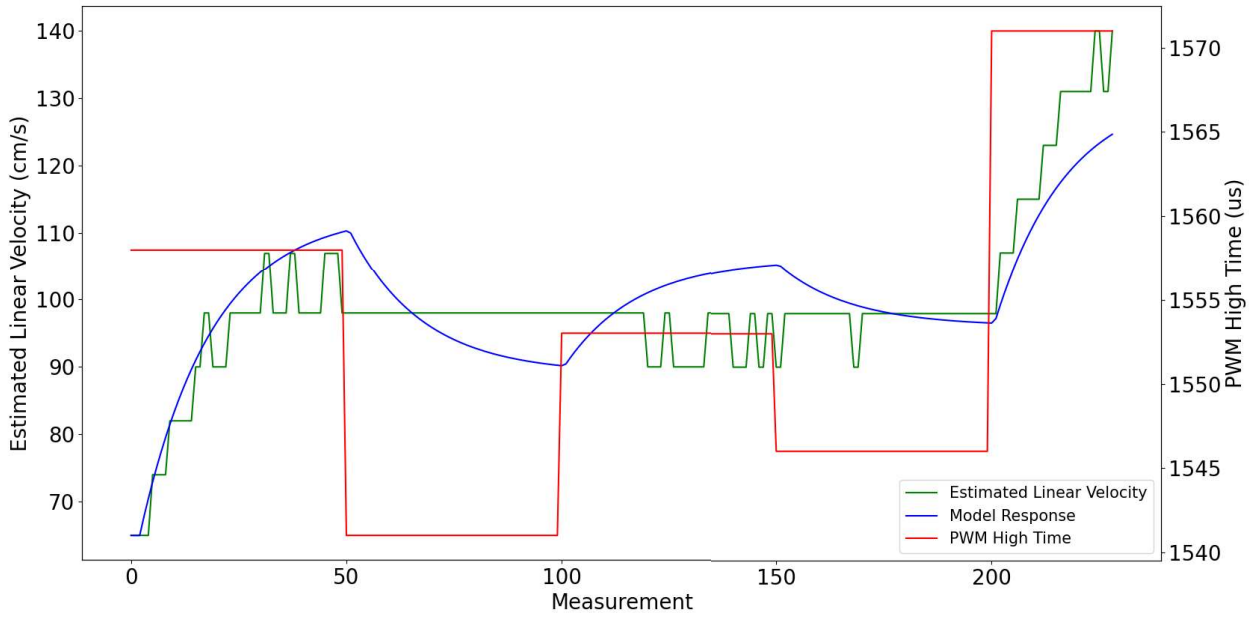


Figure 3.15: Single experiment dynamic response approximation and registered vehicle's dynamic comparison.

For each of 30 experiments, the registered car's speed was compared to that obtained from its corresponding identified model. An example of these comparatives can be observed in Figure 3.15 that has three different plots (a) the second order model approximation in blue, (b) the registered experiment's dynamic response in green, and (c) the inputted control signal in red. It can be observed that the model estimation poorly resembled the actual

dynamic response.

Table 4.4 shows the identified models' parameters for each individual experiment. Even though the experiment was repeated 30 times, the identified poles seemed to be always the same, but that wasn't the case for the zero and the gain. Also, the individual quality exhibited a huge variation among the identified models.

$$G(z) = \frac{K(z - z_1)}{(z - p_1)(z - p_2)} \tag{3.10}$$

In order to get a unique model, an average of all identified parameters has been computed. That unique model is presented in equation 3.11. The results of the statistical analysis can be observed at the bottom of Table 3.4. An average gain K value of 0.0212 was determined with a deviation of 0.0737, while the model poles average values were 0.9476 and -0.0804 with deviations of 0.0235 and 0.1004 respectively. In general, the deviation of these three last parameters showed that the obtained average values could represent the overall response of the experiments. However, this is not true for the model's zero average value of -2.776 which presented a substantial deviation of 12.4932.

$$G(z) = \frac{0.021(z + 2.776)}{(z - 0.947)(z + 0.0804)} = \frac{0.021z + 0.05885}{z^2 - 0.8672z - 0.07619} \tag{3.11}$$

In order to verify the validity of this last unique model (3.11), it was re-computed the model quality for each individual experiment and computed the *Models Quality Differential*. Notice that, in all cases, the quality has decreased. Average and deviation of such quality differential were obtained, registering an average value of -2.9022 and a deviation of 2.2513. This imply that the dynamic response achieved with the average model 3.11 got a bit worst in each of the experiments than its corresponding individual models. Unfortunately this fact limits the validity of the initial dynamic linearity assumption and therefore decreases the utility of the derived unique model.

Multiple evidences of a possible non-linearity were recognized, such as the *uncertain dead zone* dynamic response, the observed quality vs model order behaviour from Figure 3.14 and the strong dynamic model's parameters variation shown in Table 3.4.

Finally, it was concluded that a better dynamic model approximation should be achieved if a non-linear identification procedure is made. However, non-linear identification is not part of the original scope of this thesis and will therefore be left for future supplementary research.

Table 3.4: Dynamic Model Identification Experiments Results.

No.	K	$Pole_1$	$Pole_2$	$Zero$	Individual Model Quality	Average Model Quality	Models Quality Differential
1	-0.022893	0.952056	-0.060327	4.6471	10.379	11.459	-1.080
2	0.059722	0.961203	-0.041269	-0.19085	11.540	14.395	-2.855
3	0.029045	0.91977	-0.32126	-5.4316	13.491	20.727	-7.236
4	-0.044564	0.981371	-0.042409	1.7747	11.268	12.678	-1.410
5	0.0050049	0.884655	0.049223	-37.186	9.6261	11.892	-2.266
6	0.0044416	0.90877	-0.22824	-38.714	13.116	15.614	-2.498
7	-0.017260	0.966494	-0.083071	4.7955	12.592	15.635	-3.043
8	-0.026111	0.942102	-0.076235	5.0567	7.3712	9.634	-2.263
9	0.032574	0.958511	-0.023943	-1.0000	11.570	12.019	-0.449
10	0.026878	0.95947	-0.00049341	-1.3427	8.8459	9.270	-0.424
11	0.026726	0.945156	-0.011923	-1.9394	8.8927	8.933	-0.040
12	0.080688	0.93107	-0.13599	-0.56043	13.230	14.533	-1.303
13	-0.011695	0.94403	-0.30675	12.758	8.7727	15.065	-6.292
14	0.24531	0.94645	-0.19464	0.52668	7.5443	12.525	-4.981
15	0.041776	0.966590	-0.031919	-0.56292	9.8550	15.629	-5.774
16	0.024073	0.964281	-0.073536	-1.9961	8.0842	13.582	-5.498
17	-0.0060543	0.966448	-0.013319	9.4694	7.6803	7.840	-0.160
18	-0.036127	0.944195	-0.089036	4.0920	9.9405	15.411	-5.470
19	-0.030208	0.975205	-0.041353	2.4517	10.338	11.740	-1.402
20	-0.0060543	0.966448	-0.013319	9.4694	7.6803	7.840	-0.160
21	0.026878	0.95947	-0.00049341	-1.3427	8.8459	9.270	-0.424
22	0.0062852	0.960880	-0.023124	-10.608	12.577	15.716	-3.139
23	0.0050049	0.884655	0.049223	-37.186	9.6261	11.892	-2.266
24	-0.022893	0.952056	-0.060327	4.6471	10.379	11.459	-1.080
25	0.24531	0.94645	-0.19464	0.52668	7.5443	12.525	-4.981
26	-0.16125	0.934903	-0.024446	1.7491	10.075	14.336	-4.261
27	0.041776	0.966590	-0.031919	-0.56292	9.8550	15.629	-5.774
28	0.032574	0.958511	-0.023943	-1.0000	11.570	12.019	-0.449
29	0.029045	0.91977	-0.32126	-5.4316	13.491	20.727	-7.236
30	0.059722	0.961203	-0.041269	-0.19085	11.540	14.395	-2.855

Parameter	Average	Deviation
K	0.0212	0.0737
p_1	0.9476	0.0235
p_2	-0.0804	0.1004
z_1	-2.7760	12.4932
Models Quality Differential	-2.9022	2.2513

3.7 Linear Velocity Control

The development of a robust linear velocity control was limited by the recognized non-linear dynamic response and the unsuccessful identification of the vehicle dynamics. However, it was still required to achieve a sufficient linear velocity control of the platform for its autonomous driving. To achieve this, an heuristically tuned PI velocity control was implemented [57].

The control system architecture is displayed in Figure 3.16. The closed-loop system utilizes the linear velocity estimation obtained from Arduino 2, feeding this data to Arduino 1 responsible for the control signal generation. A PI controller was implemented directly in the Arduino 1 board, receiving the desired linear velocity from the Odroid XU4.

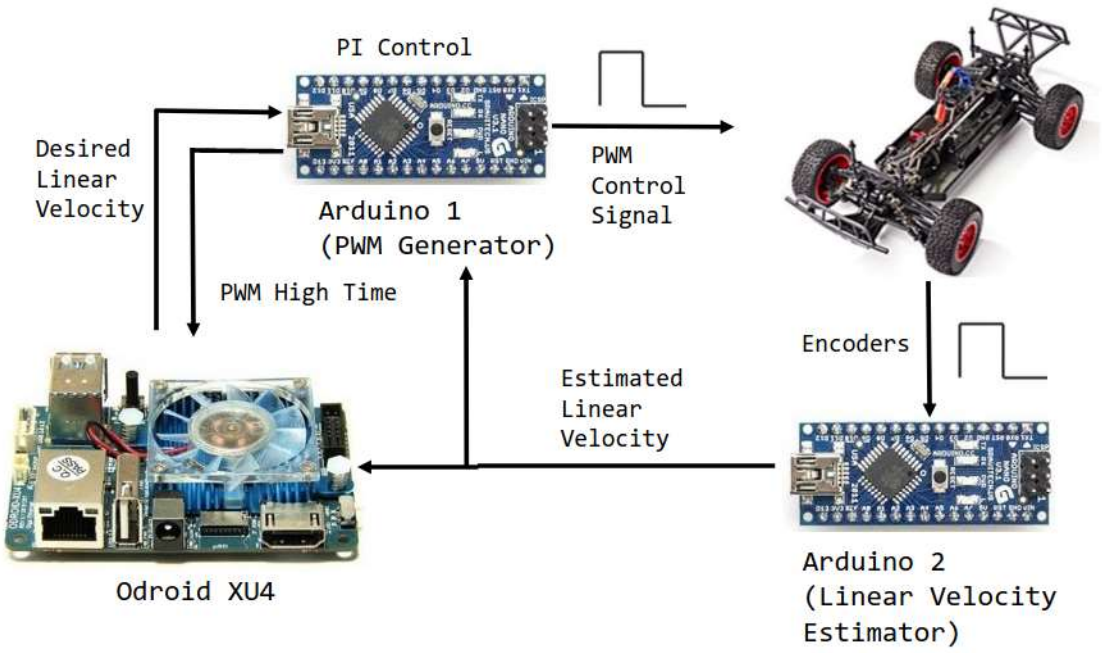


Figure 3.16: Linear Velocity Control System Architecture.

Arduino 2 shares the Linear Velocity Estimation with Arduino 1 using the I^2C communication protocol. This allows a fast data acquisition and permits the implementation of the desired control system. The estimated velocity is then subtracted to the desired velocity to compute the current linear velocity error $e(t)$. The PWM High-Time is finally computed using the PI equation 3.12 and sent to the ESC-BLCD motor. Two specific ROS topics where put on place at the Odroid XU4 to monitor the behavior of vehicle's linear velocity.

The designed PI controller utilizes the equation:

$$PWM_{\text{OUT}} = PWM_{\text{MIN}} + K_p e(t) + K_i \sum_{\tau=0}^{t} e(\tau)\Delta t \tag{3.12}$$

where:

PWM_{OUT} = PWM High-Time sent to ESC-BLCD motor.
PWM_{MIN} = Minimum startup PWM High Time = **1.531 ms**
K_p = Proportional gain.
K_i = Integral gain.
$e(t)$ = Current linear velocity error.
Δt = Sampling Period = **200 ms**.
t = The sampling moment. A positive integer that grows linearly.

The PI controller uses the offset (PWM_{MIN}). This guarantees that the linear velocity control will always operate outside the so-called *uncertain dead zone* interval where there's no certainty of the vehicle dynamic response. A representation of this low-level implementation is shown in Figure 3.17

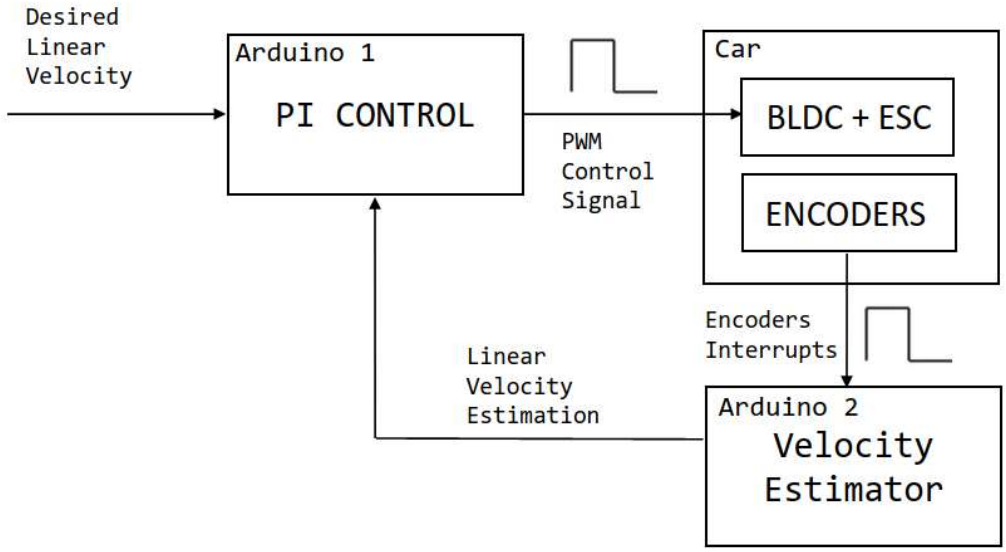


Figure 3.17: Low-level linear Velocity Control block diagram.

The controller depends on the identification of the actual linear velocity error $e(t)$ which is obtained by comparing the current desired linear velocity and the estimated linear velocity. The mentioned estimation is produced each 200 ms, corresponding to a well-defined sampling period Δt that is used to apply the desired control action.

The controller also depends on the proper tuning of the Proportional and Integral Gains. Such constants were tuned empirically by testing the implementation for multiple desired speeds. The constants were continuously tuned until a good response was achieved, with an small steady-state error and a short settling time.

The target linear velocity values were: $8Q$=65.97 cm/s, $10Q$=82.47 cm/s, $13Q$=107.21 cm/s, $15Q$=123.70 cm/s, $19Q$=156.69 cm/s, $22Q$=181.43 cm/s and $25Q$=206.17 cm/s.

These desired speeds were tested with the vehicle on the floor, using the velocity controller while the vehicle traversed autonomously a straight road. The vehicle speed was registered using specific ROS topics to be later processed. An analysis of the controller performance was subsequently developed, registering the obtained settling values.

As previously mentioned, the velocity estimations were constrained by the measurement quantiles $Q = 8.2467$ cm/s. This estimation characteristic limited the identification of the speed. For this, a tolerance of ± 2 Q from the desired speed was used to establish a settling speed interval. The registered speed would be considered in a settling state if such measurement remained inside the mentioned tolerance interval.

Table 3.5: First PI values ($K_p = 0.12$, $K_i = 0.21$).

Settling Speed (+/- 2Q)	Settling Time
65.97 cm/s	2.7 secs
82.47 cm/s	0.6 secs
107.21 cm/s	0.3 secs
123.70 cm/s	1.5 secs
156.69 cm/s	3.3 secs
181.43 cm/s	3.2 secs
206.17 cm/s	2.9 secs

The first heuristically tuned PI resulted with a K_p gain of 0.12 and a K_i gain of 0.21. That proposed PI controller presented a short-enough settling time for desired low speeds. In particular, the vehicle demonstrated a short settling time for desired linear velocities lower than 107.21 cm/s, as shown in Figures 3.18 to 3.20 and Table 3.5. However, for considerably low desired linear speeds such as 65.97 cm/s the settling time was higher (around 2.7 seconds).

The mentioned low-speed behaviour could be a reflex of the aforementioned nonlinear startup dynamics. Nonetheless, the achieved closed-loop performance was considered acceptable given the characteristics of the desired application. Furthermore, the control performance got substantially worse for higher speeds. As the desired speed value increased so did the obtained settling time (around 3 seconds) for velocities greater that 123.70 cm/s as shown in Figures 3.21 to 3.24. This high settling times were not desirable for those bigger speeds.

In order to achieve a good high-speed control of the platform, a second PI controller was developed. The same tuning process was followed, adjusting the K_p and a K_i gains empirically until an adequate high-speed control was achieved. The second PI controller resulted with a K_p gain of 0.6 and a K_i gain of 0.22.

As shown in Table 3.6 and Figures 3.27 to 3.31, this second controller exhibited a good high-speed performance, achieving short-enough settling times of around 0.7 seconds. However the settling time archived for low speeds, say lower than 107 cm/s, got considerably worse, with settling times greater than 10 seconds (compared with those settling times archived in Figures 3.18 and 3.19). This low-speed settling times were unacceptable, because the vehicle would be mainly driven at low-speeds during the autonomous driving.

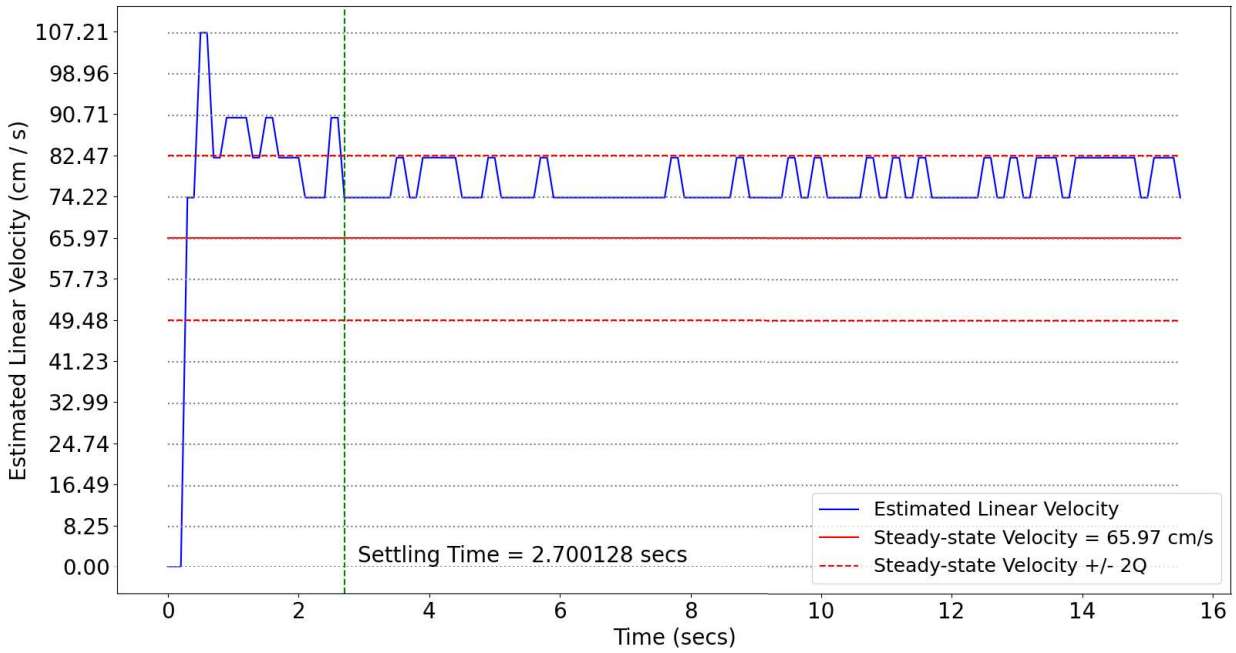


Figure 3.18: Closed-loop output-response with first PI Controller test for 65.97 cm/s desired velocity at 5Hz sampling.

The two presented PI linear velocity controllers were the best proposals achieved by the empirical tuning process. However, it was not possible to identify a unique controller that successfully achieved an adequate performance for the whole range of tested speeds. Given that the autonomous driving algorithm development would require mainly a low-speed control of the vehicle, it was decided to utilize the first PI controller.

Table 3.6: Second PI ($K_p = 0.6$, $K_i = 0.22$).

Settling Speed (+/- 2Q)	Settling Time
65.97 cm/s	18.4 secs
82.47 cm/s	11.3 secs
107.21 cm/s	0.6 secs
123.70 cm/s	0.7 secs
156.69 cm/s	0.7 secs
181.43 cm/s	0.7 secs
206.17 cm/s	0.8 secs

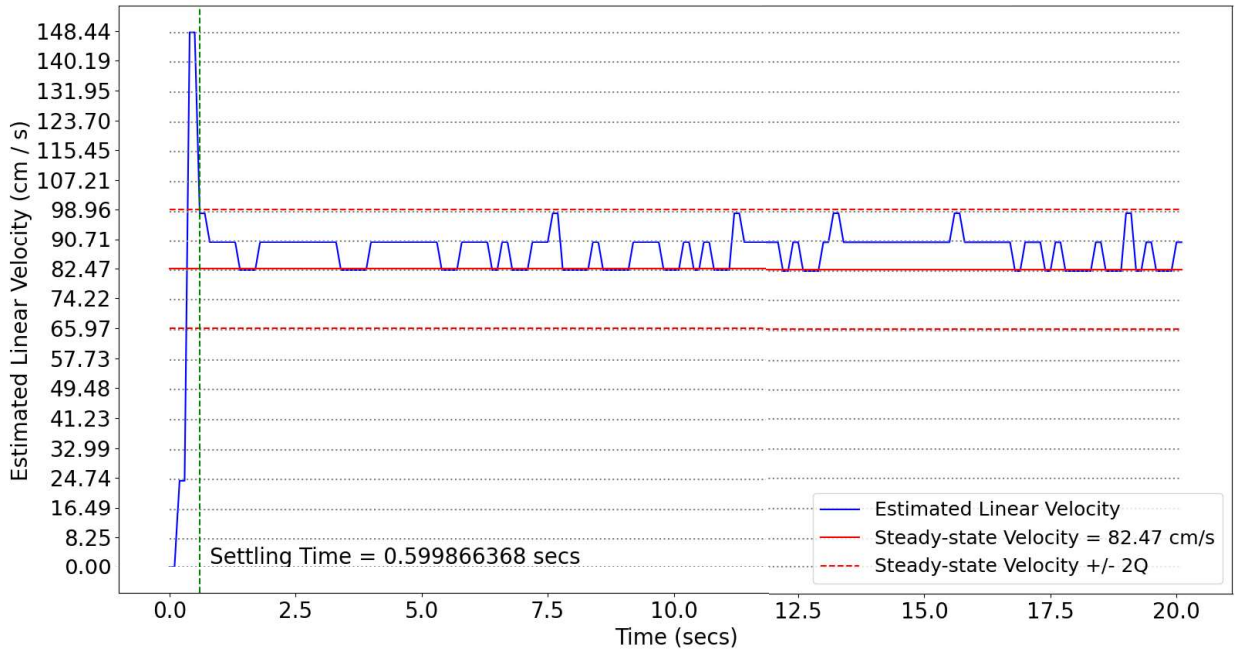


Figure 3.19: Closed-loop output-response with first PI Controller test for 82.47 cm/s settling point at 5Hz sampling.

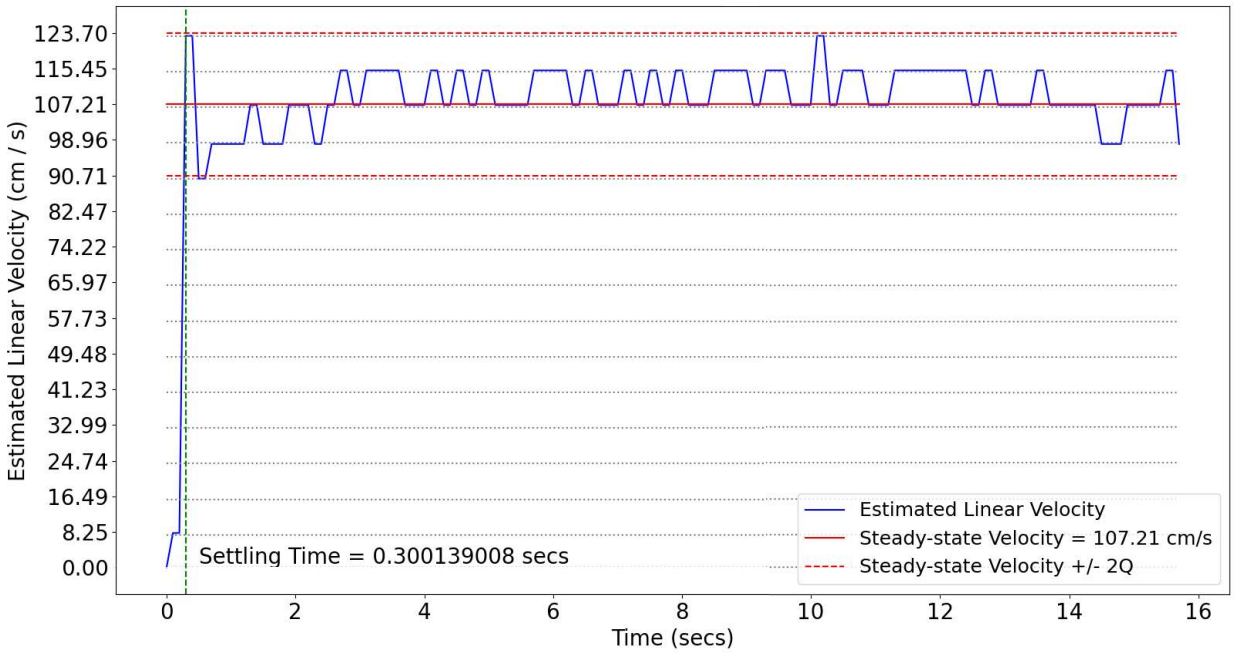


Figure 3.20: Closed-loop output-response with first PI Controller test for 107.21 cm/s desired velocity at 5Hz sampling.

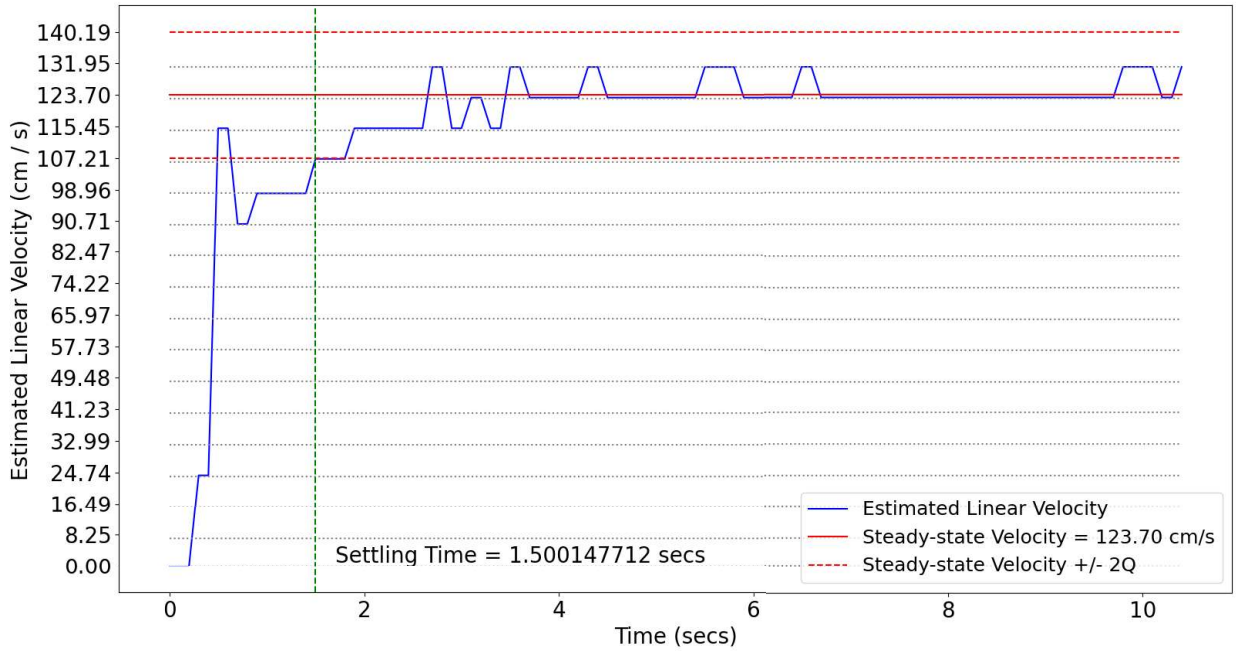


Figure 3.21: Closed-loop output-response with first PI Controller test for 123.70 cm/s desired velocity at 5Hz sampling.

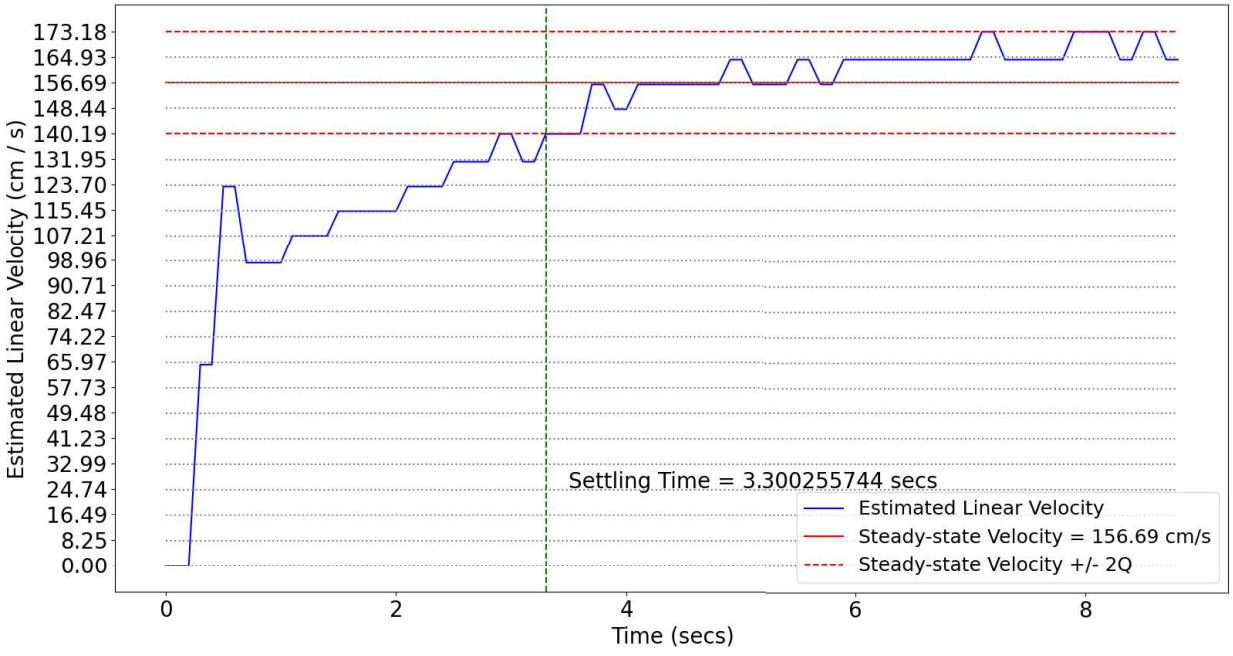


Figure 3.22: Closed-loop output-response with first PI Controller test for 156.69 cm/s desired velocity at 5Hz sampling.

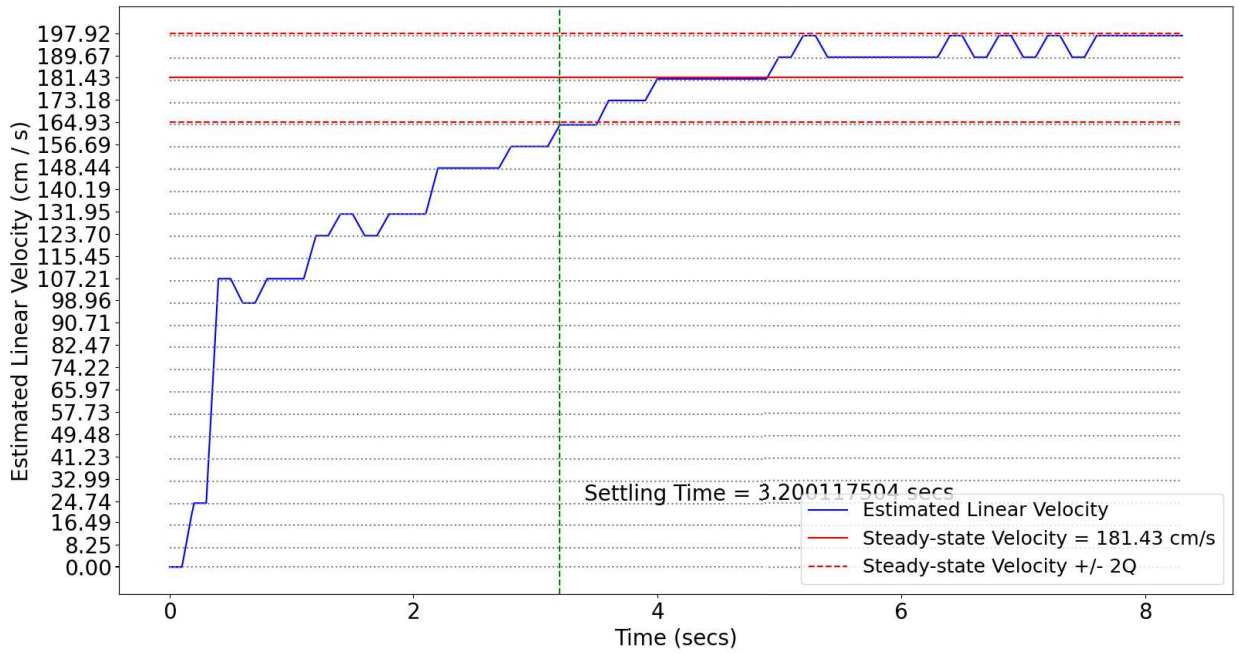


Figure 3.23: Closed-loop output-response with first PI Controller test for 181.43 cm/s desired velocity at 5Hz sampling.

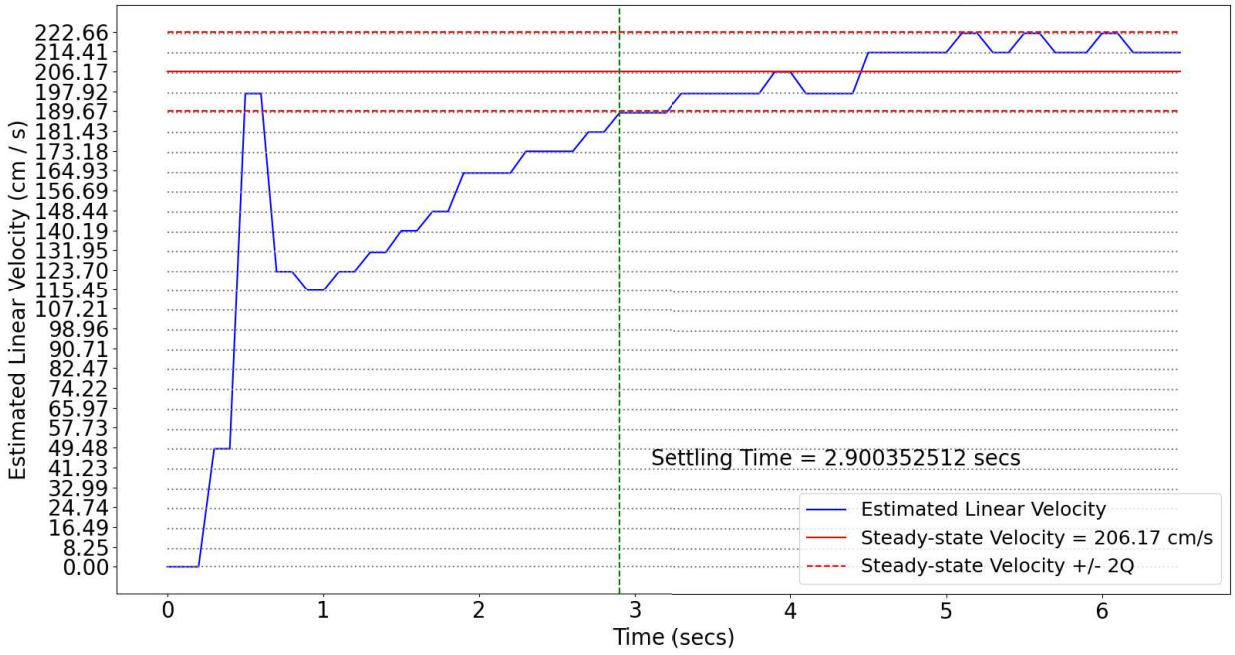


Figure 3.24: Closed-loop output-response with first PI Controller test for 206.17 cm/s desired velocity at 5Hz sampling.

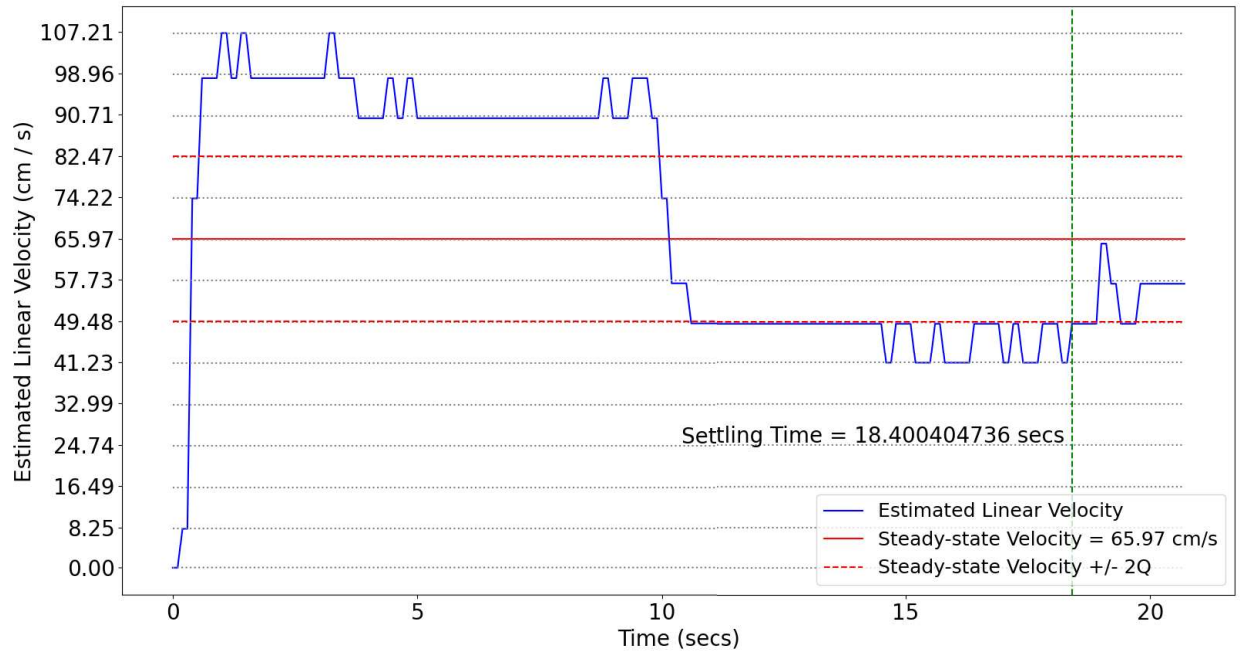


Figure 3.25: Closed-loop output-response with second PI Controller test for 65.97 cm/s desired velocity at 5Hz sampling.

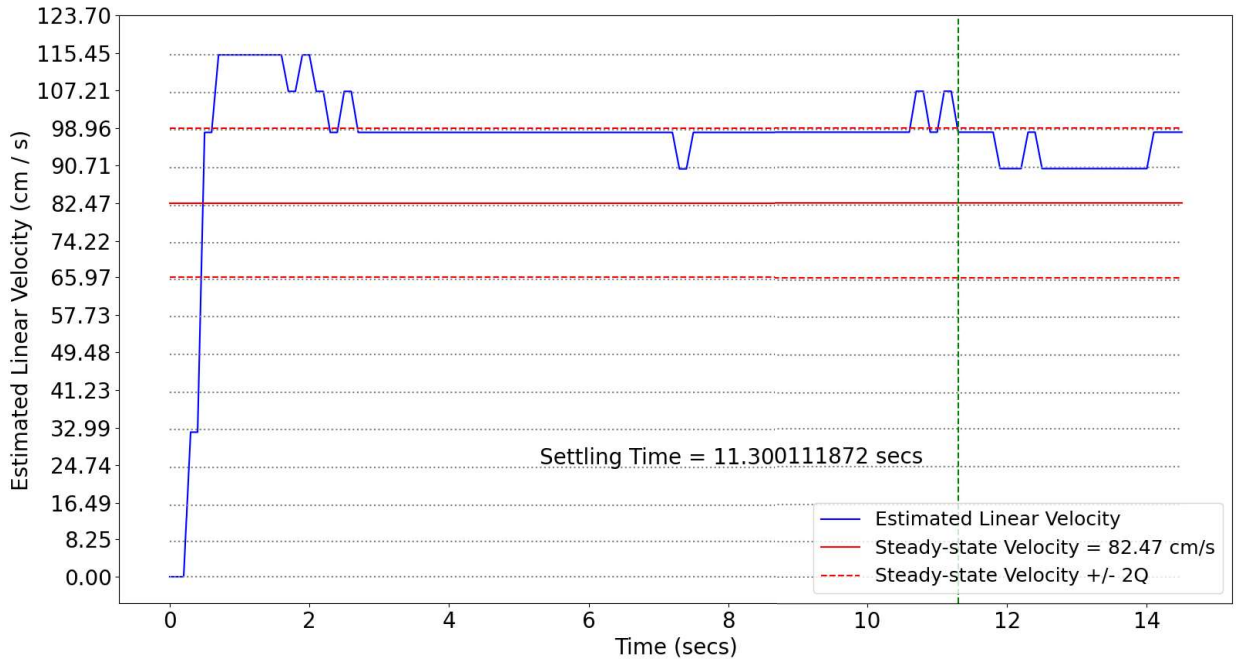


Figure 3.26: Closed-loop output-response with second PI Controller test for 82.47 cm/s desired velocity at 5Hz sampling.

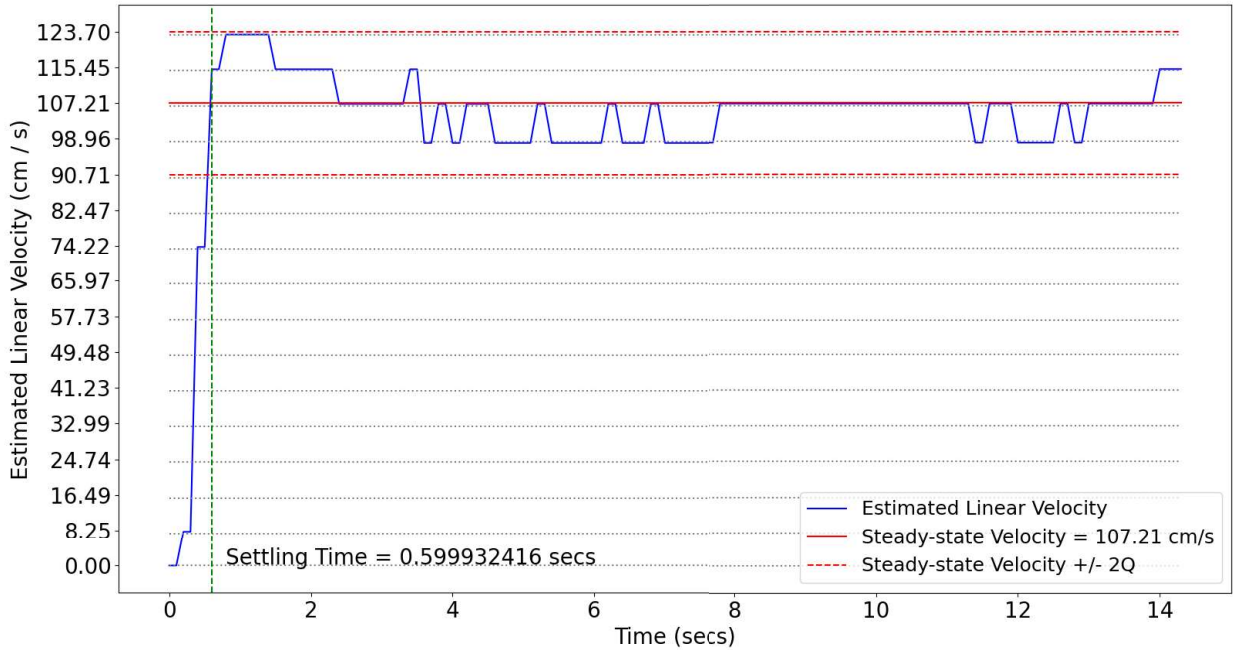


Figure 3.27: Closed-loop output-response with second PI Controller test for 107.21 cm/s desired velocity at 5Hz sampling.

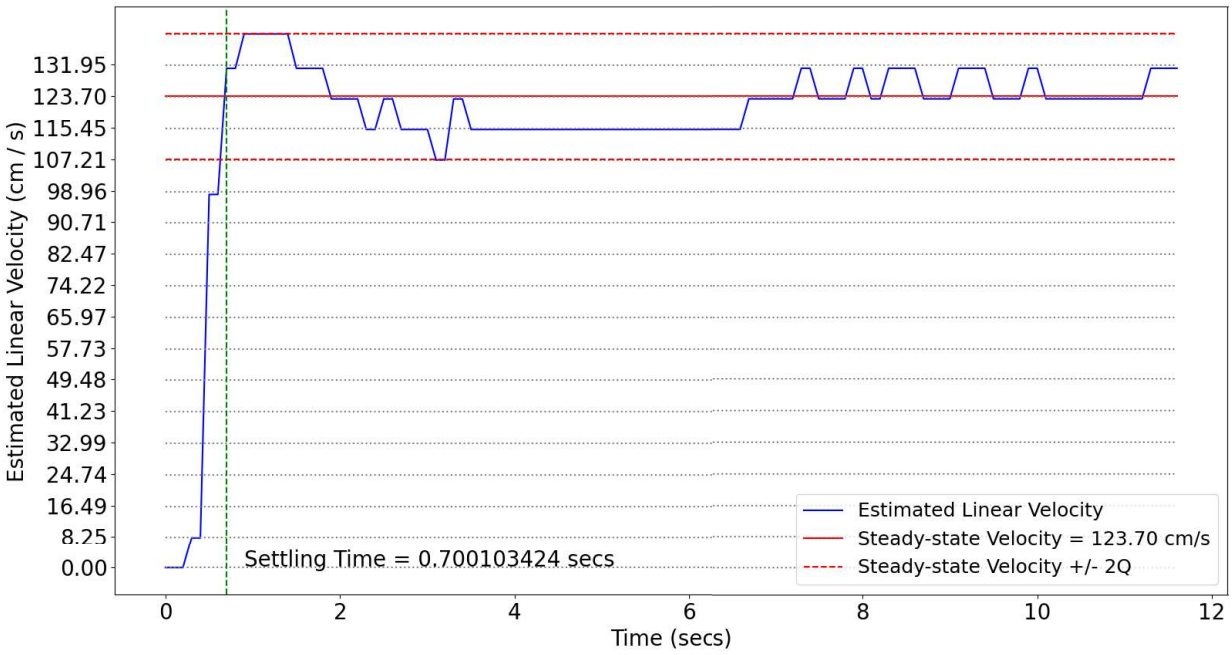


Figure 3.28: Closed-loop output-response with second PI Controller test for 123.70 cm/s desired velocity at 5Hz sampling.

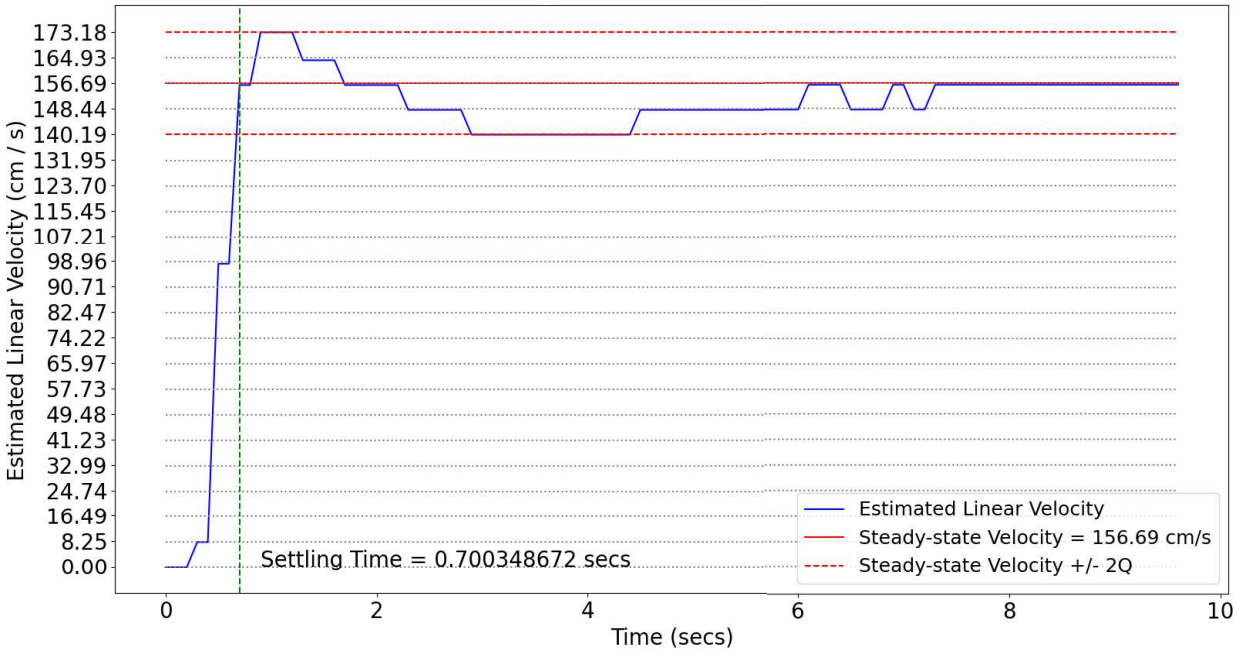


Figure 3.29: Closed-loop output-response with second PI Controller test for 156.69 cm/s desired velocity at 5Hz sampling.

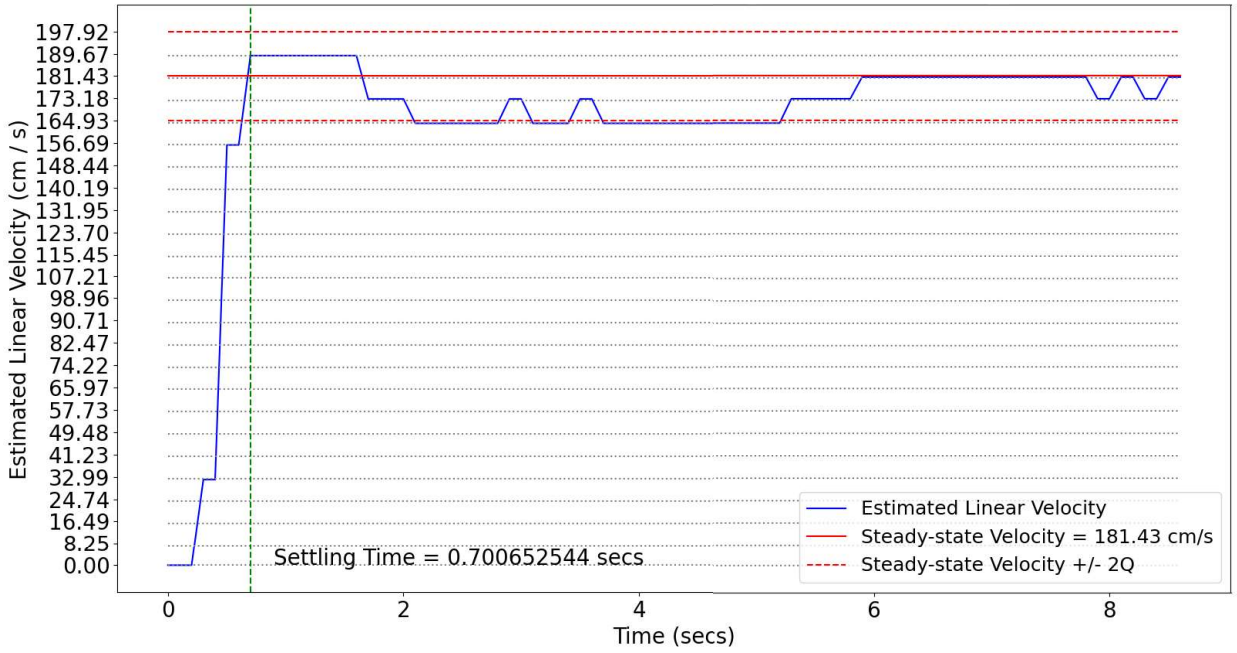


Figure 3.30: Closed-loop output-response with second PI Controller test for 181.43 cm/s desired velocity at 5Hz sampling.

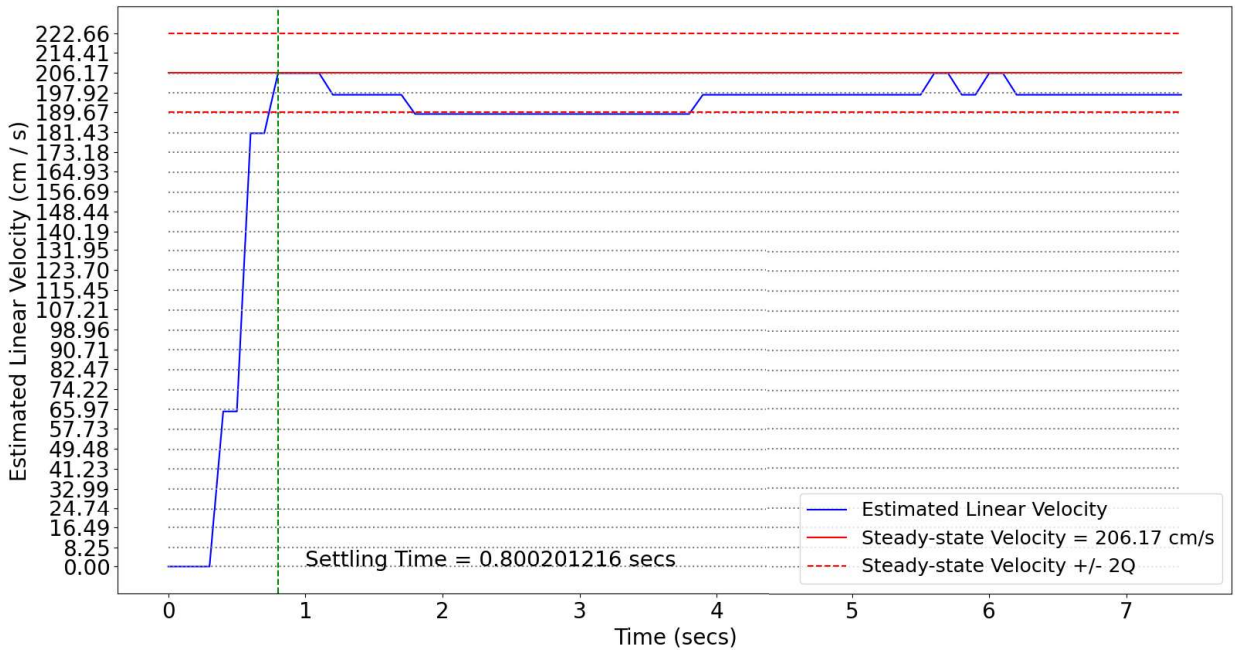


Figure 3.31: Closed-loop output-response with second PI Controller test for 206.17 cm/s desired velocity at 5Hz sampling.

3.8 Discussion

Multiple goals were reported in this chapter, regarding the linear velocity and dynamic identification of the platform. These achievements included the recognition of the vehicle's non-linear startup dynamic response or the so-called *uncertain dead zone* typical of the sensorless BLDC Motor and ESC systems. Furthermore, an adequate linear velocity estimation system was implemented, requiring the software and hardware design of an appropriate odometry system.

A proper PWM signal generator was also developed, considering the operational characteristics of the vehicle actuators, achieving a good-enough PWM High Time resolution for the proper velocity control of the platform. Moreover, these previous subsystems served as a baseline for the vehicle's startup behavior identification and the posterior recognition of adequate PWM startup High Time that could served as an offset for the posterior development of a linear velocity control system.

Finally, an analysis of the vehicle dynamic response was conducted with the intention of having a linear dynamic model, however multiple non-linear dynamic evidences were recognized that limited the adequate identification of a vehicle dynamic model. A heuristic method was then used to tune a PI controller that could regulate the vehicle's linear velocity. Experimental evidences show that it was necessary to develop two PI controllers, one for low speeds and other for high speeds.

The vehicle's linear velocity estimation and control was highly limited by the characteristics of the implemented platform. Moreover, the dynamic model identification and linear velocity control might be improved by substituting the former BLDC sensor-less motor with a shaft-sensor-BLDC motor [58] or a classic DC motor [59], avoiding the non-linear dynamics and startup behavior. However, the heuristically tuned PI controller was enough to obtain a regulated linear speed behaviour (at low speeds) required for the development of the autonomous driving algorithm and enabling the continuation with the road lane identification algorithm described in the upcoming chapter.

Chapter 4

Road Lane Identification and Autonomous Driving based on Computational Vision and Deep Neural Networks

4.1 Road Lane Identification and Autonomous Driving Process Description

Road lane identification is a basic step required for the autonomous driving. This process is based on the visual segmentation of the road lanes in an acquired environment image. Here, the segmentation is the process of recognizing the road lanes from a vehicle's front-view image perspective. The segmentation output is processed to identify left and right road lanes in a frame and then to decide what steering angle is needed to keep the AV platform in between them.

Multiple methods can be used both to segment and identify the road lanes on an acquired image. This methods include CV and DNN based algorithms [60] [61] [62] [63], allowing not only to recognize the road lanes in a given frame, but to directly generate a driving decision based on the input image [64] [65] [66]. These last proposals are mainly based on more complex processing mechanisms such as Deep Convolutional and Recursive Neural Networks, requiring high computational power. Nevertheless simpler Neural Network architectures were considered for the development of this research project as a demonstration of the capabilities of the AV platform, focusing mainly on the road lane segmentation process and serving as a baseline for further implementations of more robust autonomous driving mechanisms.

As aforementioned, one main goal was to implement both CV and DNN based road lane segmentation systems, to compare the performance that these proposals offer at different driving conditions. This allowed to identify the capabilities of the developed platform and recognize the autonomous driving performance achieved by these methodologies.

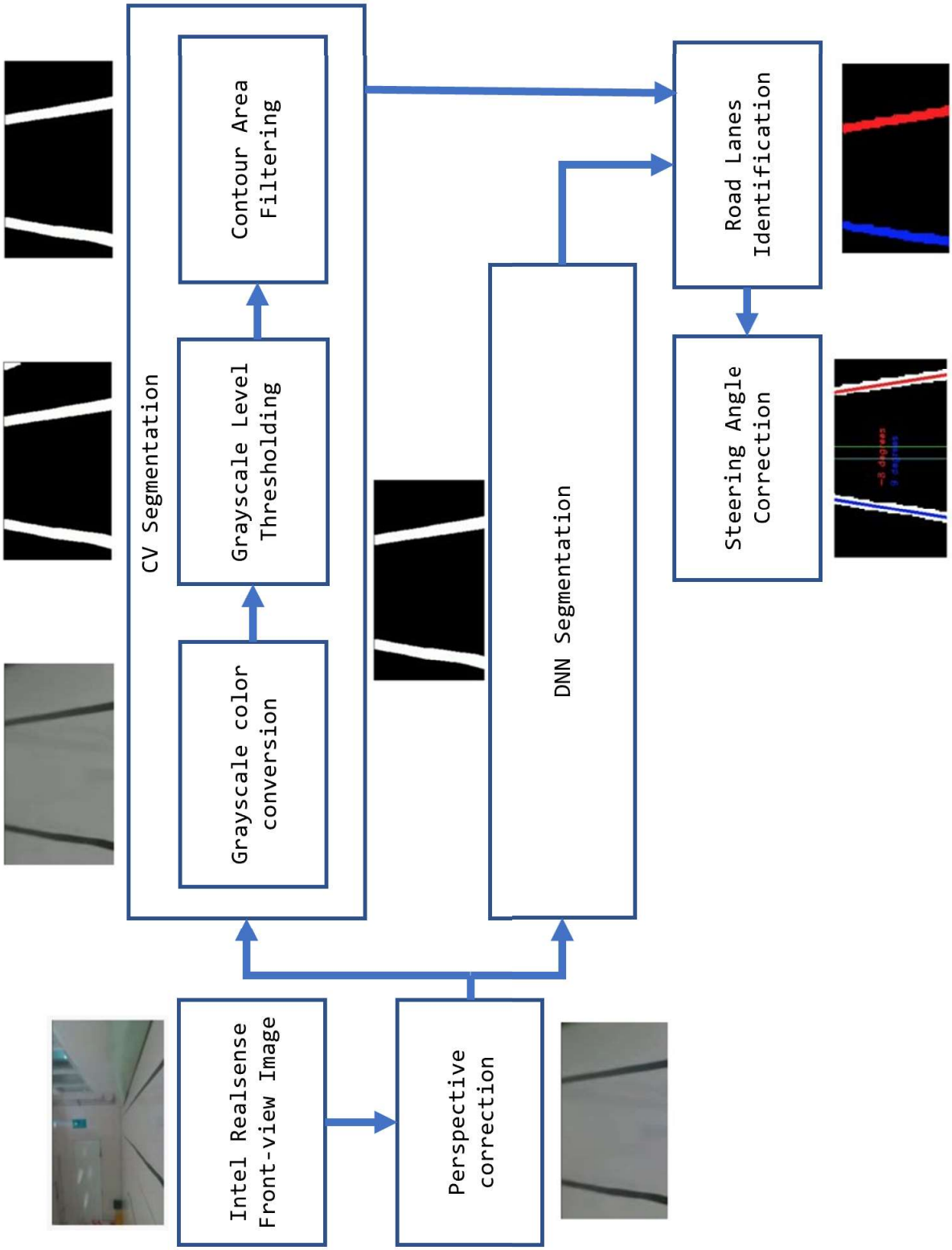


Figure 4.1: General Image Processing Pipeline.

The characteristics of the NVIDIA Jetson Nano board allowed an efficient integration of the required image processing, permitting the execution of a CV pipelines while driving based on the OpenCV Python library [42]. The image processing system was divided in five steps: (a) Intel Realsense front-view image capturing, (b) image perspective correction, (c) road lane segmentation (CV or DNN based), (d) road lane identification and (e) steering angle correction. See Figure 4.1.

On one hand, the CV segmentation proposal is based on the processing of the captured vehicle's front-view images by applying typical CV methodologies [67] [68] [69] such as blob or contour extraction, image filtering, color conversion, thresholding methods, etc. On the other hand, the DNN segmentation proposal is based on the implementation and training of a U-Net [70] Convolutional Neural Network capable of processing the acquired image and generate the desired segmentation output [71].

After segmenting the image with CV or DNN, the left and right lane identification is mainly based on CV classic methods. Conditions such as road curvature, straight trajectories or possible out-lane scenarios could be then evaluated to finally produce the steering angle correction required. An in-depth description of these processing phases will be presented in the upcoming subsections.

4.2 Vehicle's Front-View to Top-Down Perspective Transformation

The environmental image perception system consisted on the implementation of an image correction mechanism that could facilitate the road lane segmentation process required. To achieve this, a specific OpenCV perspective transformation method were used [69], permitting the acquisition of the vehicle's top-down view driving images based on the captured frames from the Intel RealSense SR300 BlasterX Senz3D Depth Camera [38].

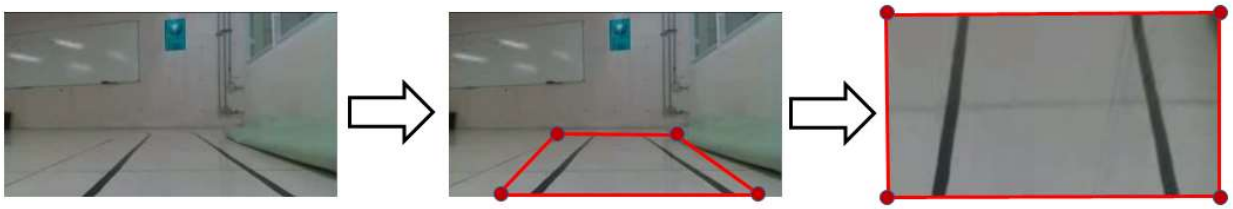


Figure 4.2: Perspective Transformation Procedure.

The perspective transformations used consisted of two steps, (a) extract the perspective transform matrix and (b) warp the source image using the transform matrix [72]. This image correction allows to define a ROI mask formed by four points from the original vehicle's front-view image. Such points were selected in a trapezoidal shape that would be distorted into the final top-down view image as represented in Figure 4.2. These vertex points were adjusted manually and iteratively, tested for multiple sample images and modified until a good-enough perspective correction was achieved, where both road lanes could be perceived inside a ROI, obtaining the top-down view as shown in Figure 4.2.

4.3 Road Lane Segmentation using Computater Vision

Once the perspective transformation is done, the next step is to segment the image into lane and no-lane sections with some degree of robustness at different light conditions [73] [74]. This segmentation could facilitate the identification of the individual road lanes in a given frame allowing to extract information such as the vehicle's position with respect to the road center and the current road curvature could be determined.

Therefore, a road was traced inside the Campus laboratories using a black tape pasted on the laboratory white floor (see Figure 4.3), while the illumination conditions were set using the laboratory light lamps placed along the whole track. The lamps are placed at the right, center, and left sections of the traced road, allowing to establish different illumination scenarios by modifying their state.

The vehicle was manually driven multiple times, traversing the traced road in both senses (clockwise and counter-clockwise) and at different light conditions. Simultaneously, the front-view images acquired throughout all the vehicle trajectories were recorded and classified based on the settled illumination. Finally, this recordings were utilized as a database for the development of the road lane segmentation system, allowing to count with multiple recordings of the different testing scenarios the vehicle would face.

The color contrast between the road lanes and the laboratory floor motivated the implementation of a color thresholding based segmentation system [10] [11] [60]. Moreover, this proposal could be tested and validated using the previously acquired recordings, comparing the achieved performance at different illumination scenarios.

The CV segmentation procedure consisted of three steps (see Figure 4.1) including (a) apply a grayscale image color conversion, (b) binarize the obtained image based on a predefined threshold, and (c) filter the detected contours based on a minimum detection area [75] [76]. Some examples of this road lane segmentation procedure can be observed in Figures 4.4, 4.5 and 4.6.

The vehicle top-down view images contained mainly dark and clear pixels given the traced road characteristics and the laboratory floor. Based on this, a grayscale representation of the current frame was obtained using the OpenCV library `cvtColor` method [77]. This color conversion could facilitate the posterior binarization of the image by applying grayscale color thresholding methods [75] [76].

As mentioned, the image binarization depended on specific grayscale color threshold values. These parameters would delimit the color values used to evaluate if a given dark pixel would be considered part of the road lane or not. Therefore a `MIN_LANE_COLOR_VAL` and `MAX_LANE_COLOR_VAL` values were required, defining the minimum and maximum dark grayscale pixel values to be segmented.

Figure 4.3: Vview of the used road track traced on the ITESM CEM Laboratories.

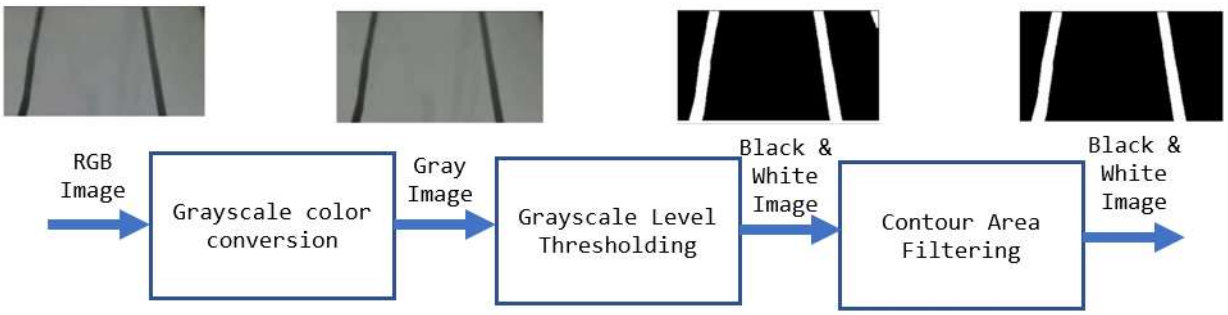


Figure 4.4: CV based Road Lane Segmentation Example 1.

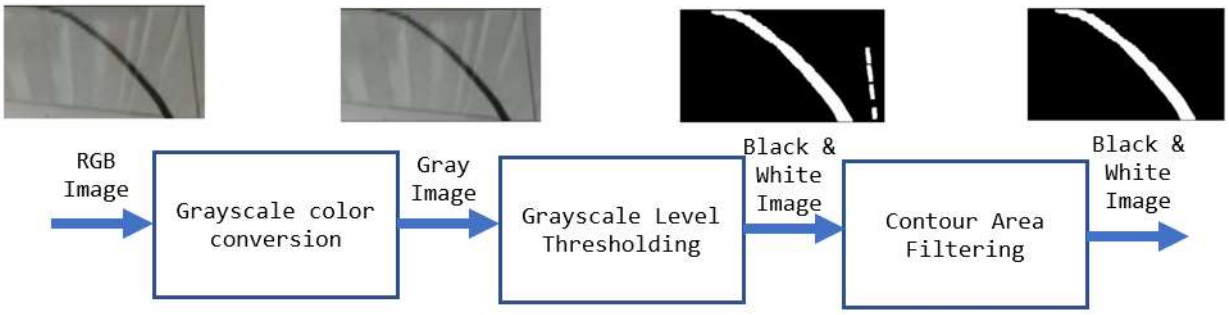


Figure 4.5: CV based Road Lane Segmentation Example 2.

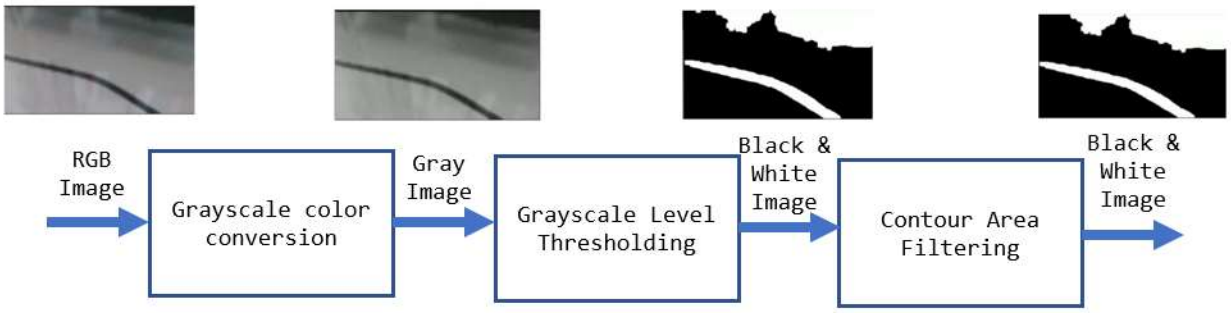


Figure 4.6: CV based Road Lane Segmentation Example 3.

An initial criteria was to establish the `MIN_LANE_COLOR_VAL` to 0, denoting the darkest possible pixel value which in general would be directly considered part of the searched black road lanes. However, in order to define the `MAX_LANE_COLOR_VAL` parameter an empirical methodology was applied, testing multiple maximum threshold values on the previously acquired vehicle trajectory videos. The goal was to test with different maximum threshold values and observe the acquired binarization outputs, finally the value that demonstrated the best apparent response throughout all the recordings was selected. This procedure yielded a `MAX_LANE_COLOR_VAL` value of 90, which was later validated by analyzing the grayscale histogram of specific top-down view frames.

An example of the mentioned histogram analysis can be observed in Figures 4.7, where the red vertical line represents the `MAX_LANE_COLOR_VAL` threshold value of 90. It is clear that the selected value succesfully divides the grayscale histogram in two classes: (a) the black grayscale pixels placed to the left of the red line and (b) the clear pixels located to the right. This two color value classes would be then used to binarize the current frame, setting to black all the clear pixels and to white the dark ones.

Nonetheless, this single frame analysis was not enough to validate that the selected threshold values would produce an appropriate segmentation response. Based on this, a histogram analysis was made with different frames at a variety of illumination conditions as shown in Figures 4.8, 4.9 and 4.10. It can be observed that the selected threshold parameters were good-enough to obtain the desired output, except at a very low illumination condition (Figure 4.10). In this scenario, the pixel intensities were considerably small, generally placed inside the established threshold limits and causing an undesired response. However, the histogram distribution showed up an apparent color class division at a lower maximum threshold value around 50, suggesting that it is still possible to adequately segment the frames if a different threshold range is used.

A deeper analysis of the threshold method response was made developing a cumulative histogram. This procedure consisted on summing the individual histograms of all the frames in each of the stored videos, to finally acquire a general cumulative histogram. The objective was to include all the detected color values throughout the whole set of light conditions tested, except those videos with the darkest light conditions. As shown in Figure 4.11, the cumulative histogram demonstrated that it is not possible to define a single `MAX_LANE_COLOR_VAL` value that could succesfully binarize any given frame at the stated illumination conditions. It can be concluded that the best possible threshold value would depend heavily on the current illumination condition, hindering the motivation to establish a single general threshold parameter.

Finally, the binarized image is filtered by contour areas [78] using OpenCV. A contour is any well-limited shape found in an image, having the possibility of extracting specific features such as its area, perimeter, centroid, etc. Any contour with an area smaller than a predefined `MIN_DETECTION_AREA` value shall be deleted from the binarized image as shown in Figures 4.4 and 4.5. This allows to diminish the presence of undesired small contours in the final segmentation output such as floor spots, dirt, etc.

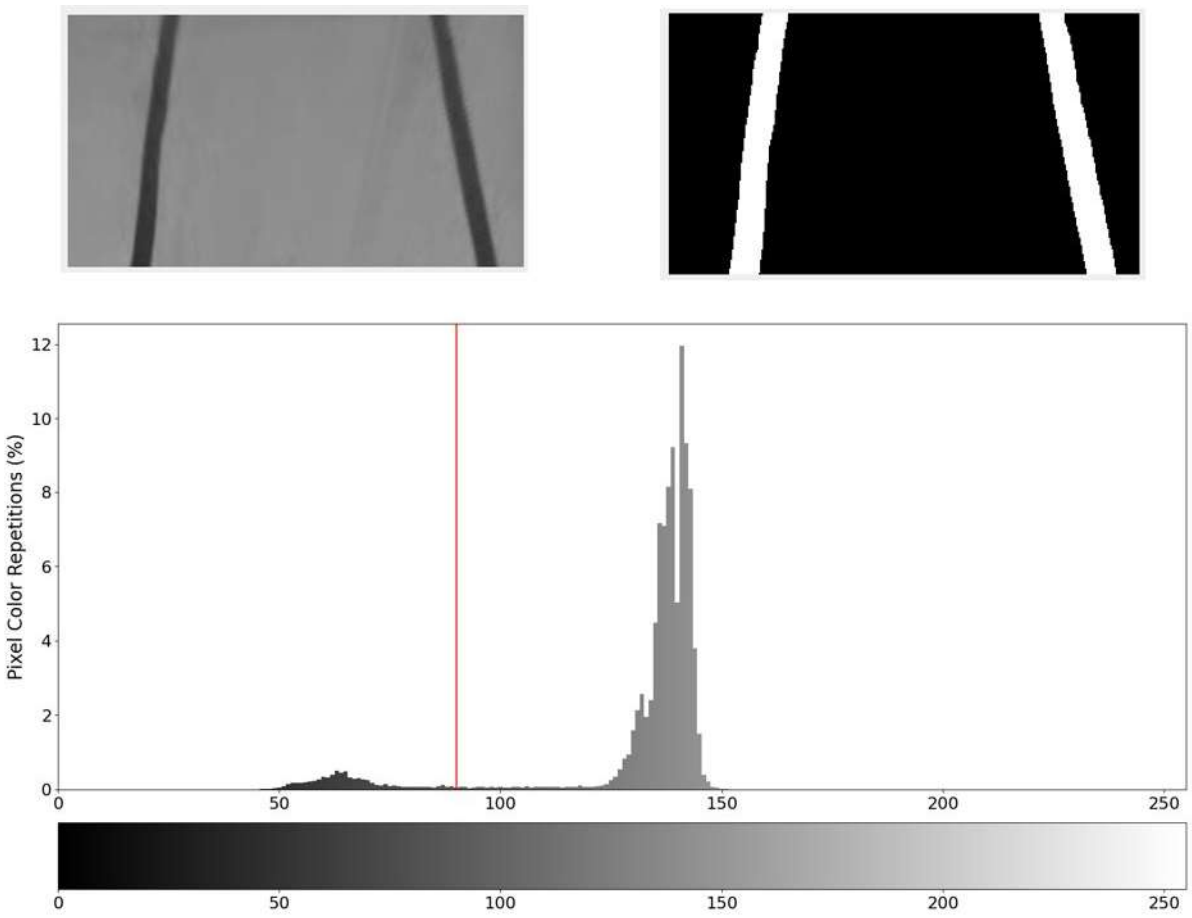


Figure 4.7: Single top-down view frame grayscale histogram at maximum illumination condition. A maximum thresholding value of 90 is illustrated with a vertical red line, the analyzed grayscale image can be observed at the upper left section of the Figure, while its binarized representation is shown at the upper right.

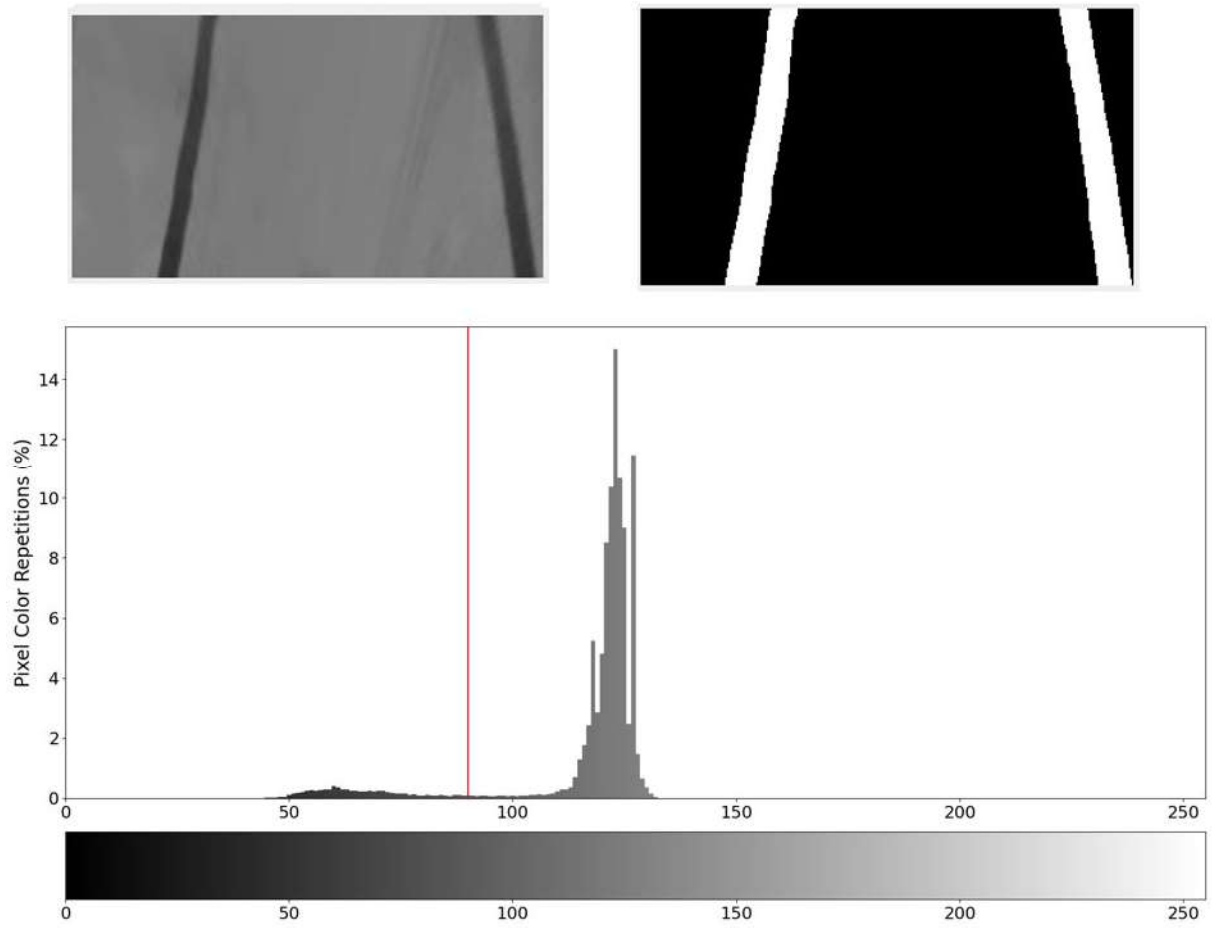


Figure 4.8: Single top-down view frame grayscale histogram at medium illumination condition. A maximum thresholding value of 90 is illustrated with a vertical red line, the analyzed grayscale image can be observed at the upper left section of the Figure, while its binarized representation is shown at the upper right.

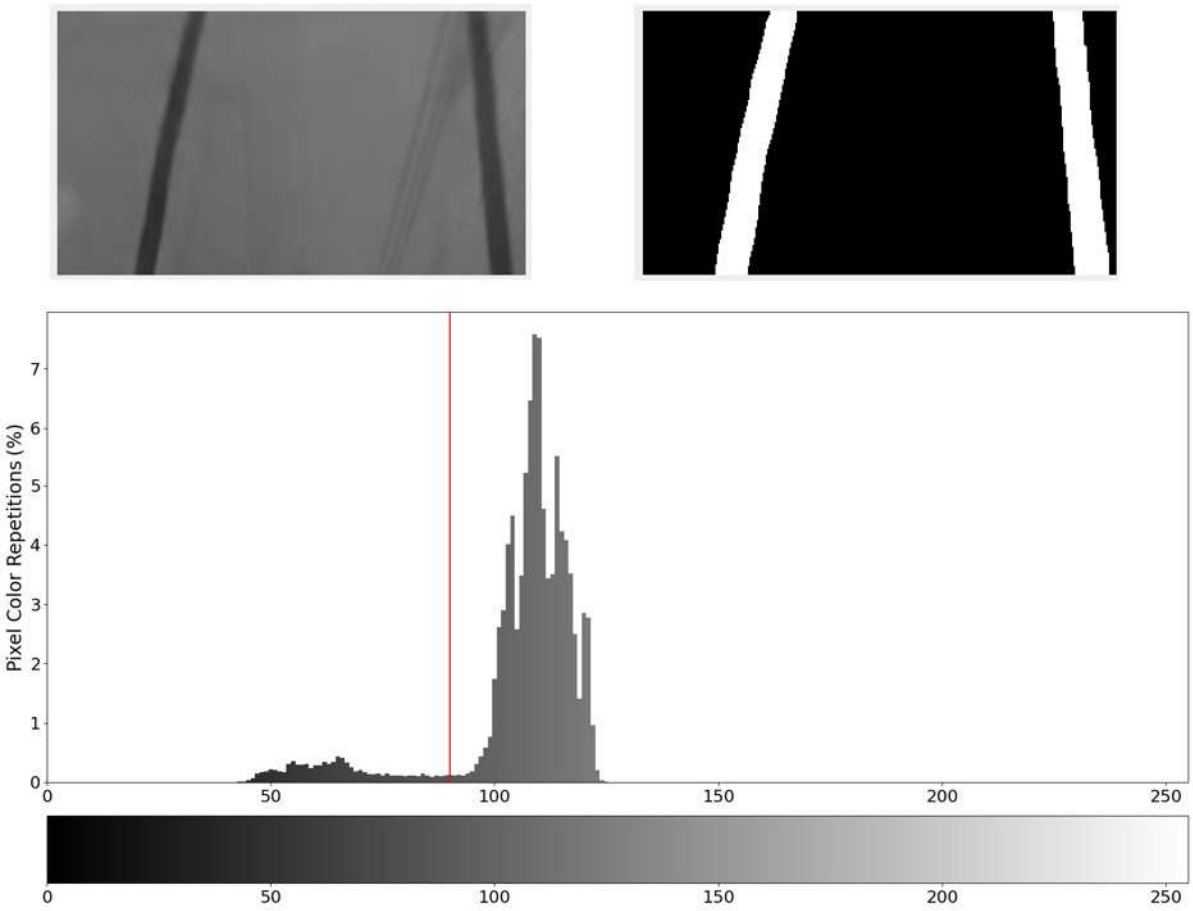


Figure 4.9: Single top-down view frame grayscale histogram at low illumination condition. A maximum thresholding value of 90 is illustrated with a vertical red line, the analyzed grayscale image can be observed at the upper left section of the Figure, while its binarized representation is shown at the upper right.

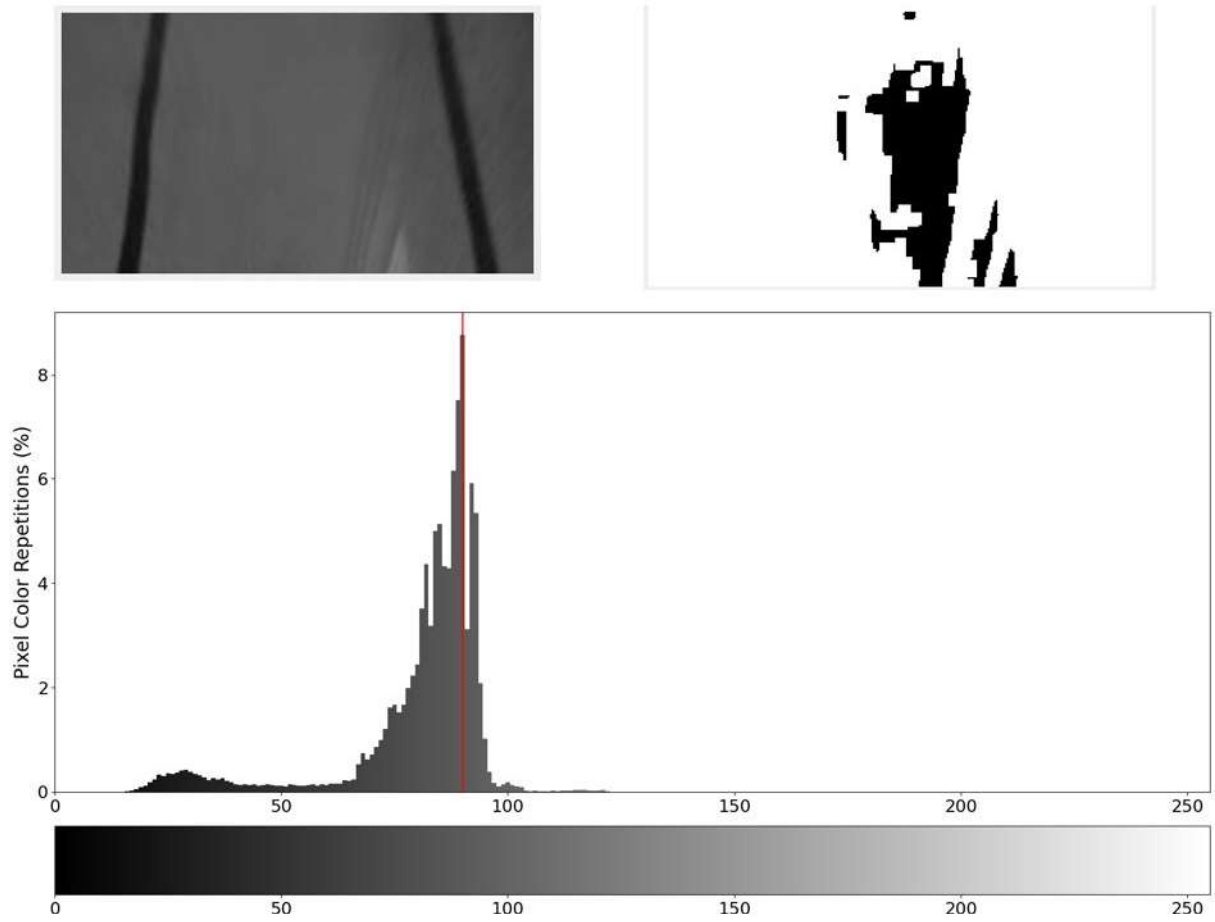


Figure 4.10: Single top-down view frame grayscale histogram at dark illumination condition. A maximum thresholding value of 90 is illustrated with a vertical red line, the analyzed grayscale image can be observed at the upper left section of the Figure, while its binarized representation is shown at the upper right.

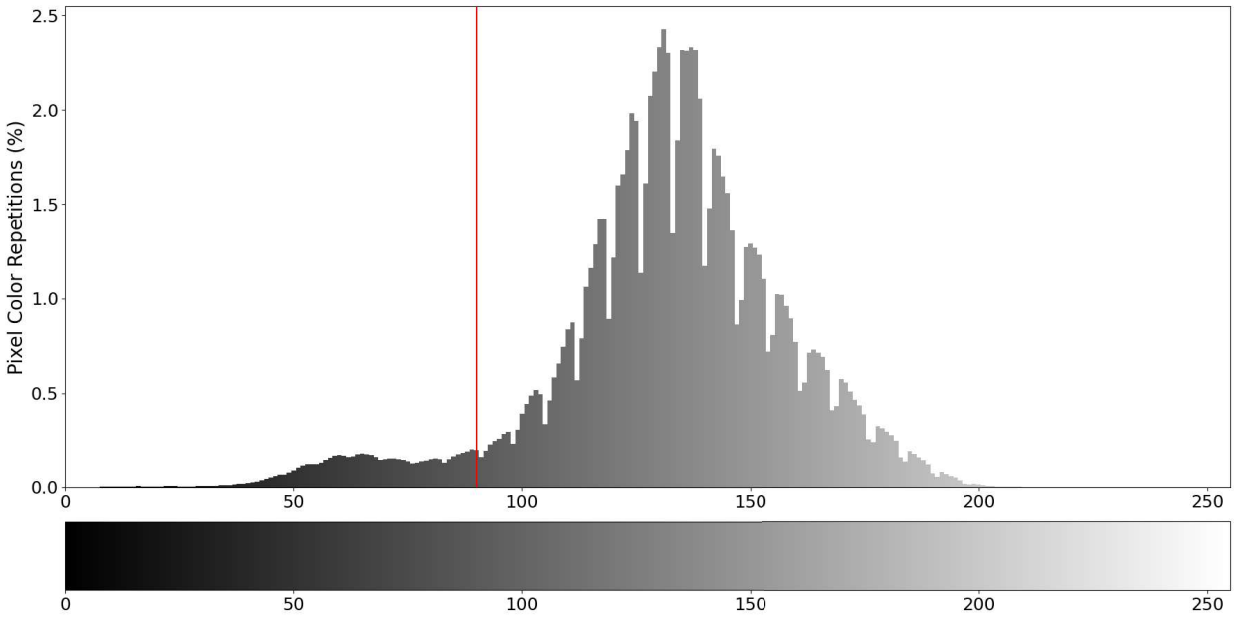


Figure 4.11: Cumulative grayscale color value histogram made by summing each individual frames grayscale color value histograms in all the recorded vehicle trajectory videos at multiple illumination conditions.

The reduction of undesired contours would help to improve the road lane identification performance, preventing the segmentation process to perceive irrelevant objects as part of the road lane. This behavior can be observed in Figure 4.5 where a small floor spot was filtered in the final segmentation output. The required `MIN_DETECTION_AREA` value was manually adjusted by testing different values and comparing the algorithm performance in multiple vehicle trajectory recordings.

However, bigger and irregular non-desired items could still be part of the segmentation, as shown in Figure 4.6. Even though more complex filtering procedures could have been used to reduce the stated undesired binarization response, they were not considered based on the desire to establish a simple CV process pipeline. Table 4.1 shows the three adjustable meta-parameters.

Table 4.1: CV based Road Lane Segmentation Configuration Parameters.

Configuration parameter	Value
`MIN_LANE_COLOR_VAL`	0
`MAX_LANE_COLOR_VAL`	90
`MIN_DETECTION_AREA`	200

4.4 Road Lane Segmentation using Convolutional Neural Networks

Deep Convolutional Neural Networks have demonstrated excellent efficiency in the development of AV platforms [62] [63]. Many self-driving car projects base their functionality on these kind of algorithms. DNN systems are generally used for complex tasks such as path planning, image identification, autonomous driving control, crash avoidance, etc. Despite the excellent performance these systems provide, they are generally computationally expensive. However, in many applications, they have proven to be better than other algorithm proposals such as CV based systems [79] [80].

As aforementioned, it was desired to implement both CV and DNN based road lane segmentation systems and compare their performance as a demonstration of the platform's capabilities. Convolutional Neural Networks offer excellent response for image segmentation tasks in the development of AV platforms, allowing the identification of multiple elements contained in a typical driving scenario. Such characteristics motivated the implementation of this kind of DNN paradigm for the vehicle's road lane segmentation system.

The implemented road lane segmentation system would be validated given multiple driving and light conditions traversing the traced road inside the ITESM CEM laboratory. Consequently, these criteria were taken into consideration for the adequate design of the DNN based segmentation system. It was needed to acquire an image dataset for the DNN training process, this dataset was formed by the individual 320 x 180 pixels frames of the multiple trajectory recordings made at low, medium, and high laboratory light conditions.

The obtained dataset consisted of 1422 top-down view BGR frames of the vehicle trajectories that were manually segmented using the `PixelAnnotationTool` software [81]. This tool allows to manually and quickly annotate large sets of images by providing specific markers as shown in Figure 4.12.

The established markers are used by the OpenCV *watershed marked* algorithm [82] to segment each component of the input images. The `PixelAnnotationTool` generates BGR images, assigning a specific color to each segmented element on the frame. This characteristic allows annotating more complex images such as a typical roads with multiple traffic signals, pedestrians, vehicles, trees, etc.

However, the project's application required only the segmentation of the road lanes and floor background facilitating the overall manual segmentation process. The segmented elements included (a) the road, (b) the road lanes, and (c) the road exterior as shown in Figure 4.13. The acquired training dataset images were posteriorly binarized by color using specific OpenCV color thresholding methods. This procedure allowed having black and white segmented training images similar to those obtained by the CV based segmentation method.

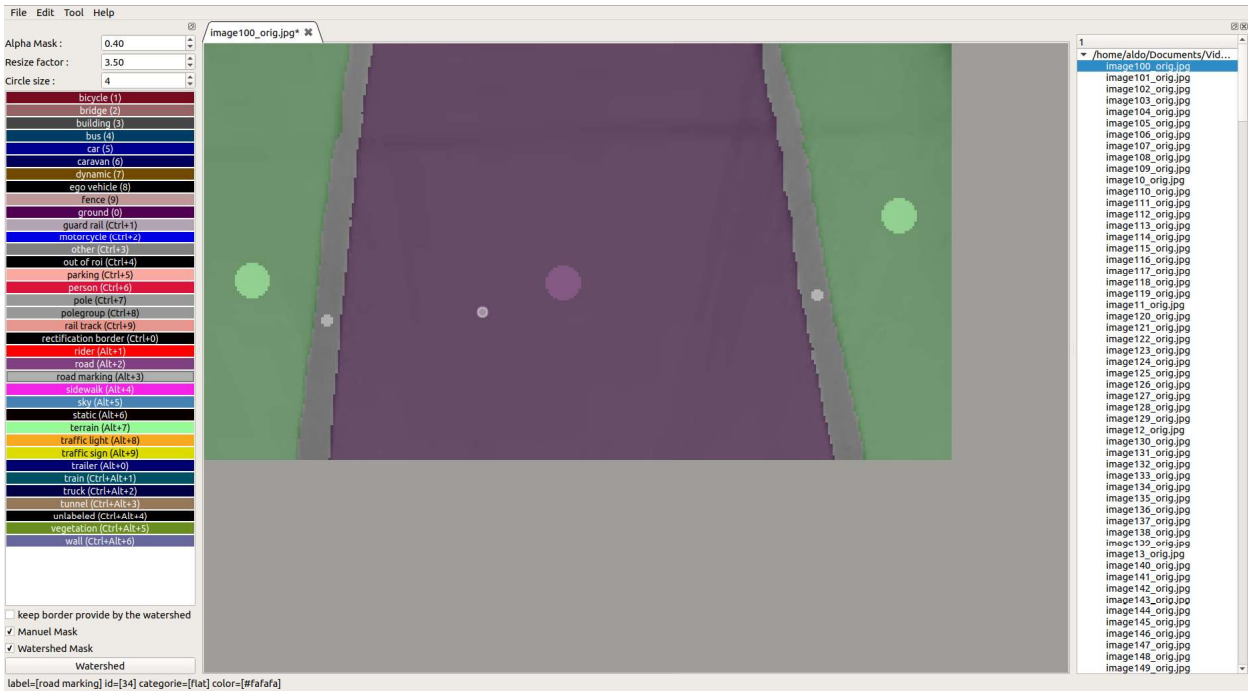


Figure 4.12: `PixelAnnotationTool` segmentation procedure example.

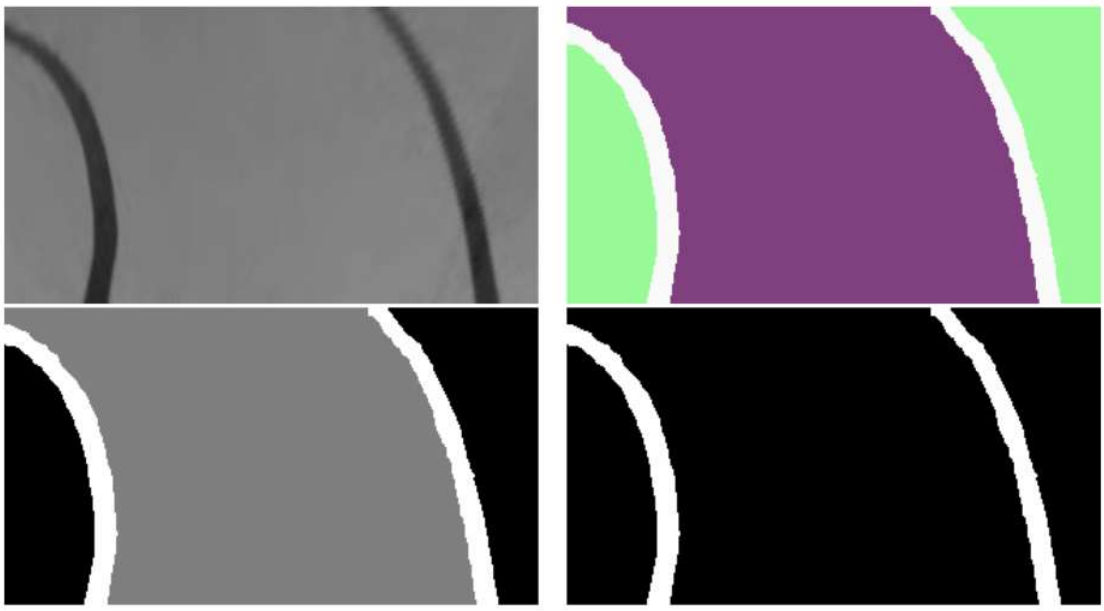


Figure 4.13: DNN training dataset output image pre-processing example.

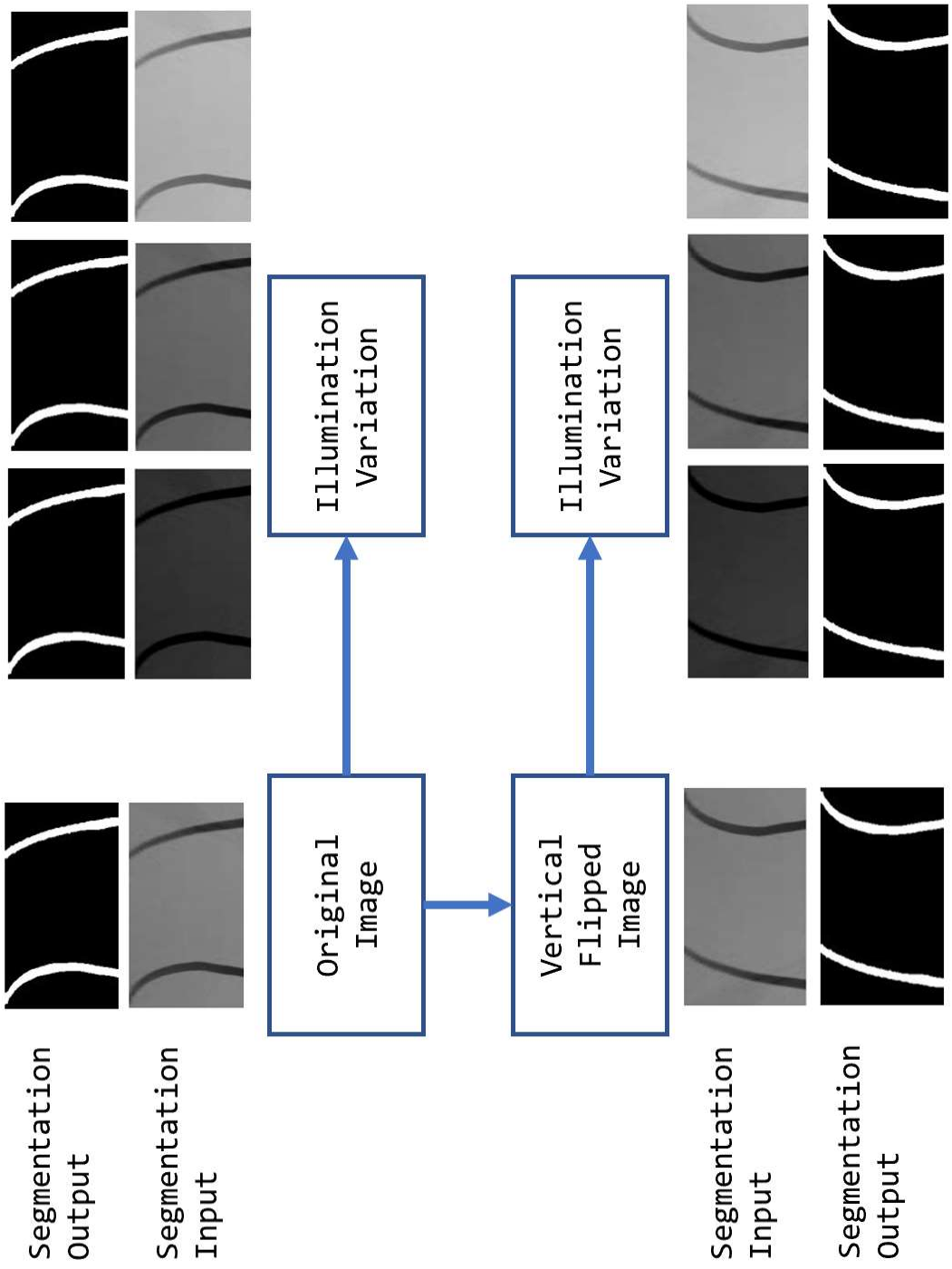


Figure 4.14: Data Augmentation Procedure Example.

In order to enhance the DNN's ability to correctly detect lanes, a data augmentation procedure was used to obtain a more complete dataset for training. This was achieved by computationally modifying the original images' orientation applying a vertical reflection and changing their illumination condition using `Tensorflow` specific methods as shown in Figure 4.14. The procedure generated a new training dataset up to 11376 images from the original 1422 manually segmented frames.

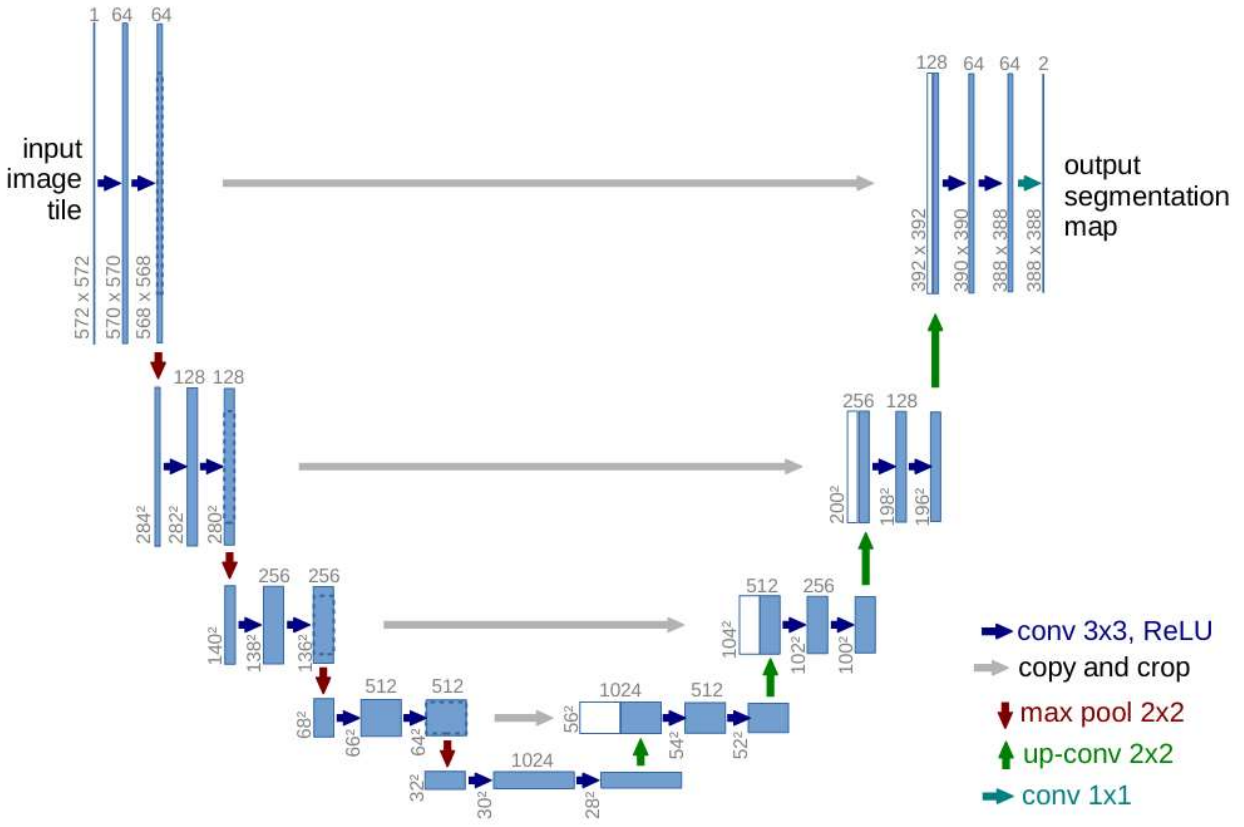


Figure 4.15: Typical U-Net architecture example. Each blue box correspond to a multi-channel feature map. The number of channels is denoted on top of the box. The frame size is provided at the lower left edge of the box. White boxes represent copied feature maps. The arrows denote the different operations. [2]

The implementation of the DNN system was made using the `TensorFlow` Python library [41], utilizing a typical U-Net Architecture [2] [70]. A U-Net is a highly efficient convolutional neural network specifically designed for image segmentation. These neural networks are based on the usage of contracting convolutional layers (feature maps extraction) and succesive expansive or deconvolutional layers (decoder), as shown in Figure 4.15, allowing the extraction of relevant image features and the generation of high resolution segmentation outputs. This network architectures require few training images and present excellent training performance when using a data augmentation training strategy [2].

As shown in Figure 4.15, typical U-Net architectures present a complex structure with multiple down and upsampling layers. Furthermore, these kinds of deep convolutional architectures are generally not suitable for mobile applications with limited computational power. However, models such as the MobileNet V2 [83] [84] offer good and efficient performance for low computational powered devices, based on the usage of depthwise separable convolutions [85] and shortcut connections between layers [84]. This characteristics motivated the usage of a MobileNet V2 system as the core of the required feature map/contracting layers of the implemented U-Net system.

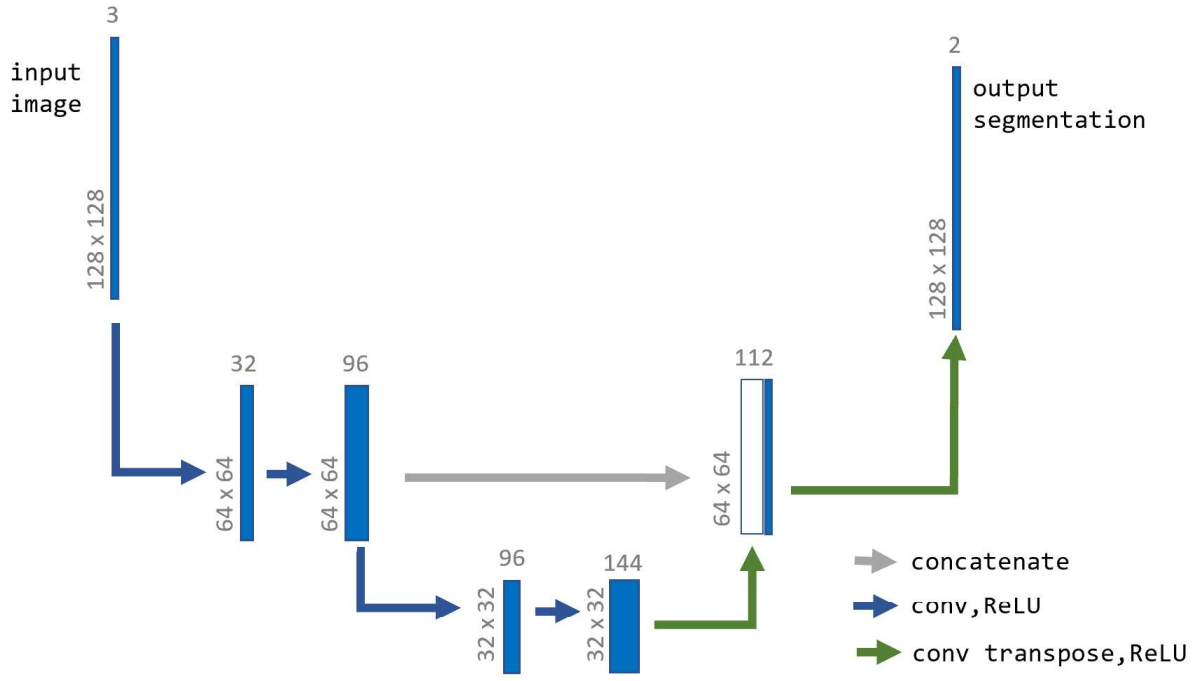


Figure 4.16: Deep Convolutional Neural Network segmentation model architecture.

The segmentation architecture proposed in this thesis can be observed in Figure 4.16, the DNN input image is the captured vehicle's top-down view BGR image resized to a 128 x 128 x 3 format, while the output is its segmented representation with only two color channels (128 x 128 x 2). Only two contracting and expansive layers are used to obtain the desired segmentation output. As mentioned, the contracting layers used the MobileNet V2 model and the expansive layers were based on typical deconvolutional layers. A single inner layer shortcut connection is used, allowing the propagation of relevant feature maps throughout the network and easing the overall training process.

The network training included a transfer learning strategy [86], meaning that the parameters of the feature map extraction layers of the network were assigned using a pretrained MobileNet V2 model provided by the `Tensorflow` Python library [87]. This methodology allowed having a more efficient and faster training of the network, requiring to train only the expansive or deconvolutional layers of the U-Net.

Before training, the whole dataset was randomly ordered. Then the network was trained using 90% of the previously sorted dataset. A specific testing dataset was defined to validate the model performance, utilizing 10% of the overall 11375 images dataset. The network training process was based on the Adam Optimizer and Categorical Crossentropy Loss Function. A training by epoch methodology was used, with 20 epochs and specific training batches of 120 images achieving a validation accuracy of 98.51%. A graph of the DNN model training and validation loss can be observed in Figure 4.19 obtained throughout the whole model training.

Some examples of the road lane segmentation response based on the trained DNN can be observed in Figures 4.17 and 4.18.

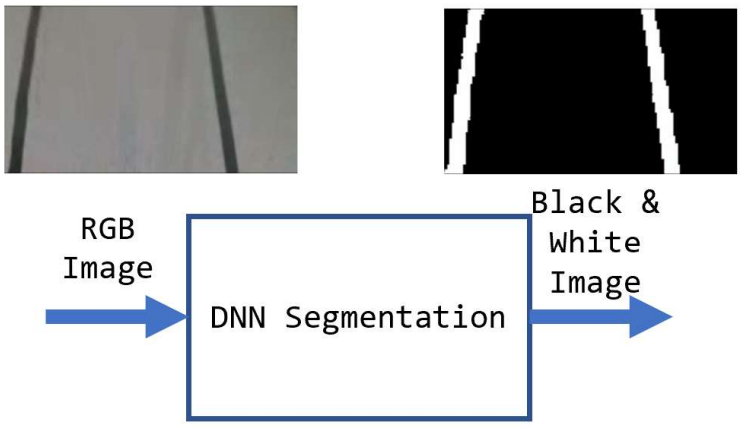


Figure 4.17: DNN based Road Lane Segmentation Example 1.

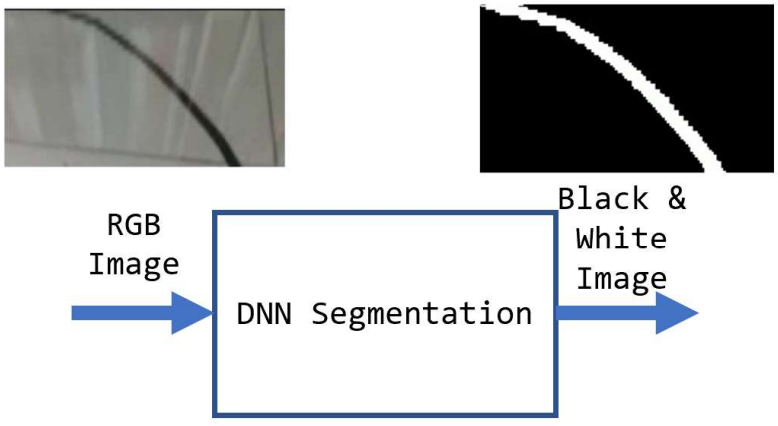


Figure 4.18: DNN based Road Lane Segmentation Example 2.

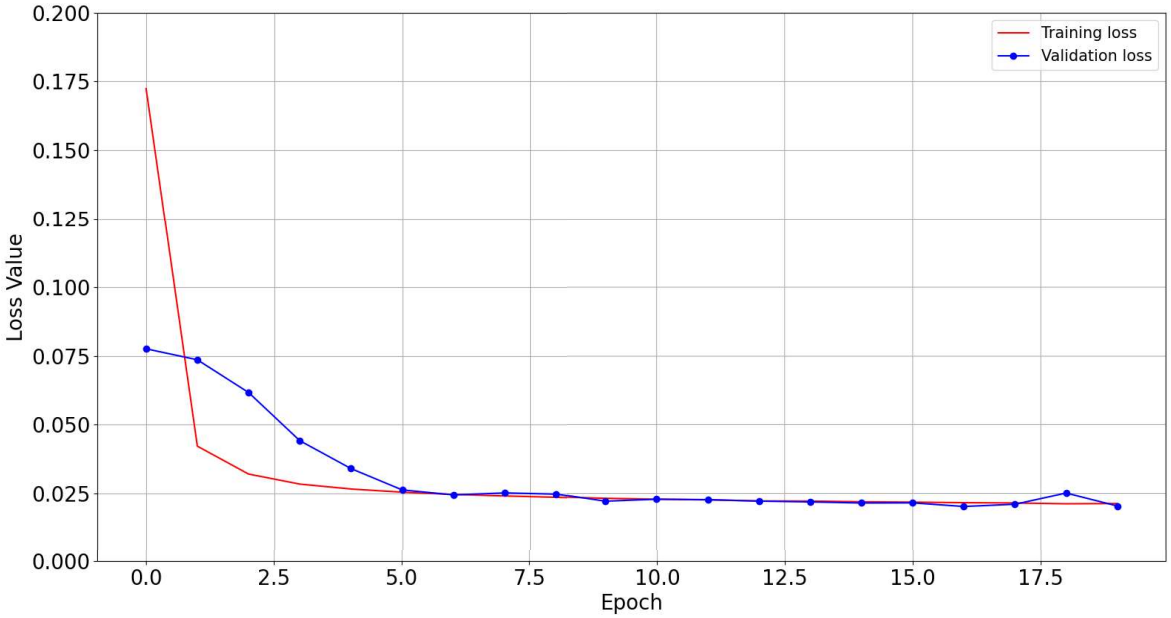


Figure 4.19: DNN training and validation loss graphs during training.

4.5 Road Lanes Identification and Classification

Road lanes identification and classification consisted of recognizing and tracking the road lane contours during a vehicle's trajectory using specific CV processing. The main objective is to properly identify left and right lanes in every single segmented top-down frame. In particular, five identification states were considered (a) `INIT`: Initial state, (b) `BOTH`: Both lanes are visible, (c) `RIGHT`: Only right lane is visible, (d) `LEFT`: Only left lane is visible, and (e) `NONE`: None of lanes are visible. This mechanism allows a posterior decision of steering angle correction.

Specific road-lane search-masks were used to extract the lane contours on a frame and used it in the next frame as a initial searching area to tacking it thought a video streaming. This searching strategy allows also to discriminate erroneous contours to be wrongly identified and permits a more efficient identification process constraining the road lane search space.

The mask-based tracking-lane method works as follows. Once a road lane contour has been found inside the image's masked region, a new lane identification mask is generated based on a Minimum Area Rectangle. A Minimum Area Rectangle is the bounding rectangle that best surrounds the whole lane contour with the minimum area required. The obtained rectangular mask is finally dilated to augment the research region for new frames. This dilation strategy facilitates recognizing the searched lane contour as it considers the possible displacement of the tracked lane in subsequent frames.

The `INIT` state is defined by an initialization condition where the road lanes identification has just begun. The algorithm must recognize and classify for the first time each of the road lanes on the current captured segmented frame. To achieve this, the vehicle is placed on a straight section of the road where both road lanes are clearly observable. The algorithm detects each road lane contour on the binarized image input by applying the specific road lane search mask. Considering the vehicle's initial position, each road lane would be located at the left or right half section of the top-down view segmented image, see Figure 4.20. This assumption facilitates the initial identification of the road lanes, defining the required lane search masks (blue and red rotated rectangles in Figure 4.20) by directly placing them at the left or right halves of the image.

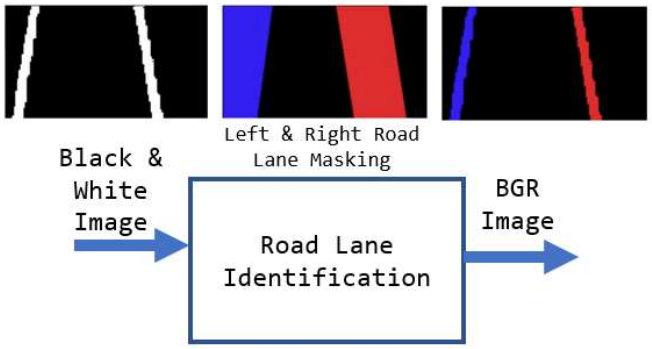


Figure 4.20: `INIT` Road Lane Identification Example.

The `BOTH` state reflects a condition where both road lanes have been already identified as shown in Figure 4.21. This classification state will occur every time the algorithm has previously found both of the searched road lanes. In this case, the last frame's left and right lanes search masks would be used as a dynamic-ROI to find the road lanes in a next captured frame. The identification is based on the premise that no considerable change in the road lanes location would be perceived from one frame to another. This allows using the previous search masks to find the new lane position in the current image, updating the search masks once the new lane contours have been found.

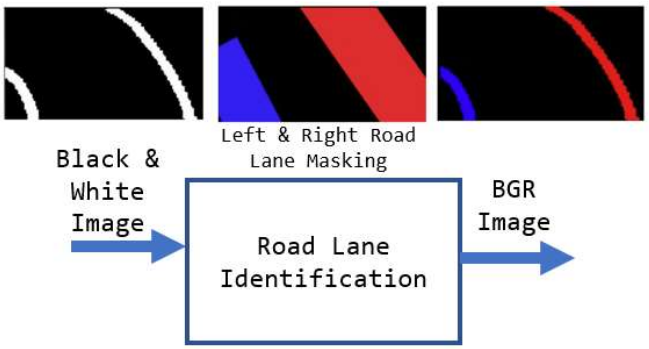


Figure 4.21: `BOTH` Road Lane Identification Example.

The `RIGHT` and `LEFT` states occur when only one of both road lanes has been identified. These states may occur during a curved section when just one road lane is perceivable or in an undesired out-road condition where the vehicle is close to a road edge or about to leave the road. In these cases, only a single lane identification mask is defined as shown in Figure 4.22. In general, a `RIGHT` or `LEFT` condition is achieved when any of the lane contours could not be found given the predefined search masks.

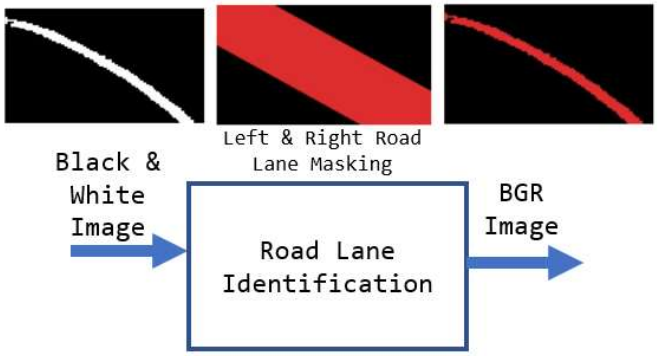


Figure 4.22: `RIGHT` Road Lane Identification Example.

The goal in the `RIGHT` and `LEFT` states is to keep on tracking the single identified lane and try to recover the missing one using a stronger steering correction. This missing lane recovery is based on the current identified lane condition, such as its location and slope. For instance, if a `RIGHT` state occurs and the identified right road lane is located sufficiently to the right in the current frame and with an adequate slope (see Figure 4.23), then a new left lane search mask will be suggested. The suggested missing lane mask would be generally placed at the right-most or left-most section of the image until the lane contour is recovered as shown in Figure 4.23 where the missing left contour was recovered using a left-most section placed search mask in blue.

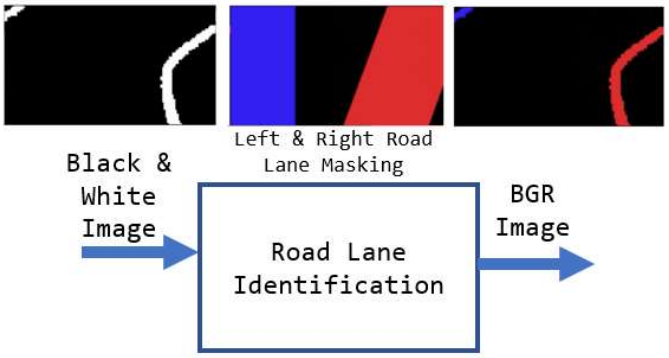


Figure 4.23: Road Lane Identification curved road section Example 1.

An important consideration made was the management of possible search masks overlapping. This condition may happen whenever both lane search masks are considerably close. This overlapping condition could be the product of an unreliable road lane segmentation output, occurring typically during tight road curves. For instance, consider that a tight left curve is traversed, the algorithm may initially recognize the left lane contour successfully. However as the vehicle traverses the road curve, the perceived lane may cross the whole frame from left to right horizontally, see Figure 4.24. This state may produce the segmentation system to progressively divide the single lane into multiple contours, causing an incorrect identification of the newly split contour as the missing right lane.

In order to avoid this incorrect road lane identifications, an analysis of the road lane search masks slopes is made. A linear projection of the identification masks slopes is made, evaluating if the generated line projections intersect inside the current frame limits. If the intersection condition is fulfilled, the algorithm determines that the two masks correspond to the same lane contour and discriminates the newly split contour preventing a miss-classification.

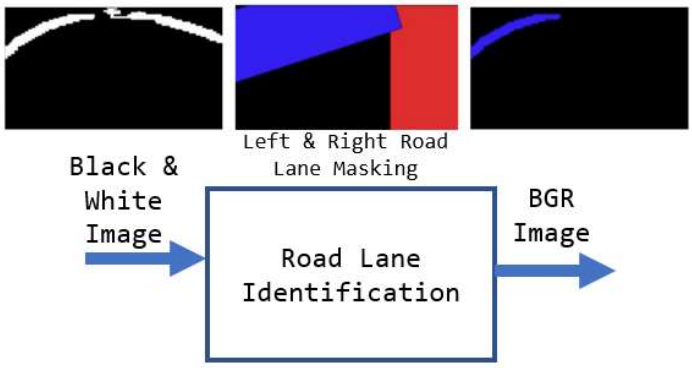


Figure 4.24: Road Lane Identification Curve Mask Example 2.

Finally, the `NONE` condition is an undesired state where both lanes are missing and mainly occurs when the vehicle has completely left the road. A `NONE` state will always be preceded by a condition where at least one lane was still recognized, implying that a previous road lane identification mask is available. The algorithm would then be conscious of the last seen road lane's possible location based on its previous mask, allowing the algorithm to search for this last seen road lane in new frames.

4.6 Identification and Correction of the Steering Angle

The road lane identification facilitated the recognition of specific vehicle driving conditions based on the road lane contours centroid location and slope. This information served to generate the required steering angle correction for the autonomous driving of the vehicle. As aforementioned, the Minimum Area Rectangle or Minimum Bounding Rectangle is a rotated rectangle that encloses a more complex shape using the minimum possible area [88].

The described method allows to define a regular and more simple shape from the detected lane contour and simplify its slope estimation, as displayed in Figures 4.25, 4.26 and 4.27, showing the obtained slope angle in read and blue on the last image of the processing pipeline. The obtained slope from both road lane contours is then used to identify and correct the current vehicle's steering angle based on the present identified lanes.

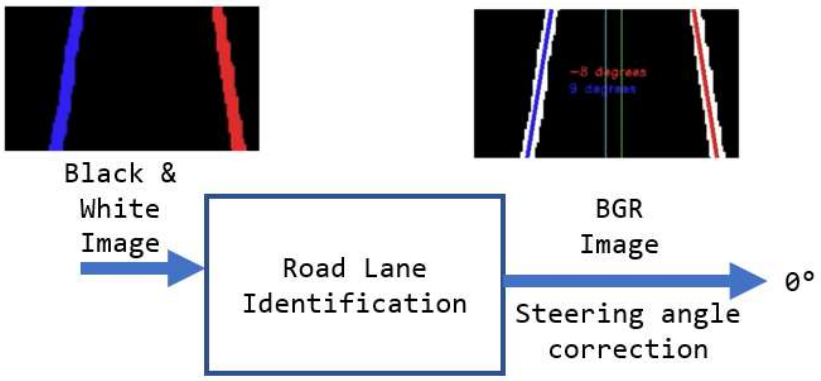


Figure 4.25: Road Lane Identification and Steering Angle Correction Example 1.

If both road lanes are vertical and parallel then the steering angle should be zero. If both road lanes are vertical but not parallel then steering angles should be the average of left and right bounding-boxes slopes. Special cases appears when only one lane is visible, as shown in Equation 4.1. If one of both road lanes is missing then the correction will depend entirely on the detected road lane slope and an specific `ANG_LANE_OFFSET`, see Figure 4.27. This strategy allows a smooth driving response during straight and curved sections of the road, applying a bigger steering correction in situations where only a single road lane has been recognized as an attempt to recover the missing lane. Nonetheless, it was also desired to achieve a driving behavior in which the vehicle remained at center of the road during the whole trajectory.

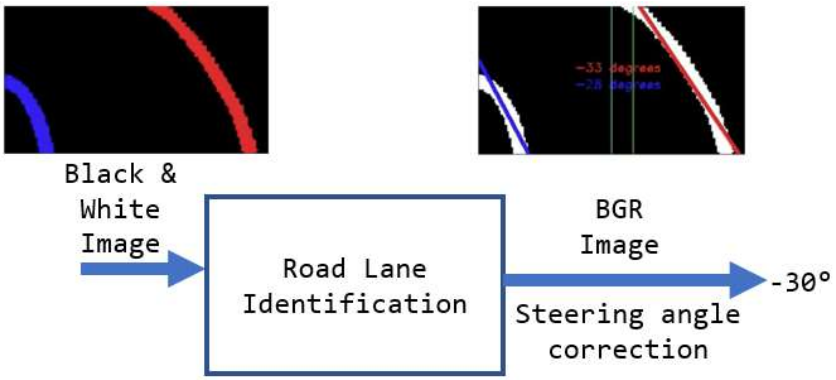


Figure 4.26: Road Lane Identification and Steering Angle Correction Example 2.

$$angle\_error = \begin{cases} \dfrac{left\_lane\_slope + right\_lane\_slope}{2} & \text{if } \texttt{BOTH} \\ left\_lane\_slope + \texttt{ANG\_LANE\_OFFSET} & \text{if } \texttt{LEFT} \\ right\_lane\_slope - \texttt{ANG\_LANE\_OFFSET} & \text{if } \texttt{RIGHT} \end{cases} \tag{4.1}$$

Where:

$left\_lane\_slope$ = angle of left bounding rectangle;
$right\_lane\_slope$ = angle of right bounding rectangle;
ANG_LANE_OFFSET = angle correction offset.

To determine if the car is at the center, a *middle_point_error* was computed. This error is the difference between the frame's center and the average of left and right centroids, see (4.2). However, depending on the current driving scenario, it is not possible to identify both road lanes at the same time, as shown in Figure 4.27, hindering the estimation of the road's center position. To cope with this situation, the estimation would depend on the current lane identification state. Therefore if only the right road lane is recognized, the central point estimation would be made by calculating the mean between the right lane centroid and the left-most location of the analyzed frame.

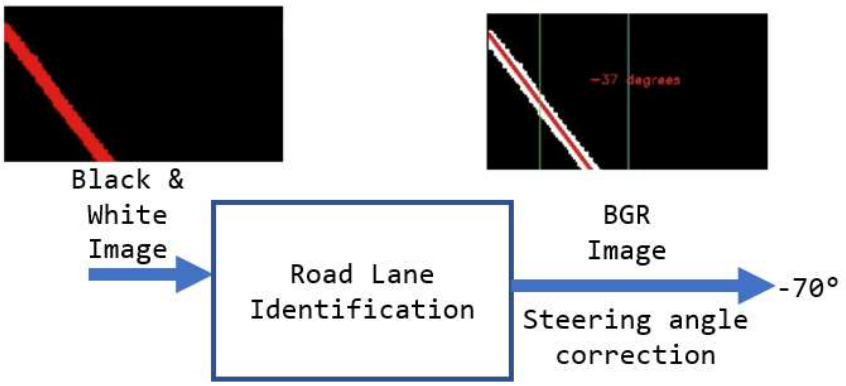


Figure 4.27: Road Lane Identification and Steering Angle Correction Example 3.

$$middle\_point\_error = lane\_middle\_point - frame\_middle\_point \tag{4.2}$$

Where:

$$lane\_middle\_point = \begin{cases} \dfrac{left\_lane\_centroid + right\_lane\_centroid}{2} & \text{if } \texttt{BOTH} \\ \dfrac{left\_lane\_centroid + frame\_right\_most\_point}{2} & \text{if } \texttt{LEFT} \\ \dfrac{frame\_left\_most\_point + right\_lane\_centroid}{2} & \text{if } \texttt{RIGHT} \end{cases} \tag{4.3}$$

$lane\_middle\_point$ = estimated road lanes middle point;
$left\_lane\_centroid$ = estimated left lane centroid horizontal point;
$right\_lane\_centroid$ = estimated right lane centroid horizontal point;
$frame\_left\_most\_point$ = frame's left most horizontal point;
$frame\_right\_most\_point$ = frame's right most horizontal point;
$middle\_point\_error$ = estimated centered driving error;
$frame\_middle\_point$ = frame's horizontal middle point.

Both the lane slope error and the vehicle's centered error were proportionally combined to produce the steering error correction, using the equations:

$$steering\_angle\_correction = \begin{cases} \texttt{ANG\_NO\_LANE\_OFFSET} & \text{if } \texttt{NONE} \text{ and previous } \texttt{LEFT} \\ -\texttt{ANG\_NO\_LANE\_OFFSET} & \text{if } \texttt{NONE} \text{ and previous } \texttt{RIGHT} \\ init\_steering\_angle & \text{Otherwise} \end{cases} \tag{4.4}$$

$$init\_steering\_angle = \texttt{ANGLE\_P} * angle\_error + \texttt{MIDDLE\_P} * middle\_point\_error \tag{4.5}$$

Where:

$\texttt{ANGLE\_P}$ = angle correction proportional constant;
$\texttt{MIDDLE\_P}$ = central driving correction proportional constant;
$\texttt{ANG\_NO\_LANE\_OFFSET}$ = offset when no lane is detected.

As shown in (4.5), a Proportional Control was tuned to guarantee an adequate driving response. Two proportional constants were defined: an `ANGLE_P` for the steering angle correction based on the current road lane slopes and a `MIDDLE_P` for the vehicle's centered driving error. Furthermore, a final consideration was made when facing a `NONE` identification state, as shown in Equation 4.4 where the maximum steering correction `ANG_NO_LANE_OFFSET` is made in an attempt to recover the last seen road lane.

These driving commands are calculated in the Jetson Nano board and published using specific ROS topics to be finally fed to the vehicle's low-level control units to achieve the desired autonomous driving behavior. The required `ANG_LANE_OFFSET`, `ANG_NO_LANE_OFFSET`, `ANGLE_P` and `MIDDLE_P` correction parameters were experimentally adjusted by testing the autonomous driving performance and validating the correct steering control until the vehicle was capable of traversing the traced road continuously. The selected configuration parameters can be observed in Table 4.2, while the final implemented code can be reviewed in [39]. Examples of steering angle corrections are shown in Figures 4.25, 4.26 and 4.27.

Table 4.2: Steering Correction Control Configuration Parameters.

Configuration parameter	Value
`ANG_LANE_OFFSET`	20
`ANG_NO_LANE_OFFSET`	60
`ANGLE_P`	0.3
`MIDDLE_P`	0.25

4.7 Discussion

In this chapter the processing capabilities of the developed AV platform were tested, implementing interesting autonomous driving mechanisms based on a road lane following strategy. Relevant tools such as CV and DNN based image processing were considered for the development of the final Road Lane Identification and Autonomous Driving system. Moreover, state of the art systems such as Convolutional Neural Networks and Computer Vision image segmentation were studied to generate an image processing pipeline capable of solving the desired complex task.

An image processing pipeline was defined, dividing the autonomous driving strategy in specific sub-steps. This processing steps were individually tested and validated using multiple manual driven vehicle trajectory recordings. This strategy allowed the modification and adjustment of the developed software when necessary in order to achieve an adequate overall performance, being specially useful for the development of the road lane segmentation systems.

A CV segmentation system was implemented based on the empirical selection of a general grayscale threshold value. The system demonstrated a good response throughout the whole set of recordings. However, it was evident that defining a single threshold value is not

enough for the adequate road lane segmentation specially at worst illumination conditions. This last could imply that the segmentation response would vary considerably specially if darker testing conditions were settled. More robust CV segmentation mechanisms could have been used, but they were left aside for further research.

A DNN segmentation system was also implemented based on a small U-Net architecture specially trained by utilizing data argumentation and transfer learning strategies, taking advantage of a previously trained MobileNet V2 network. Even though more complex network architectures could have been used for better performance, they could imply the usage of a deeper network with considerably more layers.

Given the computational power of the platform, the usage of this complex networks could have been counterproductive. In order to achieve an adequate autonomous driving response, it is desired to obtain a segmentation output as fast as possible, however deeper networks could produce greater output latencies, directly affecting the autonomous performance achieved. Nonetheless, the implemented network presented a seemingly good performance with a validation accuracy of 98.51%. Furthermore, it is recommended for future complementary works to count with more diversified training data, utilizing other data augmentation mechanisms such as image translation or rotations, or gathering more testing samples at varying conditions, as an attempt to improve the network segmentation performance.

Finally, an adequate road lane identification and steering angle correction mechanism was achieved based entirely on the CV processing of the segmentation output. It is important to acknowledge that this processing steps are the same for both the CV and DNN based segmentation outputs, allowing an unbiased comparison of the autonomous road lane following performance with respect to the segmentation strategy used. Moreover, a proportional control of the steering angle was tested as a first approach to solve the task, demonstrating a good overall response without the necessity of implementing more complex control strategies.

Chapter 5

Autonomous Driving Response

5.1 Experiment Description

The implemented autonomous driving system was tested using both CV and DNN based road lane segmentation methodologies. The goal was to compare their overall performance, and identify the most suitable proposal for the developed AV platform. To achieve this, multiple autonomous driving experiments were conducted executing the two mentioned systems at different conditions, leaving the vehicle to traverse the traced track inside the ITESM CEM laboratory (see Figure 4.3) autonomously.

Both driving systems were validated using the same testing conditions as an strategy to obtain an unbiased comparison of their performance. The testing scenarios were defined by the modification of the vehicle's driving sense and environment illumination variations. The vehicle's response at this variant conditions would be analyzed to identify the performance achieved by each driving algorithm. However, it was first required to establish an experimental procedure that allowed the quantitative comparison of both system proposals.

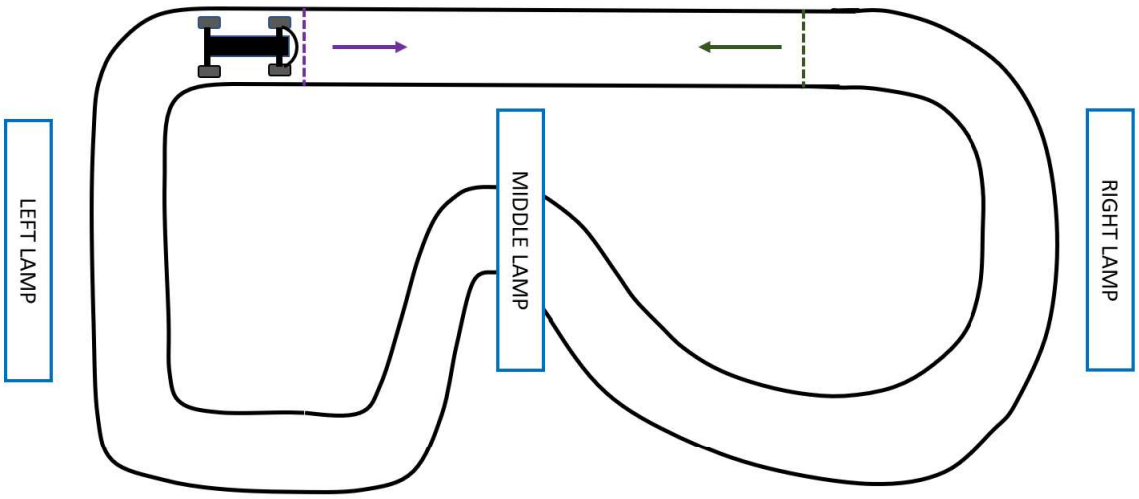


Figure 5.1: Road top-down view representation.

The road that the vehicle would traverse counted with three main light sources in the laboratory lamps distributed along the whole track, see Figure 5.1. The laboratory lamps are placed at the right, middle, and left sections of the traced road, allowing to set the desired variant illumination scenarios by modifying their state. The illumination scenarios tested included (a) All lights ON, (b) Middle and Right Lights ON, (c) Left and Right Lights ON, (d) Only Left Light ON, (e) Only Middle Light ON, (f) Only Right Light ON, and (g) All lights OFF.

It was primordial to establish similar testing conditions for each of the developed experiments. An adequate control of the illumination conditions was achieved, however it was required to define a constant startup condition given the trajectory sense to be used. To achieve this, specific startup locations were determined as shown in Figure 5.1. The clockwise direction startup location is represented with a purple dashed line, while the counter-clock wise startup location is shown in green. This startup conditions were defined to fulfill the initial road lane identification condition where the vehicle must be initially placed in a straight section of the road (see Road Lanes Identification and Classification subsection).

Figure 5.2: Examples of undesired driving condition frames obtained from the mobile camera perspective.

The autonomous driving performance metric was based on how frequently the vehicle presented an undesired driving behaviour throughout each of the developed experiments. Furthermore, an unbiased measurement was desired where the performance metric was not dependent in any way to the autonomous driving system used. To achieve this, external top-down view recordings of each of the vehicle trajectories were made using a mobile camera. This recordings were later analyzed by segmenting them frame by frame to manually determine the performance achieved.

The mentioned methodology allowed defining the desired quantitative performance metric shown in Equation 5.1, based on the total number of frames acquired for a whole vehicle trajectory and the number of frames where the vehicle demonstrated an undesired driving condition. Such error frames included situations where the vehicle was driving near the road limits or directly outside the road, see Figure 5.2. The goal was to identify how frequently this erroneous driving conditions occurred during the vehicle's autonomous driving given the settled illumination condition and the segmentation method used.

$$Error\% = \frac{OutLaneFrames}{TotalFrames} * 100 \tag{5.1}$$

Where:

$Error\%$ = Erroneous driving performance metric (%).
$OutLaneFrames$ = Number of error frames in the current analyzed vehicle trajectory.
$TotalFrames$ = Number of frames for the current analyzed vehicle trajectory.

Finally, the experimental procedure was divided in specific steps following the previously defined specifications:

1. Manually place the vehicle in its startup location.
2. Set the current experiment illumination condition by turning ON or OFF the laboratory lights.
3. Execute the autonomous driving system to be tested (CV or DNN segmentation).
4. Start the mobile recording and change the vehicle's operation mode to autonomous driving.
5. Leave the vehicle autonomously traverse the whole track five times and follow it while recording its driving behavior.
6. Stop the mobile recording once the vehicle has completed the five laps or if it completely leaves the road.
7. Repeat the procedure at the same illumination condition but changing the vehicle driving sense.
8. Determine the achieved driving performance metric using the two trajectory videos recorded.
9. Repeat the whole process with the segmentation system that has not been tested yet.

The previous steps were repeated for each illumination and trajectory sense conditions acquiring a total number of 28 trajectory videos which were manually analyzed to calculate the desired performance metric. Every undesired driving behaviour frame was counted, writing down the total number of such error frames as well as the total number of frames acquired for each video. Finally the performance data obtained for both trajectory senses at similar illumination conditions were added to obtain a single performance evaluation metric for each illumination condition tested.

5.2 Experiment Results

The overall experiments results can be observed in Table 5.1. Two entries are presented for each of the illumination conditions tested, one for the experiments were a CV based road lanes segmentation was used and one for DNN segmentation, i.e. *All lights OFF CV* and *All lights OFF DNN*. Moreover, the Table presents a *Complete out-road* indicator that marks if the vehicle did not complete the whole track, meaning that at some point of the trajectory the vehicle completely left the road. The *Total Frames*, *Out Lane Frames* and *Error%* performance measurement parameters are also presented in the table.

Table 5.1: Autonomous Driving Test Results.

Test Condition	Complete out-road	Total Frames	Out Lane Frames	Error %
All lights OFF CV	X	126	29	44.94%
All lights OFF DNN	X	106	28	52.75%
Only Left Light ON CV	X	1417	246	36.62%
Only Left Light ON DNN	X	3547	403	22.87%
Only Middle Light ON CV		9394	618	13.12%
Only Middle Light ON DNN		9491	763	16.07%
Only Right Light ON CV	X	2612	638	47.38%
Only Right Light ON DNN	X	8442	797	18.83%
Middle and Right Lights ON CV		10840	676	12.55%
Middle and Right Lights ON DNN		10891	679	12.46%
Left and Right Lights ON CV		10108	723	14.22%
Left and Right Lights ON DNN		10358	916	17.69%
All lights ON CV		9749	480	9.87%
All lights ON DNN		9768	700	14.33%

For high illumination scenarios such as *All lights ON* and *Left and Right Lights ON*, the CV based system showed up a seemingly better performance. This is reflected in the *Error %* parameter, that showed that the vehicle presented an undesired driving behavior less frequently during its whole trajectory in comparison to the DNN proposal. However, in the case of the *Middle and Right Lights ON* condition, both systems presented a very similar performance with practically the same *Error %* value. Nonetheless, a good overall performance was achieved by both systems at these high illumination conditions, accomplishing the desired autonomous driving behaviour continuously.

Surprisingly, the vehicle achieved a better driving performance with the DNN based road lane segmentation at low light conditions, even though both proposals were not capable of completing the five road laps. This is clear for the *Only Left Light ON* and *Only Right Light ON* scenarios. This is reflected in the total number of frames acquired for the whole vehicle trajectory, being greater for the DNN proposal than for the CV based, as well as the smaller *Error %* value. However the response at the *Only Middle Light ON* was not comparable to the other low illumination scenarios, as both systems were capable of maintaining a continuous autonomous driving behaviour. This could be caused by the middle lamp position on the road, as it is placed right at the center of the road, allowing a better overall lighting of the whole track compared to the other low illumination scenarios.

Finally, both systems demonstrated a similar response for the *All lights OFF* test scenarios as they were unable to complete the trajectory and directly leaving the road at startup. This could be a reflex of the previously mentioned CV threshold values variation at low light conditions, described in the Road Lane Segmentation using Computater Vision, as well as a bad response from the DNN segmentation system.

The obtained results could reflect that the CV segmentation was better adapted to high illumination scenarios whereas the DNN segmentation system was better for lower ones besides, they presented a comparable performance for medium light conditions. Video examples of the platform's response at the *All Lights ON* illumination condition can be observed in links shown at Figures 5.3 and 5.4. Both of them show the mobile camera recording captured that was later analyzed manually and the inside captured processing results throughout the whole trajectory.

(a) Inside process recording.

(b) Outside mobile recording.

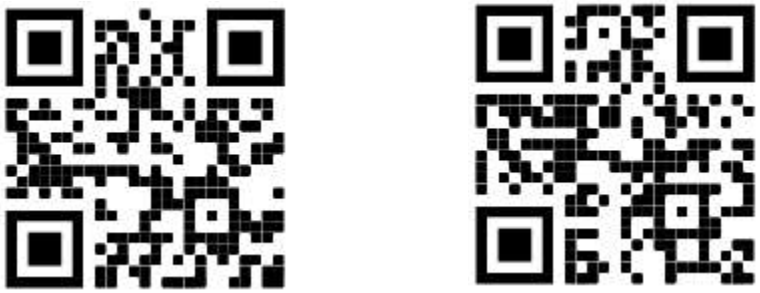
Figure 5.3: Autonomous driving response mobile recording and inner image processing pipeline Examples at All Lights ON illumination condition using a DNN based segmentation system.

(a) Inside process recording.

(b) Outside mobile recording.

Figure 5.4: Autonomous driving response mobile recording and inner image processing pipeline Examples at All Lights ON illumination condition using a CV based segmentation system.

Other autonomous driving response examples can be observed in links shown at Figures 5.5 and 5.6. These experiments were made at a *Only Left Light ON* illumination condition. Throughout the whole trajectory, it is clear that the DNN based segmentation system offered a better response as CV segmentation showed really noisy outputs (see minute 1:04 and 0:00 of Figure's 5.6 Inside process recording). Furthermore, the vehicle was capable of completing four complete road trajectories using the DNN system while the CV proposal only completed a single road lap, see Figure 5.11. Moreover, the DNN proposal got into a complete out-road situation however the inside recording did not show evidence of a bad segmentation response (see minute 1:20 of 5.5 Inside process recording).



(a) Inside process recording. (b) Outside mobile recording.

Figure 5.5: Autonomous driving response mobile recording and inner image processing pipeline Examples at Only Left Light ON illumination condition using a DNN based segmentation system.

(a) Inside process recording. (b) Outside mobile recording.

Figure 5.6: Autonomous driving response mobile recording and inner image processing pipeline Examples at Only Left Light ON illumination condition using a CV based segmentation system.

Figures 5.7 and 5.8 show links to experiment recordings at *Only Right Light ON* illumination conditions. In these cases, both autonomous driving systems tested were not capable of completing the whole set of road trajectories (see Figure 5.12) however the DNN system completed a single trajectory successfully, while the CV proposal got into the undesired complete out-road condition before even completing the first lap. It's noticeable that the vehicle completely left the road at the same track section which was the left road curve (see minute 1:06 of Figure's 5.8 Inside process recording and minute 0:47 of Figure's 5.7 Inside process recording). This could be a result of an erroneous segmentation response by both systems product of the settled illumination condition, given that such road section was the darkest being the furthest from the unique light source.

(a) Inside process recording.

(b) Outside mobile recording.

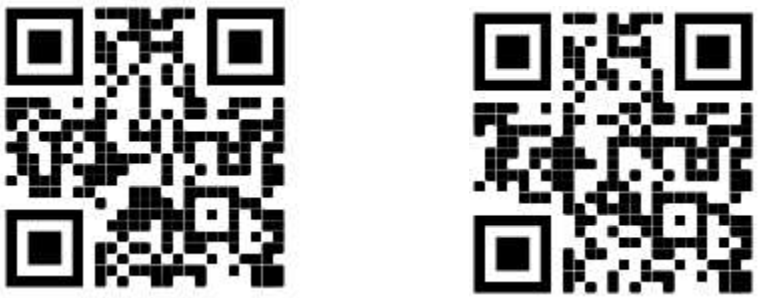
Figure 5.7: Autonomous driving response mobile recording and inner image processing pipeline Examples at Only Right Light ON illumination condition using a DNN based segmentation system.

(a) Inside process recording.

(b) Outside mobile recording.

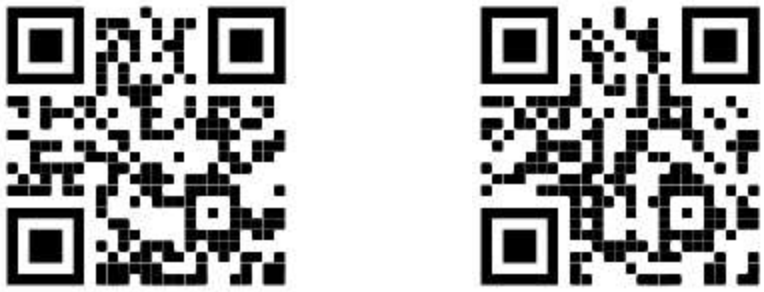
Figure 5.8: Autonomous driving response mobile recording and inner image processing pipeline Examples at Only Right Light ON illumination condition using a CV based segmentation system.

Finally, the extreme case scenarios are shown in Figures 5.9 and 5.10 including some recordings of the vehicle's performance at the *All Lights OFF* illumination condition. As shown in both proposals' Inside process recording video, the segmentation systems were incapable of segmenting the road lanes in the acquired frames, causing the vehicle to completely leave the traced road at startup (see Figure 5.13. This reflects that neither autonomous driving system proposal presented an adequate response at considerably dark scenarios as the one tested in these recordings.



(a) Inside process recording. (b) Outside mobile recording.

Figure 5.9: Autonomous driving response mobile recording and inner image processing pipeline Examples at All Lights OFF illumination condition using a DNN based segmentation system.



(a) Inside process recording. (b) Outside mobile recording.

Figure 5.10: Autonomous driving response mobile recording and inner image processing pipeline Examples at All Lights OFF illumination condition using a CV based segmentation system.

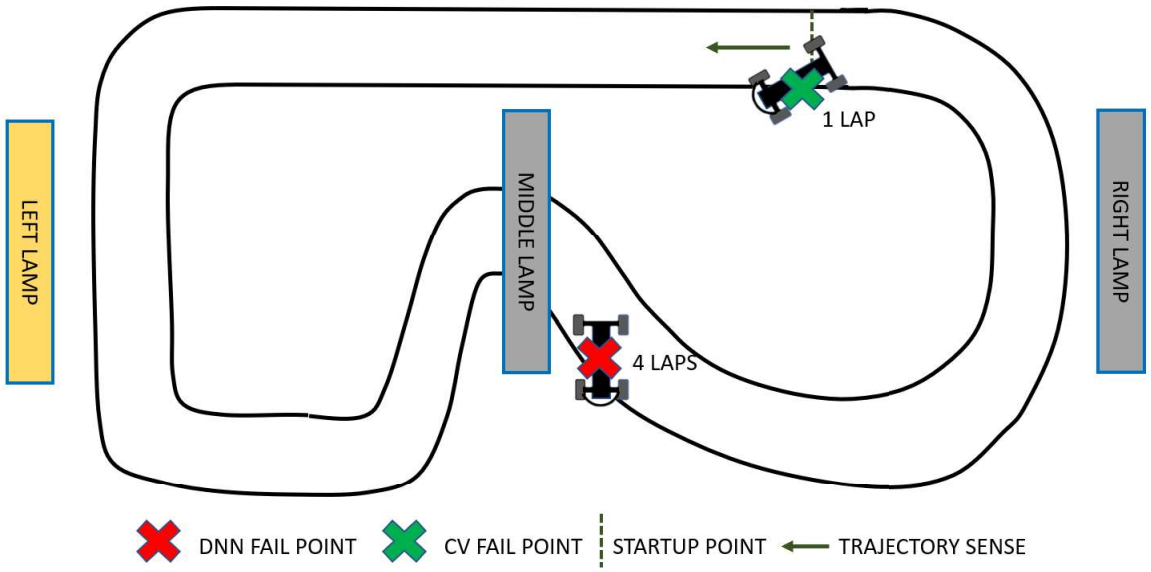


Figure 5.11: Vehicle's autonomous driving response representation using both CV and DNN system proposals at the Only Left Light ON illumination condition.

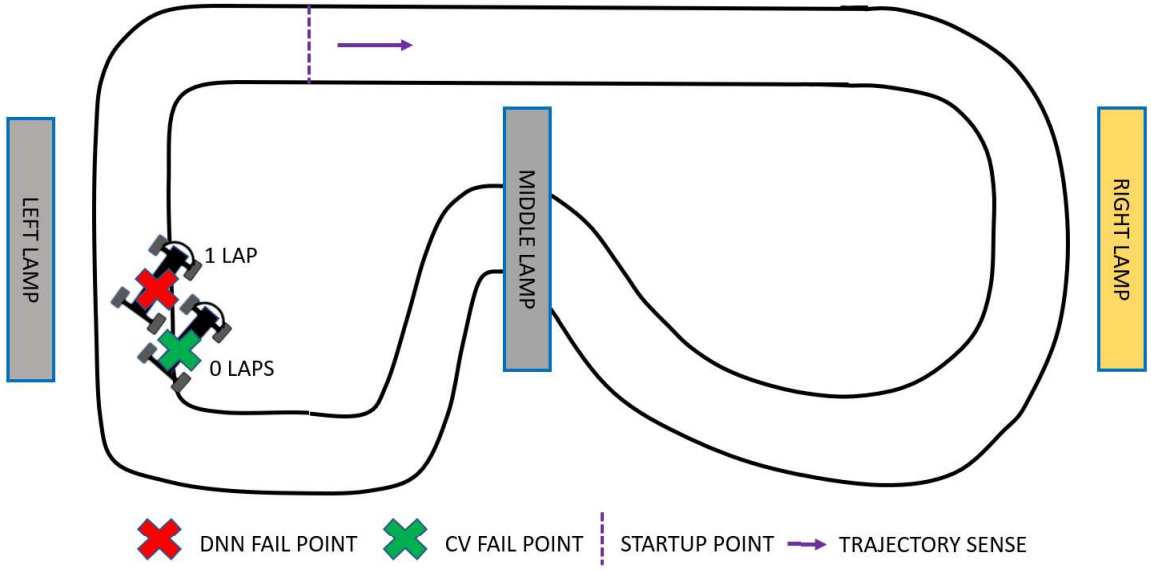


Figure 5.12: Vehicle's autonomous driving response representation using both CV and DNN system proposals at the Only Right Light ON illumination condition.

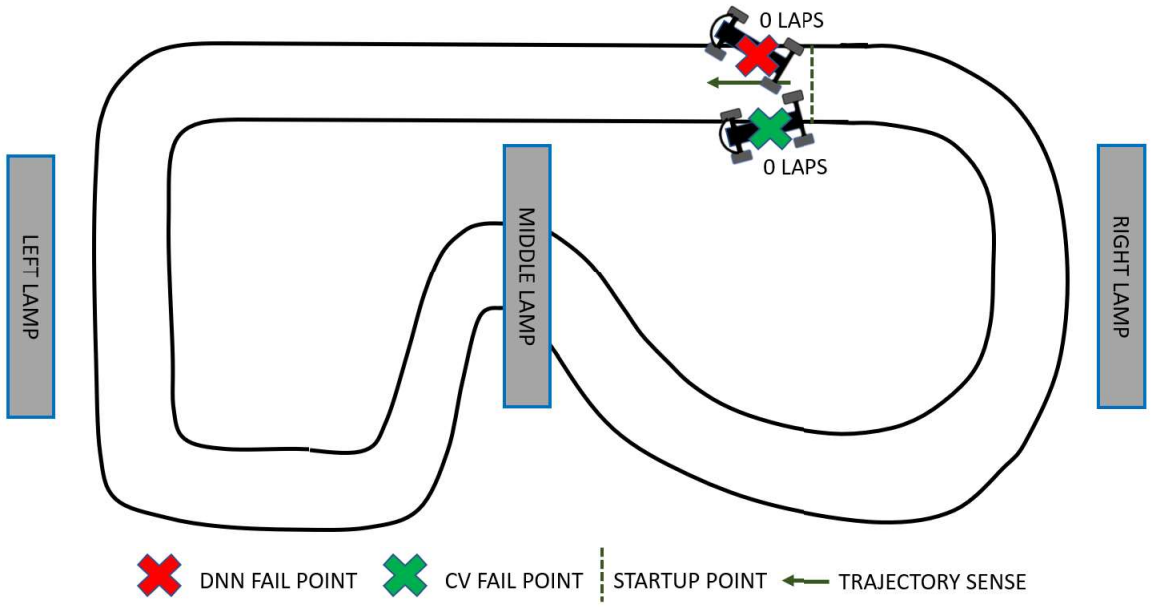


Figure 5.13: Vehicle's autonomous driving response representation using both CV and DNN system proposals at the All Lights OFF illumination condition.

5.3 Discussion

In this chapter, the developed autonomous driving systems (CV and DNN based) were tested and evaluated. An specific experimental methodology was defined to determine the autonomous driving performance achieved by each proposal and obtain an unbiased quantitative comparison. Even though the selected strategy offered a simple quantitative evaluation method, the performance estimation could have been enriched with other complementary measurements such as an image processing response evaluation. This would allow to determine erroneous road lane identifications and consider the achieved segmentation quality as part of the performance evaluation measurement.

The implemented autonomous driving strategies present a high performance dependency on the established illumination condition. Despite the good performance achieved by both systems at high illumination scenarios, there was clear evidence of a decrease in the segmentation quality as darker testing scenarios were settled. Furthermore, it was not possible to recognize a single best autonomous driving system as both proposals demonstrated pretty similar performances. This was demonstrated thanks to the calculated *Error %*, where DNN segmentation seemed the better option for low illumination conditions and CV for higher ones, while both system showed pretty similar responses at medium light scenarios. The proposal was not capable of achieving better performances even at higher illumination conditions, despite using the CV or DNN segmentation methods. This could imply that better steering correction system adjustments could be made in order to, for instance, prevent erroneous vehicle driving behaviours at curved sections of the road.

Moreover, if its important to acknowledge the segmentation output generation latency of both system proposals, as a more complex image processing is made by the DNN system with a 30 ms output latency in comparison to the CV system with a latency of 0.3 ms. This implies that the DNN segmentation system presents a substantially slower response than the CV approach. Therefore a complete evaluation of this performance trade-off could be taken into consideration for the selection of the autonomous driving system to be used, specially given the unclear performance difference between them. Further research can be done regarding the relation between the vehicle's linear velocity and the image processing output latency in order to improve the autonomous control performance.

It was evident that the general response at low illumination conditions could have been improved by developing a more robust CV or DNN segmentation system and taking into consideration the multiple possible light scenarios the vehicle would face. Besides, the developed U-Net system could have been benefited by the acquisition of a more complete training dataset, gathering training data of multiple different illumination scenarios and at different day times. Nevertheless, the platform autonomous driving capabilities were demonstrated successfully, serving as a baseline for future updates and improvements of the algorithms proposed in this project.

Future complementary work can be considered, including the possibility of implementing more complete odometry mechanisms, taking advantage of the platform's hardware. This mechanisms could include the Intel Realsense image processing and RPLidar sensor data to develop a sensor fusion system, allowing a better odometry and vehicle trajectory tracking system. The proposed autonomous performance evaluation system depends on human intervention for the manual identification of erroneous driving conditions, however further research could be done to propose automated evaluation mechanisms that could considerably enhance the comparison and development of different autonomous driving approaches.

References

[1] Hobbyking trooper pro-edition brushless sct. https://www.rcgroups.com/forums/showthread.php?2756163-Hobbyking-Trooper-Pro-Edition-Brushless-SCT. Accessed: 2020-11-21.

[2] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.

[3] Craig Quiter and M Ernst. Deepdrive. https://deepdrive.voyage.auto/. Accessed: 2020-04-01.

[4] Inc LG Electronics. Lgsvl simulator. https://www.svlsimulator.com/. Accessed: 2020-04-01.

[5] Shital Shah, Debadeepta Dey, Chris Lovett, and Ashish Kapoor. Airsim: High-fidelity visual and physical simulation for autonomous vehicles, 2017.

[6] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. CARLA: An open urban driving simulator. In Sergey Levine, Vincent Vanhoucke, and Ken Goldberg, editors, *Proceedings of the 1st Annual Conference on Robot Learning*, volume 78 of *Proceedings of Machine Learning Research*, pages 1–16. PMLR, 13–15 Nov 2017.

[7] Matthew O'Kelly, Houssam Abbas, Jack Harkins, Chris Kao, Yash Vardhan Pant, Rahul Mangharam, Varundev Suresh Babu, Dipshil Agarwal, Madhur Behl, Paolo Burgio, and Marko Bertogna. F1/10: An open-source autonomous cyber-physical platform. *arXiv*, 10, 2019.

[8] Christian Berger. From a Competition for Self-Driving Miniature Cars to a Standardized Experimental Platform: Concept, Models, Architecture, and Evaluation. 5(May):63–79, 2014.

[9] Ardashir Bulsara, Adhiti Raman, Srivatsav Kamarajugadda, Matthias Schmid, and Venkat N Krovi. Obstacle avoidance using model predictive control: An implementation and validation study using scaled vehicles. Technical report, SAE Technical Paper, 2020.

[10] Matthew O'Kelly, Hongrui Zheng, Dhurv Karthik, and Rahul Mangharam. F1tenth: An open-source evaluation environment for continuous control and reinforcement learning. In Hugo Jair Escalante and Raia Hadsell, editors, *Post Proceedings of the NeurIPS 2019 Demonstration and Competition Track*, Proceedings of Machine Learning Research. PMLR, 2020.

[11] AutoModelCar. Autonomos model. https://github.com/AutoModelCar/AutoModelCarWiki/wiki, 2018.

[12] Ugo Rosolia Jon Gonzales and Greg Marcil. Barc: Berkeley autonomous race car. http://www.barc-project.com/. Accessed: 2020-04-01.

[13] MIT. Racecar. https://mit-racecar.github.io/. Accessed: 2020-04-01.

[14] Bharathan Balaji, Sunil Mallya, Sahika Genc, Saurabh Gupta, Leo Dirac, Vineet Khare, Gourav Roy, Tao Sun, Yunzhe Tao, Brian Townsend, Eddie Calleja, Sunil Muralidhara, and Dhanasekar Karuppasamy. Deepracer: Educational autonomous racing platform for experimentation with sim2real reinforcement learning, 2019.

[15] Brian Goldfain, Paul Drews, Changxi You, Matthew Barulic, Orlin Velev, Panagiotis Tsiotras, and James Rehg. Autorally an open platform for aggressive autonomous driving, 06 2018.

[16] Siddhartha S. Srinivasa, Patrick Lancaster, Johan Michalove, Matt Schmittle, Colin Summers, Matthew Rockett, Joshua R. Smith, Sanjiban Choudhury, Christoforos Mavrogiannis, and Fereshteh Sadeghi. Mushr: A low-cost, open-source robotic racecar for education and research, 2019.

[17] W Roscoe. Donkey car: An opensource diy self driving platform for small scale cars. https://www.amazon.com/XiaoR-Geek-Starter-Platform-Raspberry/dp/B082NL3RLG. Accessed: 2020-04-01.

[18] Adhiti T. Raman, Venkat N. Krovi, and Matthias J.A. Schmid. Empowering graduate engineering students with proficiency in autonomy. *Proceedings of the ASME Design Engineering Technical Conference*, 5A-2018:1–8, 2018.

[19] Matthew O'Kelly, Hongrui Zheng, Achin Jain, Joseph Auckley, Kim Luong, and Rahul Mangharam. TUNERCAR: A Superoptimization Toolchain for Autonomous Racing. *Proceedings - IEEE International Conference on Robotics and Automation*, (January):5356–5362, 2020.

[20] Aman Sinha, Matthew O'Kelly, Hongrui Zheng, Rahul Mangharam, John Duchi, and Russ Tedrake. Formulazero: distributionally robust online adaptation via offline population synthesis. *arXiv*, 2020.

[21] Embedded Systems, Abhijeet Agnihotri, and Matthew O Kelly. ScholarlyCommons Teaching Autonomous Systems at 1 / 10th-scale. (February), 2020.

[22] David Tian. Deeppicar — part 1: How to build a deep learning, self driving robotic car on a shoestring budget. https://towardsdatascience.com/deeppicar-part-1-102e03c83f2c, 2019.

[23] Mariusz Bojarski, Davide Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Larry Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, and Karol Zieba. End to end learning for self-driving cars. 04 2016.

[24] Turnigy trooper sct-x4 1/10 4x4 nitro curso corto de camiones (rtr). https://hobbyking.com/es_es/

turnigy-trooper-sct-x4-1-10-4x4-nitro-short-course-truck-rtr.html. Accessed: 2020-11-21.

[25] Turnigy rc on/off switch. https://hobbyking.com/es_es/turnigy-receiver-controlled-switch-1.html. Accessed: 2020-11-21.

[26] Turnigy 9x 9ch mode 2 transmitter w/ module and ia8 receiver (afhds 2a system). https://hobbyking.com/es_es/turnigy-9x-9ch-mode-2-transmitter-w-module-ia8-receiver-afhds-2a-system.html. Accessed: 2020-11-21.

[27] Arduino nano. https://store.arduino.cc/usa/arduino-nano. Accessed: 2020-11-21.

[28] Odroid-xu4. https://wiki.odroid.com/odroid-xu4/odroid-xu4. Accessed: 2020-11-21.

[29] Jetson nano developer kit. https://developer.nvidia.com/EMBEDDED/jetson-nano-developer-kit. Accessed: 2020-11-21.

[30] Blasterx senz3d. https://es.creative.com/p/web-cameras/blasterx-senz3d. Accessed: 2020-11-21.

[31] Rplidar a2. https://www.slamtec.com/en/Lidar/A2. Accessed: 2020-11-21.

[32] Turnigy 2200mah 2s 25c lipo pack w/xt60. https://hobbyking.com/es_es/turnigy-2200mah-2s-30c-lipo-pack.html?queryID=61017de534bd628a334581b6a7d83735&objectID=18290&indexName=hbk_live_magento_es_es_products. Accessed: 2020-11-21.

[33] Matek pdb-xt60 (power distribution board). http://www.mateksys.com/?portfolio=pdb-xt60. Accessed: 2020-11-21.

[34] Pololu 5v, 9a step-down voltage regulator d24v90f). https://www.pololu.com/product/2866. Accessed: 2020-11-21.

[35] Ros melodic morenia. http://wiki.ros.org/melodic. Accessed: 2020-11-21.

[36] jetsonhacks. installlibrealsense. https://github.com/jetsonhacks/installLibrealsense-1/tree/vL4T32.2.1, 2019.

[37] WubinXia. Slamtec rplidar public sdk. https://github.com/Slamtec/rplidar_sdk, 2020.

[38] shiritbrook. librealsense. https://github.com/IntelRealSense/librealsense, 2021.

[39] AldoAguilar. Self driving car project. https://github.com/AldoAguilar/self_driving_car_project/tree/master, 2020.

[40] Arduino cli (command line interface) application. https://www.arduino.cc/pro/cli. Accessed: 2020-11-21.

[41] Tensorflow. https://www.tensorflow.org/. Accessed: 2020-11-21.

[42] Opencv. https://www.python.org/. Accessed: 2020-11-21.

[43] jleichty. R/c car quadrature wheel encoders. http://www.blargh.co/2013/, 2013.

[44] Dimitris Platis. Build your own android-powered self driving r/c car. https://makezine.com/projects/build-android-powered-autonomous-rc-car/, 2015.

[45] Connect two networks with one roscore. https://answers.ros.org/question/256435/connect-two-networks-with-one-roscore/. Accessed: 2020-11-21.

[46] Running ros across multiple machines. http://wiki.ros.org/ROS/Tutorials/MultipleMachines. Accessed: 2020-11-21.

[47] Understanding ros nodes. http://wiki.ros.org/ROS/Tutorials/UnderstandingNodes. Accessed: 2020-11-21.

[48] Understanding ros topics. http://wiki.ros.org/ROS/Tutorials/UnderstandingTopics. Accessed: 2020-11-21.

[49] Diego Bueno. Motor eléctrico brushless: Funcionamiento y características. https://1mecanizadoelarenal.files.wordpress.com/2013/11/motores-brushless.pdf, 2013.

[50] Howard Austerlitz. Chapter 2 - analog signal transducers. In Howard Austerlitz, editor, *Data Acquisition Techniques Using PCs (Second Edition)*, pages 6 – 28. Academic Press, San Diego, second edition edition, 2003.

[51] Clarence W. de Silva. Sensors for control. In Robert A. Meyers, editor, *Encyclopedia of Physical Science and Technology (Third Edition)*, pages 609 – 650. Academic Press, New York, third edition edition, 2003.

[52] Arduino external interrupts. https://www.arduino.cc/reference/en/language/functions/external-interrupts/attachinterrupt/. Accessed: 2020-11-21.

[53] Arduino pwm. https://www.arduino.cc/en/Tutorial/Foundations/PWM. Accessed: 2020-11-21.

[54] Arduino analog library. https://www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/. Accessed: 2020-11-21.

[55] Atmel. *ATmega328P 8-bit AVR Microcontroller with 32K Bytes In-System Programmable Flash DATASHEET*, 1 2015. Rev. 4.

[56] Gene Franklin, J.D. Powell, and M.L. Workman. Digital control of dynamic systems-third edition. page 503, 01 2006.

[57] S. Dormido. Advanced pid control - [book review]. *Control Systems Magazine, IEEE*, 26:98– 101, 03 2006.

[58] Xianhu Gao. Bldc motor control with hall sensors based on frdm-ke 02 z by :. 2013.

[59] Vikas Gupta and Anindya Deb. Speed control of brushed dc motor for low cost electric cars. In *2012 IEEE International Electric Vehicle Conference*, pages 1–3, 2012.

[60] Em Poh Ping, J. Hossen, Fitrian Imaduddin, Wong Eng Kiong, and Ubaidillah Sabino. Experimental of vision-based lane markings segmentation methods in lane detection application. *Journal of Engineering Science and Technology Review*, 2019.

[61] Miguel Angel Sotelo, Francisco Javier Rodriguez, Luis Magdalena, Luis Miguel Bergasa, and Luciano Boquete. A color vision-based lane tracking system for autonomous driving on unmarked roads. *Autonomous Robots*, 2004.

[62] Dwi Prasetyo Adi Nugroho and Mardhani Riasetiawan. Road lane segmentation using deconvolutional neural network. In *Communications in Computer and Information Science*, 2017.

[63] Leonardo Cabrera Lo Bianco, Jorge Beltrán, Gerardo Fernández López, Fernando García, and Abdulla Al-Kaff. Joint semantic segmentation of road objects and lanes using Convolutional Neural Networks. *Robotics and Autonomous Systems*, 2020.

[64] Jelena Kocic, Nenad Jovicic, and Vujo Drndarevic. An end-to-end deep neural network for autonomous driving designed for embedded automotive platforms. *Sensors*, 05 2019.

[65] N. Kehtarnavaz and W. Sohn. Steering control of autonomous vehicles by neural networks. In *1991 American Control Conference*, pages 3096–3101, 1991.

[66] Wael Farag and Zakaria Saleh. Behavior cloning for autonomous driving using convolutional neural networks. In *2018 International Conference on Innovation and Intelligence for Informatics, Computing, and Technologies (3ICT)*, pages 1–7, 2018.

[67] P Daniel Ratna Raju and G Neelima. Image segmentation by using histogram thresholding. *International Journal of Computer Science Engineering and Technology*, 2(1):776–779, 2012.

[68] J. R. Parker. Gray-level segmentation. In *Algorithms for Image Processing and Computer Vision*, chapter 4, pages 141–142. Wiley Publishing, Inc., 2010.

[69] Adrian Kaehler Gary Bradski. Image transforms. In *Learning OpenCV: Computer Vision with the OpenCV Library*, chapter 6, pages 163–172. O'Reilly, 2008.

[70] Le Anh Tran and My Ha Le. Robust u-net-based road lane markings detection for autonomous driving. In *Proceedings of 2019 International Conference on System Science and Engineering, ICSSE 2019*, 2019.

[71] Farhana Sultana, Abu Sufian, and Paramartha Dutta. Evolution of image segmentation using deep convolutional neural network: A survey. *Knowledge-Based Systems*, 201-202:106062, Aug 2020.

[72] Opencv perspective transformation. `https://medium.com/analytics-vidhya/opencv-perspective-transformation-9edffefb2143#:~:text=Perspective%20Transform%20is%20a%20feature,Transformation%20is%20applied%20to%20it.` Accessed: 2020-04-01.

[73] Ge Pingshu, Guo Lie, Qi Guodong, and Chang Jing. Lane marker line identification method in variable light environment. *Laser and Optoelectronics Progress*, 2020.

[74] Deepshikhar Tyagi, Sameer Farkade, and Upendra Suddamalla. Night time road boundary detection using adaptive averaging likelihood map over spatio-temporal gradient correspondence-STGC. In *2017 4th International Conference on Image Information Processing, ICIIP 2017*, 2018.

[75] Farid Garcia-Lamont, Jair Cervantes, Asdrúbal López, and Lisbeth Rodriguez. Segmentation of images by color features: A survey. *Neurocomputing*, 2018.

[76] Parthima Guruprasad. Overview Of Different Thresholding Methods In Image Processing. *TEQIP Sponsored 3rd National Conference on ETACC*, 2020.

[77] Color conversions. `https://docs.opencv.org/3.4/de/d25/imgproc_color_conversions.html`. Accessed: 2020-04-01.

[78] Opencv contours : Getting started. `https://docs.opencv.org/3.4/d4/d73/tutorial_py_contours_begin.html`. Accessed: 2020-11-21.

[79] Niall O' Mahony. Deep Learningvs. Traditional Computer Vision. *MaR Technology Gateway, Institute of Technology Tralee*, 2020.

[80] Hyeon-Joong Yoo. Deep convolution neural networks in computer vision: a review. *IEIE Transactions on Smart Processing and Computing*, 4:35–43, 02 2015.

[81] Amaury Bréhéret. Pixel Annotation Tool. `https://github.com/abreheret/PixelAnnotationTool`, 2017.

[82] Image segmentation with watershed algorithm. `https://docs.opencv.org/master/d3/db4/tutorial_py_watershed.html`. Accessed: 2020-04-01.

[83] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications, 2017.

[84] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[85] A basic introduction to separable convolutions. `https://towardsdatascience.com/a-basic-introduction-to-separable-convolutions-b99ec3102728`. Accessed: 2020-04-01.

[86] Abdelilah Adiba, Hicham Hajji, and Mustapha Maatouk. Transfer learning and U-Net for buildings segmentation. In *ACM International Conference Proceeding Series*, 2019.

[87] tf.keras.applications.mobilenetv2. `https://www.tensorflow.org/api_docs/python/tf/keras/applications/MobileNetV2`. Accessed: 2020-04-01.

[88] Contour features. `https://docs.opencv.org/3.4/dd/d49/tutorial_py_contour_features.html`. Accessed: 2020-04-01.