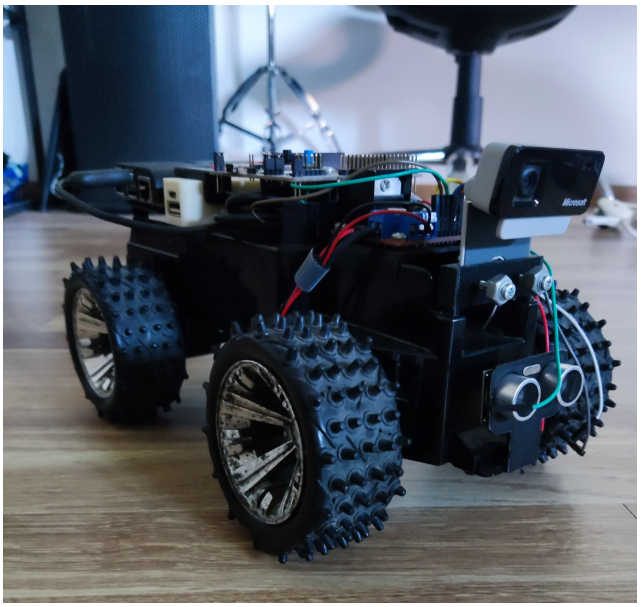
INTERNATIONAL HELLENIC UNIVERSITY

ENGINEERING SCHOOL

DEPARTMENT OF INFORMATION AND ELECTRONICS SYSTEMS ENGINEERING

THESIS

«ADVANCED DRIVER-ASSISTANCE SYSTEM»



Student:
Karageorgiadis Antonis
Registration number: ele516046

Supervisor
Giakoumis Aggelos
Rank : Lecturer

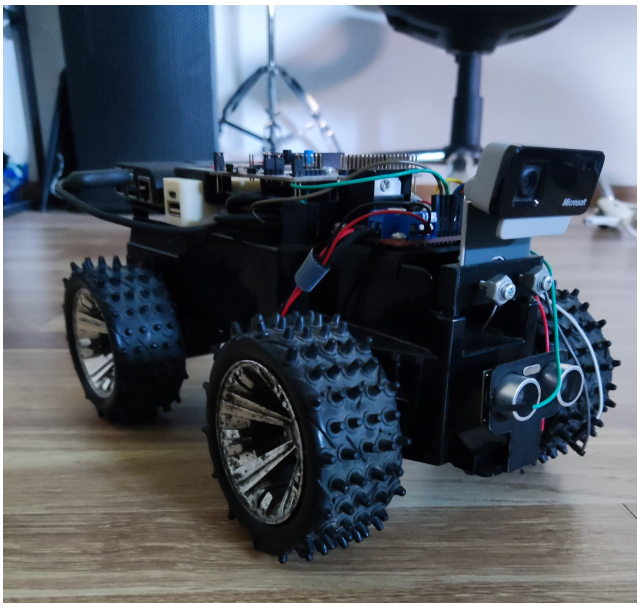
September 10, 2022

ΔΙΕΘΝΕΣ ΠΑΝΕΠΙΣΤΗΜΙΟ ΤΗΣ ΕΛΛΑΔΟΣ

ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ
«ΟΔΗΓΗΣΗ ΥΠΟΒΟΗΘΟΥΜΕΝΗ ΑΠΟ Η/Υ»



Του φοιτητή:
Καραγεωργιάδη Αντώνιου
Αρ. Μητρώου: ele516046

Επιβλέπων
Γιακουμής Άγγελος
Βαθμίδα:Λέκτορας

September 10, 2022

Thesis title Advanced Driver-Assistance System
Thesis Number. 21187
Student Name Karageorgiadis Antonis
Supervisor Name Giakoumis Aggelos
Thesis date of takeover 15-03-2021
Thesis date of completion 04-09-2022

I certify that I am the author of this paper and that any assistance I had in its preparation is fully acknowledged and referenced in the paper. I have also listed all sources from which I have used data, ideas, images, and text, whether quoted verbatim or paraphrased. In addition, I certify that this thesis was prepared by me, specifically as a thesis, at the Department of Information and electronics systems engineering IHU.

The approval of the thesis by the Department of Information and Electronics Systems Engineering of the International Hellenic University does not necessarily imply the acceptance of the author's views by the Department.

Τίτλος Π.Ε. Οδήγηση Υποβοθούμενη από Η/Υ
Κωδικός Π.Ε. 21187
Ονοματεπώνυμο φοιτητή Καραγεωργιάδης Αντώνιος
Ονοματεπώνυμο εισηγητή Γιακουμής Άγγελος
Ημερομηνία ανάληψης Π.Ε. 15-03-2021
Ημερομηνία περάτωσης Π.Ε. 4-09-2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως διπλωματική εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Η έγκριση της διπλωματικής εργασίας από το Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος, δεν υποδηλώνει απαραιτήτως και αποδοχή των απόψεων του συγγραφέα, εκ μέρους του Τμήματος.

«This paper is dedicated to my family, who supported me throughout my student career.»

Foreword

Nowadays, engineers push themselves to pursue clear paths. Whether we are electronics or electrical or mechanical engineers, we all prefer to move on a straightforward route and not get implicated in different pathways. We all choose our engineering path soon enough. In my case, I enjoyed electronics from an early age, but my true passion was mechanical engineering. Joining electronics school opened a new area on the map of my future engineering career. The study period connected me with several fields; I admired Analog circuits, had fun with Digital, and became fascinated with Power Electronics, Telecommunications, and Embedded Systems. When I had to choose a subject for my thesis, I was trying to find something that could combine electronics, mechanics, and some kind of Software. So the topic of this thesis Advanced-Driver-Assistant System, allows me to struggle with many areas of engineering. It was a great challenge with essential effects.

Περίληψη

Η συγκεκριμένη Π.Ε έχει ως θέμα « Οδήγηση υποβοηθούμενη από Η/Υ» ή αλλιώς « Advanced - Driving-Assisted-System - ADAS» στη αγγλική διάλεκτο. Η βασική ιδέα της Π.Ε είναι η κατασκευή ενός πρωτότυπού -μοντέλου οχήματος, το οποίο θα είναι θέση να βοηθήσει τον οδηγό ώστε να συμμορφώνεται σε σχέση με τον Κώδικα οδικής κυκλοφορίας (Κ.Ο.Κ.). Πιο συγκεκριμένα, το όχημα με την χρήση ηλεκτρονικού υπολογιστή θα παίρνει αποφάσεις συμφωνά της πινακίδες οδικής κυκλοφορίας, για παράδειγμα αν το όχημα συναντήσει μία σήμανση ΣΤΟΠ θα πρέπει το όχημα να τηρήσει τον κώδικά. Οι βασικοί πυλώνες της εργασίας είναι τρεις : ο εξοπλισμός (Hardware), το υλικολογισμικό (Firmware) και το λογισμικό (Software). Ο πρώτος πυλώνας αφορά την κατασκευή του οχήματος με μηχανικά μέρη και ηλεκτρονικά μέρη ,και το κομμάτι συναρμολόγησης με ότι αυτό συνεπάγεται. Ο δεύτερος πυλώνας αφορά το κομμάτι ελέγχου των ηλεκτρονικών κυκλωμάτων του εξοπλισμού(Hardware), δηλαδή την υλοποίηση ενός κώδικα , ο οποίος θα χρησιμοποιείται από τον μικροελεγκτή με σκοπό να εκτελεί συγκεκριμένες λειτουργίες (σταμάτα τους κινητήρες , επιτάχυνε, στρίψε,…κτλ.). Ο τρίτος πυλώνας , αφορά το κομμάτι της δημιουργίας ενός προγράμματος, το οποίο θα είναι υπεύθυνο για την λήψη αποφάσεων ,βασισμένων στον Κώδικα Οδικής Κυκλοφορίας Κ.Ο.Κ. Άλλα και για την εκτελέσει τους, καθώς θα αποτελεί το λογισμικό κομμάτι για την κυριά μονάδα ελέγχου του οχήματος δηλαδή του Η/Υ. Τα αποτελέσματα αυτής της Π.Ε χωρίζονται σε δυο κατηγορίες η μία έχει να κάνει με τα προβλήματα που αντιμετωπίζει το μοντέλο μου , δηλαδή δεν περιέχει ολόκληρο των Κ.Ο.Κ , δεν έχει τον απαραίτητο χρόνο για αντίδραση ,το όχημα δεν είναι αυτόνομο και δεν έχει υλοποιηθεί το κομμάτι ασφάλειας (Security).Η άλλη κατηγορία αφορά στο τι θα μπορούσε να προσφέρει στην κοινωνία ένα πλήρες λειτουργικό σύστημα, δηλαδή μείωση των ατυχημάτων, ελαχιστοποίηση του ανθρώπινου λάθους, εκπαίδευση του οδηγού, ποια είναι τα κομμάτια του παζλ που πρέπει να συμπληρωθούν για να κατασκευαστή ένα τέτοιο σύστημα ώστε να πληροί όλες τις νομικές και ηθικές υποχρεώσεις.

Abstract

The topic of this thesis is «Advanced - Driver-Assisted-System - ADAS». The basic concept is the construction of a prototype-model vehicle, which will be capable of assisting the driver in complying with the Traffic Code . The car will operate a computer to make decisions according to the highway signs. For instance, if the vehicle meets a STOP sign, it must comply with the traffic principle. The thesis's primary pillars are Hardware, Firmware, and Software. The first pillar involves the vehicle's structure with mechanical and electronic components, including the assembly procedure. The second pillar concerns the controlling role of the electronic circuits (Hardware), i.e., the implementation of a code, which the microcontroller will utilize to execute precise operations (control the engines, accelerate, shift). The third pillar involves the aspect of developing a program, which will be accountable for making decisions following the traffic code. The vehicle's central command unit (PC) software will also be responsible for the performance. There are two classifications of outcomes in this thesis. The first one has to do with the issues that our model faces, i.e., it does not include the whole of the traffic code, it does not have the vital time for response, the vehicle is not autonomous, and there are plenty of security concerns. The other classification is regarding what a complete functional system could contribute to the community, i.e., decreasing accidents, devaluation of human mistakes, driver education, and the missing pieces of the puzzle to build a complete system.

Acknowledgements

First, I want to thank my family, who supported me financially and emotionally, for completing this work. Nevertheless, my supervisor, Professor Giakoumis Angelos, also offered me the possibility to communicate with him anytime.

Contents

List of figures

List of tables

Συντομογραφιές

Π.Ε.	Πτυχιακή Εργασία
ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
Κ.Ο.Κ	Κώδικας Οδικής Κυκλοφορίας

Acronyms

A.D.A.S	Advanced Driving Assisted System
A.N.N	Artificial Neural Network
C.N.N	Convolutional Neural Network
Re.L.U	Rectify Linear Unit
M.S.E	Mean Squared Error
R.T.O.S	Real Time Operating System
G.P.O.S	General Purpose Operating System
I.S.P	Interrupt Service Routine
U.A.R.T	Universal Asynchronous Receiver Transmitter
D.M.A	Direct Memory Access
P.W.M	Pulse Width Modulation
B.M.S	Battery Management System
S.S.H	Secure Shell
S.T	STMicroelectronics
R.C.C	Reset and Clock Control
H.S.E	Hight Speed Clock
R.O.S	Robotic Opearting System
C.R.C	Cyclic Redundancy Check
C.I	Continuous Integration
C.D	Continuous Development

1 Introduction

1.1 Introduction

In times, we are facing many social and political problems. We all know about the economic crisis effects of COVID-19 and war; we all know about the energy crisis, our intention to move to new alternative power sources, and other problems too big to discuss in this paper. Population reduction [1] is a problem we may not know or it is not immediately visible. So what is population reduction? Many countries, including Greece, are facing around a 10% decrease in the population by 2050. We can assume of any reason for this situation that our people are getting old; and mainly because a large number of young people do not even give birth, we have had some severe illness in the last few years. One cause that should concern us is the deaths from road accidents. On 28^{th} March 2022 European Commission published preliminary figures on road fatalities for 2021 [2]. Car accidents killed almost 19800 people.
When Carl Benz made the first automobile in the world [3], a one-cylinder, two-stroke unit that ran for the first time on New Year's Eve, 1879. People did not care about safety rules in that period because automobiles were few. During the decades of development of automobility, we may reach those fabulous cars with top speeds and superb aerodynamics, but we are still losing many people in car accidents. Today's technology can improve our cars' safety. Nevertheless, we have a significant problem our drivers make mistakes that cost lives. In 2016 we had in the EU almost 25 699 deaths from accidents on traffic roads; this number is decreasing by decade because safety rules are more and more. Big automotive companies are working on new safety protocols, cameras for obstacle avoidance or emergency braking, speed limit warnings on the main display or vibration on the steering wheel in case of line changing, and new materials at the body of the car,that can protect or at least not harm passengers in an accident. So the problem we are dealing with is improving safety on the road and minimizing human mistakes.

1.2 Thesis Contribution

In this thesis, we develop a method to recognize Traffic signs and make decisions based on Traffic code using machine learning techniques. More precisely, after understanding the primary fragments of an image and video, we use an enormous amount of data to determine which essential Traffic sign the vehicle sees and make basic operations. The complexity of this problem has multiple subcategories; we classify them into three primaries: Image Recognition and Distance Detection and Response. Our approach is soft; begin like that; we will not develop a method with all Traffic signs; we will train a model with fundamental speed limits, stop signs, and without Traffic lights. We will use a Convolutional Neural Network (C.N.N) to build our model and try to have results as must as It good can be based on our knowledge. Our solution indicates that we will use image filtering processes before using those images in our Convolutional Neural Network. We choose to work with a standard model from Tensorflow, Mnist Classification.

Our classes are the following 30km, 50km, 70km, 120km, Stop, Turn Right Ahead and Turn Left Ahead. We used images from google search to create our dataset, and we used technics to grow our dataset because the available image was too few. For this reason, we use a webcam to capture those images at different angles and distances, so our dataset has as unique images as possible.

Our method uses the OpenCV Python library for image and video capture and pre-processing/filtering our images. In addition, we use Tensorflow with Keras API to create and train a model; we choose those tools because they have educational working models for numbers and alphabet letters identification, Mnist Classification. So this method will help us recognize the number and the letter that Traffic signs have.

1.3 Thesis overview

Chapter 2 will present all the essential information we need to understand the problem and try to create a method to solve the problem and the resulting subproblems. We will present the theory we need about image and video processing, the characteristics of traffic signs, the theory about machine learning and inferential neural networks, the skills we need for electronics such as DC motor control, the working principle of ultrasonic sensors, and the structure of a circuit to manage battery charging. We will describe the main applications for Embedded Systems (C programming), which will help us manage our system's complexity. Chapter 3 will give a complete overview of the problem we will try to solve. In addition, we will explain the known problems we will encounter during the traversal of this thesis. Problems based on the dataset, where can we find a dataset, how can we construct such a dataset, and what are the standards that determine that we will have a robust and quality dataset? How much time will it take to train a model, and what resources do we need, like do we need a super pc? Also, live video recognition has many problems compared to image recognition. How sunlight can affect the capture and processing of an image, how different camera focus angles, weather conditions, and the presence of objects (like trees) in front of the traffic sign can make our life difficult. In chapter 4, we will present our method from top to bottom so we can handle the issues mentioned in the previous chapter, starting from implementing our model for image processing and recognition and live video streaming. We will mention the implementation part of the program to run our model, i.e., using a barebone computer. We will justify our choice to use this embedded chip, what capability it offered us, how we managed it, to develop our code (C programming language) to control the different peripherals of the vehicle, including DC motors, lights, and distance sensors. In addition, we will dwell on the part of designing and manufacturing - printing the mechanical parts necessary to build our prototype car. In Chapter 5, we will verify our results and the performance of our solution; we will report if we have found problems that we did not expect or were not aware of during the implementation. The last Chapter 6 explains our results, i.e., it will act as the illustration of our conclusions. We will justify why our solution is considered successful or not. We will mention ideas to improve our method and make it workable for actual world conditions.

2 Background

2.1 Digital Image Processing

Digital Image Processing is the domain of science that affects the manipulation of a captured picture. The technics for retrieving, digitalizing, saving, transmitting, compressing, and restoring an image. We use those processes in various engineering projects, like biomedical, robotic vision, image-video transmitting (Meetings, Surgeries), 3D representation for medical purposes, and numerous Machine learning projects like face recognition.
How do we capture an image? The idea is that we use a sensor that detects and converts the variable attenuation of light waves into signals on two dimension axis (x,y) [4]. Then we have to sample (processing technic) this signal, quantized, and with an Analog to Digital converter, we digitized it [Figure 2.1]. A binary image, grayscale image, or color image are the classes of a Digital image. A **Binary Image** [5] is

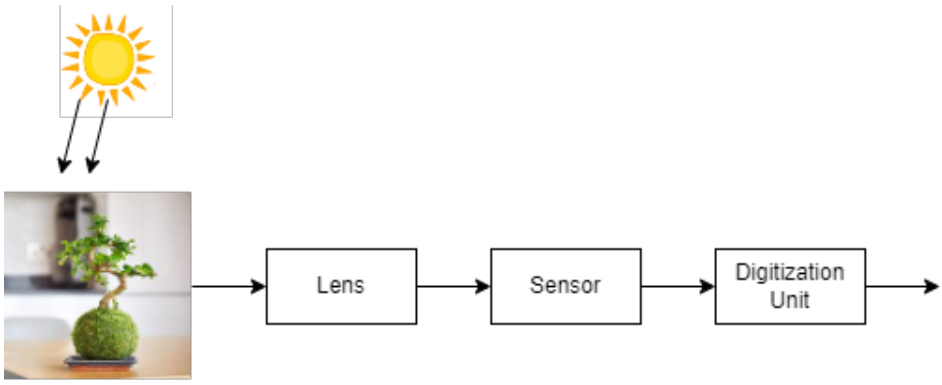


Figure 2.1: Digital Image Capture Process

the type with two levels of Brightness, Black and White (L=2, k=1). The low level "0" corresponds with the Black color, and as High "1", the White color. Storing a binary image needs fewer resources than the other classes because this type has the tiniest information and is faster to process those kinds of pictures. Binary images can provide critical information about an object's position, shape, and volume. We use them for fingerprint recognition, Optical Character Recognition(OCR), and Signature Recognition.
A **Greyscale Image** is a twodimensional array matrix $M \times N$ of pixels representing the intensity of an object (1):

$$I(i,j) \quad \text{with } i = 1,..,M \text{ and } j = 1,..N \tag{2.1}$$

where $0 \geq I(i,j) \geq (L-1)$.As **L** numbers in power of two,i.e $L = 2^k$ for $k = 8$ we have 256 shades of grey.

Digital Image represents the natural world as a set of numbers that can be stored and handled. This image is a combination of three monochromatic images. Each pixel in the color image has three segments corresponding to the brightnesses of the three monochromatic pictures.The representation of Color Digital image (2.2) :

$$I(i,j) \quad \text{with } i = 1,..,M \text{ and } j = 1,..N \tag{2.2}$$

where $0 \geq I_c(i,j) \geq (L-1)$, for each $c = 1,2,3$.

2.1.1 Traffic Signs Characteristics

If the object of our study was face recognition, then at this point, we should describe the details of our images. The significant points of the face are the nose, mouth, eyes, and ears. In our case, we are working with Traffic Signs. What are the main points of our images? As we know, the number of road signs varies from a few tens to a few thousand; it depends on the country, the size of the road network, and the particular weather conditions in the area. In Germany, for example, there are almost 1000 road signs [6]. Is it possible to remember all the signs and memorize their purpose? The answer is that we do not memorize them because that is not their purpose. A person's reaction when they see a sign must be immediate (it is critical in driving); it is impossible to think about what each sign means every time. From ancient times to the present day, we have structured the system of traffic signs based on features easily recognized by the human eye allowing the brain to make quick decisions. How would we describe a no entry sign[Figure 2.2]? For example, what features would we say that this sign has? We would say that the shape is round; it has a horizontal line, meaning that vehicles are not allowed to enter the road in that direction. Are these characteristics enough to understand the meaning of a sign? For an experienced driver, it is enough, but let us describe the characteristics of signs in detail. We can describe

Figure 2.2: No Entry

a road sign by three primary characteristics: color, shape, and symbols. Starting with the shape of a road sign, we have three types: triangular, rectangular, and circular. As for the other two characteristics, we can divide them into subcategories. The sign's color takes place into three parts: the outline, the background, and the pictogram. For example, in the Stop sign, the outline color is red, the background is white, and the pictogram - letters (S-T-O-P) are black. We defined the last feature as a Symbol, which could be letters, arrows, or even representations of people, children, animals, and vehicles. In addition, we classify road signs based on their purposes, such as warning signs, regulatory signs, informational signs, and additional signs. In this thesis, we will only use regulatory signs, but we will fully describe the characteristics of traffic signs. Let us mention the character that each sign has with the category to

which it belongs and the shape it has: Regulatory signs may have a circular shape and can define two categories based on the character, either Prohibitory or Instructional; on the other hand, informative signs which have a rectangular shape, either have as their character routing or information.

2.1.2 Color Systems

Colour is a sensation created in the brain by part of the sequence of electrical impulses that reach the brain through the optic nerve. The color information associated with the frequencies of the incident radiation is detected at the appropriate receptors and encoded within these electrical signals. The electrical impulses that relate to color information originate from specialized photosensitive receptors in the eye, the cones, which each responds to the detection of light of a particular wavelength range; this is how the human brain manages a color image. Based on this biological function, we have built various machines to store, process, and print a color image. We have more flexibility with the red, green, and blue systems (RGB) [7] and the cyan magenta and yellow systems[Figure 2.3] but there are six color systems. The first system is used in any solar device with a mobile phone screen, or tablet computer, while the second is used mainly by printers. We will only briefly mention the remaining systems, as they are not the subject of this study. Therefore, in addition to the two primary systems mentioned, we have the L.a.b model (Luminocity), RYB, which is similar to RGB, but the yellow has replaced the green, HSV (Hue, Saturation, Value) and HSL (Hue, Saturation, Luminocity). In this work, we will deal with the RGB system as we will use a digital camera and a computer; regardless, in the process of image processing, as we will describe in chapter 4, we will use a variant of the RGB system, this has to do with the library we will use for image processing.

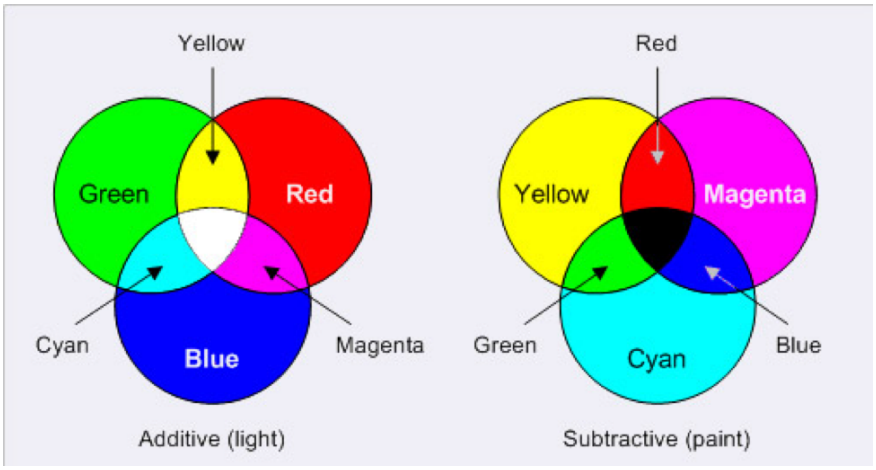


Figure 2.3: Mixing of basic and secondary colours

2.1.3 Greyscale Conversion

The RGB [8] model simplifies the design of graphical systems but creates difficulties in developing algorithms due to the correlation of the three color segments, making developing our model for image recognition markings more complicated. As we will mention below the histogram equalization procedure, we would prefer to use the HSI model, as we will have a more straightforward implementation. However, the primary method that we will follow is to transform the RGB model to Grayscale using the following equation(2.3) :

$$grayscale = 0.299R + 0.587G + 0.114B \tag{2.3}$$

Alternatively, it could define as the average value of the three color segments(but is not efficient for coding) (2.4) :

$$grayscale = 0.333R + 0.333G + 0.333B \tag{2.4}$$

2.1.4 Histogram Equalization

The histogram [9] of an image is the graphical representation in the form of bars of the intensity of each pixel of an image[Figure 2.4], where the intensity levels can be characterized as L in a range [0,255] and are defined according to the following function (2.5):

$$h(r_k) = n_k \tag{2.5}$$

Where r_k is the intensity level k in the interval [0,255] and n_k is the number of pixels in the image having an intensity level r_k.

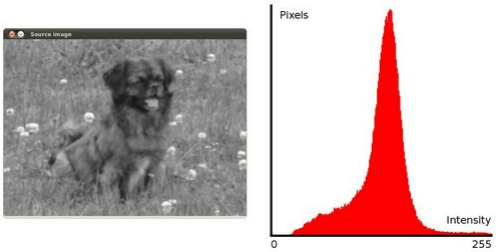


Figure 2.4: Histogram

Histogram Equalization is a computer image processing technique that improves contrast in photos. It accomplishes this by effectively spreading out the most frequent intensity values, i.e., stretching out the intensity range of the image. This method usually increases the global contrast of images when close contrast values represent its data. This technic allows areas of lower local contrast to gain a higher contrast.The green circles indicate the areas with underpopulated intensity if we look at the following [Figure 2.5]. If we apply a histogram equalization, we can see how those areas spread and the effect on our image.

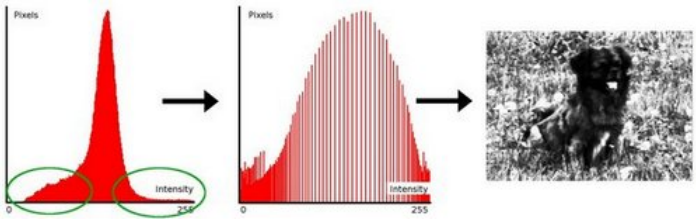


Figure 2.5: Histogram Equalization

We must use the cumulative distribution function to achieve this distribution change in the histogram. We have the original histogram for $H(i)$, while the cumulative distribution is $H'(i)$ (2.6).

$$H^{'}(i) = \sum_{0 \leq j < i} H(j) \tag{2.6}$$

2.1.5 Smoothing Images

Image smoothing [10] is used in image processing to remove noise or reduce the pixelation of an image. To achieve this, we usually use low-pass filters. As we will see in chapter 4, where we will implement such techniques, we will focus on the following techniques, for which there will be an example and explanation of what they can offer us in our model. Before explaining each technique, we should mention that Kernel filters (2.7) are convolution matrices or masks used for blurring, sharpening, embossing, and edge detection.

Kernel Filter:

$$K = \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.7}$$

The method of **Averaging** is convolving an image with a normalized box filter [Figure 2.6]. It simply takes the average of all the pixels under the kernel area and replaces the central element. In the method of **Gaussian Blurring**, instead of a box filter, we use a Gaussian kernel.

The **Median** is the method where we take the Median of all the pixels under the kernel area, and the central element is replaced with this median value. Median is highly effective against salt-and-pepper noise in an image. Interestingly, in the above filters, the central element is a newly calculated value which may be a pixel value in the image or a new value. However, in median blurring, the central element is always replaced by some pixel value in the image. It reduces the noise effectively. Its kernel size should be a positive odd integer. [11]

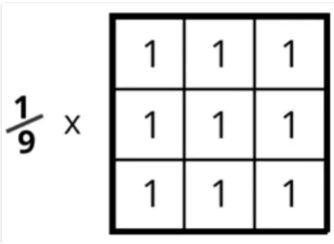


Figure 2.6: Normalized Kernel Filter Box

Last, **Bilateral** Filtering is highly effective in noise removal while keeping edges sharp. However, the operation is slower compared to other filters. A gaussian filter takes the neighborhood around the pixel and finds its Gaussian weighted average. This Gaussian filter is a function of space alone; nearby pixels are considered while filtering. It does not consider whether pixels have almost the same intensity. It does not consider whether a pixel is an edge pixel or not. So it blurs the edges also, which we do not want to do.

2.1.6 Edge Detection

Edge Detection [12] is an image processing technique [Figure 2.7] used to detect points in a digital image with discontinuities, i.e., sudden changes in the image's brightness. These points at which the brightness of the image changes abruptly are called edges (or boundaries) of the image. This process ensures significant data reduction and filters out unhelpful information, keeping only the critical information.

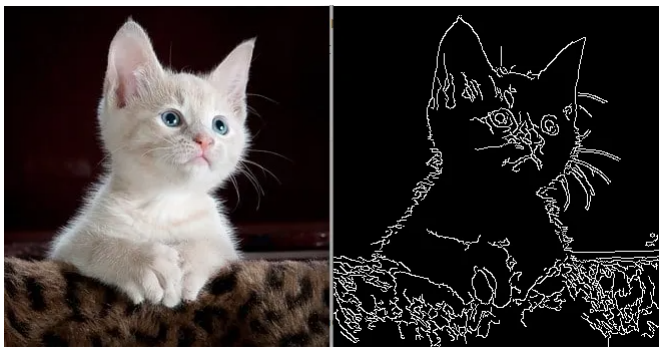


Figure 2.7: Edge Detection

There are many different methods for detecting edges; the most important ones can be categorized as Gradient and Laplacian.
In **Gradient**, we detect the peaks by looking for the maxima and minima in the first derivative of the photograph. The first derivative makes the edges smoother or thicker. The first derivative and its magnitude are:

$$\nabla f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \tag{2.8}$$

$$mag(\nabla f) = \begin{bmatrix} (\frac{\partial f}{\partial x})^2 + (\frac{\partial f}{\partial y})^2 \end{bmatrix} \tag{2.9}$$

or simply for a $3X3$ Matrix

$$\nabla f = |\frac{\partial f}{\partial x}| + |\frac{\partial f}{\partial y}| \tag{2.10}$$

A concatenated method of enhancing an image is the **Laplacian** mask filter. The operator is defined as follows:

$$\nabla^2 f = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \tag{2.11}$$

The most famous algorithms for edge detection with Gradient type: are Sobel, Prewitt, and Roberts. And for Laplacian type **Canny Edge Detection**.

Prewitt [13] is an Edge Detection technic that can detect two types of edges: Vertical and Horizontal Edges. It is known as the best operator for detecting image orientation. It uses derivative $3X3$ kernel masks one for each direction, Vertical and Horizontal . These Masks should have the following properties: an opposite sign should be present in the mask, the sum of the mask should be equal to zero, and more weight means more edge detection. The disadvantage of this method is that magnitude of the coefficient is a constant value, and we can lose diagonal details.

Sobel operator is very similar to Prewitt [14]. It was almost the same mask with only a difference; the neighbor pixel of zero has two as magnitude[Table 2.1 ,2.2] so it work with the most close pixels from the center. It is a straightforward and efficient operator; it can use to search for smooth edges. The disadvantages of this method are sensitivity to noise; it is not as accurate in finding edges, can lose information on the diagonal, and cannot detect thick and rough edges.

Table 2.1: Sobel Horizontal Mask

-1	0	1
-2	0	2
-1	0	1

Table 2.2: Sobel Vertical Mask

-1	-2	-1
0	0	0
1	2	1

Robert operator is based on calculating the sum of squares of the differences between diagonally adjacent pixels in an image by discrete differencing. The gradient approximation is then performed. It uses kernels or $2X2$ masks. The advantages of this method are the easy finding of the edges and the orientation of the image; it also includes the diagonals. However, it is sensitive to noise and is not the most accurate method.

Canny Edge is not sensitive to noise. It extracts the features of the image without affecting or altering the feature. Canny edge detector has advanced algorithm derived from Laplacian of Gaussian operator. It is a widely used optimal edge detection technique. It detects edges based on three criteria: low error rate, the edge points must be accurately located, and there should be only a single edge response. The main drawback is the complexity and performance time.

2.2 Neural Network

2.2.1 Introduction to Neural Networks

Artificial Neural Networks or Neural Networks began as an idea for the mathematical representation of biological neurons [Figure 2.9] by the mathematician Walter Pitts and the neuropsychologist Warren McCulloch in 1943 [15], their research aimed to examine the function of neurons. In order to simulate how real neurons in the human brain work, they constructed electrical circuits. In 1949 Donald Hebb wrote The Organization of Behavior, which pointed out that neural pathways grow each time they are used, is a concept fundamentally necessary for how people learn. He argued that if two nerves fire simultaneously, their connection is strengthened. In 1959 the two models ”MADELINA” and ”ADELINA” were developed by Bernard Widrow and Marcian Hoff. The names of the models came from the acronyms, Multiple ADAptive LINear Elements.
”ADELINA” was developed to analyze binary patterns so that it could predict successive bits on the telephone line. The second model, which in some ways is still in use today, was the first neural network to be applied in real life and was designed to reduce echo during a telephone call. In 1962, Widrow and Hoff developed a learning procedure that examines the value before the weight adjusts it (0 or 1) according to the rule: (Error/(Number of inputs)). It is based on the idea that while an active perceptron may have a significant error, one can adjust the values of the weights to distribute it across the network, or at least to neighboring perceptrons. Applying this rule still leads to error if the line before the weight is 0, although this will eventually correct itself. If the error is preserved and distributed across all weights, then the error is eliminated.Around 1975 Paul Werbos created the Back Propagation technique to solve the XOR problem. Back Propagation is the way for a neural network to understand its” mistakes” and learn from them by feeding its output back to its inputs and calculating the error from a set point (actual data), as shown in [Figure 2.10].
Back Propagation was a fundamental step in Neural Networks’ evolution as it has been used in all Neural Network algorithms from that day. There was no improvement in the field for the following decades, except, maybe, from the Max-Pooling method, introduced back in 1992 [16] for 3D object recognition. This method helped to the slightest shift invariance and the tolerance to deformation. Things kept going on for Neural Network models with small improving steps through the subsequent years. The problem preventing Neural networks’ evolution was mainly hardware limited to support the resources needed for such projects. Big data and the Internet of Thing’s evolution were the beginning. We all know how new generation graphic cards take it to another level. Neural Networks are considered to be the state of the art mathematical model for solving almost every difficult and complex problem in Computer science, and especially in the Artificial Intelligence field. Many models were created in the past ten years, like Recurrent Neural Network (RNN, LSTM) [17], Convolutional Neural Network [18] (CNN), or Residual Neural Networks [19] (ResNet).

The basic structure of a neural network [Figure 2.8] consists of the combination of nodes, the input layer, the output layer, and the hidden layer, which can be one or more. Each node or artificial neuron connected to another has a weight and threshold relationship. If the output from any node is above a specific threshold value, then the node is activated and sends data to the next layer. Otherwise, the data is not fed to the subsequent nodes. Neural networks depend on training data to learn and enhance their performance over time. Nevertheless, when the learning algorithms are appropriately tuned, we create powerful algorithms to sort our data quickly so that image recognition tasks can take a few minutes. The most well-known neural network we use in everyday life is the algorithms used by search engines like Google.

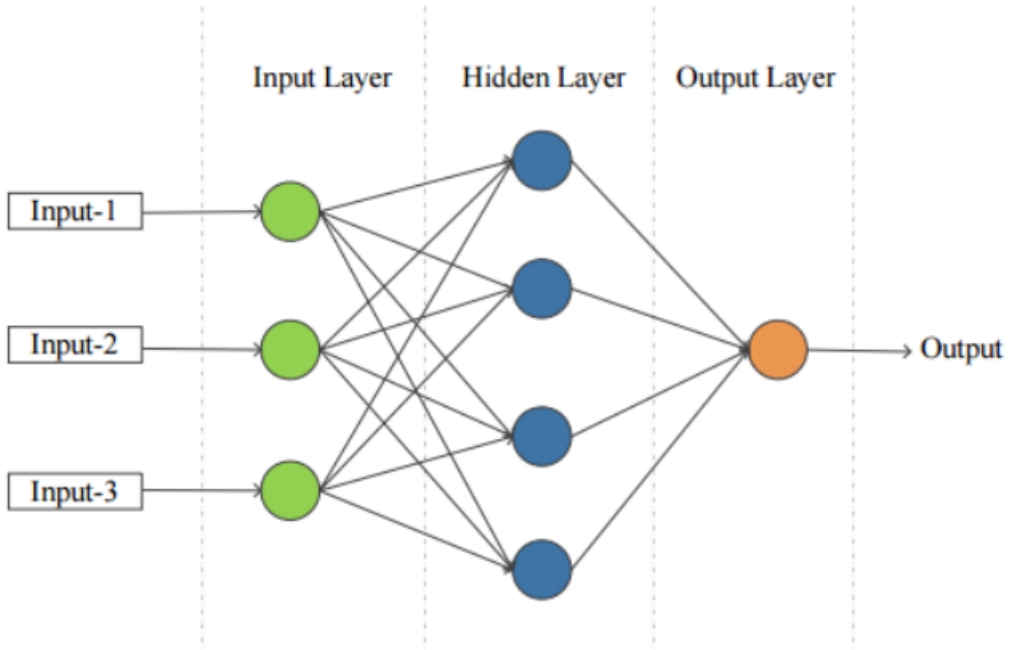


Figure 2.8: Neural Network Model

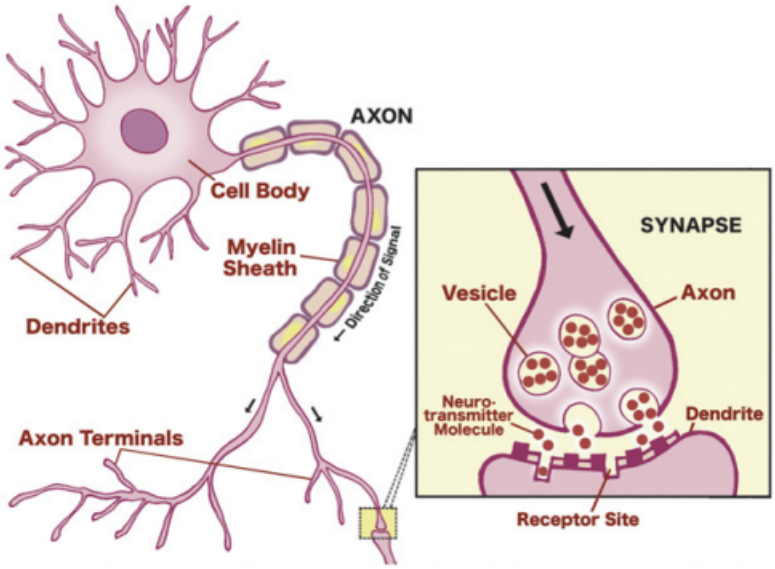


Figure 2.9: Real Neuron Humans' Brain

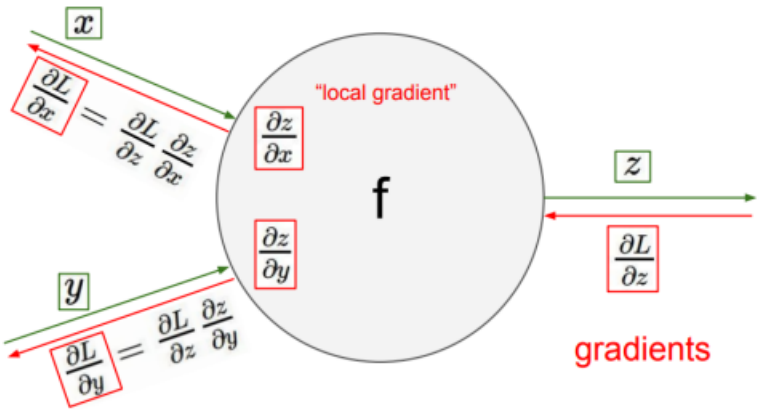


Figure 2.10: Single Neuron Back Propagation

2.2.1.1 The mathematical analysis of neural networks

Each node is a linear regression model which consists of input data weights, threshold, and output; these values are between 0 and 1 [20]. If we wanted to present the mathematical relationship, it would be the following:

$$\sum_{i=1}^{m} w_i x_i + bias = w_1 x_1 + w_2 x_2 + w_3 x_3 + bias \tag{2.12}$$

$$output = f(x) = \begin{cases} 1 & \text{if } \sum w_1 x_1 + b \geq 0 \\ 0 & \text{if } \sum w_1 x_1 + b < 0 \end{cases} \tag{2.13}$$

2.2.2 Convolutional Neural Networks

Convolutional Neural Network [21] (CNN) is a type of Artificial Neural Network (ANN) mainly used for image recognition and classification tasks. In general, a CNN model can be seen as a regularization of multi-layer Perceptrons, which are a kind of fully connected networks. By fully connected, we mean that each layer's node is linked with every node of the neighbor layer. This property attributed to multi-layer Perceptron is prone to overfitting data. So, by taking advantage of hierarchical patterns in data, CNN tries to assemble the most complex patterns into simpler and smaller ones, reducing the complexity and connectedness of the model. From the aspect of image classification tasks, CNNs demand relatively less pre-processing of data, compared to classic pattern recognition algorithms because they manage to learn the filters that fit to the extraction of feature sets that otherwise would need to be hand-crafted. It is a massive plus for the engineers, since it requires slighter human effort for the design and development of complicated solutions for such kinds of problems. By the time this text was written, CNNs were the primary architecture [Figure 2.11] used for many problems besides image classification. For example, CNNs are used now for Speech and Speaker Recognition is also gradually replacing Recurrent Neural Networks, which are used for input data based on time-sequence; an example of this kind of data is sound (or voice). Also, we should not forget that convolutional networks are inspired and are fundamental for some state-of-the-art models, like ResNets .

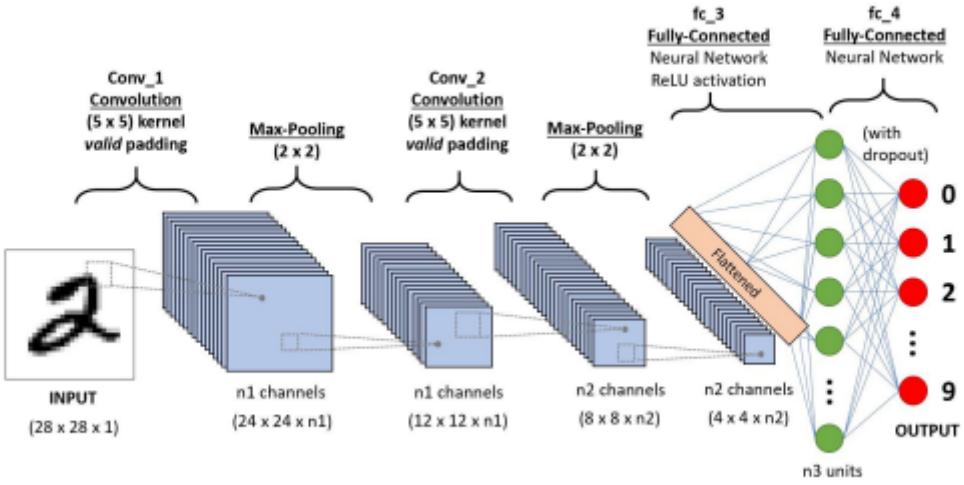


Figure 2.11: Convolutional Neural Network example

2.2.2.1 Convolution Layer

The critical piece of the Convolutional Neural Network is the Convolutional Layer [22]. It is applied to our input so that we filter the input Matrix to extract useful features for the training process. The parameters of the Convolutional Layer consist of a set of learnable filters, which have a small receptive field but extend over the entire depth of the input volume. Each filter is applied along two dimensions (height and width) at each pass through the layer. This layer computes the dot product between the filter entries and the input and produces a two-dimensional map of the activation of that filter. Thus, we train the network to activate the filters when it detects a feature. In any machine learning framework, there are two types of convolutional. One is for two-dimensional and the other for three-dimensional input volumes. When initializing the Convolutional Layer (2.26), we are mainly interested in dealing with the three primary parameters, the filter size, which specifies the final dimensions of our output, and the Kernel size, which is usually 3X3 and is, in fact, the filter that we will apply along the two dimensions. Another parameter is Stride, which determines if we have interference when applying the filter.

$$G[m,n] = (f * h)[m,n] = \sum_j \sum_k h[j,k] f[m-j, n-k] \tag{2.14}$$

where h is the filter(kernel), f is the image-vector, and m, n is number of rows and columns or image vector of size m × n.

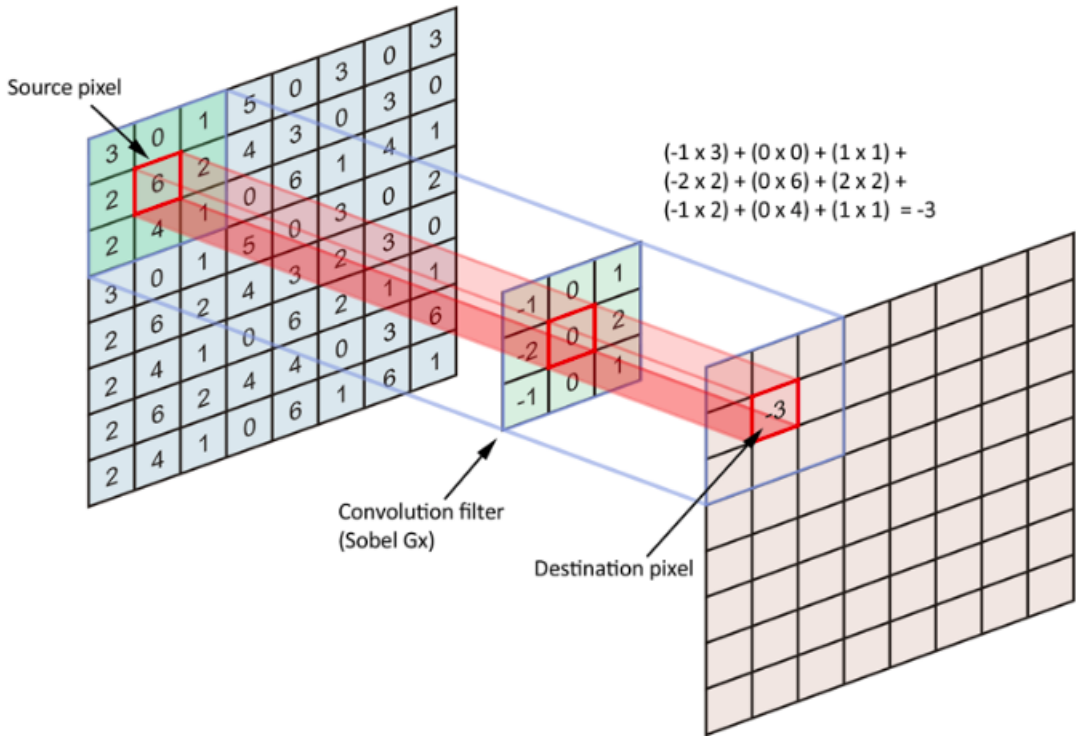


Figure 2.12: Convolutional Layer

2.2.2.2 Max Pooling

The Max Polling [23] is the next step we mainly use in Convolutional Neural networks after we apply a Convolutional Layer. It is a method that is Down-sampling the outputs from layers. We use this technic to reduce the dimensionality of our data by searching for the more significant values so we can make valuable assumptions about features for our training processes. In addition, we use that method to limit the possibility of overfitting in our model.

In the below [Figure 2.13], we have an example of the Max pooling method [Figure 2.14], our input is 4 X 4 Matrix, and we apply Max pooling 2x2 with a stride of 2. In this example, we keep only the high values of our input Matrix.

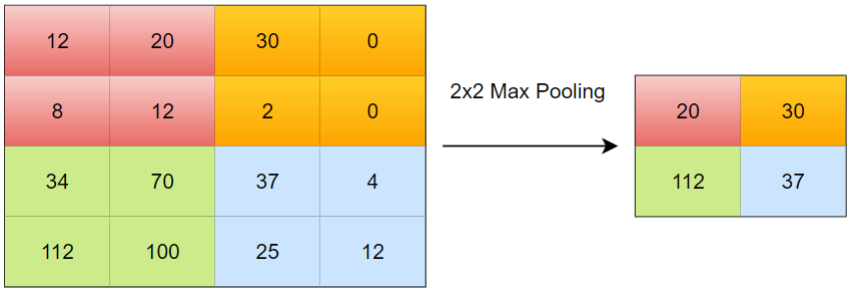


Figure 2.13: Max Pooling

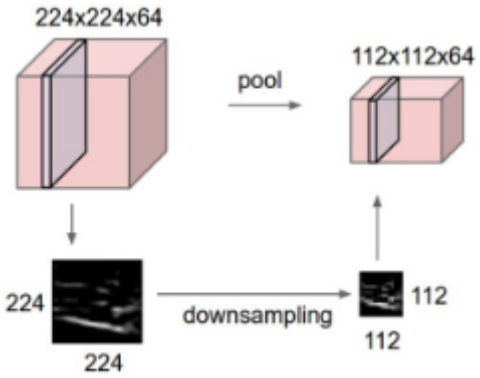


Figure 2.14: Max Pooling in a image

2.2.2.3 Activation Functions

The activation function [24] in a Convolutional Neural Network represents how the weighted sum of the input is transformed into an output from a node or nodes in a network layer. It maps the node's result values in [0, 1] or [-1, +1], depending on the function.Activation Functions can be referred to as "Transfer functions." If the output of the functions is limited, we call it a "squashing function" and another called "nonlinearity " when the output is not linear. It is often to use a nonlinear Function in Neural Networks; the most known is the Sigmoid [Figure 2.15] (2.27). However, we mainly use a variant of the Sigmoid, the ReLU –Rectify Linear Unit–[Figure 2.16], which is half rectified in $x \in [-\infty, 0]$. The function and its directive are both monotonic (increasing).

$$\sigma(x) = \frac{1}{1 + \exp^{-x}} \tag{2.15}$$

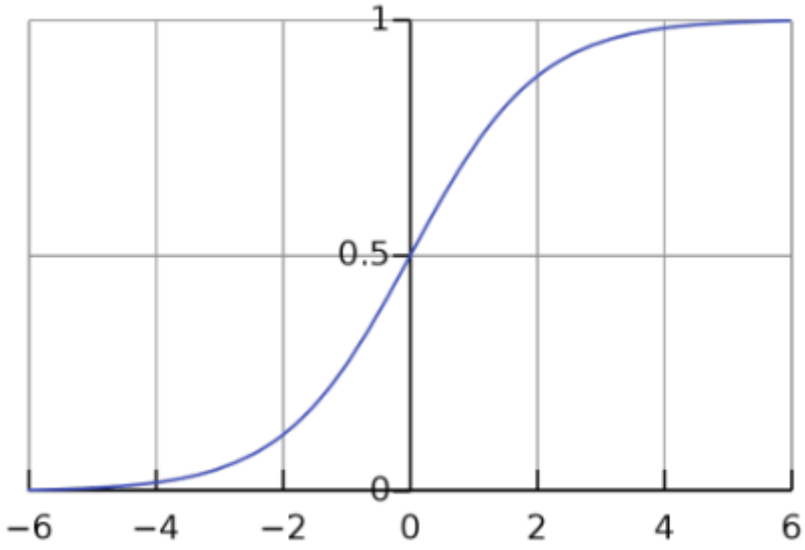


Figure 2.15: Sigmoid Function

$$R(z) = \max(0, x) \tag{2.16}$$

$$f(x) = \begin{cases} 0, & \text{if } x < 0 \\ x, & \text{if } x \geq 0 \end{cases} \tag{2.17}$$

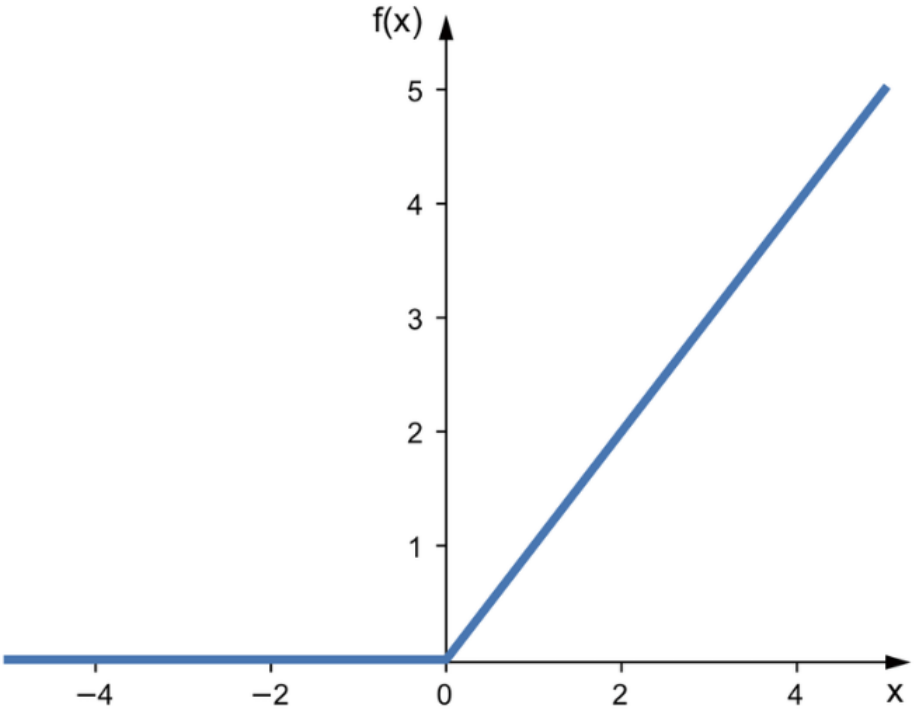


Figure 2.16: Rectify Linear Unit

2.2.2.4 Loss Function

The development of the Machine Learning Model demands many hours of trying different things and testing. This procedure birth a new question, How can we evaluate our output every time we train our Model? We have to examine the deviation between the prediction output with the actual value; with this method, we can evaluate our Model and describe if we improve our Model with the changes we make. This process can be done by **Loss Function - Error Function**. It is the graphical representation [25] of the deviation between our predicted and actual values. It is frequent ovMachine Learning to use this type of function to estimate our Model's accuracy each time we make modifications. It gives us the feedback we want to understand what we are doing. The Goal of those functions is to describe how well our data fit in the algorithm. We will present the most known functions commonly used, and it is very critical to pick the proper Loss Function so it can reflect our Model.

Mean Squared Error (MSE) [26] calculates the average error square.Specifically we calculate the average square of the difference between y predicted, with the actual value as the following equation (2.18).Below we have the graph that represents the MSE.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}\left(Y_i - \hat{Y}_i\right)^2 \tag{2.18}$$

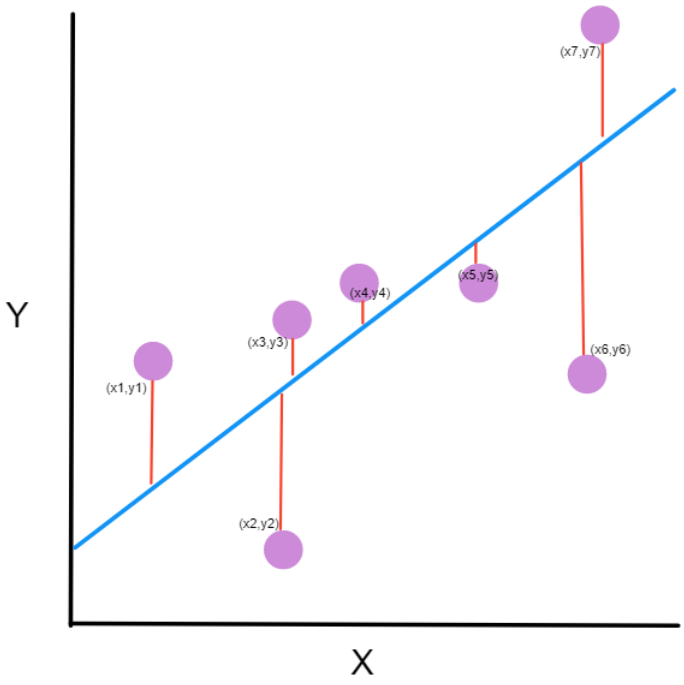


Figure 2.17: Mean Squared Error

The **Likelihood Function** (2.19) is commonly used in classification projects [27]. This function takes the predicted probability for each input and multiple them.For example,we have the following probabilities $[0.8, 0.2, 0.7, 0.3]$ for the following ground truth values $[1, 0, 1, 0]$ .The likelihood will be calculated as $0.8 \cdot 0.8 \cdot 0.7 \cdot 0.7 = 0.313$; this probability is concerned with the truth (or 1). In order to calculate the false value, we use 1-p.We have attach the a likelihood graph example [Figure 2.18]

$$L(\theta) = L\left(\theta \mid x_1, x_2, \ldots, x_n\right) = f\left(x_1, x_2, \ldots, x_n \mid \theta\right) \tag{2.19}$$

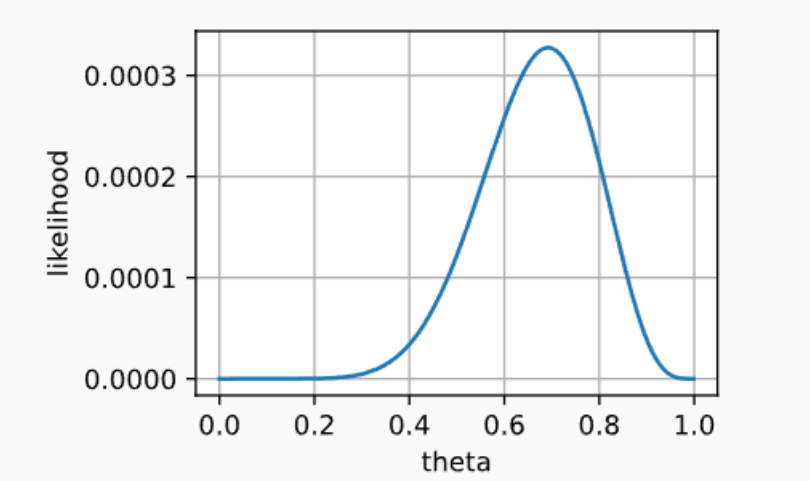


Figure 2.18: Likelihood Plot

The most used and reliable Loss Function in Convolutional Neural Network is the **Cross Entropy Loss** [Figure 2.21]. We are familiar with cross-entropy, mostly from telecommunications and signal processing. In Machine learning, we use cross entropy equation (2.20,2.21) to estimate the distance between prediction outputs (labels) and actual labels.

$$H(p, q) = E_p[-\log(q)] \tag{2.20}$$

$$H(p, q) = -\sum_{x \in X} p(x) \log q(x) \tag{2.21}$$

So, in neural networks, qi are the predicted labels, and pi Re are the true labels (pi and qi refer to the labels of a training sample i). A machine learning algorithm – like Neural Net – aims to minimize the cross entropy until its optimal point. That is the reason we need a kind of optimizer, so that cross-entropy converges faster and avoids local minimal points.

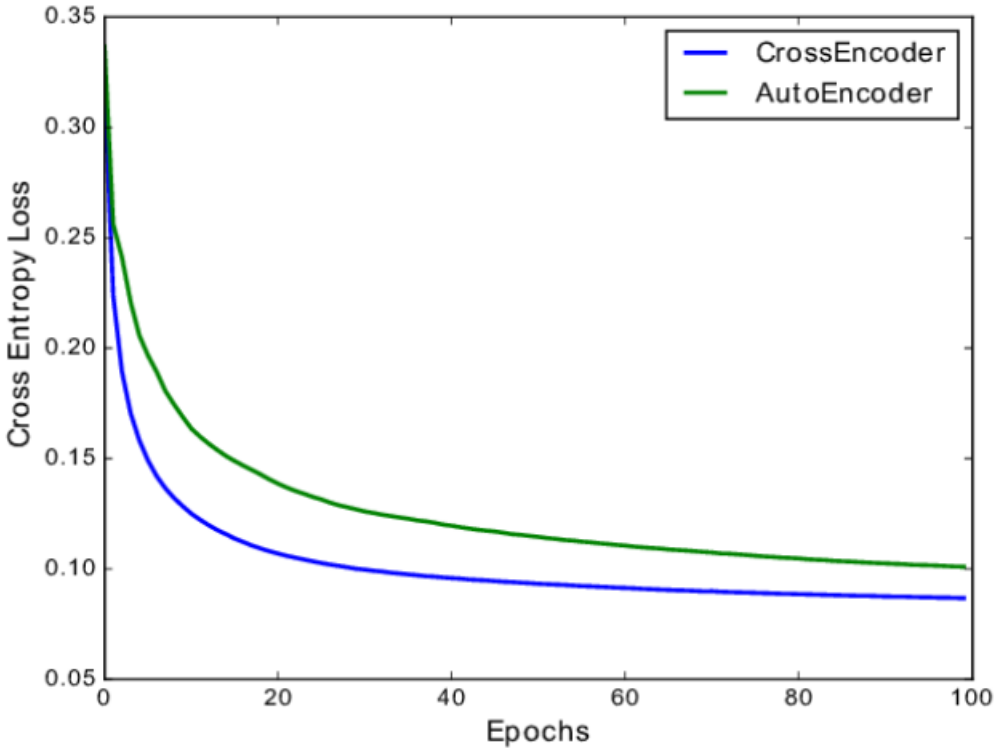


Figure 2.19: Cross Entropy Loss Plot

2.2.2.5 Adam Optimizer

Adaptive Moment estimation, or Adam, [28] is the broadest optimization algorithm that extends Stochastic Gradient Descent and RMSprop [40], and its goal is to update the network weights iteratively based on the training data. In a nutshell, it is an adaptive learning rate method, which means it computes individual learning rates for different parameters, and it uses the estimations of the first and second moments of a gradient to adapt the learning rate for each weight of the neural network. At the moment, we mean the N-th value of a random value, defined as the value of that variable to the power of N. This random variable s represents the gradient of the loss function.
We will try to explain step by step the mathematical process of Adam.In the first step, all vectors of moving averages are initialized with zeros, so m0 = 0, which is simply the slope average, and then there is the second moment, i.e., the uncentered Thus, the optimizer Adam tries to estimate the moments using exponentially the moving averages, which are computed over the gradient evaluated in the current minibatch.The moving average equation as shown below(2.22,2.23).

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1)\, g_t \tag{2.22}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)\, g_t^2 \tag{2.23}$$

Then, we have the expected values of the estimators, which are described by the following equations:

$$E[m_t] = E[g_t] \tag{2.24}$$

$$E[m_t] = E[g_t^2] \tag{2.25}$$

2.3 Embedded Systems

2.3.1 Real Time Operating System

The real-time operating system (RTOS) was invented to serve applications with real-time requests [29]. It is a scheduling enforcement structure. When our system receives a stimulus, it has to manage it in an interwoven time for each time we receive them. In our work dealing with critical events and tasks, it is mandatory to have reliability and repeatability when taking action after an event.
The RTOS must be able to process the data as it comes in, usually without buffering delays. Processing time requirements are measured in tenths of a second or less. Over the years, it has evolved from simple stems using cyclic scheduling to today's feature-rich operating environments[]. To make it more counterintuitive, real-time refers to the merit of having something that offers the execution of real requests and not the speed of program execution; speed has to do exclusively with the piece of hardware, i.e., the frequency of operation of the microcontroller, the speed of write and read, faster bus interfaces which do not determine whether the system in real-time.
The essential components [Figure 2.20] a Real-time operating system are the following: the scheduler, where we define the priority of each task. Symmetric Multiprocessing (SMP) is the number of tasks that can execute in parallel on RTOS. Function Library is an interface that helps to connect kernel and application code. This application allows us to send requests to the Kernel using a function library so that the application can give the desired results. Memory Management manages the allocation of memory to every program, which is the essential element of the RTOS. Fast dispatch latency is the interval between the termination of the task identified by the OS and the time taken by the thread in the ready queue that has started processing. User-defined data & class refer to the C or C++ structure.

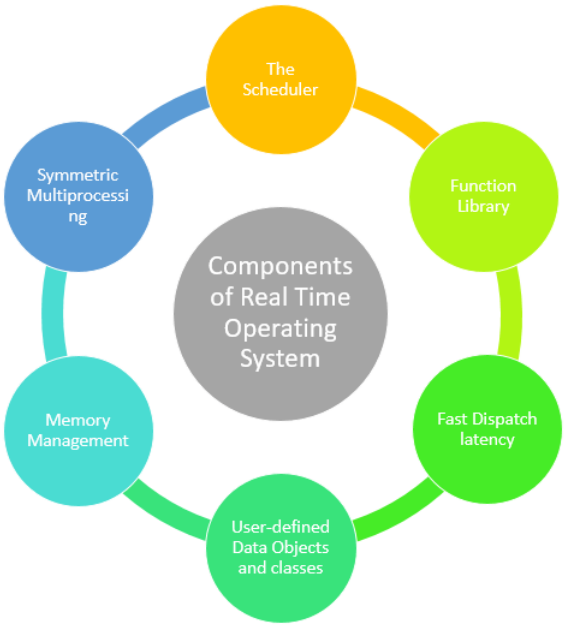


Figure 2.20: RTOS Components

In the graph below [Figure 2.21], we will compare the response time of real-time and non-real-time applications. As we can see, the y axis represents the response time, and the x represents the iterations. In a non-real-time system, we can see that we have a different response time; for the first execution iteration, we have 4s, and for the second, 1.5sec, which is a significant deviation and would be very inconsistent for an application in a vehicle braking system. On the other hand, the real-time graph we observe is constant in each iteration, and that is how we want our system to work.

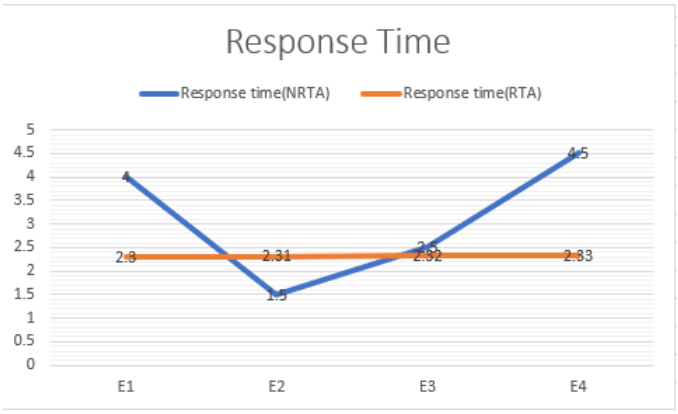


Figure 2.21: Real Time - Non Real Time Response

The main features of the real-time application:
- They are not applications that run quickly
- RTA's are temporally deterministic.
- They may have a tiny deviation in response time, but we have soft real-time applications when we have a deviation close to ms.
- Hard real-time functions must complete within a given time.

Since we have mentioned hard and soft real-time, we will introduce the types of the real-time operating system. We have three types, hard-real time, soft-real time, and firm-real time.

In **Hard- real-time**, deadlines are met with great precision; any delay will cause the system to fail. Examples of use are in medical applications, airplanes, and cars. A representative example is the car airbag system's response time after the crash sensor's activation.

The **Firm** type of RTOS also needs to follow the deadlines. However, missing a deadline may not have a significant impact but could cause undesired effects, like a massive reduction in product quality.

Soft Real-time RTOS accepts some delays by the Operating system. In this type of RTOS, a deadline is assigned for a specific job, but a delay for a short time is acceptable. So, deadlines are handled softly by this type of RTOS. An example of soft-real time is the keyboard's response time on our pc.

2.3.1.1 RTOS vs. GPOS

When we refer to a general purpose operating system (GPOS), we mean operating systems such as Linux and windows. What are the differences between GPOS and RTOS? The first main difference is the time criticality versus throughput. The first goal with RTOS is to achieve deterministic behavior; as we have mentioned, we want results on very tight deadlines. The RTOS is mainly concerned with timeliness and not so much with the amount of work. On the other hand, the GPOS aims at high performance and can manage many tasks which must be solved in a specific time. The second difference concerns the programming of algorithms in GPOS. We can use any algorithm we want, as long as we achieve high performance.RTOS is more limited and based on priority; a high-priority task will never reject other lower-priority tasks; it can only be interrupted by a higher priority task. Another difference is the latency term which is one of the most important in these systems. In GPOS, we have enough threads to add the system according to the latency; the RTOS, however, is more limited in the resource issue. Finally, RTOS is mainly used for small modules such as microcontrollers, while we have more powerful computing systems for general purposes.

2.3.1.2 FreeRTOS and others

Currently, there are several RTOS in the embedded systems market; some are free, and others have a subscription. The engineers are often faced with challenges, such as choosing a superior RTOS in quality and cost or complexity in architecture. We will present the most famous ones, starting with **FreeRTOS** from Amazon, the most popular software, and is supported by a wide range of family microcontrollers, for example, AVR, PIC, PIC32, MSP430, 8052, STM32, and many others. They are one of the few that are generally available for free. However, we can buy some support for very demanding applications from the company that develops them. The most well-known operating system is **VxWorks**, which was invented in 1987 and is a product of Aptiv, and ranks high on the list of paid software that has been used extensively in NASA projects. The most well-known platforms that support it are ARM,IA-32 and Intel 64.
Next on our list is the open source **TI-RTOS** which comes from Texas Instruments and is mainly supported by microcontrollers from the same company. Next, we have **ThreadX**, a payload belonging to Microsoft; its license starts at 12000 euros and supports most platforms. Finally, the latest and up-and-coming **Zephyr** RTOS, ideal for IoT applications and has gained a large engineering community in the last couple of years, is open source and comes from the Linux Foundation. In the context of this work came in contact with FreeRTOS and Zephyr. We did several tests to find the one we liked the most; we ended up with FreeRTOS.In chapter 4, where we will present the structure of our solution, we will explain why we came to this decision.

2.3.2 Universal Asynchronous receiver-transmitter

A serial interface can be point-to-point or multi-node.In a point-to-point interface, only two nodes are allowed on the cable: in one part, we have the transmitter, and in the other, the receiver. When this pair swaps functionality, the communication is called bidirectional, while when the role remains constant, it is called one-way [30]. Two-way is divided into two categories full-duplex when each device can send and receive data simultaneously. Half-duplex communication is when each device can send or receive data, but not simultaneously. In the multi-node interface, many nodes are allowed on the same wire. Such an interface can have one master or many (multi-master). The master is responsible for initiating the communication and managing the clock.

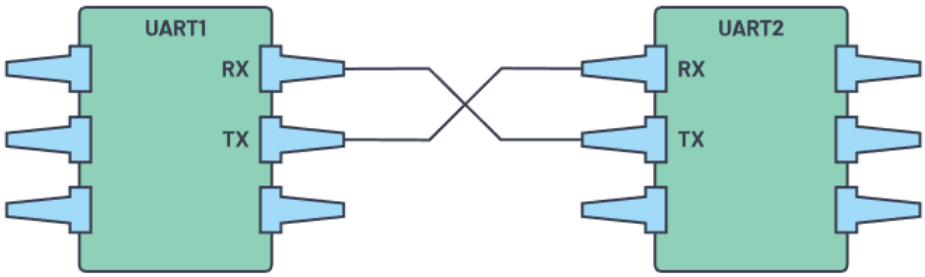


Figure 2.22: Universal asynchronous receiver-transmitter

The first protocol we think of is the Universal asynchronous receiver-transmitter (UART) when talking about device-to-device communication. It can be associated with many serial protocols transmitting and receiving serial data.in serial communication, data is transferred bit by bit using a single line or wire. In two-way communication, we use two wires. In many applications, we need to restrict the cost by minimizing the wires, so we choose the serial protocol that needs fewer wires. UART is mainly used on Embedded systems for communication between microcontrollers and computers as a hardware communication protocol.UART needs only two wires for the communication process. It is defined as an Asynchronous protocol, which means there is no clock to synchronize the outputs of the devices.[Analog] The transmitter generates a bitstream based on its clock signal, while the receiver uses its internal clock signal to sample the incoming data.Synchronization between devices can perform if we use the same baud rate. In asynchronous serial transmission, the start bit is transmitted first, then the 8 bits of the character from the low order bit to the high order bit, and finally, the stop bit [Figure 2.23].This frame can also be configured as follows: either two stop bits instead of 1, or after the last high bit, a parity bit is emitted, and finally, for each character, 7 or 9 bits are emitted instead of 8. The parity bit can be of even or odd parity and is used to check the correctness of the transmission; however, it is no longer used so often as a technique.

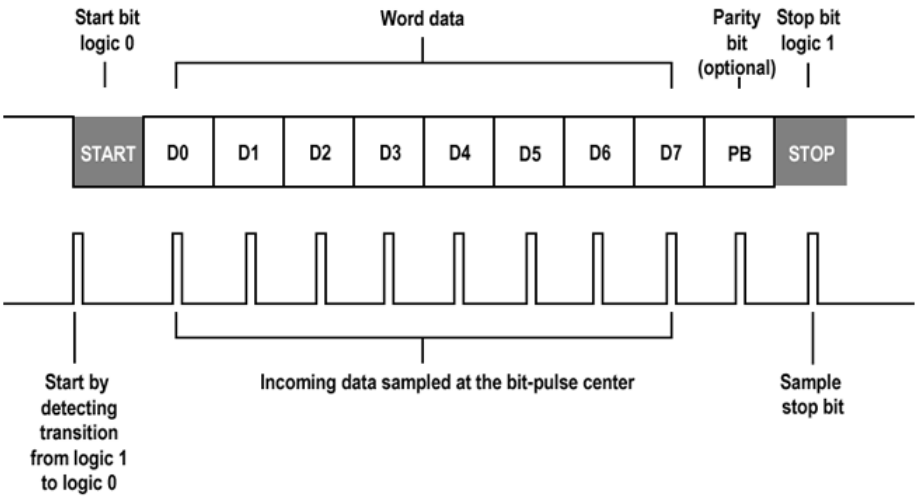


Figure 2.23: UART Frame

2.3.3 Interrupt Vs Polling

An interrupt is a mechanism provided by both the CPU of a microcontroller and the peripherals. With this mechanism, the peripheral causes the program to execute a specific routine when it wants to. More concretely, the interrupt [31] is a signal that is transmitted to the processor; such an interrupt can be either Hardware or Software and is an event that needs immediate execution. When the microcontroller receives an interrupt, it terminates the task it is performing and, at that moment and immediately executes the **Interrupt Service Routine** (ISR) [Figure 2.24]. An interrupt is defined as Hardware when an

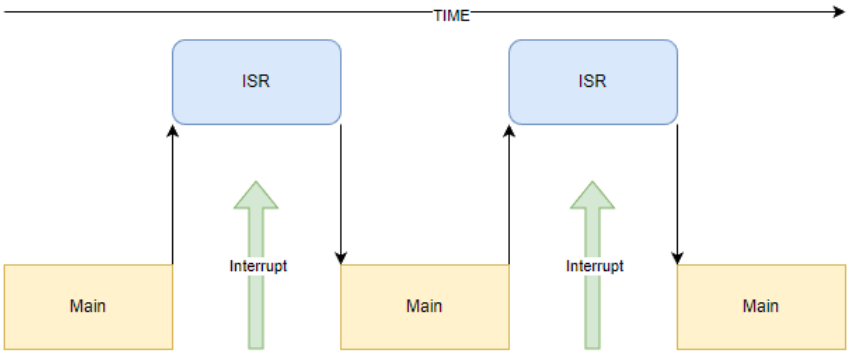


Figure 2.24: Interrupt Service Routine

external device sends a signal to the processor; a typical example is when a key is pressed on the computer keyboard. On the other hand, a software interrupt occurs when an exceptional condition occurs in the instruction set. For example, if the numerical logic unit of the processor executes an instruction to divide a number by zero, it causes a divide-by-zero exception, thus causing the computer to abandon the calculation or display an error message.
Continuous monitoring is known as **Polling** [Figure 2.25]. The processor constantly checks the status of peripheral devices, - and while doing so, it performs no other function and spends all its processing time on monitoring.

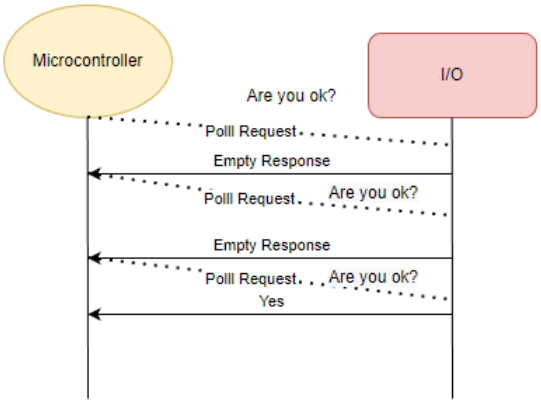


Figure 2.25: Polling Method

2.3.4 Direct Memory Access

Direct Memory Access (DMA) can use the bus as a master [Figure 2.26] to read and write to physical memory [32]. The purpose of the DMA device is to offload the software and the processor from copying large chunks of data from one memory address to another. With this method, the input and output devices send and receive data from the main memory directly, thus bypassing the processor and speeding up the memory operations. DMA devices support two modes of operation: **continuous transfer** and **scatter-gather lists**. In the continuous transfer method, the devices copy bytes sequentially, starting from one physical memory address and sending the data to another. In the scatter-gather lists method, the data is copied from a data structure called a scatter-gather list to a contiguous memory area starting from the destination address. The processor's occupation is the main difference between the Inturrept/Polling

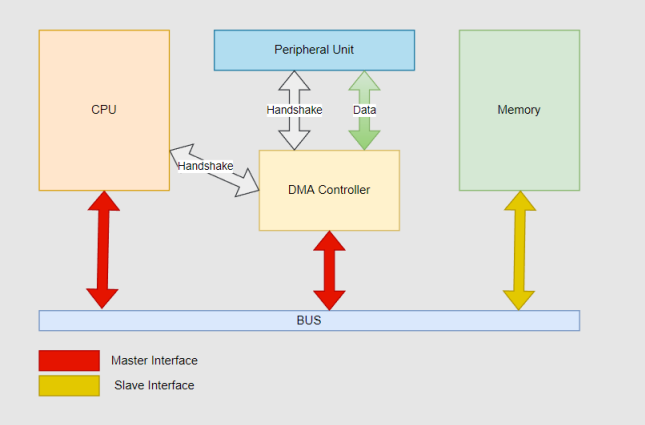


Figure 2.26: Direct Memory Access

method and the DMA method. In interrupt or polling methods, the data from the input/output devices are handled by the processor, either sending or receiving data; the processor has to interrupt its operation every time, using the bus for physical memory to write and read the memory locations. This process loads our processor and causes long implementation delays. However, in the DMA method, we leave the processor to execute its tasks without interruptions. Typical types of transfers that DMA could control :

- Memory to Memory transfers (internal to external)
- Transfers from I/O devices to memory.
- Transfers from memory to I/O devices.
- Transfers from/to communication ports and memory.
- Transfers from/to serial ports and memory

The DMA's physical source and destination address must first be written to the registers. In the first address register, we write the destination address, and the data set on a continuous block is also entered. In the Source register, the data is entered either in a continuous or scatter-gather list.

2.4 Electronics

2.4.1 Electric Moters Drive System

Several applications in our daily life use DC motors. The control of Dc motors is something that engineers face in many areas like robotics, automobiles, and several industrial applications. The control of these motors is straightforward and fully controllable; we can accurately determine the speed and direction of rotation. There are many control techniques; this paper will deal with the most famous one, the H-Bridge circuit with Pulse Width Modulation Technique.
As mentioned above, electric drive is becoming increasingly necessary daily in many applications and for a more comprehensive power range. The main reasons for using variable speed drive systems are the following:

- Energy saving
- Optimization of the application for position and speed control, e.g., robotic arms.
- Improvement of the dynamic behavior of the device.

When we say electric motion control, we refer to speed, position, and torque control. First, we should decide the type of the loop (open or closed) system, the control technique(digital or analog), and the pulse modulation method. Then, we have to consider the operating characteristics of our motors, maximum speed control range, oscillations in the generated torque, torque to load current ratio, and other characteristics that have to do with the performance of the motor we have chosen or will choose. Finally, we determine the order of magnitude to which we want the cost of the system to belong.

2.4.1.1 Electric Motor Types

Electric motors can be divided into four primary categories:

- First on the list are the DC motors[Figure 2.27] with a collector-grid system for the rotor supply. These motors are powered by rectifying the alternating voltage or using a battery. In this category are also the permanent magnet motors (the winding of the stator excitation has been replaced by a permanent magnet) and universal motors that operate with alternating voltage [33].
- The second category is the alternating current motors [Figure 2.27]. Induction and synchronous winding motors are part of this category. A sine wave signal can power these motors.
- Third, Synchronous alternating current motors. These are either permanent magnets or magnetoresistance motors. The main characteristic of these motors is that they have no rubbing surfaces (heat sinks).
- The last category concerns special purpose motors such as stepper[Figure 2.29] and hysteresis motors[Figure 2.30].

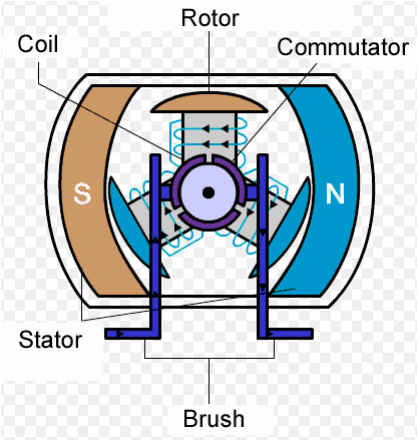


Figure 2.27: DC Motor

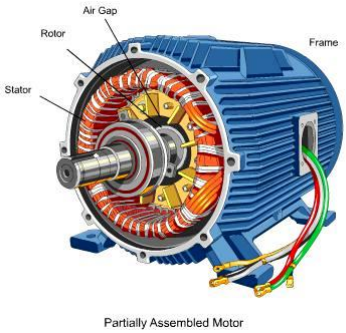


Figure 2.28: AC Motor

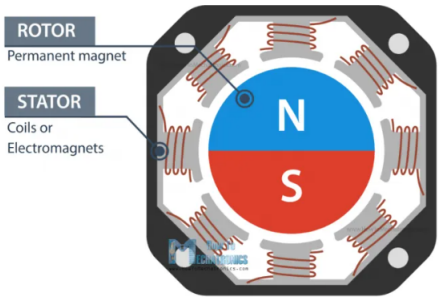


Figure 2.29: Stepper Motor

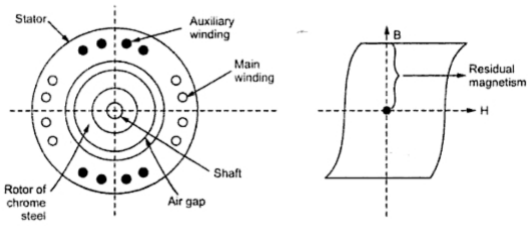


Figure 2.30: Hysteresis Motor

2.4.1.2 Pulse Width Modulation

Generally, the control of DC motors to determine the speed and the rotation is done using the Pulse Width Modulation method. In many applications, simple voltage regulation would cause a significant loss of power in the control circuit, so the Pulse Width [34]. Modulation (PWM) is used in many DC motor control applications. The fast rising and falling edges ensure that the semiconductor power devices are turned on or off as quickly as possible to minimize switching time and associated switching losses[REF]. To control the speed, using the PWM method, we require the frequency/period (2.27) of the signal to be constant and vary the pulse width; in the case of AC motors, we change the frequency too. As we observe in the following drawing [Figure 2.31], of a rectangular pulse, we have a positive pulse for 1/3 of the period, and for the other 2/3, our system is at rest. The energy provided by such a unipolar signal is equal to the area of the surface in one period of the signal. Therefore, if we change the pulse width, the signal's average value and energy provided in a given period. The duty cycle is equal(2.26):

$$Duty\,Cycle(\%) = \frac{Pulse\,Width}{Period} \times 100 \tag{2.26}$$

$$\text{PWM } Frequency = \frac{1}{Period} \tag{2.27}$$

Pulse width modulation is a widely used modulation method with many applications, such as lighting power, PID automatic control systems, class D acoustic frequency power amplifiers, and many others. We can generate the modulated signal with the following methods:

- Analog, using a comparator, we compare the input signal with a sawtooth or triangular signal. When we implement such a circuit, the voltage at the inverting input must not be less or greater than the corresponding minimum and maximum of the triangular signal. Also, the frequency of the triangular signal must be at least twice as large to ensure the Nyquist-Shannon sampling criterion. The output of the comparator will be in the form of PWM, and the voltage of the analog signal will determine the pulse width. If this voltage is higher than the triangular signal, we have a Low state, and where the analog voltage is lower than the triangular signal, we have a High state.
- Digital modulation is widespread due to the ease of controlling the circuit from any digital device such as a PC. This circuit can consist of NAND logic gates, counters, and D-Flip-Flop latches.
- With integrated circuits, we mainly use a 555 timer, which offers us many degrees of freedom in signal processing.
- Finally, we will discuss the microcontroller method in chapter 4 since this is our method.

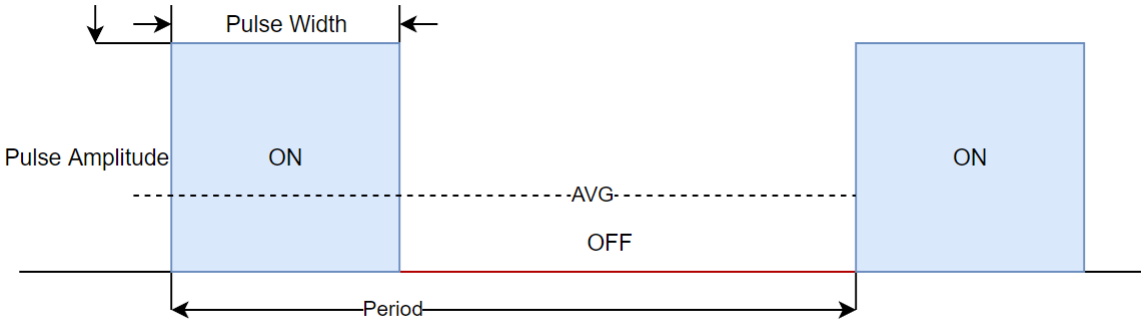


Figure 2.31: Pulse Width Modulation

2.4.1.3 H-Bridge

With the PWM method, we can control the energy offered to our load, i.e., rotation speed. However, to control the motor's direction of rotation, we have to use the H-Bridge circuit [35]. The rotation of the motors is determined by the polarity with which our load is supplied. The simplest form of the H-Bridge circuit includes four switches. As shown in the figure below, the motor rotates in the positive direction by setting switches one and four to ON and having two and three in the OFF state.
Similarly, the motor rotates in the opposite direction by reverting the states. The disadvantage of this circuit is that we cannot achieve digital selection and simultaneous change of states in the switch pairs. The next stage of this circuit is to replace the switches with bipolar compound transistors, which will be used in switch mode controlled by their base. The disadvantage of this circuit is the power loss due to the transistors' heat dissipating, and the controlling circuit is complex too. The most improved design [Figure 2.32] contains MOSFETs.Which are controlled through the gate by voltage, offering a higher switching speed and almost zero power loss.

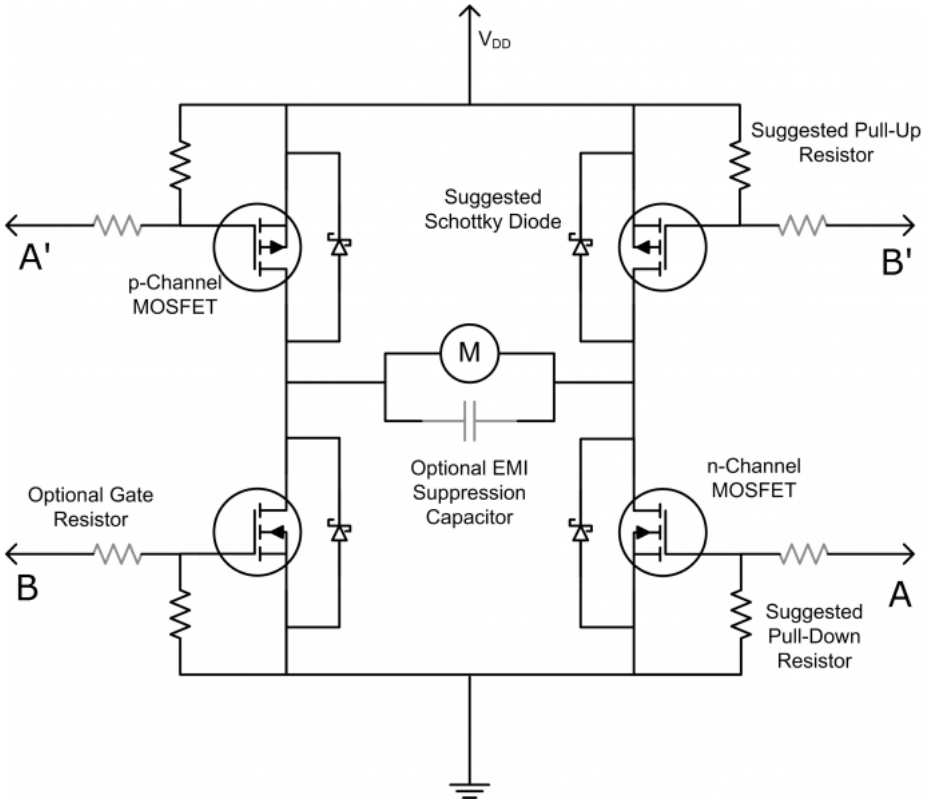


Figure 2.32: H-Bridge with Mosfet

2.4.2 Battery Management System

When we refer to the battery management system (BMS) [36], we mean the circuit that will provide us with safety and reliability when charging lithium batteries. Lithium batteries are very fragile materials and need special attention to ensure that we can perform properly and safely. First, lithium batteries are rechargeable batteries with lithium ions in their cells. These batteries are distributed in specific nominal voltages, 3.6V, 3.7V, 3.7V, 3.8V...4.2V. When we build a system consisting of several battery arrays, we must build the appropriate circuit for charging them. A specific procedure needs to charge these batteries; all the batteries must be at the same voltage before starting the charge; this is mainly the procedure to be followed by the engineer who will place them in the circuit. We follow this procedure because we want each battery to have the same charge cycle. As we will describe in chapter 4, before placing the batteries in the charging circuit, we will make some measurements on the batteries to test their condition. These batteries should be charging around 4.2V; if their voltage has dropped between 2.4 to 3V, they should not charge. Another parameter we have to deal with is temperature. For example, if a product should operate at a minimum of -25°C and the batteries datasheet guides us not to charge below 0 °C. We, as engineers, have to design a circuit that will charge the battery for a specific voltage and temperature range according to the manufacturer datasheet. As we can see in the typical datasheet of LG Li-Ion battery 18650, the operating temperature for charging is from 0°C to 40°C, and the discharging is from -20°C to 60°C. The battery can operate safely at - 20°C but cannot charge at that temperature, so we have to cut it off. Good practice for this is to use a thermistor.
The figure below shows the characteristics of the typical charging of a lithium cell battery. We have the charging current with the dotted line, and with the bold line, we have the cell voltage. The charging stages of the lithium battery are four:

- Tickle Charge restores charge to deeply depleted cells. If the Li-Ion battery cell is below 3 V, the cell charge with a constant current of 0.1C maximum.
- In- Second stage, Constant Current Charge, after the cell voltage has risen above the charge threshold, the charging current increases to perform constant current charging. This current range is between 0.2C to 1.0C.
- Third stage, Constant voltage; after the constant current stage ends, the constant voltage begins after the voltage reaches 4.2V.
- Charge termination, basically we stop charging the battery cell.

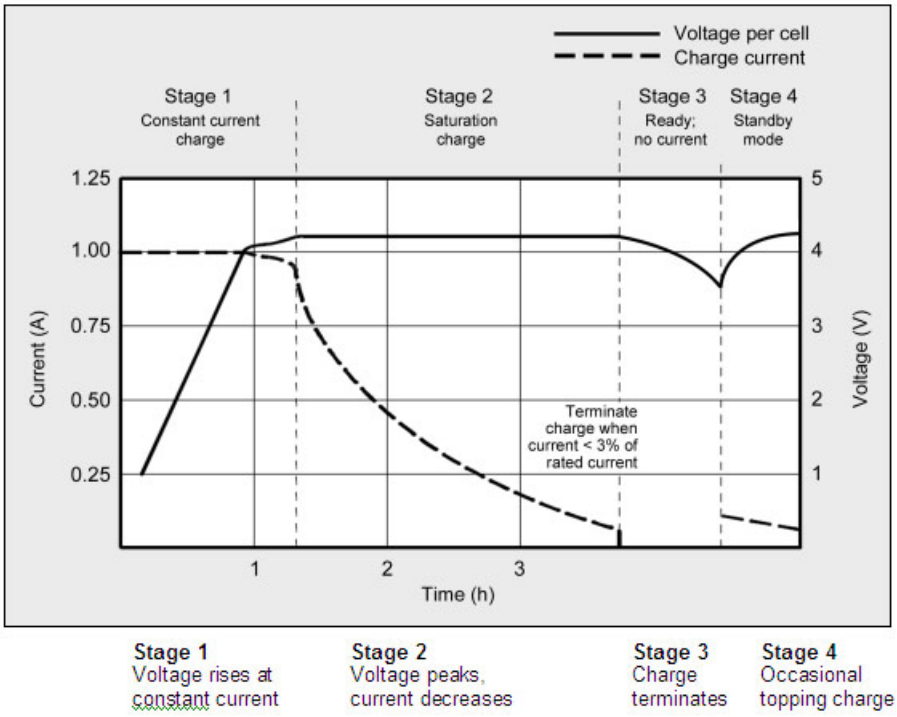


Figure 2.33: Charge states of lion batteries

2.4.3 Ultrasonic Principle

In recent years, ultrasonic technology [37] has developed quite a bit and is part of several new products. Ultrasonic sensors are used in water meters to measure flow, in cars to control the distance from the neighboring vehicle, in water level control in water tanks, in many medical applications, and in several industrial processes concerning quality management. Starting in 1880, the Curie brothers Perrie and Paul -Jacques discovered the piezoelectric effect (it came from the Greek word ”πιέζω,” which means squeeze something). While they investigated crystals like tourmaline, quartz, topaz, cane sugar, and Rochelle salt, they realized that those crystals' clouds generated electrical polarization under mechanical stress. Paul Langevin developed piezoelectric materials, which could generate and receive high-frequency mechanical vibrations. The first time Ultrasound was introduced in the world was in World War I, when the navy used that technology to search for enemy submarines. In 1920 Langevin discovered that high-power Ultrasound could generate heat in osseous tissues and disrupt animal tissues. In 1950 ultrasound was used to treat patients with Meniere disease, Parkison disease, and rheumatic arthritis patients. In 1976 we had the first commercial ultrasound machines coupled with Doppler measurements; every decade after that, Ultrasound technology took place for medical purposes more and more.
Ultrasound is high-frequency sound above the limits of frequencies the human ear can hear. The range of acoustic frequencies for the human ear is from 20Hz to 20kHz, so Ultrasound is for frequencies above 20kHz.Some animals, like bats and dolphins, can hear higher frequencies. Generally, on medical devices, we use a spectrum from 2 to 15Mhz. How can we produce Ultrasound mechanically?
As mentioned above, we use the piezoelectric effect [Figure 2.34] to produce Ultrasound. According to the piezoelectric effect, we can produce an electric charge when we apply mechanical force to a piezoelectric material. So, if we apply the same electric charge to these materials, we can produce mechanical vibrations. There are many natural and human-made materials, like quartz crystals and ceramics. The basic technic we use to create transducers is to stack piezoelectric material in different layers so we can have more efficiency.

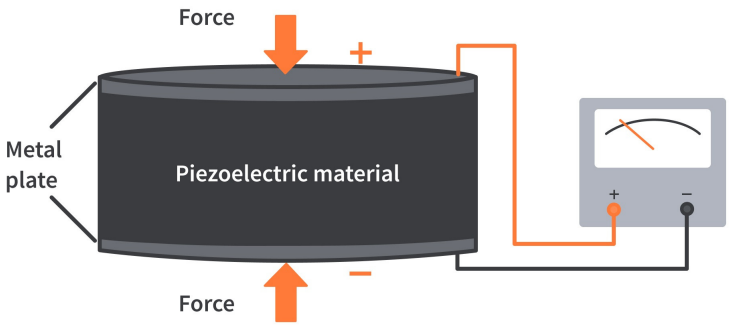


Figure 2.34: Piezoelectric Effect

The basic terminology we use in Ultrasound technology :

- **Period(T)** is the time for the sound wave to complete one cycle in **µs**.
- **Wavelength (λ)** is the length of space over one cycle.
- **Frequency (f)**is the number of cycles repeated per second and measured in hertz.
- **Acoustic velocity** (c_m) is the speed at which the wave sound travels through a medium (2.28).
- As (c) is the **speed of sound**.

$$c_m = f \cdot \lambda_m \tag{2.28}$$

The Doppler effect took the name of Austrian physicist Johan Christina Doppler[REF]. Doppler effect describes the change in the frequency or wavelength of a sound wave, which occurs across the relative motion between the source and receiver of the sound. The following equation (2.29) can describe those changes mathematically. In simple words, the Ultrasonic principle [Figure 2.35] is that if we send a pulse wave to an object, we can calculate the distance from this object but comparing the pulse wave we sent with the one we receive. In this paper, we will use an ultrasonic sensor to detect objects in front of our vehicle; in Chapter 4, we will discuss the electronic part of the specific sensor.

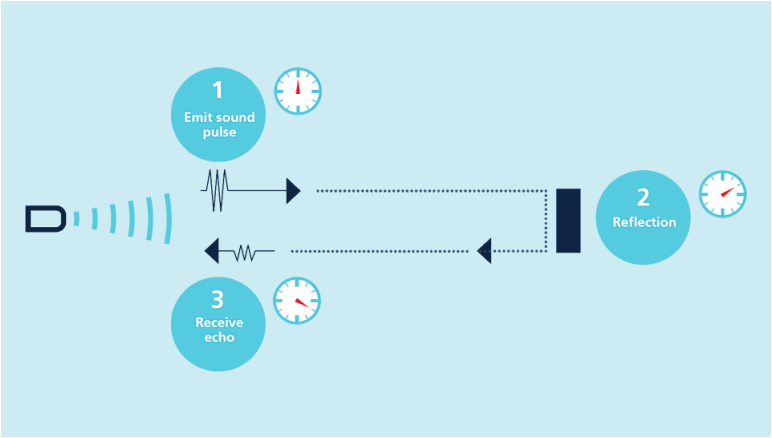


Figure 2.35: Ultrasonic Principle

$$f = \frac{c \pm U_r}{c \pm U_s} \times f_o \tag{2.29}$$

- U_r the speed of receiver relative to the medium.
- U_s the speed of source relative to the medium.

2.5 Summarize

In this chapter of our thesis, we presented all the theoretical background needed to enforce our work. First, we mentioned the essential characteristics of a digital image. Then the main characteristics of the images we will use for our database, i.e., the main elements of traffic signs. We worked out the techniques we will need to process these images to optimize our model. Then we made an extensive reference to neural networks, as they are not part of the undergraduate curriculum, so we had to demonstrate in detail the theoretical background we had to learn. The immediate next part of the chapter touched on the part of the embedded system, to be precise, the communication protocols we will use, and the structure we will follow for the code using a real-time operating system (FreeRTOS). The last part of this chapter is the electronics, which involves the charging part (Battery Management System) of our batteries, the ultrasonic sensor working principle, and the control circuit (H-Bridge) for the motors.

3 Problem Statement

3.1 Introduction

In this aspect of our work, we will set out the main issues of the problem we face. We will use a top-down structure, i.e., starting from the image recognition part and ending with the material part. The purpose of this section is to set our goals in principle. Based on these objectives/problems, we will define the whole path of our thinking for implementing our system. As we mentioned in previous chapters, the subject of our thesis is the development of an advanced driver assistance system. The main topic is the development of a convolutional neural network model for image recognition in seven traffic signs; this, as part of our work, is accompanied by subproblems for which we had an image; the same applies to the remaining aspects of the work, such as the part of the embedded, electronic and mechanical system.

3.2 Database

One of the most critical problems we will have to face is creating a data structure: images of Traffic signs. Because when we train a neural network model to identify objects, we must have a base of images with a specific quality, a variety of samples, and incredibly different samples, to achieve better performance and credibility in the results. If, for example, each class of signs has 1000 images identical to each other, with no difference in how they are taken, then our model will always be overfitting. Overfitting means we will build a model that can only recognize the specific images we have trained our model. Therefore we need to build a base with as much uniqueness in each image as possible. We can deal with this issue by collecting images from real Traffic signs; it would ensure the uniqueness of the images; however, we would have a problem with the work time as we would have to capture dozens of different images from different points on the roads.

3.3 Computing Resources

Another essential parameter we will have to solve is the necessary resources to implement the model. In the broadest part of the work, we will not need a computer or a computing unit with expensive costs and tremendous capabilities. It is ubiquitous in Machine Learning that engineers use graphics cards as they consist of multiple cores and are suitable for parallel execution of multiple computations at enormous speed. This strategy is followed quite a bit, as it speeds up the learning time of the model, which is a major stumbling block in its development. When we start building a model, we are asked to test several different parameters to achieve the model's optimal training score. Such parameters are the quantity and quality of our data images that constitute the system's input, the number of convolutional layers-the number of filtering stages, and the techniques for improving the model dropout and flatten, which in turn have several parameterizations. This process is costly when implementing such a model as it will need to be run several times until we achieve the desired results. For example, if each run of the model takes at best 30-40 minutes, and each time we run the model, the results do not improve, the number of iterations increases dramatically. It is pretty challenging to develop a model for 10-15 training cycles, especially for someone who does not have the necessary experience. Therefore, we will also choose to use a graphics card, but as mentioned in the next chapter, we will avoid buying a computer.

3.4 Overfitting and Underfitting

Two crucial factors that can degrade the construction of a Machine Learning model are Overfitting and Underfitting. When implementing a model as a central plan, we have to generalize well. When we refer to generalization, we mean how much accuracy and stability the output of our model can satisfy with different inputs concerning the data we used as input in the training part. It is a general principle in engineering that we judge a system based on the stability of the output concerning different inputs and how much we maintain the repeatability of the output. Overfitting and underfitting are two concepts that determine whether our model generalizes well. Overfitting occurs when our model tries to cover all the data points without providing the appropriate information, resulting in storing noise and acting on inaccurate information. This could drop in the performance and accuracy of the model. The model with overfitting is characterized by low bias and high variance. The main reason over-adaptation is developing is that we train the model more than we should. Underfitting usually has to do with the technique we use to avoid overfitting, as we stop training too early, so the model has not been trained complete, and it did not store enough information to be considered accurate. An underfitting model is characterized by high bias and low variance.

3.5 Live Feed

When we train the model, we will use ready-made images with specific angles of exposure and brightness. All the data we will use during the model training will generally have a consistent quality. The same applies to the testing part to see the prediction results; this concerns the phase before we install the system on our prototype vehicle. When we reach the point where we have a vehicle that can move and a capable model based on the results of the previous process, a new, more complex parameter will arise. The speed at which we move and the fact that we have a moving camera that is no longer fixed will cause us many problems in capturing images. First of all, we will not be able to focus on the traffic signs correctly, and assuming we are focusing on signs with the same aspect ratio of the vehicle, we will certainly have problems in focusing and image capture; likely, we will not be able to detect the traffic sign at all so that nothing will work. On the other hand, even if we have a sufficient size in the signs, there is an excellent chance that if the camera we use does not have an appropriate speed of acquisition for the speed that the vehicle can move, we will have a result the effect of noise in the received images, which will make it difficult to operate our model. Also, we have sudden changes in brightness and focus angles along with the movement.

3.6 Response Time

A system that concerns the driver's assistance has to work under specific time limits. Generally, when dealing with a system designed to improve driving safety, we have two main criteria: response time and the accuracy of the decision. In the previous sections of this chapter, we have discussed the quality of the results of our model; now, we will bring up the response time. It is probably the most challenging part of implementing such a project. There are too many parameters in the response part from different parts of the system, and it depends on the computer executing the algorithm to recognize the plates. It depends on the communication between this computer unit and the control unit, how the control unit

handles the various interrupts it receives from the system, how it communicates with the electronic units, and what response times our electronic units have from the bridge for the control of the motors to the motors themselves.

3.7 Hardware

The last parameter is the construction of our vehicle and counteracts all those problems that have to do with the management of the chassis, to place the processing unit, the electronics, batteries, and sensors.We are not only concerned with placing all these objects one on top of the other; we want them to be placed in a place that makes sense and improves the behavior of our system.Also, any construction we do for any part of the vehicle has to be specific, robust, and protect our inputs. It will be unfortunate if we need to change critical components of our system because they were damaged during our tests.

3.8 Summarize

In this chapter, we have presented the critical points we will have to face during the implementation of our portfolio. We began by referring to the problem of constructing a reliable and extensive database of images. We mentioned the potential problem of needing unique computing resources for the training part of the model. We also discussed two essential concepts in machine learning, overfitting, and underfitting, which are related to each other, mainly by when we choose to stop training. Finally, we focused on the problems we will face when our model is applied to a moving vehicle and what effects this can have on data acquisition. We also mentioned how important it is for a system like this to have the smallest possible response time and how we will build our vehicle to ensure that our sensors work correctly.

4 Our Approach

4.1 Introduction

In this chapter, we will present, step by step, our method for implementing our system. We will give a detailed description of how we created our base of images if we used any automation to handle a large number of images, and what is the structure of the code we used for this piece. Then we will mention the techniques we used to preprocess the inputs before starting the training. We will not quote any lines of code. However, we will present the structure of the thought for each piece of code that we will analyze using block diagrams, as we believe it is more efficient for illustrating our thinking. The next, part of the chapter, is about the concept of training and what steps we follow to arrive at the desired results; we will show the basic steps for using Google Colab. At this point, we will also analyze the structure of our neural network, how many layers it consists of, what kind of filtering we do on each one, and the sequence we use. Immediately next, the next part of the chapter is the management of the computing unit that will run the prediction, i.e., the Raspberry. We will present the steps to set up this computer, from the beginning to the end, since its correct management is crucial. Furthermore, we will refer to the code we developed to make decisions and communicate with the electronics control unit. This unit is a microcontroller of the F4 family from the F family, specifically the F401RE; we will mention the essential information we need to use the programs to develop the code and the flow of the code itself. At the end of the chapter, we will make a solvent presentation of our electronics and the mechanical parts we built.

4.2 Image Processing

In this section, we will present the steps we followed for preprocessing the images. We have already mentioned that in this thesis, we will use the programming language python for the CNN model, and the preprocessing of the images will be done with the OpenCV library. A few words about the library, its name comes from the Open Source Computer Vision Library; as we understand, it is an open source library, which makes it easily accessible. It has a large community means we can find support for every detail. On the official site, there is a detailed guide starting from the installation process of python and the library, and in the following steps, we could run the first code on our computer. This library was developed by Intel and was first released in June 2000; its structure is in C/C++ language. We chose this library because it was beginner-friendly and we could train ourselves efficiently. Lets steak to the implementation part , in order to perfom our first test test for the prepoccesing of images we will use Google Colab. The training procedure of a CNN model could last from 20 -30 minutes to hours; it is related to the complexity of our model and the range of the dataset; if we use a model with 2000 images, it will take some minutes if we use a dataset of 200000 images it will take some hours. Therefore, having mentioned the essential characteristics of traffic signs, we chose not to use image color in our model.
The first filtering stage [Figure 4.1] of our images is the conversion of the images into grayscale. This simplifies the learning process, as we do not need to deal with three different color components. It is a widespread technique in the science of Computer Vision. As we have explained, traffic signs have three main characteristics: shape, color frames, and symbols - numbers and words. Removing the color frames does not affect the performance of our model. Next stage, the photos will have to go through an equalization filter histogram to improve the contrast of the images and widen the pixel distribution.

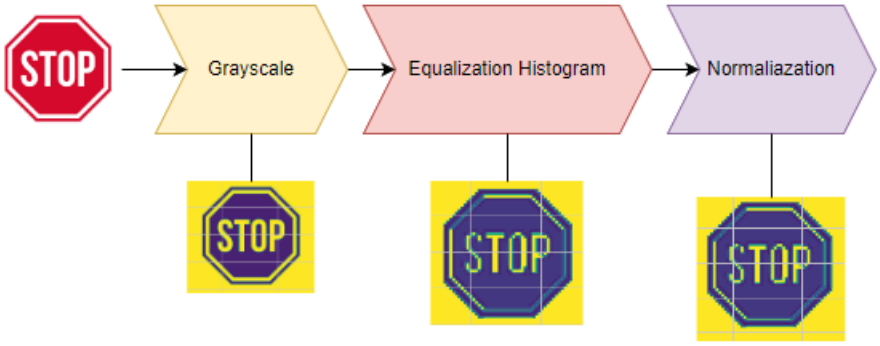
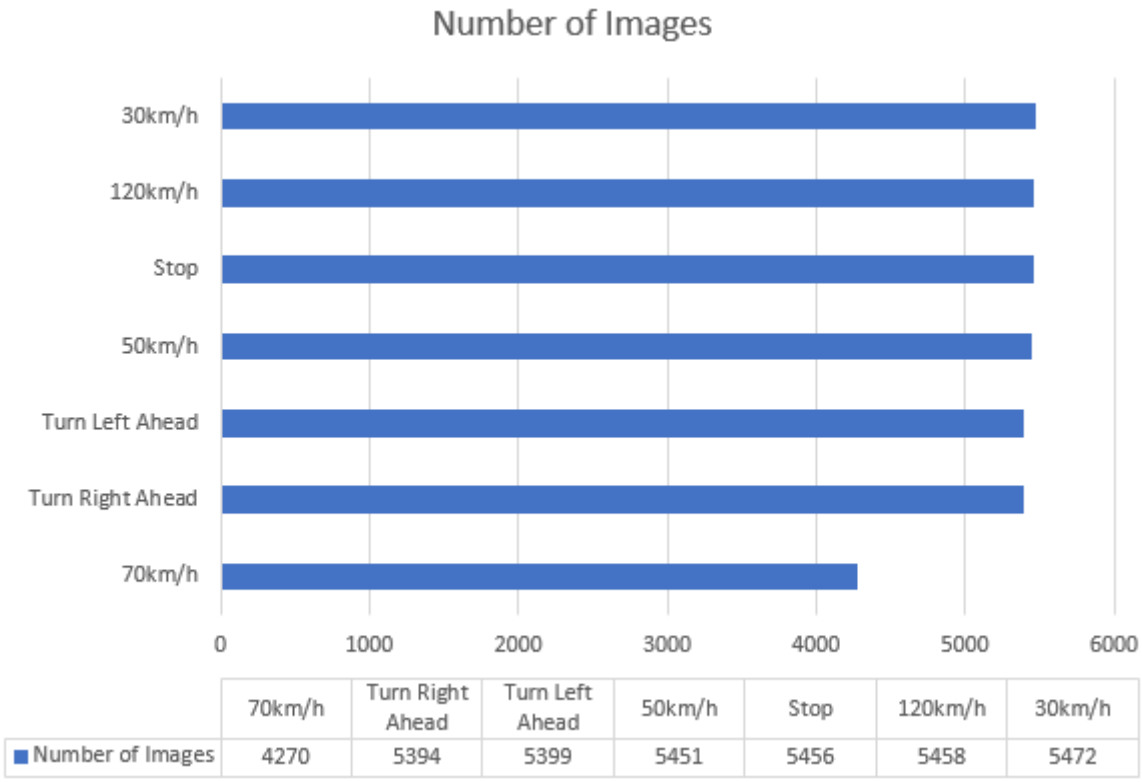


Figure 4.1: Image Processing for Training

Finally, we have to normalize the images by dividing by 255; this technique allows us to filter possible noise present in the image from the different angles that reflect light. We will follow the same procedure with the live feed from our webcam, plus we need to resize the frames to have the same type of input as our model.

4.3 Dataset

To implement an efficient model, a crucial piece of the puzzle is the database we will use. Usually, for this kind of project, there are ready-made bases. Our database will consist of 7 classes, Stop, 30km, 50km, 70km, 70km, 120km, Turn Right Ahead, and Turn Left Ahead, with almost 35000 images (5000 per class). The dataset we took as a prototype/example provided 34 classes with an expanding amount of images per class. However, our work aims not to build a traffic sign recognition model but an advanced driver assistant system. In order to build our dataset [Figure 4.2], we created a python script [Figure 4.3], and using a webcam, we took almost every image to automate. We used batches of images and tried to use different angles and light. Then we use a second python script[Figure 4.4] just for resizing images to the preferable size of 32X32 pixels. Finally, we had to upload our dataset on Google Drive so we could have access from the Google Colab.



	70km/h	Turn Right Ahead	Turn Left Ahead	50km/h	Stop	120km/h	30km/h
Number of Images	4270	5394	5399	5451	5456	5458	5472

Figure 4.2: Dataset

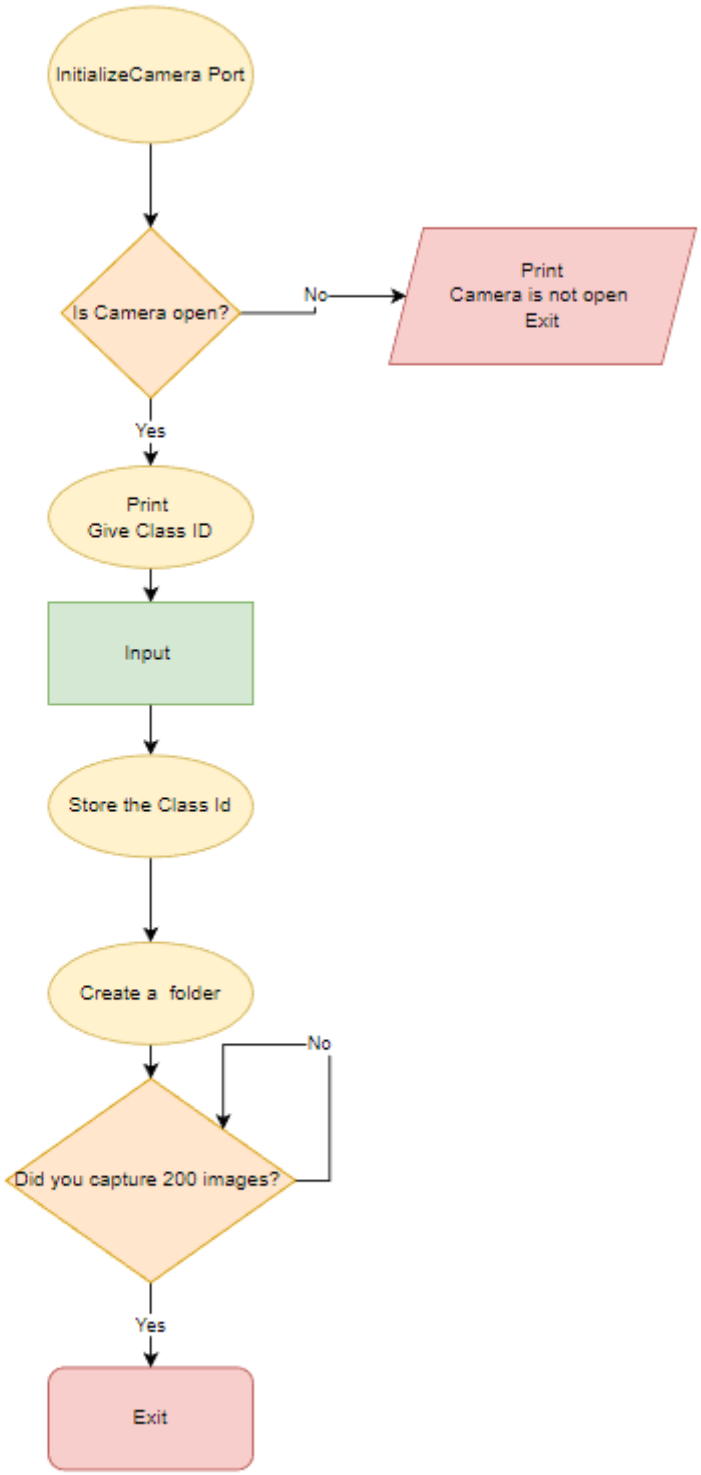


Figure 4.3: Flowchart of Auto-image capture

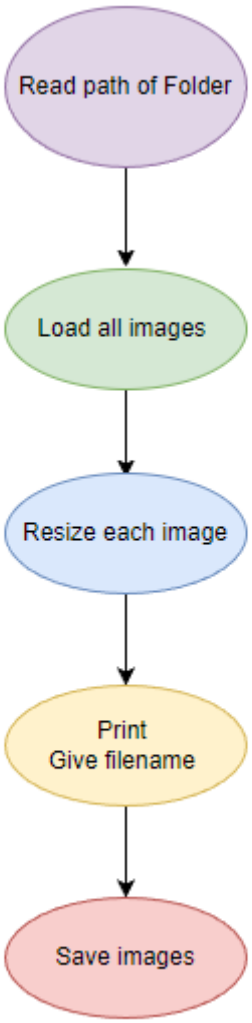


Figure 4.4: Flowchart of Auto-image resizing

4.4 CNN Model

In this section, we will present our model, how we ran the different tests, and what was our train of thought to implement an efficient model. However, we will start with some references on how to use google Colab. When implementing CNN models, we need enough resources to run the training part of the model as fast as possible. It is known that such models have to run on computers with a powerful graphics card. Since the whole process runs on the cores of the card, we felt that the cost of running such a machine would not serve the purpose of the work, as we would be diverting an amount of money that would not be returned to us. Therefore, we used Google Colab, which essentially provides us with servers that are machines that meet the specifications to run our models. We saw a big difference in the time and mode of model implementation compared to our initial expectations of running the models on our personal computer's processor.Google Collab is a block-based IDE (Integrated Development Environment), meaning we can run parts of our code separately.

The four basic steps to using Google Colab:

- First, we go to the website Google Colab.
- We choose to create a new file.
- We use our email to log in.
- From the main menu we select Runtime →Change Runtime →GPU
- *** Plus step, to use our google drive, we should add the first line of code as seen in the following [Figure 4.5].

```
from google.colab import drive
drive.mount('/content/drive')
```

Figure 4.5: Mount Google Drive

The following [Figure 4.6] presents our working flow for each time we run our model. We had to rerun our training many times until we reached our accuracy terms. We start with 10000 images as a dataset and reach 35000. In the first step, we had to upload the dataset to the project each time for every change we did to the dataset. We had to print the amount of our dataset each time to ensure we had no mistakes. Then we had to upload the labels CSV file so we could represent our dataset with labels. In the next step, in C.N.N, it is crucial to split our data between images that are going to use for the training processes, images for validation, and images for testing after our model is complete. From 34811 images, we used 6963 for testing, 22278 for training, and 5570 for validation. After splitting our data, we need to preprocess our training images, as discussed in the previous section, and reshape our images sbefore we insert them into the model so we can have a correct batch of images each time. To increase the amount of data, we use the augmented method. We rotate, zoom scale, etc., our images and make different changes to them so we can have a massive amount of different inputs. The other method we use is One Hot encoding; because our data have complexity, we need to simplify them binary. Finally, we can run our model with the original and augmented data.

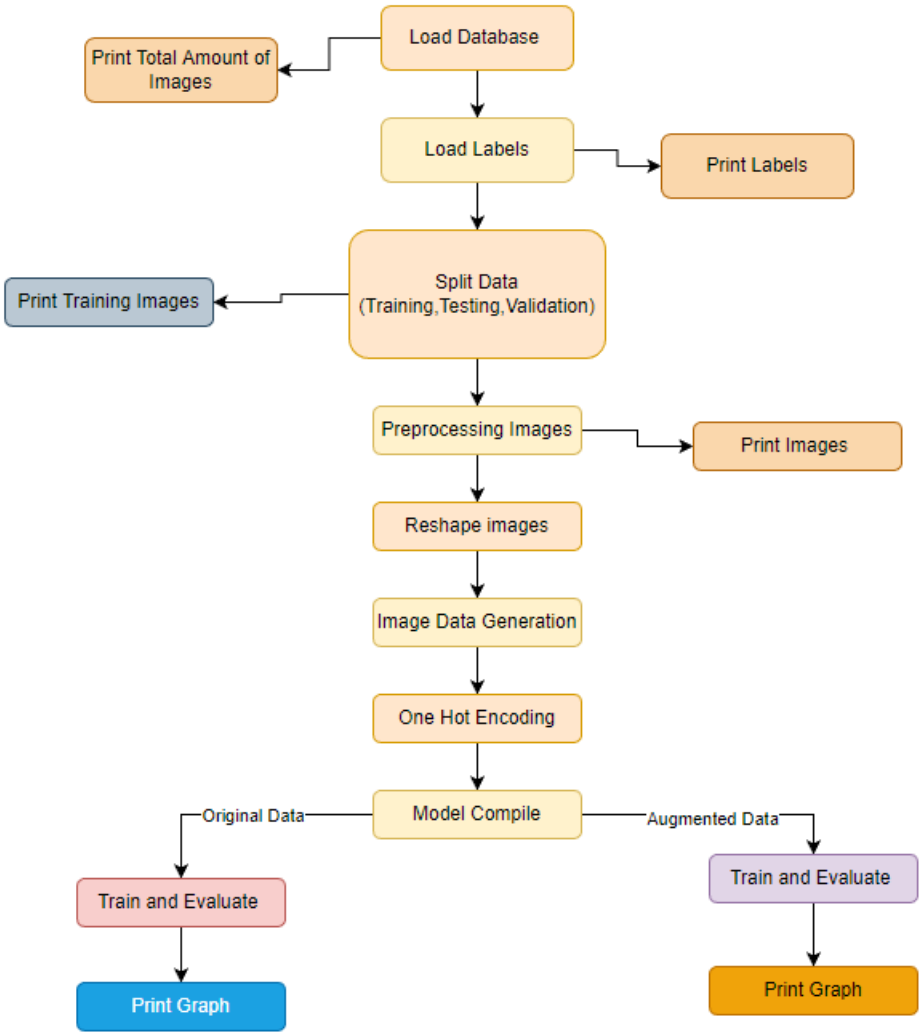


Figure 4.6: Model Training Workflow

Below is the structure [Figure 4.7] of our model[Figure 4.8, 4.9]. Initially, our input which is in the form of a matrix consisting of pixels with orders (32,32,32), is passed through the first convolutional layer, which is a 3X3 kernel with ReLu (Rectified Linear Unit) enabled dimensions and with output dimensions (30,30,32), then it is re-entered in a new layer with the same parameters and with final orders (28,28,32). Then we use the Maxpolling technique; in essence, we identify the most significant values in each dimension of the 2DMatrix and model it to get an output with the most significant values. Again this output is passed through two successive Layers with the same parameters as the previous ones but with different final dimensions. Finally, we reintroduce the Maxpolling technique and then convert our pins to one dimension (Flatten technique).

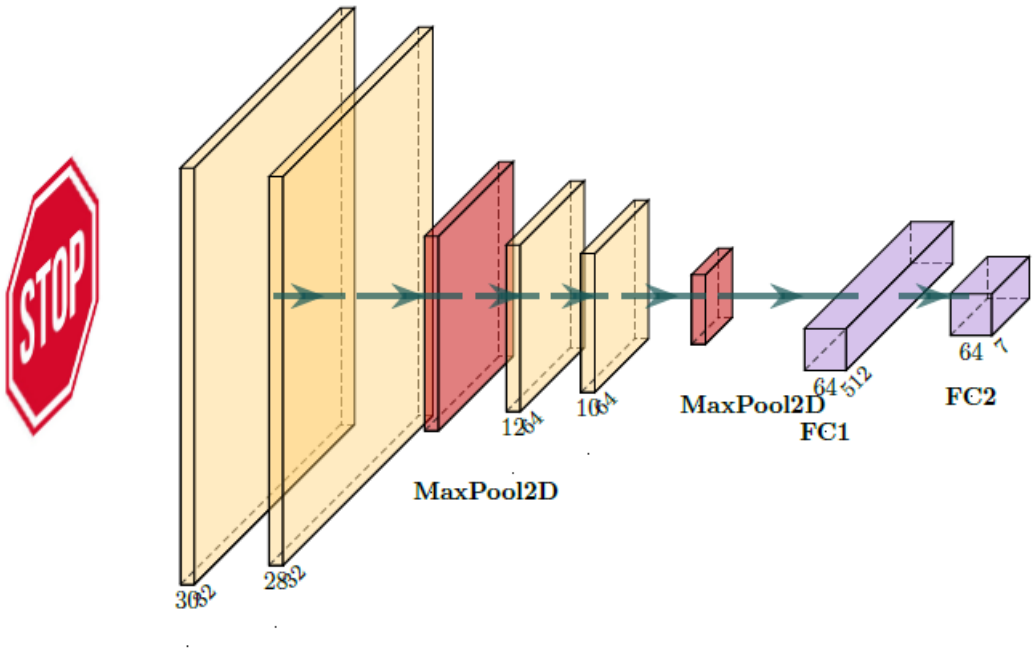


Figure 4.7: CNN architecture layers

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 30, 30, 32)        320

 conv2d_1 (Conv2D)           (None, 28, 28, 32)        9248

 max_pooling2d (MaxPooling2D  (None, 14, 14, 32)       0
 )

 conv2d_2 (Conv2D)           (None, 12, 12, 64)        18496

 conv2d_3 (Conv2D)           (None, 10, 10, 64)        36928

 max_pooling2d_1 (MaxPooling  (None, 5, 5, 64)         0
 2D)

 flatten (Flatten)           (None, 1600)              0

 dense (Dense)               (None, 512)               819712

 dense_1 (Dense)             (None, 7)                 3591

=================================================================
Total params: 888,295
Trainable params: 888,295
Non-trainable params: 0
```

Figure 4.8: CNN model

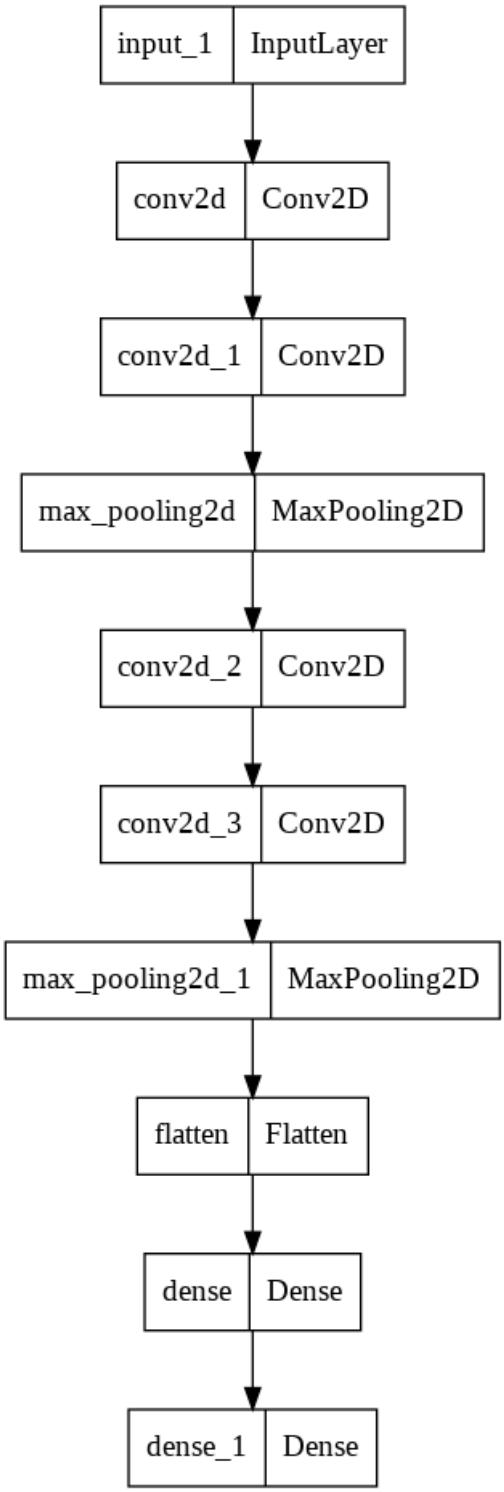


Figure 4.9: CNN model Flowchart

4.5 Embedded System

4.5.1 Raspberry Pi 4

After we complete our model for finding traffic signs, the next step is to choose a small computer that will be able to run our model so that we can detect the signs with the web-camera and send the commands to the microcontroller, which will handle the basic operations of the vehicle. We have chosen a Raspberry Pi 4 [Figure 4.10] with 2 GB of RAM as the primary data processing unit, which will take all the decisions. It will record from the camera, run our model, and then, with the instructions given for each sign, it will have to send the resulting data serial to the microcontroller.

We will describe the steps for preparing the Raspberry to be fully functional for our system. For this purpose, we need a power supply for Raspberry (recommended the original 5V / 3A) and a micro SD memory card. We observed the following online guide for selecting the right memory card, which the raspberry community has implemented. These two are very critical for the appropriate functioning of our computer. First, we need to flash our operating system (Raspberry Pi Os Lite 32bit) on the memory card; it requires software to write the image to memory. We chose to utilize the official software from Raspberry, the pi imager. No special knowledge was needed for this procedure; we had to install the pi imager on our desktop computer and start it. After inserting the memory into our computer, we first select the operating system through the pi imager; then, we select the specific memory that will be transferred. Before we started the data transfer procedure, we had to set up the computer's details, such as the connection to the local wifi, the user details, and the option to activate the secure shell, so that we could control the Raspberry remotely. Once the process of transferring the data to the microSD was complete, we inserted the memory into the Raspberry and powered it up.

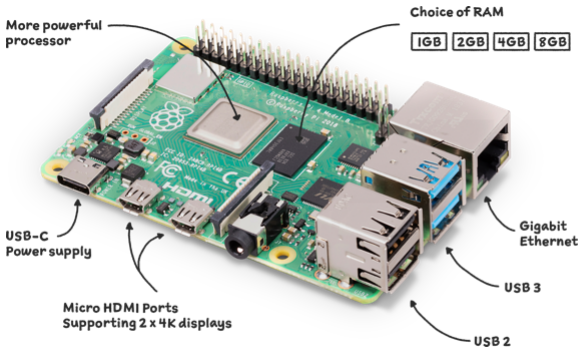


Figure 4.10: Raspberry Pi 4

4.5.1.1 Setup Raspberry environment

In order to be able to control the raspberry remotely, we used the secure-shell feature. This paragraph will present the basic process of connecting our main computer to the mini pc.We used a terminal console, specifically the Ubuntu console.

- We use the below command to start the shh section [Figure 4.11].
  $ ssh $username@192.xxx.xx.x$
- After connection establish Figure 4.12
- Update to latest version of our packages:
  $ sudo apt-get update
  $ sudo apt-get upgrade
- We need to install Opencv,so we could run our Model.
  $ pip install opencv-python
- To checkout python version and pip packages
  $ python –version
  $ python -m pip freeze
- We use python virtual environment so we could not break our set up, we have the basic commands below
  $ pip install virtualenv
  $ python -m pip freeze
  $ pip install -r requirements.txt
  $ python<version> -m venv <virtual-environment-name>
  $ source env/bin/activate
  $ deactivate

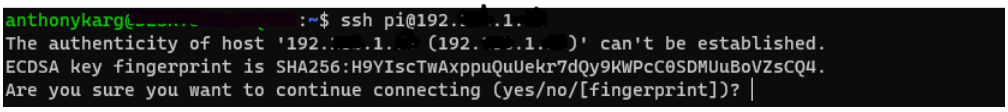


Figure 4.11: Connect SSH

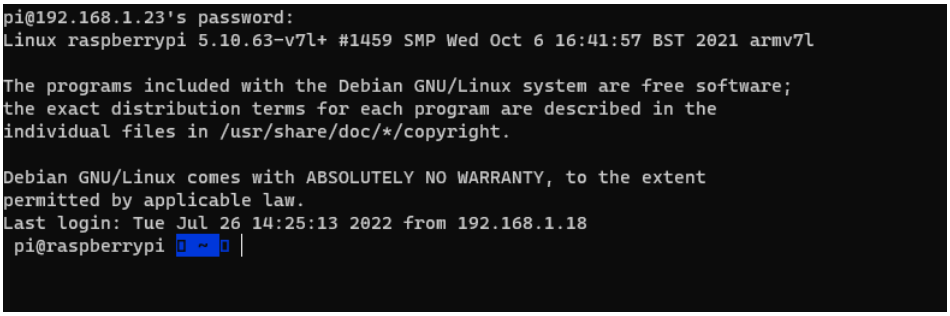


Figure 4.12: Raspberry SSH

4.5.2 Python Script

In this subsection, we describe the general idea of the python script [Figure 4.13] we use to send the data from the prediction of our model to the STM32 microcontroller. We start by executing the python script. The program will ask us to give input like Forward, Backward, Speed up or down, Turn Left or Right and Stop. In parallel with our keyboard input, the prediction model starts working, uses the webcam as input, and then gives serial communication instructions to the microcontroller.

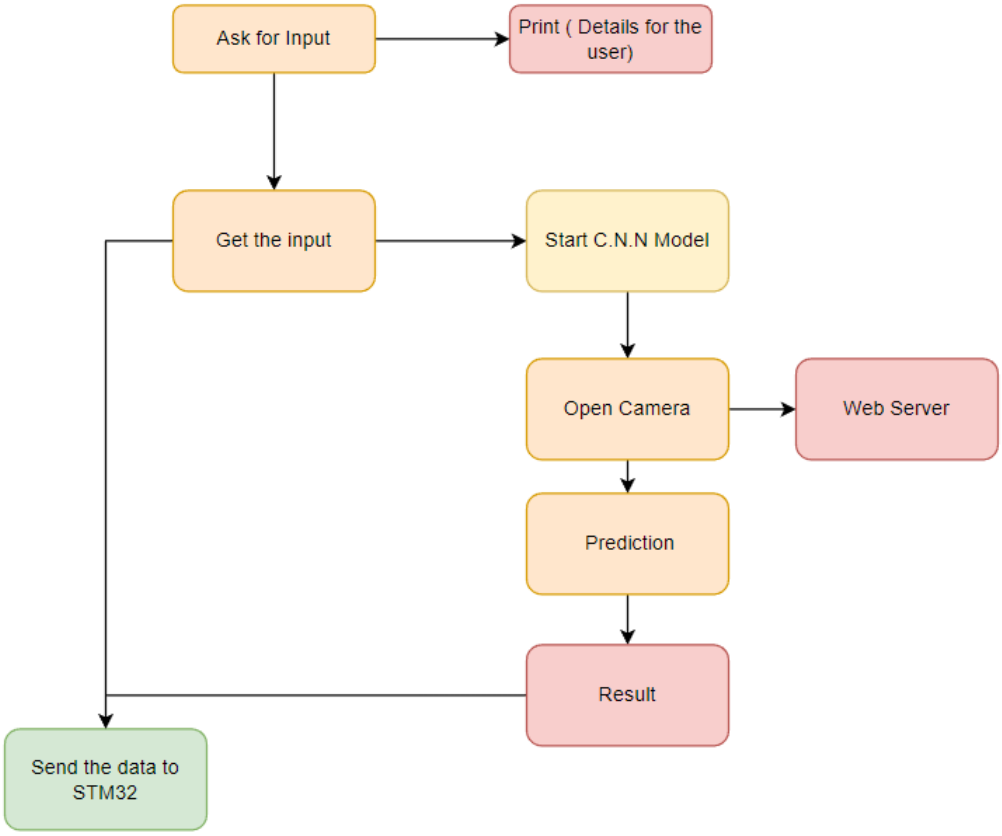


Figure 4.13: Python Scipt for Serial Communication

4.5.3 Microcontroller

The final piece in this structure, from top to bottom, is the programming piece of the microcontroller to control our vehicle. We chose to use the development board from ST, NUCLEO-F401RE Board [Figure 4.14], which has the STM32F401RE. We will not focus on the features of this particular microcontroller, we will mention the basic steps for using the software (Cube Mx, CubeIde) from ST, and at the end, we will provide the flow chart of our code. We had the option to use different compilers for our code, but we chose to use the programs from ST as they were user-friendly.

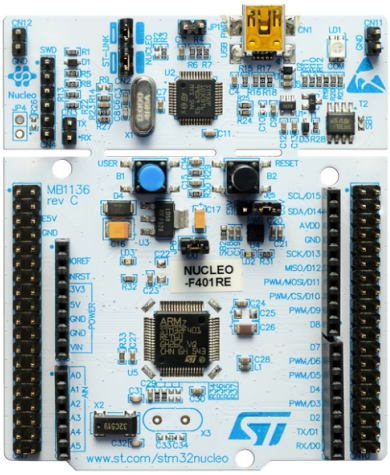


Figure 4.14: NUCLEO-F401RE

The installation of these programs is relatively easy and fast. These programs are distributed free of charge together with the libraries we use. The download of these programs can be done directly from the official page of ST Cube IDE, Cube MX. After we have explained the installation, we will show the critical points for using this software. We opened the Cube IDE and selected the workspace where we will create our projects. We could create our project from Cube IDE or Cube MX. Creating a project will lead us from one program to the other. After launching the IDE [Figure 4.15], we selected to create a new project; this took us some minutes to download all the essential updates. The program moved us to the Cube MX, where we had to select our Nucleo Board, which we were able to find by search as Nucleo F401RE.

1. We select Nucleo F401RE [Figure 4.16].
2. We give our project name as thesis.
3. We press finished so, we could create our project.
4. Then we had to initialize our MCU.

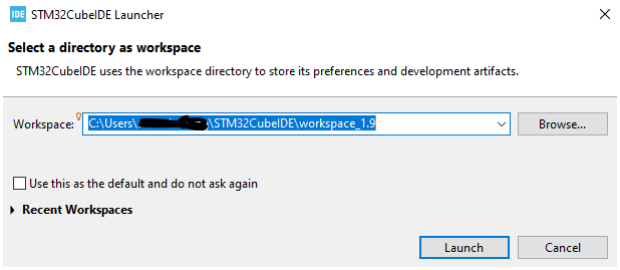


Figure 4.15: Launch Cube IDE

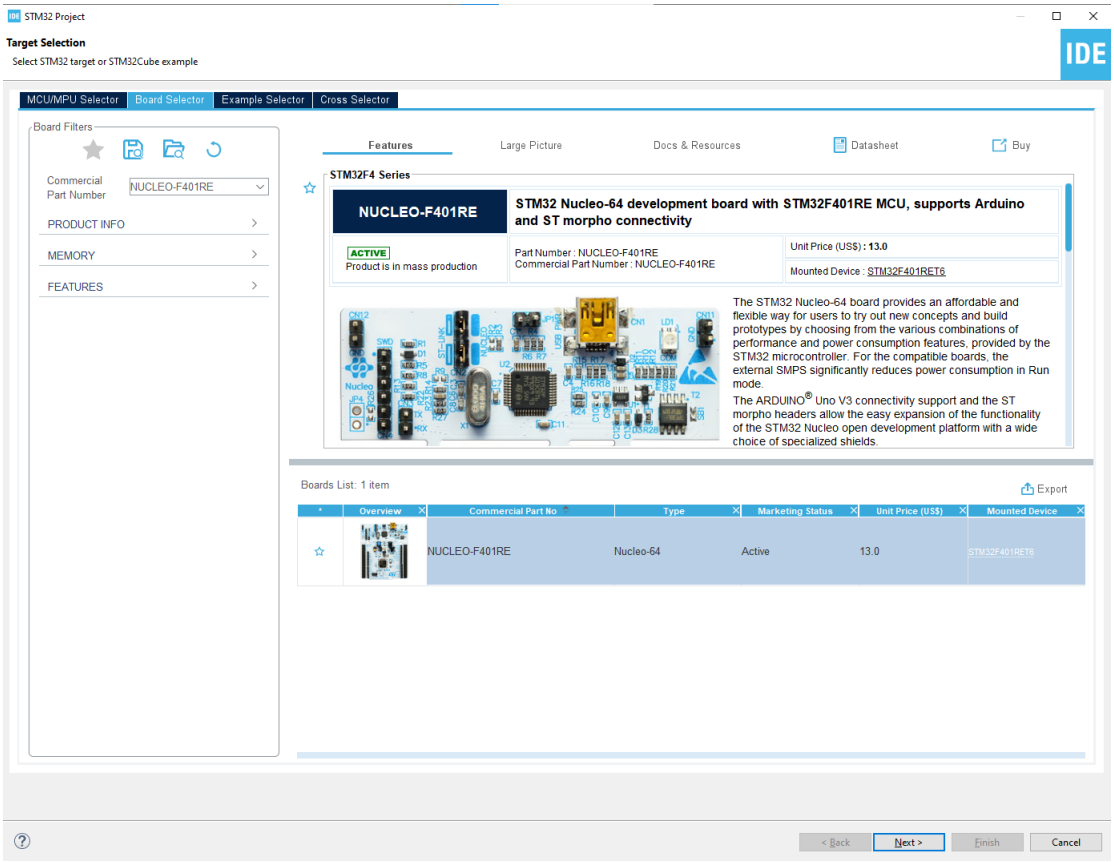


Figure 4.16: Selection of Board

We use Cube Mx to initialize our MCU with a graphical tool. Cube MX represents the MCU pinout [Figure 4.18]. We could select any Pin of the MCU and set it as input or output and give a name Table 4.1. Through the GUI of CUBE MX, we can set the communication protocols we want to use and initialize the timers, the Freertos, the tasks, and the priority each one needs. Below we will analyze these settings. The first parameter we want to be set is the Reset and Clock Control (RCC Mode) regarding internal or external crystal oscillator. We have enabled the Crystal /Ceramic resonator for the Hight Speed Clock HSE. The figure below shows the system view from Cube MX, which presents the protocols and interfaces we are using. An essential parameter is a timebase, and we enable the TIM1. We will implement an H-Bridge circuit to control our motors' speed and direction of rotaion. So we need to produce PWM signals from the F401RE by setting channels 1 and 2 of TIM3 as PWM Generation plus to make some calculations (4.1) so we have a 10kHz Frequency.

$$F = \frac{Clock}{Prescaler \cdot Counter} = \frac{84000000}{84 \cdot 100} = 10kHz \tag{4.1}$$

4.5.3.1 Clock Configuration

This paragraph will describe how we set the microcontroller's clock. The F401PE has a maximum operating frequency of 84MHZ, and we want it to operate at maximum frequency. For this purpose, we have chosen to use the High-Speed Internal (HSI) crystal, which operates at 16MHZ.To set our microcontroller at 84MHZ, we need to use the internal phase-locked loop (PLL Source) circuit(4.2). In this way, we ensure the stability of the frequency, then divide this frequency by 16. Therefore we have a 1MHZ mark which we enter again in a PLL circuit where we multiply this frequency by 336 and divide it by 4 . Finally, we get the frequency of 84MHZ; for this procedure, we consult the microcontroller's datasheet.We a screenshot of our configuration at the appendix.

$$F_{SYS} = \frac{F_{HSI}}{M \cdot P} \cdot N = \frac{16000000}{16 \cdot 4} \cdot 336 = 84MHZ \tag{4.2}$$

Figure 4.17: TIM5

4.5.3.2 Timers

Next, we intend to use Freertos; we need to define the time base of the microcontroller; this is very important, as it will play an essential role in how the scheduler will work and manage the different tasks. So, we choose to use a simple timer, TIM5 [Figure4.17] which does not need a special initialization.We need a second Timer,TIM1, as input capture mode, so we can measure the rising and falling edge from the Echo Pin of the ultrasonic sensor to calculate the distance from the object.In order to do that, we will change the polarity of the edge so we can identify when a rising edge or falling edge is coming, then, we will store the time that each edge was received, and we will use the following equations (4.3,4.4) to calculate the distance.

$$Difference = (65535 - rising\_edge\_time) + falling\_edge\_time \tag{4.3}$$

$$Distance = Difference \cdot \frac{0.343}{2} \tag{4.4}$$

4.5.3.3 DMA

As we have mentioned, we will use the technique of Direct Memory Access to transfer the data received from the serial interface to the internal flash of the microcontroller; we chose this method instead of interrupting and polling as it allows us not to bother the microcontroller with the process of storing data, so we will not mess up any task. To activate this technique, we used CubeMX, from the Core System; we selected an add-on and USART2_RX in Circular mode.

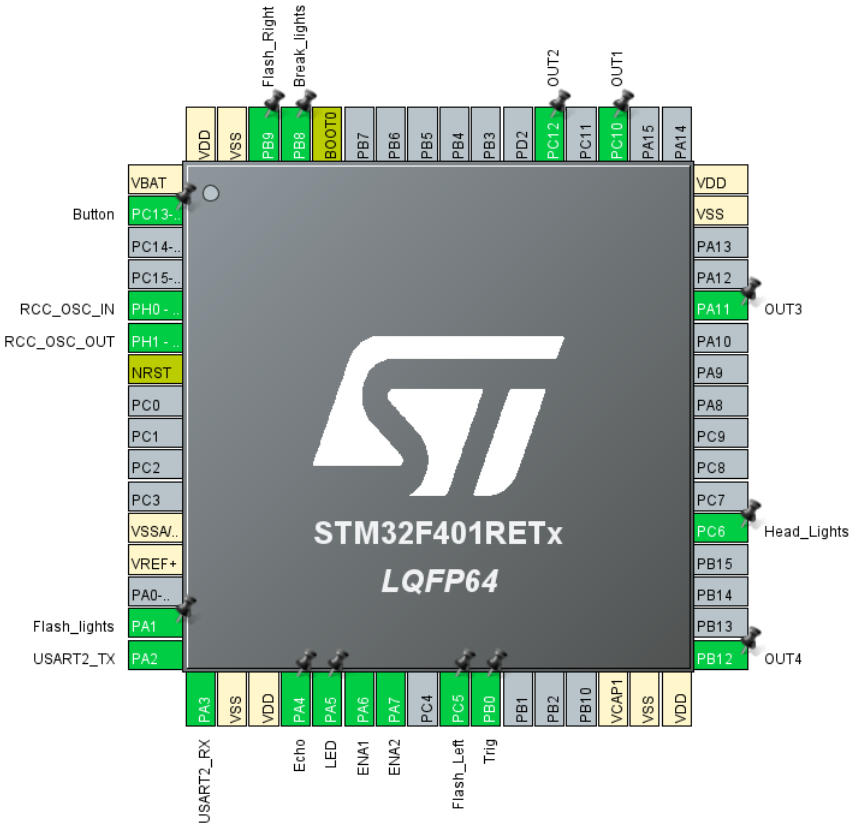


Figure 4.18: Pinout diagram of MCU

Table 4.1: Pinout Table

Pin	User Label	Purpose
PA2	USART2_TX	UART Transmit
PA3	USART2_RX	UART Receive
PA5	LED	Indicator for the Board
PA6	ENA1	Output for H-Brigde
PA7	ENA2	Output for H-Brigde
PA11	OUT3	Output for H-Brigde
PB0	Trig	Output Trigger for Ultrasonic
PB2	Break_lights	Output for Break Lights
PB7	Echo	input from Ultrasonic
PB8	Direction_Sign_Left	Output for Direction-Sign
PB9	Direction_Sign_Right	Output for Direction-Sign
PB12	OUT4	Output for H-Brigde
PC6	Head_lights	Output for Head Lights
PC10	OUT1	Output for H-Brigde
PC12	OUT2	Output for H-Brigde

For the development of the code, we used a real-time operations system, FreeRTOS, distributed with the ST software. Initially, we used Cube Mx to initialize our tasks, which will be 5. These tasks will be executed in parallel and independently, each with priority. One task was to generate a signal to activate the ultrasonic sensor. In fact, within this task, we would change the output state of the micro selector to produce a signal with a frequency of 40Hz for 10us duration. In this way, we would wake up the control circuit of the sensor. The sensor will send a 40KHz signal from the transmitter, the signal after reflection will be received by the receiver, and then we will start measuring the pulses returned to us by the sensor control circuit. In practice, we measure the pulse duration from the moment we detect a rising edge until we detect a falling edge. Using the formula below, we can calculate the distance to the obstacle. The second task examines if the distance to the obstacle is less than it should be. Suppose we exceed the threshold, a flag rise to be managed in another task to break the car. The third task manages the operation of the drive motor. In this task, we check the value of the duty cycle we have set serially. In this work, the duty cycle is considered the speed of movement. This task is responsible for both starting and breaking. The fourth task performs the routine, and the second motor, used to turn the vehicle, checks every time we send a message for a change of direction and initiates the appropriate flag to light the appropriate Direction lights. The fifth and final task manages the operation of the warning and direction signs.
In the other part of the code, which concerns the data acquisition, i.e., the communication raspberry - Nucleo Board, we use the UART protocol combined with the DMA. As we mentioned in the background, the DMA [Figure 2.26] stores the data sent directly to the processor's memory without needing to employ it. Each time the DMA's callback is called, it is checked for the type of message received if it is correct and does not contain garbage. Depending on the type of message, we raise the appropriate flags, which will be handled by a switch case that we will call in our task.

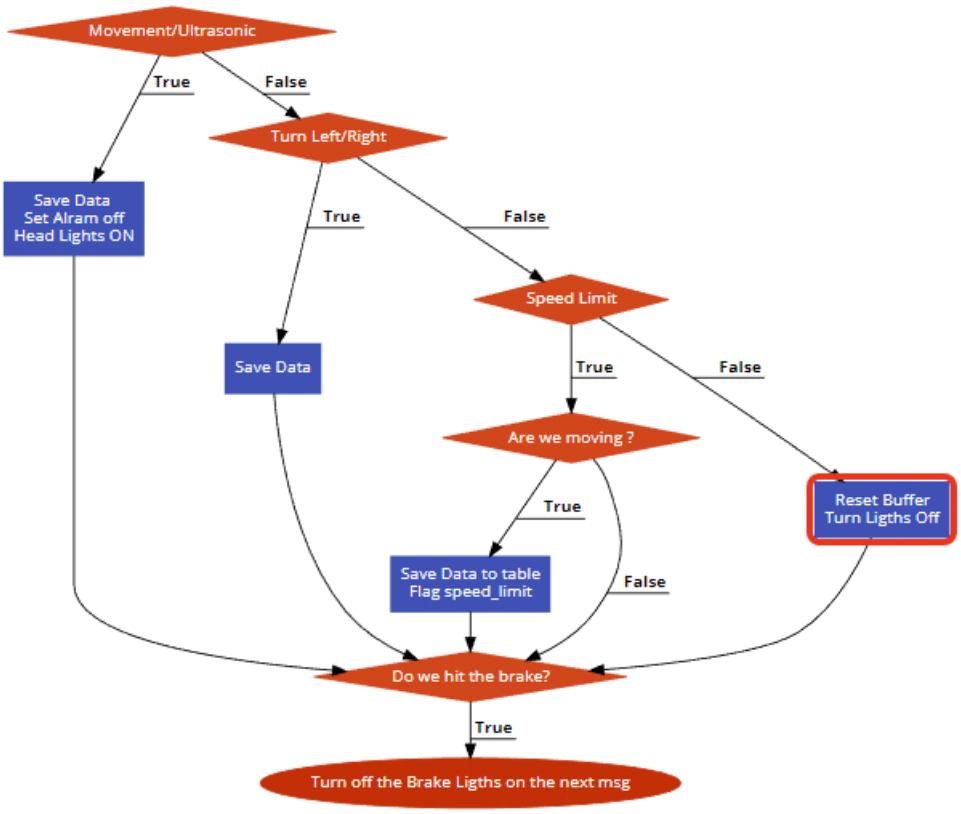


Figure 4.19: DMA Callback

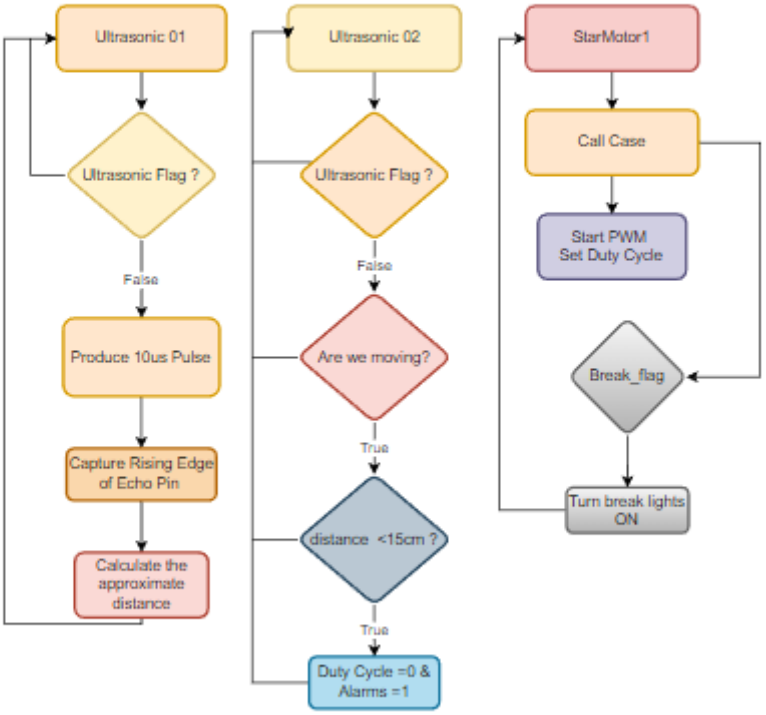


Figure 4.20: FreeRTOS Tasks 1

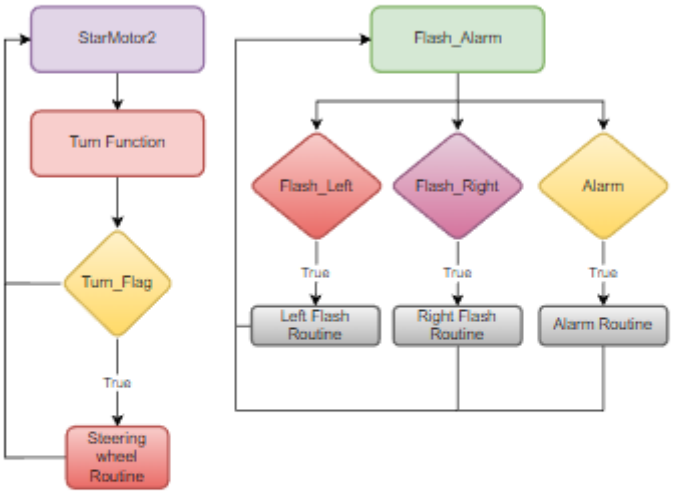


Figure 4.21: FreeRTOS Tasks 2

4.6 Version Control

During the development of a project [38], it is common to make mistakes that can have enormous consequences on the development of the work. How often have we forgotten to save a text file and work from scratch? In order to avoid possible omissions in both the programming part, the design part, and the database development part, we will use the Version Control method. When we refer to version control, we mean the complete recording of the history of the changes we have made throughout time, from the beginning of the project to its development. Engineers often use this method to have a complete picture of the changes in their code. Many times during the development of a code, we are required to make several changes to improve the functionality of our system, but these changes are not always painless since a change in the code can break an essential function in the system. If we do not have version control, we will be unable to figure out where we have created the problem. Also, we may have broken a functionality, but it may not have been noticed in a short period, so if some time passes, the fix can be very harmful. We

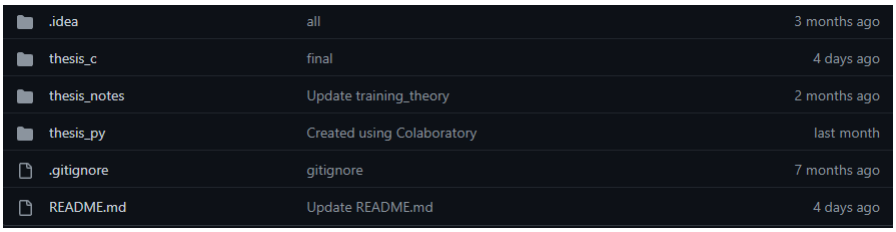


Figure 4.22: Repository

will use GitHub [39] as our internet hosting service; there are plenty of options, like Azure, BigBucket, GitLab, and many more. We chose Github because the main features are free of cost. We will use the CLI method, which means we will use a terminal to upload our code. To make it clear, Git is the version control system that provides us with tracking of our source code, and GitHub is the cloud base where we host our repositories. The repository is the structure data where we store the metadata of our source code. We will use a monorepo, meaning we will have only one repository for our project codes (python, C). We start by creating an account on Github and then create our project repository named thesis. Our repo will have the following structure, thesis_c, which will include the STM32 firmware code, thesis_py, model training, and every code we develop with python. The last folder on our repository will be about the notes we keep during this project (Background, Calculations, Examples of Codes)

4.6.1 Git Procedure

At this point, we will describe the progress of the work through git. The diagram below [Figure 4.23] describes the process we perform each time to make a change to our code. For each fix we want to make, we have to create a new branch with an appropriate name. Update our local archive, and any change we make will be applied to the branch, and then we will make a pull request, run some basic tests for system functionality and finally merge to the master.

1. We open a Git Terminal.
2. We create a Folder in the Desktop.
   $ mkdir Thesis
3. Git Clone repository.
   $ cd Thesis
   $ git clone git@github.com:tonika31/thesis.git
4. Git pull method , we download our stored data from cloud to local.
   $ cd thesis
   $ git pull
5. We made our first commit.
   $ git commit -m "begin"
6. Push our changes to master.
   $ git push

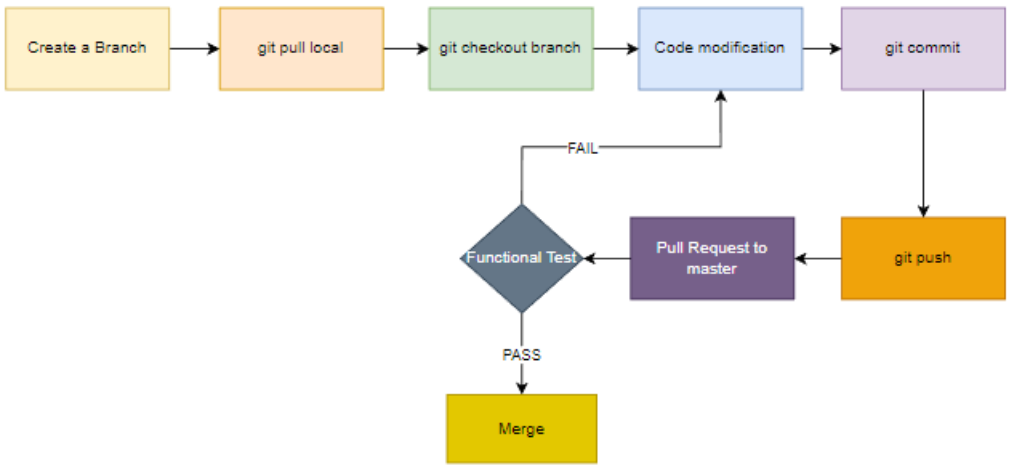


Figure 4.23: Pull Request Procedure

4.7 Electronics

In this part of the work, we will present the electronic part of our work. Includes H-bridge for motors control, Battery Management System, the ultrasound sensor, and the wiring of all the parts, such as the microcontroller, the Raspberry, and the power supplies, as shown in the figure below [Figure 4.24].

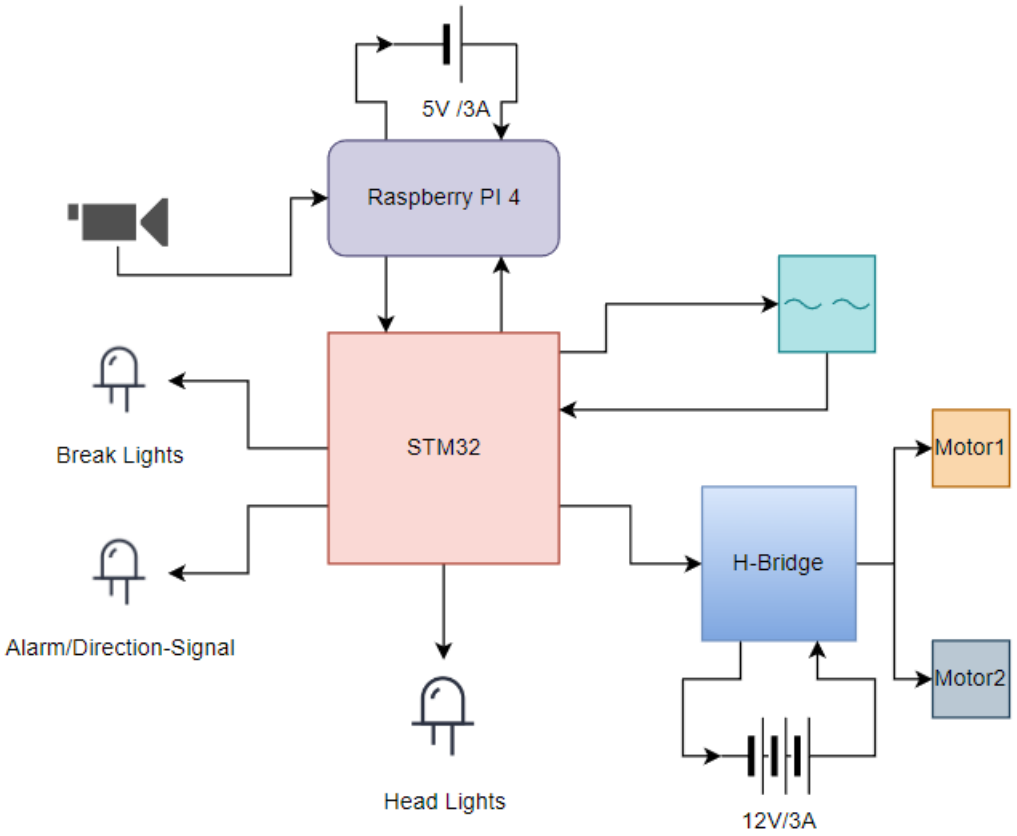


Figure 4.24: Connections Block Diagram

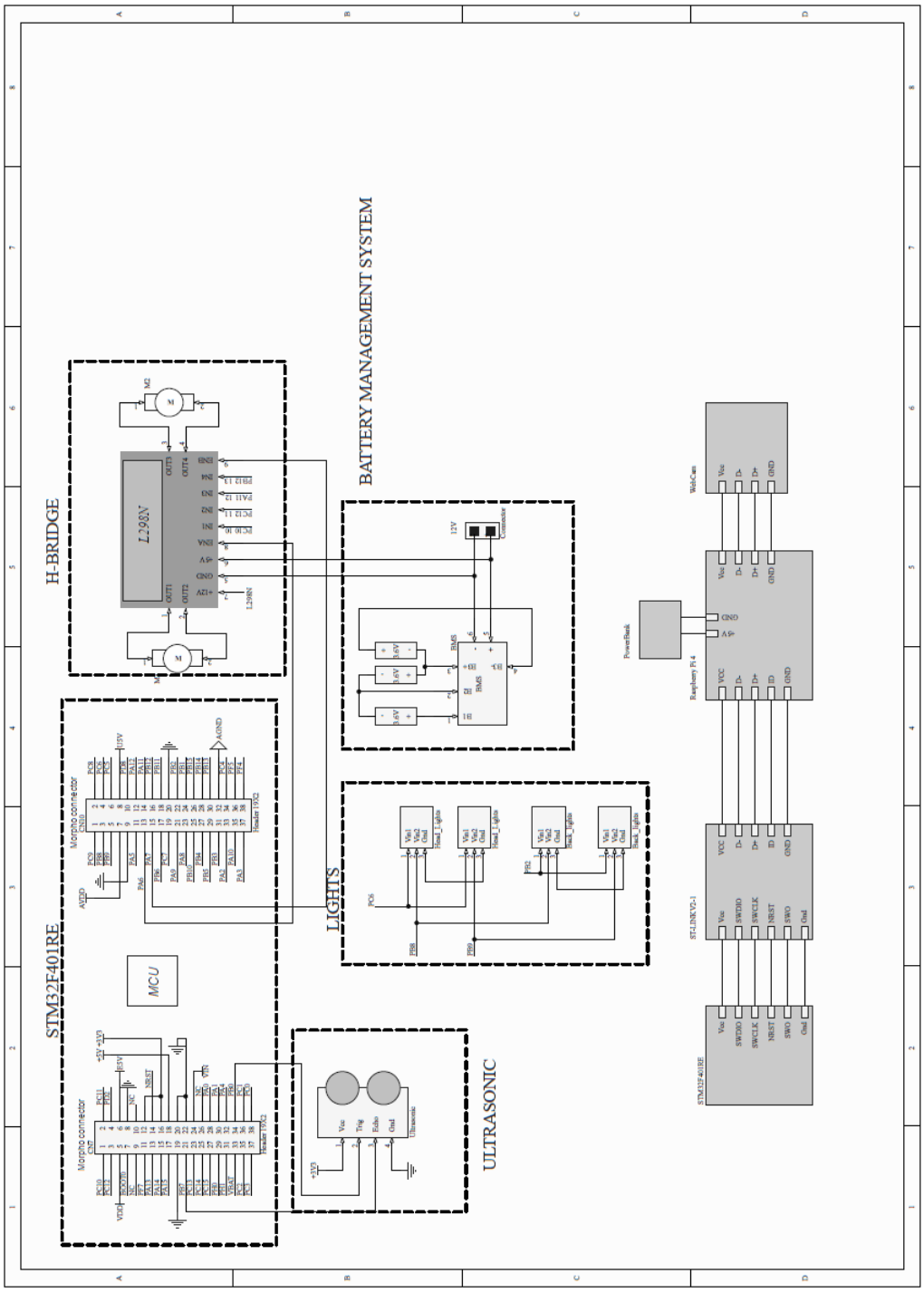


Figure 4.25: Electronic design

4.7.1 H-Bridge

The control of our motors will be done with an H-Bridge circuit. We chose to use a ready-made module [Figure 4.26] with a double H-bridge, high voltage, and current that supports the standard TTL logic. Specifically, it has the L298N [40] from ST, an integrated monolithic circuit which distributed in two packages; the Multiwatt is the one we use. It can power inductive loads, relays, DC, and stepper motors.Our board use a regulator L78M [41] in the input. Below we have the wiring diagram [Figure 4.27] for the H-Bridge. We have two outputs, one for each motor; one is for vehicle movement, and the other is for changing the vehicle's direction; in essence, it turns the two front wheels. The circuit can be powered from 5-35V; however, we have a voltage drop across the bridge chip of around 2V; we need to supply at least 12V to have the right potential at the ends of the motors, which operate at 9V. Three pins control each motor. One pin activates the first output of the bridge, while the other two pins determine the rotation time of the motor. Specifically, Enable 1 & 2 activate the respective outputs, while the signals that we have named OUT1-4 control the rotation time of the motors. On the Enable pins, we apply PWM from the outputs of the microcontroller.

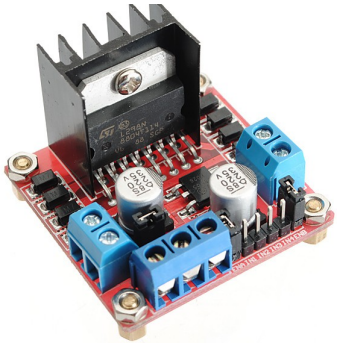


Figure 4.26: H-Bridge L298N

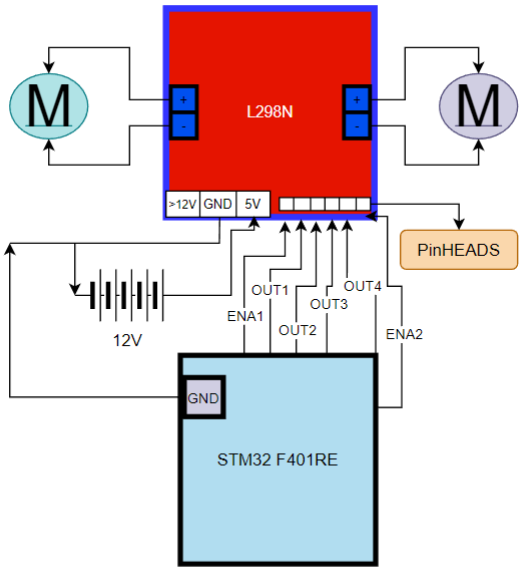


Figure 4.27: H-Bridge L298N wire connections

4.7.2 Battery Management System

To power the H-Bridge, we need 12V; for this purpose, we used three lithium-ion batteries with a nominal voltage of 3.6V. We will connect them in series to have a voltage of 10.8V. However, because we need to charge the batteries, we need to use a charging circuit for lithium batteries. We chose to use the following circuit that we buy. The batteries are connected in series, and we have three points from which we measure the Voltage so that the charging circuit is stop charging the batteries when they reach 4.2V.Our board has the S8254A chip [42], which is responsible for the protection of the batteries,and four AO4407A P-Channel Mosfet [43].

Figure 4.28: Battery Management System

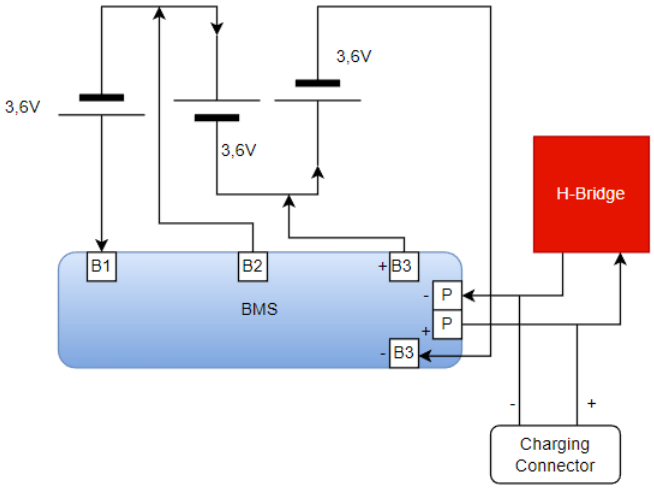


Figure 4.29: Battery Management System Connection Diagram

4.7.3 Ultrasonic Module HC-SR04

We used the HC-SR04 module for the ultrasonic sensor [Figure 4.30], which consists of two piezo sensors, one for transmission and one for the reception. In order to use the combined circuit, we have to initially send a 10us TTL signal to start the transmitter to send a 40KHz signal for eight periods, which the receiver will clear with the reflection. The specific board works with 5V and a maximum current of 15mA. It can measure from 2 centimeters to 2 meters.We use the RCWL-9300 stylus for sending the signal while receiving the RCWL-9200. The measurements we receive are passed through an LM324 [44] amplifier.

Figure 4.30: Ultrasonic Module

4.8 Mechanicals

In this part of the work, we list all the mechanical parts we designed and printed using the Fusion 360 program and the 3D printer of Creality. First, we designed a base for the raspberry board[Figure 4.31]; for this purpose, we used the engineering drawings that we found available from the company that made the board, and we used those designs for the dimensions. The rest of the design paperwork was produced by measuring according to our chassis. The following design [Figure 4.32] is for the installation of the camera but also the construction of the lights; for this purpose, we made measurements concerning our materials. Next, we have the base for our ultrasound sensor[Figure 4.33], and again we made some measurements. Finally, we have the case for the batteries[Figure 4.34] and the backlights [Figure4.35]. All the following designs have been designed by us and have been printed using PLA material.

4.8.1 Raspberry Pi 4 Holder 3D Design

Figure 4.31: Raspberry Pi 4 Holder

4.8.2 Camera and Headlights Holder 3D Design

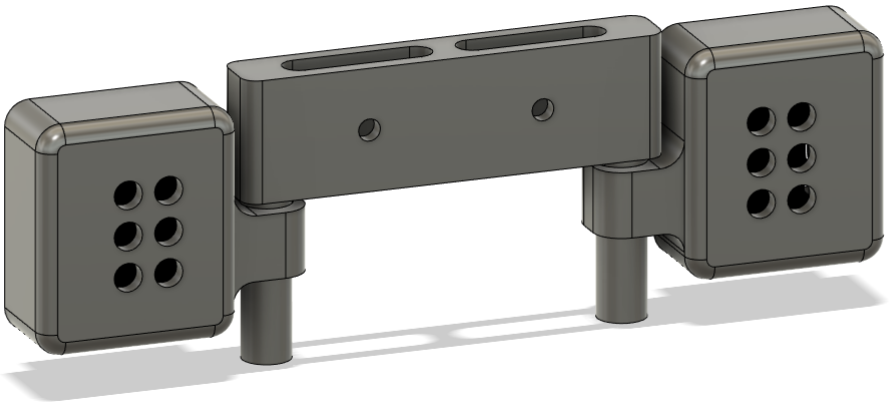


Figure 4.32: Camera

4.8.3 Ultrasonic Holder 3D Design

Figure 4.33: Ultrasonic Holder

4.8.4 Batteries case 3D Design

Figure 4.34: Batteries case

4.8.5 Back Lights 3D Design

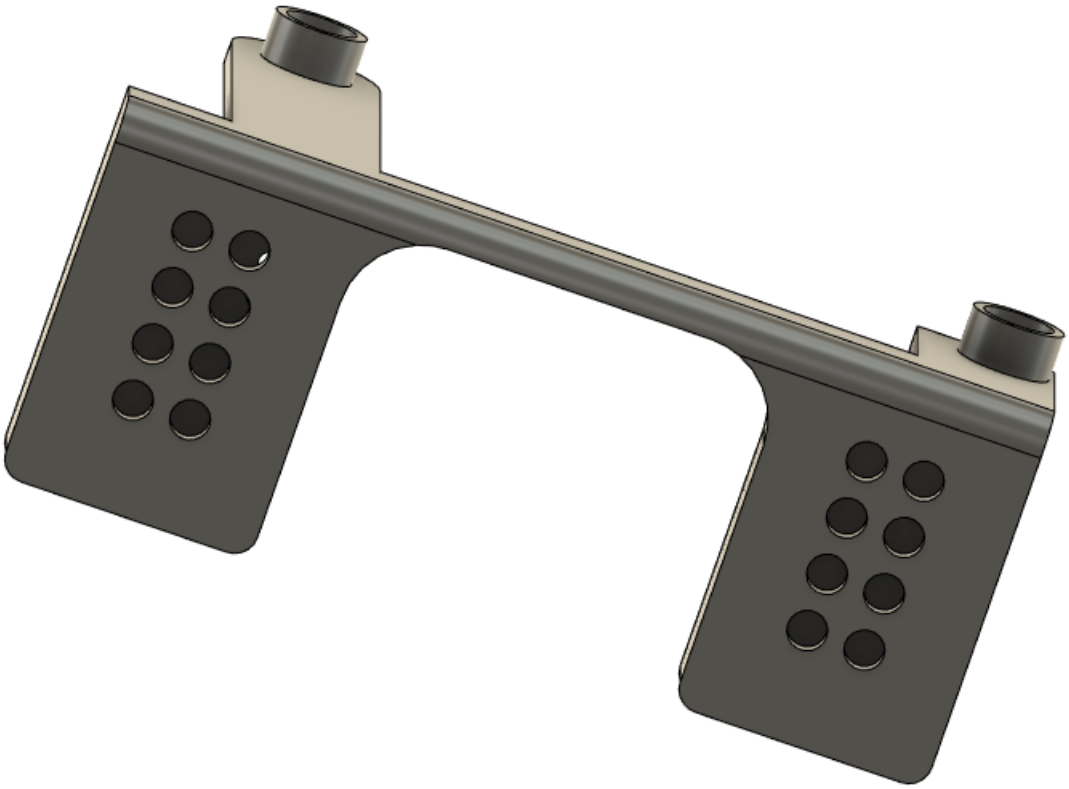


Figure 4.35: Back Lights Design

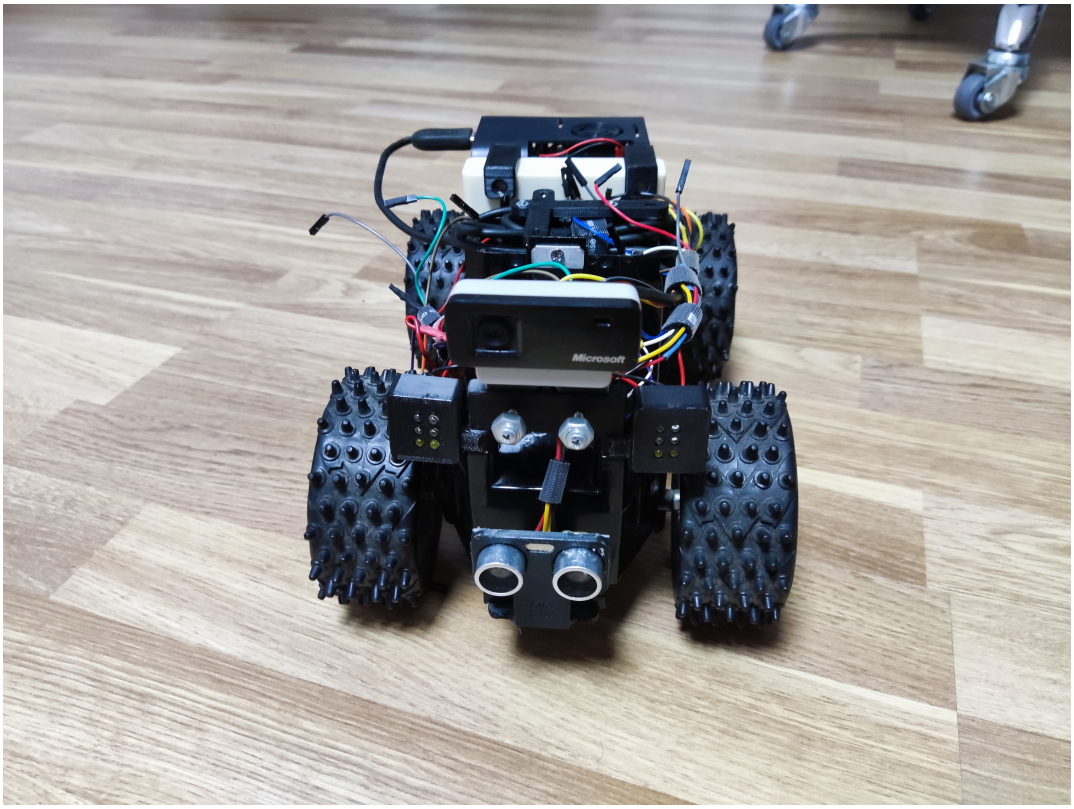


Figure 4.36: Final View 1

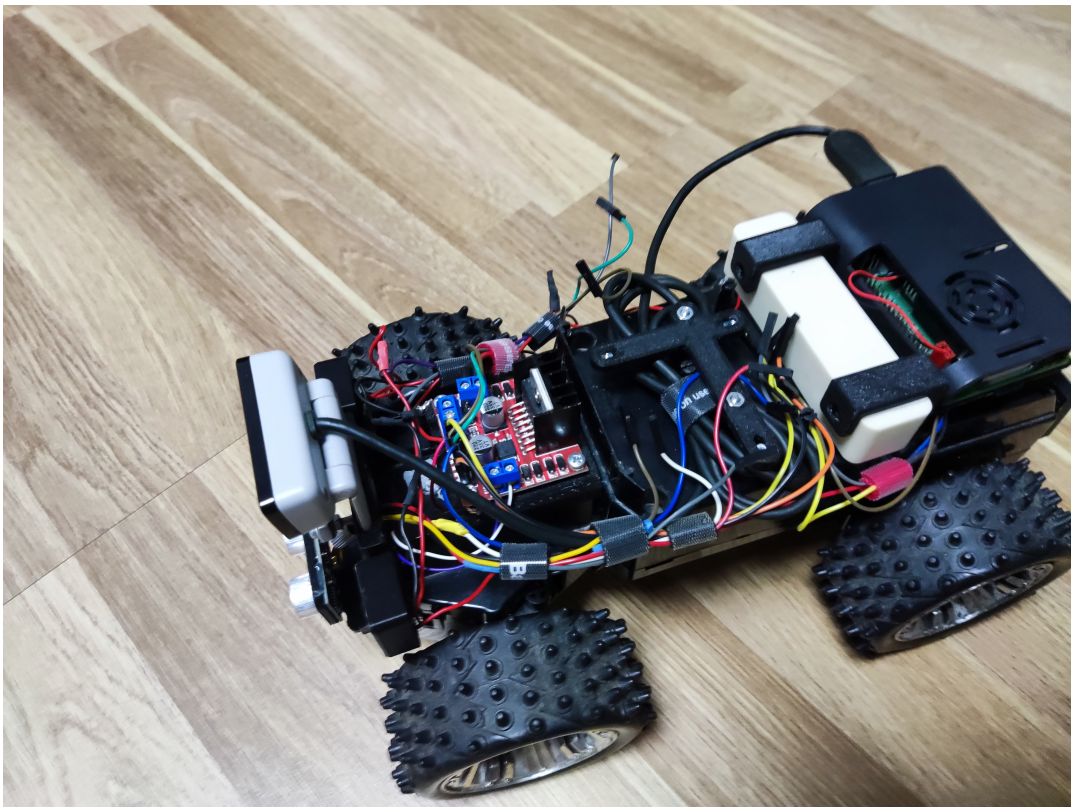


Figure 4.37: Final View 2

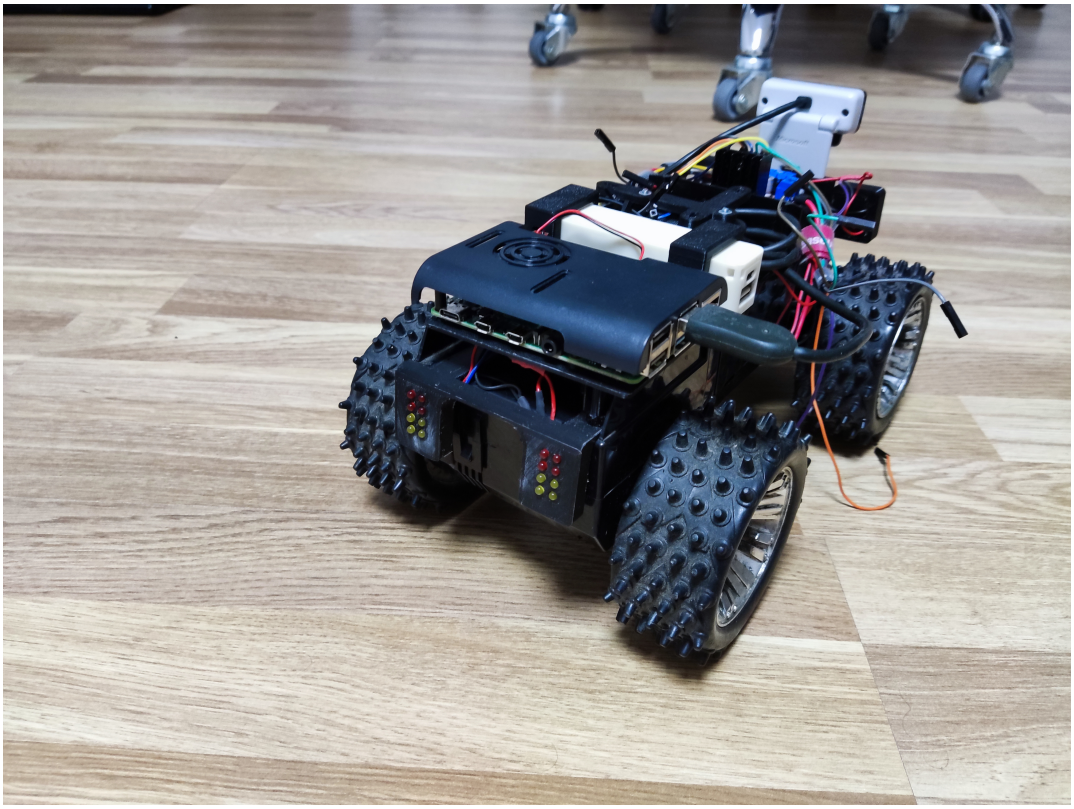


Figure 4.38: Final View 3

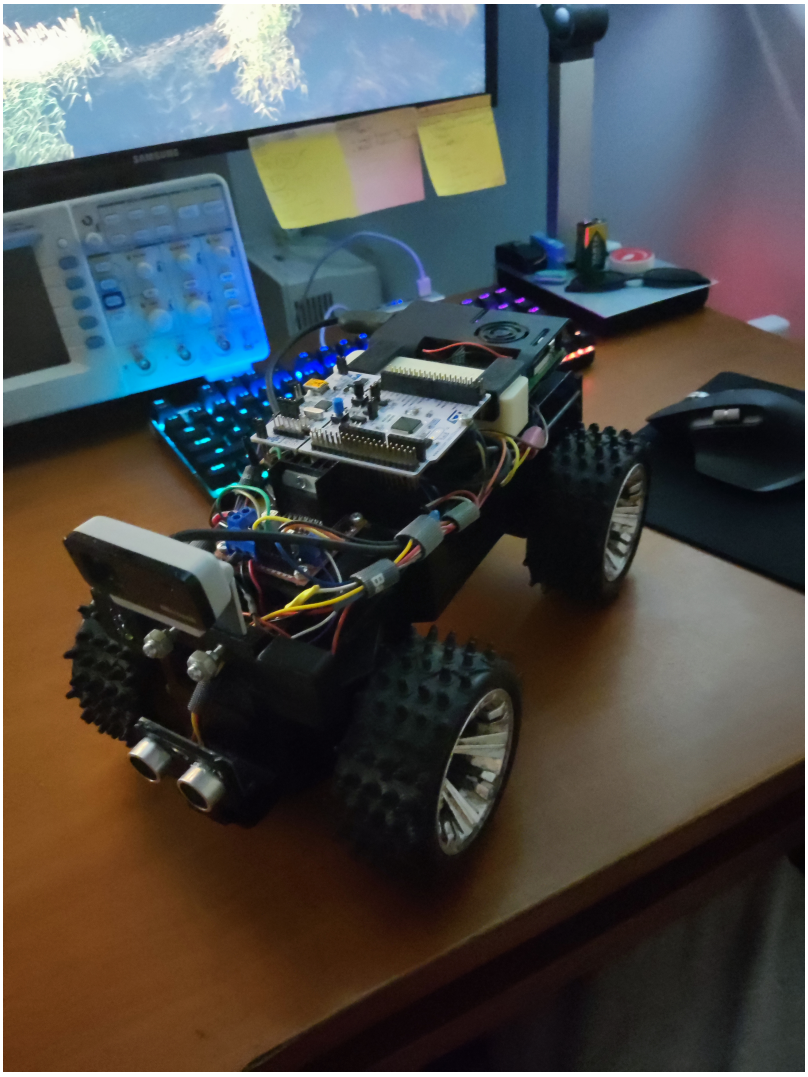


Figure 4.39: Final View 5

4.9 Summarize

In this chapter, we presented our thinking to implement our work. We started with a detailed description of the database construction that we would need to develop the neural network model. We presented the structure of our model and how it works. Then we described the structure of our code for the embedded system. Finally, we mentioned the component of the electronics and mechanical parts for the construction of the vehicle.

5 Results

At this point in the paper, we will present the results of our work concerning the land network model for identifying our classes - our flags. We will compare the efficiency of our model between the original data and the augmented. First, we will compare the graphs concerning training and validation between the two categories of data. Then we will compare the loss graphs for the same subject. Finally, we will take each class, and we will present the accuracy of each case. In the first figure [Figure : 5.1], we have the validation and training progress graph for the original data, and on the next page, we have the graph for the augmented. As described in the previous chapter, we mainly use augmented data because we want more data to improve our training accuracy. By augmented data, we mean we processed the images to create new images with differential parameters. In other words, we took batches of images, rotated them, enlarged them, made them smaller, or altered them by cutting them up or down. This is a widespread technique in Machine Learning.

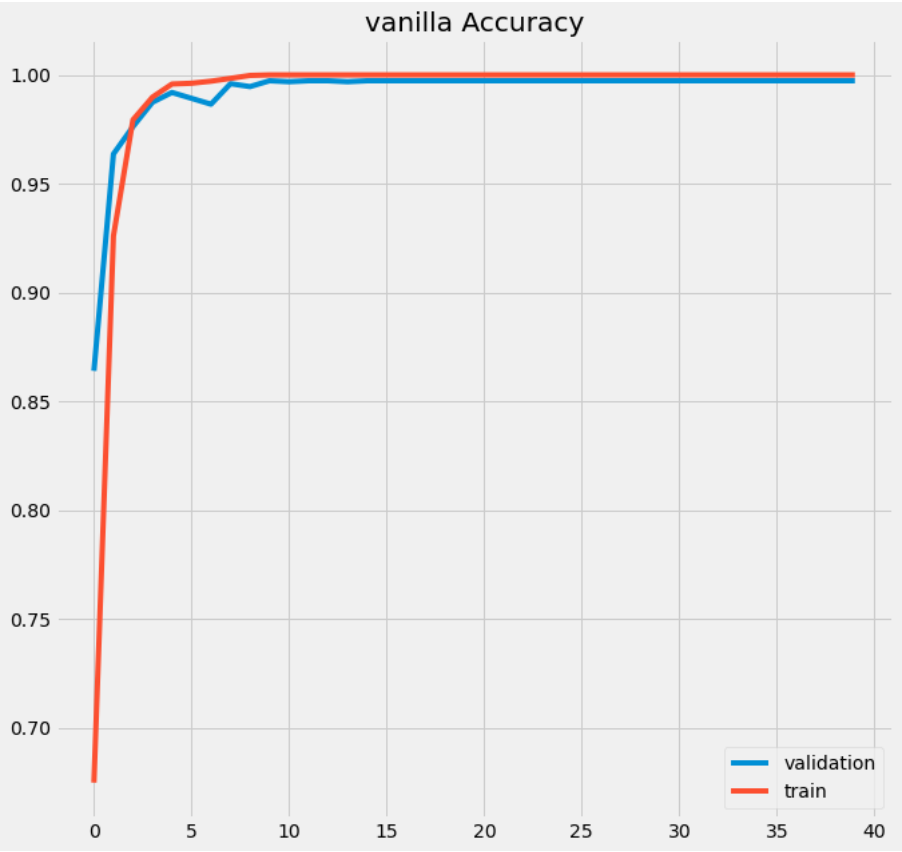


Figure 5.1: Model Accuracy Plot with Original Data

In our diagrams, we have two axes, x, which concerns epochs, and y, which concerns precision. In this type of diagram, we have to pay attention to whether the training graph coincides with the validation graph. We want the two graphs to be as identical as possible. Therefore, according to the above, the first graph has excellent performance as we can see that the two graphs coincide quite soon during the training. We could even complete the training with a shorter epoch. In the graph [Figure 5.2] with the augmented data, we can see that the two graphs do not coincide. This can lead us to two conclusions: we need more epochs for training to see if it will narrow after a certain point. On the other hand, such a graph shows us that we have the overfitting phenomenon. As we have significantly increased our data, it is ubiquitous to create this problem as we create a considerable homogeneity in our data which means that we have built a model that can only recognize our images. Therefore the way we have managed the data with this method is wrong, and it would be good to try different parameters on our images or to grow our database with authentic images.Also, from diagrams [Figure 5.3 ,5.4] concerning the loss function, we want the two graphs to coincide; as we can see with the original data, we have achieved this, while with the augmented data, we see an anomaly in the graph.We have name our graph as vanilla because its a simple C.N.N model.

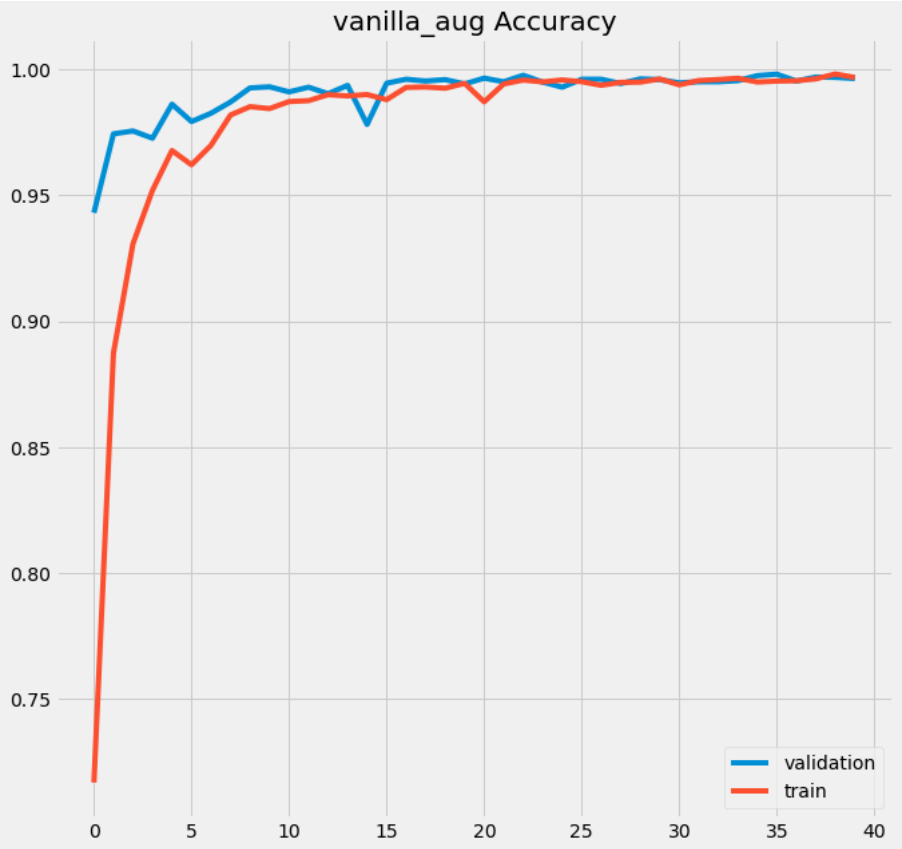


Figure 5.2: Model Accuracy Plot with Augmented Data

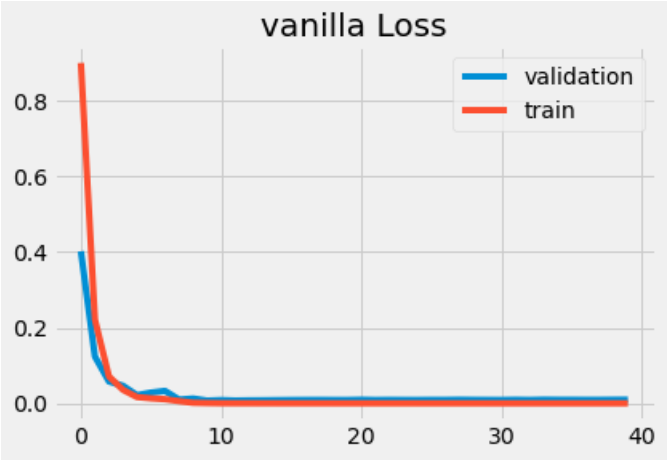


Figure 5.3: Model loss Plot with Original Data

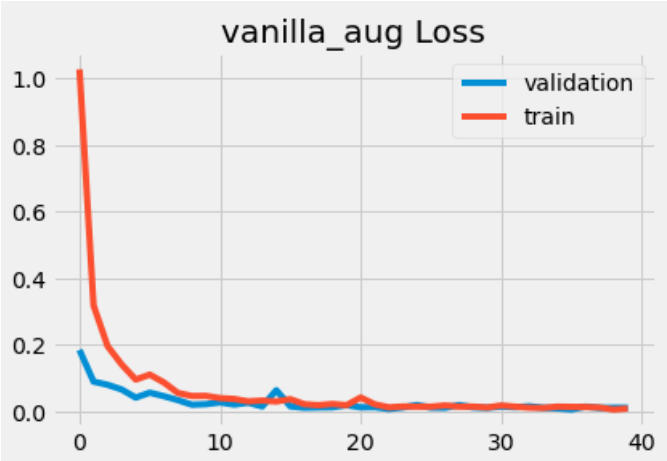


Figure 5.4: Model loss Plot with Augmented Data

At this point, we had to test our model for each class separately. Therefore, we started with a stop labeling image; after applying the processing technique we used in training, we inserted the image into the model to find the label and calculate with certainty that we had the results. We have 99% certainty that this image corresponds to class four, corresponding to STOP[Figure 5.5,5.6]. Then we tested for an image with the label 30km; with 100% certainty, this image corresponds to class 0, which is 30km [Figure 5.7. Again we applied the same experiment with the sign of 120 km, and the result showed 99% certainty that our model assigned this image to class 5, i.e., 120 km [Figure 5.8]. We have also identified a problem detecting the left and right turn signs, mainly because these two classes are pretty confused, mainly because the model we developed is more comfortable with numerical systems and alphabets.

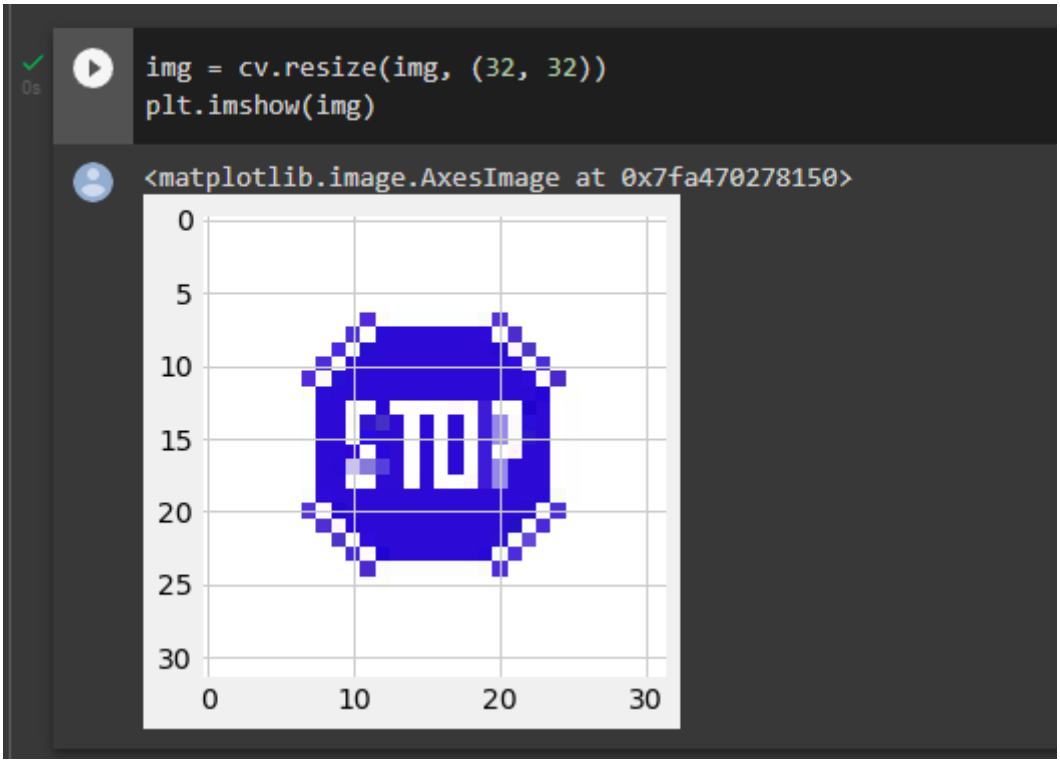


Figure 5.5: Stop Sign resize

```
[18] img = img.reshape(1, 32, 32, 1)
     predictions = model_1.predict(img)
     print(predictions)
     probabilityValue = np.amax(predictions)
     print("\n" , probabilityValue)

     [[7.4682538e-17 4.4389293e-03 3.4805616e-09 7.4762662e-05 9.9548608e-01
       2.0091645e-07 2.7504653e-08]]

      0.9954861

     classe = np.argmax(predictions, axis=1)
     print(classe)

     [4]
```

Figure 5.6: Stop Sign Prediction

Figure 5.7: 30km Sign Prediction

Figure 5.8: 120km Sign Prediction

6 Conclusion

6.1 Conclusion

In this work, we presented our approach to an advanced driver assistance system; we built a vehicle with a webcam and an ultrasound sensor to detect vehicles, the data processed by a computer (Raspberry Pi 4), and a microcontroller, the F401RE. This work aimed to recognize traffic signals, namely 30km,50km,70km,120km, Stop, Turn left ahead, Turn right ahead, using Machine Learning, and send the appropriate commands to the vehicle's motion control unit. We use the TensorFlow framework and Keras API for the Convolutional Neural Network. From the final results, we have seen that we are facing some problems that need to be solved in a future project; these problems are as follows, we have a slight failure in annihilating the signs to turn left and right, as far as the neural network part is concerned. The images we use for the live feed must have a specific size. However, we can say that this work was successful for the following reasons: First of all, it put us in a challenging position in the knowledge part; we had to deal with things we had never touched during our studies. Therefore, the work was successful as we gained new images and skills. We combined many fields: Machine Learning, Embedded Firmware, electronics, and engineering. If we exalt the electronics part, where we used ready-made boards and did not design our own, this is mainly due to the lack of materials prevailing in the market, production time, and cost of a board. All rest of the work was completed by us. We developed the Convolutional Neural Network model, the python codes for the automation of some processes, the development of the code for the microcontroller, and the construction design of the mechanical parts of the work were done by us.

6.2 Future work

6.2.1 Convolutional Neural Netowrk

It is a fact that our work is not a perfect product, so we will now present some thoughts on how to optimize this work. The first part is the part of the neural network; what needs to be done is to develop a model that can handle all the traffic codes, even the traffic lights. This requires a much more extensive database and a model that can be trained accordingly. Second, the detection of vehicles should be done through the camera and not ultrasound sensors. An attempt was made during this work, but it was not implemented due to time constraints. In addition, in the database's development process, it would be good to update it with images we capture when the vehicle moves to improve our images' uniqueness.

6.2.2 Autonomous Driving system

Another direction this work should take is autonomous driving; in particular, it would be a perfect start to integrate our existing work into a system such as the ROS Robotic Operating System. The structure and functionality of this system will give us new possibilities, such as mapping the areas we move around and developing a base where we know where there are markings, and we will be able to implement a vehicle that will move without human assistance.

6.2.3 Embedded Firmware

In the section of the Embedded Firmware, we have several changes that need to be made first to develop our communication protocol, mainly for security issues. In this work, we have not mentioned the interception section at all. We must apply several techniques to transmit messages between computing devices securely. We need to improve the transmission of messages so we can detect errors in the data, such as the CRC technique. In general, we need to see how we can improve the structure of our code. Testing will play a significant role in this part, so we must do frequent tests to see our functionality. We spent several hours debugging in the context of this project. However, it was not enough because we did not simulate the natural environment. We need to test actual conditions to judge if our system can stand on the road. We have not done any tests concerning the stability of our system under different interference, how it will affect the operation of the microcontroller, and what can go dangerously wrong in our system.

6.2.4 Continuous Integration/Continuous Development

Following on from the previous section, as far as testing is concerned, we should include the Continuous Integration/Continuous Development CI/CD methodology. It is a technique used by all developers that relate to how a team works to be able to run tests and simultaneously develop improvements to its code. The general picture is that every time we make a new change to our code, as we have explained in a previous chapter using git, we need to run some tests to see the system's functionality. Usually, this can take several minutes to hours. We spent several hours debugging and testing; this is not good, as when working alone in this particular job, time is very quickly running out. With CI/CD, we can automate the process by writing all the tests; the whole process will be done through a server.

6.2.5 Mechatronics

Our greatest desire is to build a vehicle of actual dimensions which a human being can use. Then in the mechatronics part, what we would like to do is to build a fully equipped vehicle that will be equipped with the appropriate electronic systems to manage our model. Our initial thought is to go to a more connected computing unit like Jetson and design our Battery Management System and engine control. This may have to be done in separate projects because of the amount of work involved. Also, in this work, we have 3D printed most of the parts of our construction, so we would ideally like to use this technique to build the mechanical parts for our advanced vehicle.

6.3 What did we learn?

When we took on this project, we had specific knowledge. We did not know the programming language python nor how to install it on a computer. We had not used raspberry and were unfamiliar with the Ubuntu terminal. In the embedded part, we had fundamental knowledge we learned during our studies. We had no contact with 3D design and printing. We also had no idea about AI technology or neural networks. The only knowledge we had was about electronics and board design. Therefore, we had the pleasure to be in contact with different fields, and if not all of them, certainly not all, become part of our own and be part of our career.

References

[1] Eurostat, "Eu population continues to decrease for a second year." https://ec.europa.eu/eurostat/web/products-eurostat-news/-/ddn-20220711-1#:~:text=After%20a%20first%20decline%20in,446.8%20on%201%20January%202022.

[2] Eurostat, "Road safety statistics - characteristics at national and regional level." https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Road_safety_statistics_-_characteristics_at_national_and_regional_level&oldid=463733#Road_traffic_fatalities.

[3] Mercedes-benz, "The first automobile." https://group.mercedes-benz.com/company/tradition/company-history/1885-1886.html.

[4] P. Burns, "Analysis of image noise in multispectral color acquisition," 01 2001.

[5] Ι.Ν.ΕΛΛΗΝΑΣ, *ΨΗΦΙΑΚΗ ΕΠΕΞΕΡΓΑΣΙΑ ΕΙΚΟΝΑΣ & ΒΙΝΤΕΟ*. ΛΥΧΝΟΣ, 2010.

[6] "German road signs." https://www.iamexpat.de/expat-info/driving-germany/road-signs. Accessed: 2022-08-02.

[7] A. Nemcsics and J. Caivano, "Color order systems," pp. 1–16, 01 2015.

[8] S. Chandran, "Color image to grayscale image conversion," pp. 196 – 199, 04 2010.

[9] O. Patel, Y. Maravi, and S. Sharma, "A comparative study of histogram equalization based image enhancement techniques for brightness preservation and contrast enhancement," *Signal Image Processing : An International Journal*, vol. 4, 11 2013.

[10] P. Amoako-Yirenkyi, J. Appati, and I. Dontwi, "Performance analysis of image smoothing techniques on a new fractional convolution mask for image edge detection:," *Open Journal of Applied Sciences*, vol. 06, pp. 478–488, 01 2016.

[11] "Smoothing images." https://docs.opencv.org/4.x/d4/d13/tutorial_py_filtering.html.

[12] J. Canny, "A computational approach to edge detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[13] M. Pavlović, I. Ćirić, D. Ristic-Durrant, V. Nikolić, M. Simonovic, M. Ciric, and M. Banic, "Advanced thermal camera based system for object detection on rail tracks," *Thermal Science*, vol. 22, pp. 1551–1561, 01 2018.

[14] G. Amer and A. Abushaala, "Edge detection methods," 08 2015.

[15] J. Jantzen, *Introduction to Perceptron Networks*. Technical University of Denmark, 1998.

[16] L.R.Foulds, D.Haugland, and K.Jørnsten, *A bilinear approach to the pooling problem*. 2007.

[17] A. Sherstinsky, *Fundamentals of recurrent neural network (rnn) and long short-term memory (lstm) network*. 2018.

[18] K. O'Shea and R. Nash, *An introduction to convolutional neural networks.* 2015.

[19] K.He, X. Zhang, S. Ren, and J. Sun, *Deep residual learning for image recognition.* 2015.

[20] J. García Cabello (JG Cabello), "Mathematical neural networks," *Axioms*, vol. 11(2), 02 2022.

[21] R. Y. M. Nishio and Do, "Convolutional neural networks: an overview and application in radiology," vol. 11(2), 07 2018.

[22] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," pp. 1–6, 2017.

[23] J. Nagi, F. Ducatelle, G. Di Caro, D. Ciresan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," *2011 IEEE International Conference on Signal and Image Processing Applications, ICSIPA 2011*, pp. 342–347, 11 2011.

[24] B. Ding, H. Qian, and J. Zhou, "Activation functions and their characteristics in deep neural networks," pp. 1836–1841, 2018.

[25] K. Janocha and W. Czarnecki, "On loss functions for deep neural networks in classification," *Schedae Informaticae*, vol. 25, 02 2017.

[26] T. Chai and R. Draxler, "Root mean square error (rmse) or mean absolute error (mae)?– arguments against avoiding rmse in the literature," *Geoscientific Model Development*, vol. 7, pp. 1247–1250, 06 2014.

[27] A. Etz, "Introduction to the concept of likelihood and its applications," *Advances in Methods and Practices in Psychological Science*, vol. 1, no. 1, pp. 60–69, 2018.

[28] D. Kingma and J. Ba, "Adam: A method for stochastic optimization," *International Conference on Learning Representations*, 12 2014.

[29] M. Zorkany, M. Hussein, and N. Kader, "Real time operating system for the internet of things; vision, architecture, and research directions," 03 2016.

[30] Ν.Νικολαίδης, *ΜΙΚΡΟΕΛΕΓΚΤΕΣ*. Αφοί Κυριακίδη, 2018.

[31] J. Yang, D. Minturn, and F. Hady, "When poll is better than interrupt," pp. 3–3, 02 2012.

[32] A. Ahmed, A. Aljumah, and M. Ahmad, "Design and implementation of a direct memory access controller for embedded applications," *International Journal of Technology*, vol. 10, p. 309, 04 2019.

[33] Χ.Μαδεμλής, *ΣΕΡΒΟΚΙΝΗΤΗΡΙΑ ΣΥΣΤΗΜΑΤΑ*. Εκδόσεις Τζιολα, 2016.

[34] Μ.Ν.Σπάσος and Κ.ΘΑμοιρίδης, *ΣΥΓΧΡΟΝΕΣ ΕΦΑΡΜΟΓΕΣ ΑΝΑΛΟΓΙΚΩΝ ΗΛΕΚΤΡΟΝΙΚΩΝ*. Εκδόσεις Αϊβάζη, 2015.

[35] T. Özer, S. Kivrak, and Y. Oğuz, "H bridge dc motor driver design and implementation with using dspic30f4011," *International Journal of Innovative Research in Science, Engineering and Technology*, vol. 6, pp. 75–83, 05 2017.

[36] B. Balasingam, M. Ahmed, and K. Pattipati, "Battery management systems—challenges and some solutions," *Energies*, vol. 13, p. 2825, 06 2020.

[37] K. Panda, D. Agrawal, A. Nshimiyimana, and A. Hossain, "Effects of environment on accuracy of ultrasonic sensor operates in millimeter range," *Perspectives in Science*, vol. 8, 07 2016.

[38] "What is version control." https://www.atlassian.com/git/tutorials/what-is-version-control.

[39] "Git and github tutorial for beginners." https://product.hubspot.com/blog/git-and-github-tutorial-for-beginners.

[40] STMicroelectronics, L298N, *DUAL FULL-BRIDGE DRIVER*, 2000. Rev.

[41] STMicroelectronics, L78, *Precision 500 mA regulators*, 2020. Rev 24.

[42] Ablic, S8254A, *BATTERY PROTECTION IC*, 2019. Rev. 5.3_00.

[43] Alpha&Omega, AO4407A, *30V P-Channel MOSFET*, 2013. Rev.11.0.

[44] Onsemi, LM324, *Single Supply Quad Operational Amplifiers*, 2021. Rev.11.

A Clock Configuration

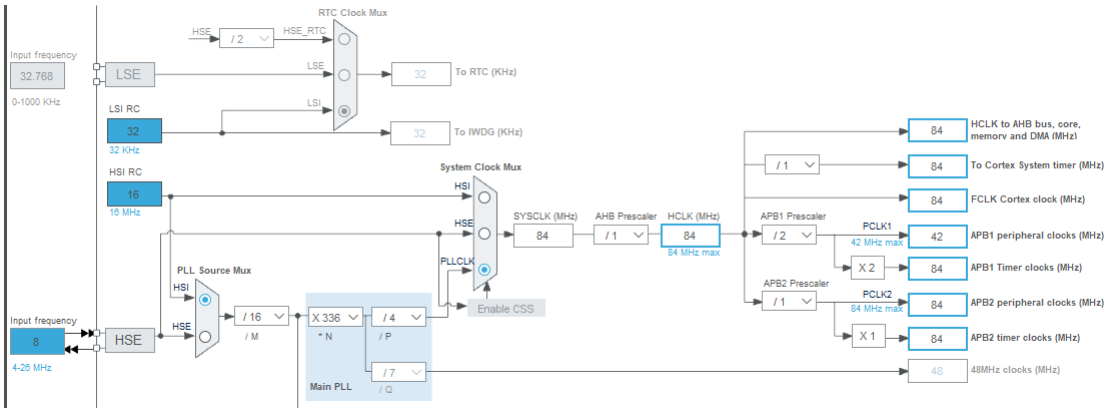


Figure A.1: Clock Configuration

B Define Variables

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  *AUTHOR : ANTHONY KARAGEORGIADIS
  *STUDENT ID: ELE516046
  *THESIS :2021 -2022
  **/
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"
#include "cmsis_os.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include "stdbool.h"
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define usTIM TIM4            // Define timer register name

uint8_t msg=0;                //Message  case detection variable

uint8_t trn=0;                // SWITCH CASE FOR TURNING RIGHT/LEFT
uint8_t duty1=0;              // PWM FOR THE FIST MOTER
uint8_t duty2=0;              // PWM FOR THE SECOND MOTER
bool    break_flag =0;              // FLAG FOR BREAK LIGHTS ON/OFF
bool  movement_flag=0;          // FLAG FOR SETTING FISRT THE ROTATION OF THE 1st MOTER AND THEN THE DUTY CYCLE
bool  trn_flag=0;
bool  flash_left=0;           // FLAG FOR TURN LEFT (FLASH)
bool  flash_right=0;                      // FLAG FOR TURN RIGHT (FLASH)
bool    alarm=0;              // FLAG FOR THE ALARMS
bool    speed_limit=0;            //FLAG FOR SPEED LIMIT SET
bool  ultra=0;              //ULTRASONIC ON/OFF

//Speed of sound in cm/usec
const float speedOfSound = 0.0343/2;    //  Speed constant
float distance;               //distance  variable for the ultrasonic
/* USER CODE END PD */
```

Figure B.1: Define Variables

C Define Tasks

```
55  TIM_HandleTypeDef htim3;
56  TIM_HandleTypeDef htim4;
57
58  UART_HandleTypeDef huart2;
59  DMA_HandleTypeDef hdma_usart2_rx;
60
61  /* Definitions for Ultrasonic01 */
62  osThreadId_t Ultrasonic01Handle;
63  const osThreadAttr_t Ultrasonic01_attributes = {
64    .name = "Ultrasonic01",
65    .stack_size = 128 * 4,
66    .priority = (osPriority_t) osPriorityNormal,
67  };
68  /* Definitions for Ultrasonic02 */
69  osThreadId_t Ultrasonic02Handle;
70  const osThreadAttr_t Ultrasonic02_attributes = {
71    .name = "Ultrasonic02",
72    .stack_size = 128 * 4,
73    .priority = (osPriority_t) osPriorityHigh,
74  };
75  /* Definitions for Motor1 */
76  osThreadId_t Motor1Handle;
77  const osThreadAttr_t Motor1_attributes = {
78    .name = "Motor1",
79    .stack_size = 128 * 4,
80    .priority = (osPriority_t) osPriorityLow,
81  };
82  /* Definitions for Motor2 */
83  osThreadId_t Motor2Handle;
84  const osThreadAttr_t Motor2_attributes = {
85    .name = "Motor2",
86    .stack_size = 128 * 4,
87    .priority = (osPriority_t) osPriorityLow,
88  };
89  /* Definitions for Lights */
90  osThreadId_t LightsHandle;
91  const osThreadAttr_t Lights_attributes = {
92    .name = "Lights",
93    .stack_size = 128 * 4,
94    .priority = (osPriority_t) osPriorityLow,
95  };
```

Figure C.1: Define Tasks

D Define functions

```c
/* Private function prototypes -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM4_Init(void);
void StartUltrasonic01(void *argument);
void StartUltrasonic02(void *argument);
void StartMotor1(void *argument);
void StartMotor2(void *argument);
void Flash_Alarm(void *argument);

/* USER CODE BEGIN PFP */
uint8_t Rx_1data[1];        // RECEIVED BUFFER
/* USER CODE END PFP */
```

Figure D.1: Define functions

E Main

```
126  int main(void)
127  {
128    /* MCU Configuration--------------------------------------------------------*/
129    /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
130    HAL_Init();
131
132    /* Configure the system clock */
133    SystemClock_Config();
134
135    /* Initialize all configured peripherals */
136    MX_GPIO_Init();
137    MX_DMA_Init();
138    MX_USART2_UART_Init();
139    MX_TIM3_Init();
140    MX_TIM4_Init();
141    /* USER CODE BEGIN 2 */
142    HAL_UART_Receive_DMA(&huart2, Rx_1data, 1);   // START THE DMA
143
144    /* USER CODE END 2 */
145
146    /* Init scheduler */
147    osKernelInitialize();
148
149    /* Create the thread(s) */
150    /* creation of Ultrasonic01 */
151    Ultrasonic01Handle = osThreadNew(StartUltrasonic01, NULL, &Ultrasonic01_attributes);
152
153    /* creation of Ultrasonic02 */
154    Ultrasonic02Handle = osThreadNew(StartUltrasonic02, NULL, &Ultrasonic02_attributes);
155
156    /* creation of Motor1 */
157    Motor1Handle = osThreadNew(StartMotor1, NULL, &Motor1_attributes);
158
159    /* creation of Motor2 */
160    Motor2Handle = osThreadNew(StartMotor2, NULL, &Motor2_attributes);
161
162    /* creation of Lights */
163    LightsHandle = osThreadNew(Flash_Alarm, NULL, &Lights_attributes);
164
165    /* Start scheduler */
166    osKernelStart();
167
168    /* We should never get here as control is now taken by the scheduler */
169    /* Infinite loop */
170    /* USER CODE BEGIN WHILE */
171    while (1)
```

Figure E.1: Main

F System Clock Configuration

```c
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct = {0};
  RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};

  /** Configure the main internal regulator output voltage
  */
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);

  /** Initializes the RCC Oscillators according to the specified parameters
  * in the RCC_OscInitTypeDef structure.
  */
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
  {
    Error_Handler();
  }

  /** Initializes the CPU, AHB and APB buses clocks
  */
  RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
                              |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;
  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Figure F.1: System Clock Configuration

G TIM3

```
229  static void MX_TIM3_Init(void)
230  {
231    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
232    TIM_MasterConfigTypeDef sMasterConfig = {0};
233    TIM_OC_InitTypeDef sConfigOC = {0};
234
235    htim3.Instance = TIM3;
236    htim3.Init.Prescaler = 84-1;
237    htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
238    htim3.Init.Period = 100-1;
239    htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
240    htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
241    if (HAL_TIM_Base_Init(&htim3) != HAL_OK)
242    {
243      Error_Handler();
244    }
245    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
246    if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK)
247    {
248      Error_Handler();
249    }
250    if (HAL_TIM_PWM_Init(&htim3) != HAL_OK)
251    {
252      Error_Handler();
253    }
254    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
255    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
256    if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
257    {
258      Error_Handler();
259    }
260    sConfigOC.OCMode = TIM_OCMODE_PWM1;
261    sConfigOC.Pulse = 0;
262    sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
263    sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
264    if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
265    {
266      Error_Handler();
267    }
268    if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
269    {
270      Error_Handler();
271    }
272    HAL_TIM_MspPostInit(&htim3);
273
274  }
```

Figure G.1: TIM3

H TIM4

```
281  static void MX_TIM4_Init(void)
282  {
283    TIM_ClockConfigTypeDef sClockSourceConfig = {0};
284    TIM_MasterConfigTypeDef sMasterConfig = {0};
285
286    htim4.Instance = TIM4;
287    htim4.Init.Prescaler = 84-1;
288    htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
289    htim4.Init.Period = 65535;
290    htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
291    htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
292    if (HAL_TIM_Base_Init(&htim4) != HAL_OK)
293    {
294      Error_Handler();
295    }
296    sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
297    if (HAL_TIM_ConfigClockSource(&htim4, &sClockSourceConfig) != HAL_OK)
298    {
299      Error_Handler();
300    }
301    sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
302    sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
303    if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
304    {
305      Error_Handler();
306    }
307
308
309  }
```

Figure H.1: TIM4

I UART

```c
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK)
  {
    Error_Handler();
  }
}
```

Figure I.1: UART

J DMA

```c
335  static void MX_DMA_Init(void)
336  {
337
338    /* DMA controller clock enable */
339    __HAL_RCC_DMA1_CLK_ENABLE();
340
341    /* DMA interrupt init */
342    /* DMA1_Stream5_IRQn interrupt configuration */
343    HAL_NVIC_SetPriority(DMA1_Stream5_IRQn, 5, 0);
344    HAL_NVIC_EnableIRQ(DMA1_Stream5_IRQn);
345
346  }
```

Figure J.1: DMA

K DMA Callback

```
408  void HAL_UART_RxCpltCallback(UART_HandleTypeDef *huart)
409  {
410    HAL_UART_Receive_DMA(&huart2, Rx_1data, sizeof(Rx_1data));                    // Start the DMA again
411    //CHECK FOR WHICH MOTER IS THE MSG YOU SENT//
412    //        70 --> F          66 --> B          85 --> U          68 --> D          83 --> S          f-->102
413    if(Rx_1data[0]==70 || Rx_1data[0]==66 || Rx_1data[0]==85 || Rx_1data[0]==68 ||Rx_1data[0]==83 ||Rx_1data[0]==102 ||Rx_1data[0]==117
414      //FORWARD,BACK,UP,DOWN,STOP,HeadLights off
415
416    {
417      msg=Rx_1data[0];                                      // SET THE VALUE FOR THE SWITCH CASE
418      alarm=0;
419
420    }
421
422    //           82 --> R         76 --> L
423    else if(Rx_1data[0]==82 ||Rx_1data[0]==76 )                            // RIGHT, LEFT MOTER2
424
425    {
426
427      trn=Rx_1data[0];                                      //SET THE VALUE FOR THE SWITCH CASE
428      HAL_GPIO_WritePin(Head_Lights_GPIO_Port, Head_Lights_Pin, 1);
429    }
430  //            65-->A           67-->C          100-->d            99-->c
431    else if(Rx_1data[0]==65 ||Rx_1data[0]==67 ||Rx_1data[0]==100 ||Rx_1data[0]==99 )                          // RIGHT, LEFT MOTER2
432
433      {
434        if( movement_flag==1){
435
436        speed_limit=1;
437        msg=Rx_1data[0];                                      //SET THE VALUE FOR THE SWITCH CASE
438        }
439
440      }
441
442    else
443
444    {
445      Rx_1data[0]=0;                                        // IF YOU SENT GARBAGE ,CLEAN THE BUFFER
446      HAL_GPIO_WritePin(Head_Lights_GPIO_Port, Head_Lights_Pin, 0);
447    }
448
449      // AFTER A BREAK YOU SHOULD TURN OFF THE BREAKLIGTHS SO YOU CAN START AGAIN
450      if (break_flag!=0)
451      {
452
453          HAL_GPIO_WritePin(Break_lights_GPIO_Port, Break_lights_Pin, 0);        // TURN OFF THE BREAK LIGHTS
454          break_flag=0;
455
456      }
```

Figure K.1: DMA Callback Code

L GPIO Initialize

```c
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct = {0};
  /* GPIO Ports Clock Enable */
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOA, LED_Pin|OUT3_Pin, GPIO_PIN_RESET);
  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOC, Flash_Left_Pin|Head_Lights_Pin|OUT1_Pin|OUT2_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin Output Level */
  HAL_GPIO_WritePin(GPIOB, Trig_Pin|Break_lights_Pin|OUT4_Pin|Flash_Right_Pin, GPIO_PIN_RESET);

  /*Configure GPIO pin : Button_Pin */
  GPIO_InitStruct.Pin = Button_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(Button_GPIO_Port, &GPIO_InitStruct);

  /*Configure GPIO pins : LED_Pin OUT3_Pin */
  GPIO_InitStruct.Pin = LED_Pin|OUT3_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);

  /*Configure GPIO pins : Flash_Left_Pin Head_Lights_Pin OUT1_Pin OUT2_Pin */
  GPIO_InitStruct.Pin = Flash_Left_Pin|Head_Lights_Pin|OUT1_Pin|OUT2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);

  /*Configure GPIO pins : Trig_Pin Break_lights_Pin OUT4_Pin Flash_Right_Pin */
  GPIO_InitStruct.Pin = Trig_Pin|Break_lights_Pin|OUT4_Pin|Flash_Right_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);

  /*Configure GPIO pin : Echo_Pin */
  GPIO_InitStruct.Pin = Echo_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  HAL_GPIO_Init(Echo_GPIO_Port, &GPIO_InitStruct);

}
```

Figure L.1: GPIO Initialize

M Switch case 1

```
467  void Cases(void)
468  {
469    switch (msg)
470      {
471
472      // 70 --> F-ORWARD
473        case 70:                                  //Forward rotation
474          HAL_GPIO_WritePin(OUT1_GPIO_Port, OUT1_Pin, 1);            // Enable out1  Determine the rotation
475          HAL_GPIO_WritePin(OUT2_GPIO_Port, OUT2_Pin, 0);          //Enable  out2  Determine the rotation
476          movement_flag=1;
477          alarm=0;
478          flash_right=0;
479          flash_left=0;
480          speed_limit=0;
481          HAL_GPIO_WritePin(Head_Lights_GPIO_Port, Head_Lights_Pin, 1);
482          if(duty1>=100)                           // DUTY CYCLE CAN TAKE I VALUE OVER 100%
483          {
484            duty1=100;
485          }
486          else
487          {
488            duty1=duty1+10;                    //Speed increase
489
490          }
491          break;
492
493    //     66 --> B-ACKWARD
494        case 66:                                  //Reverse rotation
495          HAL_GPIO_WritePin(OUT1_GPIO_Port, OUT1_Pin, 0);          // Enable out1  Determine the rotation
496          HAL_GPIO_WritePin(OUT2_GPIO_Port, OUT2_Pin, 1);          // Enable out2  Determine the rotation
497          movement_flag=1;
498          HAL_GPIO_WritePin(Head_Lights_GPIO_Port, Head_Lights_Pin, 1);
499          flash_right=0;
500          flash_left=0;
501          ultra=0;
502          if(duty1<=0)                             // DUTY CYCLE LOW LIMIT 0 , NO NEGATIVE VALUE
503          {
504
505            duty1=0;
506
507          }
508
509          else
510          {
511
512            duty1=duty1-10;                        //Slow down
513          }
514          break;
```

Figure M.1: Switch case 1

N Switch case 2

```
517       case 85:                                //Speed up
518
519         if (movement_flag==1 && speed_limit==0)
520         {
521
522
523           if(duty1>=100)                          // DUTY CYCLE CAN TAKE I VALUE OVER 100%
524           {
525             duty1=100;
526           }
527           else
528           {
529             duty1=duty1+10;                     //Speed increase
530
531           }
532
533
534         }
535         break;
536
537       // 68 --> D-OWN
538         case 68:                                //Speed down
539
540         if (movement_flag==1){
541
542           if(duty1<=0)                            // DUTY CYCLE LOW LIMIT 0 , NO NEGATIVE VALUE
543           {
544
545             duty1=0;
546
547           }
548
549           else
550           {
551
552             duty1=duty1-10;                       //Slow down
553           }
554         }
555         break;
556
557       // 83 --> S-TOP
558         case 83:                                // Stop
559
560         duty1=0;
561         break_flag=1;
562         movement_flag=0;
563
564         break;
```

Figure N.1: Switch case 2

O Switch case 3

```
558         case 83:                                // Stop
559 
560           duty1=0;
561           break_flag=1;
562           movement_flag=0;
563 
564           break;
565 
566       // 65--> 30km speed limit
567         case 65:                                // 30km Sign
568 
569           duty1=60;
570 
571           break;
572 
573       // 67--> 50km speed limit
574         case 67:                                // 50km Sign
575 
576           duty1=70;
577 
578           break;
579 
580       // 99--> 70km speed limit
581         case 99:                                // 70km Sign
582 
583           duty1=80;
584 
585           break;
586 
587       // 100--> 120km speed limit
588         case 100:                               // 120km Sign
589 
590           duty1=90;
591 
592           break;
593 
594         case 102:                               // Turn Off Head Lights
595 
596           HAL_GPIO_WritePin(Head_Lights_GPIO_Port, Head_Lights_Pin, 0);
597 
598           break;
599 
600             case 117:                                 // Ultrasonic Off
601 
602           HAL_GPIO_WritePin(Head_Lights_GPIO_Port, Head_Lights_Pin, 0);
603           osDelay(10);
604           HAL_GPIO_WritePin(Head_Lights_GPIO_Port, Head_Lights_Pin, 1);
605           ultra=1;
606           break;
607 
```

Figure O.1: Switch case 3

P Turnfunc

```
616  void turnfunc (void)
617  {
618    switch (trn)
619    {
620
621    // 82 --> R-IGHT
622      case 82:
623        flash_right=1;
624        flash_left=0;
625        HAL_GPIO_WritePin(OUT3_GPIO_Port, OUT3_Pin, 1);              // Enable out3  Determine the rotation
626        HAL_GPIO_WritePin(OUT4_GPIO_Port, OUT4_Pin, 0);             //Enable  out4  Determine the rotation
627        duty2=100;
628        trn_flag=1;
629
630        break;
631
632    // 76 --> L-EFT
633      case 76:
634        flash_left=1;
635        flash_right=0;
636        HAL_GPIO_WritePin(OUT3_GPIO_Port, OUT3_Pin, 0);              // Enable out3  Determine the rotation
637        HAL_GPIO_WritePin(OUT4_GPIO_Port, OUT4_Pin, 1);             //Enable  out4  Determine the rotation
638        duty2=100;
639        trn_flag=1;
640
641        break;
642
643    }
644  //  trn=0;
645  }
646  //Delay function only for ultrasonic
647  void usDelay(uint32_t uSec)
648  {
649        if(uSec < 2) uSec = 2;
650        usTIM->ARR = uSec - 1;                          /*sets the value in the auto-reload register*/
651        usTIM->EGR = 1;                                        /*Re-initialises the timer*/
652        usTIM->SR &= ~1;                                    //Resets the flag
653        usTIM->CR1 |= 1;                                    //Enables the counter
654        while((usTIM->SR&0x0001) != 1);
655        usTIM->SR &= ~(0x0001);
656  }
```

Figure P.1: Turnfunc

Q Ultrasonic Routine 1

```
667  void StartUltrasonic01(void *argument)
668  {
669    /* USER CODE BEGIN 5 */
670    /* Infinite loop */
671
672  uint32_t numTicks = 0;
673
674    for(;;)
675    {
676
677      if(ultra==0)
678      {
679
680
681        HAL_GPIO_WritePin(Trig_GPIO_Port, Trig_Pin, GPIO_PIN_RESET);                //Set TRIG to LOW for few uSec
682        usDelay(3);
683
684        //*** START Ultrasonic measure routine ***//
685                                                                  //1. Output 10 usec TRIG
686        HAL_GPIO_WritePin(Trig_GPIO_Port, Trig_Pin, GPIO_PIN_SET);
687        usDelay(10);
688        HAL_GPIO_WritePin(Trig_GPIO_Port, Trig_Pin, GPIO_PIN_RESET);
689
690                                                                  //2. Wait for ECHO pin rising edge
691        while(HAL_GPIO_ReadPin(Echo_GPIO_Port, Echo_Pin) == GPIO_PIN_RESET);
692
693                                                                  //3. Start measuring ECHO pulse width in usec
694        numTicks = 0;
695        while(HAL_GPIO_ReadPin(Echo_GPIO_Port, Echo_Pin) == GPIO_PIN_SET)
696        {
697          numTicks++;
698          usDelay(2);                                          // 2.8usec
699        };
700
701          distance = (numTicks + 0.0f)*2.8*speedOfSound;                    //4. Estimate distance in cm
702
703
704      }
705
706        osDelay(20);
707
708    }
709
710    /* USER CODE END 5 */
711  }
```

Figure Q.1: Ultrasonic Routine 1

R Ultrasonic Routine 2

```
719  void StartUltrasonic02(void *argument)
720  {
721    /* USER CODE BEGIN StartUltrasonic02 */
722    /* Infinite loop */
723    for(;;)
724    {
725      if (distance<15 && duty1!=0 && ultra==0)                    //Check for a object
726      {
727        duty1=0;
728        alarm=1;
729
730      }
731
732      osDelay(1);
733    }
734    /* USER CODE END StartUltrasonic02 */
```

Figure R.1: Ultrasonic Routine 2

S First Motor Routine

```
743  void StartMotor1(void *argument)
744  {
745    /* USER CODE BEGIN StartMotor1 */
746    /* Infinite loop */
747
748    for(;;)
749    {
750      Cases();
751      HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1);         //Enable PWM motor1
752
753      htim3.Instance->CCR1=duty1;               //Set the value of the Duty
754
755      if(break_flag!=0)
756
757      {
758
759        HAL_GPIO_WritePin(Break_lights_GPIO_Port,Break_lights_Pin, 1);
760  //      alarm=1;
761
762      }
763
764      msg=0;
765        osDelay(1);
766
767    }
768    /* USER CODE END StartMotor1 */
769  }
```

Figure S.1: First Motor Routine

T Second Motor Routine

```
777  void StartMotor2(void *argument)
778  {
779    /* USER CODE BEGIN StartMotor2 */
780    /* Infinite loop */
781    for(;;)
782    {
783
784     turnfunc ();
785
786     if(trn_flag==1)
787     {
788
789      HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2);         //Enable PWM motor2
790      htim3.Instance->CCR2=duty2;                 //Set the value of the Duty
791
792      osDelay(100);
793      duty2=0;
794      htim3.Instance->CCR2=0;
795      trn_flag=0;
796      trn=0;
797     }
798
799  //   flash_left=0;
800  //   flash_right=0;
801
802    }
803    /* USER CODE END StartMotor2 */
804  }
```

Figure T.1: Second Motor Routine

U Alarm Routine

```
813  void Flash_Alarm(void *argument)
814  {
815    /* USER CODE BEGIN Flash_Alarm */
816    /* Infinite loop */
817    for(;;)
818    {
819      if(flash_left==1 && break_flag==0)
820
821          {
822
823            HAL_GPIO_WritePin(Flash_Left_GPIO_Port, Flash_Left_Pin, 1);
824            osDelay(100);
825            HAL_GPIO_WritePin(Flash_Left_GPIO_Port, Flash_Left_Pin, 0);
826            osDelay(100);
827
828
829
830          }
831
832      else if  (flash_right==1 && break_flag==0)
833
834          {
835
836
837            HAL_GPIO_WritePin(Flash_Right_GPIO_Port, Flash_Right_Pin, 1);
838            osDelay(100);
839            HAL_GPIO_WritePin(Flash_Right_GPIO_Port, Flash_Right_Pin, 0);
840            osDelay(100);
841
842          }
843      else if  (alarm==1)
844
845              {
846
847
848          HAL_GPIO_WritePin(Flash_Right_GPIO_Port, Flash_Right_Pin, 1);
849          HAL_GPIO_WritePin(Flash_Left_GPIO_Port, Flash_Left_Pin, 1);
850          osDelay(100);
851          HAL_GPIO_WritePin(Flash_Right_GPIO_Port, Flash_Right_Pin, 0);
852          HAL_GPIO_WritePin(Flash_Left_GPIO_Port, Flash_Left_Pin, 0);
853          osDelay(100);
854
855              }
856
857    }
858    /* USER CODE END Flash_Alarm */
859  }
860
```

Figure U.1: STM32 Alarm Routine

V Import Libraries

```python
import subprocess
import serial
import time
import os
import sys
import glob
import argparse
import warnings
from Predict import l

instr_tuple = ("F", "B", "S", "U", "D", "R", "L", "f", "u")
# Menu Function
def menu():
    print("----------------------------")
    print("[F] Start the engine")
    print("[S] Hit the Break")
    print("[B] Backward")
    print("[U] Speed up")
    print("[D] Speed down")
    print("[R] Turn Right")
    print("[L] Turn Left")
    print("[f] Head Lights off")
    print("[u] Ultrasonic off")
    print("----------------------------")
```

Figure V.1: Import Libraries

W Check the input

```python
def num_check(instr_tuple,num):
    counter = 0
    for x in instr_tuple:
        if x == num:

            break
        else:
            counter += 1
            if counter == 8:
                os.system('cls')
                warnings.warn("Wrong Instruction")
                menu()
```

Figure W.1: Check the input

X While

```
menu()
if __name__ == '__main__':
    print("Available Serial Ports:")
    print(serial_ports())

COM = input('Choose COM port:')
stm32 = serial.Serial(port=COM, baudrate=115200, timeout=.01)

while True:
    subprocess.run("Predict.py")
    orders(num,l)
    num = input("Give instruction:")
    num_check(instr_tuple, num)
    value = write_read(num)
```

Figure X.1: While

Y Predict 1

```python
import numpy as np
import cv2 as cv
import tensorflow as tf
from tensorflow import keras



model = keras.models.load_model('C:/Users/antonis.karag/Desktop/Thesis/backup_thesis/python_serial/model (1).h5')

frameWidth = 640  # CAMERA RESOLUTION
frameHeight = 480
brightness = 180
threshold = 0.75  # PROBABLITY THRESHOLD
font = cv.FONT_HERSHEY_SIMPLEX
##############################################
# SETUP THE VIDEO CAMERA
cap = cv.VideoCapture(2)

def getClassName(classNo):
    if classNo == 0:
        return 'Speed Limit 30 km/h'
    elif classNo == 1:
        return 'Speed Limit 50 km/h'
    elif classNo == 2:
        return 'Speed Limit 70 km/h'
    elif classNo == 3:
        return 'Speed Limit 120 km/h'
    elif classNo == 4:
        return 'Stop'
    elif classNo == 5:
        return 'Turn Right'
    elif classNo == 6:
        return 'Turn Left'
```

Figure Y.1: Predict 1

Z Predict 2

```
while True:

    ret, img = cap.read()
    img = np.asarray(img)
    img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    cv.imshow("Processed Image", img)
    img = cv.equalizeHist(img)
    img = cv.GaussianBlur(img, (5, 5), 0)
    img = img / 255

    img = cv.resize(img, (32, 32))
    img = img.reshape(1, 32, 32, 1)
    cv.putText(img, "CLASS: ", (20, 35), font, 0.75, (0, 0, 255), 2, cv.LINE_AA)
    cv.putText(img, "PROBABILITY: ", (20, 75), font, 0.75, (0, 0, 255), 2, cv.LINE_AA)
    predictions = model.predict(img)
    # plt.imshow(img)
    # plt.show()
    # PROCESS IMAGE
    classes = np.argmax(predictions, axis=1)
    # print(classes)
    # print(str(classes.all()))
    # print(getClassName(classes))
    probabilityValue = np.amax(predictions)
    if probabilityValue > threshold:
        # print(getCalssName(classIndex))
        # cv.putText(img, str(classes) + " " + str(getClassName(classes)), (120, 35), f
        #             cv.LINE_AA)
        # cv.putText(img, str(round(probabilityValue * 100, 2)) + "%", (180, 75), font,
        # cv.imshow("Result", img)
        print(getClassName(classes))
        l = getClassName(classes)
    if cv.waitKey(1) and 0xFF == ord('q'):
        break
```

Figure Z.1: Predict 2