University POLITEHNICA of Bucharest
Faculty of Electronics, Telecommunications and Information Technology

**Autonomous R/C Car**

# Diploma Thesis

**Submitted in partial fulfillment of the requirements for the degree of** *Engineer*
**in the domain** *Electronics, Telecommunications and Information Technology*
**study program** *Applied Electronics*

**Thesis adivsor(s)**
**Iulian Dumitru Năstac**

**Student**
**Mihnea Manea**

**Year 2020**

## Statement of Academic Honesty

I hereby declare that the thesis *Autonomous R/C Car*, submitted to the Faculty of Electronics, Telecommunications and Information Technology in partial fulfillment of the requirements for the degree of *Engineer* in the domain Electronic Engineering, study program *Applied Electronics* is written by myself and was never submitted to any other faculty or higher learning institution in Romania or any other country. I declare that all information sources I used, including the ones I found on the Internet, are properly cited in the thesis as bibliographical references. Text fragments cited "as is" or translated from other languages are written between quotes and are referenced to the source. Reformulation using different words of a certain text is also properly referenced. I understand plagiarism constitutes an offence punishable by law. I declare that all the results I present as coming from simulations and measurements I performed, together with the procedures used to obtain them, are real and indeed come from the respective simulations and measurements. I understand that data faking is an offence punishable according to the University regulations.

Bucharest, Iunie 2020.
Student: Mihnea Manea

.....

# Content

# Figure List

# Table List

# Acronym List

Euro NCAP - European New Car Assessment Programme
ADAS = Advanced Driver-Assistance Systems
LDW = Lane Departure Warning
LKA = Lane Keep Assist
LCA = Lane Centering Assist
TSR = Traffic Sign Recognition
CAS = Collision Avoidance System
GPS = Global Positioning System
AEB = Autonomous Emergency Breaking
EU = European Union
UN ECE = United Nations Economic Commission for Europe
I/O = Input/Output
USB = Universal Serial Bus
IDE = Integrated development environment
SBC = Single Board Computer
GPIO = General-purpose input/output
WiFi = Wireless Fidelity
MB = Mega Bytes
RAM = Random Access Memory
HDMI = High-Definition Multimedia Interface
ARM = Advanced RISC Machine
OS = Operating System
SD = Secure Digital
PC = Personal Computer
OpenCV = Open Source Computer Vision Library
BSD = Berkeley Software Distribution
API = Application Programming Interface
i.e = id est
GUI = Graphical User Interface
LED = Light-Emitting Diodes
fig. = Figure
PWM = Pulse With Modulation
GND = ground
CPU = Central Processing Unit
BGR = Blue Green Red
RGB = Red Green Blue
FPS = Frames Per Second
RoI = Region Of Interest

# Chapter 1
# Introduction

Advanced driver-assistance systems represent a very important field in the automotive industry nowadays. More and more car manufactures implement this type of systems in their cars. Moreover, some of them even introduced a certain level of autonomy in their products, for example Tesla. The authorities started to introduce mandatory ADAS safety systems to ensure better transport security, so it is correct to assume that there is a continuity in time for these systems. In addition to that, this is a relatively new automotive field, so there is much to discover and develop. The future seems to be going in this direction, in the way of autonomous cars. Along with this motivation, my passion for cars made me choose this project.

There are a lot of types of ADAS functions that can be implemented on a car, from the basic ones, like anti-lock braking systems and electronic stability control to advanced ones such as lane departure warning, adaptive cruise control and traffic sign recognition. These functions work based on a series on inputs from different sensors. These sensors can be LIDAR, video cameras, Hall-effect sensors, light sensors and many more. The newest functions are based on innovative technologies, LiDAR and computer vision using cameras. This project focuses only on computer vision and machine learning, using only a camera, not other sensors.

The aim of this paper is to integrate some ADAS features in oder to achieve a baseline degree of autonomy into a toy car. This car is capable of recognizing and keeping lanes, recognizing stop signs and other cars in front of it. The project uses a custom power train, everything being assembled from modules, specifically made to fit the purpose of the project. The methodology used was a step by step approach, each function introduced was ordered by complexity. The result highlights the objectives of the project.

In Figure 4.1, the final car is presented.

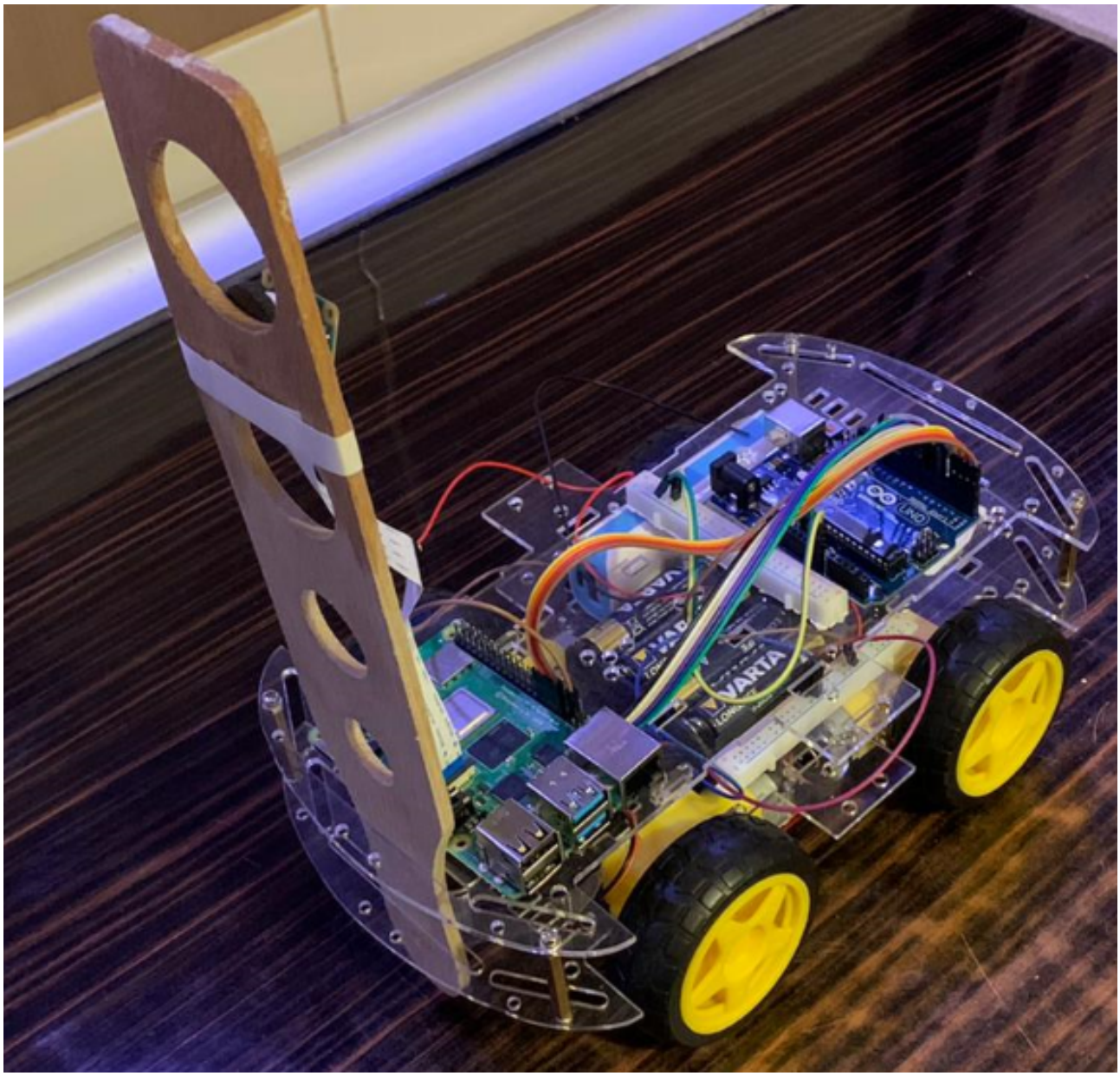


Figure 1.1: Final image of the car

# Chapter 2
# Implemented Functions

## 2.1 Moving Functions

The moving function is the building block in this project. In order for the project to be able to implement other ADAS functions, this basic moving ability had to be implemented. The car is able to move forward, backward, stop, left and right using four motors. More about the hardware implementation in the chapter 2. The simple principle behind this function is to steer the car without the need of a servo motor or complicated mechanics for the four wheels. It is based on the idea of steering a tank. When the car needs to steer, left for example, the wheels on the opposite side (in our example the right side) will spin at a higher rate than the wheels of the other side (left side). Differential spinning rates of the two sides assures different steering angles of the car.

The substantial advantage of this type of implementation is its simplicity along with its reliability. More moving parts and greater mechanical complexity means a higher breaking rate. Because of the homogeneity, the code is easier to develop and the cost of hardware is significantly lower. More about code implementation in the Chapter 4.

Unfortunately, this type of steering has some disadvantages, being a simple mechanism also means that the steering angles cannot be fine controlled as in a classical servo steering used in real cars. This seems to be a problem if the car is supposed to have great speed and agility, however, this is not the scope of this project. One other disadvantage is that this type of implementation requires the car to have four motors, which increases the power consumption and the weight. However, this means better traction for a four-wheel drive vehicle.

The stop function is consequential, as it serves other more complex functions in the process of making the car autonomous. It has roles in the sign detection and object detection, these functions will be detailed below.

## 2.2 Lane Keep Function

In real world cars the lane departure warning system is a feature designed to warn the driver when the car moves out of its lane unintentionally. The system knows there is an intention of changing the lane when the turn signal is on. This system is designed to decrease the number of accidents because of driver error. National Highway Traffic Safety Administration (NHTSA) began studying whether to mandate lane departure warning systems and frontal collision warning systems on automobiles[8].

There are multiple types of such systems depending of their active intervention. The Lane Departure Warning (LDW) system has no control over the steering. It just warns the driver when the car steers out of its lane, the warnings can be audible, visual, or even steering wheel

vibrations. This is the most basic lane recognition system in the car industry.

Lane keeping assist (LKA/LKS) adds a new function to LDW. This function allows the system to take control of the steering in the potential case that the vehicle is moving. In case of an unaware driver, with no input at the steering wheel, the system works well in the first few seconds, keeping the car in the lane. After some time with no steering input, the system will bounce the car away from the lane edge. This is not the behavior wanted, so a new, more advanced system appeared, which can keep the car centered in lane and relieve the driver of the task of steering.

This system is called lane centering assist (LCA), also called auto steer in the industry. This system is used by Tesla to implement their autopilot feature. Together with other functions this system allows implementation of autonomous cars. The limitations of this system appear when a tight turn follows or the road markings are not visible, in these situations the driver will be asked to take control of the car. The assistant works based on a camera mounted on the top of the windshield. All of these systems are considered to be ADAS implementations. Figure 2.1 shows the principle of lane assist systems.

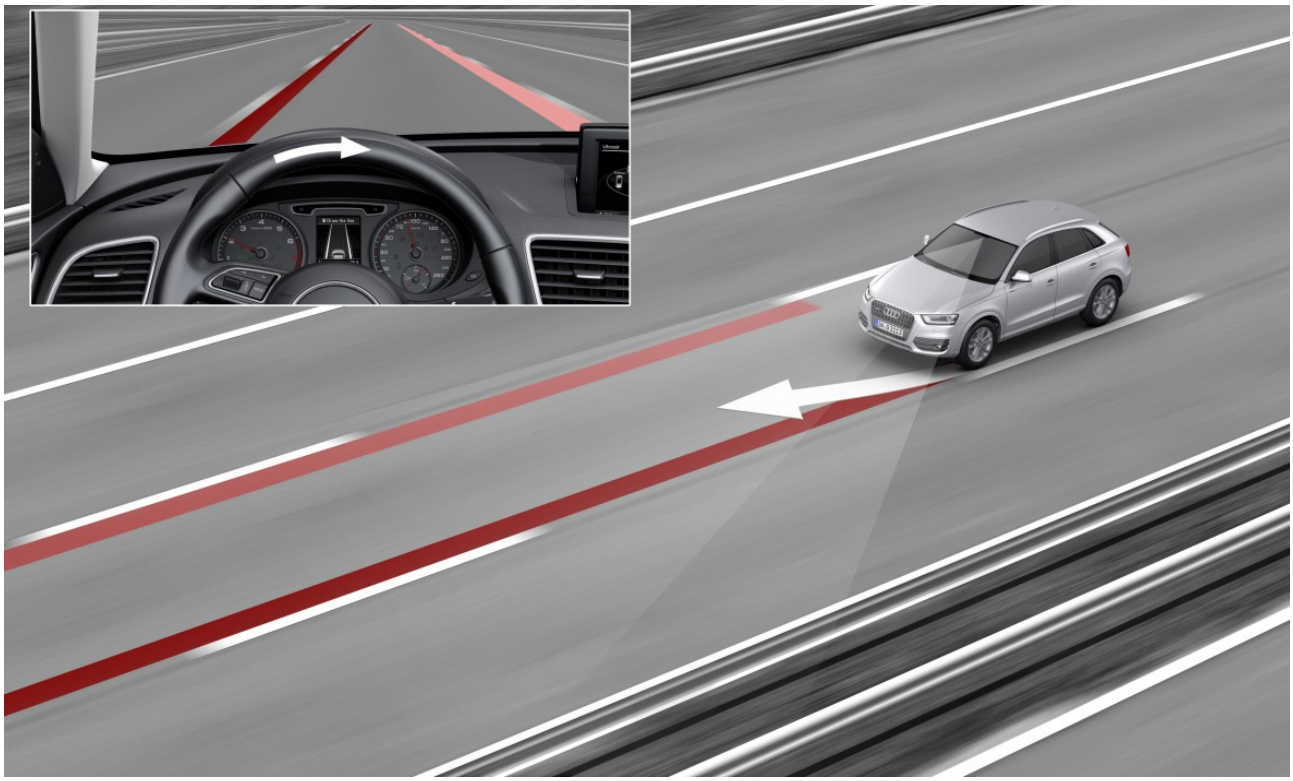

Figure 2.1: Lane centering assist[1]

The lane keep function is inspired by LCA and together with the other functions described in this chapter will lead to a degree of autonomy of the car. The system's principle is relatively simple, it recognizes the lanes printed on the project's test track and tries to keep the car in the center of these lanes. It is very important that the lanes are clear and visible, for the system to work. Although the LCA automotive systems are implemented using Canny edge detection and Hough transform, our project will only use Canny edge detection, as the steering method differs and the speed is lower than on a conventional car. More about the hardware and software implementation of this feature are written in chapter 3 and 4.

## 2.3 Traffic Sign Detection Function

Traffic sign recognition, also known as TSR is a new technology that appeared in the automotive world. It helps the driver keeping track of the signs present on the roads. As many new roads are being built and more traffic signs are mounted on the roads, the experience of driving and being aware of all the signs can be overwhelming, especially for older drivers.

This ADAS system was invented to help the driver by showing him the signs that he just passed by. For example, the system can show the speed limit of that portion of the road, if the car passed near a speed limit sign indicating that restriction. This system is rather complex as it requires cameras, machine learning and computer vision algorithms. TSR function can be seen in the next diagram (Figure 2.2).

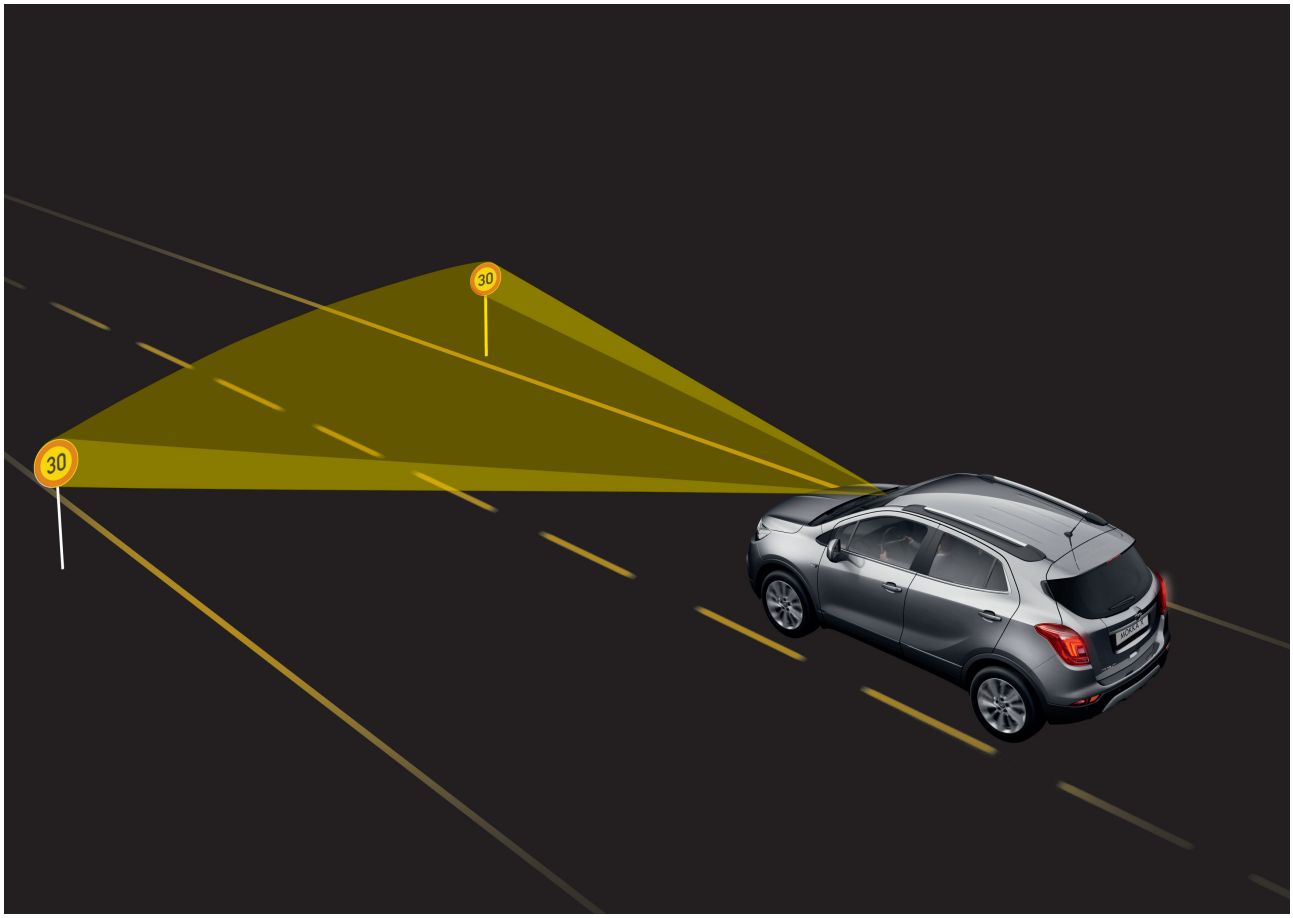


Figure 2.2: Traffic Sign Recognition[2]

In this project, a "Stop" sing recognition system is implemented. This is inspired by TSR but it only has the ability to detect "Stop" signs. The car detects the stop sign and computes the distance to it. Once the threshold distance is met, the car stops for an amount of time and then continues its journey. More about hardware, distance threshold, time value and algorithms in chapters 4 and 5.

## 2.4 Obstacle Detection

The collision avoidance system (CAS), commonly known as a pre-crash system, is an automotive feature designed to avoid or reduce the severity of a collision[9].

The system works in a simple way. The system monitors the speed of the vehicle and the obstacles in front of it (other cars) and their speed. When the car approaches too fast to an obstacle, the system will intervene and avoid a crash. First, a warning will be raised to the driver. In the case of no respone, the system will take action. There are several sensors used for achieving obstacle detection: radars, LIDARs, cameras or even GPS sensors. Pedestrians can also be detected with this system.

Collision avoidance system range from widespread systems mandatory in some countries, such as autonomous emergency braking (AEB) in the EU, agreements between automakers and safety officials to make crash avoidance systems eventually standard, such as in the United States, to research projects including some manufacturer specific devices[10].

Advanced emergency braking system (AEBS), as defined by UN ECE regulation 131, is considered as: a system that can automatically detect a potential forward collision and activate the vehicle braking system to decelerate the vehicle with the purpose of avoiding or mitigating a collision[11].

According to Euro NCAP, AEB has three characteristics[12]:

- Autonomous: the system acts independently of the driver to avoid or mitigate the accident.
- Emergency: the system will intervene only in a critical situation.
- Braking: the system tries to avoid the accident by applying the brakes.

Time-to-collision could be a way to choose which avoidance method (braking or steering) is best appropriate[13]. . This means that the AEB systems that are currently on the market use both steering and braking to avoid a crash. The function is illustrated in the Figure 2.3.

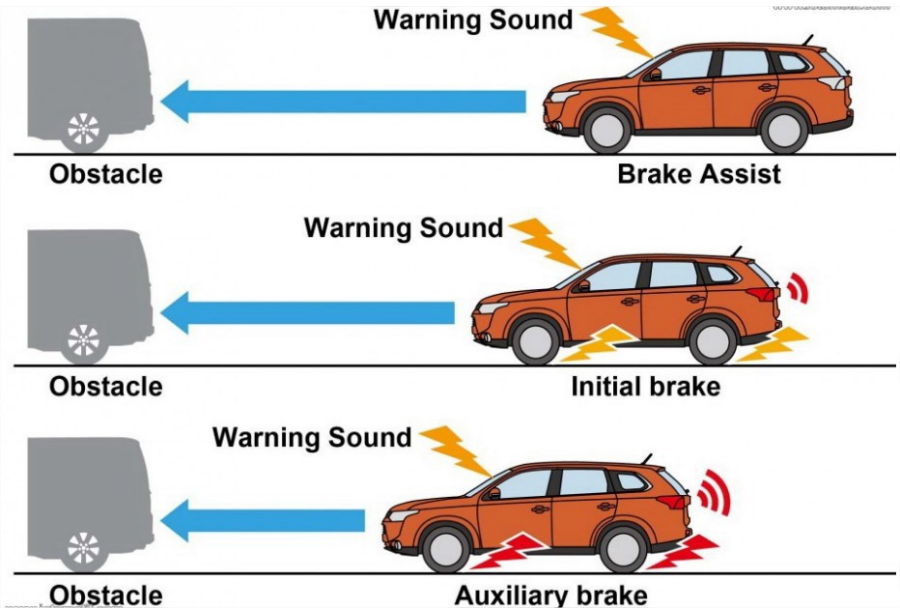


Figure 2.3: Autonomous Emergency Breaking[3]

In this project, the object detection function is inspired by the systems described above. The feature acts as an AEB system which only has the ability to break in order to avoid the

crash. It is solely based on a camera, computer vision and machine learning, no other sensors or technologies are implied. It functions in a simple way, similarly to the „Traffic sign detection" function. When the system detects the obstacle, it computes the distance to it and when the threshold distanced is reached, the car will stop, it will not move unless the object (the car in front of it) moves. The car is trained to recognize another toy car in front of it, however it has the ability to only see the car model that was trained for. The function can be optimised to recognize multiple models of vehicles if trained on a larger dataset. For the demonstration of this function, only one type of car obstacle has been trained. More technical details are given in chapters 3 and 4.

# Chapter 3
# Technologies Used In Project

## 3.1 Arduino

Arduino is an open-source company which is producing both micro-controllers and the software part required for programming the board. Besides that, Arduino has an active community of developers who create and supervise projects using Arduino. This means that this project can be continued and scaled to a larger deployment.

The board is a simple micro-controller which uses a variety of controllers and microprocessors from different manufacturers, for example Atmel. They can range from 8-bit micro-controllers up to 32-bit. The general advantage of an Arduino is that it has a multitude of I/O ports ranging from analog to digital. These pins can be interfaced to various expansion modules (shields) or other circuits (breadboards). It also has a universal serial bus (known as USB) which helps the user to connect the micro-controller to a computer and load the program. There are several types of Arduino boards: Arduino UNO, Arduino Mega, Arduino Pro and so on. The programming environment uses C or C++ programming language and the company also provides an IDE perfectly compatible with all the boards. This board, represented in Figure 3.1 is used for simple controlling tasks in this project, not image processing or other intensive computing.

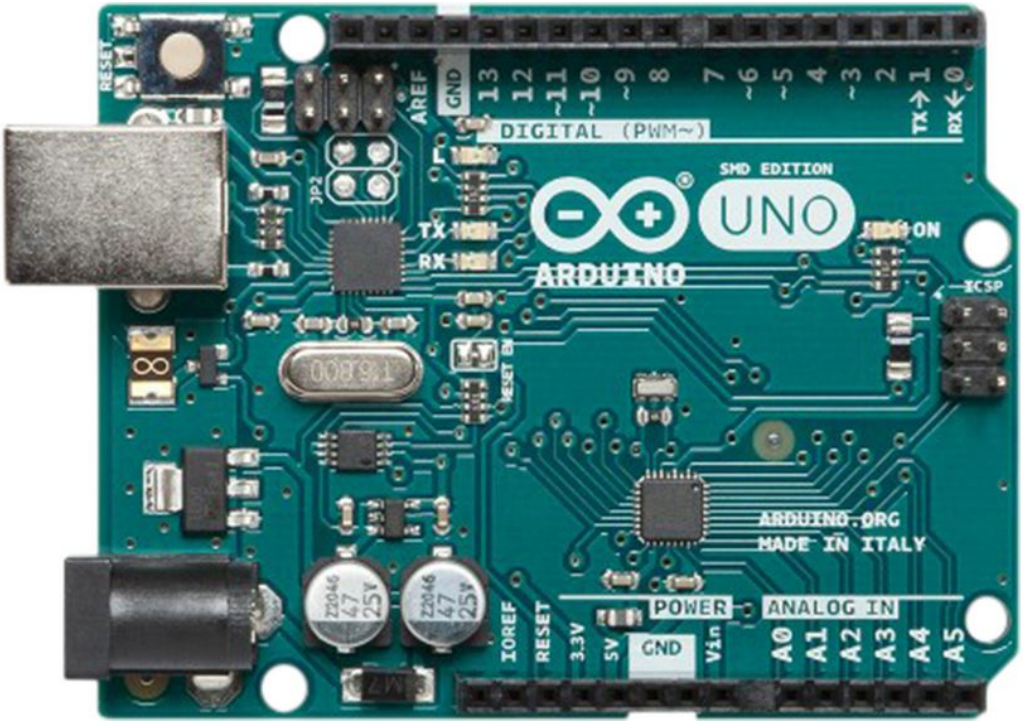


Figure 3.1: Arduino UNO Board[4]

## 3.2 Raspberry Pi

Raspberry Pi is a single-board computer (Sbc) of small dimensions created by Raspberry Pi Foundation. The aim of this project was to create a cheap small product capable of advanced programming[14]. The "pi" from its name comes from the initial developing programming language called Python.

Although this computer is slower than a desktop personal computer or a laptop, it still has the power to run some impressive tasks. In chapter 4 it will be shown the multitude of tasks this board can handle. Raspberry Pi has a special characteristic: it has General-Purpose Input/Output (GPIO). This expands the capabilities of the mini computer to a very powerful micro-controller. The board also has an integrated WiFi chip and can connect to the internet via wireless.

Depending on the model, the board can have between 258 MB and 8GB of RAM. The Raspberry Pi 4 B+ used in this project also has an USB type C input, used for power in mainly, and an HDMI port for connecting it to a monitor or TV. It also has a camera input for connecting a wide range of cameras to the board. The Sbc has an ARM cortex processor similar to the processors found in smartphones. There are several types of Raspberry Pi boards: Raspberry Pi 1 A, Raspberry Pi 1 B, Raspberry Pi 2B, Raspberry Pi 3 B, Raspberry Pi 4 B+ and so on.

The optimized operation system for a Raspberry Pi is Raspbian, which is derived from Debian. It is a Linux distribution, which makes it perfectly suitable for programming and developing software. The official OS must be copied to an SD card and then inserted in the board. The Raspbian OS comes with a lot of useful applications, some of them also used in this project. The Geany IDE and RealVNC application played a crucial part in this project's success. Geany is used as a programming environment whereas RealVNC allows the user to connect remotely to the board's desktop from a PC or a smartphone (iOS and Android), similarly to Team Viewer. This allows for faster testing of the projects without having to plug the board into a monitor through HDMI. Figure 3.2 shows a representation of a Raspberry Pi computer.


Figure 3.2: RaspberryPi Board[5]

## 3.3 C++

C++ is very known programming language used in many applications, especially in embedded ones which require efficiency and speed. C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation. It is almost always implemented as a compiled language, and many vendors provide C++ compilers, including the Free Software Foundation, LLVM, Microsoft, Intel, Oracle, and IBM, so it is available on many platforms[15].

## 3.4 OpenCV

OpenCV (Open Source Computer Vision Library) is an open-source BSD-licensed library that includes several hundreds of computer vision algorithms. The document describes the so-called OpenCV 2.x API, which is essentially a C++ API, as opposed to the C-based OpenCV 1.x API (C API is deprecated and not tested with "C" compiler since OpenCV 2.4 releases)[16].

This library is used in almost all projects that use computer vision. It is a very powerful tool capable of computing real time image processing algorithms. This technology represents the core of these projects, as the main computer vision related functions are implemented using it.

## 3.5 Haar Feature-Based Cascade Classifier for Object Detection

The object detector described below has been initially proposed by Paul Viola [Viola01] and improved by Rainer Lienhart [Lienhart02]. First, a classifier (namely a cascade of boosted classifiers working with Haar-like features) is trained with a few hundred sample views of a particular object (i.e., a face or a car), called positive examples, that are scaled to the same size (say, 20x20), and negative examples - arbitrary images of the same size[6].

After a classifier is trained, it can be applied to a region of interest (of the same size as used during the training) in an input image. The classifier outputs a "1" if the region is likely to show the object (i.e., face/car), and "0" otherwise. To search for the object in the whole image one can move the search window across the image and check every location using the classifier. The classifier is designed so that it can be easily "resized" in order to be able to find the objects of interest at different sizes, which is more efficient than resizing the image itself. So, to find an object of an unknown size in the image the scan procedure should be done several times at different scales[6].

The word "cascade" in the classifier name means that the resultant classifier consists of several simpler classifiers (stages) that are applied subsequently to a region of interest until at some stage the candidate is rejected or all the stages are passed. The word "boosted" means that the classifiers at every stage of the cascade are complex themselves and they are built out of basic classifiers using one of four different boosting techniques (weighted voting). The basic classifiers are decision-tree classifiers with at least 2 leaves. Haar-like features are the input to the basic classifiers, and are calculated as described below. The current algorithm uses the following Haar-like features[6]. Figure 3.3 describes some features of the algorithm.

The feature used in a particular classifier is specified by its shape (1a, 2b etc.), position within the region of interest and the scale (this scale is not the same as the scale used at the detection stage, though these two scales are multiplied). For example, in the case of the third line feature (2c) the response is calculated as the difference between the sum of image pixels under the rectangle covering the whole feature (including the two white stripes and the black stripe in the middle) and the sum of the image pixels under the black stripe multiplied by 3 in order to compensate for the differences in the size of areas. The sums of pixel values over a rectangular regions are calculated rapidly using integral images[6].

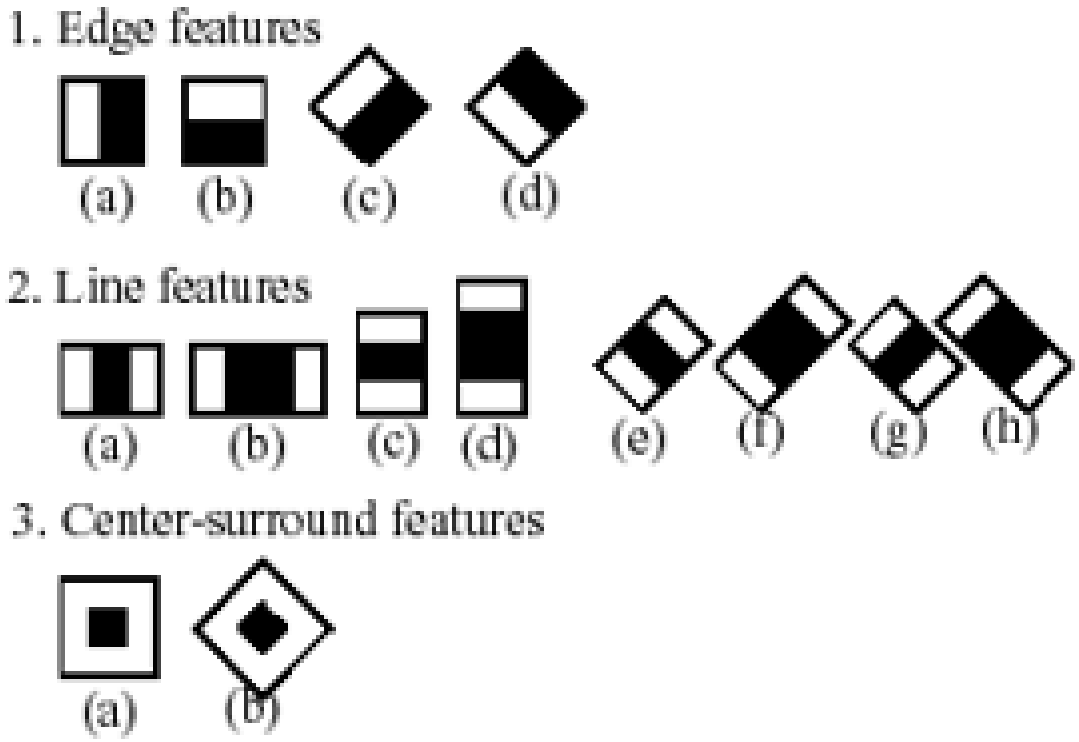


Figure 3.3: Cascade Classifier[6]

## 3.6 Cascade Trainer GUI

Cascade Trainer GUI is a program that can be used to train, test and improve cascade classifier models. This tool basically implements the cascade classifier training form open CV into a graphical user interface. In the project, the Haar feature based cascade classifier was implemented[17].

# Chapter 4
# Hardware

## 4.1 Hardware Components

The project required numerous hardware components, from simpler ones for the power train, to more complicated one for object detection. Here is the list of them all:

**Development tools**

- PC, preferably laptop
- Smartphone
- Multimeter
- LED lamps (for working and testing in the night)
- Soldering iron
- USB wires
- Screwdrivers
- Tweezers
- Scissors

**Car (Figure 4.1)**

- Chassis
- Four wheels
- Four 12V DC motors
- L298n motor driver
- Four AA batteries
- Two external power banks, one capable of 12V output
- 12V power adapter (for testing)
- USB wires, normal wires
- Breadboard

- Arduino UNO R3
- Raspberry Pi 4 B+, 2GB RAM
- Raspberry Pi camera
- Raspberry Pi camera wire
- Wood stick

**Test track (Figure 4.2)**

- Blanket with road like color
- White stripes
- Small stop sign
- Small car (1969 Ford Mustang replica)

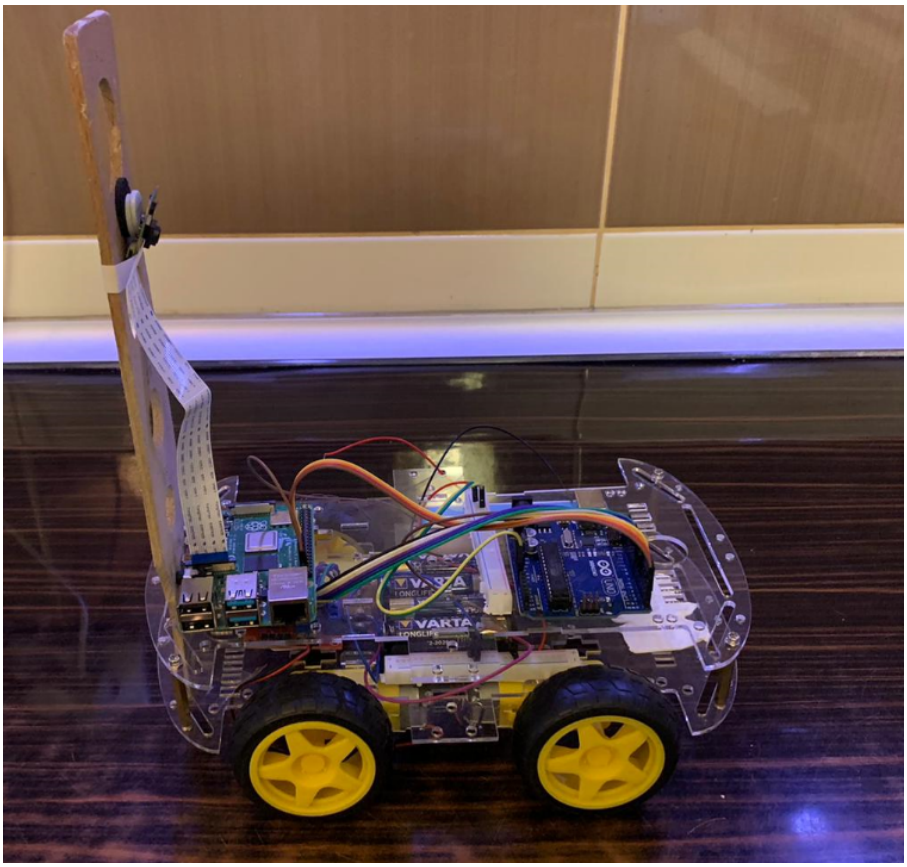


Figure 4.1: Car

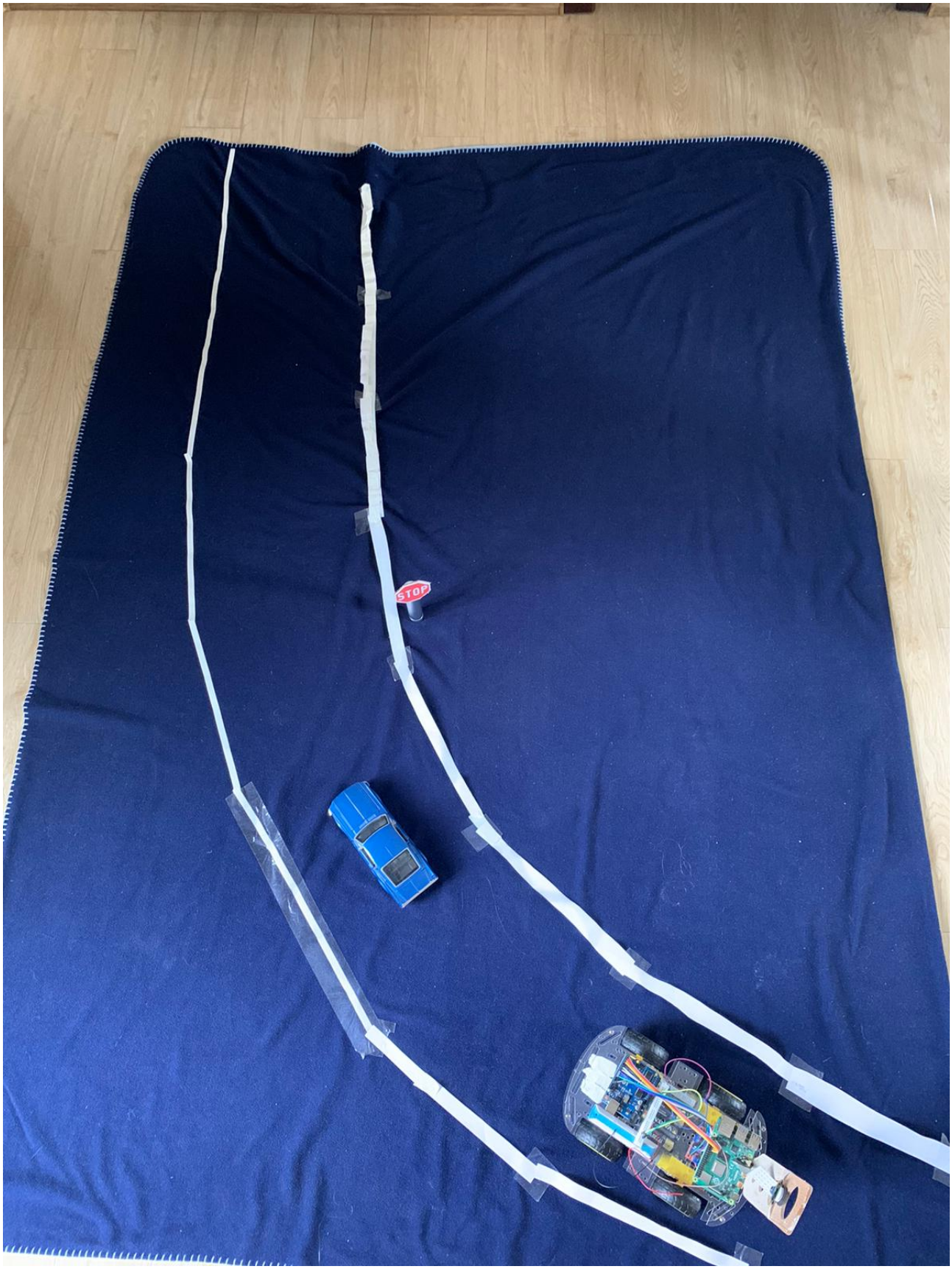


Figure 4.2: Track

## 4.2 Power Train Connections

For an easier understanding of the wiring refer to the figure Figure 4.3 and the electrical scheme Figure 4.4.

The power train of this project car is mainly composed of four 12V DC motors and one L298n dual H-bridge driver. As it can be seen from the scheme, the motors are placed on the extremities of the car and are coupled two by two. This means that the motors on one side are connected together, and then connected to the driver. A parallel connection is achieved. This is done because they need to act as a single motor and have the same command. For example, if the car needs to take a right turn, both of the left side motors will spin at the same rate and in the same direction, while the motors on the right side will have a slower or zero spinning rate.

The connection is also needed by the L298n driver's output pins, as it is only able to control two motors. The motors are connected to the driver in the following way: left-side motors are connected to OUT3 and OUT4 pins and right-side motors to the OUT1 and OUT2 pins, see Figure 4.4. The type of connection design that allows two motors to behave like a single motor assures a straightforward way for a four-wheel drive.

The L298n motor driver is powered by a 12V external battery or by an 12V wall socket adapter for testing. It is connected through the 12V pin (VSS) and ground pin. Control of the driver and the motors is done by the slave device (Arduino). The pins of the driver which connect it to the Arduino are IN1, IN2, ENA (corresponding to the right-side motors) and IN3, IN4 and ENB (corresponding to the left-side motors), see Figure 4.4. IN pins control the direction of the motors, while the EN pins control the spin rate of the motor through PWM.

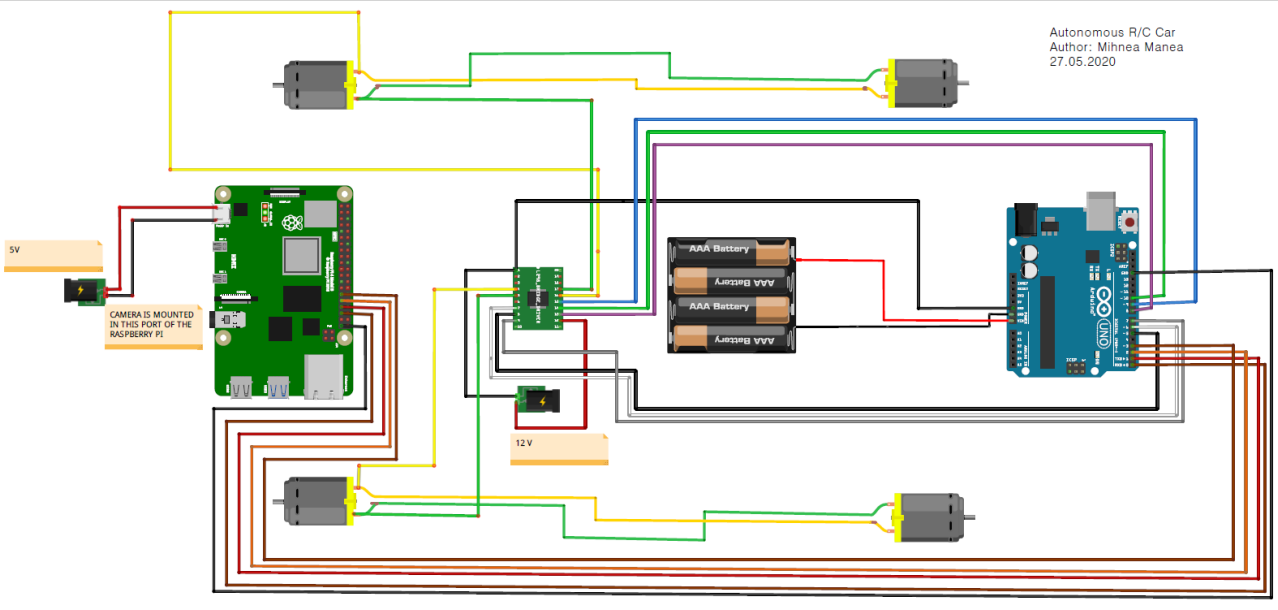


Figure 4.3: Wiring Diagram

## 4.3 Slave Device Setup (Arduino)

As the title of this subchapter suggests, the device used as a slave in our project in an Arduino UNO R3. The device is used to control the drivers and the motors through the Moving

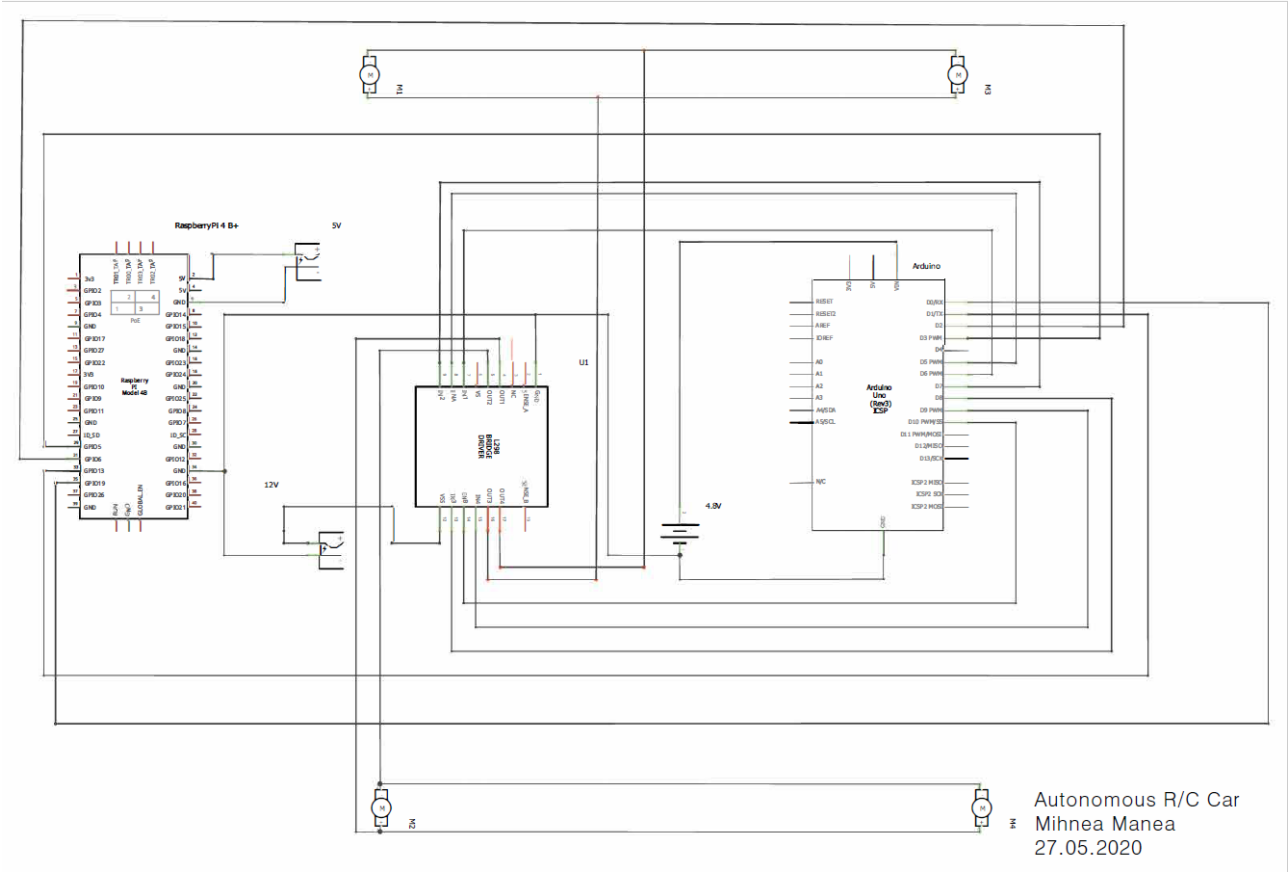


Figure 4.4: Electrical Scheme

functions. This device is further connected to a master one (Raspberry Pi).

The Arduino board is powered up by a set of four AA batteries mounted on the chassis and connected to 5V and GND pins of the Arduino.

It is connected to the driver using the digital pins ranging from digital pin 5 to digital pin 10. The first three pins are used for controlling the right-side motors while the others for the left ones. It is mandatory that the pins 5 and 10, which are connected to the EN pins of the driver, to be PWM compatible, see Figure 4.4. The enabling of the motors, namely the spinning rate or the voltage applied at their inputs, must be controlled with PWM. The Arduino has also a GND pin connected to the ground of the driver (it is mandatory to have a common ground between the driver and the slave device in order for the commands to work), see Figure 4.4.

The other connections of this board are with the master device (Raspberry Pi). They are done through pins D0, D1, D2 and D3, see Figure 4.4. These pins are also digital and a problem may raise when one tries to program the Arduino with a PC through USB while the master device is powered up and connected. The problem comes from the fact that pins D0 and D1 are also RX and TX pins, see Figure 4.3. If one wants to program the board, he or she should check that the master device is either powered off or disconnected from D0 and D1 pins. Arduino also has a common ground (GND pin) with the master device in order for the connections to work, this can be seen on the Figure 4.4. In fact, if one sees the whole picture, all the elements have one common ground.

## 4.4 Master Device Setup (Raspberry Pi)

The brain of this project is represented by the Raspberry Pi computer. The variant chosen for this project is 4 B+ with 2GB of RAM because it has reasonable power without overheating like the 4GB variant without cooling. From tests run in the project, the board did not exceed 75 degrees Celsius (the limit for it being 85 degrees). It is connected to both the slave device and the camera. It is powered by a 5V external battery mounted on the chassis of the car and connected to the USB Type C connector of the board.

This master device is connected to the slave by 4 pins. These pins are: pin 29 (GPIO 5), pin 31 (GPIO 6), pin 33 (GPIO 13) and pin 35 (GPIO 19). They can also be seen in the Figure 4.4. The communication between the two devices is digital, as the pins described above connect do the digital pins D0, D1, D2 and D3 of the Arduino. This type of connection provides a 4-bit data transfer between the devices, meaning we can send 16 commands from the master to the slave. A GND pin is connected to a GND pin of the Arduino to ensure a common ground.

The most important sensor of the project is represented by the camera. This Raspberry Pi Camera is connected in the camera port of the board through a special cable. The camera is mounted on a wood stick in the back of the car chassis,see Figure 4.1. It is mounted around 15 cm above the chassis and tilted at around 30 degrees from the stick (which is perpendicular to the car) to ensure great visibility of the environment.

## 4.5 Assembly of the Components of the Car

The chassis and the motors are all assembled using screws and nuts, as they have to be strong and sustain a reasonable weight. The electronic components, the boards, the driver and the supply batteries are all assembled on the chassis using double sided tape. The wood stick of the camera is mounted on the back of the chassis without screws or tape, it fits perfectly, see Figure 4.1.

## 4.6 Track, Obstacle and Sign Assembly

The testing track, obstacle and stop sign were mandatory for both developing and testing this project. They were carefully chosen to resemble real life signs, cars and roads (Figure 4.2). These objects also represent the proof of concept of the project. While the car was designed to perform the best on this track, it can also run on other tracks with road like colors and standard stop signs. However, the object (represented by a 1969 Ford Mustang model) must be almost the same.

The test track is made out of a navy-blue blanket (resembles the road), 2 by 1.4 meters with some white stripes glued to it. It is important for the test surface not to be very reflexive.

The stop sign is made out of a printed sign mounted on a piece of cardboard (Figure 4.5). The image and the shape are standard. The pole of the sign is made out of metal for steadiness on the track.

The object is represented by a small blue model car (Figure 4.6).


Figure 4.5: Stop Sign


Figure 4.6: Ford Mustang

# Chapter 5
# Slave Device Software

## 5.1 Environment Setup

The coding language used for programming the Arduino board is Arduino programming language which is basically built as a framework on top of C++ programming language. The IDE used for programming the board is called Arduino IDE and has been installed on a Windows platform.

The steps are easy for loading the code on the microcontroller. The code must be written using the IDE and then compiled, verified and uploaded. The uploading is done through an USB cable, and the IDE must be specially configured for the specific board used. In this case, the IDE was configured to upload the code to the Arduino Uno R3.

As mentioned above, the steps of preparing the environment for uploading the code to an Arduino are straight forward, one must not forget to remove any active connections on the RX and TX pins (Figure 5.1) of the Arduino, otherwise the upload of the code will fail.

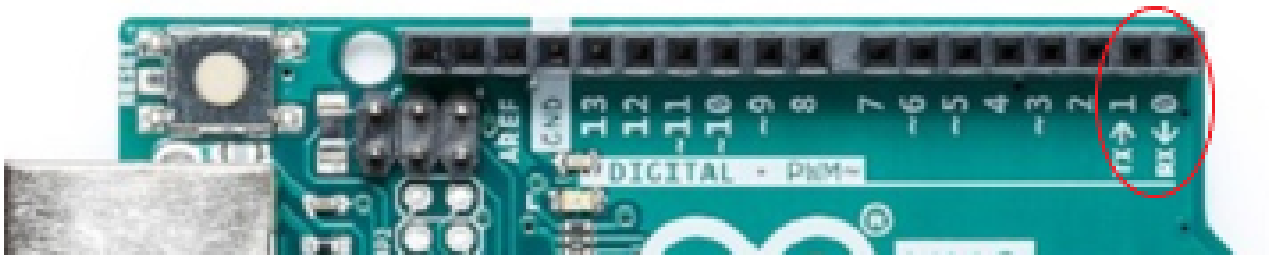


Figure 5.1: Arduino RX TX Pins

## 5.2 Pins Setup

For easier understanding and maintainability of the code, constants were declared with specific names for the pins. This was done both for the pins that connect to the driver and also for the ones that connect to the Raspberry Pi. For example, the following line `const int EnableR = 5;` allows us to name pin 5, which represents ENA of the driver. EnableR is a representative name, as the role of ENA is to enable the motors on the right side. The digital pins from the master were also named Dx, representing digital pin number x.

There is a standard practice for initializing pins of the Arduino when using them. For doing this, a `setup()` function is required, usually put in the beginning of the code. In this function the pins themselves and their roles are specified. For example, EnableR pin (previously named) is used to command the driver, so this means it must be an output pin, the instruction

`pinMode(EnableR, OUTPUT);` does just this. In an opposite manner, Dx pins used for communication with the master device (receiving commands from it) are defined as input pins. As it can be observed from the code, some other auxiliary constants are defined.

## 5.3 Moving Functions

These basic functions of the car are achieved by sending commands to the motor driver. The functions are named with suggestive notations, for example `Forward(), Left()` and so on.

The functions are implemented using the six pins available for connection with the driver. Let us take the example of the `Forward()`function:

```
    void Forward()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,110);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,110);
}
```

One can observe that for each side of the car, we have three commands: High and Low pins control the direction of the motor. The value of the pins can be either 1 or 0, `HIGH` meaning 1 and `LOW` otherwise. This can be also seen from the digital instruction (`digitalWrite`) given to the microcontroller (digital means binary). It is obvious that for moving the car forward the combination used above is necessary. There is the same combination on both sides, left and right.

The other instruction of the three (from only one side) is an analog one. This means that the value at the output can have more values (256 from being an 8-bit microcontroller). This command controls the speed of the motors. Higher value means higher voltage (motors' speed is controlled in voltage) and a higher spinning rate.

In the case of the `Forward()` function the instructions are somehow symmetrical on both sides. This is however, not true for all the moving functions. It is only true for `Forward()`, `Backward()` and `Stop()` functions. The other steering functions have different speeds on each side (controlled by Enable pin) in order to achieve the steering. Different speed ratio of the sides assures the spinning. In spite of this change, the architecture of the functions is the same. There are more `Left()` and `Right()` functions (2 for left and 2 for right) to ensure multiple levels of steering.

Let us take the example of `Right3()` function to better understand how it works:

```
    void Right3()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,110);
digitalWrite(HighR, LOW);
```

```
digitalWrite(LowR, HIGH);
analogWrite(EnableR,60);
}
```

It can be seen that the structure is the same and moreover, the direction (indicated by `High` and `Low` pins) is the same as for the Forward() function. The difference appears in the analog domain, where Enable pins have different values depending on the side. As being a right function, this steers the car to the right, so it is obvious that a higher speed of the motors on the left side is needed in order to achieve that. In this example, the speed of the right side of the motors is not zero, meaning that the car turns right, but with a smaller steering angle. The ratio between the two sides speed is `110 / 60 = 1.83` and has been determined by trial and error.

The other turning functions work on the same principle and will not be detailed here. There are two more functions, `Stop()` and `StopDetection()` which, as the name suggests, stop the car. The `StopDetection()` function has some delays implemented and has the ability to start the car after a certain amount of time. However, the delay function seems to not be working properly (meaning the two functions are the same) and the solution is to integrate the delay into the master device. It will be shown later how to do that.

The principle of these two functions is simple, write 0 on `Enable` pins for both sides and the motors will stop.

## 5.4 Communication with the Master Device

The communication between the two devices is done through 4 digital pins. This means that we have 16 available commands.

Function `Data()` is responsible of reading the information coming from the Raspberry Pi computer. This function is easy to understand as all it does is to read the information on the pins and convert it into decimal. Being digital pins, we have to use `digitalRead()` for reading. As an example, `a = digitalRead(D0);` reads the information available on pin `D0` and transfers it into a type integer variable. `D0` pin represents the least significant bit while `D3` is the most significant one. The conversion from binary to decimal is done using the following formula `data = 8*d+4*c+2*b+a` where data is the final decimal value and a, b, c and d are the binary values read from the pins.

The data value will be further used in the loop functions which is the main function of the slave device. Here the data value will be checked in an infinite loop and some decisions will be made according to it. This is how the slave is controlled by the master. Let us take an example:

```
if(data == 0)
{
    Forward();
}
```

If the value read from the master is 0, then proceed with the `Forward()` function. In this implementation only 9 values of the data are possible (because there are 10 moving functions

implemented) out of 16 available. Each and every function of the slave is controlled in this way. The Figure 5.2 below shows the concept diagram.

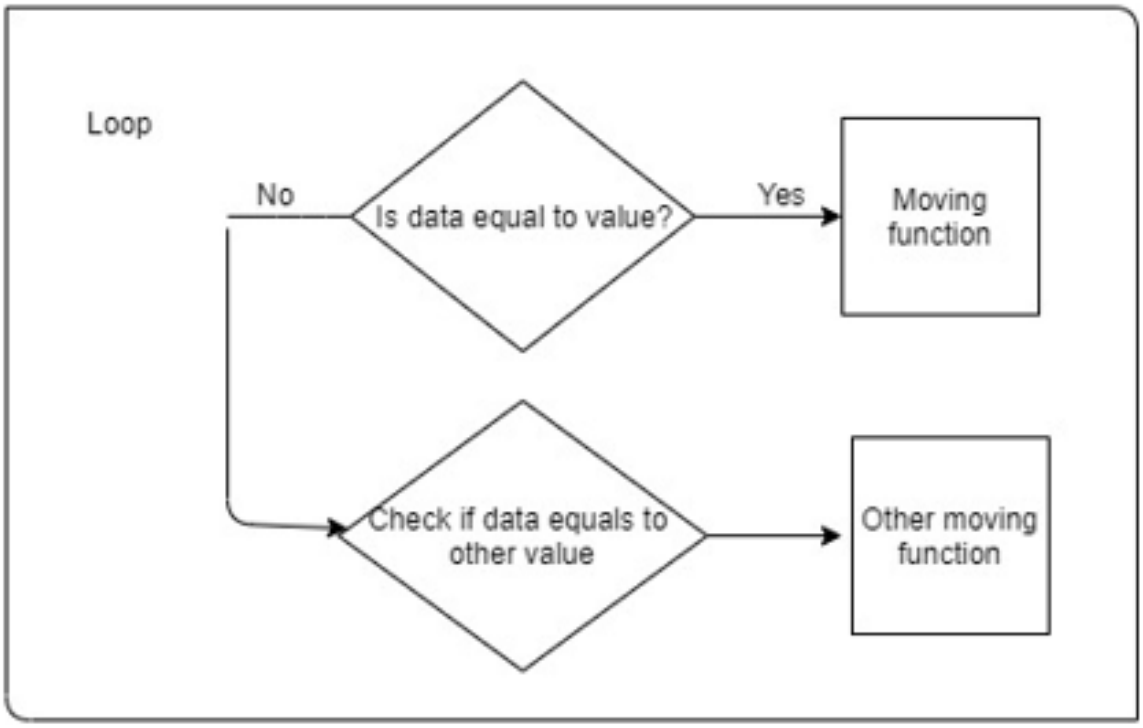


Figure 5.2: Moving Function Diagram

One very important aspect in the `loop()` function is to not forget to call the `Data()` functions, as this is the one that will provide the data values. Without calling this function, the communication is not possible.

## 5.5 Conclusions

Although the code from the slave device might appear complicated, in fact it is very simple to read and understand. It is well divided in setup, moving and communication functions and very easy to maintain or update. The communication function, although convoluted at first, proves simple and efficient with a well thought strategy (digital communication through pins). The most complex part of the slave code implementation was to find the right values for the speed of the motors for each function. This was done by trial and error on the testing track. The project is open for updates and new functions, as the communication with the master allows implementation of up to 16 commands, yet only 10 are used.

# Chapter 6

# Master Device Software -Lane Assist-

## 6.1 Environment Setup

The first step in setting up the environment on the Raspberry Pi is to flash the operating system into an SD card that will later be put inside the Raspberry. This process is uncomplicated and it uses an official tool called "Raspberry Pi Imager", which can be downloaded from their official website.

The next step is to power on the master device connecting it to a monitor using a HDMI to mini HDMI cable. This step is required for activating some graphical tools like CPU load and temperatures, setting up the WiFi as well as enabling VNC viewer support. After doing this, the board is ready to connect remotely to any PC.

One very important step was to create an OpenCV environment. This was done by cloning OpenCv from GitHub. In this step, all the packages that were necessary were installed. Then, building OpenCV with CMake was necessary. After this, all the libraries were included in the programming editor (Geany) using package configurators.

The next step was to install Raspicam and WiringPi libraries on the board. Rapsicam library permits working with the camera while WiringPi library allows working with the pins present on the board. Also, the camera support had to be activated from the Raspberry Pi.

## 6.2 Code to Capture Video and Images

Capturing video and static images is an essential part of the code. This is what was firstly done when writing the master code began. Some variables were initialized on the first part of the code as the code started to implement more and more functions.

The camera initialization is the first step done in order to capture video, this code is present in the main function and assures the user that the camera is connected, otherwise an error will be raised ("`Failed to connect`"). As it can be seen from the code, this also returns the identification number of the camera.

```
Setup(argc, argv, Camera);
cout<<"Connecting to camera"<<endl;
if (!Camera.open())
{
    cout<<"Failed to Connect"<<endl;
}
cout<<"Camera Id = "<<Camera.getId()<<endl;
```

As one can observe, a `Setup()` function is called in the initialization of the camera. That function allows for parameters control of the frame such as width, height, contrast and so on. This function is implemented in the beginning of the code. The values for the resolution (`HEIGHT` and `WIDTH`) have been chosen by trial and error for the project's test track. One important parameter is the FPS, which controls in how much frames per second the camera is recording. The value 0 is special, representing the maximum number of FPS the camera is able at a respective moment.

```
    void Setup ( int argc,char **argv, RaspiCam_Cv &Camera )
{
Camera.set ( CAP_PROP_FRAME_WIDTH, ( "-w",argc,argv,420 ) );
Camera.set ( CAP_PROP_FRAME_HEIGHT, ( "-h",argc,argv,240 ) );
...
```

The capturing part is done by the `Capture()` function, shown below. `Camera.grab()` instruction allows for grabbing the next frame from the camera, while `Camera.retrieve()` decodes and returns the grabbed frame. In this case, the next frame is put in the variable `frameBGR`. The other instructions allow for conversion between RGB and GBR color spaces, this is done because the frame is retrieved in BGR color space and RGB is needed later. To capture continuous frames (video) a `while(1)` loop was integrated in the main function. The `Caption()` functions is written inside it.

There is also an FPS counter integrated, which computes exactly how many frames per second are obtained after all the calculations are done. For doing this, the "chrono" library is used an the principle is very simple. The current time of the begging of the `while(1)` loop is registered and at the end of it, the current time is again registered. One second divided by the difference of the two time values, gives the real number of frames per second, see the code below.

```
    void Capture()
{
Camera.grab();
Camera.retrieve(frameBGR);
cvtColor(frameBGR, frame, COLOR_BGR2RGB);
cvtColor(frameBGR, frame_stop, COLOR_BGR2RGB);
cvtColor(frameBGR, frame_object, COLOR_BGR2RGB);
}

std::chrono::duration<double> elapsed_seconds = end-start;
float t = elapsed_seconds.count();
int FPS = 1/t;
```

To display frames, the `imshow` instruction is used in the `main()` function. The project is using `imshow()` for every frame displayed on the screen. The frames displayed have relevant names for better understanding.

## 6.3 Region of Interest

Region of interest of the frame means focusing on one specific frame and not the whole one. This must be done in order to achieve a perspective view, which will later be discussed in this chapter. Figure 6.1 shows how a region of interest looks in the frame. The red line defines the region of interest. This region will only be used in lane detection, it is not related with sign and obstacle detection.

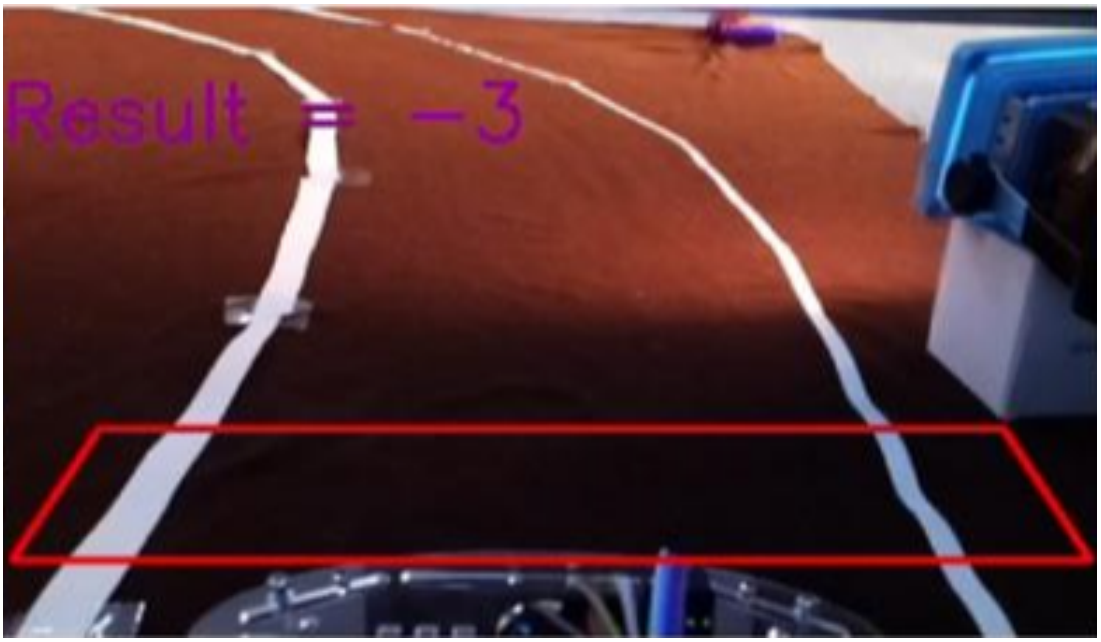


Figure 6.1: Region of Interest

The region was defined by adding four points to the frame. The points are added by trial and error and then are united to create this shape. The code line below defines the four points shown above.

```
Point2f Source[] = {Point2f(38,160), Point2f(382,160),
Point2f(5,210), Point2f(415,210)};
```

Then, the red lines are drawn using these four points. This step is only needed for calibration, the lines not being mandatory for line detection function. The points are drawn using the `line()` function together with the `Source[]` array of our points. The drawing is done inside the `Perspective()`. Function `Scalar()` represents the color in BGR color space and the last parameter is the thickness of the line.

```
void Perspective()
{
line(frame, Source[0], Source[1], Scalar(0,0,255), 2);
line(frame, Source[1], Source[3], Scalar(0,0,255), 2);
line(frame, Source[3], Source[2], Scalar(0,0,255), 2);
...
```

## 6.4 Perspective Transformation

After applying the perspective wrap, the car is able to see the lanes taken out from the region of interest in a straight manner. It resembles a bird eye view, meaning it can see everything from above (Figure 6.2).

Figure 6.2: Perspective View

Using the region of interest, the car can reach Figure 6.2 just by changing perspective. For achieving this, four points will be defined in the `Destination[]` array (shown below). The algorithm from the region of interest also applies here.

```
Point2f Destination[] = {Point2f(80,0), Point2f(280,0), Point2f(80,240),

Point2f(280,240)};
```

For changing the perspective, the function `getPerspectiveTrandform()` will be used and the result will be stored in a new variable. The final step is to create a new image using this matrix and the `warpPerspective()` function. This function applies a perspective transformation to an image.

```
Matrix = getPerspectiveTransform(Source, Destination);
warpPerspective(frame,framePerspective, Matrix, Size(400,240));
```

The obtained frame Figure 6.2 will also be displayed on the desktop of the Raspberry Pi. And the `Perspective()` function will be called in the `main()` program.

## 6.5 Threshold Operations

Lane detection will be done on the frame obtained on the previous subchapter. However, a color image will be hard to process and will be converted into a grayscale image. The new gray image will also be saved in a new variable. The conversion is done using the function `cvtColor()` with the instruction `COLOR_RGB2GRAY`.

```
cvtColor(framePerspective, frameGray, COLOR_RGB2GRAY);
```

After doing this, a process of binarization is required. Binarization means the process of converting a pixel image into a binary one, the last one will only have two colors, full black and full white. This can be achieved by using a built in function called `inRange()` of the OpenCV library. The functions receives the source and destination frames and the low and high thresholds of the white color:

```
inRange(frameGray,150, 255,frameThreshold);
```

After the operation, something similar to Figure 6.3 should be obtained from Figure 6.2.

Figure 6.3: Binarized Perspective

The threshold values of the white color have been specifically adjusted for bright daylight and also a cloudier environment in such way that only the lanes become white and no other surrounding objects (the color of the test track also helps). However, the car does not recognize the lanes in the night unless it has a light source to illuminate the track. The entire threshold calibration was done by trial and error.

## 6.6 Canny Edge Detection

Canny Edge Detection is a popular edge detection algorithm. It was developed by John F. Canny in 1986. It is a multi-stage algorithm and each stage will be presented[7].

1. Noise reduction
    - Since edge detection is susceptible to noise in the image, first step is to remove the noise in the image with a 5x5 Gaussian filter.
2. Finding Intensity Gradient of the Image

- The smooth image is then filtered with a Sobel kernel in both horizontal and vertical direction to get first derivative in horizontal direction (Gx) and vertical direction (Gy). From these two images, we can find edge gradient and direction for each pixel as follows in Figure 6.4:

$$Edge\_Gradient\ (G) = \sqrt{G_x^2 + G_y^2}$$
$$Angle\ (\theta) = \tan^{-1}\left(\frac{G_y}{G_x}\right)$$

Figure 6.4: Edge Gradient

- Gradient direction is always perpendicular to edges. It is rounded to one of four angles representing vertical, horizontal and two diagonal directions[7].

3. Non-maximum Suppression

- After getting gradient magnitude and direction, a full scan of image is done to remove any unwanted pixels which may not constitute the edge. For this, at every pixel, pixel is checked if it is a local maximum in its neighborhood in the direction of gradient, see Figure 6.5.

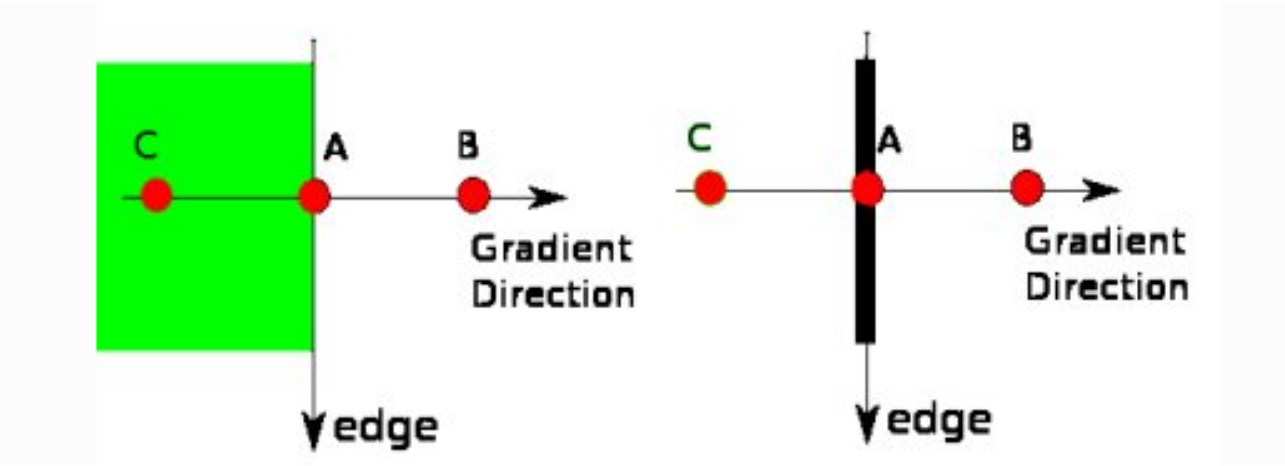


Figure 6.5: Suppression[7]

- Point A is on the edge ( in vertical direction). Gradient direction is normal to the edge. Point B and C are in gradient directions. So point A is checked with point B and C to see if it forms a local maximum. If so, it is considered for next stage, otherwise, it is suppressed (put to zero). In short, the result you get is a binary image with "thin edges"[7].

4. Hysteresis Threshold

- This stage decides which are all edges are really edges and which are not. For this, we need two threshold values, minVal and maxVal. Any edges with intensity gradient more than maxVal are sure to be edges and those below minVal are sure to be non-edges, so discarded. Those who lie between these two thresholds are classified edges or non-edges based on their connectivity. If they are connected to "sure-edge" pixels,

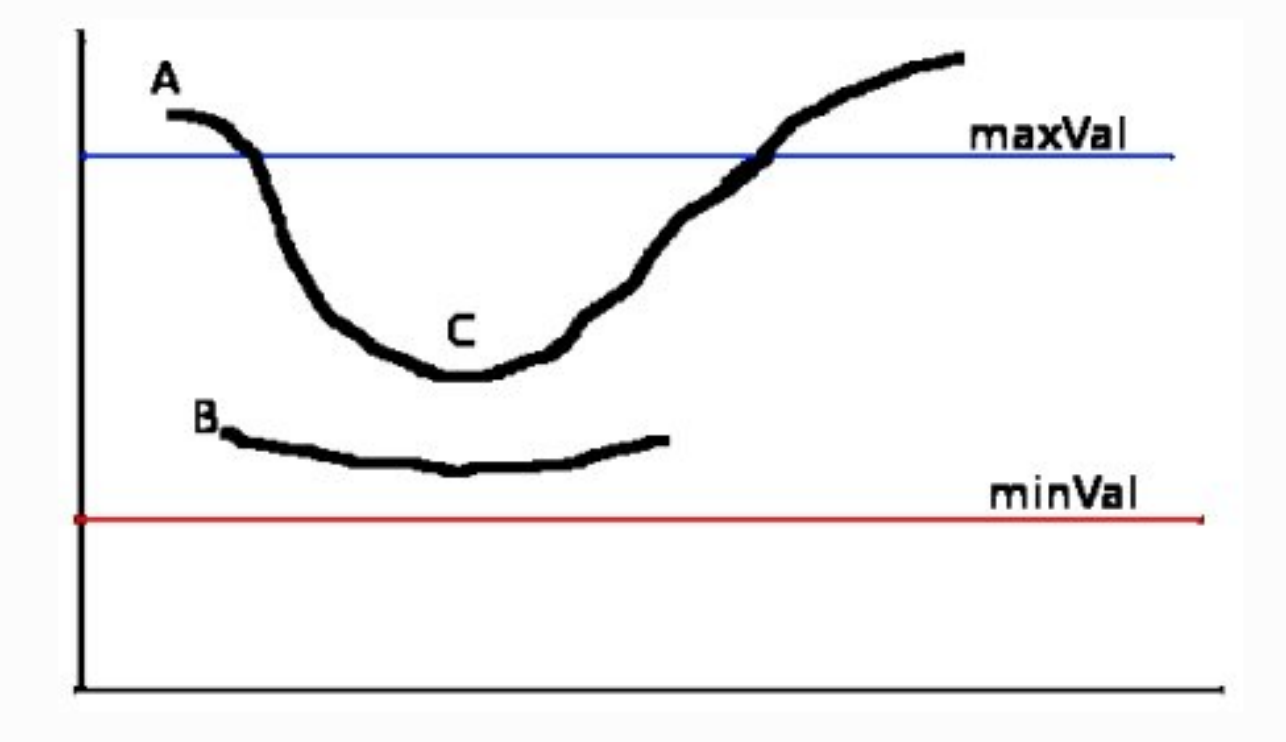


Figure 6.6: Threshold[7]

they are considered to be part of edges. Otherwise, they are also discarded[7], see Figure 6.6.

- The edge A is above the maxVal, so considered as "sure-edge". Although edge C is below maxVal, it is connected to edge A, so that also considered as valid edge and we get that full curve. But edge B, although it is above minVal and is in same region as that of edge C, it is not connected to any "sure-edge", so that is discarded. So it is very important that we have to select minVal and maxVal accordingly to get the correct result. This stage also removes small pixels noises on the assumption that edges are long lines. So what we finally get is strong edges in the Figure 6.7.

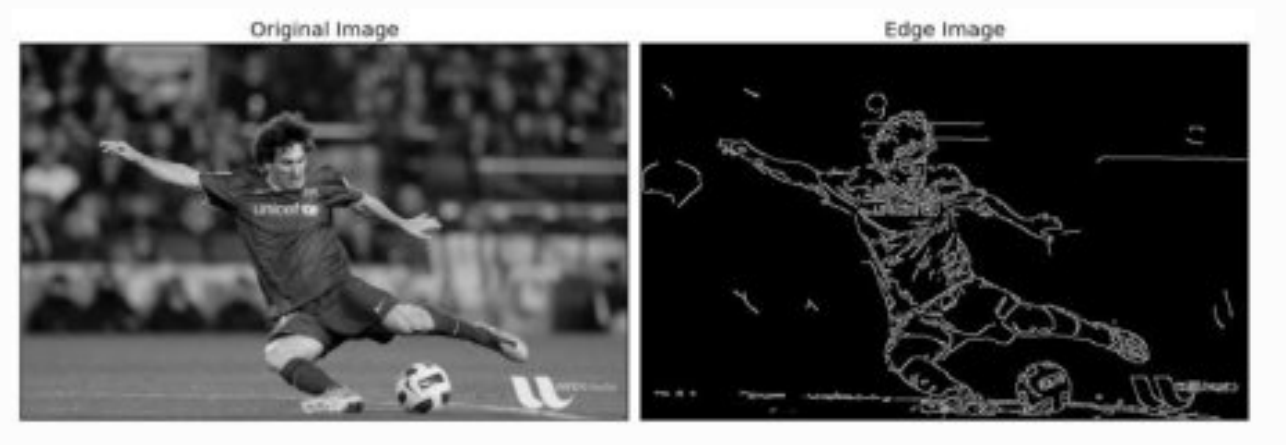


Figure 6.7: Example[7]

- This algorithm was used in this project for detecting the edges of the lanes. The description above may seem complicated, however OpenCV implements this algorithm in a single function called Canny(). This function was implemented inside the

`Threshold()` function `Canny(frameGray, frameEdge, 100, 500, 3, false);`.

- In this project, Canny edge detection is applied to the grayscale image of the perspective described in the chapter above. Minimum threshold for the hysteresis process is 100, the second threshold is 500 and the Sobel filter is 3 (creates a 3x3 matrix) while the parameter for the equation for finding gradient magnitude is false as it is not needed. The threshold parameters were also chosen by trial and error and highly depend on the test track surface. Higher thresholds might be necessary for a more reflexive surface. In Figure 6.8 the detection of the lanes form the grayscale image using Canny detection can be seen.

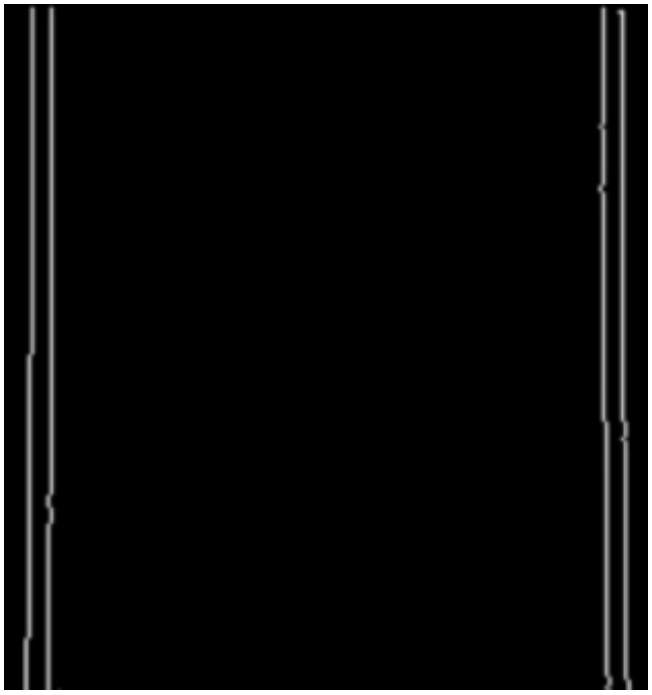

Figure 6.8: Canny Edge Detection

- After this step, the car shows on the Raspberry pi desktop the merging of the edge image (Figure 6.8) and the Figure 6.3 obtained after doing the threshold operations. The code is written in the `Threshold()` function and the final image obtained is called `frameFinal`, `add(frameThreshold,frameEdge,frameFinal);`.
- The resulting image looks similar to Figure 6.9. It is obvious that this final frame is composed by the images described above just by simply looking at it. The edges of the Canny detection are visible and filled with the binarized image obtained after doing threshold operations.

## 6.7 Lane Finding (stategy and implementation)

The process of finding the lanes from Figure 6.9 is rather complex, but the strategy is easy to understand. First of all, one may think of obtaining a histogram of the image, this is what is done in this project.

Let us consider Figure 6.10

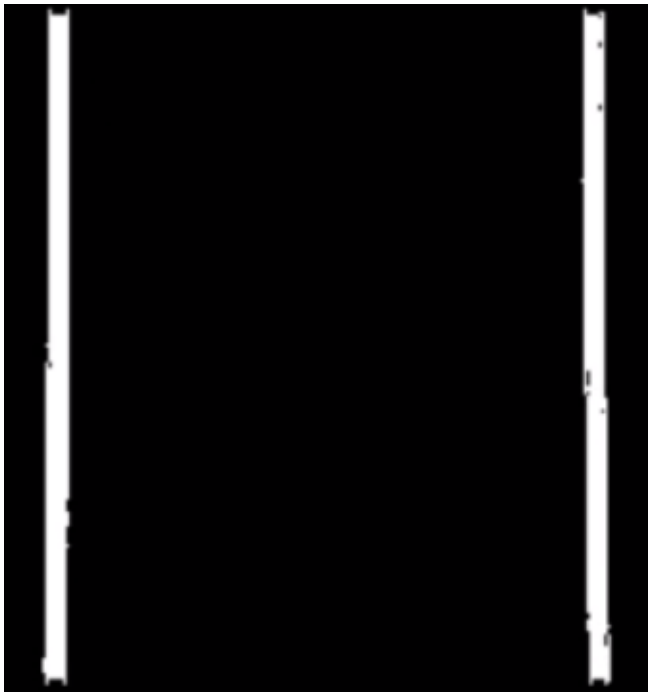

Figure 6.9: Final Image

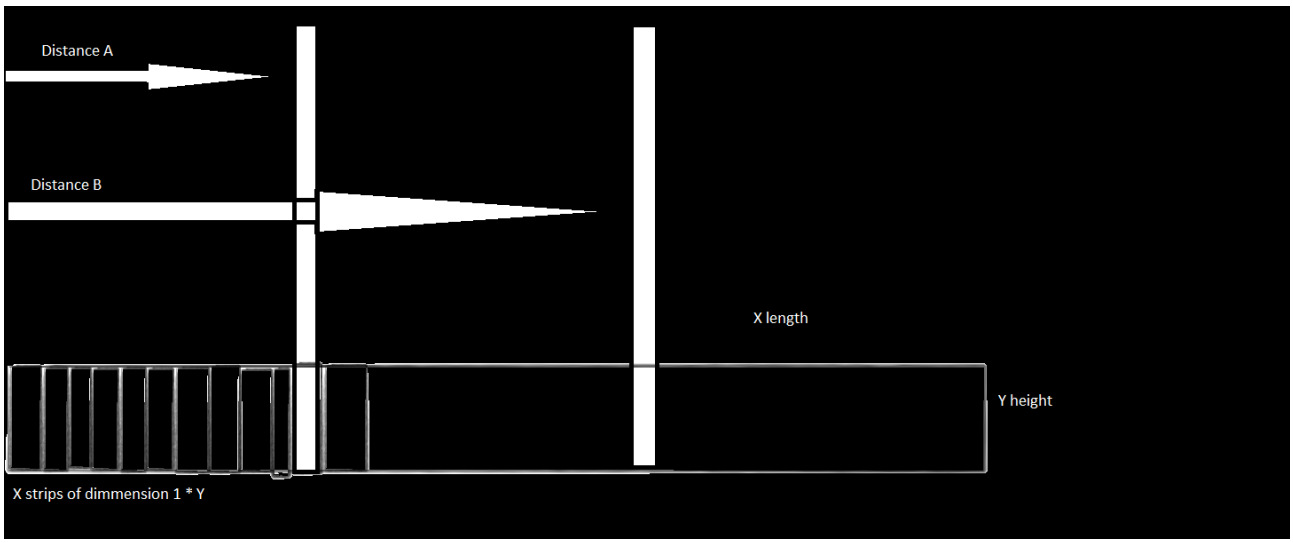


Figure 6.10: Distances

This is a scheme of the final recognition figure and it is based on Figure 6.9. To achieve lane detection the following strategy is implemented. In order to know where the lanes are, one must know distance A and distance B. In order to obtain these distances, a new region of interest is created (it can be seen in the bottom part of Figure 6.10. The X and Y dimensions for the rectangular region are chosen using trial and error, their exact value is not taken into consideration now. It can be observed that this region was divided in X pixel strips, this means that each strip has a length of only one pixel and a height of Y pixels. So, a kind of histogram is made here. Then, all the intensities of all the pixels in one strip are multiplied with the number of pixels in the strip. For example, the first strip seen in the picture has all the pixels black, so the equation will be `Y * 0 = 0`. This value will be stored in an array. For the first strip that represents a lane (white strip from the left lane) the value will be `Y * 255 = Z`. All the values from the strips will be stored in the array (so the array will have Y values) and then compared to each other. The values that are different from 0 and equal or close to Z will be classified as representing a lane. This is how the position of the lanes in the picture will be known.

From the software point of view, a new function `Histogram()` will be created to achieve exactly the algorithm described above. An array design to hold the values is created and the region of interest is achieved using the `Rect()` function. Then the region of interest is divided in strips and each pixel intensity is normalized using the `divide()` function. In the end, the value obtained from each strip is pushed into an array (`histogramLane`). Using this array, one may find the position of the lanes just looking at the values.

To find the lanes inside the array, a new `LaneFinder()` function is created. The main idea behind this function is using an iterator (which is basically a pointer) that goes through the array. The array is split in pieces, the first lane being searched until the position 130 and the second one being searched from position 230 until the end of the array. This method assures a faster searching of the values inside the array. The values were again chosen by trial and error. The distance is then computed using the begging of the array and the location inside it pointed by the iterator. Finally the distance to the lanes is known and two green lines can be drawn where the software detects the lanes, see Figure 6.11. The functions described in this subchapter must also be called in the while loop located in the main function.

Figure 6.11: Green Lines

## 6.8 Lane Centering

For this achieving this task, a new function called `LaneCenter()` will be created. The functioning of it is very simple, a center line is computed using the positions of the two lanes, so basically a middle point between them is computed :

```
laneCeneter = (RightLanePos - LeftLanePos) / 2 + LeftLanePos;
```

The two variables `RightLanePos` and `LeftLanePos` contain the position of the two lanes and the middle point between them is computed with a simple formula presented above. The middle line is also displayed with green color as it depends on the position of the lanes (the position of this line is called `laneCenter`.

A fixed middle line position is computed using the middle of the frame (width of the frame is divided by 2 to obtain the position). The position of this line can be slightly changed during calibration stage to match the green center line. The line is drawn with blue color and represents the reference(`frameCenter`). The distance between the green center line and the fixed blue one will tell the car in which way it needs to steer. This distance is also computed in this function (`result = laneCenter - frameCenter`) and the result is displayed on the screen using stringstreams in Figure 6.12.


Figure 6.12: Displaying result

## 6.9 Master Slave Communication

The master slave communication is essential in this project as it allows us to connect the Arduino board, which is responsible to control the motors, to the master device, Raspberry Pi, which must be in total control of the project. The communication has been described from the slave device's point of view in chapter 5, now a master device's point of view will be approached.

It is common that for using the pins on any board one must first initialize the pins. This project will use the Wiring Pi library for programming the Raspberry Pi GPIO.

The first step is to include the Wiring Pi library into the project, this is done using include instruction. Then the library will be used begging with an essential setup line:

```
wiringPiSetup();
pinMode(21, OUTPUT);
pinMode(22, OUTPUT);
pinMode(23, OUTPUT);
pinMode(24, OUTPUT);
```

This line initializes the library and allows for working with pins. Then, a common procedure of declaring the pins as output comes. It can be observed that the process is straight forward and the communication is not something very complex as many may think.

As described in chapter 4, the pins used for communication are pin 21, pin 22, pin 23 and pin 24. These pins are digital, this means that on each pin one can send either 1 or 0. This allows 16 different commands to be sent to the slave device, more than enough for implementing this project. The state of each pin (1 or 0) can be set using the `digitalWrite()` instruction. The master will send binary data to the slave, which will transform it into decimal numbers.

## 6.10 Sending Moving Commands for Lane Keeping

The whole principle of these functions lays on the data called 'result' resulted in chapter 6.8. This data is now used for determining in which direction the car should move. The functions are integrated into the loop located in the main function. They are very simple and have the architecture on an if else instruction:

```
if(result >= -10 && result <= 10)
{
digitalWrite(21, 0);
digitalWrite(22, 0);
digitalWrite(23, 0); //send 0
digitalWrite(24, 0);
cout<<"Forward"<<endl;
}
```

Let us take the example of the forward function. The result variable obtained from the `Histogram()` function is analyzed and a decision is made based on its value. For the car to move forward, the modulus of the result variable should be smaller than one. Then a binary 0 will be sent through the pins and the name of the function will be displayed in the console.

The same algorithm is done for all the possible moving functions implemented on the Arduino board. A list of commands for all of them will be included below, Table 6.1.

| Moving Function | Decimal Number |
|---|---|
| Forward | 0 |
| Right2 | 2 |
| Right3 | 3 |
| Left2 | 5 |
| Left3 | 6 |
| Stop | 7 |

Table 6.1: Moving Command

The threshold of the result variable for each moving function have been determined by trial and error. The command 7 is for the `Stop()` functions and will be later described as it relates

with sign detection and obstacle detection.

At this step, the lane keeping function can be declared done. At this point, the car is able to move inside lanes autonomously.

# Chapter 7

# Master Device Software
# -Sign and Obstacle Detection-

## 7.1 Code for Image Capturing

In order for the cascade classifier to work, a training dataset with the images of the object is required. In order to do this, RaspberryPi camera was used along with a C++ code for capturing images. Something similar to this code has been presented in Chapter 5, this though, has small changes.

The setup of the camera is the same, the differences appear when a for loop instruction is used to control how many pictures are taken. Changing the iteration parameter allows us to control exactly how many loops will be. For some applications one may need 100 pictures, for other 50 pictures, it can be controlled from the for loop instruction.

The instruction `imwrite("Obj"+to_string(i)+".jpg", frame)` saves the image with a specific name for easier access, while the `wiatKey()` instruction is essential, as it allows the capturing of a photo only in the user presses one random key. This is very useful as one may need time for rearranging the scene before taking a picture.

## 7.2 Stop Sign Detection

For achieving the stop sign detection, machine learning was used. With the help of Cascade Trainer GUI a network has been trained for recognition. This program outputs a .xml file that contains the classifier and can be put inside the code for recognition. It represents the backbone of recognition.

The stop sign used was described in Chapter 4. In order for the classifier to work, a number of positive images (Figure 7.1) and a number of negative images (Figure 7.2) were used.

Both for sign detection and for obstacle detection a number of 700 negative and 130 positive pictures were used. For sign detection the training time was very high, around 7 hours, but the results are really good. In the tests run before putting the .xml file onto the car, the recognition rate was around `90%`, which is more than enough considering the final frames per second of the car are between 7 and 8.

The Cascade Trainer GUI has many parameters to be introduced before training the network. These parameters were again chosen by trial and error, only the sample height and width were chosen taking into consideration the aspect ratio of the positive images.


Figure 7.1: Positive Image

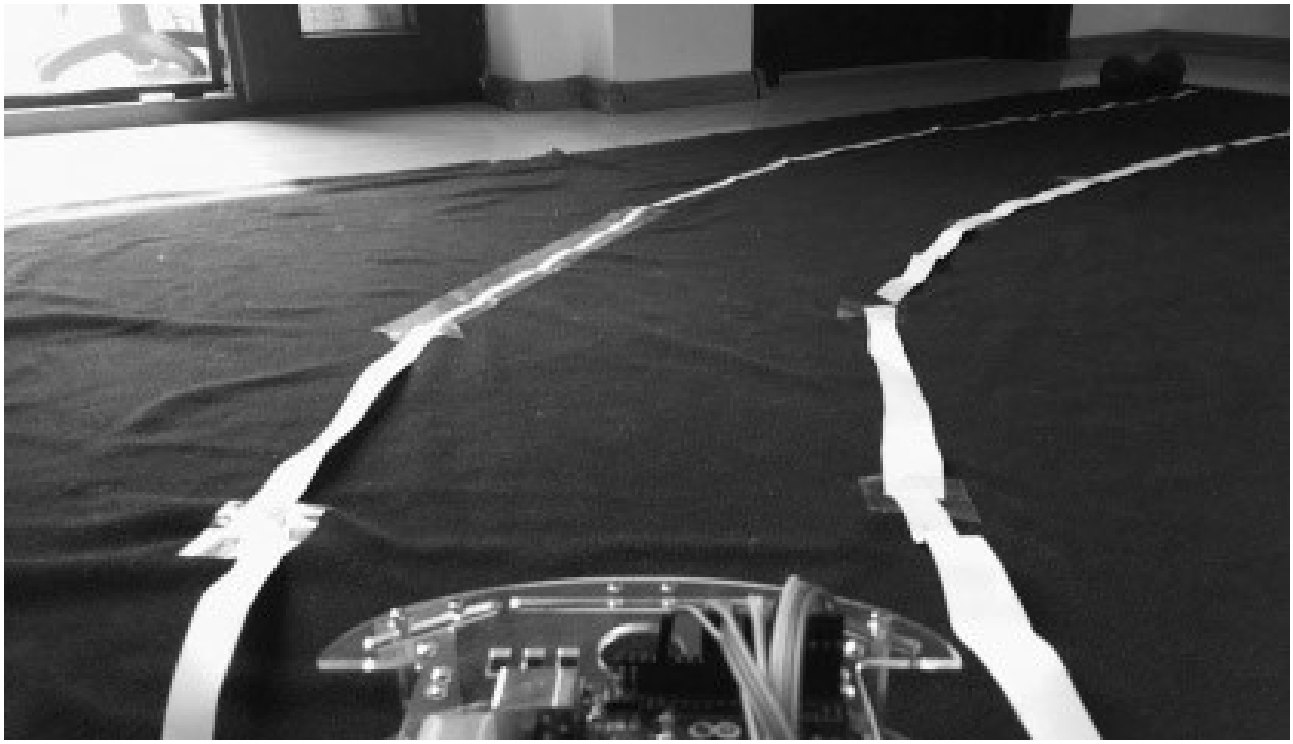

Figure 7.2: Negative Image

After training the network, the .xml file was loaded into the code. The detection is implemented in the function `StopSignDetection()`. The .xml is loaded into a `CascadeClassifier` object using `.load()` instruction:

```
if(!Object_Cascade.load("//home//pi//Desktop//Integrare machine learning//Object_cascade
{
printf("Unable to open object cascade file");
}
```

A new reason of interest is crated and converted into a grayscale image, as the function uses an unedited frame (uses the what the camera sees in Figure 7.3). The frame is then equalized using the `equalizeHist()` function.


Figure 7.3: Raw Image

The main instruction of this function is `.detectMultiScale` which is applied to the equalized frame and the `CascadeClassifier` and outputs a vector of points which indicates where the points of the object are. Using this vector, a strategy used before in this project is applied. Some red lines are drawn around the detected sign Figure 7.4. The frame was also cut to only display and process the left part of the screen, as the road signs are almost always put on the left (in right hand drive traffic). This increases frames per second, detection time and efficiency overall.

As it can be observed in Figure 7.4, there is also a distance to the sign displayed. The distance is also computed in the `StopSignDetection()` function and uses linear equations. For achieving an exact distance, two test positions of the stop sign were taken. One at 15 cm to the car and one at 30 cm. Using the linear equation `y = mx + c` and the points p1 and p2, the parameters for `dist_stop` were computed :

```
dist_stop = (-0.8423) * (p2.x - p1.x) + 77.647;
```

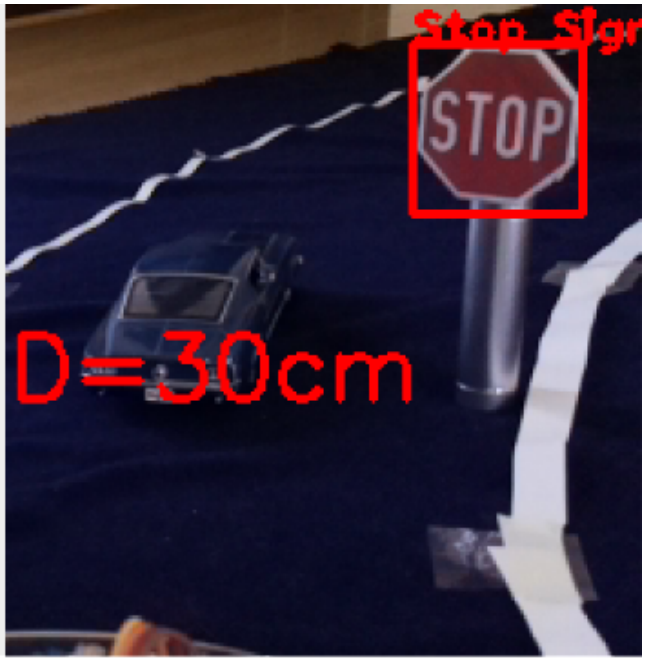


Figure 7.4: Stop Sign Detection

The `P2.x - P1.x` difference represents the height of the detected sign. It is very useful as it shows a linear relation to the distance at which the car is to the sign (a higher distance means a lower height).

As it was mentioned in the previous chapter, there are two more commands to send to the slave device. One of it is sent when the distance to the stop is less than 20 and higher than 5. This if condition is written in main and ensures that the car will stop for 8 seconds when it recognizes the stop. After these 9 seconds the car will continue its journey. The binary value of 7 is sent when the condition is met and the it corresponds to the `Stop()` function of the slave device (it will stop the car).

The trick of the command is that the car will basically sleep for 8 seconds:

```
if(dist_stop>5 && dist_stop < 20)
{
digitalWrite(21, 1);
digitalWrite(22, 1);
digitalWrite(23, 1); //send 7
digitalWrite(24, 0);
cout<<"Stop Sign"<<endl;
dist_stop = 0;
sleep_for(8s);
digitalWrite(21, 0);
digitalWrite(22, 0);
digitalWrite(23, 0); //send 0
digitalWrite(24, 0);
cout<<"Forward"<<endl;
sleep_for(500ms);
goto Stop_Sign;
}
```

Then a force command of the Forward() function will be sent (binary 0 will be sent) and kept for 0.5 seconds to ensure that the car passed the sign and does not detect it anymore.

At this point, the sign detection is completed and fully functional with a rate of detection of 90%, more than acceptable for this application.

## 7.3 Obstacle Detection

The strategy used for the obstacle detection is almost the same with the one used in the subchapter above. The algorithm will not be again presented. Only the small changes will be discussed. Of course, another classifier .xml file was used obtained from other photos. The same amount of positive and negative photos was used along with the same capturing code. In Figure 7.5 and Figure 7.6 the obstacle and the environment used for training can be seen.

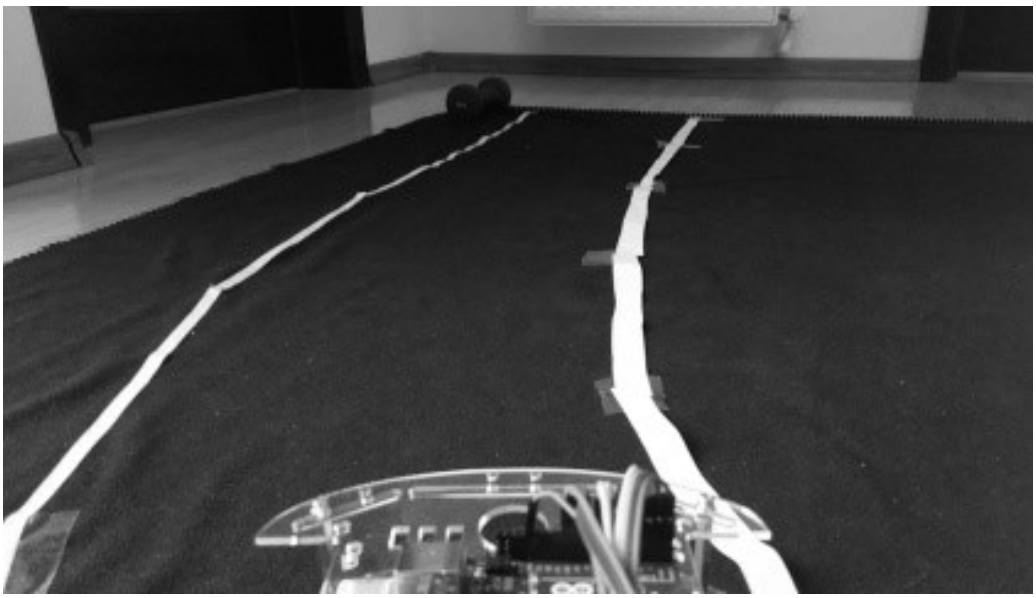

Figure 7.5: Negative Image

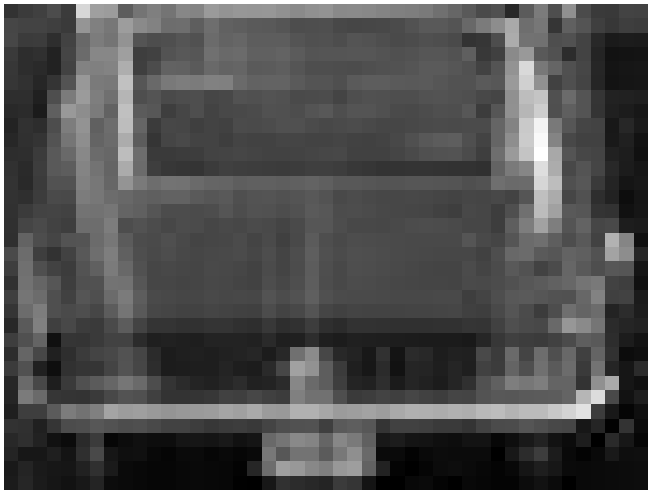

Figure 7.6: Positive Image

Only the back end of the obstacle car was taken into account when training (the positive images contain only the back) for faster training times and efficiency.

While the code for the function `ObstacleDetection()` is the same as the one for the `StopsignDetection()` function, the command sent after (for slave control) is slightly different. Also, when computing the distance, the linear equations were solved again as in the previous chapter. The command sent this time is a binary 8 and the function does not use sleep anymore. The behavior of the car is slightly different as it does not move until the object moves out of the threshold distance. So the command calls for the `Stop()` function of the slave device until the distance is high enough or the object is no longer detected. The distance threshold is 20 cm.

With this function completed and functional, the project is completed and achieves all the functions that was designed to do from the beginning.

## 7.4 Final Image

In this subchapter, the final images displayed on the desktop of the car can be seen in Figure 7.7. It is very useful for one to see what the car sees and processes to achieve all the tasks it is supposed to do. In the left side of the image, the console can be seen. It displays all the decisions that the car takes based on the respective variables.

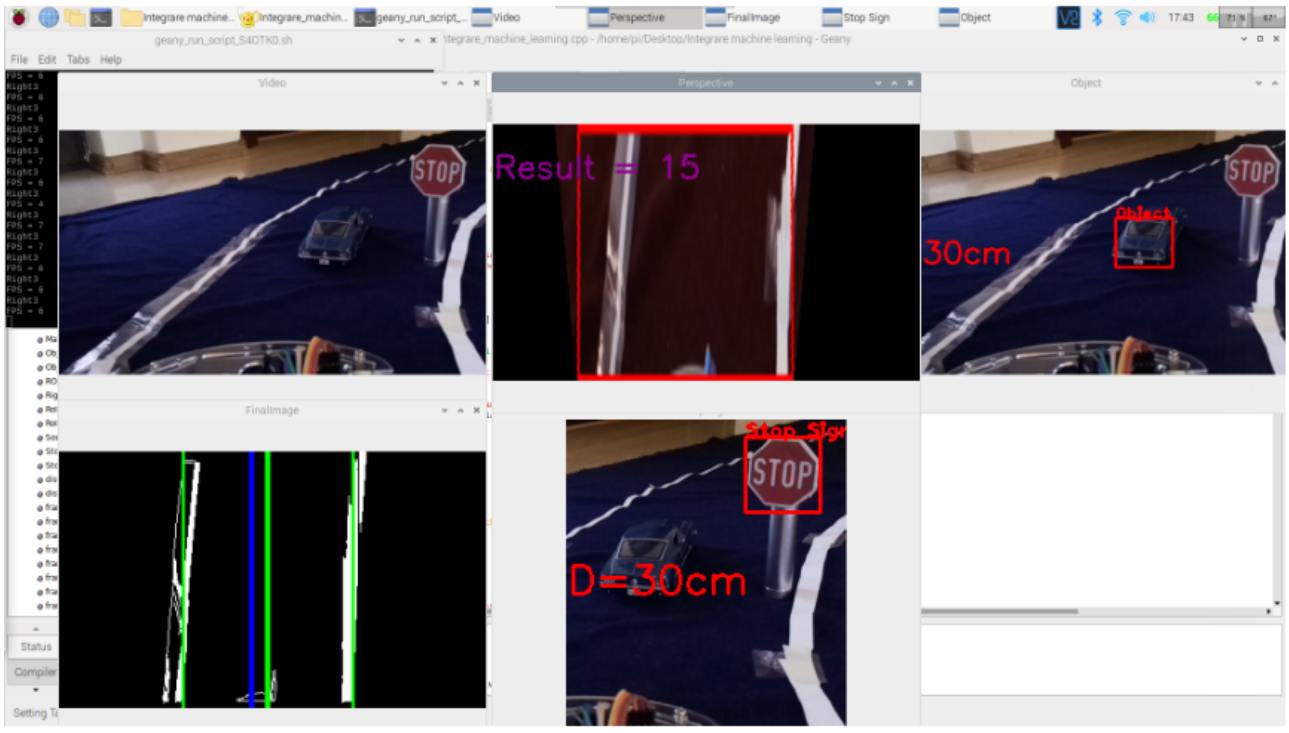


Figure 7.7: Final Desktop Image

# Chapter 8
# Testing and Problems

## 8.1 Threshold Operations

Threshold operations are a complicated subject, as the variables must be carefully chosen to obtain the best possible results. These operations were discussed in chapter 6. To obtain the best ones several tests were made in many light conditions. Of course, these parameters can be individually adjusted for one type of lighting, but the aim of the project was to obtain some variables that work well in almost all lighting conditions. Test were conducted during sunny days, indoors, outdoors, in rainy days and night conditions. After taking the tests, the numbers were calibrated for almost all indoor conditions, as the test track was located mainly there. The car sees the lanes well in all conditions excepting very poor light, where it needs the help of some artificial light, for example a LED headlight. Reflexive surfaces can be a slight problem and need to be avoided, the final image of lane detection Figure 8.1 shows a detection where one of the lanes is very reflexes a lot of light. However, the effect is not high enough to confuse the algorithm of the car.

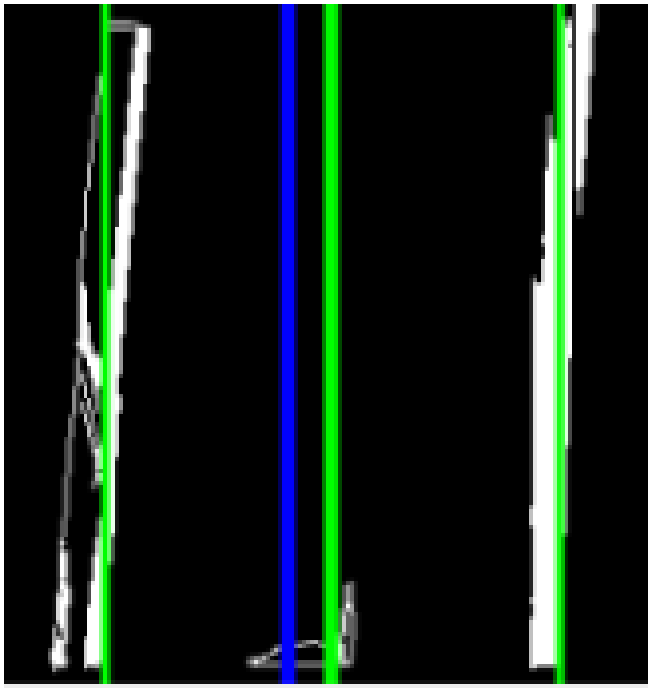

Figure 8.1: Reflexion Problem

## 8.2 Motors Speed

One of the greatest challenges encountered in this project was one that seemed really simple at the begging. Due to the way the car steers, more turning angles were implemented. However, the car did not turn as expected for a long time. Very fine adjustments were made to the motors

input voltages in order to solve this problem, also many different turns were drawn on the track to make the testing possible. The forward speed of the car was also finely tuned, as the master device did not have time to process all the information when the car was going with a high speed. It was discovered that the optimal spinning rate of the motors is about 43% of the total capacity.

## 8.3 Software Problems

The software adaptation and coding process was smooth, only one major problem was encountered with the WiringPi library. The problem was that the library was no longer available at the official web resource, so a walk-around had to be done, fortunately I was able to find another source.

## 8.4 Supply Problems

Due to the current period we are in, the components arrived with a very high delay, making the development of the project move slowly than anticipated. A major problem was with the power supply for the motors as they run on 12V. A high power 12 V battery is hard to find, and all the 5V batteries available were bad for boosting up the voltage (were not even able to power up the RaspberryPi). This is why most of the tests and the demos are done using a 12V wall power supply.

# Chapter 9
# Conlcusions

After successfully developing and implementing ADAS and autonomous functions with this project, I can safely say that this was a highly rewarding experience. Working with openCV and machine learning in an embedded project is very educative and useful at the same time. The computer vision technology advances every day and is and will be very important in the technology branch. Almost all new cars integrate computer vision algorithms in their software nowadays and the trend continues.

The development of this project was not easy, many times difficult problems were encountered, but with a little thinking and skill all of them were solved. This helps a lot with the learning process as one can especially develop skills when trying to solve a problem.

In conclusion. I think this project successfully implements the newest functions and abilities available in the automotive industry. It is a very challenging project that requires a lot of knowledge and work but also brings high rewards.

# Bibliography


[1] Audi lane assist. https://www.audi-technology-portal.de/files/images/975/active_lane_assist1_em_large.jpg.

[2] Gm traffic sign recognition. https://gmauthority.com/blog/wp-content/uploads/2017/01/General-Motors-Opel-Eye-on-Opel-Mokka-X-Traffic-Sign-Recognition.jpg.

[3] Autonomous emerghency breaking. https://miro.medium.com/max/932/1*nRxRkYqTTMbZeoSUknzDlg.jpeg.

[4] Arduino. https://www.distrelec.ro/Web/WebShopImages/landscape_large/6-/02/ArduinoUNO_SMD_A000073_30101956-02.jpg.

[5] Raspberry pi. https://s12emagst.akamaized.net/products/24703/24702612/images/res_66a7193689d5441235891cd2c77e667a_full.jpg.

[6] opencv cascade classification. https://docs.opencv.org/2.4/modules/objdetect/doc/cascade_classification.html.

[7] Canny detection. https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_canny/py_canny.html.

[8] Hamid Umar Zakir Abdul. *Current Collision Mitigation Technologies for Advanced Driver Assistance Systems–A Survey, PERINTIS eJournal.* 2016.

[9] Araz Lim, Hazel Si Min; Taeihagh. *Algorithmic Decision-Making in AVs: Understanding Ethical and Technical Concerns for Smart Cities.* 2019.

[10] Automakers, safety officials make crash avoidance systems standard by 2022. https//:cars.com.

[11] *Uniform provisions concerning the approval of motor vehicles with regard to the Advanced Emergency Braking Systems (AEBS) - Appendum: 130, Regulation: 131.*

[12] Euro NCAP. *Autonomous Emergency Breaking.*

[13] Puwadech; Nagai Masao Hayashi, Ryuzo; Chatporntanadul. *Improvement of Trajectory Tracking Performance in Autonomous Collision Avoidance by Steering. 7th IFAC Symposium on Advances in Automotive Control. Tokyo.* 2013.

[14] Raspberry pi: Cheat sheet. https://www.oxley.com/domain/silicon.com/.

[15] Bjarne Stroustrup. *The C++ Programming Language.* 1997.

[16] opencv. https://docs.opencv.org/3.4/d1/dfb/intro.html.

[17] Cascade classifier gui. https://amin-ahmadi.com/cascade-trainer-gui/.

# Annex A
# Slave Device Code

```
#include <math.h>
const int EnableL = 10;
const int HighL = 8; // LEFT SIDE MOTOR
const int LowL =9;
const int EnableR = 5;
const int HighR = 6; //RIGHT SIDE MOTOR
const int LowR =7;
const int MEDIUM_SPEED = 100;
const int HIGH_SPEED = 150;
const int D0 = 0; ///raspberry pi 21 LSB
const int D1 = 1; ///raspberry pi 22
const int D2 = 2; ///raspberry pi 23
const int D3 = 3; ///raspberry pi 24 MSB
int a,b,c,d,data;
void Data()
{
a = digitalRead(D0);
b = digitalRead(D1);
c = digitalRead(D2);
d = digitalRead(D3);
data = 8*d+4*c+2*b+a; //binary to decimal conversion
}
void setup() {
Serial.begin(9600);
pinMode(EnableL, OUTPUT);
pinMode(HighL, OUTPUT);
pinMode(LowL, OUTPUT);
pinMode(EnableR, OUTPUT);
pinMode(HighR, OUTPUT);
pinMode(LowR, OUTPUT);
pinMode(D0, INPUT);
pinMode(D1, INPUT);
pinMode(D2, INPUT);
pinMode(D3, INPUT);
}
void Stop()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, LOW);
analogWrite(EnableL,0);
digitalWrite(HighR, LOW);
digitalWrite(LowR, LOW);
analogWrite(EnableR,0);
}
void Forward()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,110);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,110);
}
void Backward()
{
digitalWrite(HighL, HIGH);
digitalWrite(LowL, LOW);
analogWrite(EnableL,10);
digitalWrite(HighR, HIGH);
digitalWrite(LowR, LOW);
analogWrite(EnableR,10);
}
void StopDetection()
```

```
{
analogWrite(EnableR, 0);
analogWrite(EnableL, 0);
delay(9000);
analogWrite(EnableR, 150);
analogWrite(EnableL, 150);
delay(1000);
}/
*
void Left1()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,120);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,MEDIUM_SPEED);
}
void Left2()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,100);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,MEDIUM_SPEED);
}
*/
void Left3()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,60);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,110);
}
void Left4()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,0);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,100);
}/
*
void Right1()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,MEDIUM_SPEED);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,120);
}
void Right2()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,MEDIUM_SPEED);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,100);
}
*/
void Right3()
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,110);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,60);
}
void Right4()
```

```
{
digitalWrite(HighL, LOW);
digitalWrite(LowL, HIGH);
analogWrite(EnableL,100);
digitalWrite(HighR, LOW);
digitalWrite(LowR, HIGH);
analogWrite(EnableR,0);
}
void loop()
{
Data();
if(data == 0)
{
Forward();
}
else if(data == 1)
{
Right3();
}
else if(data == 2)
{
Right3();
}
else if(data == 3)
{
Right4();
}
else if(data == 4)
{
Left3();
}
else if(data == 5)
{
Left3();
}
else if(data == 6)
{
Left4();
}
else if(data == 7)
{
StopDetection();
}
else if(data > 8 )
{
Stop();
}
}
```

# Annex B
# Master Device Code

```
#include <opencv2/opencv.hpp>
#include <raspicam_cv.h>
#include <iostream>
#include <chrono>
#include <thread>
#include <ctime>
#include <wiringPi.h>

using namespace std;
using namespace cv;
using namespace raspicam;

//Image processing variables
Mat frameBGR, frame, Matrix, framePerspective, frameGray, frameThreshold, frameEdge,
    frameFinal,frameFinalDuplicate;
Mat ROIline;
int LeftLanePos, RightLanePos, frameCenter, laneCenter, result;

RaspiCam_Cv Camera;

stringstream ss, ss1, ss2;

vector<int> histrogramLane;

Point2f Source[] = {Point2f(38,160), Point2f(382,160), Point2f(5,210), Point2f(415,210)};//
    definirea puctelor pt aria de interes
Point2f Destination[] = {Point2f(80,0), Point2f(280,0), Point2f(80,240), Point2f(280,240)};


//Machine learning variables
CascadeClassifier Stop_Sign_Cascade, Object_Cascade;
Mat frame_stop, RoI_Stop, gray_Stop, frame_object, RoI_Object, gray_Object;
vector <Rect> Stop, Object;
int dist_stop, dist_object;


void Setup ( int argc,char **argv, RaspiCam_Cv &Camera )
  {
    Camera.set ( CAP_PROP_FRAME_WIDTH,  ( "-w",argc,argv,420 ) );
    Camera.set ( CAP_PROP_FRAME_HEIGHT,  ( "-h",argc,argv,240 ) );
    Camera.set ( CAP_PROP_BRIGHTNESS, ( "-br",argc,argv,50 ) );
    Camera.set ( CAP_PROP_CONTRAST ,( "-co",argc,argv,50 ) );
    Camera.set ( CAP_PROP_SATURATION,  ( "-sa",argc,argv,50 ) );
    Camera.set ( CAP_PROP_GAIN,  ( "-g",argc,argv ,50 ) );
    Camera.set ( CAP_PROP_FPS,  ( "-fps",argc,argv,50 ));

}

void Perspective() //connects the points and creates a perspective, Scalar() defines the line
    color of the connecting lines
{
  line(frame, Source[0], Source[1], Scalar(0,0,255), 2);
  line(frame, Source[1], Source[3], Scalar(0,0,255), 2);
  line(frame, Source[3], Source[2], Scalar(0,0,255), 2);
  line(frame, Source[2], Source[0], Scalar(0,0,255), 2);
  /*
  line(frame, Destination[0], Destination[1], Scalar(0,255,0), 2);
  line(frame, Destination[1], Destination[3], Scalar(0,255,0), 2);
  line(frame, Destination[3], Destination[2], Scalar(0,255,0), 2);
  line(frame, Destination[2], Destination[0], Scalar(0,255,0), 2);
  */
  Matrix = getPerspectiveTransform(Source, Destination); //transforms the source perspective
    into destination perspective
```

```
60
61   warpPerspective(frame,framePerspective, Matrix, Size(400,240));//perspective transform from
       the initial frame
62
63 }
64
65 void Capture()  // video capturing function
66 {
67   Camera.grab();
68   Camera.retrieve(frameBGR);
69   cvtColor(frameBGR, frame, COLOR_BGR2RGB);//bgr to rgb color space convert
70   cvtColor(frameBGR, frame_stop, COLOR_BGR2RGB);
71   cvtColor(frameBGR, frame_object, COLOR_BGR2RGB);
72
73
74 }
75
76 void Threshold() //applies an image transform to obtain gray tones and finally a black and
       white image
77 {
78   cvtColor(framePerspective, frameGray, COLOR_RGB2GRAY);
79   inRange(frameGray,150, 255,frameThreshold);
80   Canny(frameGray, frameEdge, 100, 500, 3, false); //canny edge detetion function
81   add(frameThreshold, frameEdge, frameFinal);//merging the black and white image with the
       canny edge detection image
82   cvtColor(frameFinal, frameFinal, COLOR_GRAY2RGB);//we cannot convert from gray to color,
       used for something else
83   cvtColor(frameFinal, frameFinalDuplicate, COLOR_RGB2BGR);//so the bottom image won't be cut
84 }
85
86
87 void Histrogram()
88 {
89   histogramLane.resize(360); //make it 360 length
90   histogramLane.clear();
91
92   for(int i=0; i<360; i++) //it gives the width of the frame wich was changed to 360 for
       easier computations
93   {
94     ROIline  = frameFinalDuplicate(Rect(i,140,1,100)); //process the 360 tiny strips from
       reason of interest(ROI)
95     divide(255, ROIline, ROIline);
96     histogramLane.push_back((int)(sum(ROIline)[0]));// sum of pixels
97
98   }
99 }
100
101 void LaneFinder()
102 {
103 vector<int>:: iterator LeftPtr;
104 LeftPtr = max_element(histogramLane.begin(), histogramLane.begin() + 130); //search for the
       first lane in histogram
105 LeftLanePos = distance(histogramLane.begin(), LeftPtr);
106
107 vector<int>:: iterator RightPtr;
108 RightPtr = max_element(histogramLane.begin() + 230, histogramLane.end()); //search for
       second lane in histogram
109 RightLanePos = distance(histogramLane.begin(), RightPtr);
110
111 line(frameFinal, Point2f(LeftLanePos, 0), Point2f(LeftLanePos, 240), Scalar(0,255,0), 2); //
       draw first line
112 line(frameFinal, Point2f(RightLanePos, 0), Point2f(RightLanePos, 240), Scalar(0,255,0), 2); //
       draw the last line
113 }
114
115 void LaneCenter()
116 {
117 laneCenter = (RightLanePos-LeftLanePos)/2 + LeftLanePos; //midpoint between the two lines
118 frameCenter = 180; //the middle of the frame to be calibrated to match the middle of the lanes
119
120 line(frameFinal, Point2f(laneCenter, 0), Point2f(laneCenter, 240), Scalar(0,255,0), 3); //the
       middle line between the two lanes drawn with green
121 line(frameFinal, Point2f(frameCenter, 0), Point2f(frameCenter, 240), Scalar(255,0,0), 3); //
       the middle line of the frame drawn with blue
122
123 result = laneCenter-frameCenter;
```

```
124 }
125
126
127 void StopSignDetection()
128 {
129   if(!Stop_Sign_Cascade.load("//home//pi//Desktop//Integrare machine learning//Stop_cascade5.
      xml"))
130   {
131   printf("Unable to open stop cascade file");
132   }
133
134   RoI_Stop = frame_stop(Rect(210,0,210,240));
135   cvtColor(RoI_Stop, gray_Stop, COLOR_RGB2GRAY);
136   cvtColor(RoI_Stop, RoI_Stop, COLOR_BGR2RGB);
137   equalizeHist( gray_Stop, gray_Stop); //equlalizes all the intesities of the garyscale image
138   Stop_Sign_Cascade.detectMultiScale(gray_Stop, Stop);  //function in cascade class detecting
      stops and puttting them in vector Stop
139
140   for(int i=0; i<Stop.size(); i++)
141   {
142   Point p1(Stop[i].x, Stop[i].y);
143   Point p2(Stop[i].x + Stop[i].width, Stop[i].y + Stop[i].height); //use the points to draw a
      rectangle aropund the stop sign
144
145   rectangle(RoI_Stop, p1, p2, Scalar(0, 0 ,255), 2);
146   putText(RoI_Stop, "Stop Sign", p1, FONT_HERSHEY_PLAIN,1 , Scalar(0 , 0, 255), 2);
147
148   dist_stop = (-0.8423) * (p2.x - p1.x) + 77.647; //linear equation for obtaining the distance
149   ss1.str(" ");
150   ss1.clear();
151   ss1<<"D="<<dist_stop<<"cm";
152   putText(RoI_Stop, ss1.str(), Point2f(1,130), 0, 1, Scalar(0,0,255), 2);
153
154   }
155
156 }
157
158
159
160 void ObjectDetection()
161 {
162   if(!Object_Cascade.load("//home//pi//Desktop//Integrare machine learning//Object_cascade.xml
      "))
163   {
164   printf("Unable to open object cascade file");
165   }
166
167   RoI_Object = frame_object(Rect(0,0,420,240));
168   cvtColor(RoI_Object, gray_Object, COLOR_RGB2GRAY);
169   cvtColor(RoI_Object, RoI_Object, COLOR_BGR2RGB);
170   equalizeHist( gray_Object, gray_Object); //equlalizes all the intesities of the garyscale
      image
171   Object_Cascade.detectMultiScale(gray_Object, Object);  //function in cascade class detecting
       stops and puttting them in vector Stop
172
173   for(int i=0; i<Object.size(); i++)
174   {
175   Point p1(Object[i].x, Object[i].y);
176   Point p2(Object[i].x + Object[i].width, Object[i].y + Object[i].height); //use the points to
       draw a rectangle aropund the stop sign
177
178   rectangle(RoI_Object, p1, p2, Scalar(0, 0 ,255), 2);
179   putText(RoI_Object, "Object", p1, FONT_HERSHEY_PLAIN,1 , Scalar(0 , 0, 255), 2);
180
181   dist_object = (-0.8423) * (p2.x - p1.x) + 77.647; //linear equation for the object
182   ss2.str(" ");
183   ss2.clear();
184   ss2<<"D="<<dist_object<<"cm";
185   putText(RoI_Object, ss2.str(), Point2f(1,130), 0, 1, Scalar(0,0,255), 2);
186
187   }
188
189 }
190
191
192
```

```
193
194
195 int main(int argc, char **argv)
196 {
197   using namespace std::this_thread;     // sleep_for, sleep_until
198   using namespace std::chrono_literals;
199   wiringPiSetup();
200   pinMode(21, OUTPUT);
201   pinMode(22, OUTPUT);
202   pinMode(23, OUTPUT);
203   pinMode(24, OUTPUT);
204   Setup(argc, argv, Camera);
205   cout<<"Connecting to camera"<<endl;
206   if(!Camera.open())
207     {
208     cout<<"Failed to connect"<<endl;
209     return -1;
210     }
211     cout<<"Camera Id = "<<Camera.getId()<<endl;
212
213     while(1)
214     {
215     auto start = std::chrono::system_clock::now();//fps measurement
216
217     Capture();
218     Perspective();
219     Threshold();
220     Histrogram();
221     LaneFinder();
222     LaneCenter();
223     StopSignDetection();
224     ObjectDetection();
225
226       if(dist_stop>5 && dist_stop < 20) //first condition put because if there is no stop sign
       the dist_stop variable will be 0
227       {
228       digitalWrite(21, 1);
229       digitalWrite(22, 1);
230       digitalWrite(23, 1);   //send 7
231       digitalWrite(24, 0);
232       cout<<"Stop Sign"<<endl;
233       dist_stop = 0;
234       sleep_for(8s);
235       digitalWrite(21, 0);                    //stop sign detection
236       digitalWrite(22, 0);
237       digitalWrite(23, 0);   //send 0
238       digitalWrite(24, 0);
239       cout<<"Forward"<<endl;
240       sleep_for(500ms);
241
242       goto Stop_Sign;
243       }
244
245
246       if(dist_object>5 && dist_object < 20) //first condition put because if there is no stop
      sign the dist_stop variable will be 0
247       {
248       digitalWrite(21, 1);
249       digitalWrite(22, 1);
250       digitalWrite(23, 1);   //send 7    obstacle detection
251       digitalWrite(24, 0);
252       cout<<"Obstacle"<<endl;
253       dist_stop = 0;
254
255
256       goto Obstacle;
257       }
258
259
260
261
262       if(result >= -10 && result <= 10)
263       {
264       digitalWrite(21, 0);
265       digitalWrite(22, 0);
266       digitalWrite(23, 0);   //send 0
```

```
267       digitalWrite(24, 0);
268       cout<<"Forward"<<endl;
269
270       }
271
272       /*else if(result > 5 && result < 10)
273       {
274       digitalWrite(21, 1);
275       digitalWrite(22, 0); //send 1
276       digitalWrite(23, 0);
277       digitalWrite(24, 0);
278       cout<<"Right1"<<endl;
279
280       }*/
281
282       else if(result >10 && result < 15)
283       {
284       digitalWrite(21, 0);
285       digitalWrite(22, 1); //send 2
286       digitalWrite(23, 0);
287       digitalWrite(24, 0);
288       cout<<"Right2"<<endl;
289
290       }
291
292       else if(result >= 15)
293       {
294       digitalWrite(21, 1);
295       digitalWrite(22, 1); //send 3
296       digitalWrite(23, 0);
297       digitalWrite(24, 0);
298       cout<<"Right3"<<endl;
299
300       }
301       /*else if(result < -5 && result > -10)
302       {
303       digitalWrite(21, 0);
304       digitalWrite(22, 0); //send 4
305       digitalWrite(23, 1);
306       digitalWrite(24, 0);
307       cout<<"Left1"<<endl;
308
309       }*/
310
311       else if(result < -10 && result > -15)
312       {
313       digitalWrite(21, 1);
314       digitalWrite(22, 0); //send 5
315       digitalWrite(23, 1);
316       digitalWrite(24, 0);
317       cout<<"Left2"<<endl;
318
319       }
320
321       else if(result <= -15)
322       {
323       digitalWrite(21, 0);
324       digitalWrite(22, 1); //send 6
325       digitalWrite(23, 1);
326       digitalWrite(24, 0);
327       cout<<"Left3"<<endl;
328
329       }
330
331   Stop_Sign:
332   Obstacle:
333
334      ss.str(" ");
335    ss.clear();
336    ss<<"Result = "<<result;
337    putText(framePerspective, ss.str(), Point2f(1,50), 0,1,Scalar(150,0,150), 2);//writes the
         result between the center of the frame and the middle of the lanes
338
339    namedWindow("Video", WINDOW_KEEPRATIO);
340    moveWindow("Video", 80, 100);
341    resizeWindow("Video", 640, 480);
```

```
342   imshow("Video", frameBGR);
343
344   namedWindow("Perspective", WINDOW_KEEPRATIO);
345   moveWindow("Perspective", 730, 100);
346   resizeWindow("Perspective", 640, 480);
347   imshow("Perspective", framePerspective);
348
349   namedWindow("FinalImage", WINDOW_KEEPRATIO);
350   moveWindow("FinalImage", 80, 590);
351   resizeWindow("FinalImage", 640, 480);
352   imshow("FinalImage", frameFinal);
353
354   namedWindow("Stop Sign", WINDOW_KEEPRATIO);
355   moveWindow("Stop Sign", 730, 590);
356   resizeWindow("Stop Sign", 640, 480);
357   imshow("Stop Sign", RoI_Stop);
358
359   namedWindow("Object", WINDOW_KEEPRATIO);
360   moveWindow("Object", 1300, 100);
361   resizeWindow("Object", 640, 480);
362   imshow("Object", RoI_Object);
363
364
365   waitKey(1);
366
367   auto end = std::chrono::system_clock::now();
368   std::chrono::duration<double> elapsed_seconds = end-start;
369
370   float t = elapsed_seconds.count();
371   int FPS = 1/t;
372   cout<<"FPS = "<<FPS<<endl;
373
374   }
375   return 0;
376 }
```

# Annex C
# Image Capture Code

```
#include <opencv2/opencv.hpp>
#include <raspicam_cv.h>
#include <iostream>
#include <chrono>
#include <ctime>
#include <wiringPi.h>
using namespace std;
using namespace cv;
using namespace raspicam;
Mat frame;
void Setup ( int argc,char **argv, RaspiCam_Cv &Camera )
{
Camera.set ( CAP_PROP_FRAME_WIDTH, ( "-w",argc,argv,420 ) );
Camera.set ( CAP_PROP_FRAME_HEIGHT, ( "-h",argc,argv,240 ) );
Camera.set ( CAP_PROP_BRIGHTNESS, ( "-br",argc,argv,50 ) );
Camera.set ( CAP_PROP_CONTRAST ,( "-co",argc,argv,50 ) );
Camera.set ( CAP_PROP_SATURATION, ( "-sa",argc,argv,50 ) );
Camera.set ( CAP_PROP_GAIN, ( "-g",argc,argv ,50 ) );
Camera.set ( CAP_PROP_FPS, ( "-fps",argc,argv,0));
}
int main(int argc, char** argv)
{
RaspiCam_Cv Camera;
Setup(argc, argv, Camera);
cout<<"Connecting to camera"<<endl;
if (!Camera.open())
{
cout<<"Failed to Connect"<<endl;
}
cout<<"Camera Id = "<<Camera.getId()<<endl;
for (int i=0; i<90 ; i++)
{
Camera.grab();
Camera.retrieve(frame);
cvtColor(frame,frame, COLOR_BGR2GRAY);
imshow("Sample", frame);
imwrite("Obj"+to_string(i)+".jpg" , frame);
waitKey();
}
return 0;
}
```