Spring -2021

EEE 494

Final design report

e-Visoneers

The report is submitted to: Dr. Cem Direkoğlu

Members:

Name	ID	Phone number
Abdelaziz Al-Najjar	2269512	+966503031947
Mustafa Almoghrabi	2329142	+905528032469
Omar Ammar	2203453	+905338430960

Contents

Executive summary 6

Introduction 7

Previous designs 8

Design Description 9

Top-down system description 9

System components 12

Arduino UNO: 12

Raspberry Pi: 13

L298N Motor Driver: 14

6v DC motors: 15

Bluetooth module HC-05: 15

Raspberry pi camera module: 16

Overall description of the system 17

System level flowchart 18

Description of individual sub-systems 19

Raspberry pi subsystem 19

Arduino Uno subsystem 20

Related algorithms 21

Machine vision algorithms 21

Control Algorithm 24

Results and analysis of performance test 27

Arduino testing 27

Testing preparation 28

Lane detection testing 31

Object detection testing 32
User Manual 34
Required Software: 34
Steps to use the car: 34
Important Notes in Autonomous mode 39
Complications and solutions 40
Problems regarding the power systems 40
Problems regarding control system: 41
Problems regarding machine vision part: 41
Further discussion 42
Safety issues: 42
Societal impact: 42
Potential environmental effects: 42
Deliverables 43
Total cost 44
Conclusion 45
References 46
Appendices 47

Figure 1 Layer 1: 2D top view 9
Figure 2 Layer 2, 2D, Top view 10
Figure 3 Layer 3 , 2D top view 11
Figure 4 Arduino UNO 12
Figure 5 Raspberry pi 3 B+ 13
Figure 6 L298 motor driver 14
Figure 7 DC motors 15

Figure 8 Bluetooth module 15
Figure 9 Raspicam v2 16
Figure 10 System Block Diagram 17
Figure 11 System flowchart 18
Figure 12 Raspberry pi subsystem 19
Figure 13 Arduino UNO subsystem 20
Figure 14 General flow chart 21
Figure 15 Lane detector algorithm flow chart 22
Figure 16 Object detection flow chart 23
Figure 17 Mode detection 24
Figure 18 Autonomous mode concept 25
Figure 19 Manual mode concept 26
Figure 20 Simulation testing 27
Figure 21 Test track 28
Figure 22 Traffic lights 29
Figure 23 Stop sign 29
Figure 24 Object car 30
Figure 25 E-visioneers RC car 30
Figure 26 Lane detection test (P1) 31
Figure 27 Lane detection test (P2) 31
Figure 28 Object detection test 32
Figure 29 Object detection test (P2) 33
Figure 30 Booting up switch 34
Figure 31: VNC viewer 35
Figure 32: Raspberry Pi's Desktop 35
Figure 33: E-visioneers folder 35
Figure 34: Execute the code 36
Figure 35: Lane and objects detection views 36
Figure 36; Arduino Bluetooth RC Car Application 37
Figure 37: Mobile app Connecting to RC Car – Step 1 37
Figure 38: Mobile app Connecting to Car - step 2 37

Figure 39: Mobile app Connecting to RC Car - step 3 38
Figure 40: Switching between Manual and Autonomous Modes 38
Figure 41: Mode indicator on Mobile app 38

Executive summary

This project aims to develop a self-driving car that can stand to solve the current driving system's problems, such as traffic delays, and more importantly, traffic collisions caused by the human driver. As [1] stated, more than 1.35 million deaths happen yearly due to driver errors, and the number is increasing every year. Moreover, the self-driving car with a machine vision-based warning system will also provide the car market variety of applications, like shipping items fast, transportation, and fast emergency transportation. Designing the self-driving car will go through four significant steps:

• Dividing the project into smaller but precise tasks

• Simulating each block separately

• Integrating all the blocks into one major block

• Fix the problems that occur when integration

e-Visioneers has three members with different academic interests, each of which has a different part in implementing the project. Abdelaziz Al-Najjar was responsible for generating the machine vision algorithm used for lane detection and recognizing objects like other cars, humans, animals, and obstacles. Mustafa Almoghrabi was responsible for designing the car's control system, which includes the DC motor and the front wheels. He also developed the communication system between the vehicle and the user. Omar Ammar used his knowledge in power electronics to design the car's power system, which must power all particles in the car for a sufficient time.

The self-driving car proposed had two control modes. Manual with the help of a warning system supported with machine vision algorithms. Autonomous as the car drives itself and take decisions at the right time based on the HD camera. One of the significant challenges in this project is to make a well-built prototype with a reasonable amount of money. According to our calculations, we estimated that this project would cost around 1400 TL. The team is done with the research and design phase and considering the available options to have an efficient product.

Introduction

The existence of cars started to become dangerous in man-kind life, especially with the massive increase in cars. This increase reached the extent that 1.35 million deaths worldwide due to vehicle crashes [1]. This considerable number of accidents is mainly due to human error and inattention. From here, the idea of self-driving cars came as a good assistant in improving road safety and offering new mobility options for millions of people, especially old-aged and people with low vision. The proposed self-driving car guides itself using visual information from its environment with an HD camera. The data obtained from the camera are sent directly to the Raspberry pi that plays a significant role in sending control signals to different parts of the car. It is worth mentioning that the proposed car will be existent with two modes: Manual with the help of a warning system supported with machine vision algorithms to detect all kinds of obstacles, and autonomous as the car will drive itself and take decisions at the right time. Many companies started to invest in these self-driving cars like "Waymo," which began in 2016 in developing, and in 2020 they started self-driving taxi service [2]. The following project can be another step towards more complex and more significant scale projects. It is considered relatively cheap with a cost of 1400 TL. This final design report will explain the design and algorithms used in a self-driving car and describe the subsystems' functionality inside it. It will also explain the concepts used in image processing and control systems, in addition to the results of the tests conducted until reaching the final product. This report also gives user manual explaining in detail how the car can be used, and at the end, some discussions about this project was presented.

Previous designs

In this section is a quick review of previous autonomous RC car designs, giving a brief background and comparing it to E-Visioneer's project:

1. **WPI autonomous RC car:**
    - Full Autonomous RC car.
    - Detecting objects in environments surrounded
    - Runs only on battery power.

The design described in this report is better since it can detect lanes, runs on 2 different power supplies in case a power supply fails.

2. **Donkey car:**
    - Full Autonomous RC car with camera.
    - Utilizes Raspberry pi only without Arduino.
    - Detects objects and Lanes.

The design described in this report is better since it can detect lanes, utilizes two different microprocessors on as main and the other as slave in order not to overheat the Raspberry Pi.

3. **Kyle's autonomous RC car:**
    - Full autonomous RC car with Ultra sonic sensors.
    - No camera to identify objects.
    - No machine vision algorithms since there is no camera.

This design described in this report is better since it identifies objects using machine vision algorithms and moves accordingly, beside it can detect lanes and move on certain track.

Design Description

Top-down system description

Note: All sketches are just for illustration, meaning that scales are not 100% real, in order to show everything clearly.

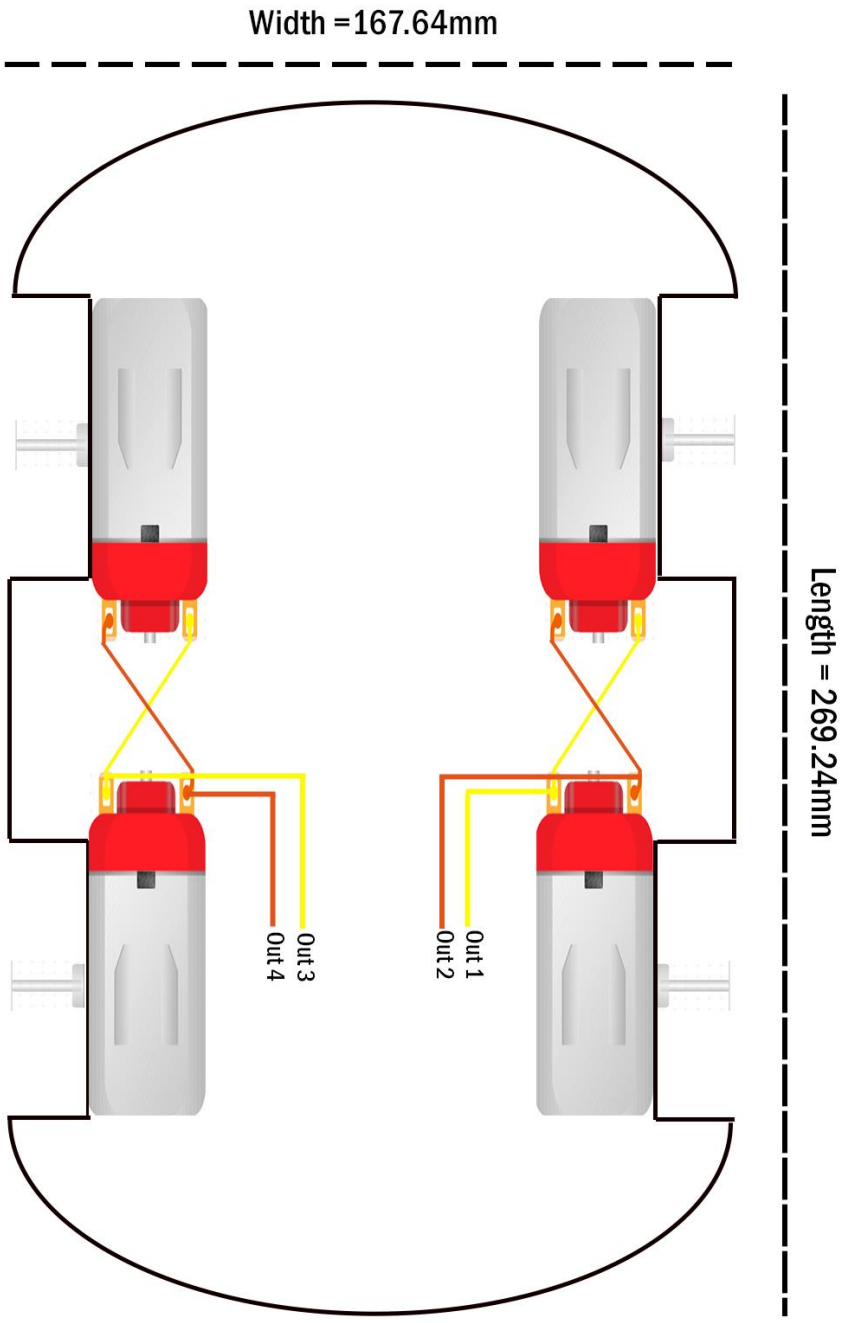

Figure 1 Layer 1: 2D top view

** All wires labeled with "Out" next to it are going to be connected to its corresponding place L298N driver that is located in the second layer.*

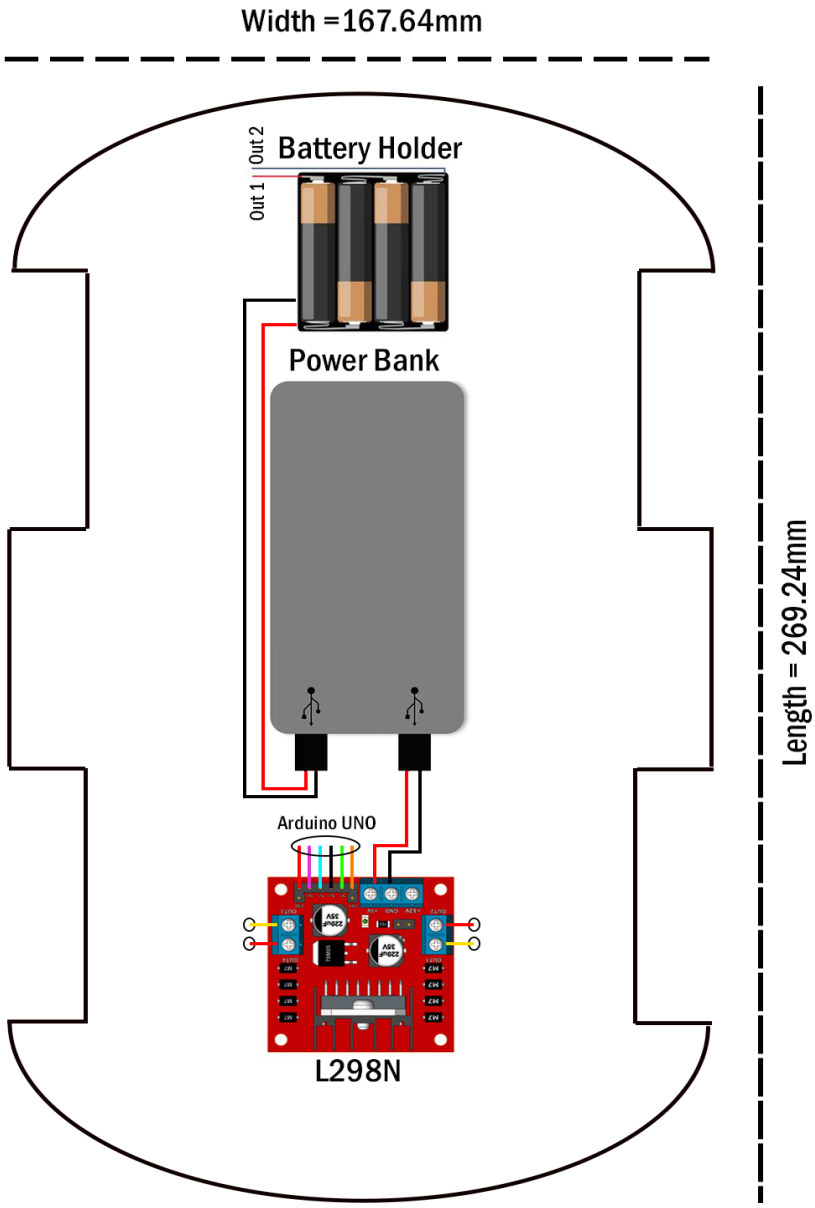

Figure 2 Layer 2, 2D, Top view

** The two wires coming out of the USB cable connected to the power bank labeled with "Out 1 & Out 2" are going to be connected to the breadboard in third layer.*

*** Al the six wires coming out of the L298N driver are going to be connected to Arduino Uno in the third layer*

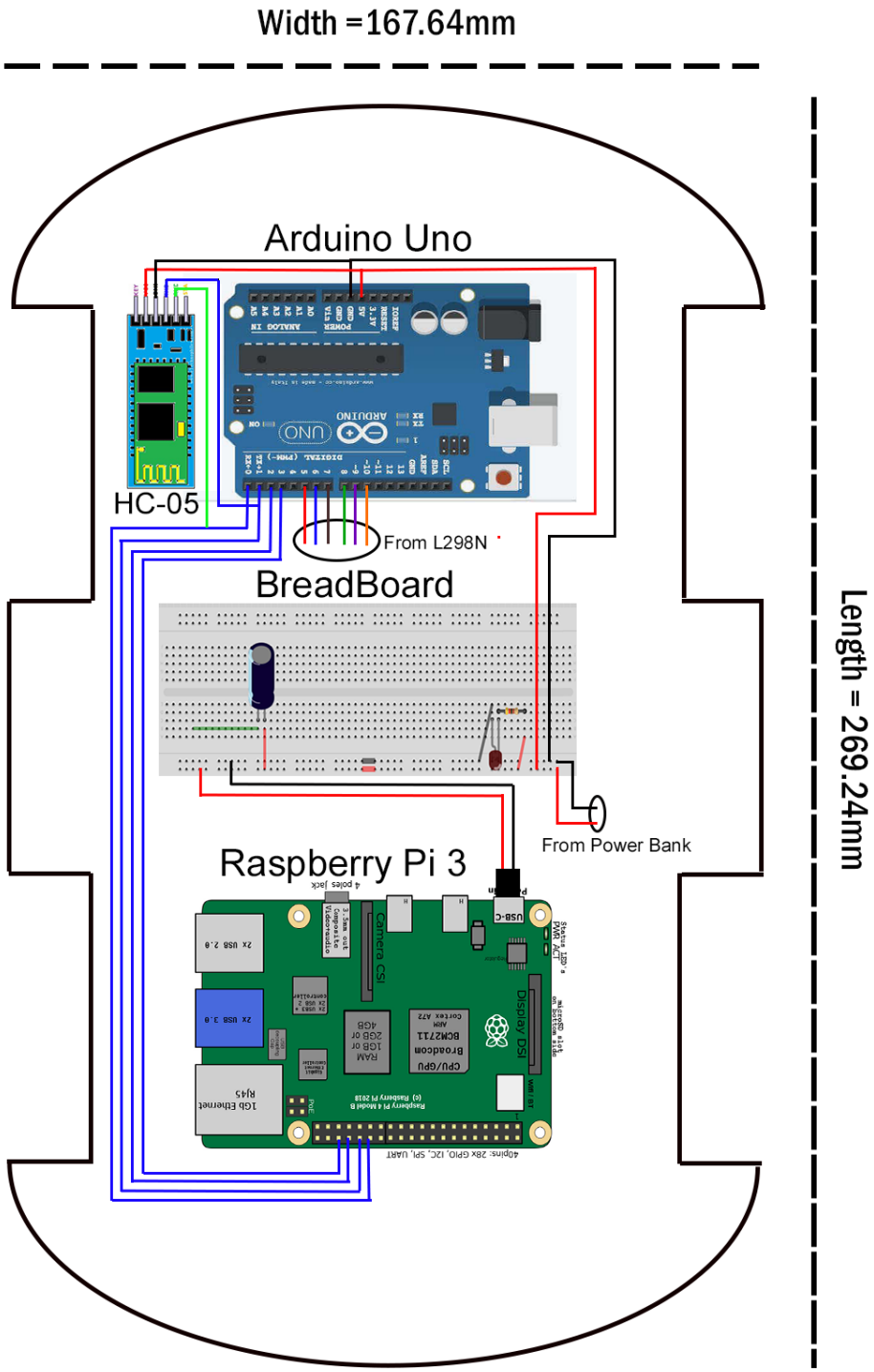

Figure 3 Layer 3 , 2D top view

System components

Starting with some of the components used in this project, The following section will explain all of the components used individually.

Arduino UNO:

The well-known microcontroller Arduino UNO was used as a slave device in the design (as will be explained later). This device will receive commands from the master device (Raspberry pi) and control the 4 DC motors, which are connected to the wheels of the car. The following figure shows Arduino UNO individually.

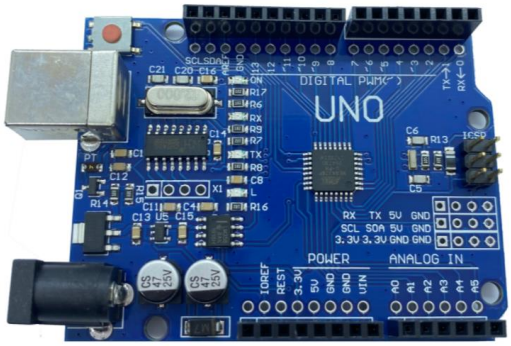

Figure 4 Arduino UNO

Raspberry Pi:

This device was used in the design as a master device. Its main task is to process the images coming from the camera device, and then decide the right movement by sending the commands to the slave device (Arduino UNO) as mentioned before. The algorithms used in machine vision process will be discussed later in details. The following figure shows the Raspberry pi used in the design.

Figure 5 Raspberry pi 3 B+

L298N Motor Driver:

L298n motor driver is a dual H-bridge motor driver which allows the control of speed and direction of DC motors at the same time. The module can drive DC motors that have voltages between 5 and 35V, with a peak current up to 2A. The following figure shows the DC motor drive used in the design.

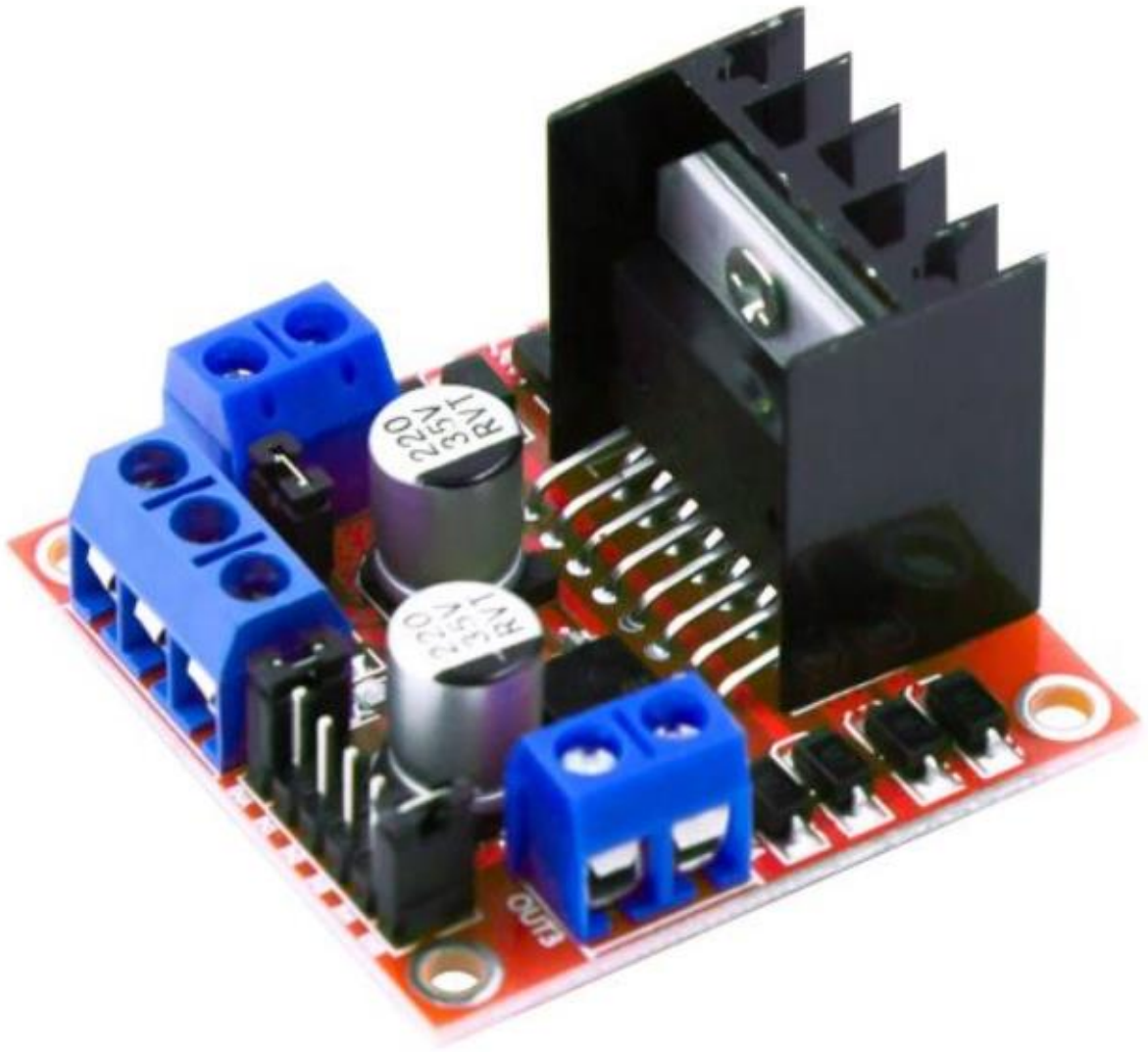

Figure 6 L298 motor driver

6v DC motors:

4 motors will be used to move the car wheels. The speed of the motors, as mentioned in manual sheet, is 125 rpm, and the torque is 0.8kg, cm. The optimum voltage to drive the motors is about 5 volts. The following figure shows the DC motors used in the design.

Figure 7 DC motors

Bluetooth module HC-05:

The Bluetooth module in the design is used in the manual mode to control the car, and to activate the autonomous mode as will be described later. This module works on a 2.4 GHz frequency with ISM band with a GFSK modulation. The optimal voltage working on it is 3.3v DC. The following image shows the module used in the design.

Figure 8 Bluetooth module

Raspberry pi camera module:

The raspberry pi camera module used in the design to collect the images in front of the car and send it later to the raspberry pi in order to process it. The camera used in our design was Raspicam 2.0. with 8-megapixel native resolution sensor that can support up to 1080p30 video. The following figure shows the Raspicam 2.0 used in the design. .

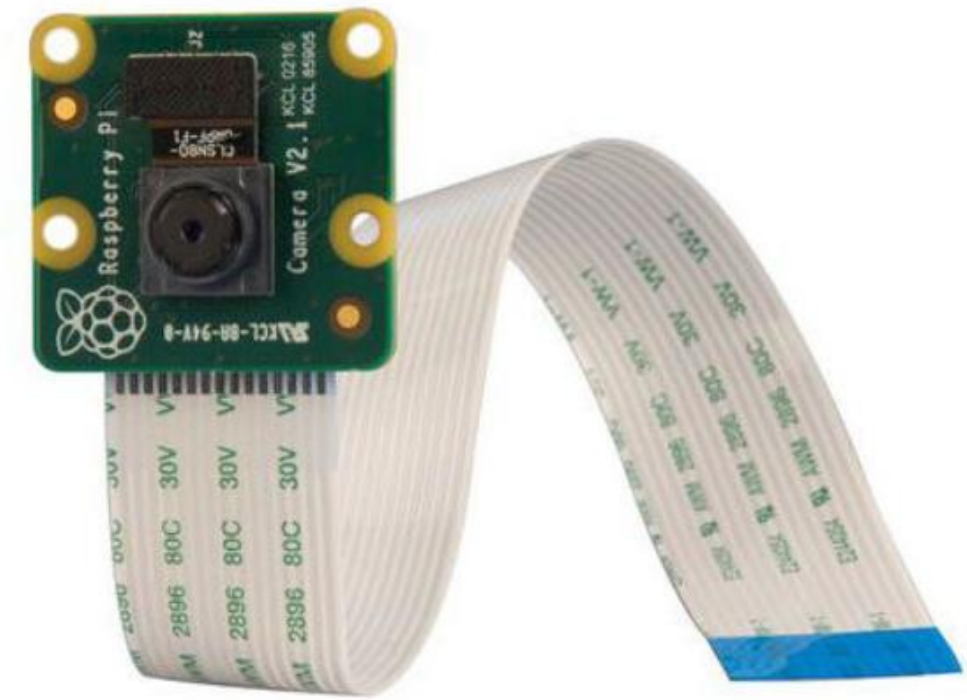

Figure 9 Raspicam v2

Overall description of the system

This system, which was fully designed, consists of 2 main subsystems and peripherals used to control the 4 DC motors connected to 4 wheels; These subsystems are the Raspberry pi subsystem and Arduino subsystem. The peripherals will be an external camera, which will provide images to the system, and the controller, which will work in the manual mode. Basically, Raspberry pi will be defined as a master controller. It will process the images coming from the external camera using machine learning algorithms and then give the right decision accordingly. Meanwhile, Arduino UNO will be recognized as a slave controller and responsible for four DC motors' speed according to the orders coming from the master device (Raspberry pi).

Figure 10 below shows a system block diagram.

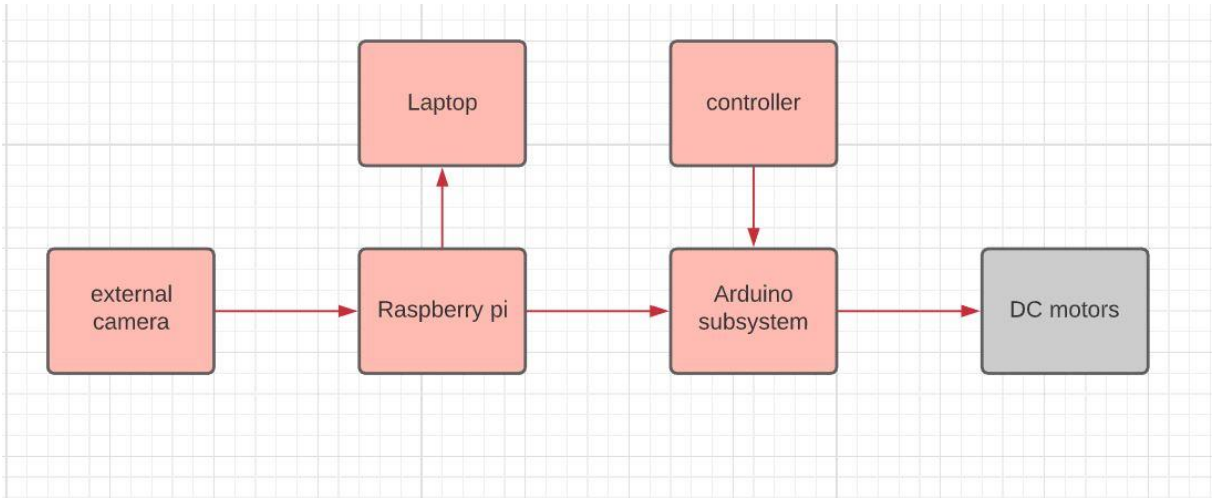

Figure 10 System Block Diagram

System level flowchart

Figure 11 below shows the system level flowchart.

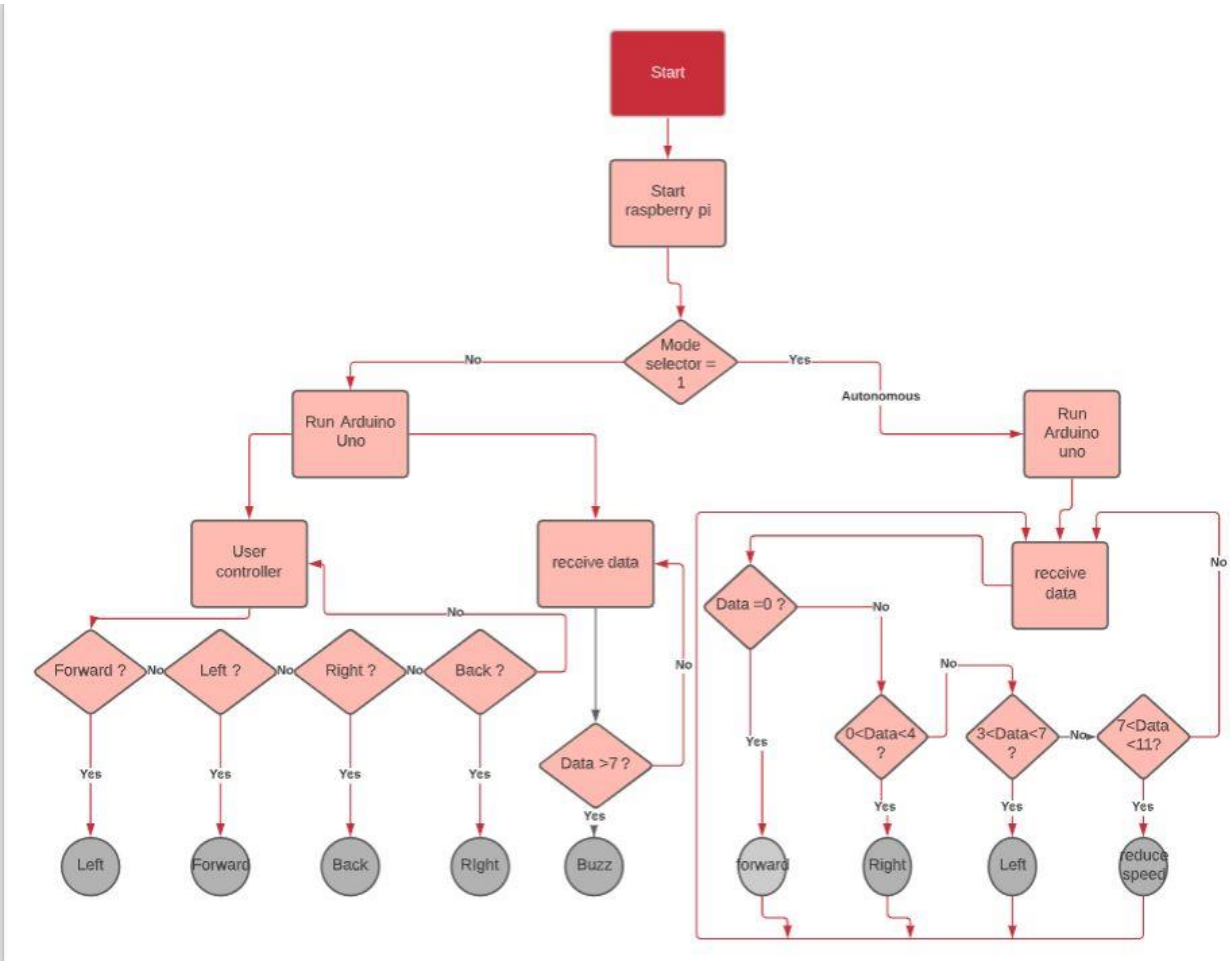

Figure 11 System flowchart

Description of individual sub-systems

Raspberry pi subsystem

Figure 3 below illustrates a block diagram of the Raspberry Pi 3 B+ module subsystem. The Raspberry Pi will be powered up using a power bank of 10000mAh Capacity that delivers 3A and a voltage of 5V; before the power supply, there will be a capacitor of 6400µF and 35 volt tolerance that will have a job of opposing any sudden change in voltage. The raspberry pi 3 will have 1GB of RAM dedicated to executing machine vision algorithms. We decided to add Arduino UNO to the system as a slave device to get the operation from the raspberry pi and perform the controller role. An RPi camera module will be connected to the Raspberry Pi to analyze the environment and execute machine vision algorithms developed by our team members. In Autonomous mode, the Raspberry Pi will analyze the environment using the camera and the machine vision algorithms, sending signals to Arduino Uno to either move, stop, or turn the car. In manual mode, the raspberry pi will do the same job, but instead, it will not control the vehicle; it will only send warning signals to notify the user that there is an object.

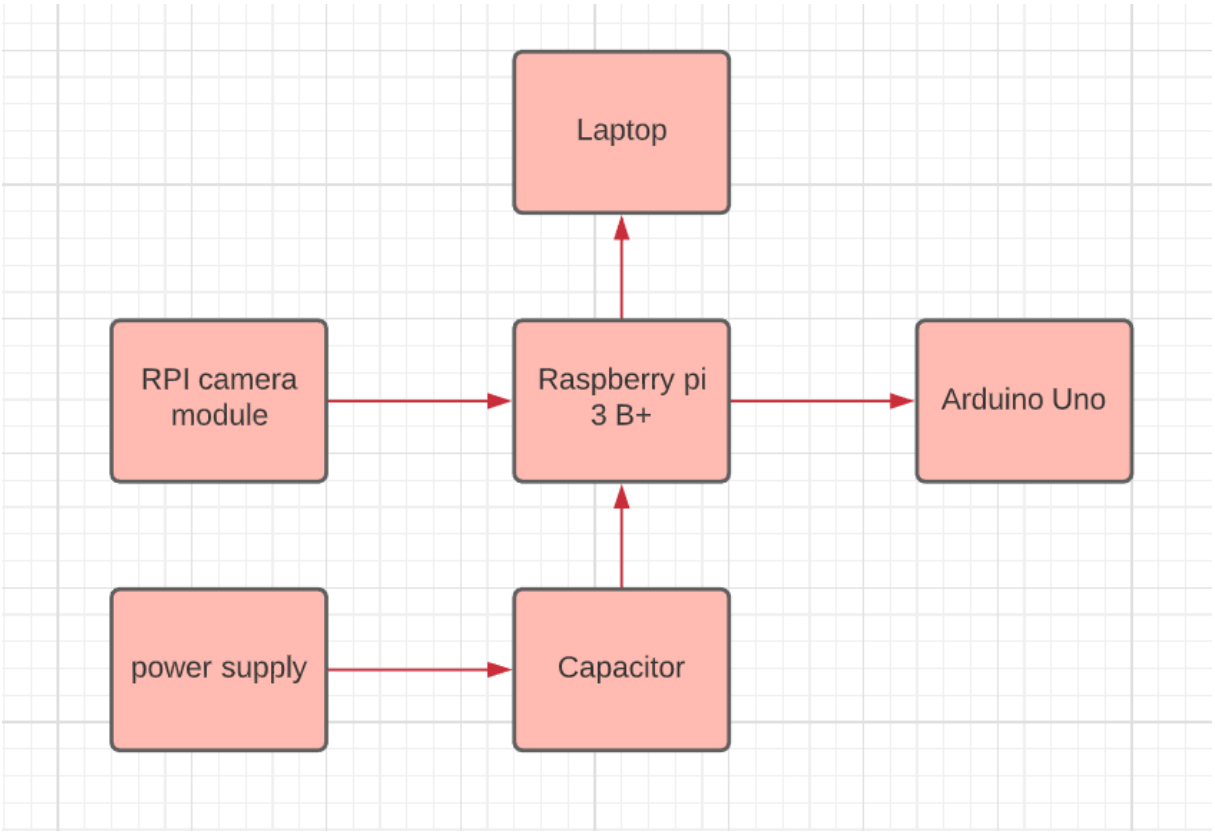

Figure 12 Raspberry pi subsystem

Arduino Uno subsystem

The Figure below shows the flowchart of Arduino UNO utilized in the project. The Arduino UNO will also be supplied by the same power bank used for the Raspberry Pi 3 B+. Firstly, Arduino will be connected to the Raspberry Pi; the Raspberry Pi will provide digital signals in both control and autonomous mode. Secondly, the Arduino will be connected to L298N to provide it with instructions to run or stop the DC motors. Finally, it will be connected to a Bluetooth module HC-05 used in the manual mode; the HC-05 will connect the Arduino with an external controller that decides the car's movement.

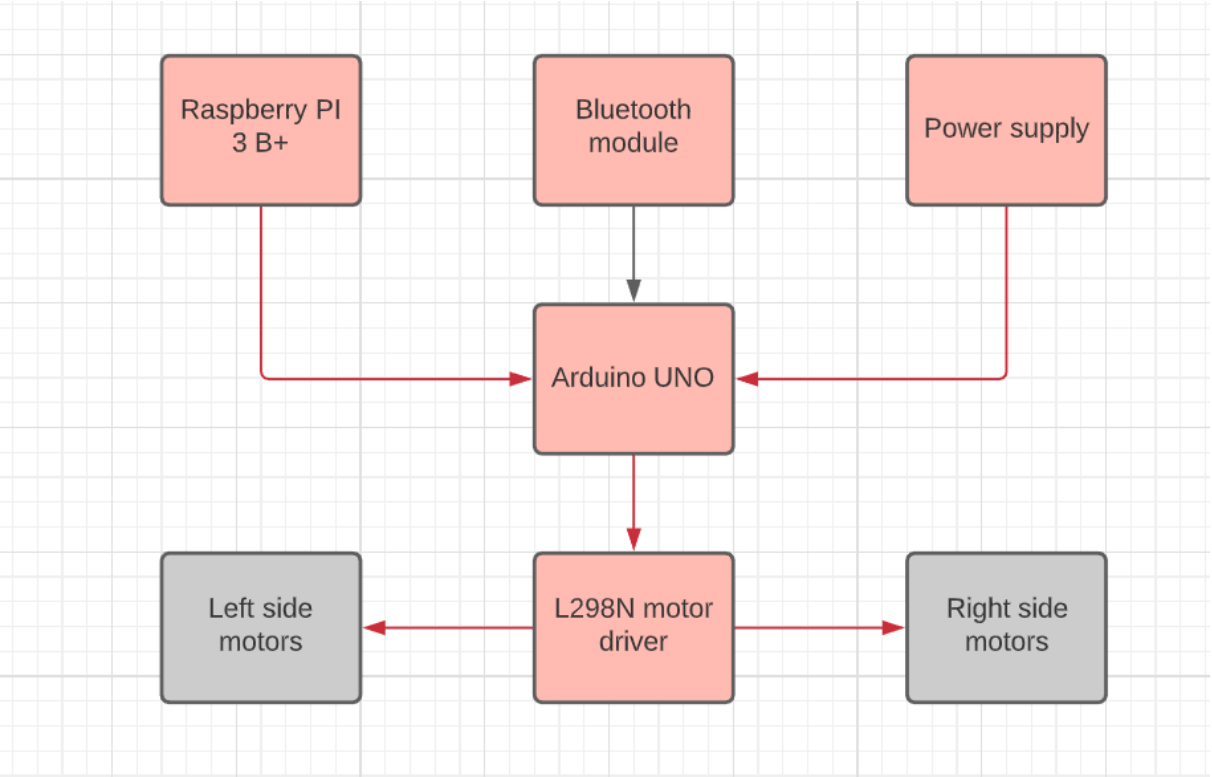

Figure 13 Arduino UNO subsystem

Related algorithms

Machine vision algorithms

During the research process, it has been decided that using the Raspberry pi (master device) for capturing image frames, these frames will go through stages of lane detection and object recognition algorithms simultaneously. After collecting all information from the algorithms, it will send them to the Arduino UNO (the slave device) to perform the car motors' necessary operations, as will be described later.

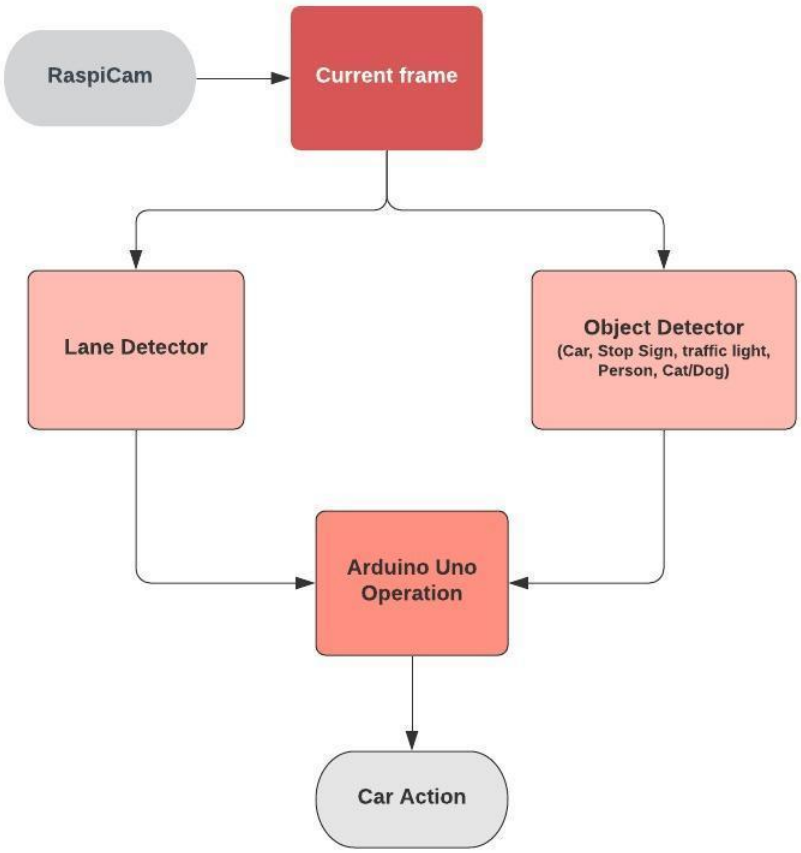

Figure 14 General flow chart

Firstly, the lane detection algorithm which is shown in figure 15 below. This algorithm is enough to detect the lanes which the car will go through next, not what it will go through after 30 or more cm. The reason is thanks to the object detection algorithm; the car can know what will come later. Hence, the algorithm starts by generating the "Region of Interest," which is the first 5 to 15 cm of the road in front of the car. All that is left is to define these lanes as if it were the left or right lane,

and if the road is going right or left. For that, the frame will be divided vertically into two halves. Using the histogram method to define the maximum value for each and store its position. The maximum value in the left half of the frame will represent the left lane, and the maximum value of the right half will represent the right lane. Then by adding both lane's positions and dividing them by 2, and adding the result to the left lane position, we can know the lane's center which later will be subtracted from the frame center and store the value in Result. If the Result is larger than zero, then the information to be sent to the Arduino that the lanes are moving Right. Likewise, Arduino will receive move left order if the Result is smaller than 0. Otherwise, the car will move forward. The test of this algorithm result will be shown in the simulations section.

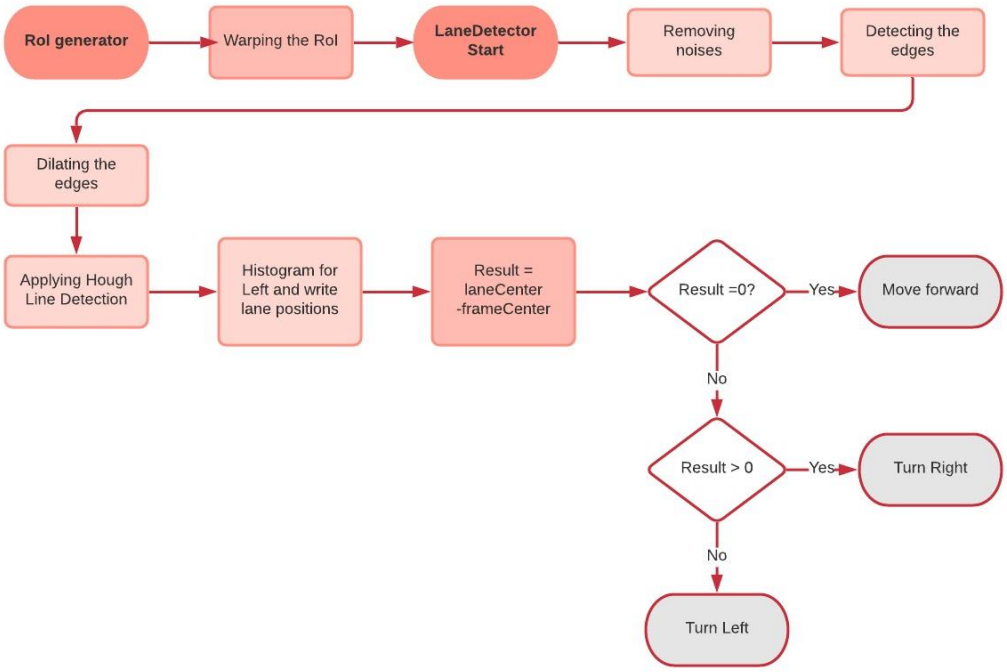

Figure 15 Lane detector algorithm flow chart

Next, the object detection algorithm. The algorithms used could detect cars, stop signs, persons, and animals such as cats and dogs; The algorithm starts by loading the XML trained model into the program, then drawing a box around the detected object in each frame, and finding the linear relation between the box width and the car's distance. In case the distance is less than 20 cm, directly a signal will be sent to the slave device (Arduino) to change a lane (in case of an object), or to wait for couple of seconds in case of finding a stop sign. as shown in Figure 18 below

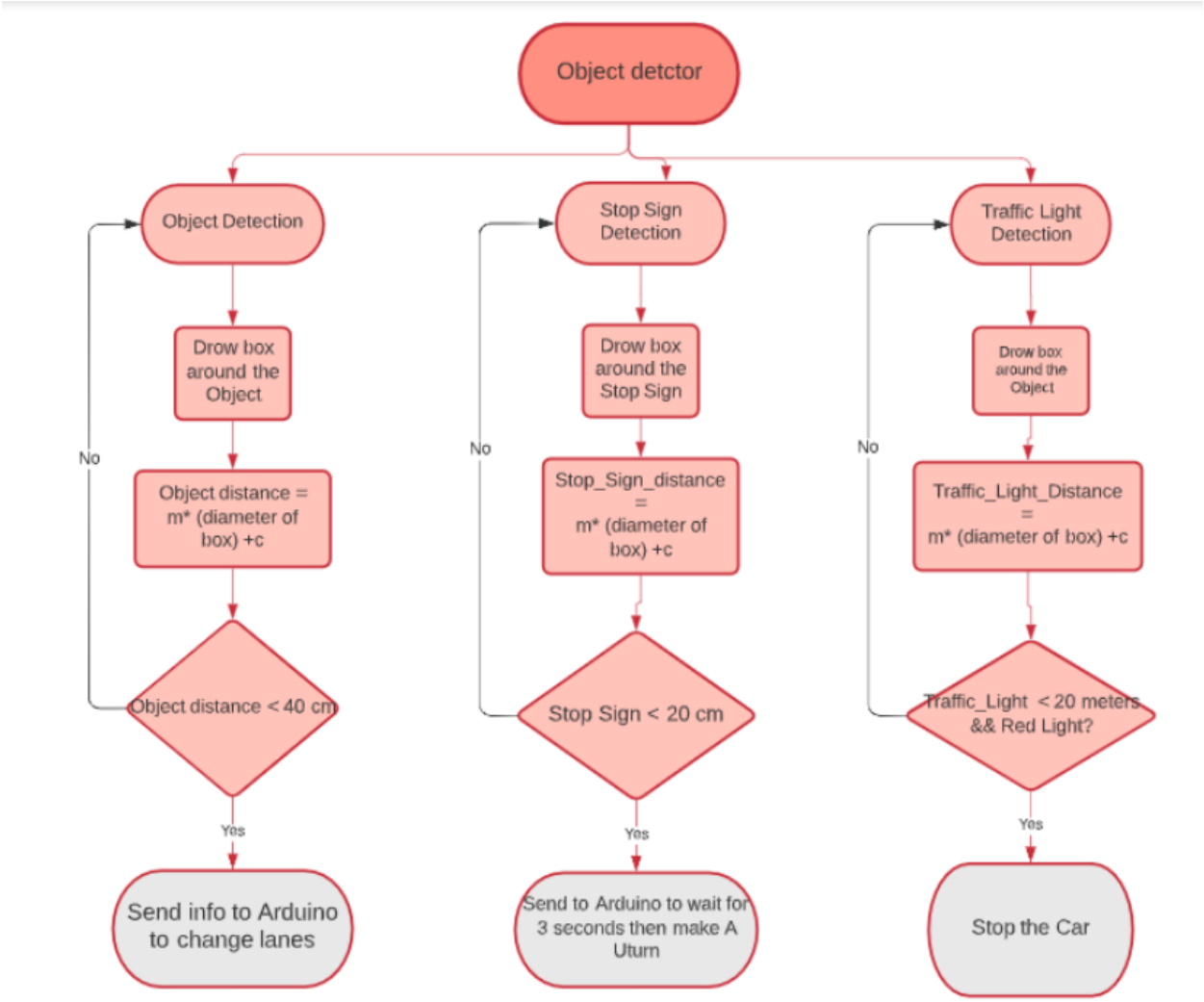

Figure 16 Object detection flow chart

Control Algorithm

The motion of the vehicle, in general, will be detected firstly by the mode desired by the user. One pin will be assigned in Arduino UNO to connect with raspberry pi, which will choose whether the mode is manual or autonomous. Figure 17 below shows the very basic algorithm of the system.

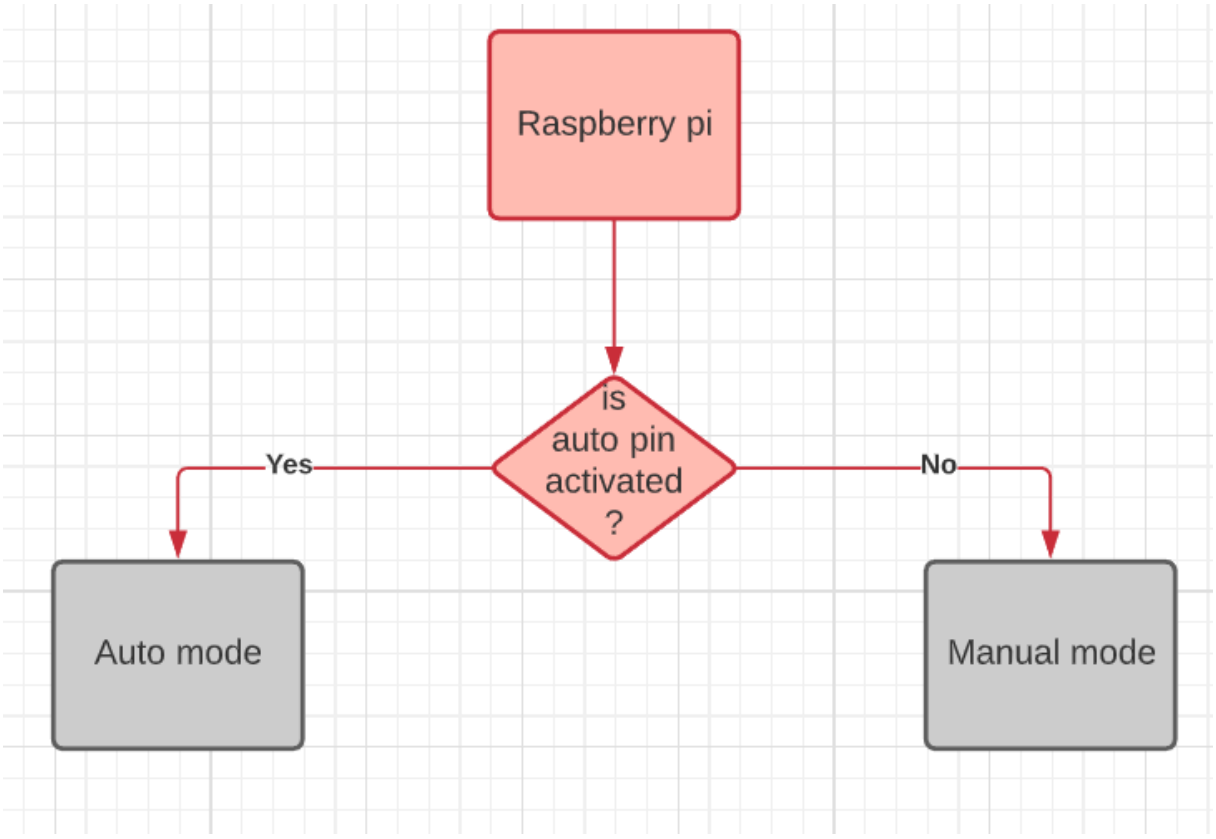

Figure 17 Mode detection

Autonomous Mode concept

As mentioned previously, if the auto pin is activated, the Arduino UNO will start sending forward signals to the right and left side motors. Meanwhile, the Arduino will keep receiving control signals from the master device (Raspberry pi) by connecting 4 digital pins of Arduino with Raspberry pi. which corresponds to 16 different controlling motions.

As mentioned above, the Raspberry pi uses a camera and algorithms to detect the car's position according to the lanes and the positions of any obstacles. Hence two groups of signals from raspberry pi to Arduino Uno can be considered (lane detection signals and object detection signals).

The first group is the lane detection signals. According to these signals, the slave device will consider the basic motions: forward, back, right, and left. the second group is the warning signals of an obstacle; Arduino UNO will control the motors with other kinds of motions like reducing the speed slightly if the object is still far, reducing more if the item became closer, stopping completely if the object is too close, or stopping for a little bit of time if a stop sign has been found. (More complex motions may be added later). Figure 18 below shows a conceptual algorithm for the autonomous mode.

Note that the right and left motions are planned to be considered in three degrees. For example, in the right motion, slightly right, moderately right, and extremely right.

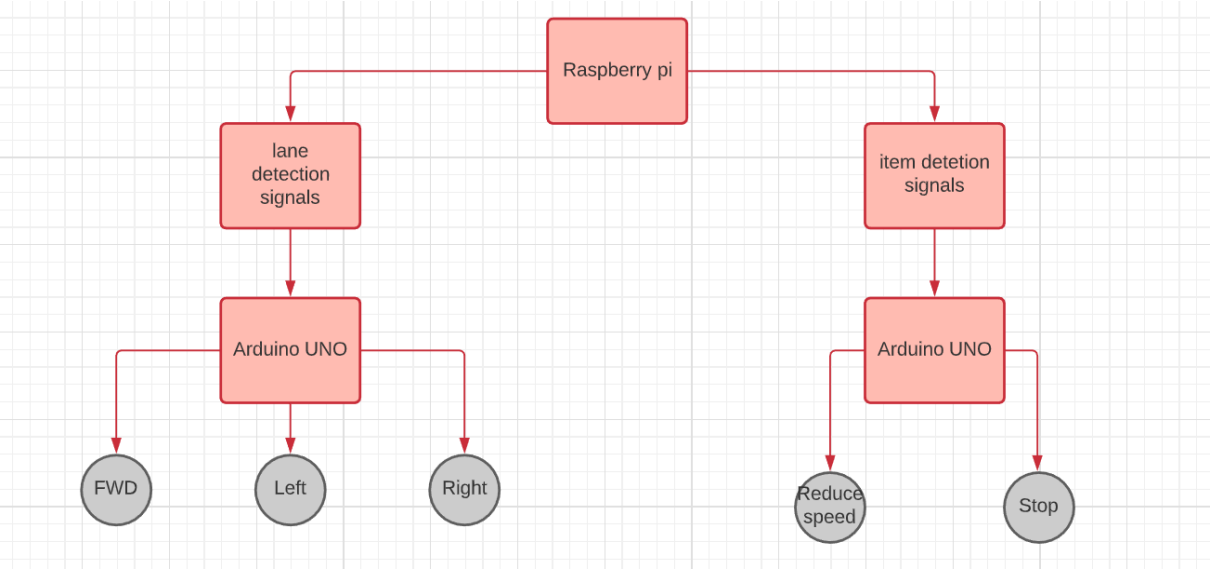

Figure 18 Autonomous mode concept

Manual mode concept:

In this mode, an external controller will be used. It is decided to be a Bluetooth device connected to the Arduino UNO and driven by a mobile application.

This mode's idea is that when the autonomous pin is not activated, the Bluetooth module will be activated and start to receive signals from the external controller. Here the group of lane detection signals received by the Raspberry pi will not be considered. Unlike the obstacle detection signals that once received, Arduino will send an output signal to a small buzz connected to it and warn the driver. Figure 19 below shows a conceptual algorithm of the manual system

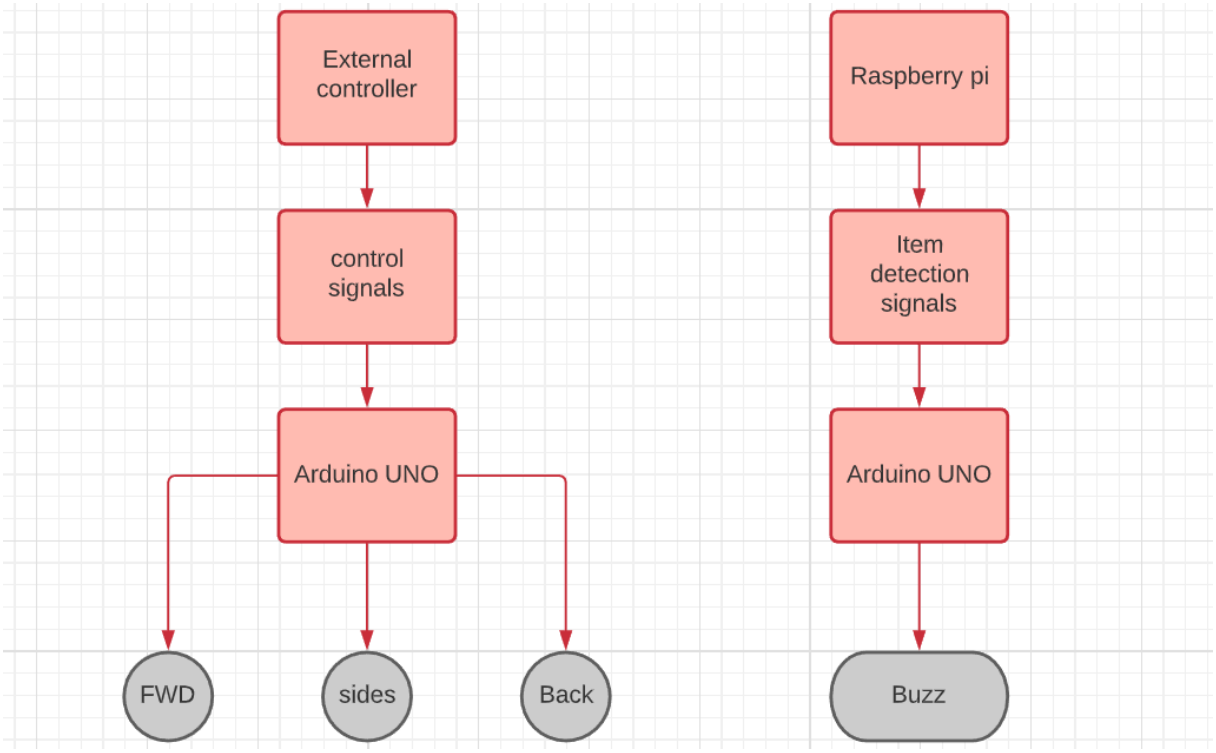

Figure 19 Manual mode concept

- It should be noticed that the codes used for both object detection and control system are provided in appendix below.

Results and analysis of performance test

Arduino testing

In this section, some simulations and testing are shown. In the simulation part, Proteus was used to simulate the whole system; as the figure below shows, L298N motor driver and Arduino UNO alongside the 4 DC motors have been placed. Two motors are connected in parallel for every side, then these two motors were connected to their respective ports in L298N driver, as following: Right1 & Right2 are connected to OUT3 & OUT4 ports in L298N, and Left1 & Left2 are connected to OUT1 & OUT2 ports in L298N. Next, the L298n driver was connected to both power supply and Arduino; connections are as follows: ENA, IN1, IN2, IN3, IN4 & ENB ports of L298N are connected PIN10, PIN9, PIN8, PIN7, PIN6 & PIN5 of Arduino UNO, respectively. The ports ENA, INT1, INT2 are responsible for routing power to the left side DC motors, similarly, ENB, INT3, INT4 are responsible for routing power to the right-side DC motors. Simultaneously, the L298N driver is connected to the power bank, illustrated in Proteus as a DC source. Connections are as follows: 12V port is connected to the positive terminal of the USB cable coming from the power bank, and GND port of L298N is connected to the negative (ground) terminal of the USB cable coming from the power bank. Finally, the Arduino UNO 9V pin was connected to the 5V port of L298N, and the GND pin was connected to the ground, resulting in powering up the Arduino UNO.

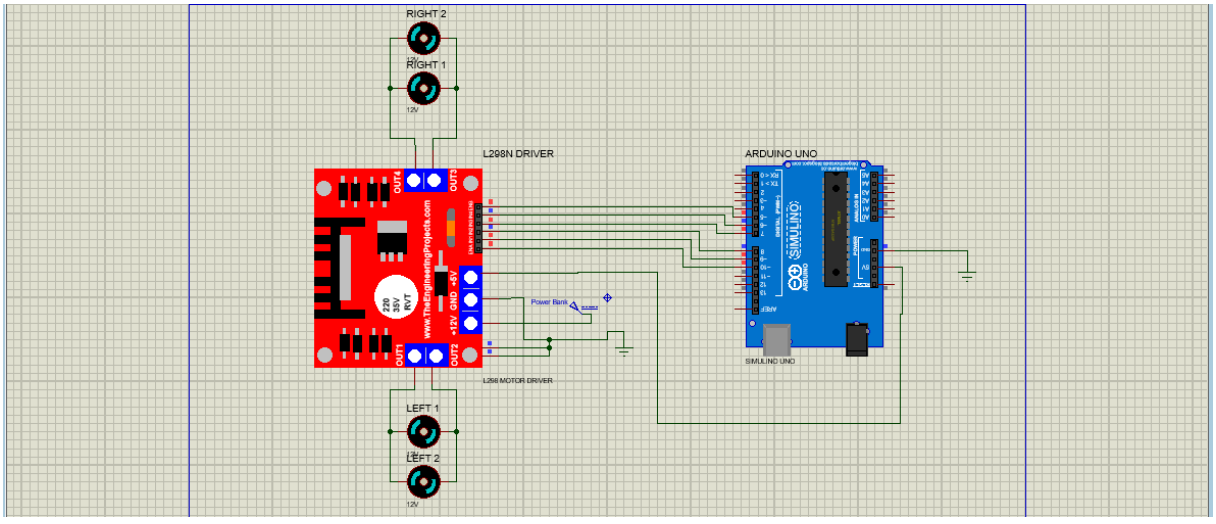

Figure 20 Simulation testing

Testing preparation

After simulating the Arduino and its peripherals, like DC motors and motor drive, it is time now to imitate the real environment by adding some components like the following ones.

- **Test track:** that is 5 meters long with two turns, first is a 30-degree right turn then another 30-degree left turn. The track has two lanes, every lane has a width of 30cm, as shown below.

Figure 21 Test track

- **Traffic Light:** normal traffic light with three light red, yellow, and green. The height of the traffic light is 17.6cm. A picture is shown below:

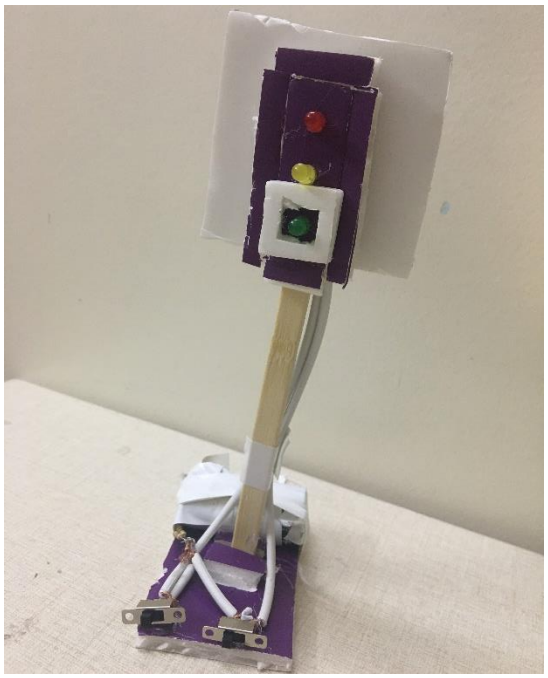

Figure 22 Traffic lights

- **Stop Sign:** normal red stop signs with white text. The height of the sign is 15.3cm. A picture is manifested below:

Figure 23 Stop sign

- **Object (Car):** An object used to be detected to pass it in the test. A picture is manifested below:

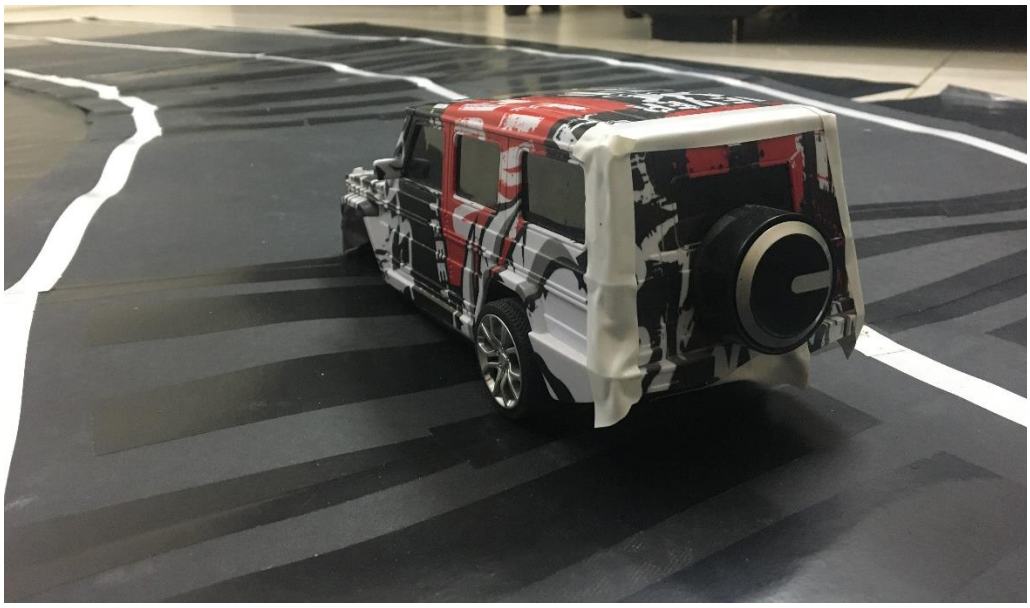

Figure 24 Object car

- **Designed RC Car:** E-visioneers's car that will be tested, which is shown in the figure below:

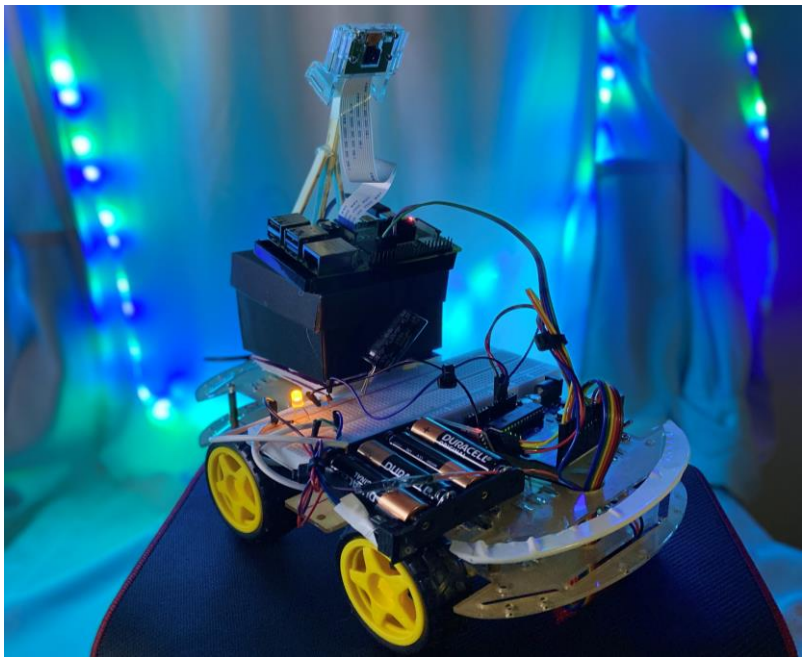

Figure 25 E-visioneers RC car

Lane detection testing

Following the lane detection algorithm mentioned in figure 15 above, the Raspberry Pi module was able to detect the lanes correctly as it can be seen in the figure 21 below. The lane detection alone was able to capture frames at 28 frame per second, which is very close to real time.

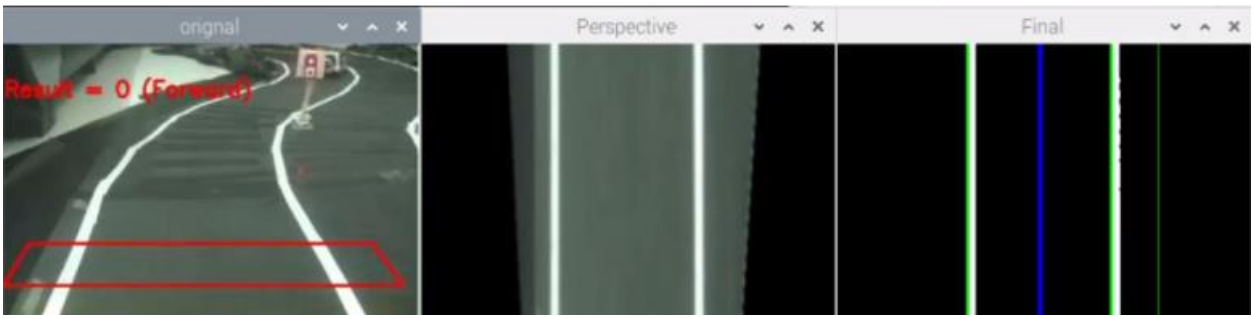

Figure 26 Lane detection test (P1)

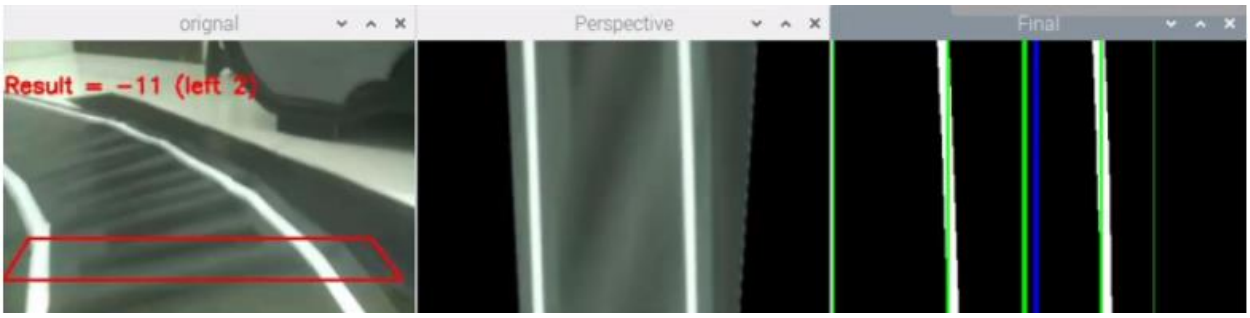

Figure 27 Lane detection test (P2)

from Figure 21 above, the algorithm successfully returned a 0 result when the lanes were straight, this result will be sent to the Arduino to give the order to the motors to move in a straight line. Moreover, Figure 22 shows the case when the road is moving slightly to the left, hence the result returned a negative 11 which again will be sent to UNO to perform the slightly left turn operation.

Object detection testing

Firstly, the supposed plan was to use YOLOv3 tiny to detect objects, but unfortunately, it was very heavy on the raspberry pi as it was generating a frame every 5 seconds, which is not applicable at all. So, the next plan was to train the model using a Cascade Trainer GUI developed by Amin Ahmadi [6], which uses OpenCV to train these models, which generates very solid XML models based on Haar features of the object. Fortunately, due to OpenCV library, integrating these models with the Raspberry Pi code was easy. The obtained result was quite satisfying as These XML models was correctly detecting objects at 18 frame per second. When running object detection algorithms simultaneously with lane detection ones, the raspberry pi was able to detect 10 frame per second, which was still acceptable regarding the application of this project. Upon testing, the camera was able to detect a stop sign from 22 cm as shown in figure 23 below, which is quite acceptable.

Figure 28 Object detection test

Another object could be detected was the small car put on the road for a purpose of testing. The following figure shows a detection of the car upon 31 cm before changing the direction, which is also considered acceptable in the scope of testing.

Figure 29 Object detection test (P2)

After testing the car in the autonomous mode 10 times, the following results have been obtained:

- Detecting stop sign successfully was 8 times which corresponds to 80%
- Traffic Light detection was 7 times which corresponds to 70%
- object detection was 9 times which corresponds to 90%
- Following the track correctly was 6 times, which corresponds to 60%

It can be concluded that the obtained results were good enough to achieve a good performance.

User Manual

Required Software:

1. Arduino Bluetooth RC Car mobile application
2. Computer with VNC viewer remote desktop

Steps to use the car:

1. Press on the switch circled in red illustrated in the figure beneath:

Figure 30 Booting up switch

2. Booting up the computer and Downloading VNC viewer.
3. Connect the Raspberry pi and the computer to the same network.

4. Connect the computer to the raspberry pi using VNC viewer, by typing the IP address of the Raspberry pi as shown in the figure below:

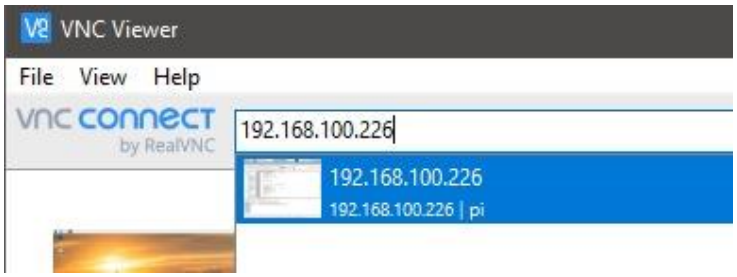

Figure 31: VNC viewer

5. On the screen, Raspberry pi desktop will appear, you need to press on the e-visioneers's folder, as manifested below:

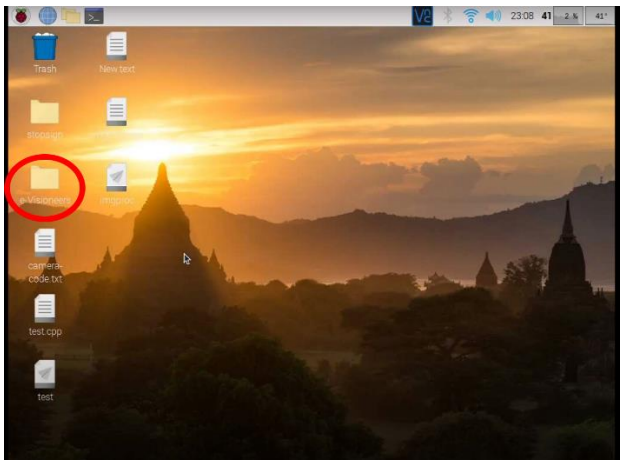

Figure 32: Raspberry Pi's Desktop

6. Then press on the first icon that is called e-visioners as manifested below:

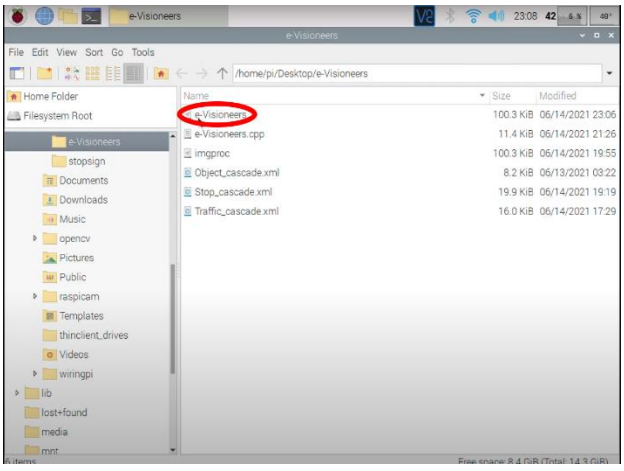

Figure 33: E-visioneers folder

7. A window will pop-up, press on “Execute in Terminal”, as manifested below:

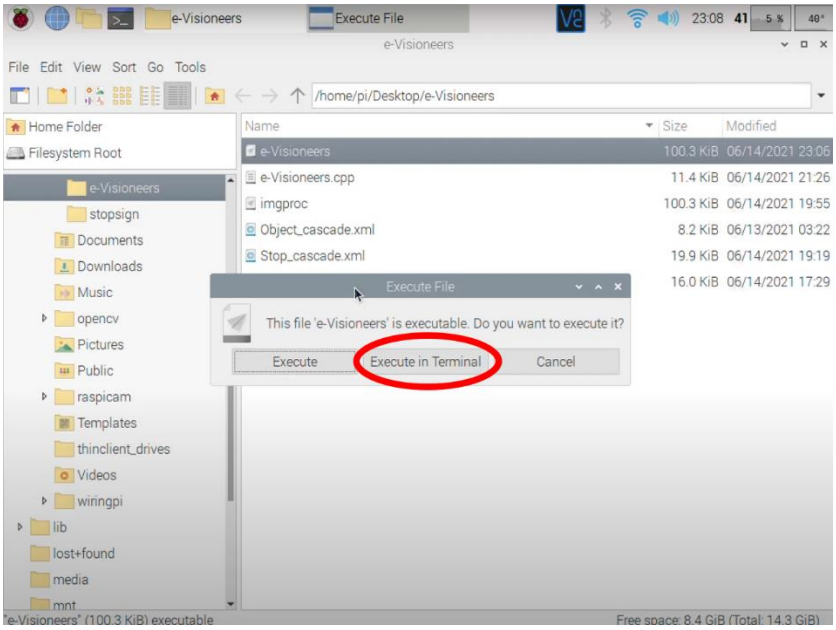

Figure 34: Execute the code

8. Now six different widows will pop up, the three at the top is for lane detection, the bottom three are for traffic light, car, and stop sign detection, respectively, as shown in the figure below:

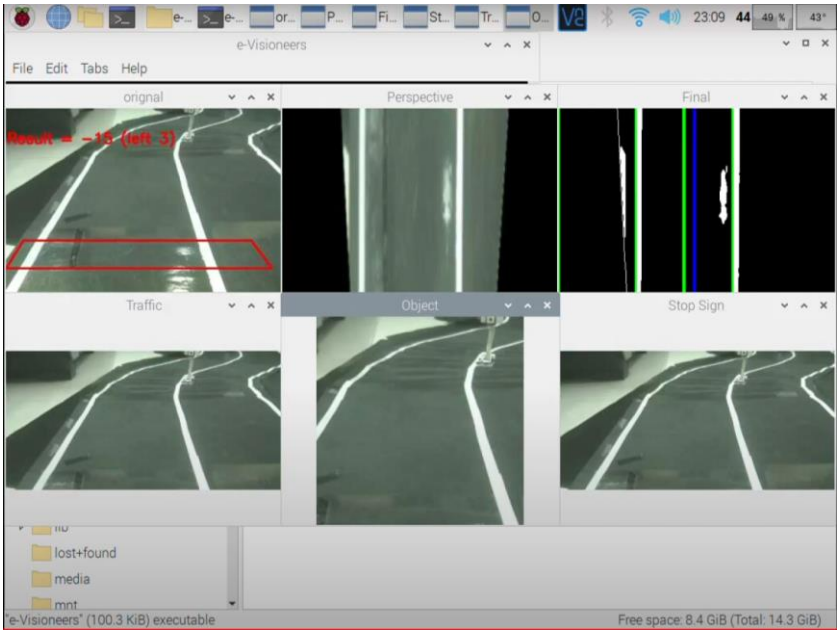

Figure 35: Lane and objects detection views

9. Now, you need to connect the mobile application. Go to Play store on android devices or Apple store on Apple devices, and download Arduino Bluetooth RC controller, the application icon is shown below:

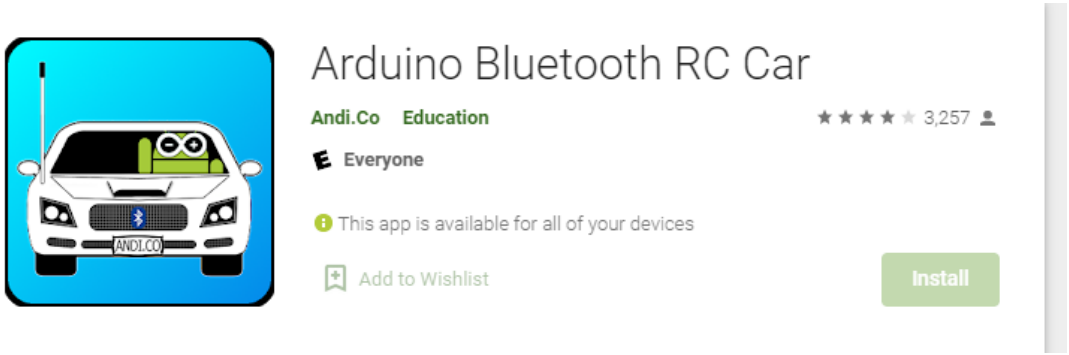

Figure 36; Arduino Bluetooth RC Car Application

10. Open the application and press on the top left circle, as manifested below:

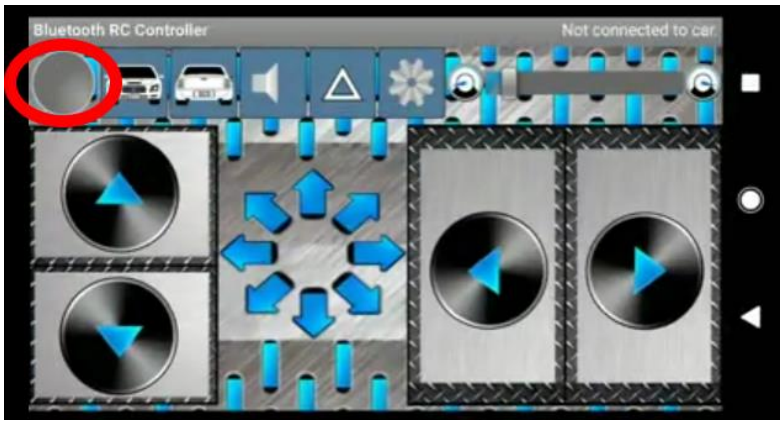

Figure 37: Mobile app Connecting to RC Car – Step 1

11. Press "Connect to car" which is highlighted in the picture beneath:

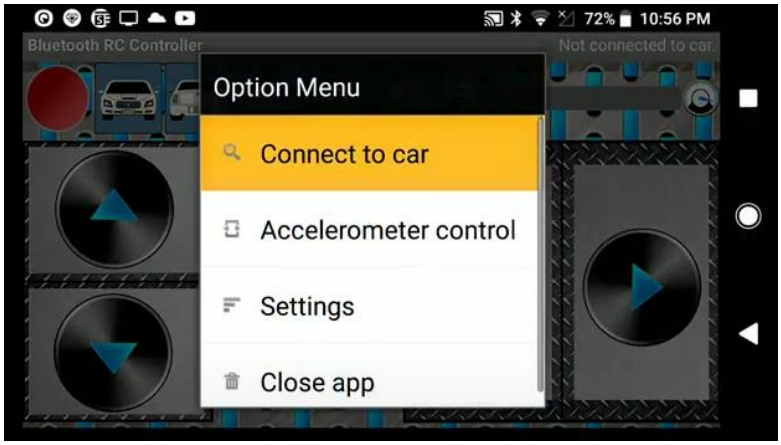

Figure 38: Mobile app Connecting to Car - step 2

12. Press on “Scan for Devices”, then HC-05 will appear. Now press on HC-05 to connect to the car, as shown in the figure below:

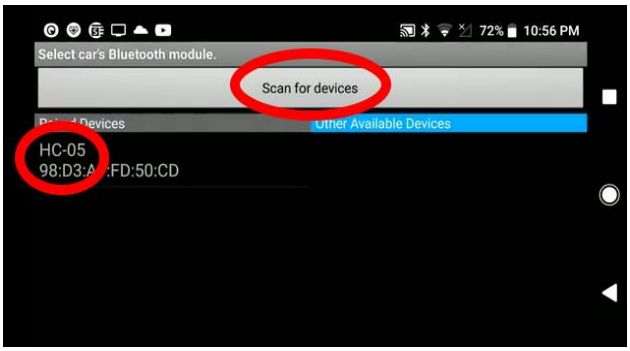

Figure 39: Mobile app Connecting to RC Car - step 3

13. Now the circle mentioned in step 10 will light green indicating that you have successfully connected to the RC car. You can switch between Manual and Autonomous mode by pressing on the two buttons, as illustrated below in the figure:

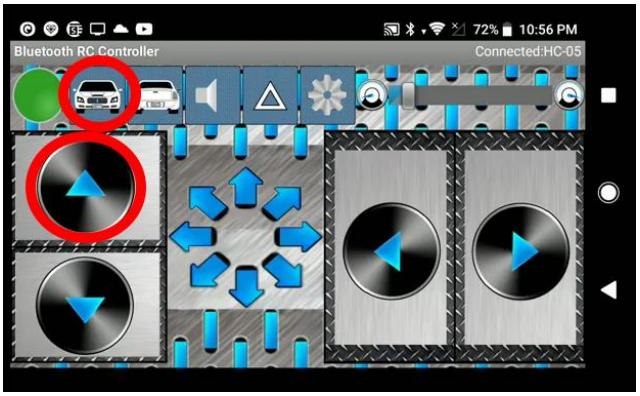

Figure 40: Switching between Manual and Autonomous Modes

When the lights are on in the top button it indicates that the car is in autonomous mode, else it is in manual mode, as manifested below:

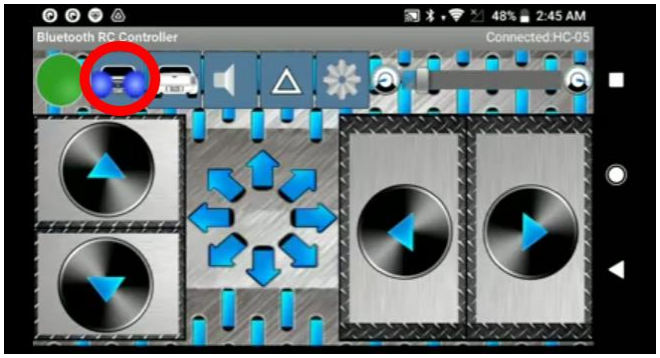

Figure 41: Mode indicator on Mobile app

14. Congratulation now you have successfully connected to the RC car, and you can use it.

Important Notes in Autonomous mode

- When the RC car detects a Traffic light being red it will stop at 40cm distance. When the light turns green it will commence moving after 2 seconds.
- When the RC car detects a car around 30cm away from the object. It will stop for 3 seconds then it will pass the car.
- When the RC detects a stop sign it will fully stop for 3 seconds then it will take a U turn.

Complications and solutions

During the advanced stages of implementing the project, some problems have been faced, which is normal like any other projects. The faced problems can be classified as follows.

Problems regarding the power systems

A) Insufficient power supply:

At the beginning the design had only one power bank as the only power supplier. But due to the presence of 4 DC motors, they were drawing more power than expected. Due to that, the other components such as Raspberry Pi did not get enough power to be boot up.

As a solution, extra 4 AA batteries were added to a battery holder that was connected in series with the power bank. This was enough to make the whole system work for around 35 minutes perfectly.

B) Unavailability of voltage regulator:

To enhance the design stability, we chose to integrate a voltage regulator in the design that was parallel with the Raspberry Pi 3 B+. But the issue was it was not available anywhere. As a solution we chose to connect a capacitor with capacity of 6400 µF with 35v capacity that was connected in parallel to the Raspberry Pi to oppose any sudden change of voltage and enhance the stability of the system.

Problems regarding control system:

One problem happened related to the dimensions of the car; the momentum was pushing the car a few centimeters, which gives inaccurate distance measurements. The solution could be solved by adjusting the distance equations accordingly. Particularly, to let the car stop upon a distance of 20 cm, the distance which was sent to Arduino was 15 in order to compensate the error which was already mentioned. Also, the turning mechanism was needed to be divided into three levels, which gives the car movement more smoothability.

Problems regarding machine vision part:

- The lane detection algorithm was sometimes confused between the lanes and the white floor around it, so a wider frame was detected. Solving this problem was by adding extra black paper after the lanes to cover the white floor.
- The camera was needed to be about 30 cm height. Unfortunately, the cable was just about 13 cm. which forced the team to use a box under the raspberry pi in order to lift the camera to the desired height

Further discussion

Safety issues:

Considering the body of the RC car, the most important concern of the team was to keep the vital part of the car safe and away from any potential damage that can happen during the movement of the car. Hence, the design was aiming to keep the raspberry pi, which is the most important part in the car, away from any possible damage. It can be realized from the top-down system description that the position of every single part was chosen carefully to guarantee the best performance.

Societal impact:

The idea of self-driving cars is growing up sharply these days, which is expected due to their advantages on the individual from one side, as it will have an impact on saving many innocent souls who were passing away due to accidents because of the human indifference or unintentional mistakes. Hence, this project can be a base which will pave the way for more complex and complicated undergraduate and graduate capstone projects in the field of machine learning based objects like cars and drones.

Potential environmental effects:

Considering the electrical cars in general, which started to be widespread among Europe and America, the idea of electrical self-driving car can be a good step in humanity’s struggle against pollution, which is mainly caused by the toxic materials coming out form the traditional cars. Hence, considering this type of projects, can motivate the governments and world organizations to invest more in the field of electric cars which use the clean energy.

Deliverables

The customer can anticipate from this project:

• Task time report illustrating the amount of time was spent on every individual task.

• The software implemented to simulate the project and for any further enhancements.

• Report on all the materials, equipment, and parts used in the project, including their specifications.

• A catalog that manifests how to use the RC car in both controlled and autonomous modes.

• The research report consists of various conducted research that is necessary to upgrade and develop the project.

• The final product, Self-Driving RC Car with a Machine Vision-based Warning System

Total cost

The following table shows the cost of the project including the shipping costs.

Component	Unit(s)	Cost ($)	From
Raspberry pi 3 B+	1	44.99	Amazon
Raspberry pi case + cooling fan	1	9.99	Amazon
Arduino UNO	1	13.98	Amazon
Chassis+4 motors	1	18.99	Amazon
Rpi Camera module	1	23.5	Amazon
L298N Driver	1	6.89	Amazon
10000 mAh Power Bank	1	12	Aziz
HC-05 Bluetooth module	1	7.99	Amazon
Breadboard	3	9.99	Amazon
USB cables	5	8.99	Amazon
Male-female Jumpers	40	8.96	Amazon
Battery Holder	1	6.99	Amazon
Total		173.26+shipping fees	

Conclusion

In the end, it can be stated that this final design report aims to give a full idea about the final product which built in the previous few months. Implementing this project's duration was reasonable and enough to proceed with all the steps, starting from design until testing the product and presenting it in the final version. In the last few pages, each group member's roles have been explained in detail, in addition to their efforts in finding solutions to the problems they have faced. The main aim of the achieved car with its two modes (autonomous and manual) is to decrease the number of accidents every year due to human errors. The achieved car can deliver the solution, which is provided with a machine vision-based warning system and can detect the obstacles in addition to its ability to drive itself. Several steps will be followed, starting with designing different car parts like power systems and control systems and then programming the Raspberry pi according to the peripheral devices connected to it like HD camera. As mentioned before, the camera has detected different types of obstacles by implementing machine vision algorithms. Proudly, the design, with its affordable price, will prove itself as a unique product and have a good impact on future research and development in this field to save more innocent souls from traffic accidents.

References

[1]: Global status report on road safety 2018. World Health Organization (n.d.). Retrieved from https://www.who.int/publications-detail-redirect/9789241565684

[2]: Jin Kang, J., Ahmed, M., & Haskell-Dowland, P. (2020, October 16). Robot take the wheel: Waymo has launched a self-driving taxi service. Retrieved from https://theconversation.com/robot-take-the-wheel-waymo-has-launched-a-self-driving-taxi-service-147908

[3]: Ashton, J. (2019). Autonomous RC Car Platform. Retrieved from: https://web.wpi.edu/Pubs/E-project/Available/E-project-032219-094723/unrestricted/MQP_Final_Paper.pdf

[4]: Donkey Car.(n.d.). Donkey Car. Retrieved from: https://docs.donkeycar.com/

[5]:Kyle. (2020). Autonomous RC Car. Retrieved from: https://www.instructables.com/Autonomous-RC-Car/

[6]: Cascade Trainer GUI - Amin (amin-ahmadi.com)

Appendices

The following few pages show the codes used to build the machine vision algorithms and control system using Raspberry pi and Arduino

```
// need to connect tto arduino then machine learning


#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, frame1, Matrix, framePers, frameGray, frameThresh, frameEdge, frameFinal, frameFinalDuplicate, frameFinal1;
Mat ROILane;
int LeftLanePos, RightLanePos, frameCenter, laneCenter, Result;

int manual_mode=0;  // 1 for manual mode

RaspiCam_Cv Camera;

stringstream ss;

vector<int> histrogramLane;

Point2f Source[] = {Point2f(25,190),Point2f(355,190),Point2f(0,230), Point2f(385,230)};
Point2f Destination[] = {Point2f(100,0),Point2f(300,0),Point2f(100,300), Point2f(300,300)};

//Machine Learning variables
CascadeClassifier Stop_Cascade, Object_Cascade, Traffic_Cascade;
Mat frame_Stop, RoI_Stop, gray_Stop, frame_Object, RoI_Object, gray_Object, frame_Traffic, RoI_Traffic, gray_Traffic;
vector<Rect> Stop, Object, Traffic;
int dist_Stop, dist_Object, dist_Traffic;

 void Setup ( int argc,char **argv, RaspiCam_Cv &Camera ) {
    Camera.set ( CAP_PROP_FRAME_WIDTH,  ( "-w",argc,argv,400 ) );
    Camera.set ( CAP_PROP_FRAME_HEIGHT,  ( "-h",argc,argv,300 ) );
    Camera.set ( CAP_PROP_BRIGHTNESS, ( "-br",argc,argv,60 ) );
    Camera.set ( CAP_PROP_CONTRAST ,( "-co",argc,argv,60 ) );
    Camera.set ( CAP_PROP_SATURATION,  ( "-sa",argc,argv,30 ) );
    Camera.set ( CAP_PROP_GAIN,  ( "-g",argc,argv ,50 ) );
```

```
void Capture() {
        Camera.grab();
    Camera.retrieve( frame);

    cvtColor(frame, frame_Traffic, COLOR_BGR2RGB);
    cvtColor(frame, frame1, COLOR_BGR2RGB);
    cvtColor(frame, frame_Stop, COLOR_BGR2RGB);
    cvtColor(frame, frame_Object, COLOR_BGR2RGB);
    cvtColor(frame, frame,COLOR_BGR2RGB);



}


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);
        line(frame,Source[2], Source[0], Scalar(0,0,255), 2);
        Matrix = getPerspectiveTransform(Source, Destination);
        warpPerspective(frame1, framePers, Matrix, Size(400,300));
        GaussianBlur(framePers, framePers, Size(5, 5), 0);
}

void Threshold()
{
        Rect roi(80,0,230,300);
        Mat mask(framePers.size(), CV_8UC1, Scalar::all(0));
        mask (roi).setTo(Scalar::all(255));
        cvtColor(framePers, frameGray, COLOR_RGB2GRAY);
        inRange(frameGray, 170, 255, frameThresh);
        Canny(frameGray,frameEdge, 300, 200);
        add(frameThresh, frameEdge, frameFinal1);
        cvtColor(frameFinal1, frameFinal1, COLOR_GRAY2RGB);
        frameFinal1.copyTo(frameFinal, mask);
        cvtColor(frameFinal, frameFinalDuplicate, COLOR_RGB2BGR);   //used in histrogram function only

}
```

```
void Histrogram()
{
    histrogramLane.resize(400);
    histrogramLane.clear();

    for(int i=0; i<400; i++)       //frame.size().width = 400
    {
        ROILane = frameFinalDuplicate(Rect(i,140,1,100));
        divide(255, ROILane, ROILane);
        histrogramLane.push_back((int)(sum(ROILane)[0]));
    }
}


void LaneFinder()
{
    vector<int>:: iterator LeftPtr;
    LeftPtr = max_element(histrogramLane.begin(), histrogramLane.begin() + 180);
    LeftLanePos = distance(histrogramLane.begin(), LeftPtr);

    vector<int>:: iterator RightPtr;
    RightPtr = max_element(histrogramLane.begin() +220, histrogramLane.end());
    RightLanePos = distance(histrogramLane.begin(), RightPtr);

    line(frameFinal, Point2f(LeftLanePos, 0), Point2f(LeftLanePos, 300), Scalar(0, 255,0), 2);
    line(frameFinal, Point2f(RightLanePos, 0), Point2f(RightLanePos, 300), Scalar(0,255,0), 2);
}

void LaneCenter()
{
    laneCenter = (RightLanePos-LeftLanePos)/2 +LeftLanePos;
    frameCenter = 196;

    line(frameFinal, Point2f(laneCenter,0), Point2f(laneCenter,300), Scalar(0,255,0), 3);
    line(frameFinal, Point2f(frameCenter,0), Point2f(frameCenter,300), Scalar(255,0,0), 3);
    Result = laneCenter-frameCenter;
}
```

```
void Stop_detection()
{
    if(!Stop_Cascade.load("//home//pi//Desktop//e-Visioneers//Stop_cascade.xml"))
    {
        printf("Unable to open stop cascade file");
    }
    RoI_Stop = frame_Stop(Rect(0,0,400,200));
    cvtColor(RoI_Stop, gray_Stop, COLOR_RGB2GRAY);
    //equalizeHist(gray_Stop, gray_Stop);
    Stop_Cascade.detectMultiScale(gray_Stop, Stop);

    for(int i=0; i<Stop.size(); i++)
    {
        Point P1(Stop[i].x, Stop[i].y);
        Point P2(Stop[i].x + Stop[i].width, Stop[i].y + Stop[i].height);

        rectangle(RoI_Stop, P1, P2, Scalar(0, 0, 255), 2);
        putText(RoI_Stop, "Stop Sign", P1, FONT_HERSHEY_PLAIN, 1,  Scalar(0, 0, 255, 255), 2);

        dist_Stop = (-0.531)*(P2.x-P1.x) + 57.2;
       ss.str(" ");
       ss.clear();
       ss<<"Dis = "<<dist_Stop<<"cm";
       putText(RoI_Stop, ss.str(), Point2f(1,130), 0,0.7, Scalar(20,20,255), 2);

    }
}

void Traffic_detection()
{
    if(!Traffic_Cascade.load("//home//pi//Desktop//e-Visioneers//Traffic_cascade.xml"))
    {
        printf("Unable to open traffic cascade file");
    }

    RoI_Traffic = frame_Traffic(Rect(0,0,400,200));
    cvtColor(RoI_Traffic, gray_Traffic, COLOR_RGB2GRAY);
    //equalizeHist(gray_Traffic, gray_Traffic);
    Traffic_Cascade.detectMultiScale(gray_Traffic, Traffic);
```

```
    for(int i=0; i<Traffic.size(); i++)
    {
        Point P1(Traffic[i].x, Traffic[i].y);
        Point P2(Traffic[i].x + Traffic[i].width, Traffic[i].y + Traffic[i].height);

        rectangle(RoI_Traffic, P1, P2, Scalar(0, 0, 255), 2);
        putText(RoI_Traffic, "Traffic Light", P1, FONT_HERSHEY_PLAIN, 1,  Scalar(0, 0, 255, 255), 2);
        dist_Traffic = (-1.43)*(P2.x-P1.x) + 67.18;

       ss.str(" ");
       ss.clear();
       ss<<"Di = "<<dist_Traffic<<"cm";
       putText(RoI_Traffic, ss.str(), Point2f(1,130), 0,0.7, Scalar(20,20,255), 2);

    }

}

void Object_detection()
{
    if(!Object_Cascade.load("//home//pi//Desktop//e-Visioneers//Object_cascade.xml"))
    {
        printf("Unable to open Object cascade file");
    }

    RoI_Object = frame_Object(Rect(100,0,200,200));
    cvtColor(RoI_Object, gray_Object, COLOR_RGB2GRAY);
    //equalizeHist(gray_Object, gray_Object);
    Object_Cascade.detectMultiScale(gray_Object, Object);

    for(int i=0; i<Object.size(); i++)
    {
        Point P1(Object[i].x, Object[i].y);
        Point P2(Object[i].x + Object[i].width, Object[i].y + Object[i].height);

        rectangle(RoI_Object, P1, P2, Scalar(0, 0, 255), 2);
        putText(RoI_Object, "Object ", P1, FONT_HERSHEY_PLAIN, 1,  Scalar(0, 0, 255, 255), 2);
        dist_Object = (-0.625)*(P2.x-P1.x) + 73.25;

       ss.str(" ");
       ss.clear();
```

```
    }

}

int main(int argc,char **argv)
{
        wiringPiSetup();
    pinMode(21, OUTPUT);
    pinMode(22, OUTPUT);
    pinMode(23, OUTPUT);
    pinMode(24, OUTPUT);
    pinMode(25, OUTPUT);

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

 while(1)
   {
        auto start = std::chrono::system_clock::now();

    Capture();
    Perspective();
    Threshold();
    Histrogram();
    LaneFinder();
    LaneCenter();
    Stop_detection();
    Traffic_detection();
    Object_detection();
```

```
if (dist_Stop > 5 && dist_Stop < 50)
{
    digitalWrite(21, 0);
    digitalWrite(22, 1);    //decimal = 14
    digitalWrite(23, 1);
    digitalWrite(24, 1);
    cout<<"Stop Sign"<<endl;
    dist_Stop = 0;

    goto Stop_Sign;
}

if (dist_Object > 5 && dist_Object < 40)
{
    digitalWrite(21, 1);
    digitalWrite(22, 0);    //decimal = 13
    digitalWrite(23, 1);
    digitalWrite(24, 1);
    cout<<"Object"<<endl;
    dist_Object = 0;

    goto Object;
}

if (dist_Traffic > 5 && dist_Traffic < 50)
{
    digitalWrite(21, 1);
    digitalWrite(22, 0);    //decimal = 9
    digitalWrite(23, 0);
    digitalWrite(24, 1);
    cout<<"Red Traffic"<<endl;
    dist_Traffic = 0;

    goto Traffic;
}
```

```
if ( Result ==0 )
{
    digitalWrite(21, 1);
    digitalWrite(22, 1);    //decimal = 1
    digitalWrite(23, 1);
    digitalWrite(24, 1);
    cout<<"Forward"<<endl;
    ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (Forward)";
   putText(frame, ss.str(), Point2f(1,50), 0,0.7, Scalar(0,0,255), 2);
}

else if (Result >0 && Result <7)
{
    digitalWrite(21, 0);
    digitalWrite(22, 1);    //decimal = 10
    digitalWrite(23, 0);
    digitalWrite(24, 1);
    cout<<"Right1"<<endl;
            ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (right 1)";
   putText(frame, ss.str(), Point2f(1,50), 0,0.7, Scalar(0,0,255), 2);
}

        else if (Result >=7 && Result <15)
{
    digitalWrite(21, 1);
    digitalWrite(22, 1);    //decimal = 3
    digitalWrite(23, 0);
    digitalWrite(24, 0);
    cout<<"Right2"<<endl;
            ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (right 2)";
   putText(frame, ss.str(), Point2f(1,50), 0,0.7, Scalar(0,0,255), 2);

}
```

```
    else if (Result < 80 && Result >=15)
{
    digitalWrite(21, 0);
    digitalWrite(22, 0);    //decimal = 12
    digitalWrite(23, 1);
    digitalWrite(24, 1);
    cout<<"Right3"<<endl;
            ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (right 3)";
   putText(frame, ss.str(), Point2f(1,50), 0,0.7, Scalar(0,0,255), 2);
}

    else if (Result <0 && Result >-7)
{
    digitalWrite(21, 1);
    digitalWrite(22, 0);    //decimal = 5
    digitalWrite(23, 1);
    digitalWrite(24, 0);
    cout<<"Left1"<<endl;
       ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (left 1)";
   putText(frame, ss.str(), Point2f(1,50), 0,0.7, Scalar(0,0,255), 2);
}

   else if (Result <=-7 && Result >-15)
{
    digitalWrite(21, 0);
    digitalWrite(22, 1);    //decimal = 6
    digitalWrite(23, 1);
    digitalWrite(24, 0);
    cout<<"Left2"<<endl;
       ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (left 2)";
   putText(frame, ss.str(), Point2f(1,50), 0,0.7, Scalar(0,0,255), 2);
}
```

```
else if ( Result >-80 && Result <=-15)
{
    digitalWrite(21, 1);
    digitalWrite(22, 1);    //decimal = 7
    digitalWrite(23, 1);
    digitalWrite(24, 0);
    cout<<"Left3"<<endl;
       ss.str(" ");
   ss.clear();
   ss<<"Result = "<<Result<<" (left 3)";
   putText(frame, ss.str(), Point2f(1,50), 0,0.7, Scalar(0,0,255), 2);
}

Stop_Sign:
Traffic:
Object:



namedWindow("orignal", WINDOW_KEEPRATIO);
moveWindow("orignal", 0, 100);
resizeWindow("orignal", 340, 255);
imshow("orignal", frame);

namedWindow("Perspective", WINDOW_KEEPRATIO);
moveWindow("Perspective", 340, 100);
resizeWindow("Perspective", 340, 255);
imshow("Perspective", framePers);

namedWindow("Final", WINDOW_KEEPRATIO);
moveWindow("Final", 680, 100);
resizeWindow("Final", 340, 255);
imshow("Final", frameFinal);

namedWindow("Stop Sign", WINDOW_KEEPRATIO);
moveWindow("Stop Sign", 680, 355);
resizeWindow("Stop Sign", 340, 255);
imshow("Stop Sign", RoI_Stop);
```

```
    namedWindow("Perspective", WINDOW_KEEPRATIO);
    moveWindow("Perspective", 340, 100);
    resizeWindow("Perspective", 340, 255);
    imshow("Perspective", framePers);

    namedWindow("Final", WINDOW_KEEPRATIO);
    moveWindow("Final", 680, 100);
    resizeWindow("Final", 340, 255);
    imshow("Final", frameFinal);

    namedWindow("Stop Sign", WINDOW_KEEPRATIO);
    moveWindow("Stop Sign", 680, 355);
    resizeWindow("Stop Sign", 340, 255);
    imshow("Stop Sign", RoI_Stop);

        namedWindow("Traffic", WINDOW_KEEPRATIO);
    moveWindow("Traffic", 0, 355);
    resizeWindow("Traffic", 340, 255);
    imshow("Traffic", RoI_Traffic);

    namedWindow("Object", WINDOW_KEEPRATIO);
    moveWindow("Object", 340, 355);
    resizeWindow("Object", 340, 255);
    imshow("Object", RoI_Object);


    waitKey(1);
    auto end = std::chrono::system_clock::now();
    std::chrono::duration<double> elapsed_seconds = end-start;

    float t = elapsed_seconds.count();
    int FPS = 1/t;
    cout<<"FPS = "<<FPS<<endl;

   }

    return 0;


}
```