TecnoCampus
Escola Superior Politècnica

Centres universitaris adscrits a la
upf. Universitat Pompeu Fabra Barcelona

**Electrical & Automation Engineering**

# INTELLIGENT ROBOT

## Memory

**ANDRÉS GARCÍA SEVILLA**

**THESIS ADVISOR: MARCOS FAUNDEZ**

SPRING 2022

TecnoCampus
Mataró-Maresme

To my parents, brother, and friends thanks to whom I became the person I’m today.

# Abstract

The aim of this Final Thesis Project is to shed some light on the utility of implementing user-friendly robot assistants in the everyday world, accessible to the population, in terms of cost and simplicity of use.

This study examines and explores the capabilities of building a helping robot with easily accessible hardware and applying complex software concepts, such as computer vision and machine learning while maintaining a low development budget during production.

With the goal in mind of developing an assistant robot skilled to follow several paths, such as drawn lines or a person, we created an object-recognition system connected to some motors that allows the robot to perform its function accordingly. The programmed code devised encompasses a library which enables the robot to recognize a wide range of different objects, used as specific input, which will trigger the motors so that the robot carries out alternative movements in line with the entries. Such movements are also coded, with Python programming language.

The project also attempts to find the perfect union between these complex software and electronic hardware, as well as with the creation of a resilient aluminum robot's chassis, capable of withstanding the carriage of considerable weight on it, thus helping the user to carry items along the way.

This design might be profitable today, especially as machine learning software development is flourishing nowadays.

# Index

# Figure index

# Index of Tables

# Glossary of terms

| | |
|---|---|
| API | AdaFruit Pi |
| CAD | Computer Aided Design |
| COCO | Common Objects in Context |
| CPU | Central Processing Unit |
| DC | Direct Current |
| GND | Ground |
| GPIO | General Purpose Input Output |
| HD | High Definition |
| L298N | Type of motor module driver |
| MDF | Medium Density Fiberboard |
| ML | Machine Learning |
| OD | Object Detection |
| Open CV | Object recognition software |
| OS | Operating System |
| PCB | Printed Circuit Board |
| PWM | Pulse Width Modulation |
| RPI | Raspberry Pi |
| SBC | Single Board Computer |
| USB | Universal Serial Bus |
| Vcc | Common Collector Voltage |

# 1. Objective

## 1.1 Purpose

The primary motivation behind the project is based on the initiative to facilitate the use of robotic assistants in businesses and private homes without the need for prior knowledge in programming or complex robotics operations.

The desired purpose would be composed of a combination of simplicity and elegance in a way that delivers the final product and tries to adhere to the initial idea explained in the following point.

## 1.2 Object and Scope

This project aims to attain a fusion between theoretical knowledge and empirical elaboration in the technical field. In this way, it is expected to obtain modern knowledge of Electronic Engineering combined with the teaching of state-of-the-art software.

The central focus of attention of the current Final Thesis Project, through the union between software and electronic hardware, and together with the knowledge learned during teaching, is intended to develop a robotic model capable of facilitating the life of the person who is in possession of it.

To this end, the project turns to the commercial field of small and medium-sized companies, aiming to implement a series of robots that can precisely perform the required work without needing any additional programming. That is thanks to an easy-to-follow guide provided to the user that contains a series of instructing signals, which the user himself can direct to the robots.

Another desired key application is the capability of the robot to follow a person and communicate while carrying considerable weight on its back, which would be perfectly suitable for the elderly when shopping.

The documentation concerning the development of the work will follow the guidelines and regulations dictated by the Pompeu Fabra University of Barcelona and Tecnocampus ESUPT for the Final Degree Projects.

The estimated time of dedication for the correct completion of the project consists of around 500/ 600 hours. The margin shown represents the possible improvement of the final product in case of premature completion.

The extensive programming knowledge required for the project implementation will be acquired from libraries and web pages, as indicated in the bibliography.

Once all the assembly processes are completed, the robot shall be able to identify symbols through one of the cameras and send signals to the actuators to perform their corresponding tasks.

# 2. Background and need for information

## 2.1 Background

This project starts from work already done in Python with image processing, such as the line follower robot in Open CV. But it broadens it so that it is handy for the user through the changes highlighted in the previous section.

The main problem facing today's society lies in the speed at which all software systems advance on par with their hardware counterpart. That causes the need for constant learning by technicians responsible for complex instruments.

However, the day-to-day citizen does not have certain bases to keep up with improvements in technology. For this reason, the involvement of the engineer in simplifying consumer products for everyday use is of the utmost importance.

In the field of civil aid robotics, nowadays this simplification is not quite well implemented since it usually requires personal manufacture for each client or the constant supervision of technical teams responsible for the robots' well-functioning. A clear example of that is in the mega warehouses where Amazon must manage the inventory of products. This case is the ultimate foundation to optimize the time and expenses of the company. However, not everyone has access to the number of logistics and means involved in applying this modern solution.

The project is based on this last point and aims to improve public access to the use of the utility robot.

## 2.2 Necessary Information

This section reveals the most important features to consider when developing the proposed project.

### 2.2.1 Practical function of the design

The robot will be able to move within its environment thanks to the usage of 4 wheels attached to the vehicle chassis. It is mainly designed for flat terrains, without much roughness, to achieve peak performance at the early stages of the prototype.

The early project prototype will interact with its surroundings objects by having a surface capable of carrying a load on its back.

It will be powered by an internal battery attached to the chassis. The optimal battery will power 12V in parallel to the motor controllers and another battery to power the RPI with 5.1V and a serial USB connection.

The brain of the robot will be stored in the Raspberry Pi, a British made single-board computer, which is excellent for robotics development due to its cost-effective price and effective build.

The robot will be able to detect itself in the world, thanks to implementing the world coordinates into the SBC. This will be enabled through the usage of cameras to perceive the environment, remaining the main point of software programming. The goal would be achieving the robot to be aware of its surroundings in all sorts of situations by using image processing and machine learning.

### 2.2.2 Appearance role in the design's function

The robot's appearance will not be the focal point of the project, even though it will be preferable to achieve an aesthetic and completed look.
For it, the preferable constituent materials would be aluminum -for the prototype chassis- and the wheels, a 3D printed design. The screws and gears will be conventional metal. All the materials will need to be prepared to undergo the thermal properties of their surroundings and the internal circuitry.
It is of utmost importance to manage to get a stable and rigid design. For that instance, the robot body will be mostly quadricorn, resembling a day-to-day car.

### 2.2.3 Materials suitable for the design

The properties of the material will determine its suitability for a design. For the current project with robotics, aluminum is the preferred choice for building the chassis and basic iron metal is used for the mechanical connections shown in the prototyping section.

In the electronics department materials such as tin for the soldering of some parts is a necessity apart from the classic PCB boards and cables included.

On the other hand, the wheels are an important factor in the prototyping phase and a 3D flexible printing material is used for the rotatory parts while a more robust printing material forms the outer part of the object.

### 2.2.4 Appropriate construction methods for the design

The most appropriate methods to build the desired result are the cutting and shaping with the assembly of parts using the needed screws, glues, solder etc. The molding and the casting for the 3D is also very important moving forward.

A particular material can only be worked in a limited number of ways. The method of construction will therefore be determined by the chosen material, the availability of manufacturing facilities and the production costs.

### 2.2.5 Key Technik Factors

The programming of the robot's SBC will be made in the Python programming language dueto its extended reach into the desired results of the project.

#### 2.2.5.1 Open CV

Open CV is a free to use python library that implements a range of functions directed to improve the communication between video inputs and image procession methods using Artificial intelligence and machine learning. For this project this library will be crucial to make a cross-

bridge between the real world and the brain of the robot. This will be explained further in the prototyping section.

#### 2.2.5.2 Programming Structure

The initial problem is quite simple to display; it is wanted that the robot follows a determined path or object, and when it encounters signals, it must be able to recognize them and act accordingly to the user's guide.

The initial idea of the programming was to use signals with numbers inside the object to interpret it by the robot and thus making it do actions depending on the signal. After some testing this was found to be very basic, and the final program takes it to another level. The program not only is able to recognize objects/signals and assign them to actions, but it is made in a way that is incredibly easy for the developer to change such assignments to objects in a manner that complements the user wishes.

With this code embedded into the robot, it can interpret each signal in the programming phase by implementing machine learning techniques and ensuring that the robot knows how to differentiate between the different applications

The final available routines of the robot will be the possibility of entering following mode in which the robot follows the desired object or entering line following routine mode in which will follow a determined path and act accordingly to the signals.

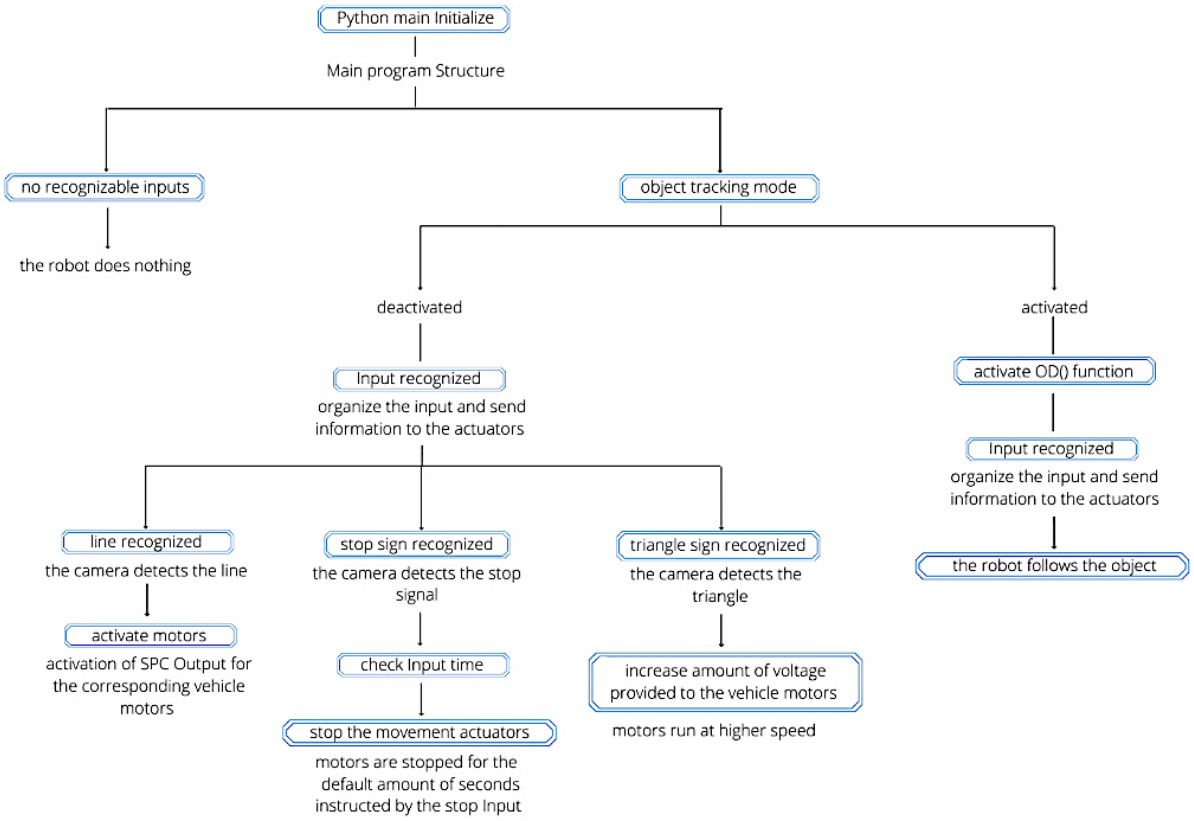



Figure 2.1 Flowchart of the robot's utilities

Figure 2.1 represents a prototype of the very first utilities of the robot.

### 2.2.5.3 SBC

To make the program-robot connection, a Raspberry Pi 4 board (RPI4) will be used, which guarantees a certain number of ports that can be very useful when connecting it with the Inputs and Outputs of the system. Due to the unavailability of the first proposed board in the preliminary concepts, the Raspberry Pi 4 was the chosen answer for the brain of the project.

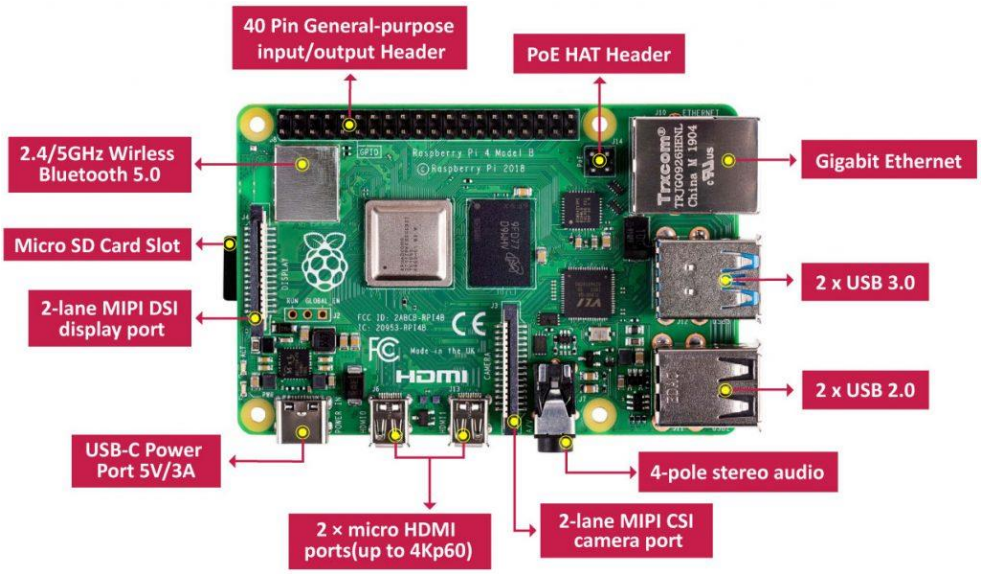



Figure 2.2 Raspberry Pi 4

With the RPI4, a lot of new mechanics are introduced to the original concept of the robot, such as the wireless control via Bluetooth or Wi-Fi with a virtual machine connected to the board. Due to budget limits and available stock reasons, the used model is the 2Gb. Although using the 4Gb model is highly recommended for better performance in these types of image recognition, this limited performance is a problem in the robot cognitive overall skill.

Most of the programming process is developed outside the Raspberry board and implemented once it works properly. That is due to the different OS used in both the laptop and the board. Raspberry uses Raspbian, which is a Debian-based operating system for Raspberry Pi. Since 2013, it has officially been provided by the Raspberry Pi Foundation as the primary operating system for the Raspberry Pi family of compact single-board computers.

Further information is provided in the prototyping section.

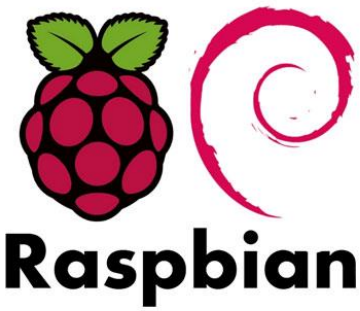



Figure 2.3 Raspbian

#### 2.2.5.4 Cameras

The selection of a camera capable of capturing a good video quality and frames remains crucial to the current project. That is obviously due to the dependence on object-detection of the robot for doing actions, depending on the input.

In this case, considering the budget and the processing power of the Raspberry board used, it was selected the Logitech C270 Webcam HD of 720p/30fps. It has a good balance between quality, price, and requirements to run effectively.


Figure 2.4 Logi 720 CAM

### 2.2.5.5 Robot Chassis and Mechanical Parts

The chassis will be made of aluminum bars to achieve a simple squared structure that will provide the robot with low-weight robust integrity towards the environment.

DC motors will be implemented to control the robot's angular and linear movement. These will be positioned at the chassis, the upper chamber.

To power the initial prototype, the usage of a 12V battery will be necessary and a 5V battery for the RPI itself.

More explanation can be found in the prototype phase.

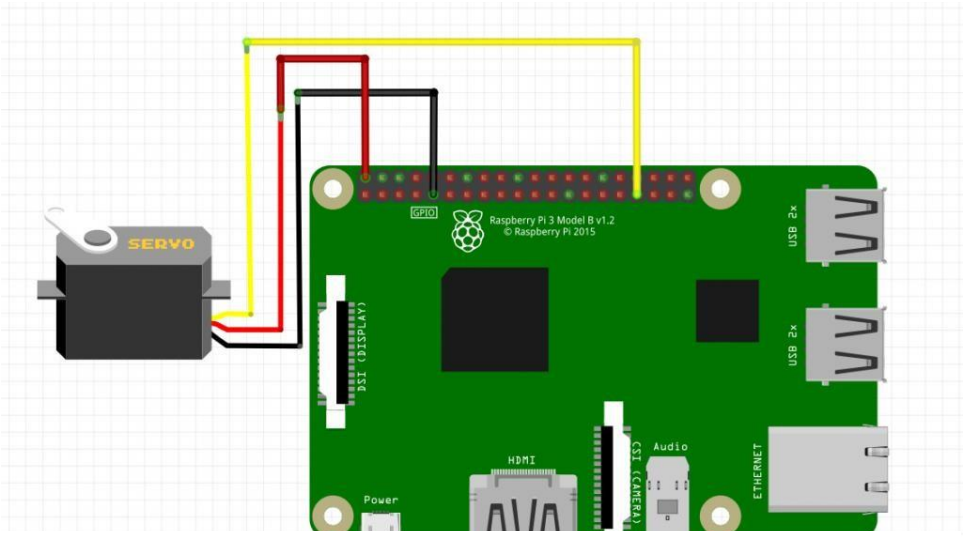



Figure 2.5  Motor-RPI4 Connection

### 2.2.6 Development Structure & Methodology

The main goal of the project is to achieve a robot capable of following instructions and of following an individual. These are the main objectives, and the final utilities of the robot will grow from those roots.

The first steps are based on the development of an order tracking system by image recognition. Thanks to this image recognition system, the robot will be able to identify different video inputs and translate them into actions. The underlying mechanism of this input and output translation is based on Machine Learning, previously programmed in Python programming language with several libraries and boards, such as the Raspberry Pi. This key point will be referred to as Operation Brain. The resources used for the correct fulfillment of the operation will only be the Python software and the camera to test image inputs and outputs.

Once the approval has been given to the Operation Brain, the base program will be connected to the SBC, which will transfer the information to the camera and the robot's mechanical transmission system. This process is called Operation Implant, and it will be in constant upgrade during the building process.

Parallel to this last point, the mechanical-electronic development of the motor actuators and chassis of the automaton, which we will name Operation Body, is launched. On the other hand, the materials used to fabricate the chassis are tested and molded into different prototypes until reaching an optimal design.

Finally, it should be said that the insertion of space and temperature sensors to perfect and humanize the product is attractive. We will call this point as Operation Senses.
After considering all the variables and resources, the methodology used will be the following:
The methodology will be heavily influenced by shipping times of the components and the current chip shortage in the world.

To monitor the project the software Monday Project Management will be used and will help with the deadlines’ completion.

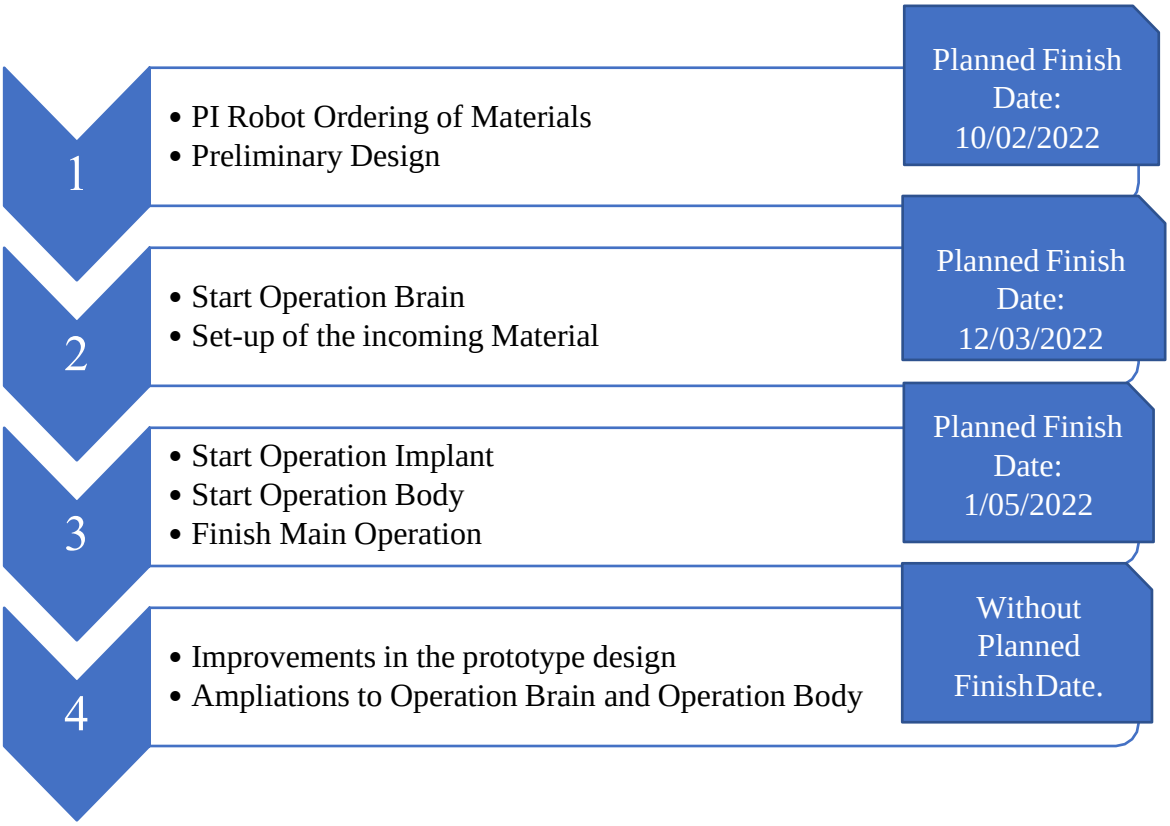



Figure 2.6 Project Milestones

#### 2.2.6.1 Initial Risks

The implementation of the software into the hardware can bring debugging problems. For this reason, the implementation process requires most of the development phase time and will be solved with all the necessary material.

Furthermore, the mechanical work can affect the initial budget by the misplacement or damage of parts. A margin of error has been applied to the initial costs to solve this hypothetical problem.

# 3. Prototyping

## 3.1 Chassis

For the desired chassis in the prototype, aluminum bars were used to make a base able to support the considerable weight of the components plus the load and final weight.

Figure 3.1 Aluminum Bars dismounted

The construction of the chassis is composed of 2 long bars of 500mm and 3 shorter bars of 250 mm that make the rectangular form more robust.
There was a numerous reason on why Aluminum was chosen to build the initial prototype in the first place:

The aluminum bars allow you to fit different accessories that are made to fit the T-Track shape of the sides. On the other hand, is very aerodynamic while being light and robust than steel bars and all of this by being arguably better looking from a esthetically point of view.

.

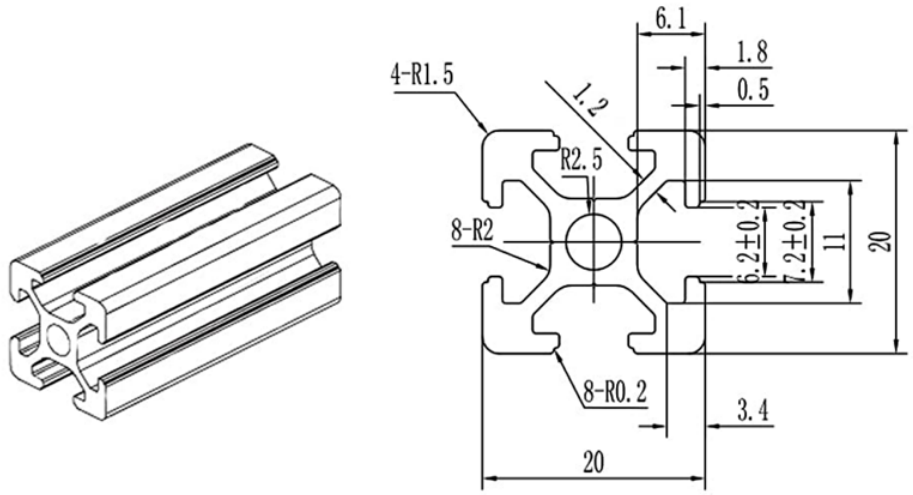



Figure 3.2 Aluminum Bars measures

| Name | Surface (mm) | Length (mm) | Units | Import (€) |
|---|---|---|---|---|
| Aluminum bars | 40x40 | 500 | 2 | 24,00 |
| Aluminum bars | 40x40 | 250 | 3 | 20,00 |
| Edge suport | 39x39 | 35 | 10 | 16,50 |
| | | **TOTAL** | | **60,50** |

Table 1 Informative Basic Budget

To put the aluminium bars together, 8 bar connections were made using iron connectors that were attached to the edges as support.

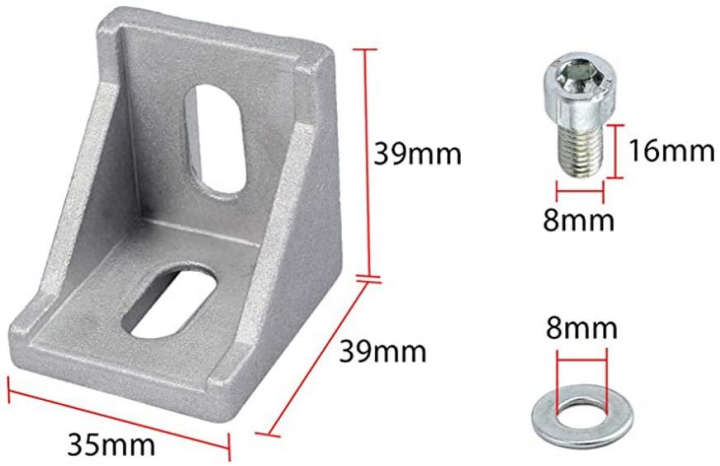



Figure 3.3 Connectors measures

Figure 3.4 Basic Prototype Chassis

The result is a well-rounded and robust chassis that weights around 7kg and with enough space and surface to place the required components.

## 3.2 Wheels

One of the interesting properties of the robot are the characteristics of the wheels, it will use the Mechanum wheel, sometimes called the Swedish wheel or Ilon wheel after its inventor, Bengt Erland IIon [1923-2008] who designed and patented them in 1972. These wheels are tireless

and have a series of rubberized external rollers obliquely attached to the rim with each of them attacked to a powertrain which generates a propelling force perpendicular to the roller axle that can be vectored into a normal longitudinal or transverse movement independently of the current vehicle position.

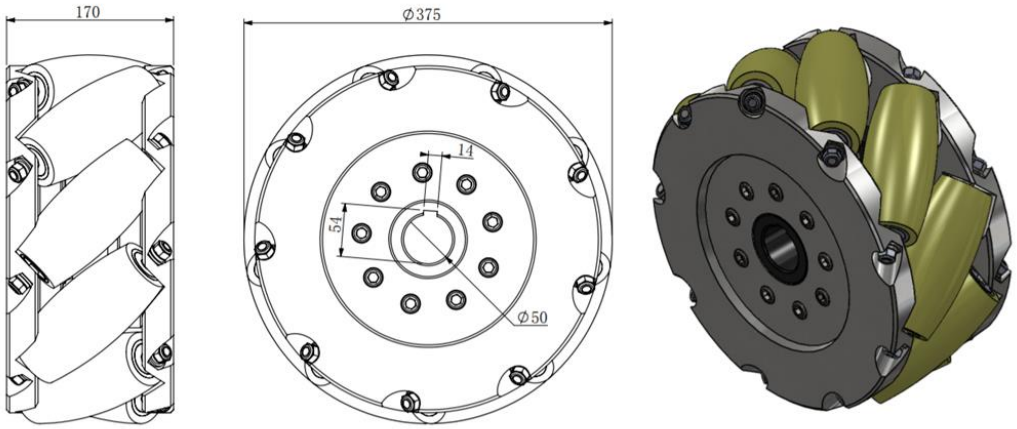



Figure 3.5 Wheel prototype

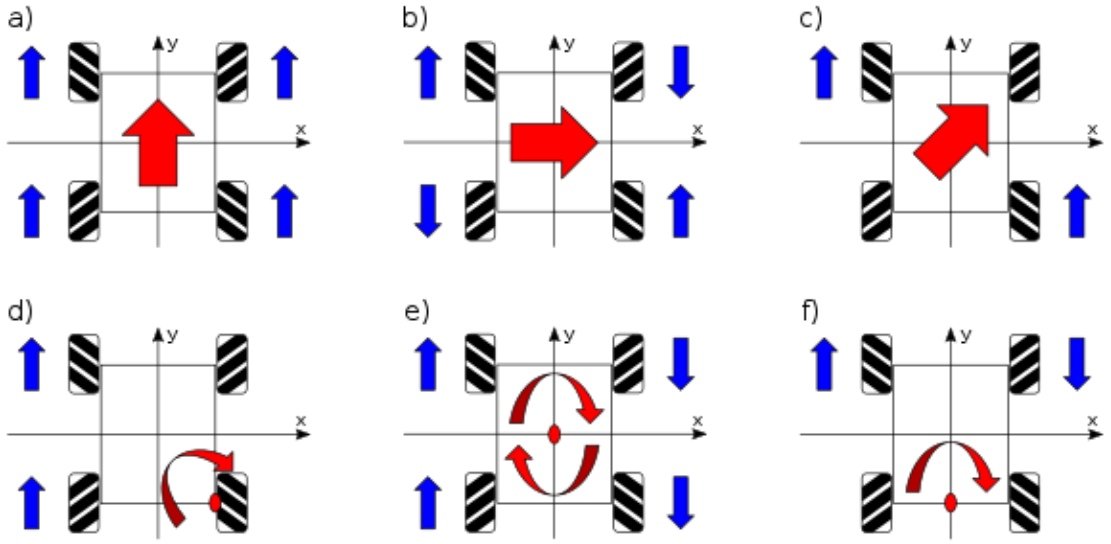



Figure 3.6 Four wheels motion

High prices regarding the wheels resulted in the decision to design and implement a personal design inspired by some CAD and SolidWorks schemes available for open source online.

After some editing in Solid-Works to adjust the diameter of the wheel to a more universal connection for the future motor these are the results:

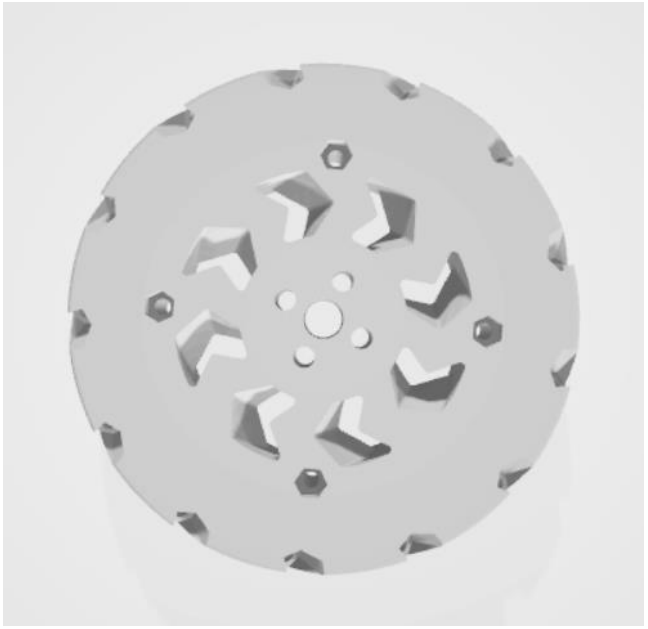


Figure 3.7 Outer surface Mecanum Wheel

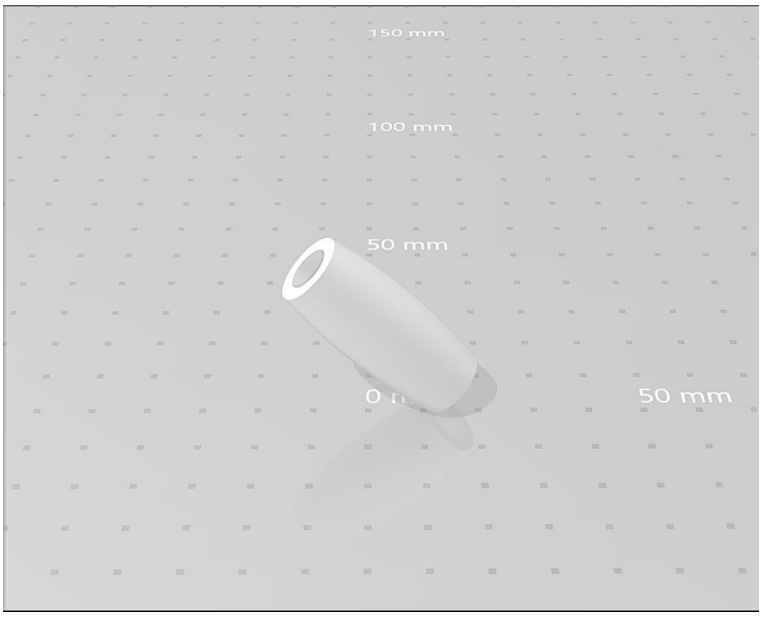



Figure 3.8 Mecanum Wheel Roller

Figure 3.9 Solid Works Full 3D Wheel

Figure 3.10 Printed Wheels (3D)

These results are quite good in the 3D printing department. It could be considered that the 3D printed wheels may not be as good as the ones in the market regarding quality and durability.

The programming of the software directed to control the wheels will depend on the image detection inputs, and it will follow the combinations of movement dictated by the Mecanum wheels. This way, a good omnidirectional movement will be implemented.

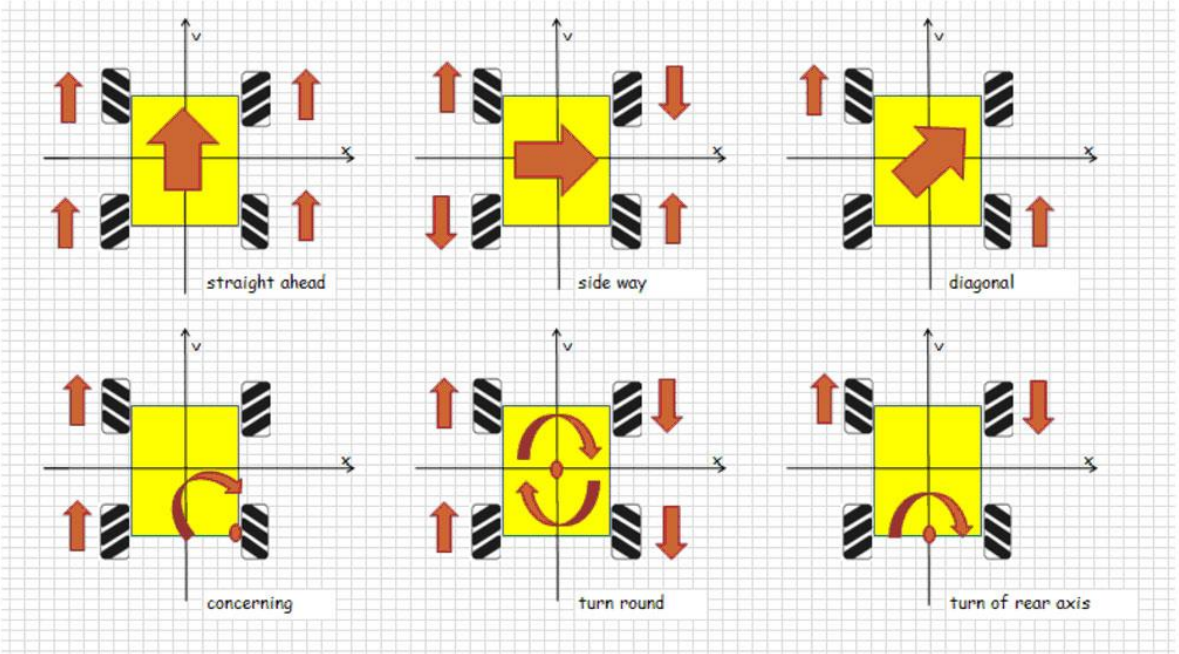



Figure 3.11 Movement formulas

## 3.3 Motors

Something very interesting happened with the motors. It was decided that a DC motor with a big torque (like the DC motors of electric chairs) and low rpm would be ideal for the whole robot because speed is not a necessity in this case.

After ordering the 4 DC motors selected after some research, a delay in the delivery of the items occurred. War in Ukraine happened, and these motors were coming from a company in St. Petersburg. This caused the order to be cancelled after 3 weeks of delay. These shocking events caused a tremendous impediment to progress in the motor-software development, and it was necessary to redirect efforts into researching more available and optimal motors. In summary, sanctions to Russia delayed the development of the thesis. However, a good motor for the prototype was found, a 7.2V DC motor was implemented for the initial robot and put into the respective four wheels while being connected to the controller.

After some testing, it was decided that the selected motor worked correctly even thou a lower voltage motor such as the 6V ones were also found to be optimal for the design. The final prototype can carry both kind of motors with the only downside of changing some code to adjust to the PWM used.

## 3.4 Software Object Recognition

For the initial test of the first code for the robot a total recognition of 91 different objects was achieved thanks to the training of a machine learning model.

Always taking into consideration the capacity of the board, it wasn't optimal to work with high demanding object detection software, so the solution was to work with raw python programming and Open CV, a python library with functions that makes the work with camera detection easier.

All the code used is original work by the creator of the thesis, which integrates different previous codes regarding object detection. The code will be uploaded to a personal GitHub, and available for open use.

Here it is seen various examples of the working code. The purpose of these identifications is to take actions depending on the identification of these images.

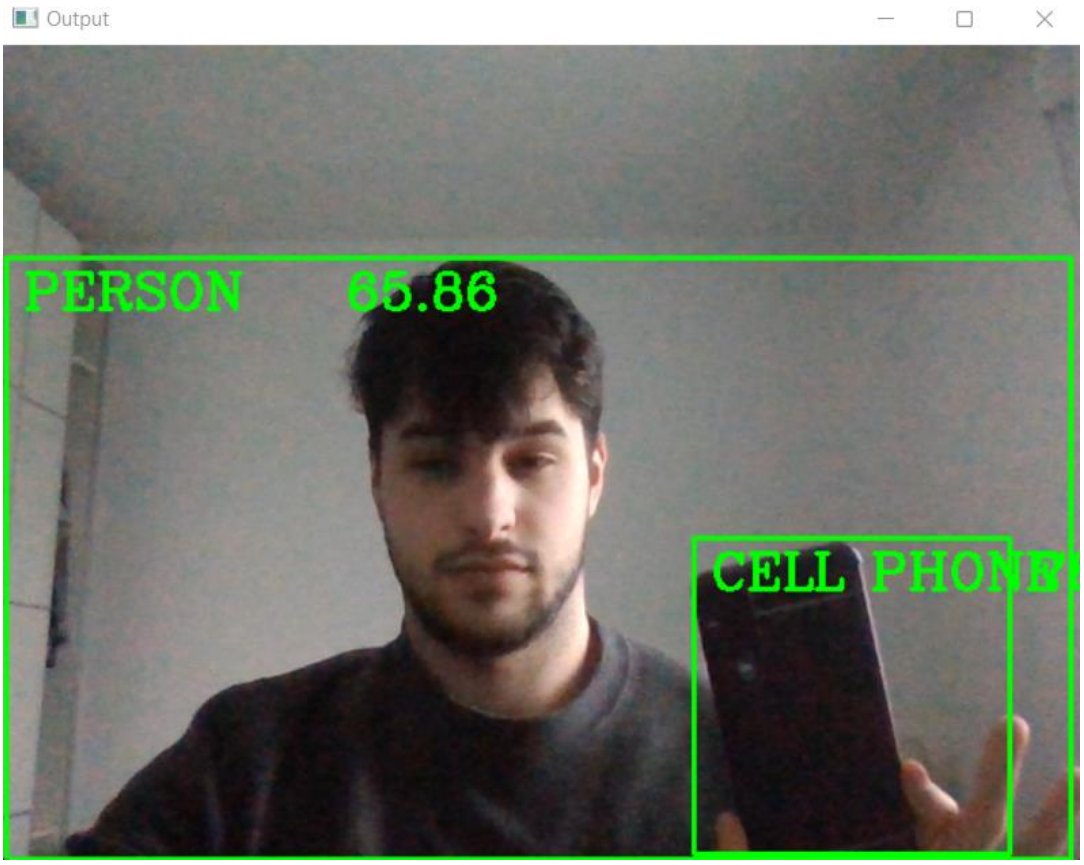



Figure 3.12 Viewable code output

The number that features on Figure 3.12 corresponds to the probability at which the model is sure of what it is assigning the label. In this case, the robot is 65.86% certain that the object recognized is a person. This recognition can be upgraded by increasing the percentage required to label the input image. However, bearing in mind available budget and resources, it is best not to have an extremely accurate device.

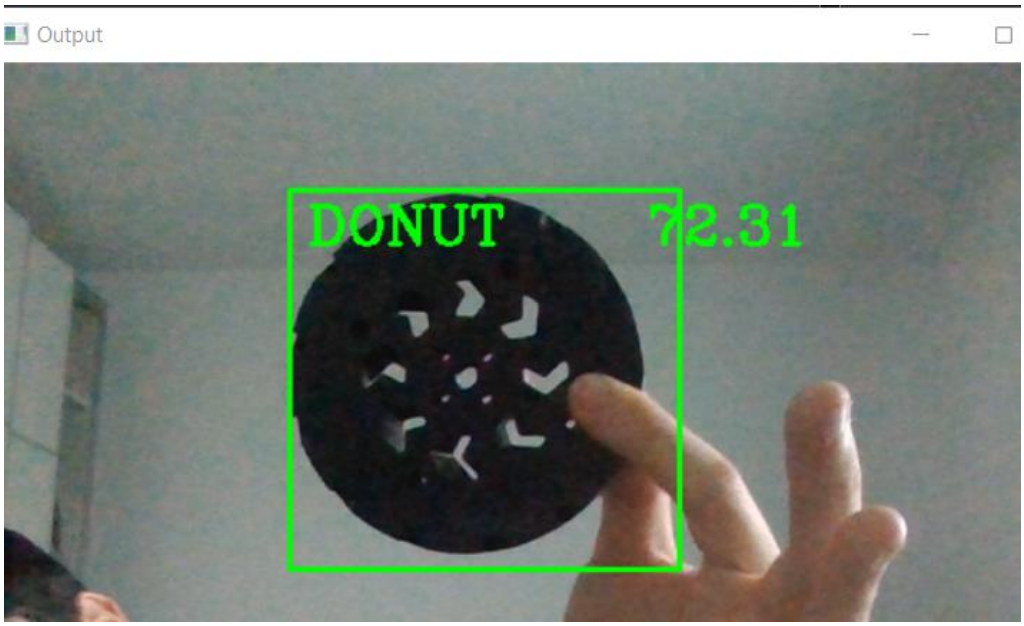



Figure 3.13 Error Example nº1

As seen in Figure 3.13, the level of certainty of the model when labeling is not very high.

For an improved execution of the recognition model, the dataset COCO (Common Objects in Context) was implemented. It was created to advance image recognition by containing challenging, high-quality datasets for computer vision.

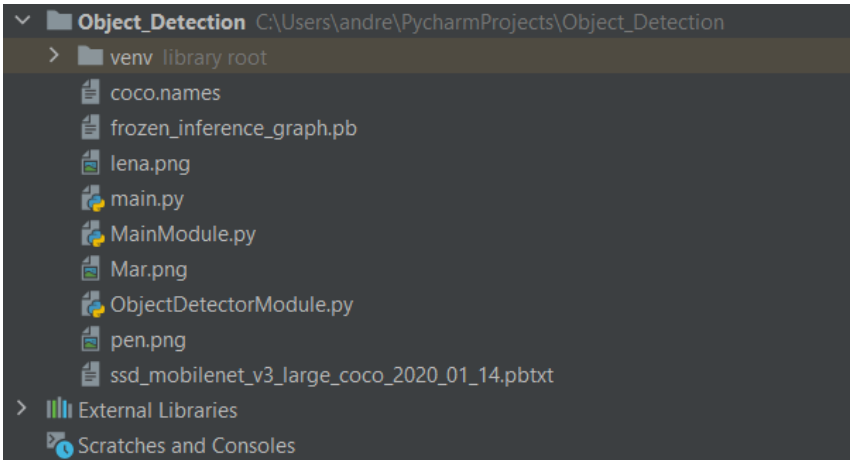



Figure 3.14 Object Detection Folder

It works with pre-defined points in a set of coordinates (17 in this case) which will appoint a result depending on the position of said points.

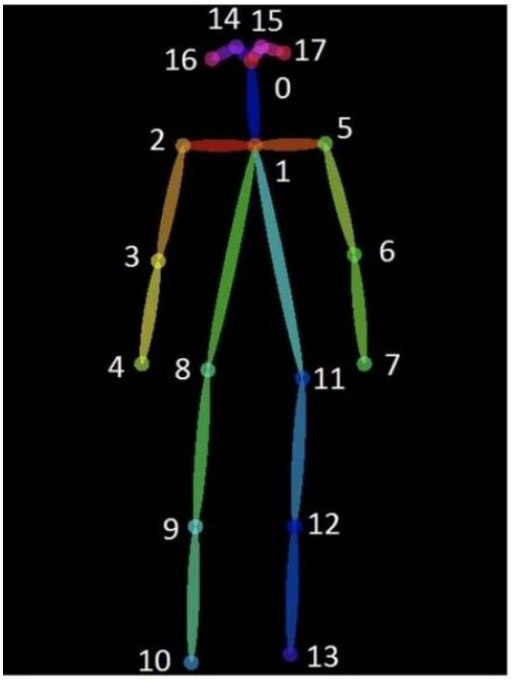



Figure 3.15 Set of coordinates for object "Person"

This model will be improved with the implementation of TensorFlow, a software capable of training visual detection models. It will be taken into consideration that the COCO dataset is trained with hundreds of thousands of images already, so the basic objects will be easier to recognize.

The code developed is made into a module to simplify the primary recognition function applied to the main program of the robot.

The only problem with this code is the difficulty of processing that involves. The problem with the performance of the controller is currently being solved.

```python
import cv2


classNames= []
classFile = "coco.names"
with open(classFile,"rt") as f:
    classNames = f.read().rstrip('\n').split('\n')

configPath = "ssd_mobilenet_v3_large_coco_2020_01_14.pbtxt"
weightsPath = "frozen_inference_graph.pb"

net = cv2.dnn_DetectionModel(weightsPath,configPath)
net.setInputSize(320,320)
net.setInputScale(1.0/ 127.5)
net.setInputMean((127.5, 127.5, 127.5))
net.setInputSwapRB(True)
def getObjects(img,thres,nms,draw=True,objects=[]):

    classIds, confs, bbox = net.detect(img,confThreshold=thres,nmsThreshold=nms)
    #print(classIds,bbox)
    if len(objects) == 0:
        objects = classNames
    objectInfo = []
    if len(classIds) != 0:
        for classId, confidence,box in zip(classIds.flatten(),confs.flatten(),bbox):
            className = classNames[classId - 1]
            if className in objects:
                objectInfo.append([box, className])
                if (draw):
                    cv2.rectangle(img,box,color=(0,255,0),thickness=2)
                    cv2.putText(img,className.upper(),(box[0]+10,box[1]+30),
                    cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
                    cv2.putText(img,str(round(confidence*100,2)),(box[0]+200,box[1]+30),
                    cv2.FONT_HERSHEY_COMPLEX,1,(0,255,0),2)
    return img,objectInfo
if __name__ == "__main__":
    cap = cv2.VideoCapture(0)
    cap.set(3, 1280)
    cap.set(4, 720)
    # cap.set(10,70)
    while True:
        success, img = cap.read()
        result,objectInfo = getObjects(img,0.5,0.2,objects=["bottle"])
        #print(objectInfo)
        cv2.imshow("Output",img)
        cv2.waitKey(1)
```

Figure 3.16 First unsuccessful prototyping code

After some testing of the code, this first resolution to the problem was found to be not optimal, when connected to the Raspberry Pi 4 it could not run properly and it carried a considerable amount of latency which made the actuators misplace the orders.

To solve this massive setback a new code had to be implemented with the help of the TensorFlow software to adapt better to the capabilities of the Raspberry. This resulted in a great optimized python script that can keep up with the hardware characteristics.

TensorFlow is an end-to-end open-source platform for machine learning. It has a comprehensive, flexible ecosystem of tools, libraries and community resources that lets researchers push the state-of-the-art in ML and developers easily build and deploy ML powered applications.

The final code for the object recognition is shown below with the example of the object following option enabled and connected to the motors plus all the commentary explaining the line-to-line functionality.

```
#Andres Garcia Sevilla
#Tecnocampus 2022, July
#TFG --> Following Object and Actuator Robot

#########################################################################################################################
#import of all important modules
import os
import argparse
import cv2
import numpy as np
import sys
import time
from threading import Thread
import importlib.util
import RPi.GPIO as GPIO
from time import sleep
from Test_Motor_2 import *
```

Figure 3.17 Modules imported in the code

The OD() function is the responsible for the enabling of the object following routine.

```
def OD():

# Set-up of the camera and resolution configuration (resolution is not a variable used so a change would mean changing the software as well)
# Also set-up of the cmd command script and the camera capture of all the environment
# Set-up of the pathing of the script to combine functions with the Motors scripts
# Set-up of the pathing for the Deep Learning model and fetching of the AI

###############################################################################################################
    class VideoStream:

        def __init__(self,resolution=(640,480),framerate=30):
            # Initialize the PiCamera and the camera image stream
            self.stream = cv2.VideoCapture(0)
            ret = self.stream.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc(*'MJPG'))
            ret = self.stream.set(3,resolution[0])
            ret = self.stream.set(4,resolution[1])

            # Read first frame from the stream
            (self.grabbed, self.frame) = self.stream.read()

        # Variable to control when the camera is stopped
            self.stopped = False

        def start(self):
        # Start the thread that reads frames from the video stream
            Thread(target=self.update,args=()).start()
            return self

        def update(self):
            # Keep looping indefinitely until the thread is stopped
            while True:
                # If the camera is stopped, stop the thread
                if self.stopped:
                    # Close camera resources
                    self.stream.release()
                    return

                # Otherwise, grab the next frame from the stream
                (self.grabbed, self.frame) = self.stream.read()

        def read(self):
        # Return the most recent frame
            return self.frame

        def stop(self):
        # Indicate that the camera and thread should be stopped
            self.stopped = True
```

Figure 3.18 OD() function for robot's following routine

The following arguments are added to accelerate the calling of the function from the RPI4 terminal:

```
# Define and parse input arguments
parser = argparse.ArgumentParser()
parser.add_argument('--modeldir', help='Folder the .tflite file is located in',
                    default='/home/pi/Desktop/Main_TFG/Sample_TFLite_model')
parser.add_argument('--graph', help='Name of the .tflite file, if different than detect.tflite',
                    default='detect.tflite')
parser.add_argument('--labels', help='Name of the labelmap file, if different than labelmap.txt',
                    default='labelmap.txt')
parser.add_argument('--threshold', help='Minimum confidence threshold for displaying detected objects',
                    default=0.4)
parser.add_argument('--resolution', help='Desired webcam resolution in WxH. If the webcam does not support the resolution entered, errors may occur.',
                    default='720x480')
parser.add_argument('--edgetpu', help='Use Coral Edge TPU Accelerator to speed up detection',
                    action='store_true')

args = parser.parse_args()

MODEL_NAME = args.modeldir
GRAPH_NAME = args.graph
LABELMAP_NAME = args.labels
min_conf_threshold = float(args.threshold)
resW, resH = args.resolution.split('x')
imW, imH = int(resW), int(resH)
use_TPU = args.edgetpu
```

Figure 3.19 Addition of terminal arguments

After the Function basic configuration, the model is called to be processed by the TensorFlow procedures:

```
# Import TensorFlow libraries
# If tflite_runtime is installed, import interpreter from tflite_runtime, else import from regular tensorflow
# If using Coral Edge TPU, import the load_delegate library
pkg = importlib.util.find_spec('tflite_runtime')
if pkg:
    from tflite_runtime.interpreter import Interpreter
    if use_TPU:
        from tflite_runtime.interpreter import load_delegate
else:
    from tensorflow.lite.python.interpreter import Interpreter
    if use_TPU:
        from tensorflow.lite.python.interpreter import load_delegate

# If using Edge TPU, assign filename for Edge TPU model
if use_TPU:
    # If user has specified the name of the .tflite file, use that name, otherwise use default 'edgetpu.tflite'
    if (GRAPH_NAME == 'detect.tflite'):
        GRAPH_NAME = 'edgetpu.tflite'

# Get path to current working directory
CWD_PATH = os.getcwd()

# Path to .tflite file, which contains the model that is used for object detection
PATH_TO_CKPT = os.path.join(CWD_PATH,MODEL_NAME,GRAPH_NAME)

# Path to label map file
PATH_TO_LABELS = os.path.join(CWD_PATH,MODEL_NAME,LABELMAP_NAME)
```

Figure 3.20 Loading the tensorflow model

The first name of the testing list used by the machine learning algorithm appeared as an error named “???” so it had to be manually removed:

```
# Load the label map
with open(PATH_TO_LABELS, 'r') as f:
    labels = [line.strip() for line in f.readlines()]

# Have to do a weird fix for label map if using the COCO "starter model" from
# https://www.tensorflow.org/lite/models/object_detection/overview
# First label is '???', which has to be removed.
if labels[0] == '???':
    del(labels[0])
```

Figure 3.21 Cleaning of errors in the model list

Now the imported model must be imported into the program and take out the details of each recognized objects in the selected frame, in this case, the camera of the robot.

```
##################################################################################################################
# Deep Learning phase of the code, here we take the Lite model that was created via TensorFlow software and
# import it into the program. I also added the possibility to add a google TPU that improves the performance but
# due to budget capacity it was left out, in case of adding this hardware the program would recognize it and
# greatly imporve the speed of recognition.

    # Load the Tensorflow Lite model.
    # If using Edge TPU, use special load_delegate argument
    if use_TPU:
        interpreter = Interpreter(model_path=PATH_TO_CKPT,
                                  experimental_delegates=[load_delegate('libedgetpu.so.1.0')])
        print(PATH_TO_CKPT)
    else:
        interpreter = Interpreter(model_path=PATH_TO_CKPT)

    interpreter.allocate_tensors()

    # Get model details
    input_details = interpreter.get_input_details()
    output_details = interpreter.get_output_details()
    height = input_details[0]['shape'][1]
    width = input_details[0]['shape'][2]

    floating_model = (input_details[0]['dtype'] == np.float32)

    input_mean = 127.5
    input_std = 127.5

# Check output layer name to determine if this model was created with TF2 or TF1,
# because outputs are ordered differently for TF2 and TF1 models
outname = output_details[0]['name']

if ('StatefulPartitionedCall' in outname): # This is a TF2 model
    boxes_idx, classes_idx, scores_idx = 1, 3, 0
else: # This is a TF1 model
    boxes_idx, classes_idx, scores_idx = 0, 1, 2

# Initialize frame rate calculation
frame_rate_calc = 1
freq = cv2.getTickFrequency()

# Initialize video stream
videostream = VideoStream(resolution=(imW,imH),framerate=30).start()
time.sleep(1)

#for frame1 in camera.capture_continuous(rawCapture, format="bgr",use_video_port=True):
```

Figure 3.22 Setup of video

Once the setup is finished, the actual code comes into play by defining the current frame and putting the input data coming from the TensorFlow to work:

```
##################################################################################################################
# After adding the DL model here we capture the video coming from the port 0 of the raspberry in which it is connected
# the Logitech camera and it captures and recognizes the objects around the environment. It is a constant loop of capturing
# in which the most important part of the code logic resides.

    while True:

        # Start timer (for calculating frame rate)
        t1 = cv2.getTickCount()

        # Grab frame from video stream
        frame1 = videostream.read()

        # Acquire frame and resize to expected shape [1xHxWx3]
        frame = frame1.copy()
        frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
        frame_resized = cv2.resize(frame_rgb, (width, height))
        input_data = np.expand_dims(frame_resized, axis=0)

        # Normalize pixel values if using a floating model (i.e. if model is non-quantized)
        if floating_model:
            input_data = (np.float32(input_data) - input_mean) / input_std

        # Perform the actual detection by running the model with the image as input
        interpreter.set_tensor(input_details[0]['index'],input_data)
        interpreter.invoke()

        # Retrieve detection results
        boxes = interpreter.get_tensor(output_details[boxes_idx]['index'])[0] # Bounding box coordinates of detected objects
        classes = interpreter.get_tensor(output_details[classes_idx]['index'])[0] # Class index of detected objects
```

Figure 3.23 Processing of information given by the module

An important factor for the detection of the objects position (4.5.4 explanation) is needed, in here a tolerance is selected with the defined coordinates of the center of the camera:

```
scores = interpreter.get_tensor(output_details[scores_idx]['index'])[0] # Confidence of detected objects


#Find the centroid and create a tolerance rectangle
#tolerance is used to make the middle rectangle bigger for the robot to consider the centering of the object optimal
tolerance = 40
(h, w) = frame.shape[:2] #w:image-width and h:image-height
cv2.circle(frame, (w//2, h//2), 3, (255, 255, 255), -1)
#cv2.circle(frame, ((w+150)//2, (h)//2), 3, (255, 255, 255), -1)
cv2.rectangle(frame, ((w//2)-tolerance, (h//2)+tolerance), ((w//2)+tolerance, (h//2)-tolerance), (10,255,0),2)
```

Figure 3.24 Tolerance adjustment of the camera in relation to the object

In this example the selected item to follow is a simple water bottle, this can be adjusted to the desired one like a person or signal. Instead of writing boxes around every recognizable object that the model gives, the code only will draw the required centroids and rectangles around the object that is wanted to follow thus making a more optimal use of the CPU.

```
# Loop over all detections and draw detection box if confidence is above minimum threshold
# Here we select the object we want to follow
for i in range(len(scores)):
    if ((scores[i] > min_conf_threshold) and (scores[i] <= 1.0)):
        object_name = labels[int(classes[i])]
        if object_name == "bottle":
            # Get bounding box coordinates and draw box
            # Interpreter can return coordinates that are outside of image dimensions, need to force them to be within image using max() and min()
            ymin = int(max(1,(boxes[i][0] * imH)))
            xmin = int(max(1,(boxes[i][1] * imW)))
            ymax = int(min(imH,(boxes[i][2] * imH)))
            xmax = int(min(imW,(boxes[i][3] * imW)))

            #draw the rectange
            cv2.rectangle(frame, (xmin,ymin), (xmax,ymax), (10, 255, 0), 2)
            #draw the centroid of the rectange
            cv2.circle(frame, ((xmin+xmax)//2,(ymin+ymax)//2), 3 , (255,255,255), -1)

            centroid_x =  (xmin+xmax)//2
            centroid_y =  (ymin+ymax)//2
            print(xmax,xmin)
            print(w//2)
```

Figure 3.25 Detection and drawing around the side object (eg. bottle)

For the last important part of this example the camera-actuators connection is shown below, by importing the motor script module developed with functions. (The motor module software is shown in the 3.5.2 point). Also, it follows the 3.5.4-point principles and applies it to the real life.

```
#STOP turning and move forward
if (xmax > (w//2)) and (xmin < (w//2)):

    if centroid_y < (h//2) - tolerance:
        print ("forward")
        run()

    elif centroid_y > (h//2) + tolerance:
        print("backward")
        backward()

    else:
        print("stop")
        stop()

#Turn right condition
elif centroid_x > ((w//2)):
    print("turn right")

#Turn left Condition
elif centroid_x < ((w//2)):
    print("turn left")
```

Figure 3.26 Correction of movement with kinematic functions

When running the code, the bottle is set to the right of the centroid of the camera which makes the robot send the information to the motors telling them to turn right and adjust the position. The numbers shown on the shell are the centroid position of the recognized object relative to the camera coordinates.

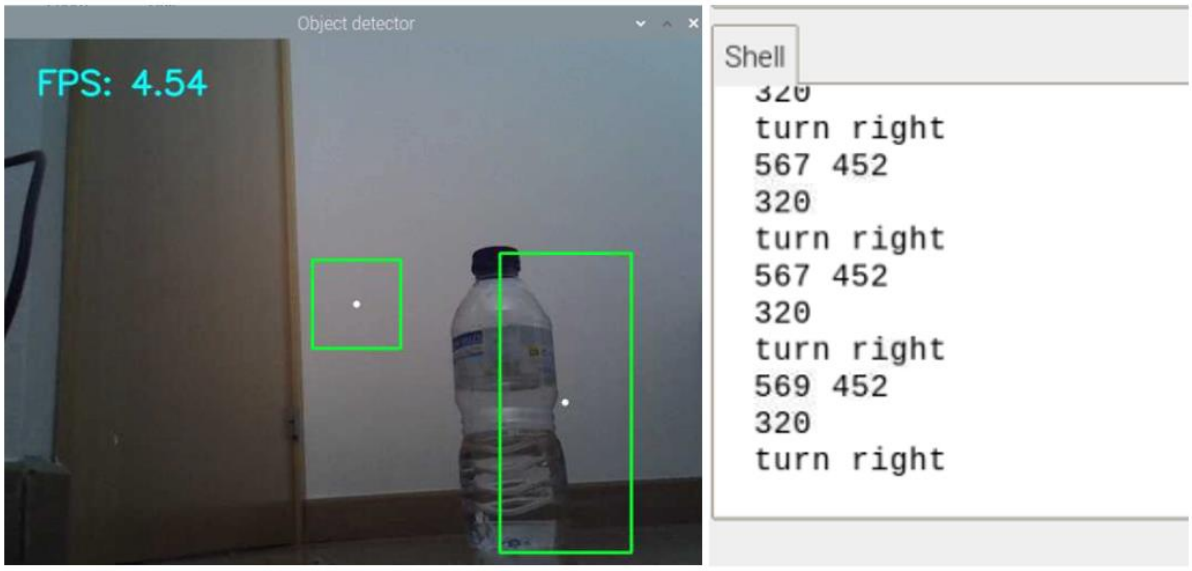



Figure 3.27 Object recognized moved to the right

If the bottle is set to the left, the actuators are told to turn left to fix the centroid of the camera to the centroid of the recognized object.

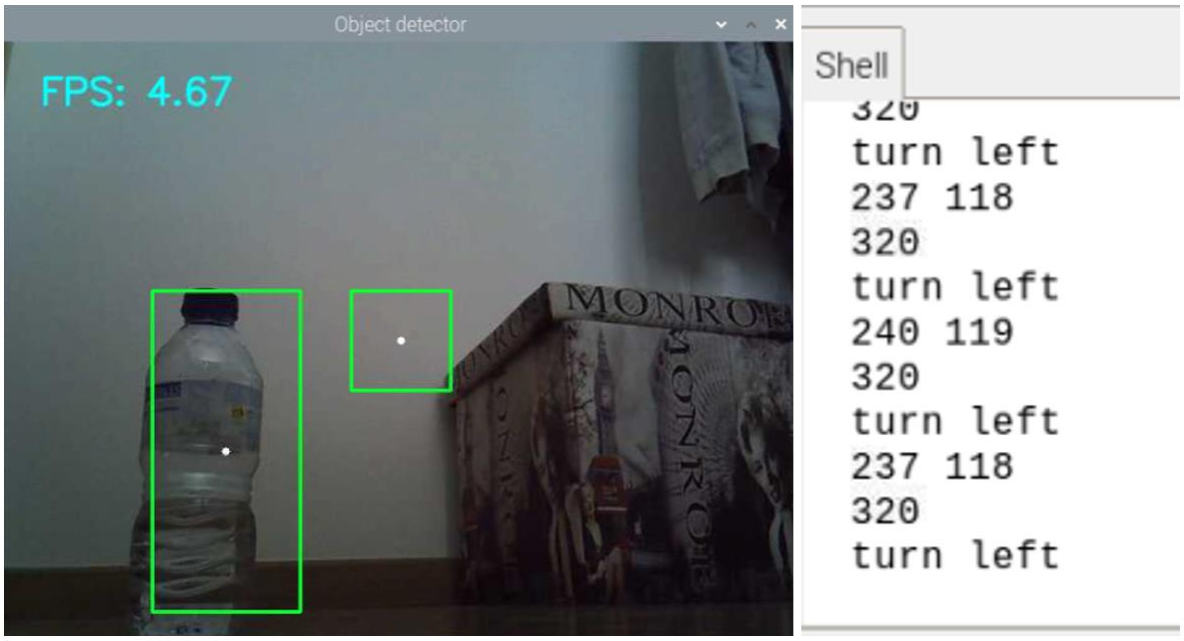



Figure 3.28 Object recognized moved to the left

Once is set inside the tolerance level in the center, if the center of the camera is below of the bottle centroid the robot is called to move forward.

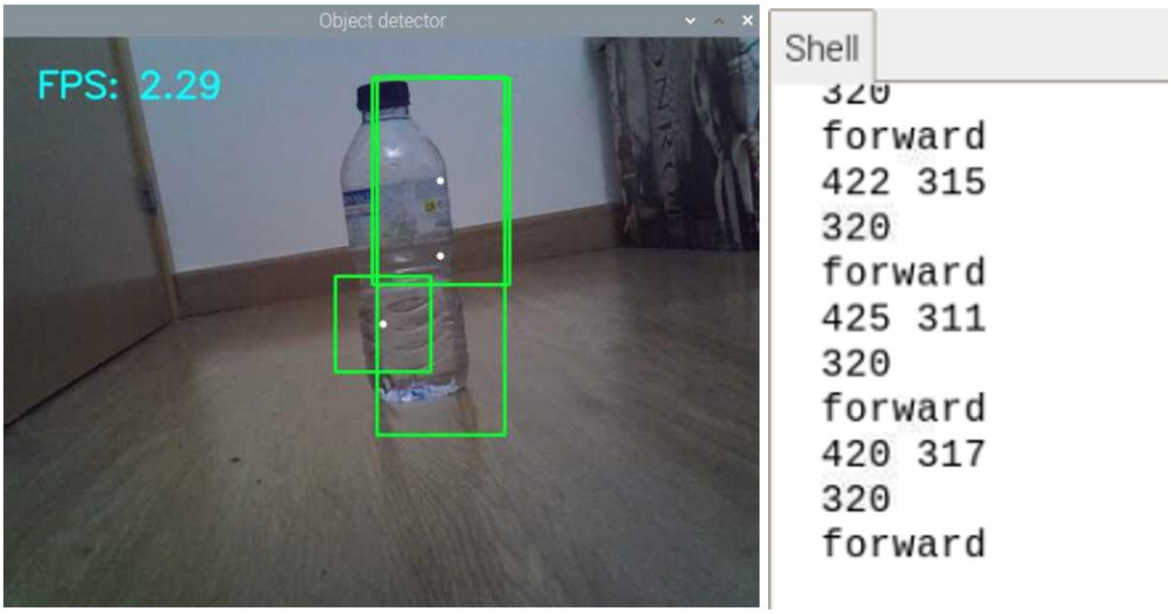



Figure 3.29 Forward recognition

Finally, and once the object centroid is inside the tolerance zone of the camera coordinates center, the actuators are told to stop.

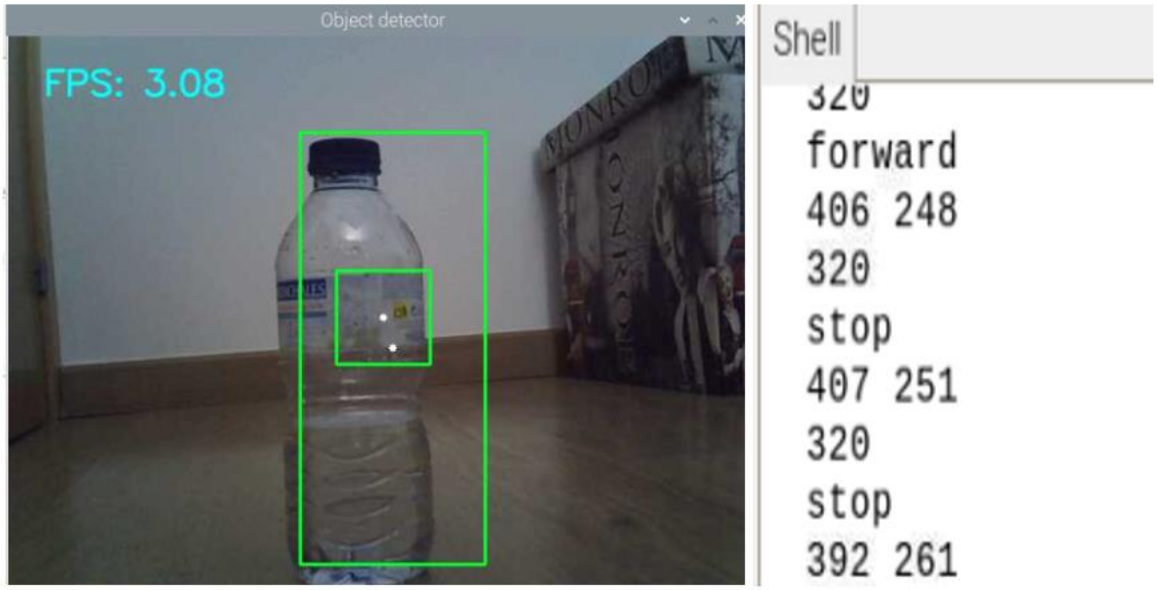



Figure 3.30 Stop recognition

## 3.5 Implementation of Software-Hardware Actuators

### 3.5.1 L298N Motor Driver Module

The motor connection and control will be managed by a L298N motor driver module. The number of driver modules necessary for the control of motors is given by the following formula:

$$\frac{nM}{2} = nDm \qquad (3.1)$$

nM: Number of motors

nDm: Number of driver modules

In this case, and for the control of 4 required motors, the number of modules of L298N needed will be 2.

An L298N consists of two H-bridges, one for output A and one for output B.

An H-bridge is a component widely used in electronics to power a load so that we can reverse the direction of the current that passes through it.

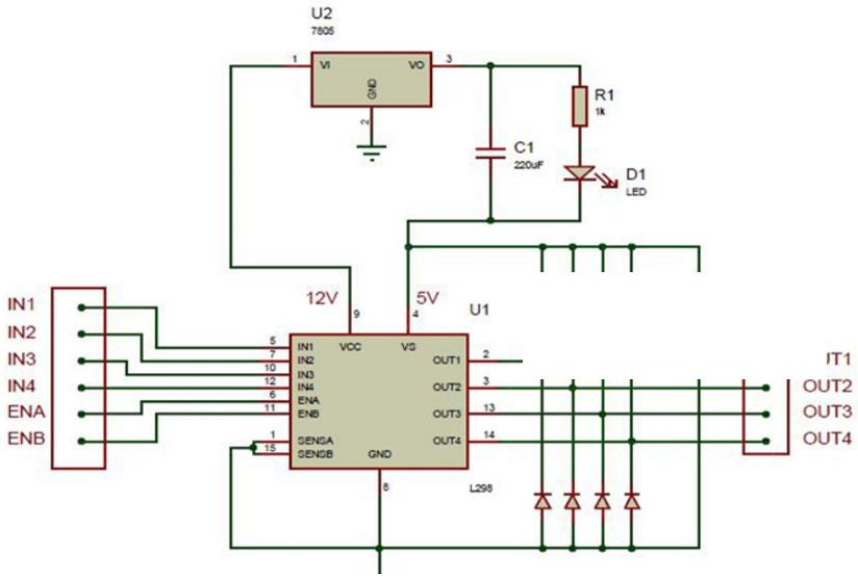



Figure 3.31 Inside scheme H-bridge motor controller

Internally an H-bridge is a formation of 4 transistors connected between Vcc and GND, with the load supplied between them. If drawn in a sketch, the set has the shape of an "H" from which it receives its name.

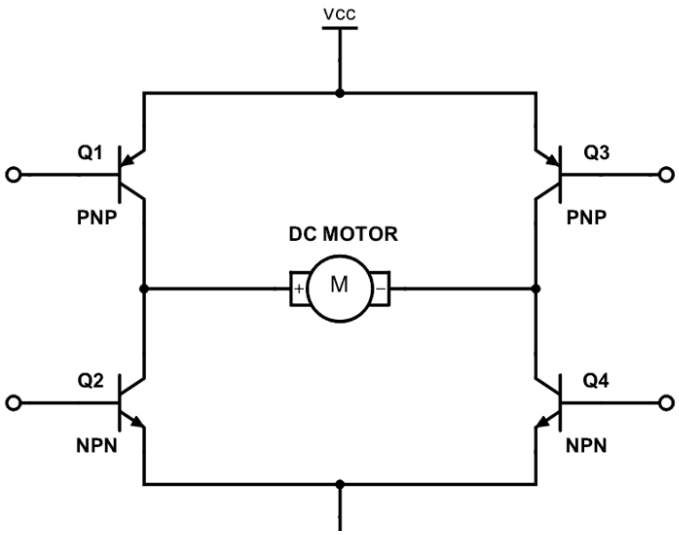



Figure 3.32 H-bridge scheme

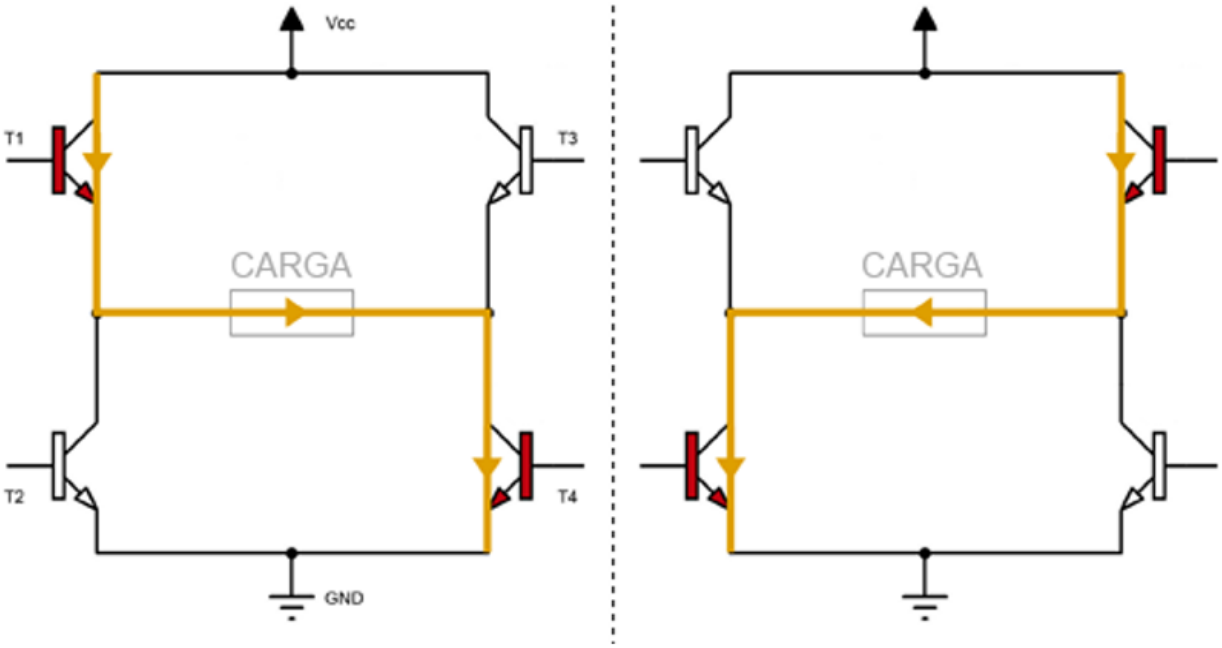



Figure 3.33 H-bridge working current flow

Acting on the 4 transistors, activating the diagonally opposite transistors of each branch, we can vary the direction in which the current passes through the load.

By simultaneously connecting the upper or lower transistors, we can put the load Vcc or GND respectively, a configuration that we will use as a brake.

Finally, we must never turn on both transistors of the same branch (left or right) since we will be causing a short circuit between Vcc and GND.

The L298N board incorporates electronics that simplify the connection to the H-bridge, grouping the connections into 3 accessible pins (for each output) and eliminating the possibility of generating a short circuit

The L298N connection board incorporates a voltage input, a series of jumpers to configure the module, two outputs A and B, and the input pins that regulate the speed and direction of rotation.

### 3.5.2 DC Motor

Once connected and ready to function, a Python script will be necessary to organize the desired actions of the motors. But first, it is required to understand how the DC motors work and how to apply this knowledge to a programming structured text.

Direct Current motors use electricity and transform it into motion by exploiting the electromagnetic induction principle. This electromagnetic law allows an electromagnetic force to appear across an electrical conductor in a changing magnetic field.

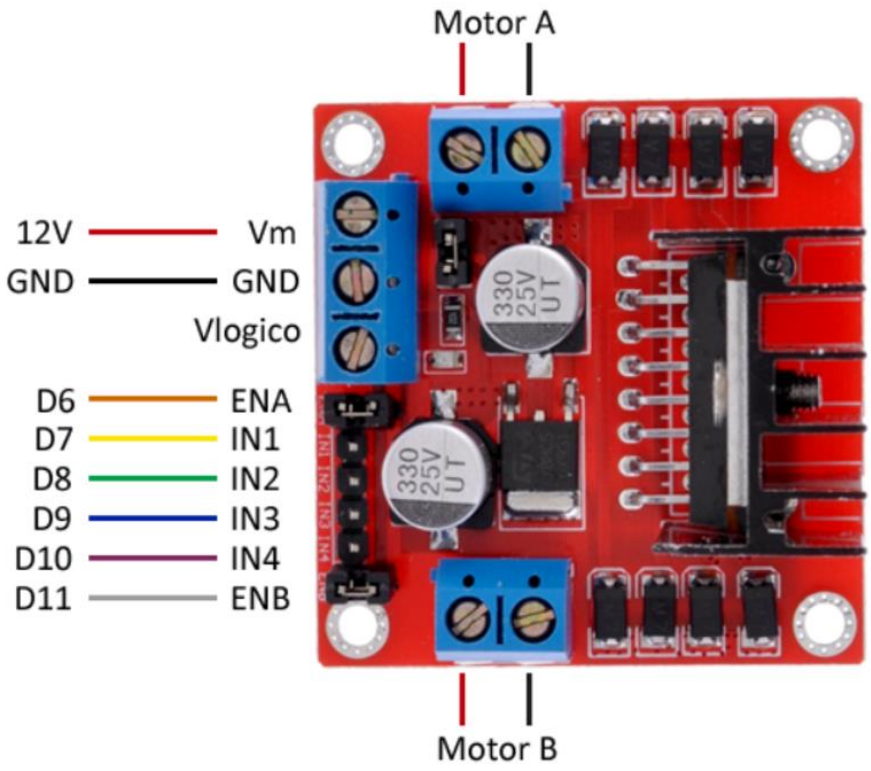



Figure 3.34 DC motor

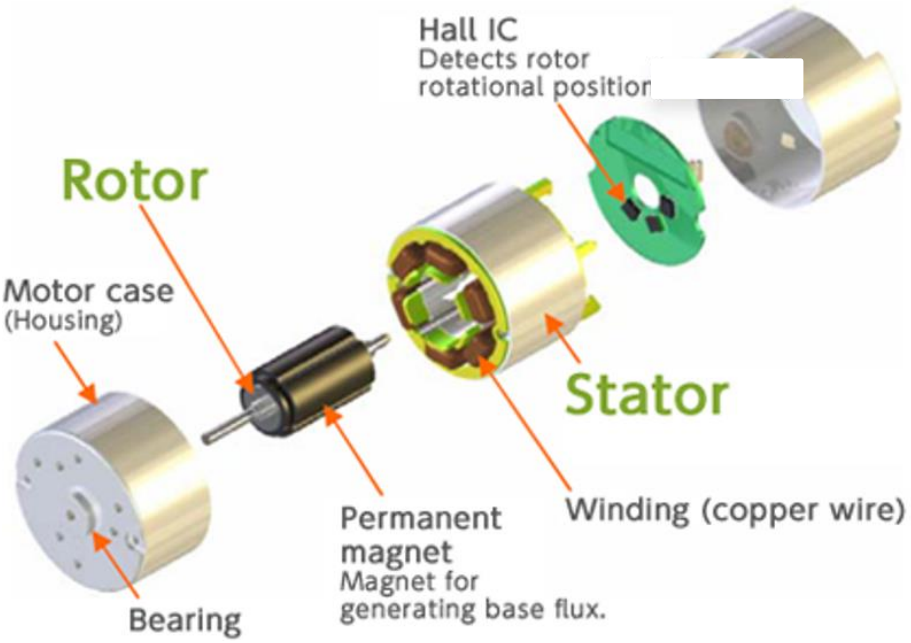



Figure 3.35 DC motor explosive view

A motor uses a stator that generates the changing magnetic field around a turning coil of wire called rotor or armature connected to a battery (basically an electromagnet). This will create a magnetic field around the wire.

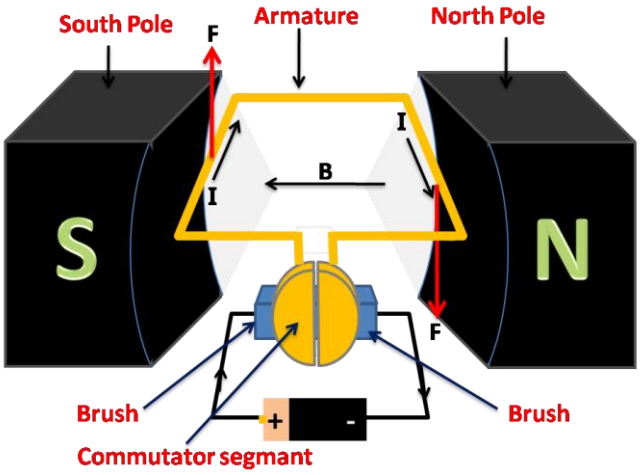



Figure 3.36 Inner principle working of a DC motor

The key to producing motion is positioning the electromagnet within the magnetic field of the permanent magnet (the field runs from its north to south poles). The armature experiences a force described by the left-hand rule.

This interplay of magnetic fields and moving charged particles (the electrons in the current) results in the torque (depicted by the red arrows in Figure 3.36), which makes the armature spin. With a single 180 degree-turn by the constant polarity change it is possible to mimic the needed rotatory movement for the robot wheels.

With this information about the nature of the movement source of the robot it is possible now to design a structured programming script for the motors inside the Raspberry Pi 4 and using python as a controlling code.

It is important to consider the pins used for the motor driver module connection because this will be important for the script development.

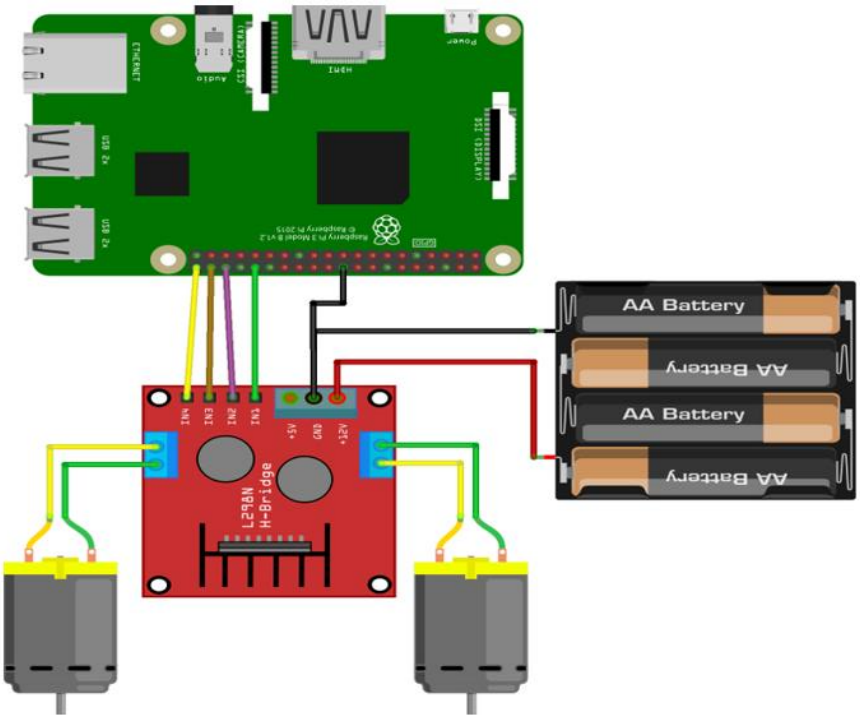
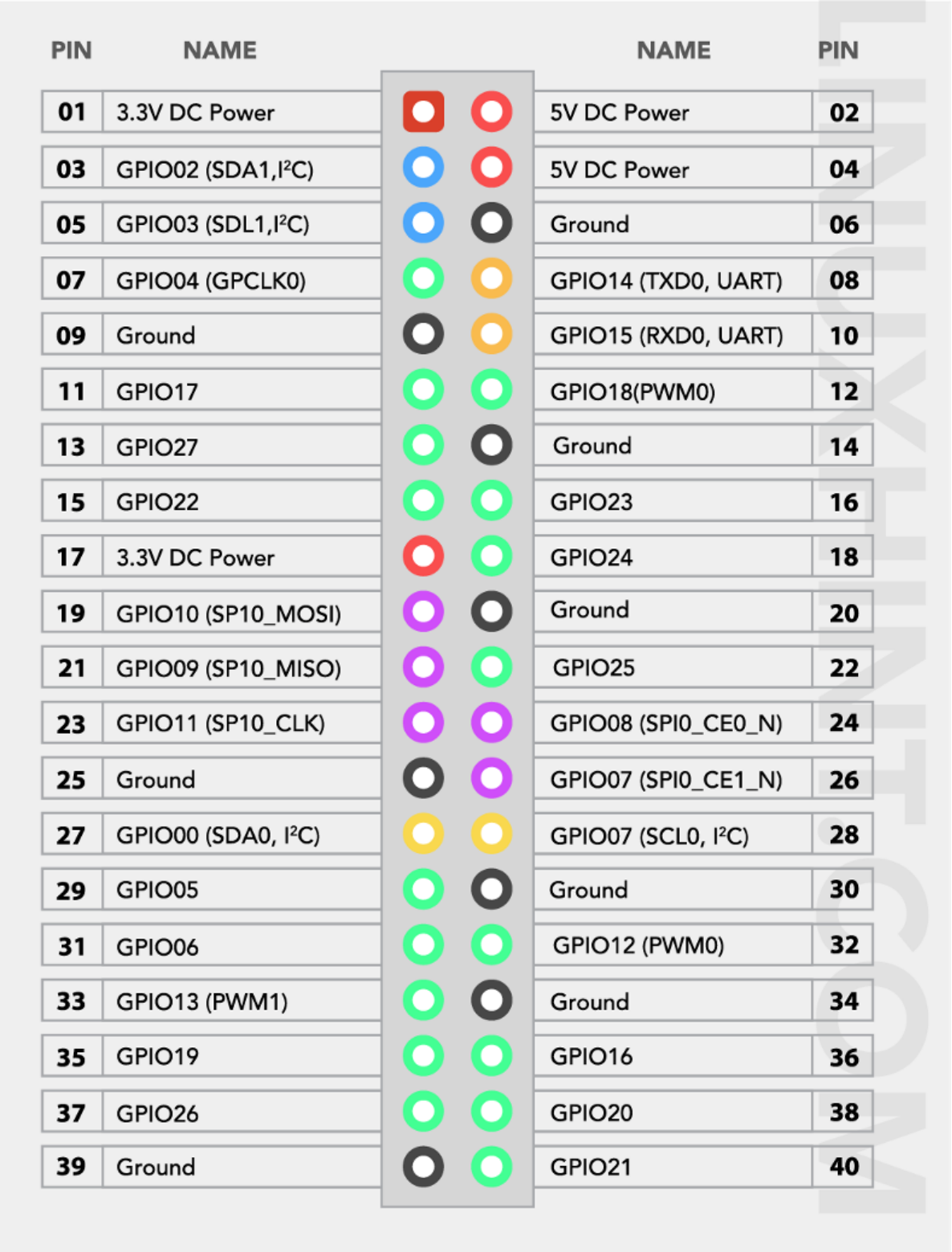



Figure 3.37 Simple working model of L298N

Luckily, the RPI4 does have two connections to 5V DC Power, and an extra PWN module controller is not needed to distribute the signal between the different extensions of the robot.



| PIN | NAME | NAME | PIN |
|---|---|---|---|
| 01 | 3.3V DC Power | 5V DC Power | 02 |
| 03 | GPIO02 (SDA1,I²C) | 5V DC Power | 04 |
| 05 | GPIO03 (SDL1,I²C) | Ground | 06 |
| 07 | GPIO04 (GPCLK0) | GPIO14 (TXD0, UART) | 08 |
| 09 | Ground | GPIO15 (RXD0, UART) | 10 |
| 11 | GPIO17 | GPIO18(PWM0) | 12 |
| 13 | GPIO27 | Ground | 14 |
| 15 | GPIO22 | GPIO23 | 16 |
| 17 | 3.3V DC Power | GPIO24 | 18 |
| 19 | GPIO10 (SP10_MOSI) | Ground | 20 |
| 21 | GPIO09 (SP10_MISO) | GPIO25 | 22 |
| 23 | GPIO11 (SP10_CLK) | GPIO08 (SPI0_CE0_N) | 24 |
| 25 | Ground | GPIO07 (SPI0_CE1_N) | 26 |
| 27 | GPIO00 (SDA0, I²C) | GPIO07 (SCL0, I²C) | 28 |
| 29 | GPIO05 | Ground | 30 |
| 31 | GPIO06 | GPIO12 (PWM0) | 32 |
| 33 | GPIO13 (PWM1) | Ground | 34 |
| 35 | GPIO19 | GPIO16 | 36 |
| 37 | GPIO26 | GPIO20 | 38 |
| 39 | Ground | GPIO21 | 40 |

Figure 3.38 Raspberry Pi4 PIN distribution

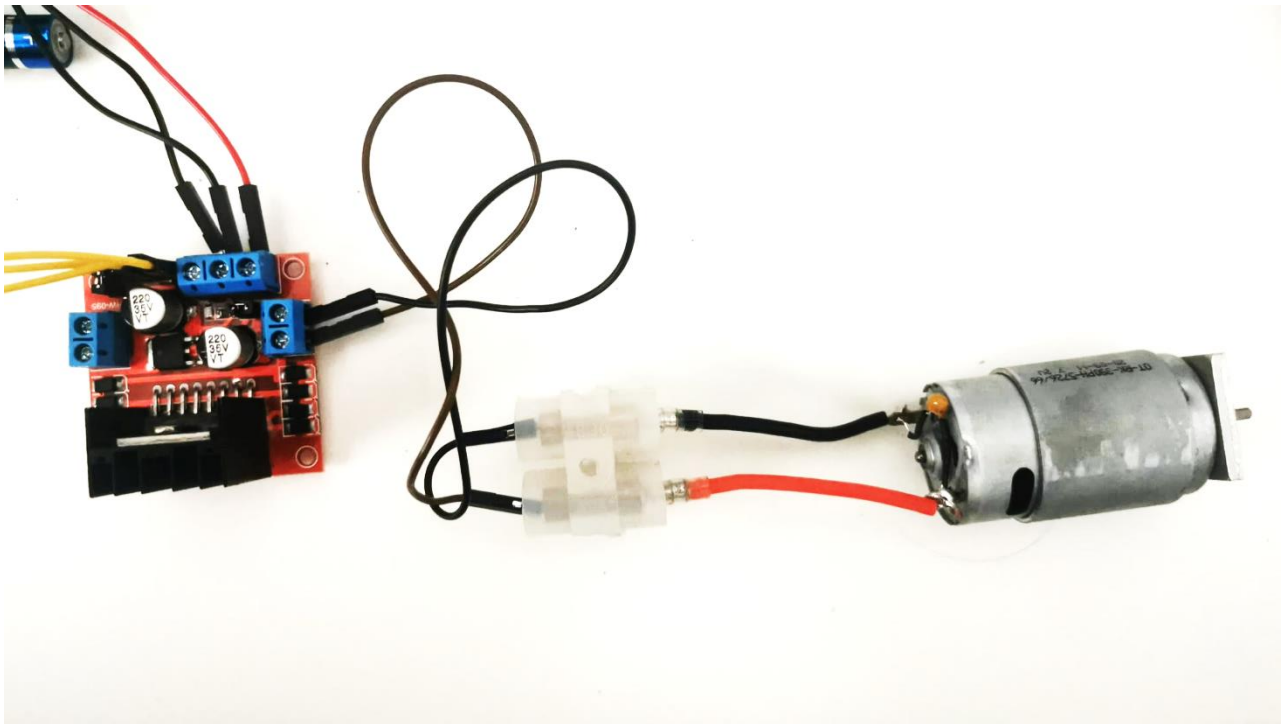


Figure 3.39 Example of connection applied to the prototype

### 3.5.3 Robot Kinematics into Software

For the final step before programming the motor control, it is necessary to understand the desired motion. For this instance, the designed kinematics of the robot movement is of utmost importance to complete the software.

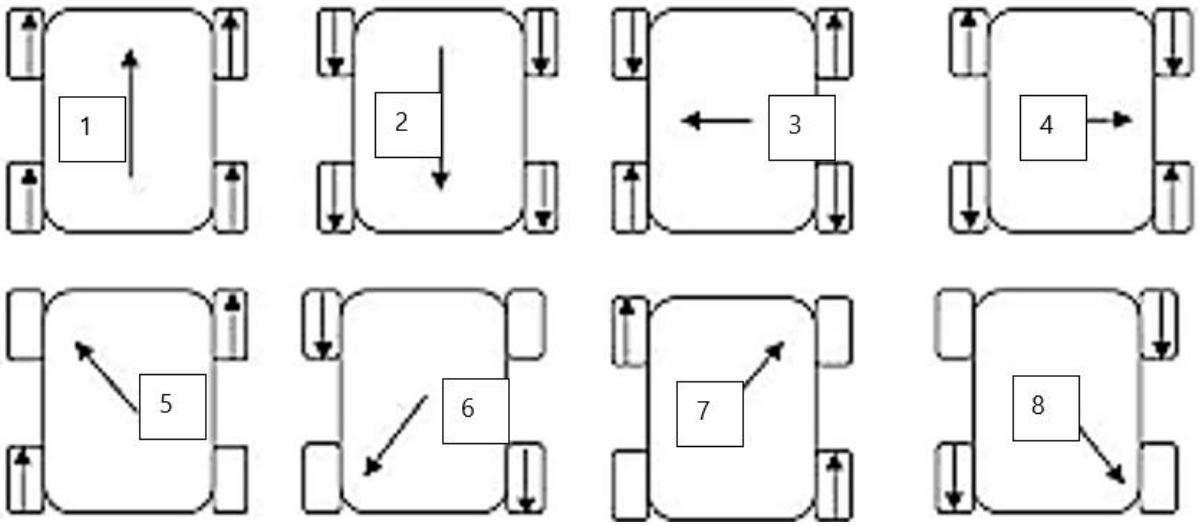



Figure 3.40 Kinematics with respective command number

The robot has eight possible movements. This translate into eight different functions that can be implemented in the final program depending on the camera input. For instance, in Figure 3.40, the forward motion will require the forward running of the four motors at the same time, while a left horizontal motion will use the backward motion of motors and forward motion of motors.

The first option used is programming using the L298N module principles.

```
#importing of modules

import RPi.GPIO as GPIO
import os
import time



#Motor 1 PINS of the Raspberry PI 4
in1a = 23
in2a = 24
ena = 18

#Motor 2 PINS of the Raspberry PI 4
in1b = 6
in2b = 5
enb = 19

#Motor 3 PINS of the Raspberry PI 4
in1c = 17
in2c = 27
enc = 22

#Motor 4 PINS of the Raspberry PI 4
in1d = 16
in2d = 20
```

```
end = 21

#Configuration of the uP
GPIO.setmode(GPIO.BCM)

#Set-up of PINS for the enable command
GPIO.setup(ena,GPIO.OUT)
GPIO.setup(enb,GPIO.OUT)
GPIO.setup(enc,GPIO.OUT)

#Set-up of PINS for the motor A
GPIO.setup(in1a, GPIO.OUT)
GPIO.setup(in2a, GPIO.OUT)

#Set-up of PINS for the motor B
GPIO.setup(in1b, GPIO.OUT)
GPIO.setup(in2b, GPIO.OUT)

#Set-up of PINS for the motor C
GPIO.setup(in1c, GPIO.OUT)
GPIO.setup(in2c, GPIO.OUT)

#Set-up of PINS for the motor D
GPIO.setup(in1d, GPIO.OUT)
GPIO.setup(in2d, GPIO.OUT)


#Outputs of the Pulse width module.
pwm_a = GPIO.PWM(ena,500)
pwm_b = GPIO.PWM(enb,500)
pwm_c = GPIO.PWM(enc,500)
pwm_d = GPIO.PWM(end,500)
```

```
#start-up of the PWN at 0
pwm_a.start(0)
pwm_b.start(0)
pwm_c.start(0)
pwm_d.start(0)

# Motor A

def  forward_motor_a():
   GPIO.output(in1a,False)
   GPIO.output(in2a,True)

def backward_motor_a():
   GPIO.output(in1a,True)
   GPIO.output(in2a,False)

def stop_motor_a():
    pwn_a.stop

#Motor B

def  forward_motor_b():
   GPIO.output(in1b,False)
   GPIO.output(in2b,True)

def backward_motor_b():
   GPIO.output(in1b,True)
   GPIO.output(in2b,False)

def stop_motor_b():
    pwn_b.stop
```

```
#Motor C

def  forward_motor_c():
   GPIO.output(in1c,False)
   GPIO.output(in2c,True)

def backward_motor_c():
   GPIO.output(in1c,True)
   GPIO.output(in2c,False)

def stop_motor_c():
    pwn_c.stop


#Motor D

def  forward_motor_d():
   GPIO.output(in1d,False)
   GPIO.output(in2d,True)

def backward_motor_d():
   GPIO.output(in1d,True)
   GPIO.output(in2d,False)

def stop_motor_d():
    pwn_d.stop

############################################################

#Start of the movement functions

#Forward
def forward_movement():
```

```
    forward_motor_a()
    forward_motor_b()
    forward_motor_c()
    forward_motor_d()
#Backward
def backward_movement():
    backward_motor_a()
    backward_motor_b()
    backward_motor_c()
    backward_motor_d()
#Stop
def stop_movement():
    stop_motor_a()
    stop_motor_b()
    stop_motor_c()
    stop_motor_d()
#left horizontal
def left_horizontal():
    backward_motor_a()
    forward_motor_b()
    forward_motor_c()
    backward_motor_d()
#right horizontal
def right_horizontal():
    forward_motor_a()
    backward_motor_b()
    backward_motor_c()
    forward_motor_d()
#diagonal left up
def diagonal_left_up():
    stop_motor_a()
    forward_motor_b()
    forward_motor_c()
```

```
    stop_motor_d()
#dagonal left down
def diagonal_left_down():
    backward_motor_a()
    stop_motor_b()
    stop_motor_c()
    backward_motor_d()
#diagonal right high
def diagonal_right_high():
    forward_motor_a()
    stop_motor_b()
    stop_motor_c()
    forward_motor_d()
#diagonal right low
def diagonal_right_low():
    stop_motor_a()
    backward_motor_b()
    backward_motor_c()
    stop_motor_d()
```

This script dictates the core functions of movements depending on the selected PINS above.

After completing the programming of the motors, it is quite big and it could be simplified by adding a AdaFruit DC Stepper Motor Hat for Raspberry Pi, which streamlines the process of programming. This component is out of stock due to the current chip shortage.

Now that it is programmed, it is possible to apply the full schematic needed made with the Fritzing software.

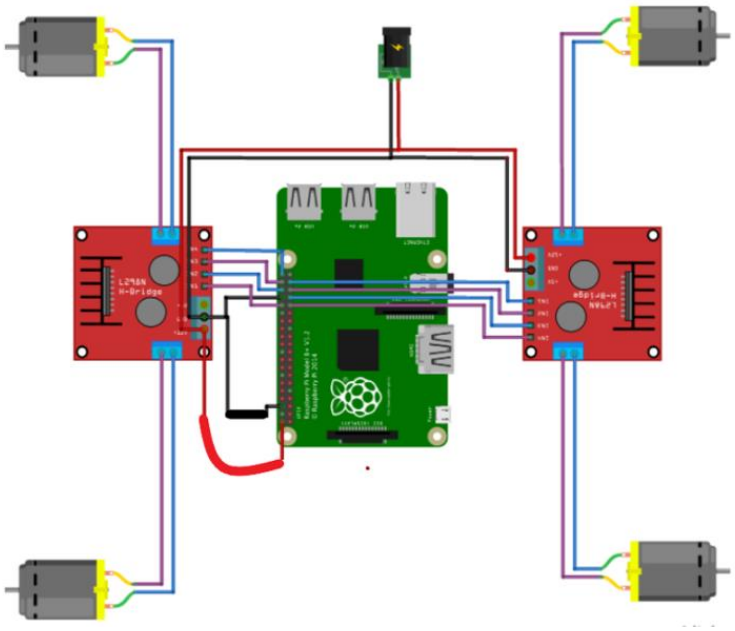


Figure 3.41 Connection wanted between motor drivers and Raspberry Pi

Once the functions defining each possible movement of the robot are completed, it is possible to execute the visual recognition script with the additional movements addon created and the defined functions in the object recognition script.

The movement numbers of the commands correspond to the movements indicated in Figure 3.40.

| **Movement Nº** | **Function name .py** |
|---|---|
| 1 | forward_movement(): |
| 2 | backward_movement(): |
| 3 | left_horizontal(): |
| 4 | right_horizontal(): |
| 5 | diagonal_left_up(): |
| 6 | diagonal_left_down(): |
| 7 | diagonal_right_high(): |

| | |
|---|---|
| 8 | diagonal_right_low(): |
| STOP | stop_movement(): |

Table 2 Programmed list of functions regarding the movement

### 3.5.4 Camera Software Development for Object Following

With the object recognition and the movement of the motors operational, a problem occurs in essence of the thesis objective. It is wanted that the robot localizes objects and acts accordingly to the user wishes or simply follow an object. But the robot does not have enough information to know how to associate the detected object to the movement of its wheels.

For addressing this problem, a cartesian mathematic solution is presented:

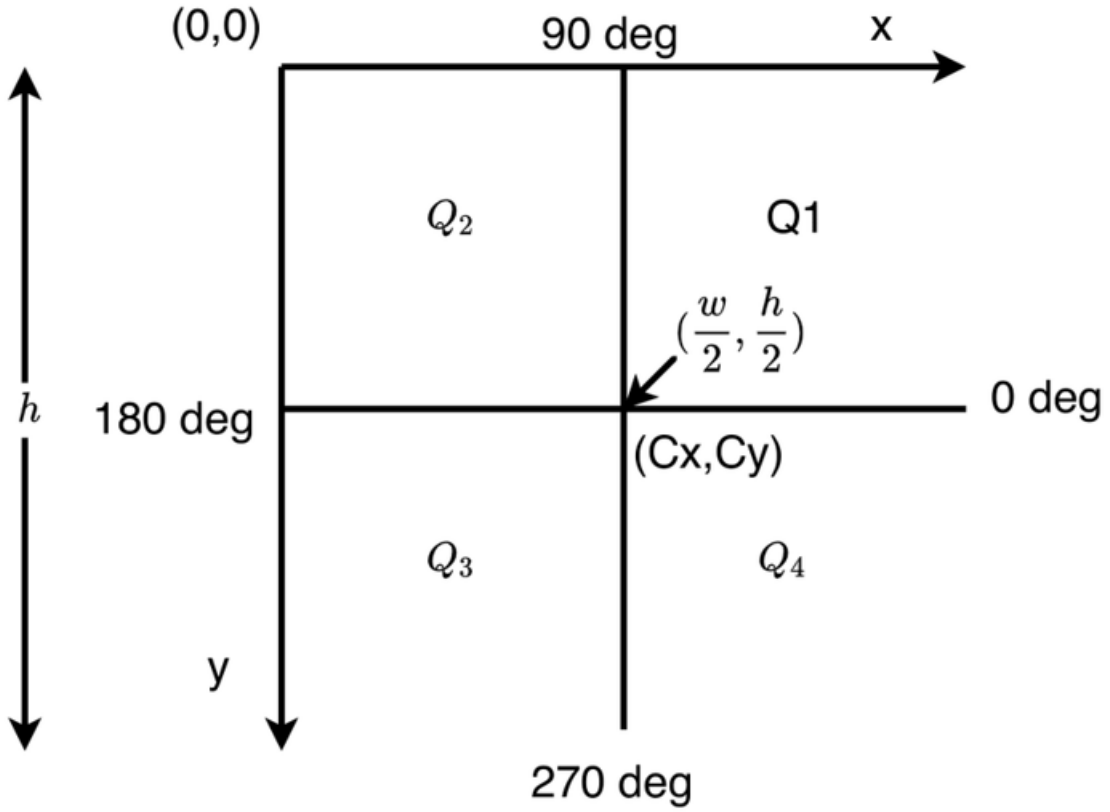



Figure 3.42 Camera frame quadrants

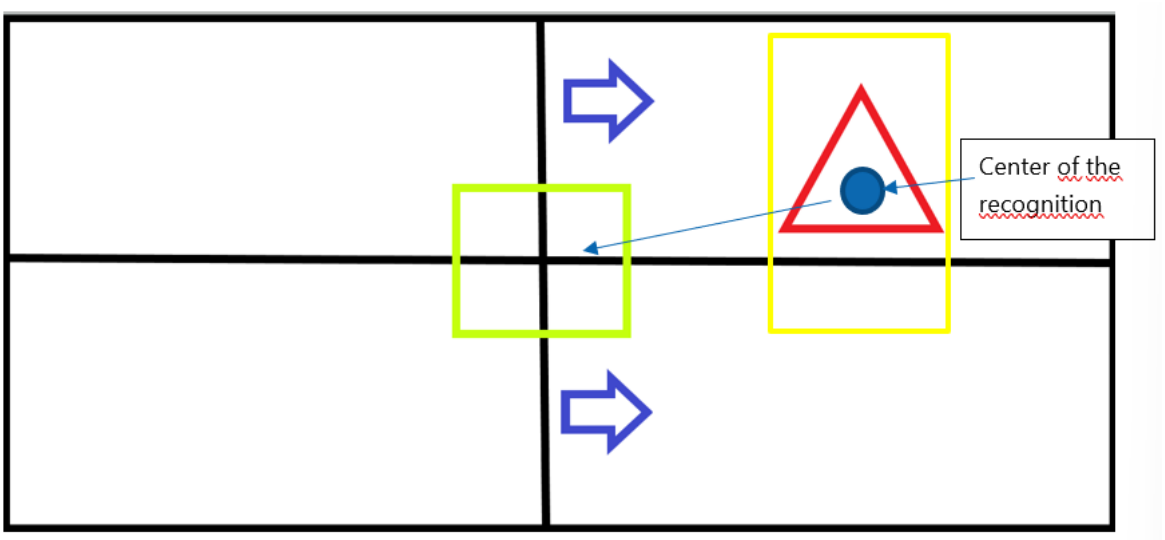



Figure 3.43 Wanted centering of object

Using the camera as a real point in the real world, it is possible to dictate a quadratic view for the robot to associate the movements with the object detection.

As seen above, the input captured by the camera can be seen as four quadrants of the entire video capture. For the robot to follow an object, it will be necessary that the object coordinates are always in the center of the entire capture. The robot will try to maintain the object into the central (Cx,Cy) position.

By working with the current system and CPU power there will be tolerance errors of every type. To solve this issue, a smaller square is drawn with (Cx,Cy) as its center. Every object inside the smaller square will be considered correct and in position with the robot position matrix.

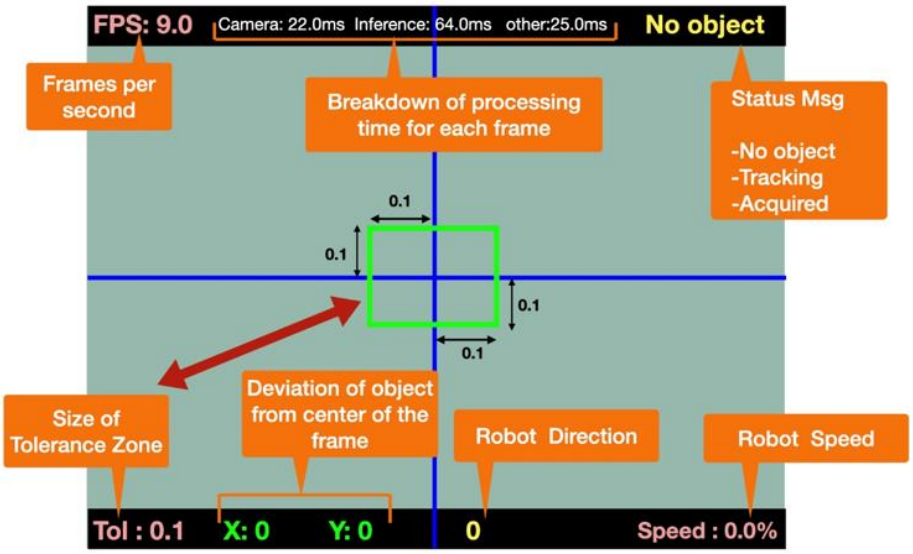



Figure 3.44 Tolerance idea for camera

The intelligence will know that there is a triangle in the image. However, it will be forced to position the triangle view (which has a rectangle around its center identifying it) into the tolerance box in the middle, thus making the robot turn right with the function described in the code.

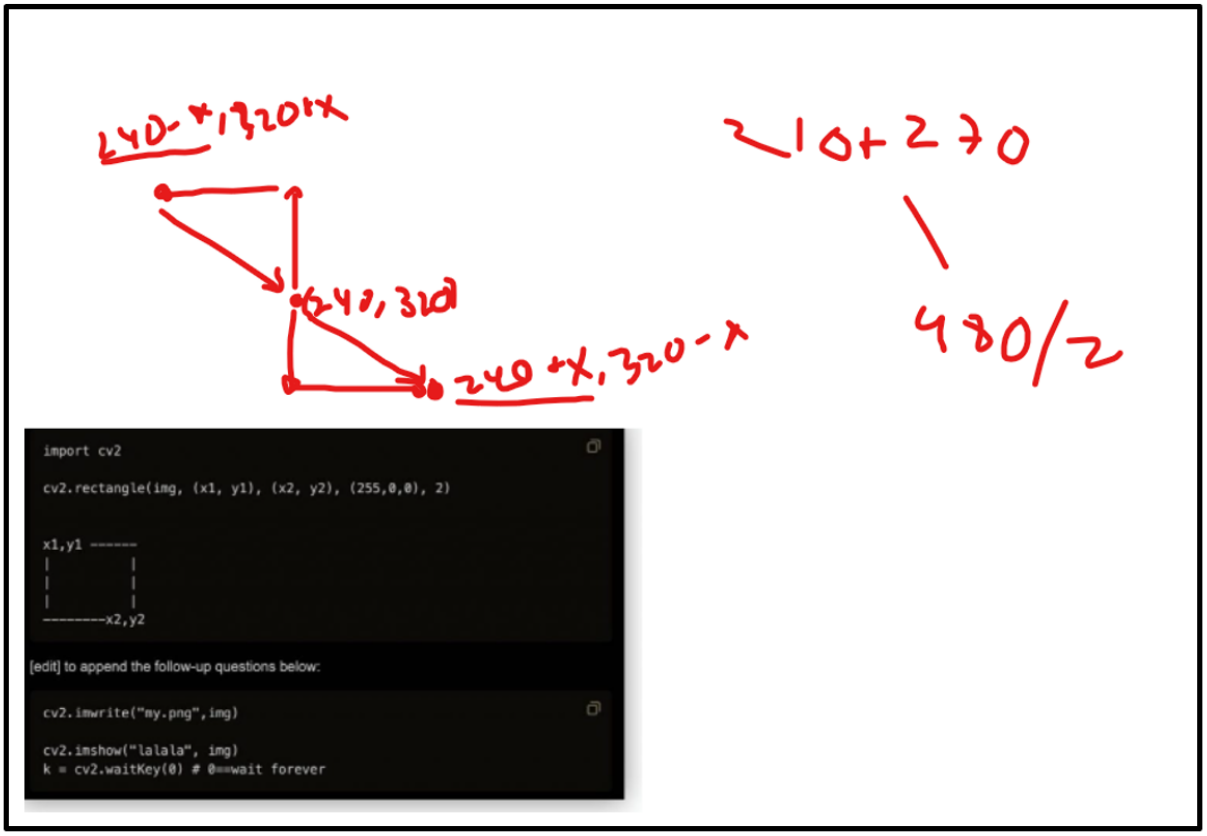



Figure 3.45 calculations for the application of tolerance rectangles

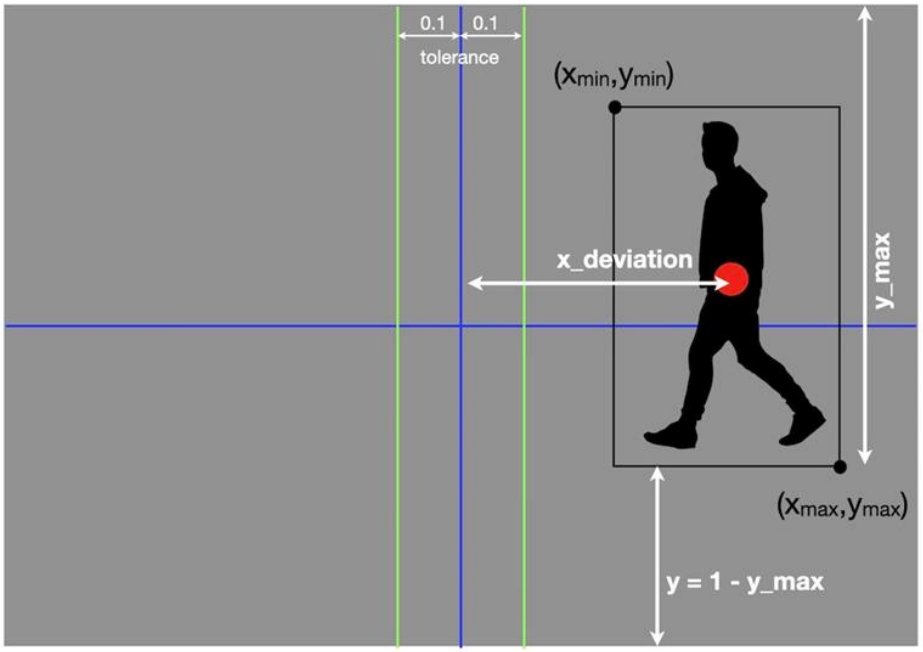



Figure 3.46 Deviation consideration proposal

This solution generates unlimited possibilities of configuration for the result, but an important one is a possibility of following a human to help carry items or whatever the owner wants.

The result may experience lag resulting from the CPU calculation power capacity and the velocity of the desired object to recognize.

It is crucial to bear in mind that the world in which the robot lives is bidimensional and perfectly plane; there is no addition of the z-axis. Therefore, it is not equipped to move the camera up and down.

### 3.5.5 PWM Control

After the configuration of the motors and the cameras inputs, there was a vital need of speed control of the motors due to the DC motors running at full speed every time when being actioned. So how can motors, an analog controlled device, be controlled if the signals are 1 or 0?

It is known that digital signals working inside the motor module controller and the RPI4 have two positions: ON or OFF, or better known as 1 or 0. Analog signals, on the other hand, can be on, half-way, two-thirds, off or all the way to an infinite number of positions between 0 and 1.

In electronics analog and digital are handled differently but are often put together to work in harmony.

Often in engineering there is a need to translate that digital output into an analog device so it can understand the desired behavior needed. A solution to tackle this problem uses the Pulse Width Modulation or PWM.

PWM is a tricky technique because it is not a true analog output in nature. Instead, it mimics an analog-like result by applying power in pulses of regulated voltage.

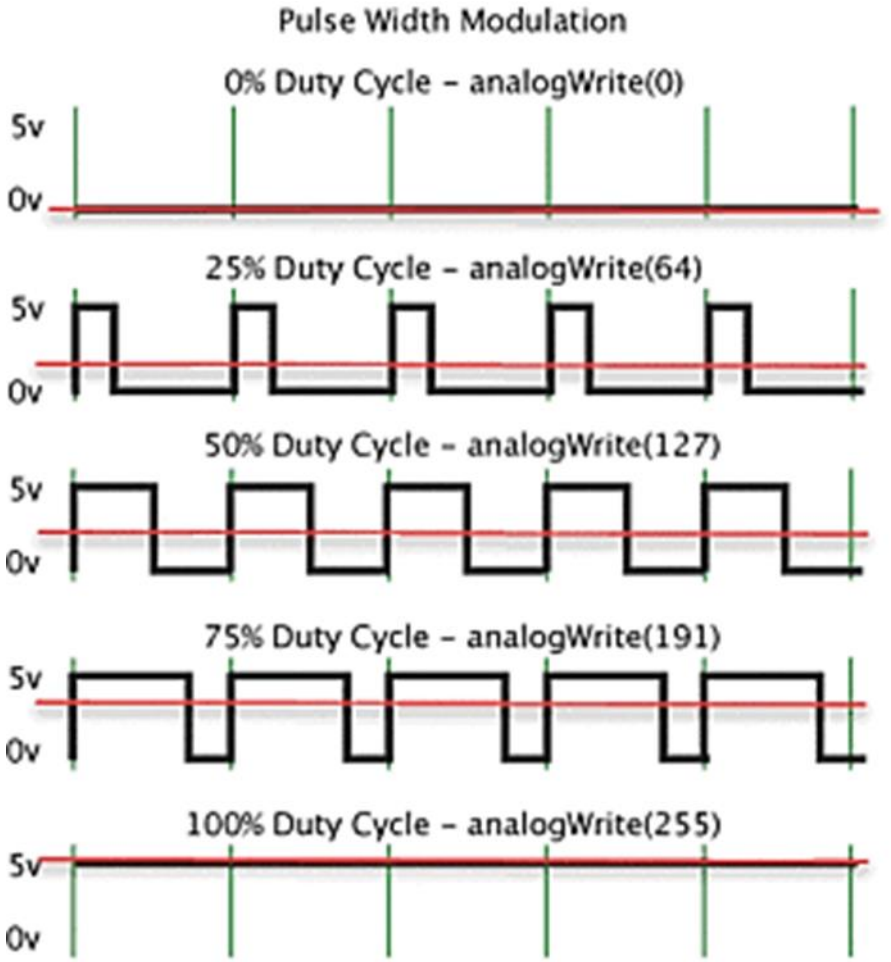



Figure 3.47 PWM diferent duty cycles

For this case, the voltage given by the Raspberry Pi 4 to the L298N is a maximum of 5 volts, which will always be the value in case there is an ON signal coming out of the PIN. If applied, the PWM method can regulate the final voltage otorgued to the motor controller, thus tricking the mentioned controller into giving the motor less or more voltage. As it is known, these variations in voltage will vary the resulting speed of the DC motor.

It is taken into consideration that in this case, the PWM is efficient due to the first law of motion, not stopping the motor abruptly when the power goes off and on in small intervals of time. This plays a crucial role in the result of the general behavior.

*"LAW I. Every body perseveres in its state of rest, or of uniform motion in a right line, unless it is compelled to change that state by forces impressed thereon".*

Now that it is clear what the PWM is, how do we translate its inner behavior into aspects of tangible electronic concepts such as frequency or software?

For this, maximum voltage, frequency, and the duty cycle are the factors to consider when applying this method to the software. If taken the duty cycle and multiplied by the maximum voltage level, an average voltage level will result in what the motor is seeing at that moment in time.

$$Instant\ Duty\ cycle * Maximum\ Voltage\ Level = Insta.\ Average\ Voltage \qquad (3.2)$$

The duty cycle can change to affect the average voltage that the motor experiences. The frequency of the cycles can increase. The pulse can be increased even in length. These can all happen together, too, but in general, it's easier to think of as either duty cycle increasing or frequency increasing to increase the speed of the motor.

The only thing that hasn't changed in all of this is the high voltage level. Because "no" is always the same for the digital output, merely flicking the output on-and-off at varying speeds and for changing periods of time. This is how you get pulse width modulation to fake an analog output. MCUs are digital. An example of something that can create a true analog output would be a transducer.

To sum up, to control the average voltage, it is only possible (via software) to vary the instant duty cycle with a fixed power source powering the RPI and the controller. To do this, a python API is inside the RPI.GPIO library is used and introduced into the code. First we set the PIN (ena = pin 18) wanted to produce the PWM and the desired Hz (500):

*pwm_a = GPIO.PWM(ena,500)*

Following by the initial set-up of the duty cycle at the beggining of the code:

*pwm_a.start(0)*

Now the system is set up, but a final consideration is to be taken; the RPI only gives 5V, but the controller translates this 5V into 12V in this case (it depends on the powering of the controller). A controlled duty cycle would mean different speeds:

| Duty Cycle (%) | Average Voltage – 12V (V) | Average Voltage – 9V (V) |
|---|---|---|
| 0 | 0 | 0 |
| 25 | 3 | 2.25 |
| 50 | 6 | 3.5 |
| 60 | 7.2 | 5.4 |
| 75 | 9 | 6.75 |
| 80 | 9.6 | 7.2 |
| 100 | 12 | 9 |

Table 3 Motors duty cycle to Voltage relation

To control the current test motor of 7.2V, a duty cycle of 60% for a Vcc = 12V would work or a 80% for a Vcc = 9V as well. Because of this data, we can choose the 12Vcc source to have more overall control. However, the L298N takes up to 3V out of the Vcc to function properly. So, the actual duty cycle will be superior to the ideal one.

When testing the PWM software with the motors, a problem occurred that was not taken into much thought before. That is that the motors usually need a peak starting voltage for them to be able to start-up properly.

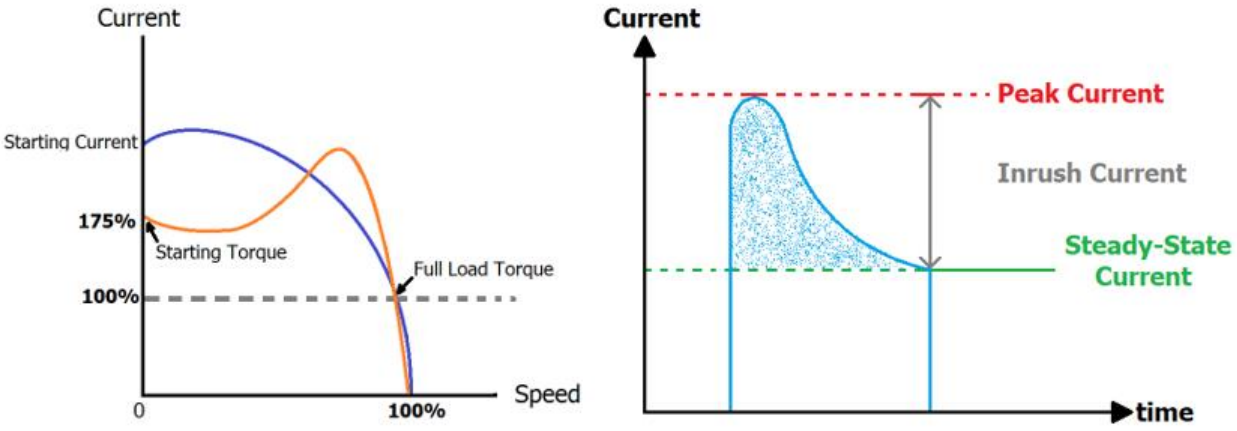



Figure 3.48 Current peaks

To solve the inrush current problem via software, the starting duty cycle was set to a 100% for 1ms before the DC motor started running so it would be able to use the maximum voltage(current) to rotate and after said time the duty cycle will change into the nominal speed of the motor.

*Pwm_a.ChangeDutyCycle(100)*
*Sleep(0.01)*
*Pwm_a.ChangeDutyCycle(nominal)*

## 3.6 Shape detection and Lane following

Implementing a routine that can recognize a predetermined lane and give information to the motors so that the robot follows the lane is possible with the pixel summation method.

First, the pixel summation method uses the recognition of an image and all its bits value, which are added in the frame to give out a determined value. This is accomplished by turning the frame into a grayscale, in which the numbers range from 0 to 255, as of the 8-bit value integer standard.

$$2^8 = 256 \rightarrow [0 - 255] \qquad (3.3)$$

With the value 0 assigned to a black bit colour and the value 255 to a White bit color, respectively.

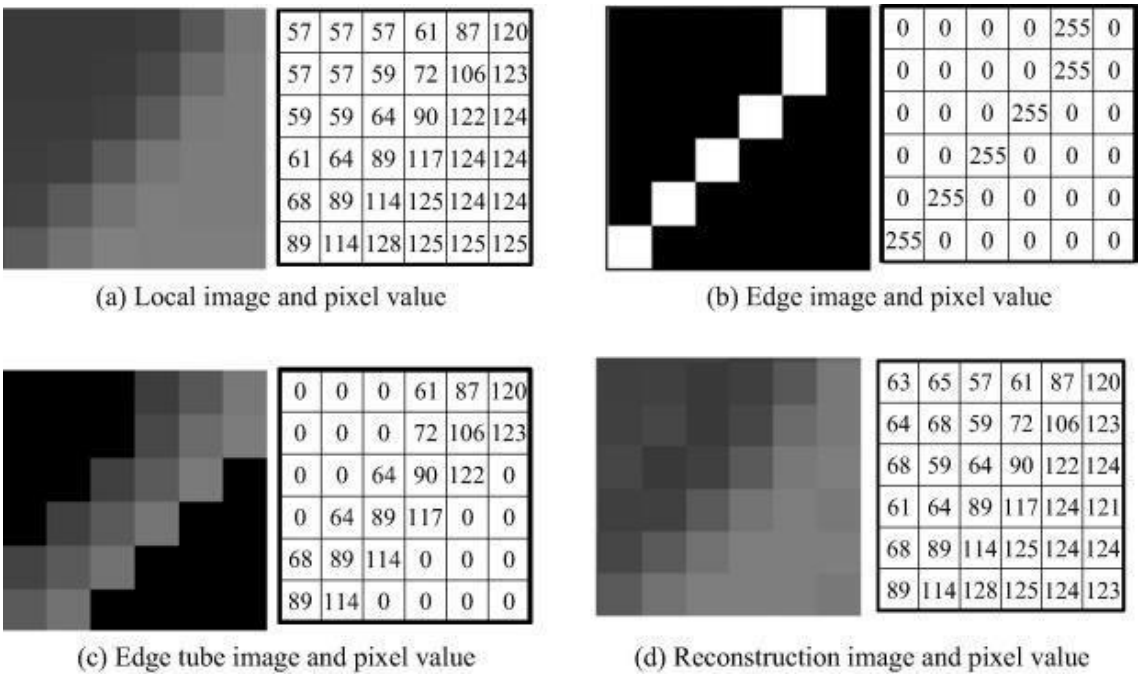



Figure 3.49 Examples of grey-scaling binary matches

This method is widely used for the differentiation of objects by its colour, and it is very effective in the fields of self-driving objects that lack radars and with clearly determined paths. As seen in the previous figures, the frame image introduced to the robot is converted into a binary colour composition of black or white, and the bits of value 255 will be the ones used to determine the road.

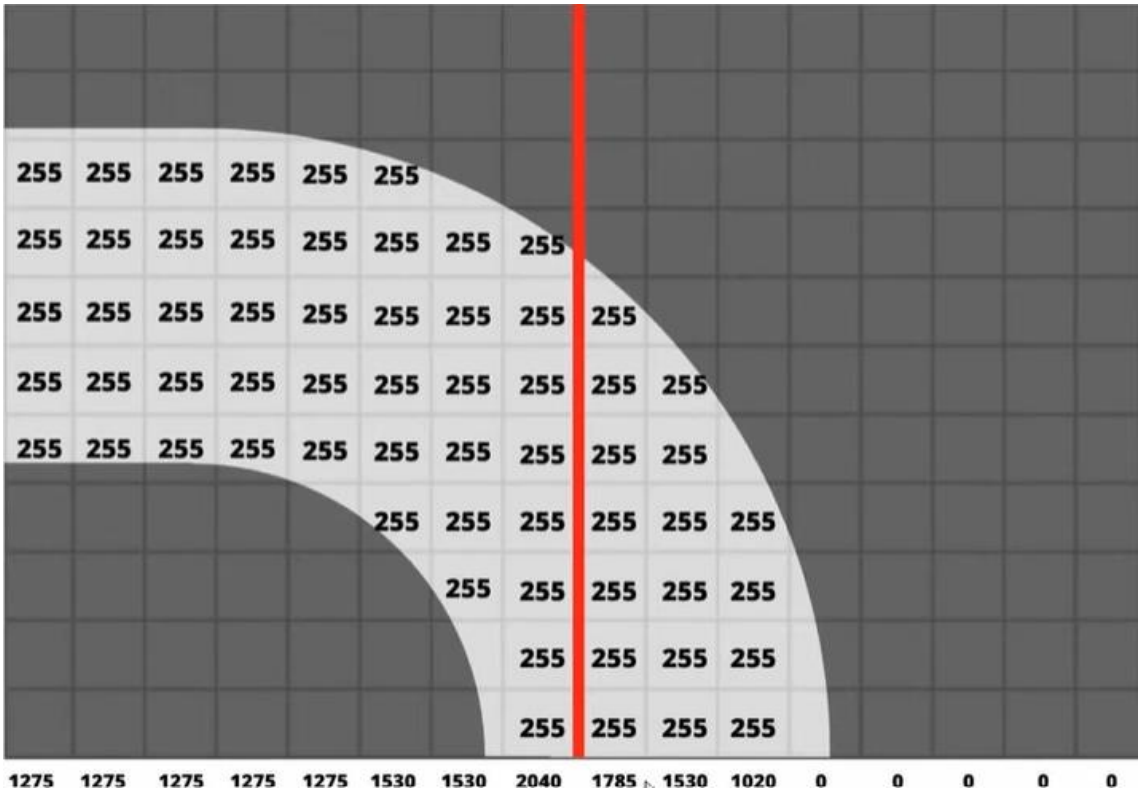



Figure 3.50 Summatory of pixels

As a result of this method, the robot will follow a white line using the colour recognition detector. It is possible to use edge detection to complete it. A thresholding function is created to differentiate the wanted path by using the gray scaling of the frame.

```
1 def thresholding(img):
2     hsv = cv2.cvtColor(img,cv2.COLOR_BGR2HSV)
3     lowerWhite = np.array([85, 0, 0])
4     upperWhite = np.array([179, 160, 255])
5     maskedWhite= cv2.inRange(hsv,lowerWhite,upperWhite)
6     return maskedWhite
```

Figure 3.51 Threshold programming

Figure 3.52 Lane with applied threshold

It was also tried to implement the Canny Edge detection technique, and the results were optimal as well.

Figure 3.53 Lane with applied canny edge

In the next step, a problem occurs; the images taken at the start of the programming were taken with the camera placed with a bird view perspective. It is wanted that the robot has the camera looking onward and not looking at the floor. Still, the bird-eye view is perfect for optimal work of the code. Thus to maintain this, a warping methodology is used.

With the warping of the image, the robot can identify the curves of the lane at the correct time. This is widely used in the motor industry to help the user identify its surroundings better with

a simulation of a third view perspective. It is crucial to clarify that these cars use cameras from all points of view and the current robot only uses one limiting its knowledge to only one plane of view.

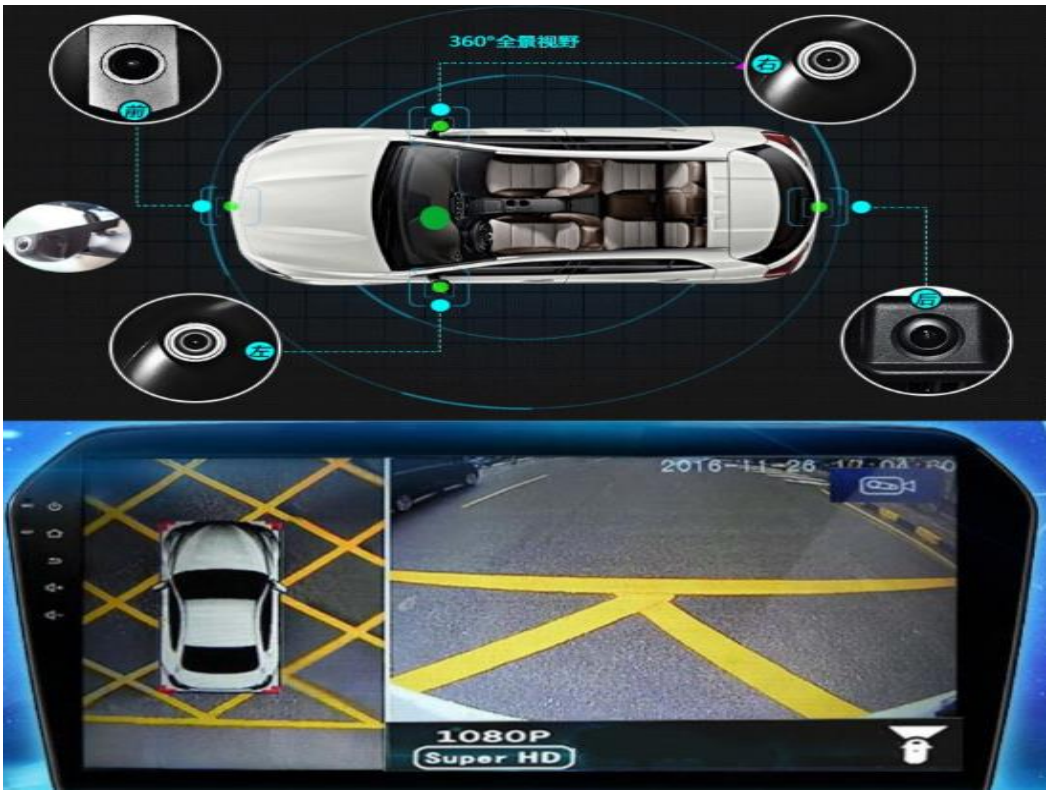



Figure 3.54 Bird-eye perspective in modern cars

Luckily, with the Open CV software there is a function already installed within the module that performs the warping methodology.

```
def warpImg (img,points,w,h,inv=False):
    pts1 = np.float32(points)
    pts2 = np.float32([[0,0],[w,0],[0,h],[w,h]])
    if inv:
        matrix = cv2.getPerspectiveTransform(pts2,pts1)
    else:
        matrix = cv2.getPerspectiveTransform(pts1,pts2)
    imgWarp = cv2.warpPerspective(img,matrix,(w,h))
    return imgWarp
```

Figure 3.55 Image warping function

To capture the key 4 vertex points of the rectangle shape road, two functions are used with the help of the getTrackbarPos(), which is Function in Python OpenCV that returns the current position of the specified trackbar. The function takes two arguments. The first is for the trackbar name and the second one is the window name which is the parent of the trackbar. It returns the trackbar position.

```
29  def initializeTrackbars(intialTracbarVals, wT=480, hT=240):
30      cv2.namedWindow("Trackbars")
31      cv2.resizeWindow("Trackbars", 360, 240)
32      cv2.createTrackbar("Width Top", "Trackbars", intialTracbarVals[0], wT // 2, nothing)
33      cv2.createTrackbar("Height Top", "Trackbars", intialTracbarVals[1], hT, nothing)
34      cv2.createTrackbar("Width Bottom", "Trackbars", intialTracbarVals[2], wT // 2, nothing)
35      cv2.createTrackbar("Height Bottom", "Trackbars", intialTracbarVals[3], hT, nothing)
36
37
38  def valTrackbars(wT=480, hT=240):
39      widthTop = cv2.getTrackbarPos("Width Top", "Trackbars")
40      heightTop = cv2.getTrackbarPos("Height Top", "Trackbars")
41      widthBottom = cv2.getTrackbarPos("Width Bottom", "Trackbars")
42      heightBottom = cv2.getTrackbarPos("Height Bottom", "Trackbars")
43      points = np.float32([(widthTop, heightTop), (wT - widthTop, heightTop),
44                           (widthBottom, heightBottom), (wT - widthBottom, heightBottom)])
45      return points
```

Figure 3.56 Trackbar functions used to detect the current contour points

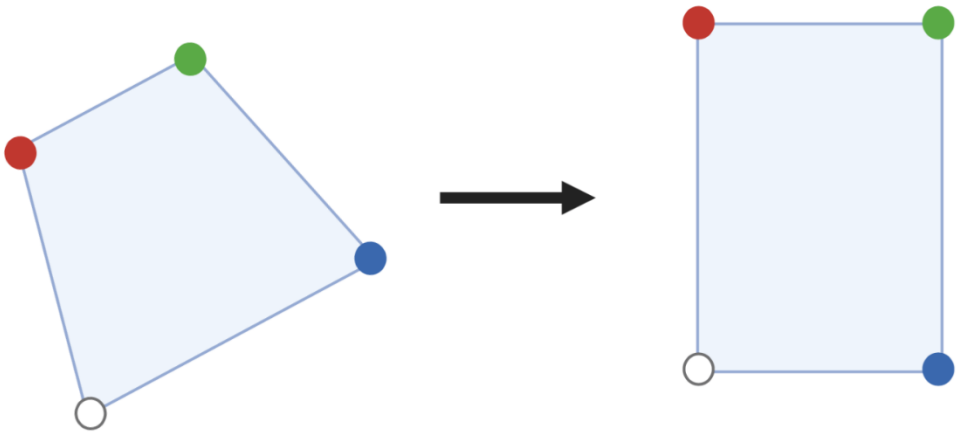


Figure 3.57 Warping depiction

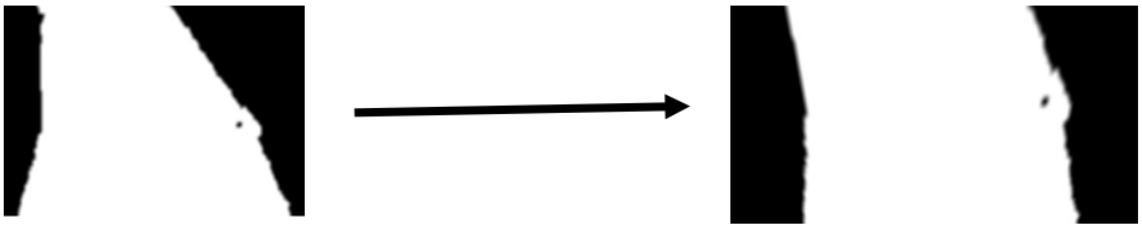


Figure 3.58 Warping applied to lane

The next step after getting the bird eye perspective is to apply the pixel summation method to understand where the curve is going.

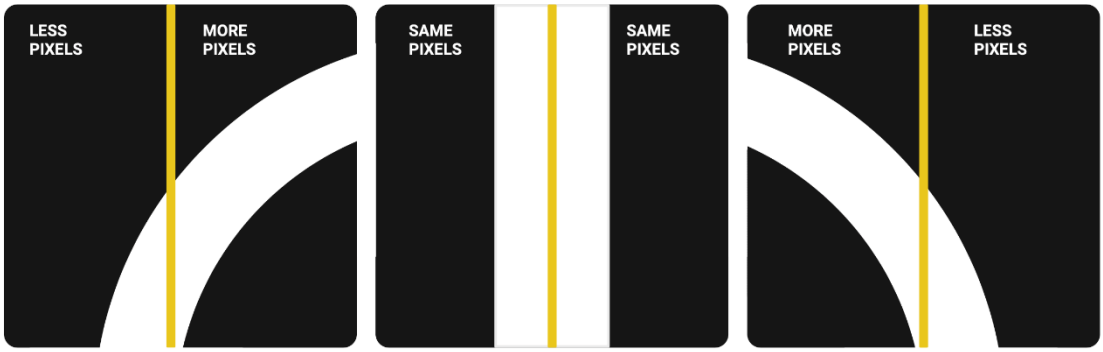



Figure 3.59 Motor logic with camera control in lane

As explained before, the number of white pixels is added to a total number and depending on which side of the camera has more value, the robot will turn to said direction.

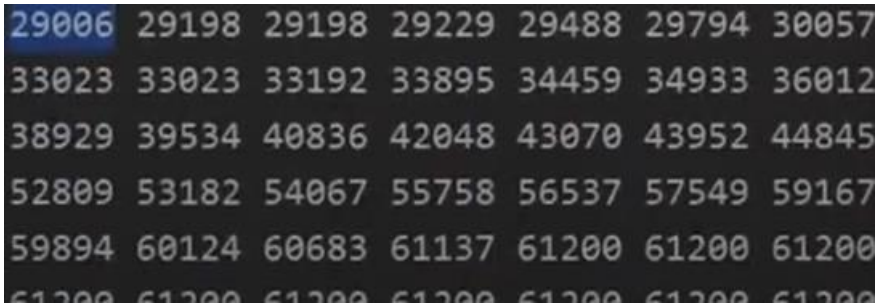



Figure 3.60 Addition of arrays of bits

When the addition is translated into the commands 'go right' or 'go left' or 'stay in the centre', the routine to follow a line is created at least.

The only problem regarding the project and the solution is the capacity of processing power of the Raspberry pi 4 of 2GHz. It was found that, as well as other aspects of the project, the small processing power limits the usability of the camera to implement the desired line tracking.

This was compared to the much stronger version of the SBC, the RPI4 8GHz, which appeared to run smoothly. This confirmed the viability of the code, but the costs increased almost by 40% of the whole robot prototyping process, remaining a heavy liability, so it was decided to implement a much simpler solution.

## 3.7 Used line routine if CPU is not Optimal

As explained, the CPU couldn't withstand the optimal processing power needed to apply the visual recognition of the lane so instead of overusing the camera a solution is implemented with analogic infra-red sensors.
The IR sensor is an electronic sensor that measures the infrared light radiation from objects in its field of view, it can detect or measure the infrared radiation or the changes in said field from an outer source or inbuilt source. It is widely used to keep track of the changes in the environment.

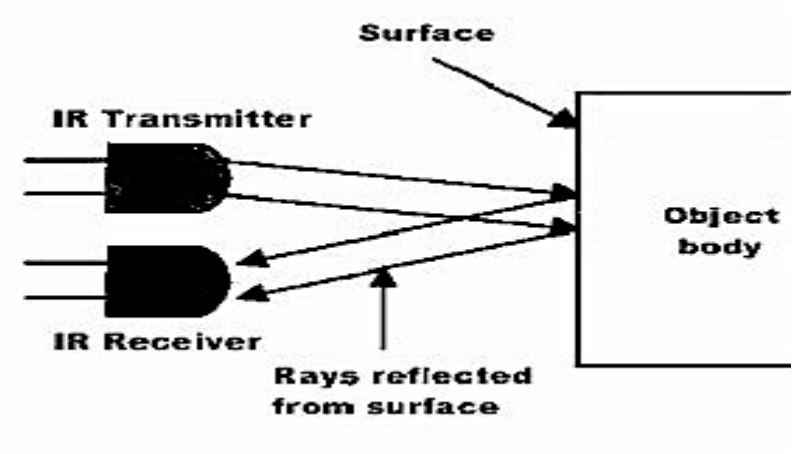



Figure 3.61 IR working on body

The IR is crucial for the robot to be able to keep track of a line without using much CPU in the process which is perfect for the current situation. The IR LED transmitter can transmit the light and the photodiode (Receiver) waits for the transmission to come back and it will only return if

its reflected by a surface. The key aspect of the implementation resides in the characteristic that only the white colour surface can completely reflect the IR transmission whereas the black colour surface will completely omit the light.

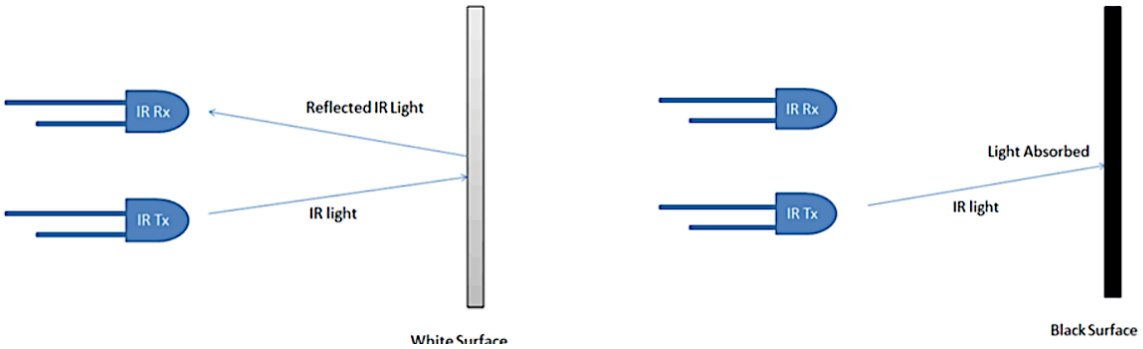



Figure 3.62 IR working on color

Two IR sensors are used to check if the robot is on the correct track with the line and these are connected directly to the functions representing the motors behaviour. The sensors are placed in one on either side of the line and if they detect the black line, they send information of action to the motor driver modules.

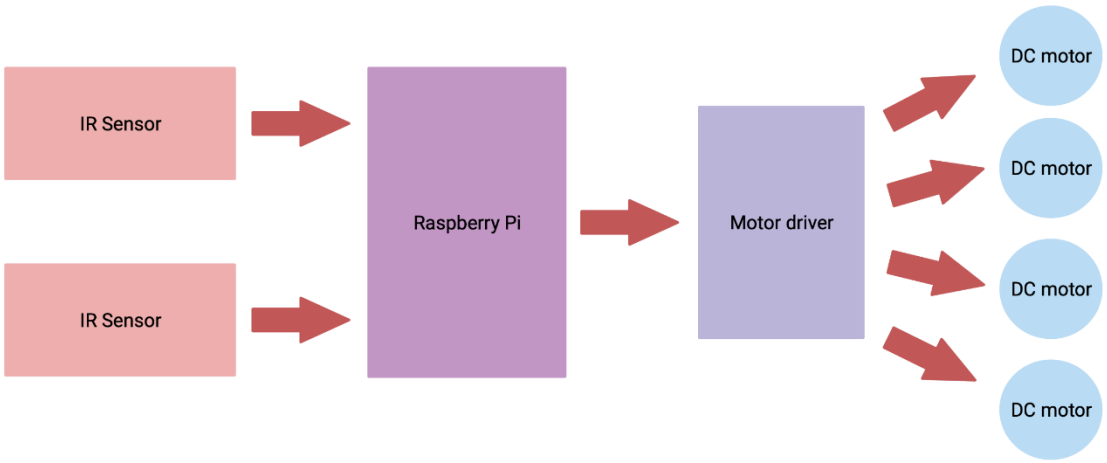



Figure 3.63 Programming sequence

For the forward movement the two sensors won't detect the black lane meaning its on the right path.

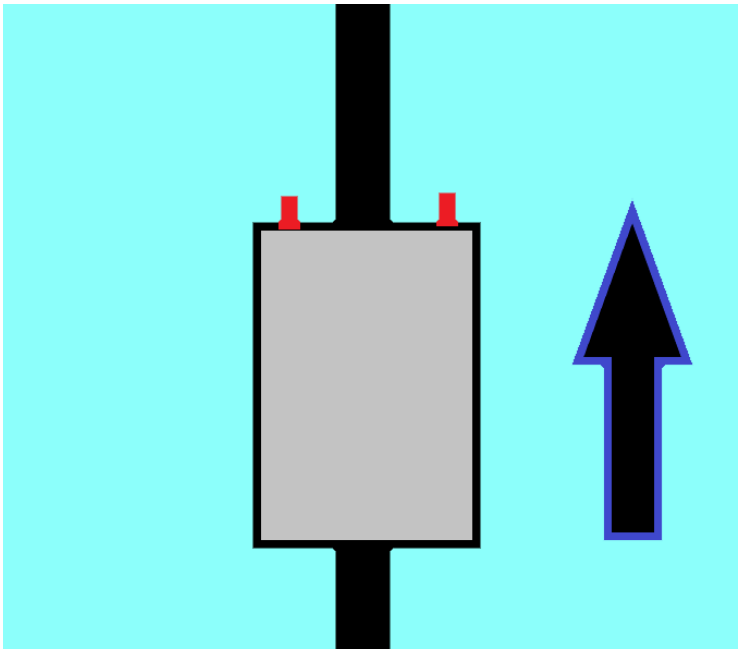


Figure 3.64 Robot in forward position

If the left sensor touches the black line, the robot will turn to the left and if the right sensor touches, it will turn to the right. Considering that the camera is also being used for identification of signals, the stop function won't depend on the IR sensors but on the signal recognition of the Stop signal received on the camera.

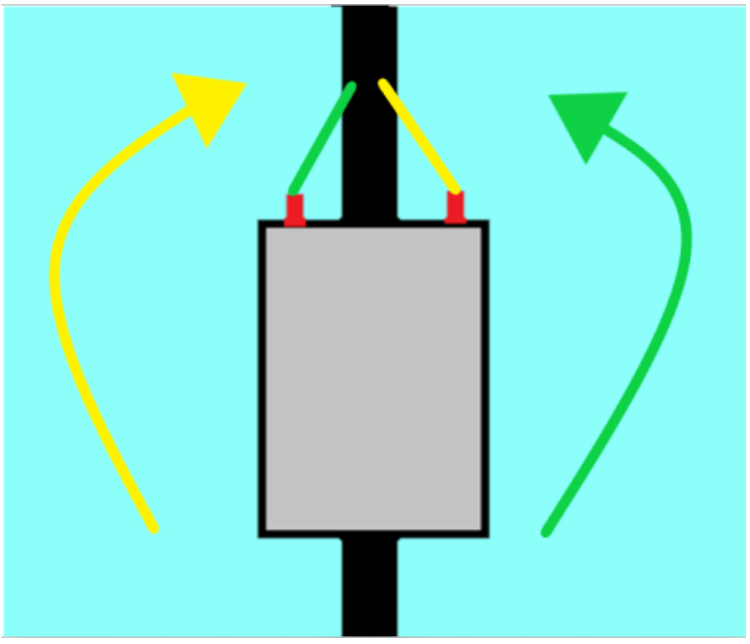


Figure 3.65 Robot with respective routines of turning

Now that the basic understanding of how it should work, the code applied is very simple in comparison to the visual recognition of the lane due to the processed information is analogue and direct instead of a digital cascade of images.

First, the sensors act as inputs from the point of view of the RPI4 so the GPIO PINS connected to the outputs of the sensor must be configured as inputs, in this case the PINS 25 and 26 are used.

```
import RPi.GPIO
# Setup of the sensors

GPIO.setmode (GPIO.BCM)
GPIO.setup(25,GPIO.IN) #GPIO 25 -> Left IR out
GPIO.setup(26,GPIO.IN) #GPIO 26 -> Right IR out
```

Now to make it cooperative with the main functions of movement the iterations of condition are implemented:

```
if(GPIO.input(25)==True and GPIO.input(26)==True): #both white move forward
        forward_movement()
elif(GPIO.input(25)==True and GPIO.input(26)==False): #turn left
        diagonal_left_up()
elif(GPIO.input(25)==False and GPIO.input(26)==True): #turn right
        diagonal_right_high()
```

Now the cool magic touch happens when the combination of sensors and camera comes to play, if this code is applied to the TensorFlow object recognition it is possible to assign objects as signals placed on the visual field of the road.

Once the code is applied to the main function of lane following, the robot will follow the lane until a signal is seen in which case an action will be performed.

For the first time the robot was supposed to recognize a circle, triangle, and rectangle but to help with the current software and to not overload the RPI4 with code to process 3 types of different objects were selected.

These 3 items are a stop sign, a keyboard, and a mouse for the current system of 2GHz RPI. It was found that with this system the robot worked better but with an upgrade in SBC a recognition of any shape would be possible in combination with the written software or if the object recognition model from the machine learning testing changed but due to, again, CPU usage it is not viable.

With this said, the only changes needed to apply to the current OD() function would be the following:

```
If object_name == “stop sign”:
        Stop_movement()  #previously programmed with motor module
        Sleep(5)
        continue

Elseif object_name == “keyboard”:
        right_horizontal()  #previously programmed with motor module
        sleep(5)
        left_horizontal()   #previously programmed with motor module
        continue

Elseif object_name == “mouse”:
        Left_horizontal()  #previously programmed with motor module
        Sleep(5)
        Right_horizontal()  #previously programmed with motor module
        continue
```

In the end, achieving the recognition of these objects while tracking the lane with the sensors result in the same desirable outcomes as with the shapes. In this case if applied, the bottle will stop the car for 5 seconds before returning to move, the keyboard will make the robot move horizontally to the right as in the user wants the robot to move to the side to be able to load

some weight and the same case for the mouse but here it will move to the left before moving to the right back at the track and following with the procedure.

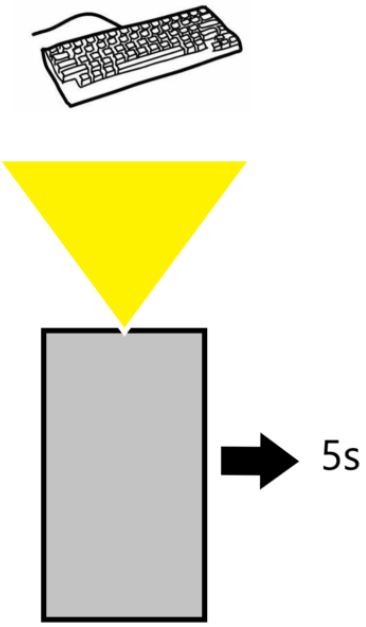



Figure 3.66 Detection of keyboard with movement

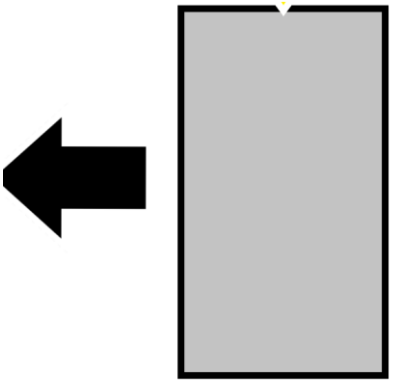


Figure 3.67 Movement return

Once the software was properly implemented some tests were applied to calibrate the results. The tests started with the generic stop signal used for day-to-day driving signaling.


Figure 3.68 Stop signal used

When calling the function name of Lane_following() the camera is activated and the IR sensors are validated to the PINS.

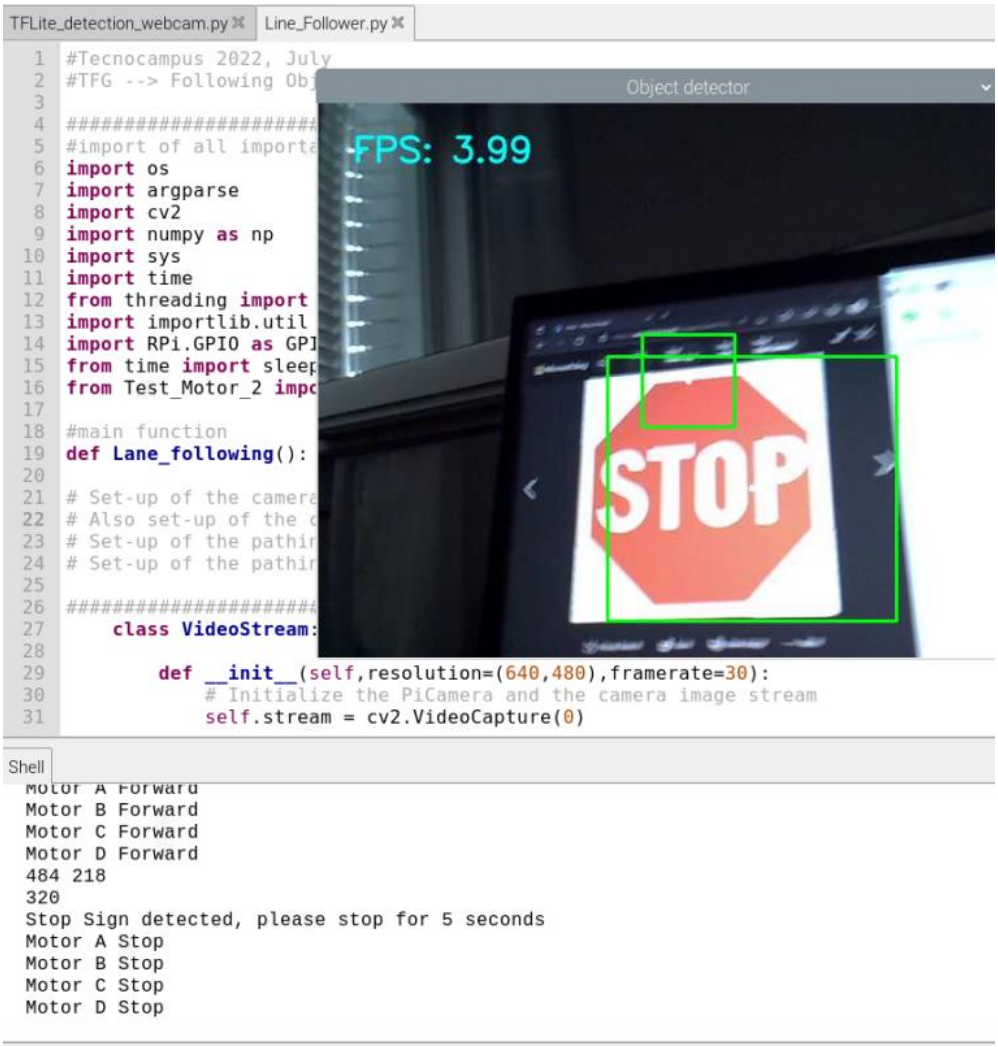



Figure 3.69 First Encounter with STOP signal

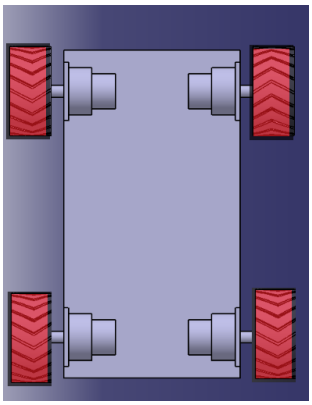


*Figure 3.70 Stop Function*

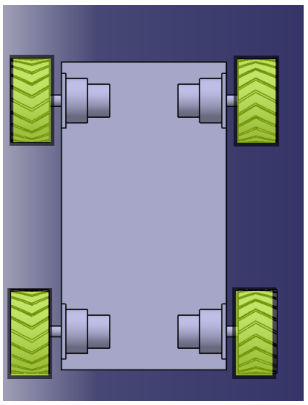


*Figure 3.71 Forward Function (yellow indicates clockwise movement)*

After stopping the 4 outputs of current to the 4 motors it waits a predeterminate 5 seconds before actioning the motors again.

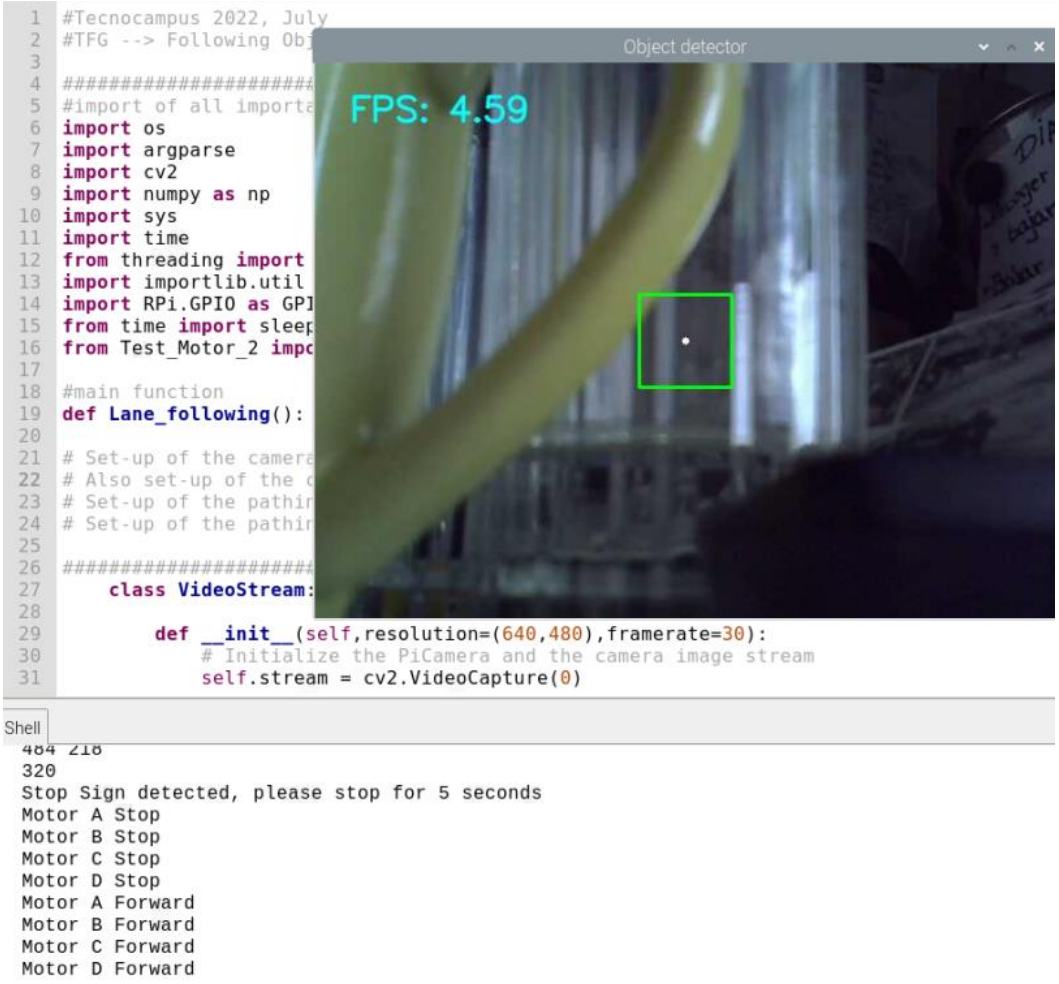



Figure 3.72 5 seconds after obtaining the STOP signal

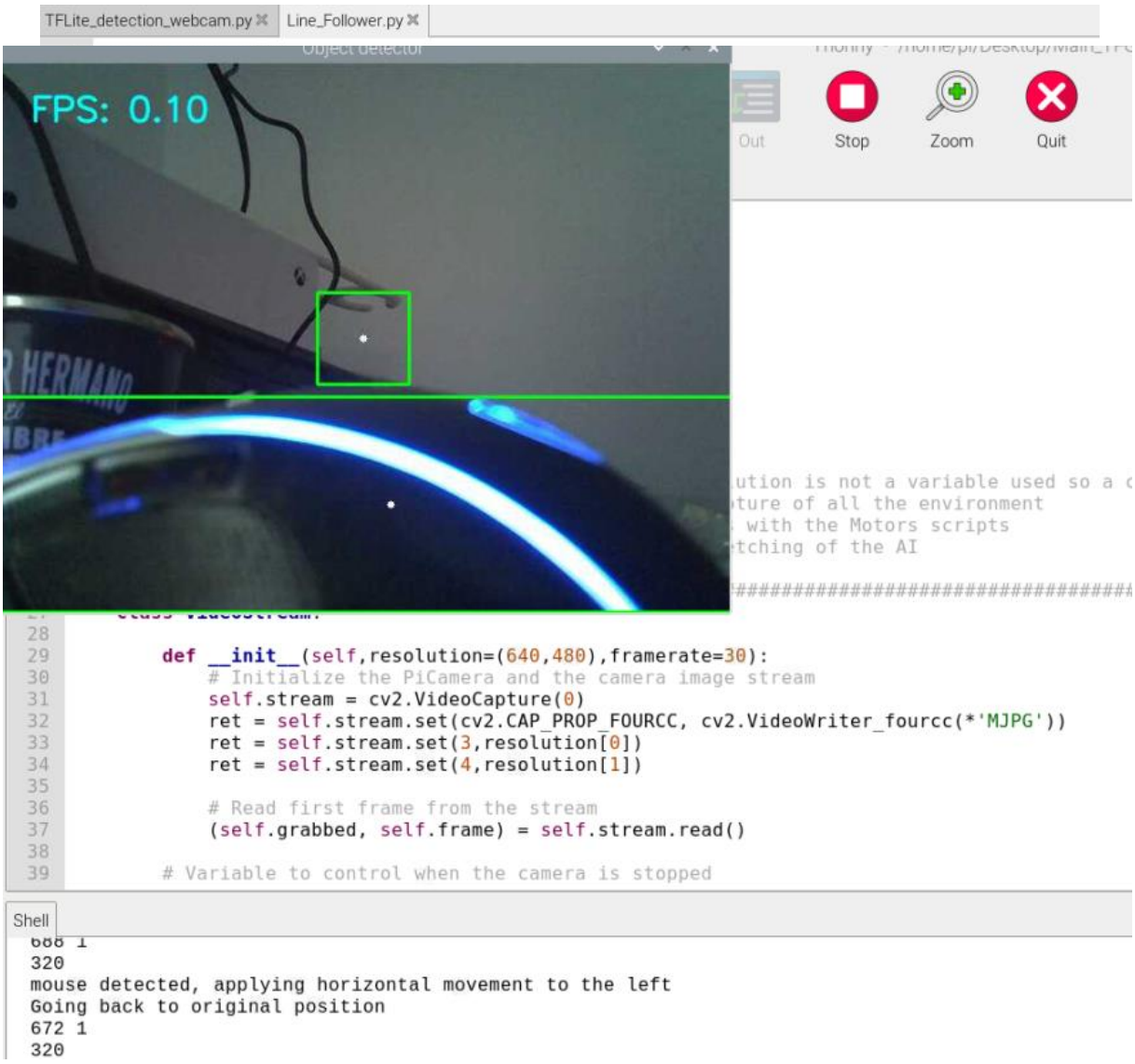



Figure 3.73 Identification of the mouse

Now for the keyboard testing occurs the desired result as well, when it is detected, the motors activate by the usage of the horizontal right function.

To finish with the signals utilized a test on the mouse was initialized and the result is the same as the keyboard but to the left.

# 4. Results of prototyping and Conclusions

To give an end to the development of the product, the first prototyping structural build is presented in a way that is easily applied to any chassis selected for the construction of the robot while taking into consideration the key hardware parts.

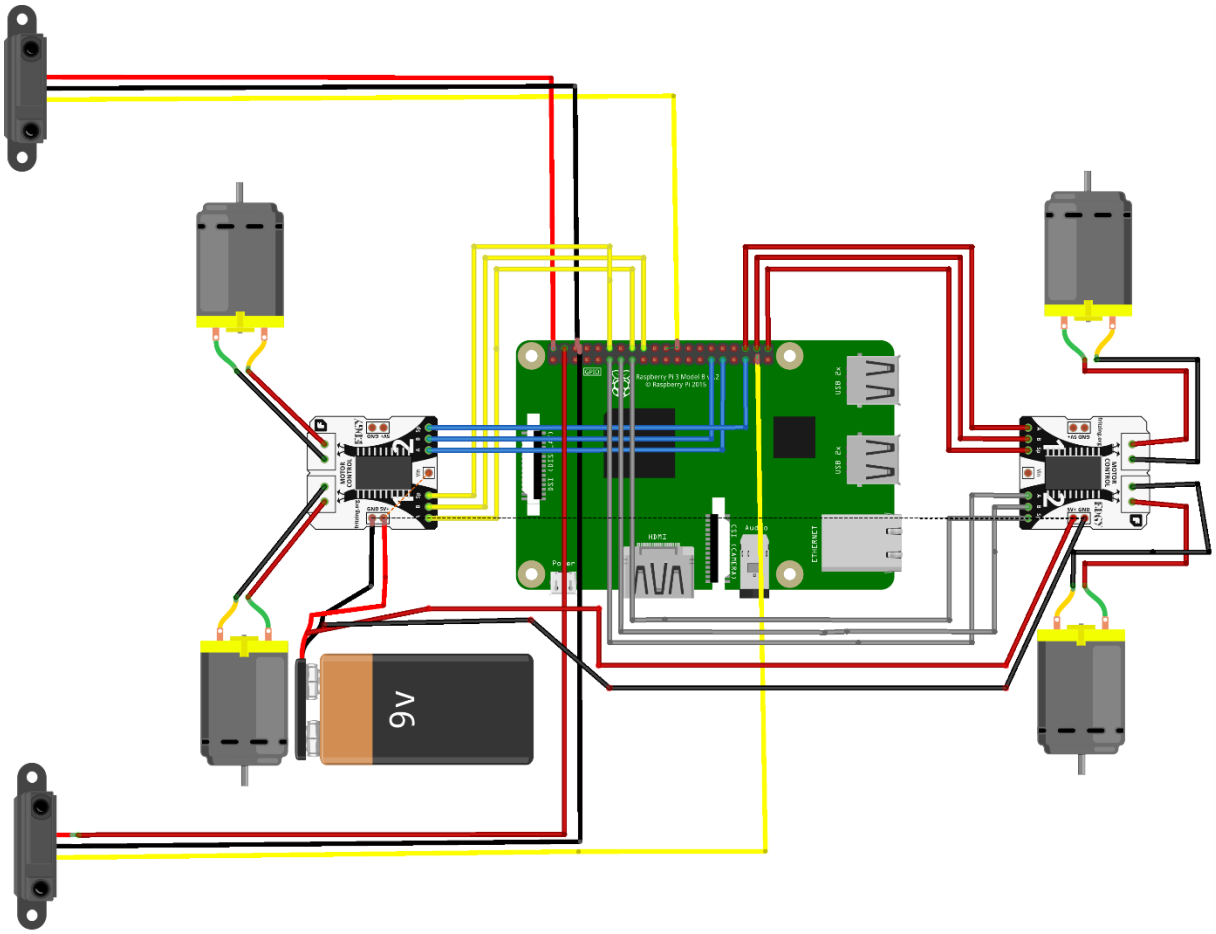



Figure 4.1 Complete connection scheme

Figure 4.1 shows a battery of 9V, which could be replaced by a 12V one in case more motor power transmission is desired. All the connections shown are in line with the current programming and the programmed pins. An electrical scheme was developed to improve further bettering of the robot.

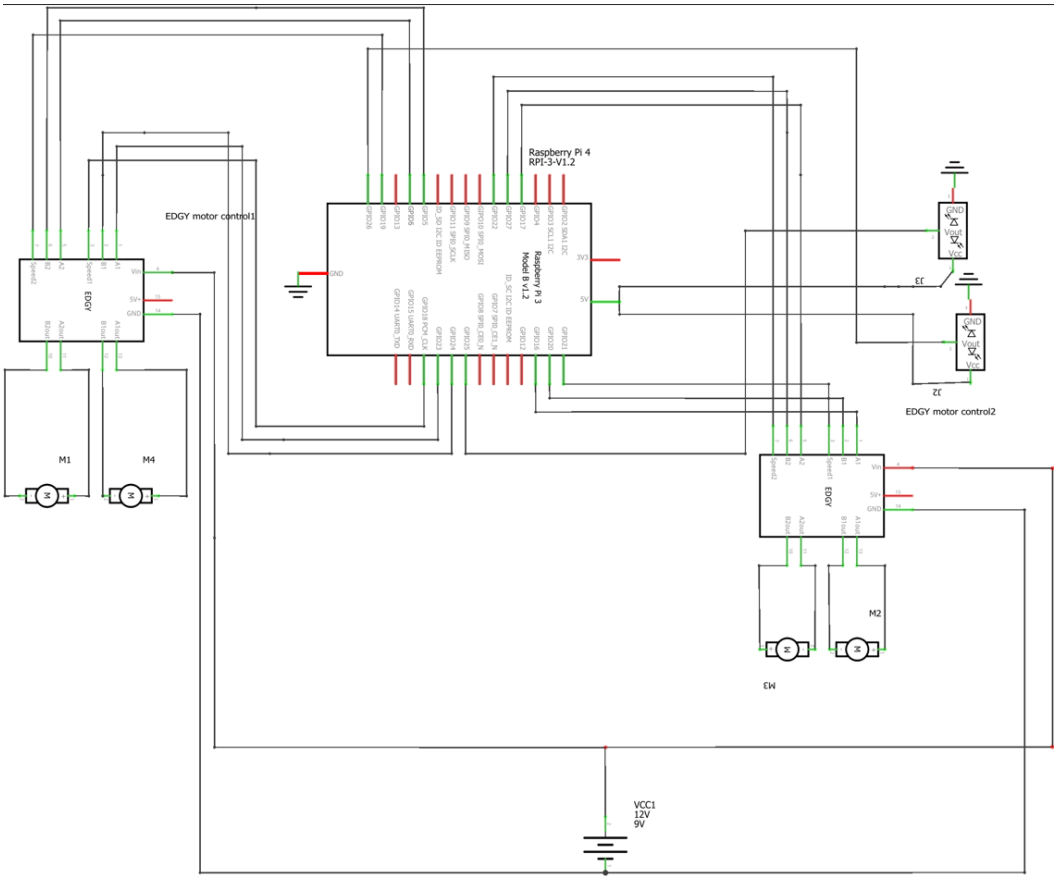



Figure 4.2 Electrical Scheme

As said before, the Battery applied is 9V but could be swapped by a 12V one. After testing the electrical connexions, the correct working of the system was confirmed, and this made possible the proposition of developing a PCB with the key components to facilitate the overall production of the final product. The result of the PCB is not optimized and was not tested, but a first prototype is introduced for further improvement, the paths that cross with another path are on different sides of the board it cannot be appreciated due to the same colour being used.

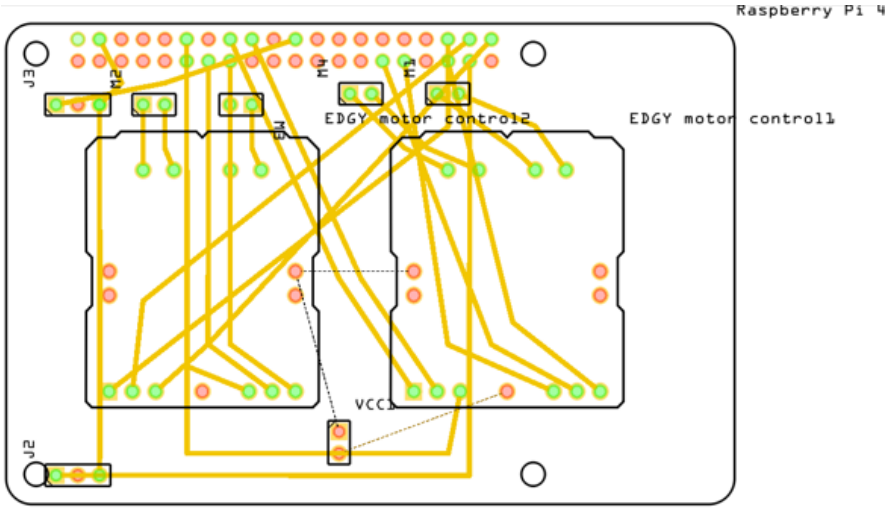



Figure 4.3 PCB first prototype without optimal pathing

The PCB consists of the hardware soldered onto the top of the SBC to make it more compact.

A final problem occurred with the final usage of the chassis for the robot, and it was not possible to have the fully functional robot for the writing of the documentation report, but the functionality regarding the sensors, the program and the actuators were successfully tested without the main chassis implementation.

This meant that with the shown wiring with the software embedded into the RPI4 and the selected items for its construction, the robot could fit any chassis with a reasonable weight in which the only important consideration would be the correct selection and application of 6 to 7.2V motors to cooperate with the software routines.

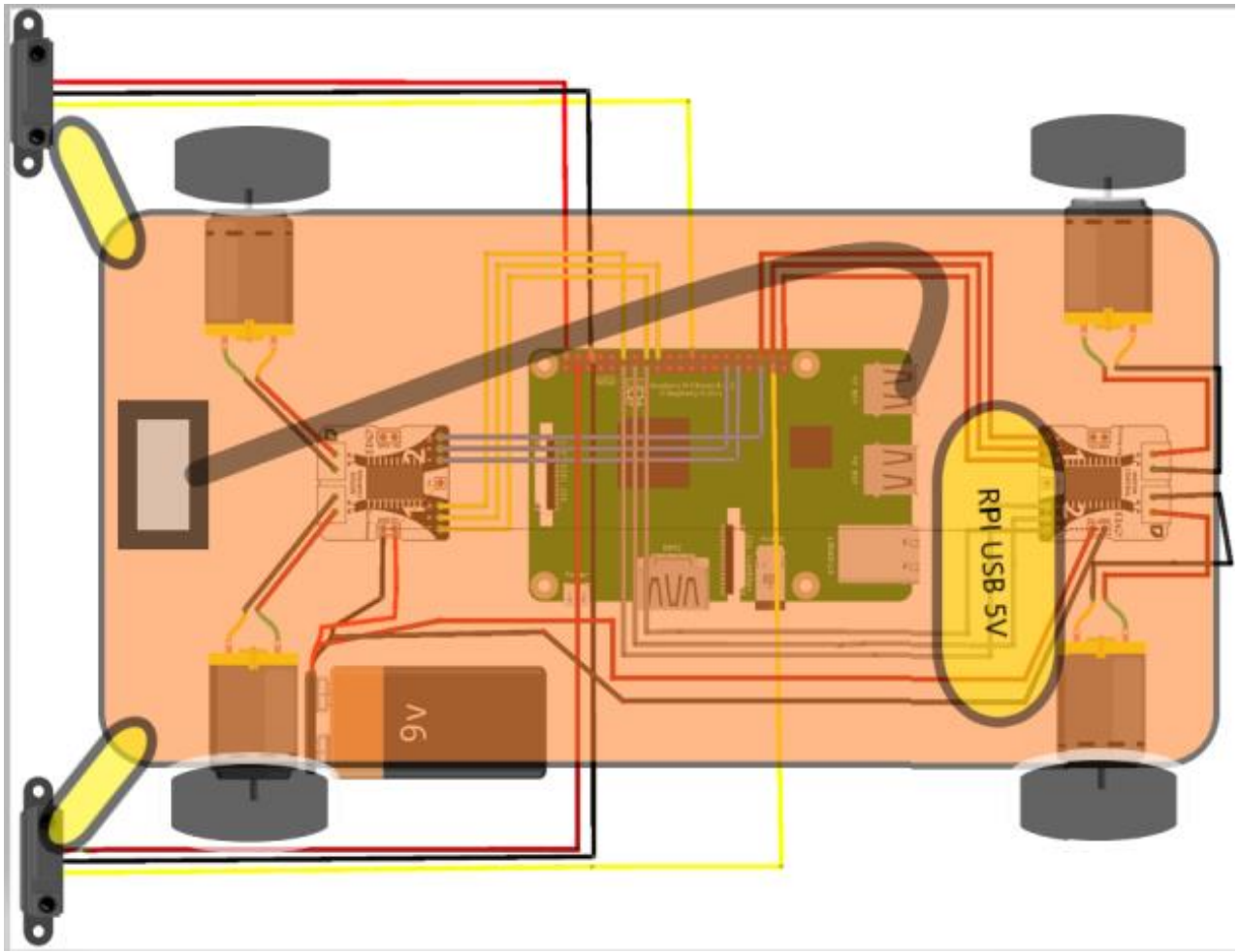



Figure 4.1 Final look of connections

When operating with the Lane following routine the following behaviour happens in a predetermined circuit put together by the user:

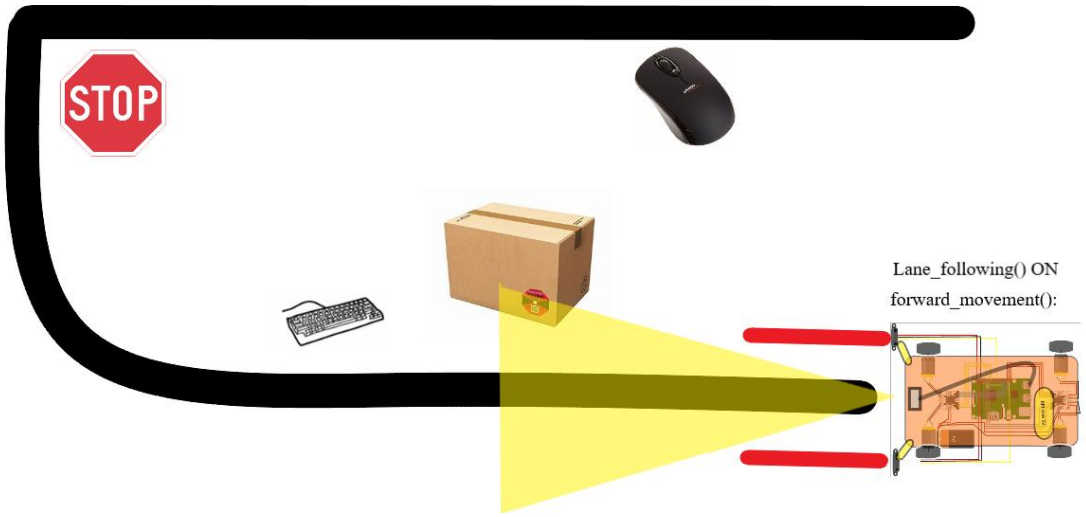



Figure 4.2 Start of the circuit

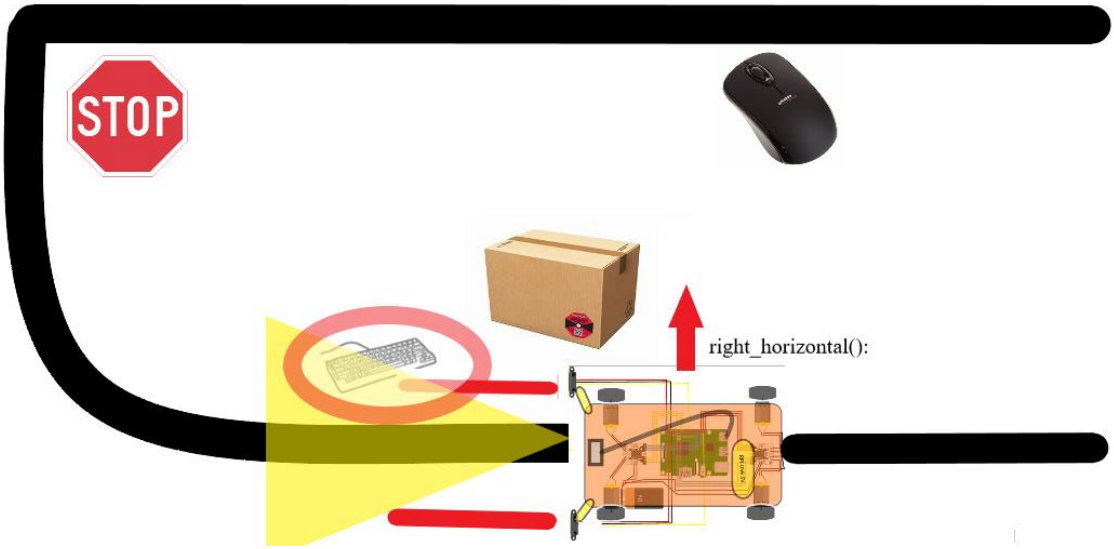



Figure 4.3 Detection of object assigned to horizontal movement to the right

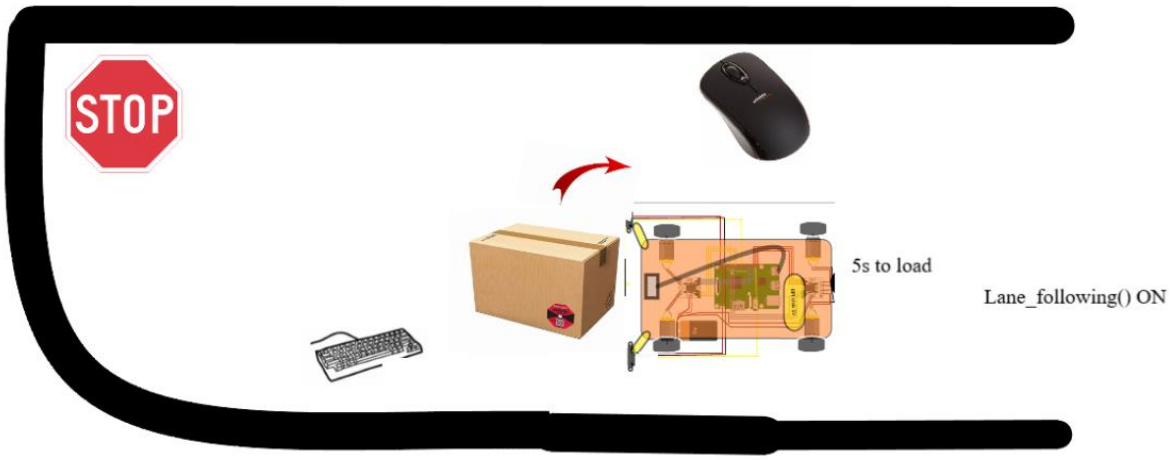



Figure 4.4 Loading of package once positioned


Figure 4.5 Curve assimilation and correction to the right


Figure 4.6 Stop sign interpreted and motors stopped


Figure 4.7 Right turn activated by IR sensors

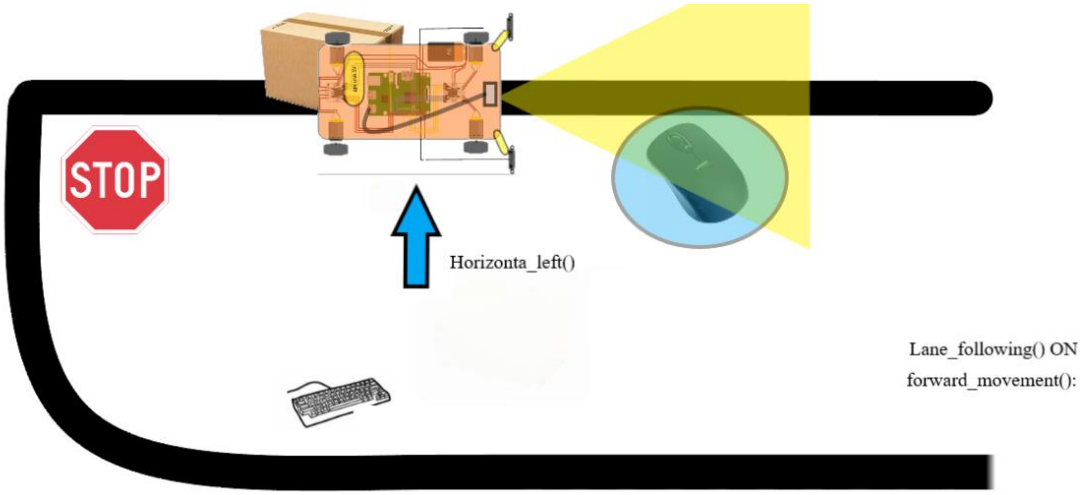



Figure 4.8 Horizontal left movement activated by interpretation of object

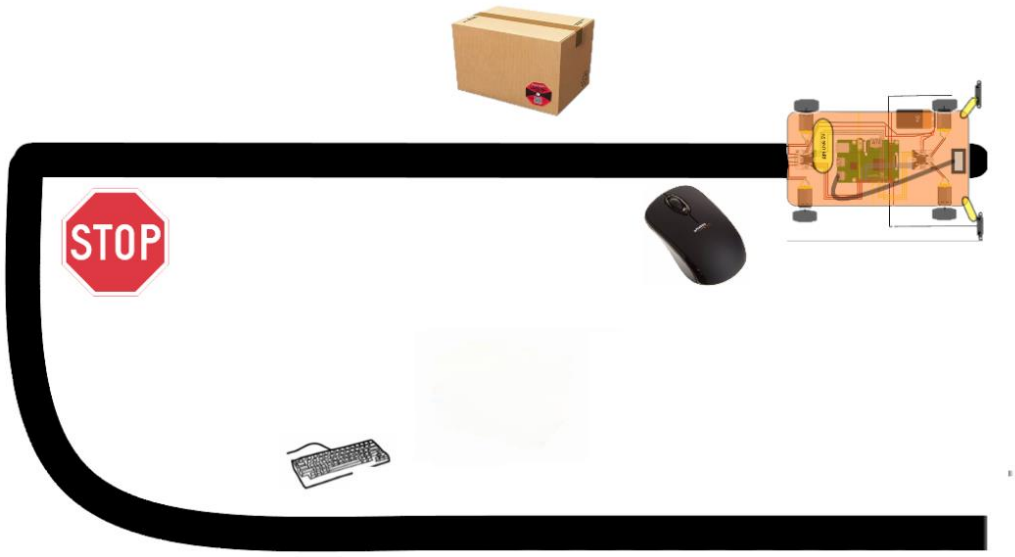



Figure 4.9 End of line

Now if the OD() function is activated the result is the following of an object that first the program asks what item we want to follow from a list given to the user with the 89 possibilities. It allows the user to be able to have a moving backpack behind to carry packages or house chore items such as clothes.

A developing of a robot that helped with the logistics of carrying loads with the possibility of following a beacon or by the usage of a track with different signals with each having a purpose was the initial idea. Was the final result acceptable?

The results can confirm the consistency of the initial proposal through the whole development of the overall functional prototype. An optimal result was achieved with some downsides from the original plan like not having the fully functional robot with its chassis attached for the documentation process, the overheating of some components that damaged the speed of development or the not planned overuse of CPU in the SBC. However, these difficulties were studied, and the better solutions appeared out of range for the initial budgeting which were acknowledged and documented in a way that with the incrementation of the investment a perfect prototype would be easy to produce.

The initial objectives of developing a track system and a following system have been tackled with the software working to perfection. Also, the implementation of the actuators to the software was also a success like the ability to make the camera cooperate with the motor drivers and the IR sensors with the SBC at the same time which was challenging at first.

A personal reflexion shows that even thou a lot of problems appeared during the process, a new milestone to solve them was always implemented and that is what made this project a good engineering experience, if there is a problem find a solution or an alternative to reach the final goal with the documentation available.

This project made usage of concepts from 12 subjects studied in the degree such as digital electronics, instrumentation, analogue electronics I & II, Microprocessors, robotics, control engineering, electrotechnology, English, circuit theory, computer science, object physics and concepts picked during the Erasmus programme like the consideration of the PCB’s or the PID implementations.

It was great going from having zero knowledge on computer vision models, the raspberry inner workings and the many python libraries used, to being able to perform at an optimal level if put in a situation with said items involved.

# 5. References


[1] Abualkibash, M., ElSayed, A., & Mahmood, A. (2013, January 1). *Highly Scalable, Parallel and Distributed Adaboost Algorithm Using Light Weight Threads and Web Services on A Network of Multi-Core Machines (*Research Gate. Retrieved April 16, 2022, from https://www.researchgate.net/figure/Summation-of-all-pixels-on-top-and-to-the-left-of-x-y-is-the-integral-image-value-at-x-y_fig1_237054341

[2] Electronics HUB. (2018, February 24). *IR sensor information*. Retrieved April 14, 2022, from https://www.electronicshub.org/interfacing-ir-sensor-with-raspberry-pi/#:~:text=The%20IR%20Sensor%20Module%20has,used%20a%20simple%205V%20Buzzer.

[3] Fritzing. (n.d.). *Schemes Software*. Retrieved June 5, 2022, from https://fritzing.org/

[4] GitHub. (n.d.). *Integration of code*. Retrieved February 2, 2022, from https://github.com/

[5] Hassan, M. (n.d.). *Computer Vision Training*. CV Zone. Retrieved March 25, 2022, from https://www.computervision.zone/

[6] Krishna, S. (2020, April 2). *Initial wheel prototype*. STL Finder. Retrieved March 8, 2022, from https://www.stlfinder.com/model/mecanum-wheels-JT0Che44/2929837/

[7] Logitech. (n.d.). *Camera Datasheet Documentation*. Retrieved February 1, 2022, from https://www.logitech.com/fr-ch/products/webcams/c270-hd-webcam.html

[8] Microsoft. (n.d.). *Coding Environment used in the document*. Visual Studio Code. Retrieved January 1, 2022, from https://code.visualstudio.com/

[9] Open CV. (n.d.). *Object Detection software*. Retrieved January 5, 2022, from https://opencv.org/

[10] Python. (n.d.). *Python Documentation*. Retrieved February 5, 2022, from https://www.python.org/

[11] Raspberry. (n.d.). *Raspberry PI 4 OS*. Raspbian. Retrieved January 1, 2022, from https://www.raspbian.org/

[12] RS Components. (n.d.). *DC motor information*. Retrieved March 3, 2022, from https://ie.rs-online.com/web/generalDisplay.html?id=ideas-and-advice/dc-motors-guide#:~:text=DC%20motors%20take%20electrical%20power,fixed%20within%20the%20output%20shaft.

[13] TensorFlow. (n.d.). *TensorFlow AI Models for Object Detection*. TensorFlow.Com. Retrieved March 3, 2022, from https://www.tensorflow.org/resources/learn-ml?gclid=EAIaIQobChMIrtDh0oyq-AIVoRkGAB1h0ggUEAAYASAAEgIlQPD_BwE