CSE489: Machine Vision

Project: Vision – Controlled Car

SUBMITTED BY:

Abdelaziz Asaad Abdelaziz 14p8118

Shorouq Mohamed Ali Mahmoud Arafa 14p2006

Senior 2

TABLE OF CONTENTS

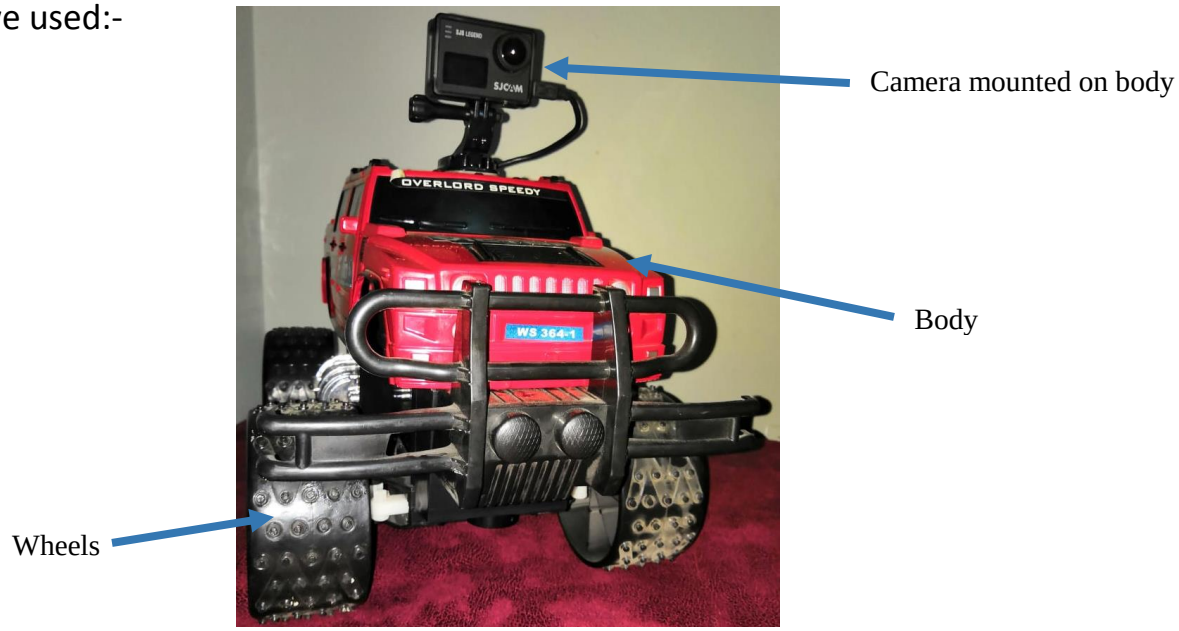
INTRODUCTION

Autonomous mode in road vehicles has gained popularity in the last few decades. In order to provide highly automated driver assistance systems, different types and combinations of sensors have been presented. However, most of these approaches apply quite sophisticated and expensive sensors, and hence, the development of a cost-efficient solution remains a challenging problem.

In this project, we used a camera to control the body of a toy RC car. Non of the remote controlling features were used, just the body of the car and its wheels. The vision input from the camera is processed using image processing techniques which determine the mode of the car, and sends signals accordingly to the motors. The modes include the car moving forward, backward, right or left, depending solely on the live input video through the camera. The following sections in this report will demonstrate the steps of assembling such a car and its components, the communication between the components and how they are powered, as well as the code implemented which enables the car to be controlled via the camera.

CAR COMPONENTS

In order to control a car, we need to have the body and wheels first. The approach we took was to have our efforts mostly focused on the computer vision code required for the project, hence we chose to use a ready-made RC car body to ease the mechanical design phase and enable us to quickly begin implementing and testing our codes. Below is the RC car body we used:-

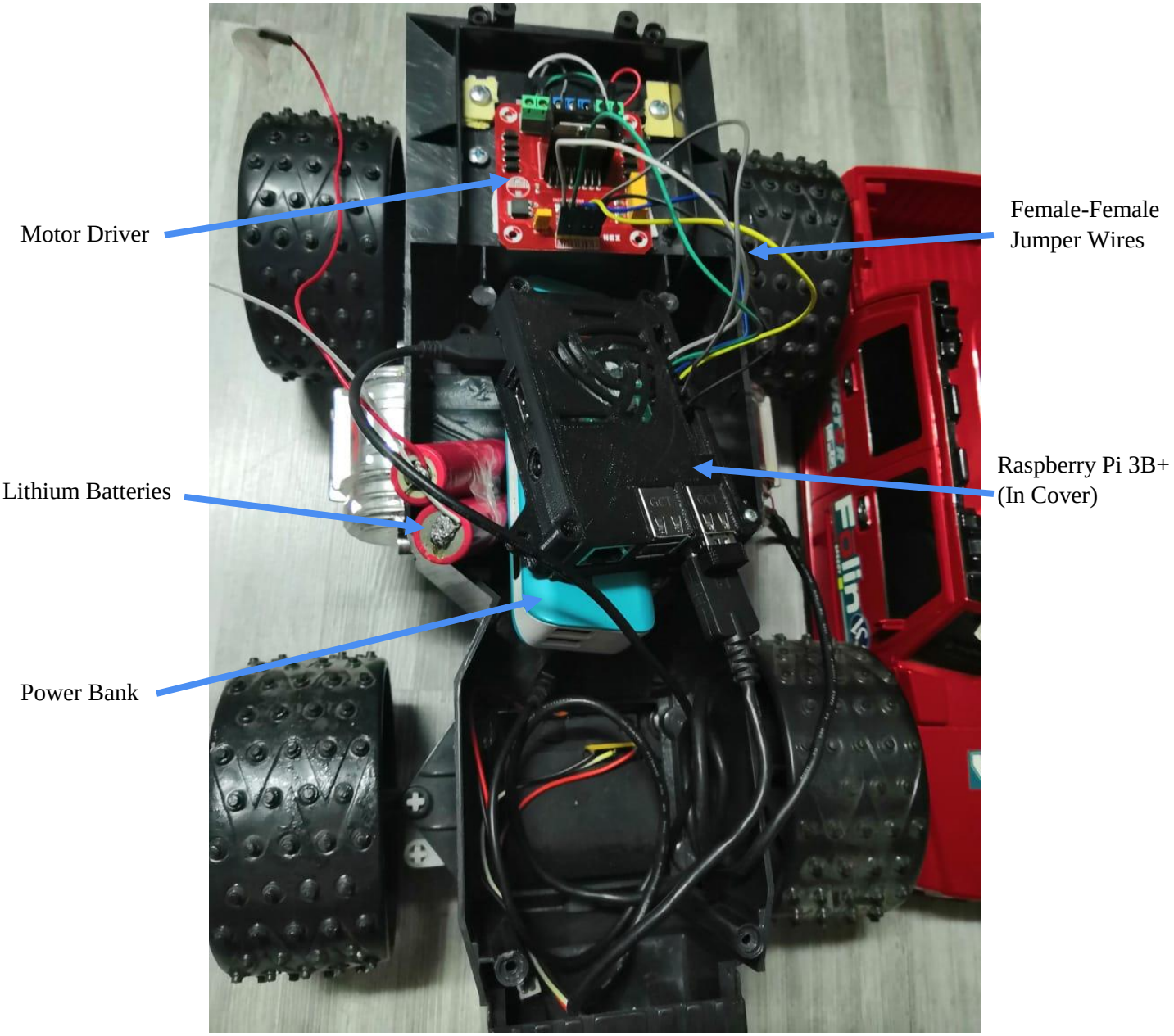


RC Car Body

1. Electrical Components

The electrical components that were included in the RC car were not suitable due to them being vague in manufacturing, hence some components were removed and replaced with components we could control using our available microprocessors and powering systems.

Below is an image of the RC car's chassis, where most the electrical components were fixated. The components will be explained in more detail in the section following the image.



This is another view of the car to shop how the DC motors are placed in the car:-

DC Motor (left rear wheel)

Note: The previous image is not how the components are arranged within the car, we just separated the components from under each other for us to be able to show them and point them out. In reality they are placed on top of each other in an ordered manner to fit under the car's body.

Below is a list of the electrical components that were added to the car:-

1) **Mini DC Gearbox Motors – Straight Shaft (2x)**

Specifications:-

- Voltage: 4- 6 V DC (The supply voltage we used is 5V)
- No Load Speed: 90 +/- 10rpm
- No Load Current: 190mA (max.250mA)
- Minimum Torque: 800 gm.cm

Placed as a differential robot, both at rear wheels.

2) **SJCam S6 Legend Action Camera**

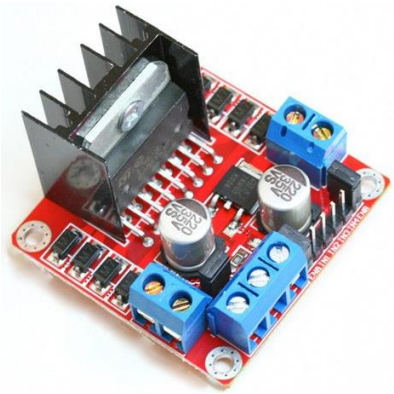
Specifications:-

- Ultra HD 4k Video Camera- 4K@24FPS 2K@30FPS Videos, 16MP high definition photo by 166 degree adjustable wide-angel and 2.5 aperture

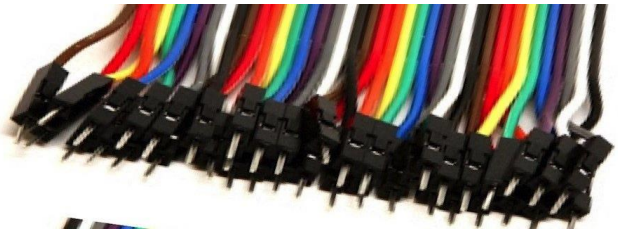
3) **L298 Module Red Board**

Specifications:-

- The L298 Stepper Controller makes it easy to drive two DC motors.
- Double H bridge drive
- Chip L298N (ST NEW)
- Logical voltage 5V
- Drive voltage 5V-35V
- Logic current 0mA-36mA
- Drive current 2A per channel (MAX single bridge)
- Max power 25W
- Weight 30g



4) **Jumper Wires for Connections (Female-Female and Male-Female)**



Batteries

The powering of the components is very important because we need to allow the car to move around without being tied up by wires to a computer, for example, as a source of power. Therefore, we looked for the systems that needed powering, the voltages and current ratings required and used appropriate batteries and power supplies for each.

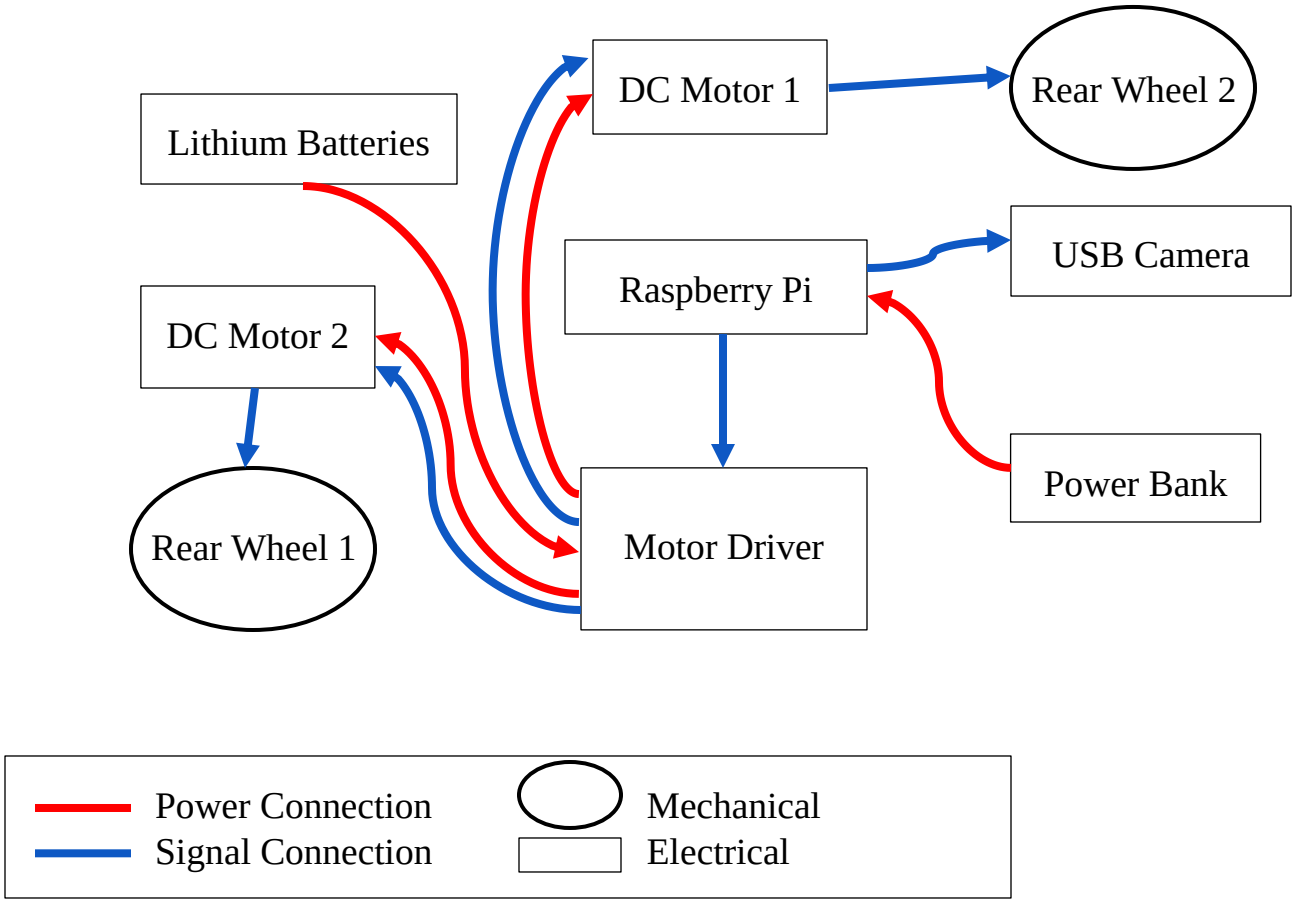
a) Microprocessor: Raspberry Pi

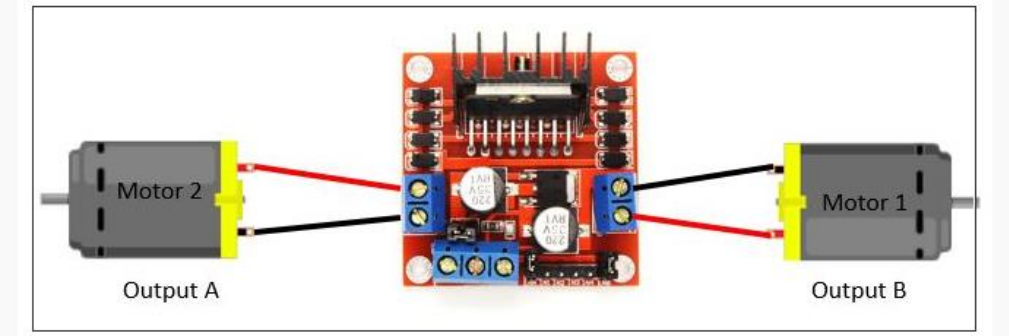
As for the microprocessor, the recommended input voltage is 5V, and the recommended input current is 2A. We used a *power bank* which supplied a voltage of 5V and current of 2.1A to the Raspberry Pi.

b) Motors

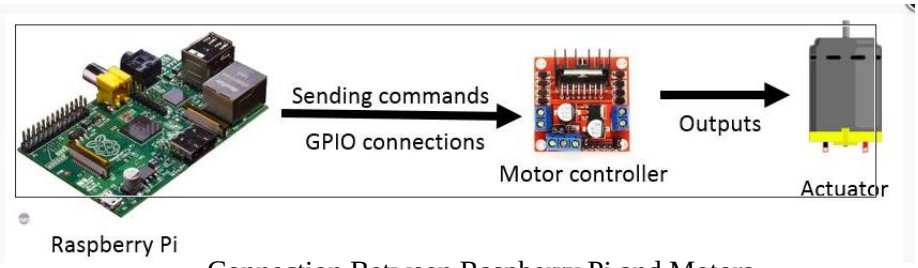
The rest of the components were powered up using 2 Cells (connected in parallel) of 8.4 volt lithium batteries, with a current supply of 3A.

Connections Between Components





Connection Between Drivers and Motors



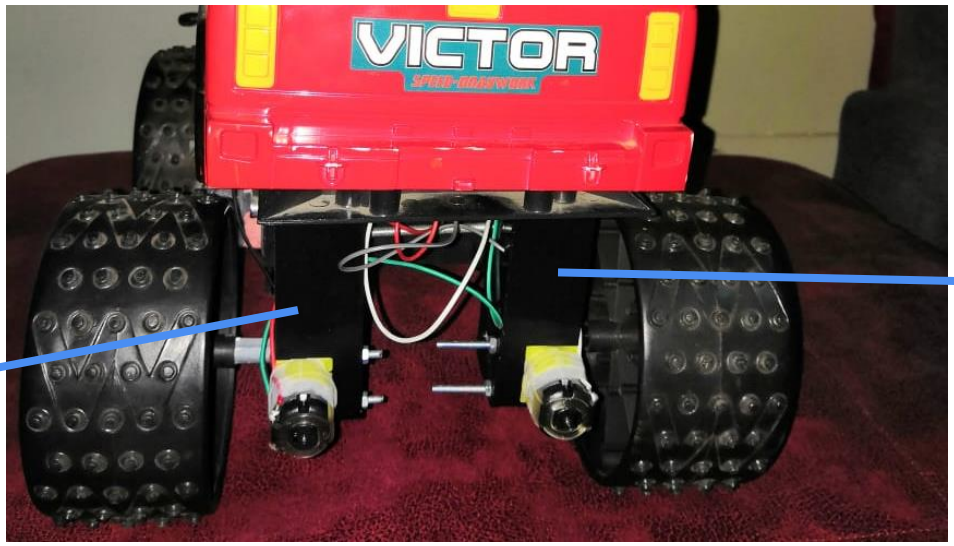
Connection Between Raspberry Pi and Motors

2. Mechanical Components

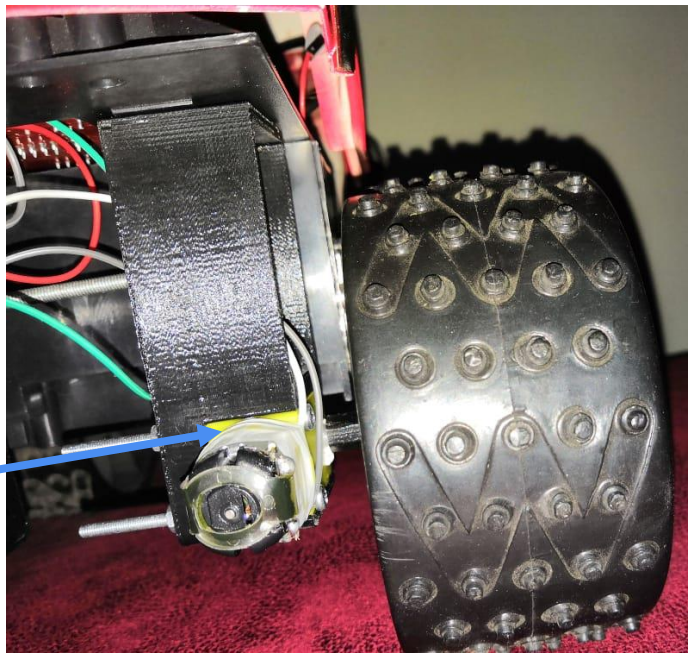
The RC Car was not ready for the electrical components that needed to be added for the vision control, hence some mechanical supports were used to fixate some of the electrical components.

The first electrical components that needed support was the DC motors at the wheels. The motors that originally came with the car were not suitable for our method of control, hence were removed and replaced with another 2 DC motors, one for each rear wheel.

The supports were designed using Inventor CAD software and 3D printed. The image below shows how the supports were used to fixate the motors:-



Motor Supported in Fixation to Body



The second electrical component that needed fixation was the camera. The camera's position and vision area is crucial in order to obtain good results, therefore, to guarantee the position of the camera at all times when the car is moving, it needed a strong enough support, as the one shown below:-

Camera Support

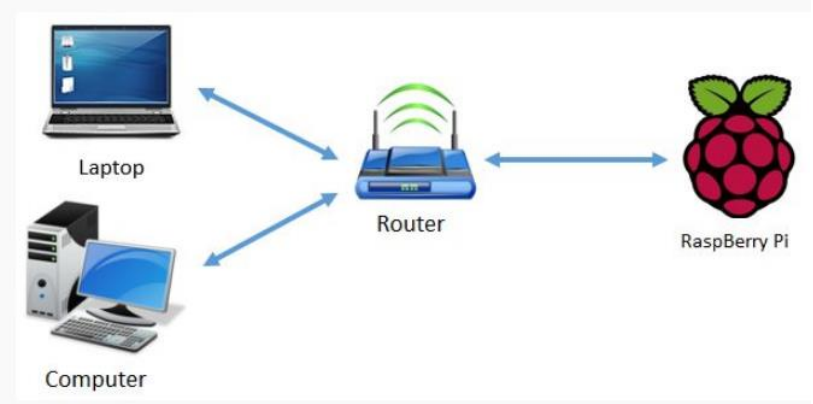
Hole to let wire through

A small opening was made in the ceiling of the car's body to allow the wire of the camera to go through.

3. Microprocessors

The microprocessor chosen to be used for this project is the Raspberry Pi 3B+, along with a 16GB SD card. The operating system on the Pi that was used is Raspbian. It is used often as a microprocessor to apply computer vision in any computer that could be small scale. The Raspberry Pi has an acceptable cost, small size and specs (clock, CPU, RAM, and other ETHERNET). The Raspberry Pi is capable of embarking peripherals such as USB ports and also the ability to integrate features such as actuators and sensors in a set called external links GPIO, including pins outputs / digital inputs, UART, I2C, SPI, audio, 3v, 5v and GND.

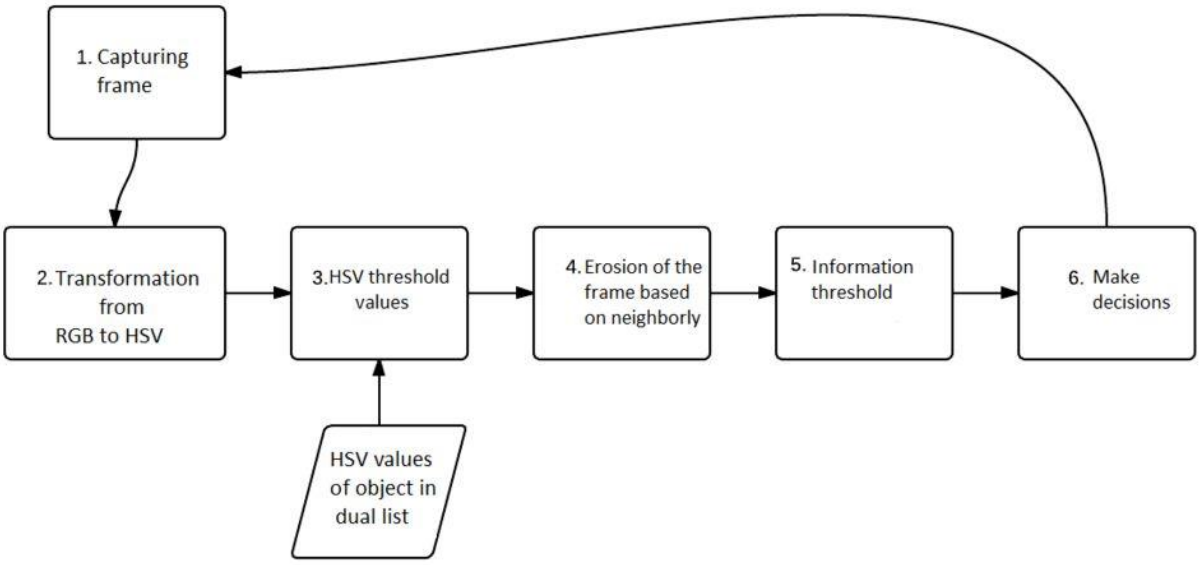
The way in which the Pi was used was via a wireless internet connection.



CONTROL & ALGORITHM

The OpenCV library is where all the processing operations of the video is done and the information extracted from this processing too, making decisions robotics platform and other components that enable the dynamic system. The version of OpenCV used is OpenCV-3.4.2, and the version of Python used is Python 2.7.

A computer vision system (CVS) was created and it functions as shown below:-



These functions are executed repeatedly and for real values of the dynamic characteristics of the object, up to six times a second time.

Here is how the algorithm, present in the appendix, works:-

1) First it was necessary to capture the image or the frame. The system default is RGB color, this color system is represented in the webcam frame obtained through the basic colors: red (Red), Green (Green) and blue (Blue).
2) After the captured image, the conversion from RGB color system to the color HSV (hue, saturation, and value) was undertaken, this model is similar to the recognition by the human eye colors. Since the RGB (red, green and blue) system has the colors based on combinations of the primary colors (red, green and blue) and the HSV system defines colors as their color, sparkle and shine (hue, saturation, and value), facilitating the extraction of information. The conversion from RGB to HSV is using the "cvtColor" native OpenCV function, which converts the input image from an input color system to another function.
3) With the image in HSV model, it was necessary to find the correct values of HSV minimum and maximum color of the object that will be followed. To save these values, we made two vectors with minimal HSV and HSV maximum color object as values: minimum Hue (42) Minimum saturation (62) Minimum brightness (63) Maximum Hue (92) Maximum Saturation (255) Maximum Brightness (235). So the next step to generate a binary image, the relevant information may be limited only in the context of these values. These values are needed to limit the color pattern of the object. A function of comparing the pixel values with the standard values of the inserted vector was used. The result was a binary image providing only one value for each pixel.
4) Having made the segmentation, resulting in the binary image, it is noted that noise is still present in the frame. These noises are elements that hinder the segmentation (including obtaining the actual size) of the object. To fix this problem, it was necessary to apply a morphological transformation through operators in the frame, so that the pixels were removed that did not meet the desired standard. For this, the morphological operator EROSION, who performed a "clean" in the frame, reducing noise contained in it was used.
5) Then the "Moments" function was used, which calculates the moments of positive contour (white) using an integration of all pixels present in the contour. This feature is only possible in a frame already present and without noise, so that the size of the contour of the object is not changed by stray pixels in the frame, which hinder and cause redundancy in information.

It was necessary to find the area of the contour and its location coordinates in the frame to be made the calculations of repositioning the chassis. The calculation of the area of the object performs the binary sum of positive, generating the variable M00 and recorded in the variable "area".

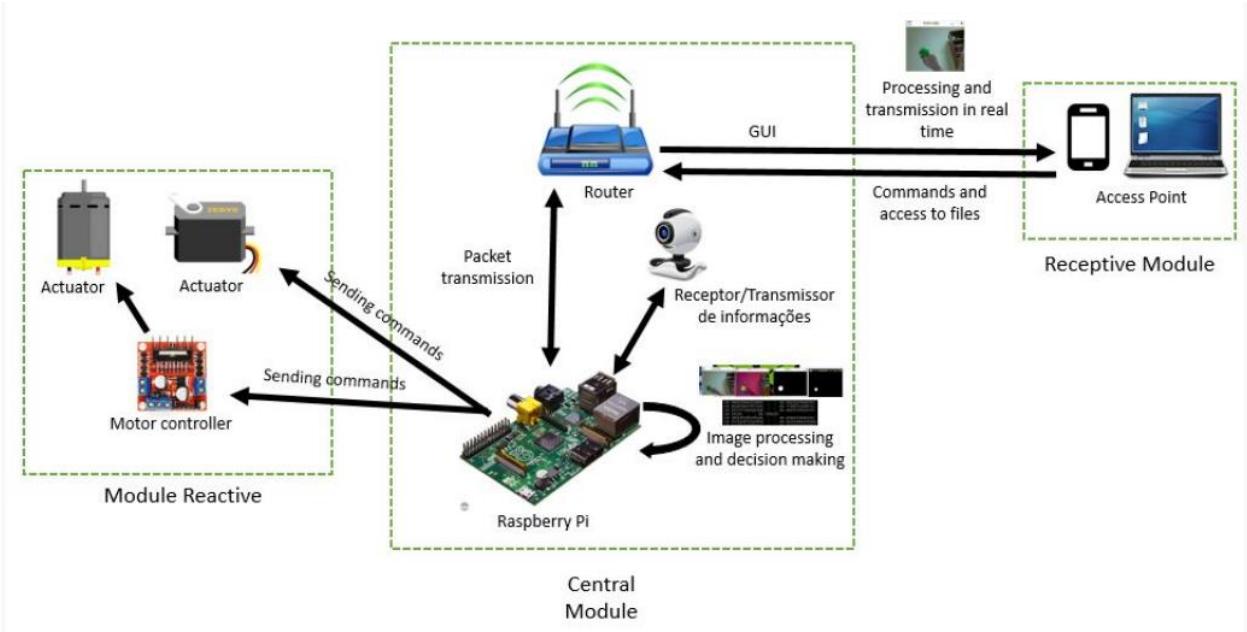
The specificity of the contour refers to an object, not a polygon. This value is found an approximate area of positive pixels (white) that make up the object. If this value is null area, is disregarded the existence of an object color treated (if the "green" color) in the frame. Using this feature will help accomplish the movement of the robot approaching and distancing of the target object, trying to treat the problem of depth. That is, the fact that the object is approaching or distancing from chassis.

6) The values received in the coordinate (x, y) refers to the placement of the found segmentation of the object relative to the frame and to facilitate the interpretation of the information which is being drawn from the coordinate information, a function that draws a circle at the centroid was applied.

Control

With DC motors, the chassis is made to move forward, backward, left and right. The horizontal & depth adjustment movement of the coordinate system is done by assuming binary-encoded digital pulses per pin only two values (0 and 1) performed by the control module, the H-bridge through the input pin. These pins are connected by jumpers and connected via GPIO on RPI. The GPIO library had to be installed on the Pi to enable us to work with its GPIO's.

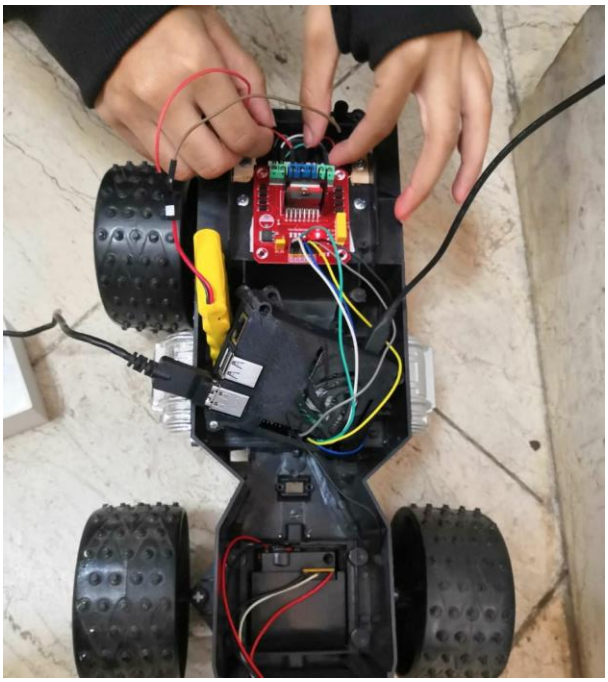
The general communication and control between components is illustrated below:-

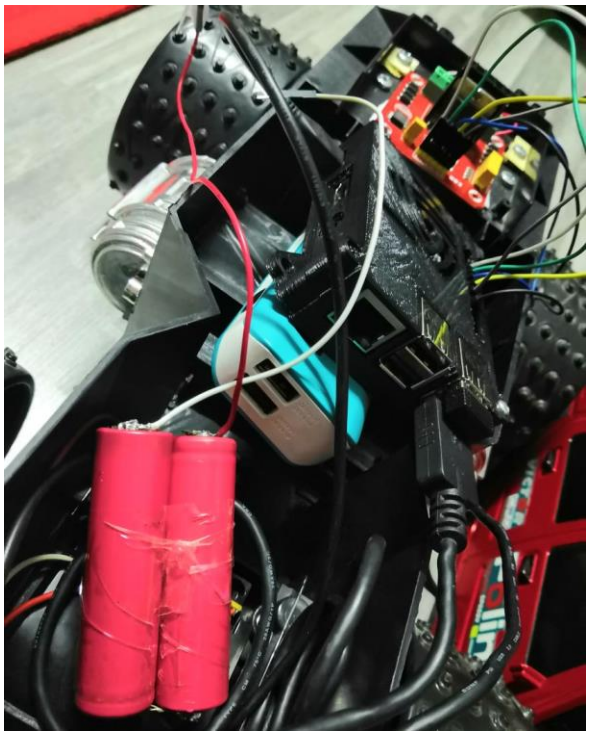


OUTPUT, CHALLENGES & NOTES

After assembling the car and its components, as well as downloading the necessary libraries, such as GPIO and numpy, and building the code, we were ready to try out the code on the assembled car.

Below are some images of procedure of testing and assembly:-



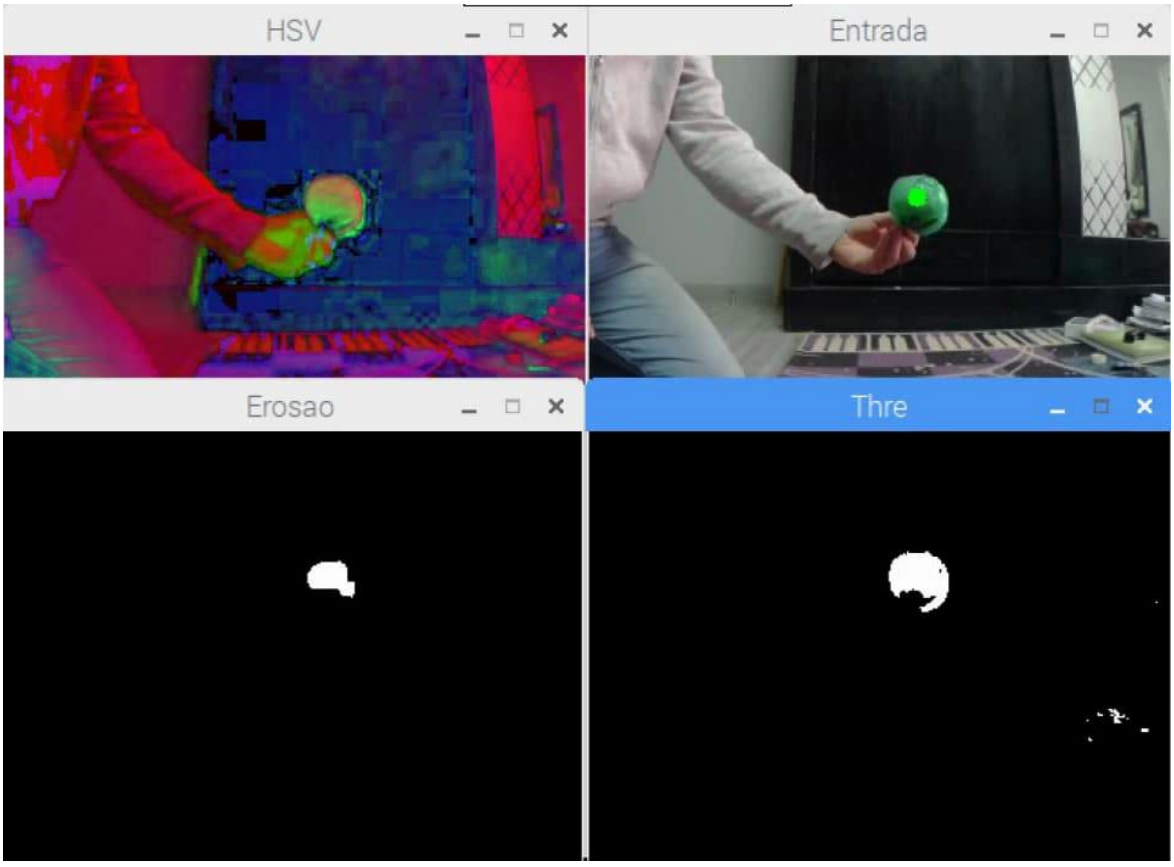


The code worked well after some tweaks were made to fix some lines, and the car successfully moved forward when a green object moved away from it, and moved further backwards when a green object moved closer to it. However, the right and left functions didn't appear to make and right and left movements to the car.

We later discovered that the motors could not actuate such a heavy car in the right and left modes, and it only moved the car forward and backward. To verify that the code was correct and that the right and left functions were sending the correct demands, we lifted the car from the ground while altering it via the camera to move right/left. The wheel s successfully moved right and left, but stopped when placed on the floor again. Hence, this was an issue purely related to the mechanical/power stage of the car, and that the code was going to work perfectly on a car with a lighter weight. Unfortunately, due to the approaching deadline we continued with the RC car we had instead of buying a new lighter one and beginning the assembly all over again.

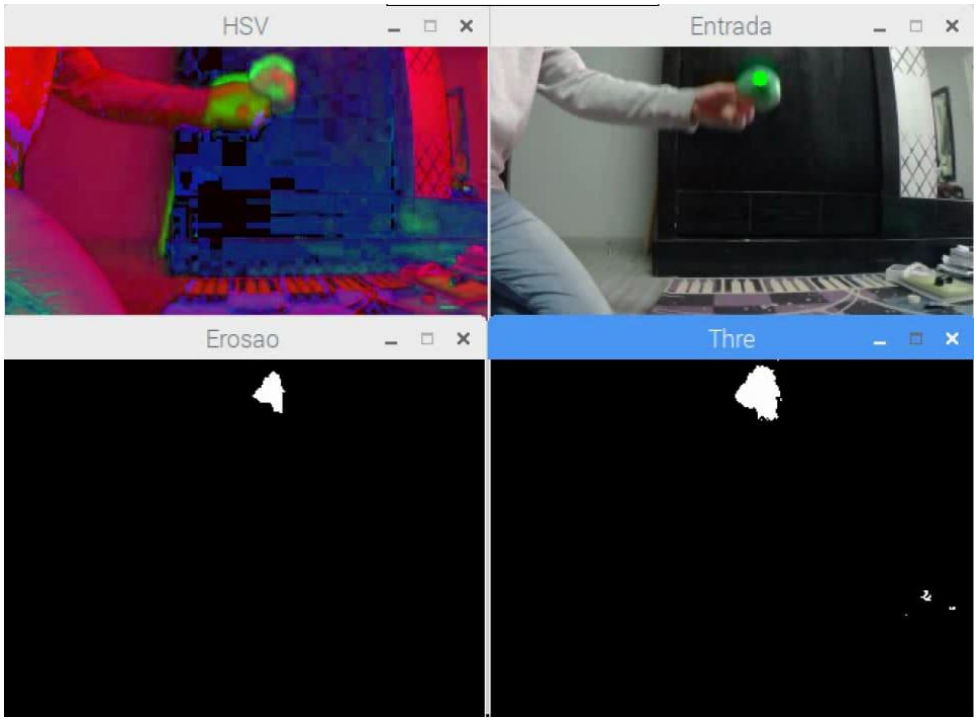
Hence the note we wanted to include here was about the video, as it will show how the wheels respond well to the right\left modes when lifted off the ground, but the forward and backward motion will be visible and moving on the ground perfectly.

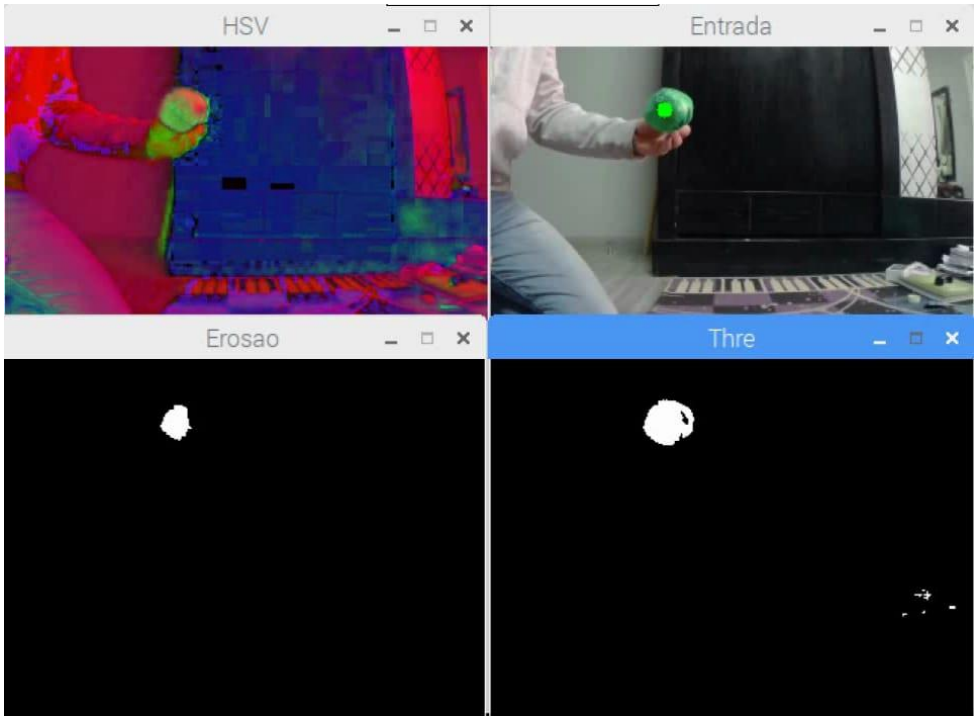
Below are some images obtained after running the code. These images demonstrate how each level of the code gives an output according to the input from the video.



A green ball was moved towards and away from the vehicle's camera. The results of this is shown above. The top left image shows the HSV function output, the top right image shows the final dot drawn in the centre of the green area spotted by the camera, and the bottom images show how the green ball was separated into a white area which will be used for coordinates of the car (done using thresholding) and the rest of the input's colours were changed into black, the area that the car will omit.

Below are different outputs for different positions of the green ball while testing the code:





APPENDIX

Vision-Controlled Car Code

The following code is the code we used for the vision-controlled car in our project. It is divided up into 4 main parts, each doing a certain function:-

#---------- 1) GPIO Initialization Functions ----------

```
# Description: This algorithm describes the simple implementation of OpenCV using Raspberry Pi and USB Camera
# Functions: Digital Image -> HSV Transformation -> Binary Image -> Binary Erosion -> Find Area -> Find Coordinates
# Functions 2: Draw circle in the centroid (x, y) -> pin declaration -> declaration of motor movement functions
# Functions 3: perform depth movement robot (area)
# Libraries: OpenCV, Python, GPIO, NumPy, cv2
#-------------------------------------------------------------
import cv2 as cv
import cv2 as cv2
import time
import numpy as np
import RPi.GPIO as gpio

gpio.setmode(gpio.BOARD)
# Turn off alerts
gpio.setwarnings(False)
#--------------------------------------
#Declares pins as output GPIO - Motor A
#motor A activation pin on the RPi
gpio.setup(7, gpio.OUT)
#motor A activation pin on the RPi
gpio.setup(11, gpio.OUT)
#Start Pin 13 as output - Motor A
gpio.setup(13, gpio.OUT)
#Start Pin 15 as output - Motor A
gpio.setup(15, gpio.OUT)

#--------------------------------------
```

```
#Declares pins as output GPIO - Motor B

#motor B activation pin on the RPi
gpio.setup(26, gpio.OUT)

#motor B activation pin on the RPi
gpio.setup(16, gpio.OUT)

# Start Pin 18 as output - Motor B
gpio.setup(18, gpio.OUT)

#Start Pin 22 as output - Motor B
gpio.setup(22, gpio.OUT)
```

#---------- 2) Motor Driver Control Functions ----------

```
#--------------------------------------
# Allow the L298N to be controlled by the GPIO
#--------------------------------------
#Initial Values - True - Motor A Enabled
gpio.output(7, True) #Motor A - RPi 1
gpio.output(11, True) #Motor A - RPi 2
#--------------------------------------
#Initial Values - True - Motor B Enabled
gpio.output(26, True) #Motor B - RPi 1
gpio.output(16, True) #Motor B - RPi 2
#--------------------------------------
```

#---------- 3) Motor Movement Direction Functions ----------

```
def Front():
# Motor 1
 gpio.output(13, True)
 gpio.output(15, False)
# Motor 2
 gpio.output(18, False)
 gpio.output(22, True)

def Back():
# Motor 1
 gpio.output(13, False)
```

```
 gpio.output(15, True)
# Motor 2
 gpio.output(18, True)
 gpio.output(22, False)

def Stop():
# Motor 1
 gpio.output(18, False)
 gpio.output(22, False)
# Motor 2
 gpio.output(13, False)
 gpio.output(15, False)

def Right():
# Motor 1
 gpio.output(13, True)
 gpio.output(15, False)
# Motor 2
 gpio.output(18, True)
 gpio.output(22, False)

def Left():
# Motor 1
 gpio.output(13, False)
 gpio.output(15, True)
# Motor 2
 gpio.output(18, False)
 gpio.output(22, True)

#def Adjust(area):
 # if(area<=120):
  #    Right()
  #elif(area>=600):
   #   Left()
  #else:
   #   Stop()
```

#---------- 4) Image Processing Functions ----------

```
# USING FUNCTION INRANGE TO CHANGE RGB-HSV
# FOR THAT WE HAVE TO DEFINE THE LIMITS OF H, S AND VALUES

# HSV range we used to detect the colored object
# In this example, set to a green ball(green color in general)
Hmin = 42
Hmax = 92
Smin = 62
Smax = 255
Vmin = 63
Vmax = 235

#Default RED
#Hmin = 0
#Hmax = 179
#Smin = 131
#Smax = 255
#Vmin = 126
#Vmax = 255

# An array of HSV values is created (minimum and maximum)
rangeMin = np.array([Hmin, Smin, Vmin], np.uint8)
rangeMax = np.array([Hmax, Smax, Vmax], np.uint8)
# Image processing function
def processing (Input):
    imgMedian = cv2.medianBlur(Input,1)
    imgHSV = cv2.cvtColor(imgMedian,cv2.COLOR_BGR2HSV)
    imgThresh = cv2.inRange(imgHSV, rangeMin, rangeMax)
    imgErode = cv2.erode(imgThresh, None, iterations = 3)
    return imgErode
#-------------------------------------------------------------

cv.namedWindow("Input")
#cv.namedWindow("HSV")
#cv.namedWindow("Thre")
```

```
cv.namedWindow("Negactive")

capture = cv2.VideoCapture(0)

# Parameters of the size of the capture image
width = 160
height = 120

# Minimum area to be detected
minArea = 50 #close to 80 cm

#Center of axes
center = width/2

# Center Limit
may = width/3

# Define a size for the frames (discarding PyramidDown)
if capture.isOpened():
  capture.set(3, width)
  capture.set(4, height)

while True:
    ret, Input = capture.read()
    image_processed = processing (Input)
    moments = cv2.moments(image_processed, True)
    imgHSV = cv2.cvtColor(Input,cv2.COLOR_BGR2HSV)
    imgThresh = cv2.inRange(imgHSV, rangeMin, rangeMax)
    imgErode = cv2.erode(imgThresh, None, iterations = 3)
    #moments = cv2.moments(imgErode, True)
    area = moments['m00']
    if moments['m00'] >= minArea:
     x = moments['m10'] / moments['m00']
     y = moments['m01'] / moments['m00']
     cv2.circle(Input, (int(x), int(y)), 5, (0, 255, 0), -1)
     if(area<=120):
      Front()
```

```
      elif(area>=600):
       Back()
      else:
       Stop()

# If the limit is greater than the distance from the center that is considered centered
       if (x - center) >= may:
          Left()
             # If the limit is lower
       elif (center - x) >= may:
            Right()
      print(area)
      #Adjust(area)
     else:
      Stop()

     cv2.imshow("Input",Input)
     #cv2.imshow("HSV", imgHSV)
     #cv2.imshow("Thre", imgThresh)
     cv2.imshow("Negative", imgErode)

     if cv2.waitKey(10) == 27:
         break
          cv2.DestroyAllWindows()
          gpio.cleanup()
```