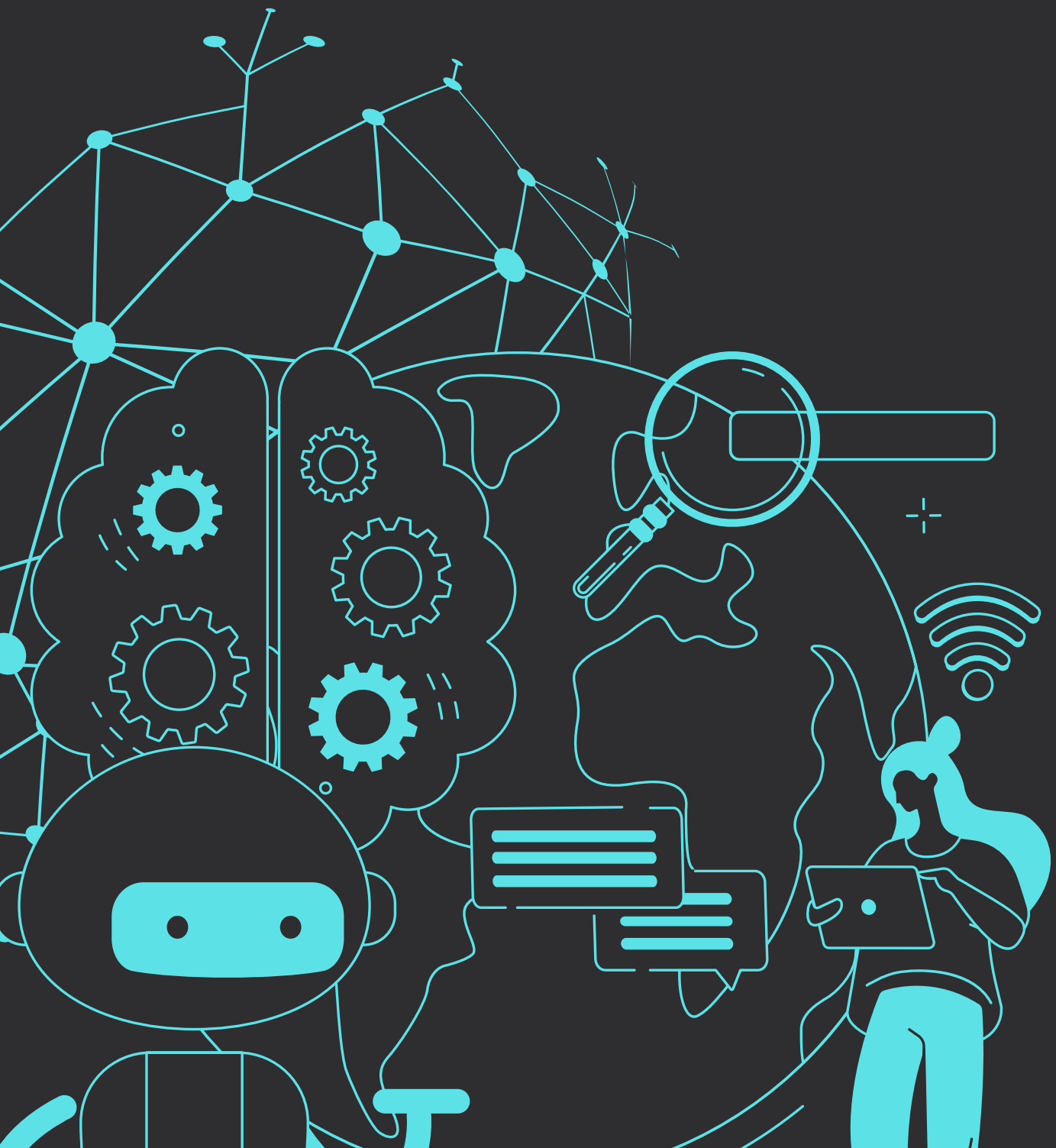
Future Engineers WRO 2024

Team 1844

TERRATECH

Vedant Dadhaniya & Devansh Harivallabhdas



Index

Contents	Page Number
Introduction	1
Part List	2
Final Schematic Diagram	5
Mobility Management	6
Assembly Of The Robot	7
Reason Of Choosing Such Parts	10
Power and Sensory Management	11
• Motor Driver	11
• Distance Sensor	14
Servo Motor	16
Obstacle Management	18
• Blocks Placed On Path	18
• Items Obtained From Code	22
Specifications Of The Robot	24
Final Code	27
• Open Challenge	27
• Challenger Round	32
References	37

WRO Team 1844-TerraTech

Introduction

Vedant Dadhaniya (Left Side)

Vedant is a STEM enthusiast who has recurringly been participating in the World Robot Olympiad since 2019 and this is his 4th time participating for it. He began by playing with jigsaw puzzles when he developed interest in LEGO which soon led to introduction to the world of robotics through EV3. Over the years, he has developed many skills in robotics including 3D part designing, electrical wiring, and languages such as Python, C#, HTML/CSS, and Java. He brings his computer vision skills with Raspberry Pi which he had worked on during the COVID Pandemic. He is an avid reader and also likes to play the guitar.

Devansh Harivallabhdas (Right Side)

Devansh is a Robotics and stem enthusiast from Ahmedabad. It is his first time at World Robot Olympiad. He is 14 and studies in class 9. As a child , Lego building was his favourite activity . He discovered his love for wires , motors and circuits in the Covid lockdown 4 years ago. Ever since he is pursuing it . He also has keen interest in Environmental applications ,Physics and chemistry and a passionate urban farmer. His dream is to create technologies and solutions that can impact the conservation of our environment.

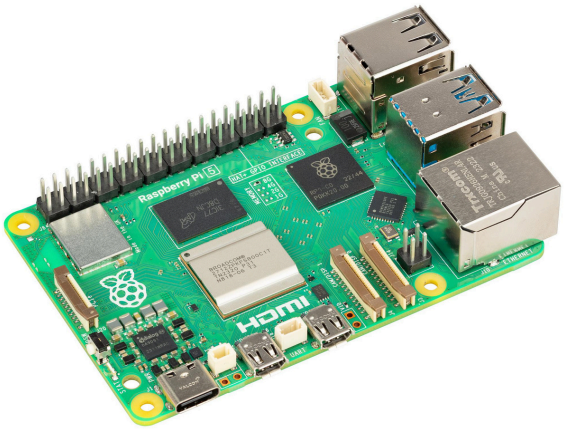
Why the name TerraTech?

Terra is Latin for the word earth , since the theme for this WRO is Earth Allies.
And Tech as we all know is the shortform of the word technology.

Part List

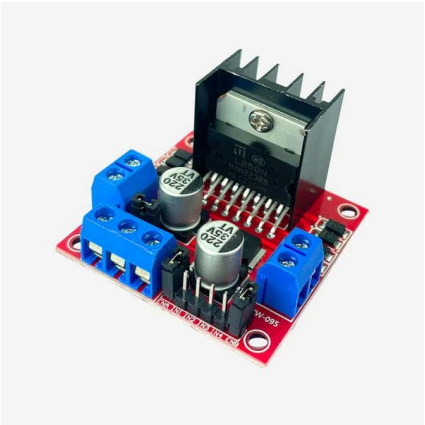
Raspberry Pi 4B

The Raspberry Pi 4B features a powerful quad-core 1.5GHz CPU and options for 2GB, 4GB, or 8GB RAM. It supports dual 4K display output via two micro-HDMI ports and offers multiple connectivity options, including USB 3.0, Gigabit Ethernet, and Wi-Fi. We are using 8gb RAM version of the Raspberry Pi.It helps us control the robot single handedly with only 1 microcontroller needed for all required functions including (but not limited to) image detection, servo control, managing sensory data and managing rear motor PWM output.



L298N Motor Driver

The L298N Motor Driver is a dual H-bridge controller capable of driving two DC motors or a stepper motor. It supports a maximum current of 2A per channel, with an input voltage range of 5V to 35V. Featuring built-in thermal protection and a compact design, it's perfect for robotics projects, allowing precise motor control and direction management. We are using this motor driver to control the rear axle motor of our robot with a supply voltage of 7.6V

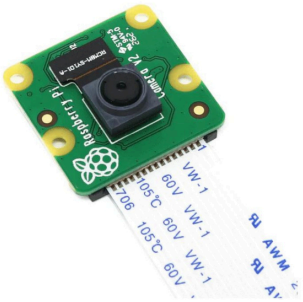


VL53L1X Distance Sensor

The VL53L1X is a high-performance time-of-flight distance sensor with a measurement range of 4 cm to 4 m. It provides accurate distance readings with fast response times, making it ideal for robotics and automation applications. The sensor features low power consumption and is compatible with various microcontrollers (including RaspberryPi 4b), enabling seamless integration into projects requiring precise distance measurement. We have used 3 such sensors - one for front , right & left each

Raspberry Pi Camera Module v2

The Raspberry Pi Camera Module V2 features an 8MP Sony IMX219 sensor, capable of capturing 1080p video at 30 frames per second. It offers improved low-light performance and supports various image formats. With a small form factor and a dedicated CSI interface, it's perfect for photography, video streaming, and computer vision projects, making it an essential accessory for Raspberry Pi enthusiasts. We have used this to detect blocks along our path.



110 RPM BO Motor

This is simple dc motor connected to scale down gears to reduce speed and increase torque of the motor. Input of 5-12 V can be given to these motors for the proper functioning of them. The rear axle is connected to 1 motor according to the rules.

Servo Motor

This is a 9 gram servo motor which has a turning radius of 180 degrees , we have connected its horn to a custom designed steering mechanism which helps us steer our robot with ease.

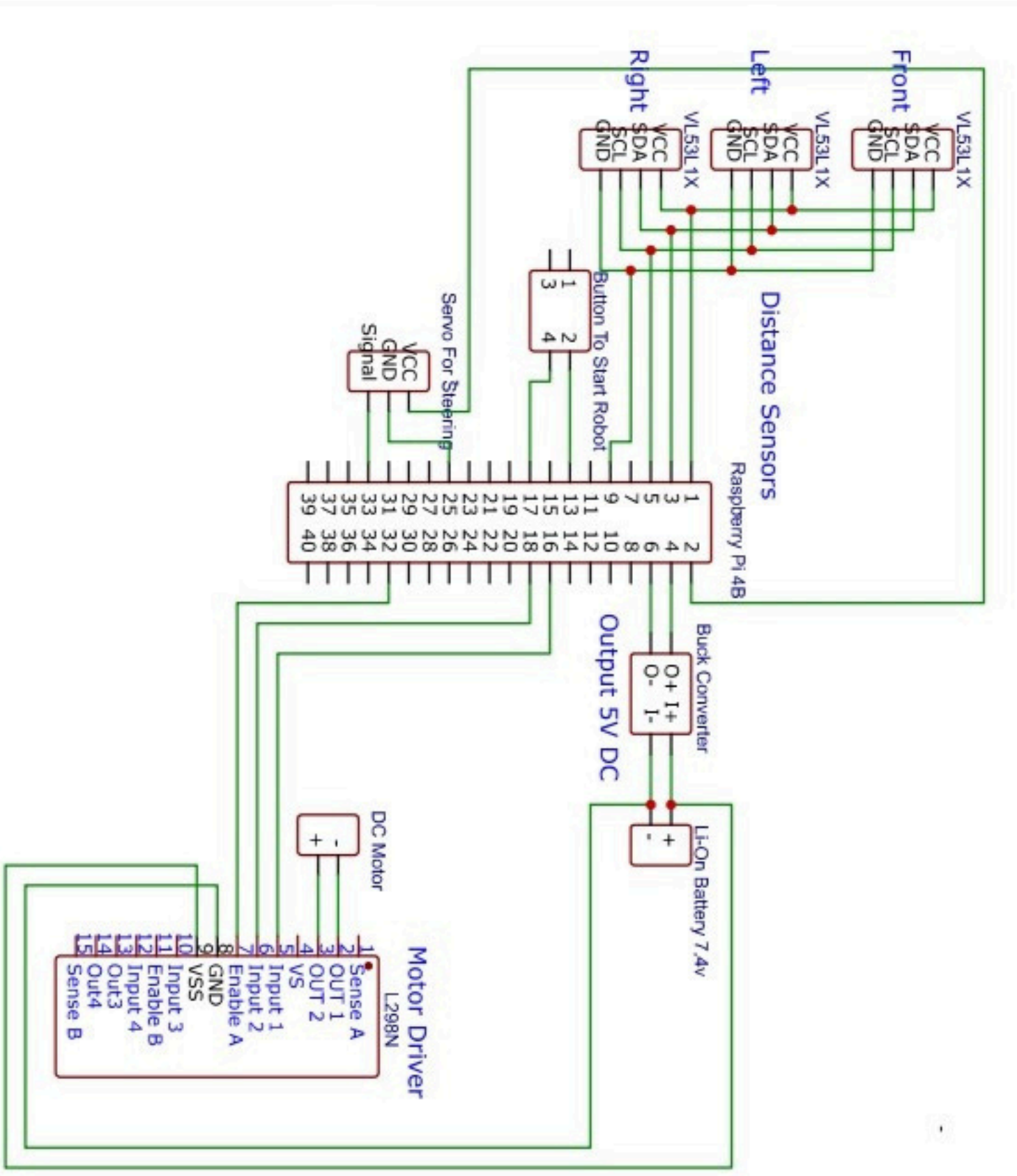
HW 272 Voltage Converter Module

We are using this module to convert 7.4 volts given by our battery and converting it into 5V 3A uninterupted output.

7.4V 2600mah Li-On Battery

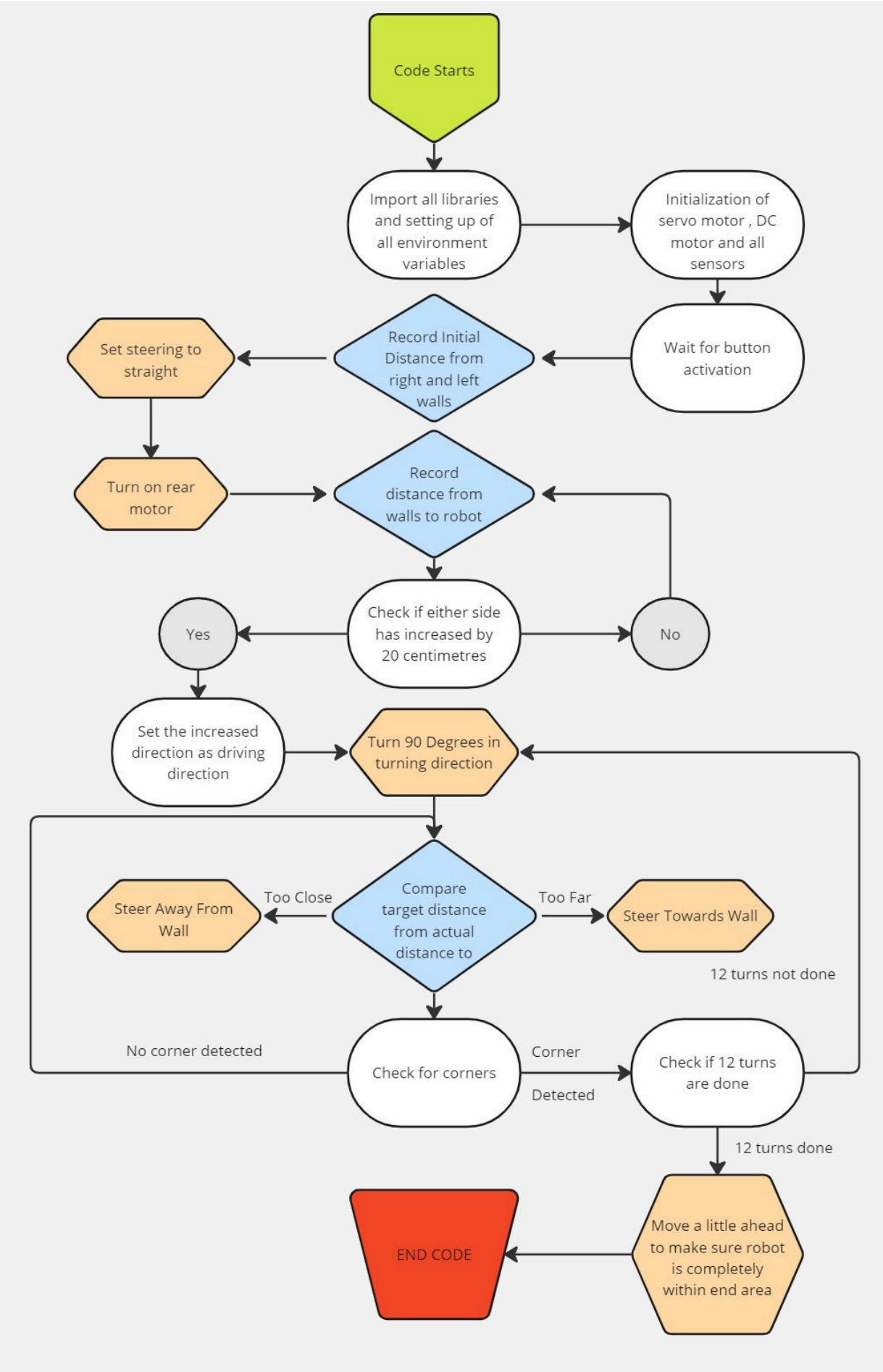
This Li-On battery provides 7.4v to our motors and 5v to the Raspberry Pi 4B. Since motor`s power fluctuations cause voltage drops on the Raspberry Pi we are using 2 such batteries , One powering the motor system and one powering the Raspberry Pi

Final Schematic Diagram



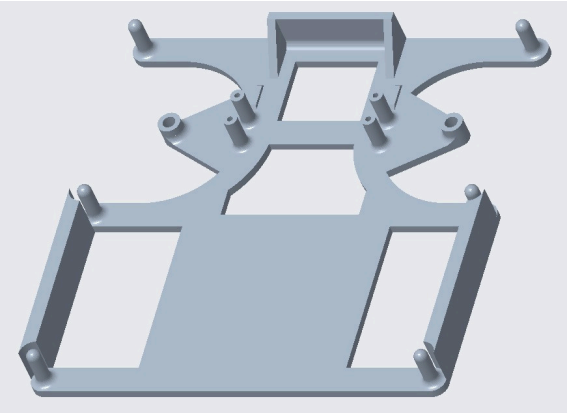
Mobility Management

Flow Chart explaining the co-relationship between different elements of our robot like sensory and motor management during open round

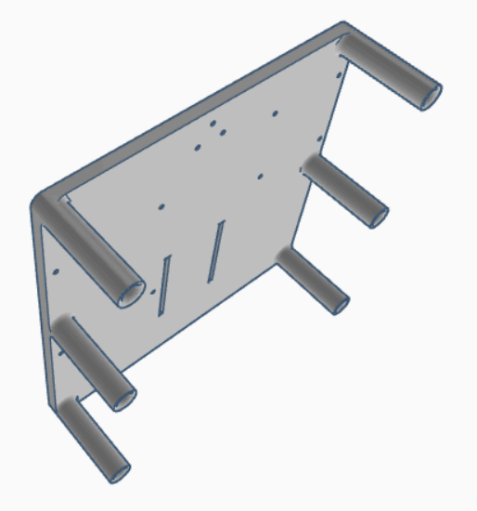


Assembly Of The Robot

3D Parts Used

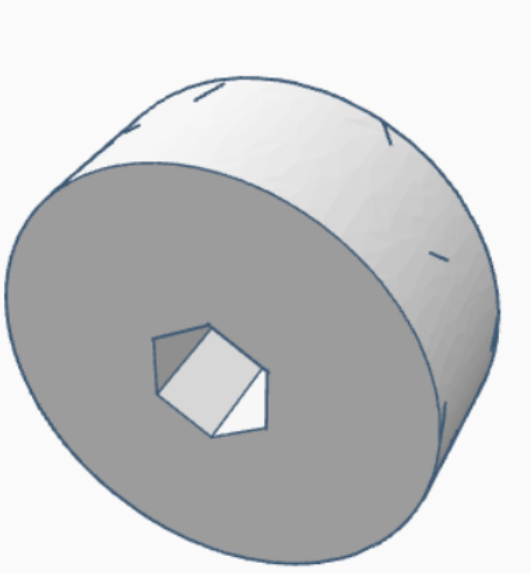


Top Plate

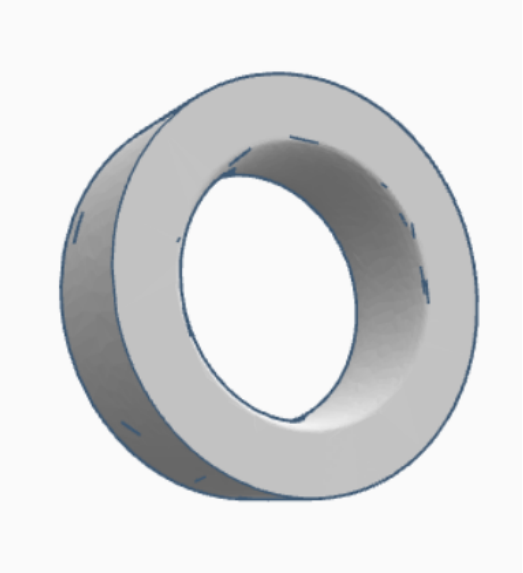


Bottom Plate

These plates are fitted on top of each other and are used to house all our electronics. Raspberry Pi , L298N Motor Driver , Battery and voltage conversion module are on top while the BO motor and Servo motors are mounted on the bottom

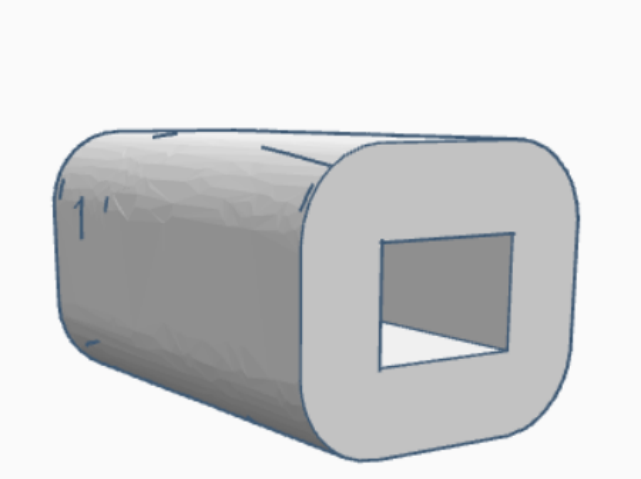


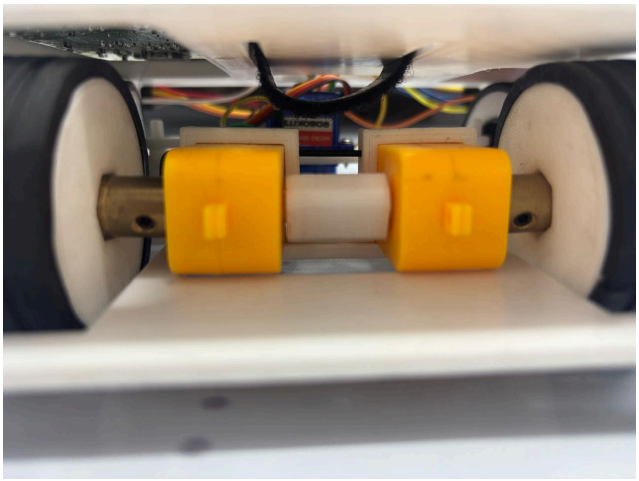
Rear Tires



Front Tires

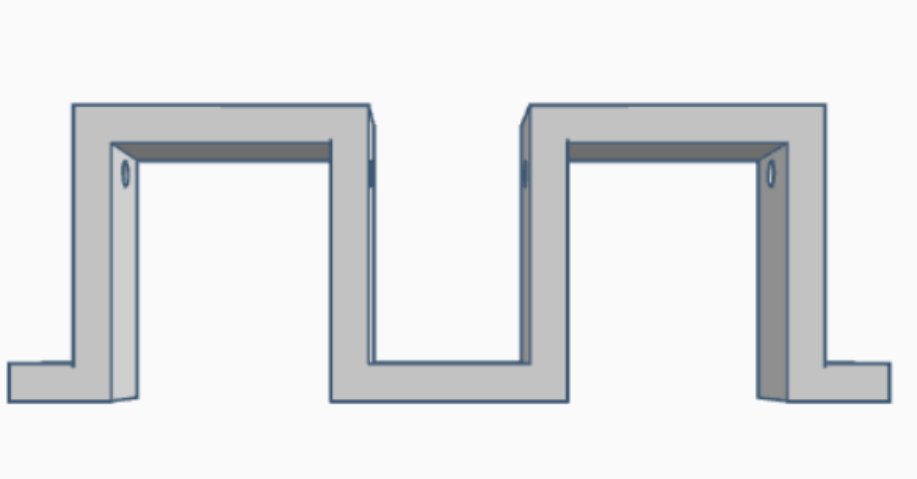
Our Tires are of two types the front and rear tires which are both coated with rubber for grip which supports them on the ground and prevents skidding , the rear tires are made to have a hexagonal coupling in them which is fitted onto the BO motor shaft while the ones in the front are made to hold the steering mechanisms





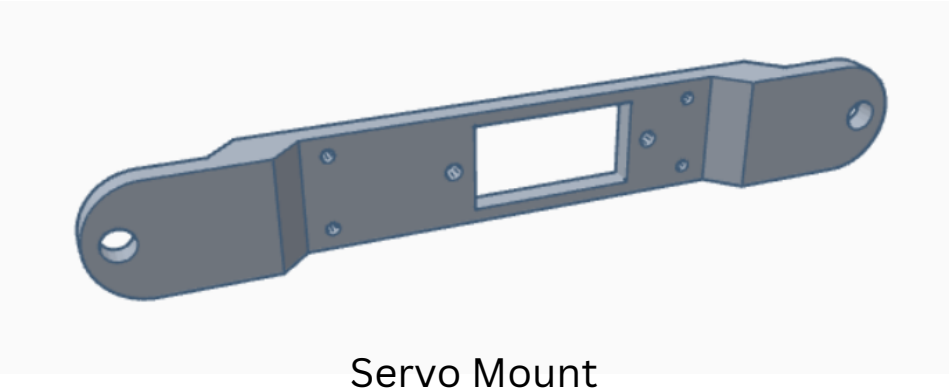
BO Coupling

This is coupling is used to join the shafts of 2 BO Motors from which only one motor is functional tThe other bo motor has the dc motor removed and just the shell which is loosely attached via the coupling. This helps support the other wheel up and prevents it from warping the axle

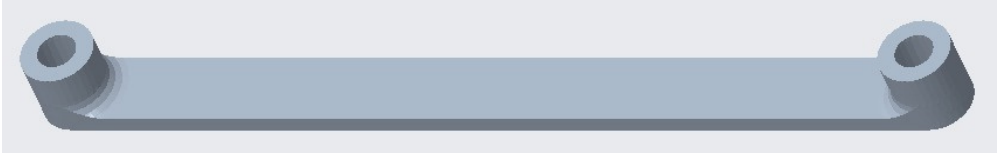


BO Motor Mount

Mounts both BO Motors onto the bottom plate with M2.5 screws

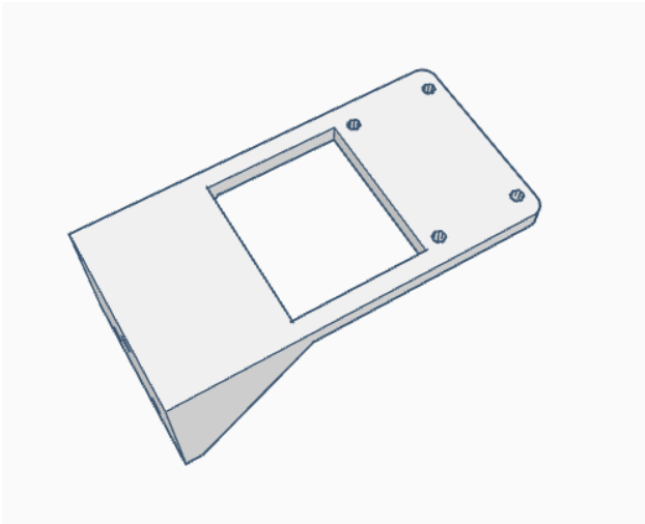


Servo Mount



Steering Rod

These mechanisms mount the servo motor and help the turning axle of the servo motor



Camera Stand

This directly screws (M2.5) onto the base plate and also holds the camera

Reason Of Choosing such parts-

Raspberry Pi 4B

This is the most commonly used microcontroller. Its example codes are readily available and it has a lot of information online. It also has all functions and requirements that fit this criteria.

L298N Motor Driver

This motor driver is quite intelligible use compared to other motor drivers.
It is viable to control motors via digital and PWM pins of the Raspberry
Pi 4b . It has plenty of power at its disposal.

VL53L1X Distance Sensors

These sensors are more efficient and fast than outdated ultrasonic sensors and provide a distance measurement upto 4 m in length with a accuracy of upto 1mm
.

Servo Motor

The 9 gram servo motor is the most commonly used servo , not only is it cheap and readily available , it also is quite powerful in respect to its size.

Raspberry Pi Camera v2

The camera provides a stable 8 mega-pixel feed for a small amount of power required . It also directly connects to the Raspberry Pi motherboard through a flex cable and absolutely no other connections are required.

110 RPM BO Motor

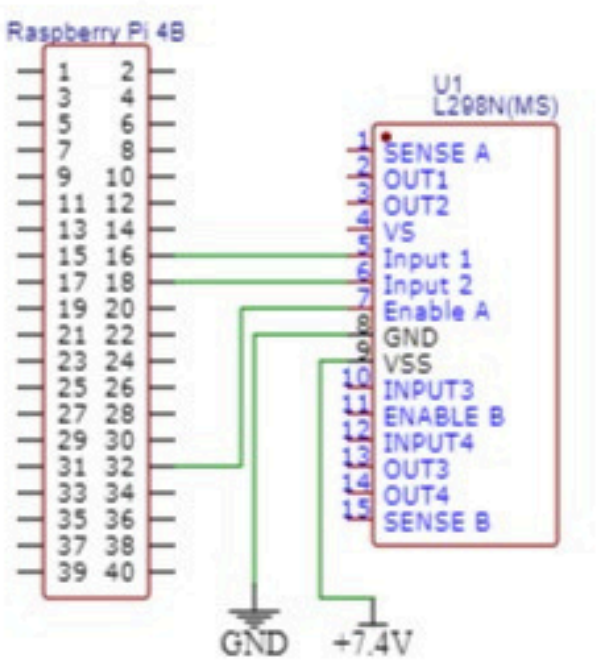
This motor provides a decent speed compared to the amount of torque it gives , which is 1kg-cm, it is elementary to use along with its connectivity to L298N Motor Driver

Battery And Voltage Converter

The voltage converter gives out a stable 5v 3A output and is also quite reliable to use.
The battery is readily available and also has enough battery capacity to support the robot for a minimum of 60 minutes of runtime for the Raspberry Pi and 45 minutes of runtime on the motor

Power And Sensory Management

L298N Motor Driver



The L298N is connected to the raspberry pi via its GPIO and PWM Pins for speed for one motor only, A GPIO 16 High gives a forward command while a GPIO 18 HIGH gives a reverse command.

Test Code

```
"""
============================== IMPORT REQUIRED LIBRARIES ==============================
"""
import RPi.GPIO as GPIO          # type: ignore
from time import sleep, time

"""
============================= SETUP ENVIRONMENT VARIABLES =============================
"""
# GPIO Pins
in1       = 23
in2       = 24
en        = 12
servo     = 13
btn       = 27

# Logic variables
mot_speed   = 50
btn_stat    = 0
servo_angle = "58"        # Servo Angle - 30 Full Right | 58 center | 86 Full Right
btn_req     = False       # True if button is required to be pressed
""
================================== SETUP GPIO PINS ==================================
"""
GPIO.setmode(GPIO.BCM)
# INPUT/OUTPUT Pins Assigned
GPIO.setup(btn , GPIO.IN)
GPIO.setup(in1 , GPIO.OUT)
GPIO.setup(in2 , GPIO.OUT)
GPIO.setup(en , GPIO.OUT)
```

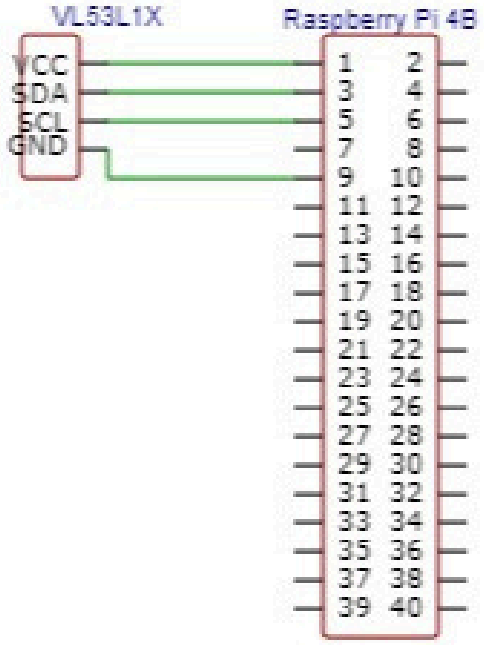
```
GPIO.setup(servo , GPIO.OUT)

# Default GPIO Status Assigned
GPIO.output(in1, GPIO.LOW)
GPIO.output(in2, GPIO.LOW)

# Assign Pins for PWM (Pulse Width Modulation)
motor  = GPIO.PWM(en    , 1000)
servo1 = GPIO.PWM(servo , 50)
"""
========================== WAIT FOR BUTTON TO BE PRESSED ==========================
"""
start = time()
while btn_req:
   if GPIO.input(btn) == GPIO.HIGH:
      btn_stat = time() - start
      if btn_stat > 1:
         btn_req = False
   else:
      btn_stat = 0
      start = time()
"""
================================== TURN ON MOTORS ==================================
"""
servo1.start(100)
motor.start(100)
motor.ChangeDutyCycle(mot_speed)
print("\n")
print("The default speed & direction of motor is LOW & Forward.....")
print("r-run s-stop f-forward b-backward l-low m-medium h-high e-exit")
print("\n")

# Set servo angle
servo1.ChangeDutyCycle((float(servo_angle) / 18))
sleep(0.5)
servo1.ChangeDutyCycle(0)

# Motor Direction
x = 'f'
"""
============================== RUN MOTOR FOR n SECONDS ==============================
"""
n = 2
c = time()
while (time() - c) < n:
   if x == 'r':
      print("run")
      if temp1 == 1:
         GPIO.output(in1, GPIO.LOW)
         GPIO.output(in2, GPIO.HIGH)
         print("forward")
         x = 'z'
elif x == 's':
 print("stop")
 GPIO.output(in1, GPIO.LOW)
 GPIO.output(in2, GPIO.LOW)
 x = 'z'
```

```
elif x == 'f':
    print("forward")
    GPIO.output(in1, GPIO.LOW)
    GPIO.output(in2, GPIO.HIGH)
    temp1 = 1
    x = 'z'
 elif x == 'b':
    print("backward")
    GPIO.output(in1, GPIO.LOW)
    GPIO.output(in2, GPIO.HIGH)
    temp1 = 0
    x = 'z'
 elif x == 'l':
    print("low")
    motor.ChangeDutyCycle(25)
    x = 'z'
 elif x == 'm':
    print("medium")
    motor.ChangeDutyCycle(50)
    x = 'z'
elif x == 'h':
    print("high")
    motor.ChangeDutyCycle(75)
    x = 'z'
 elif x == 'e':
    GPIO.cleanup()
    motor.stop()
    servo1.stop()
    print("GPIO Clean up")
    break
 else:
    print("<<<  wrong data  >>>")
    print("please enter the defined data to continue.....")
```

How To Use-

Enter a numeric number which controls the amount of seconds the motor runs for and also a speed from 0 to 100 can be given by changing the mot_speed variable's value in the code to change pwm value which directly effects speed.

VL53L1X Distance Sensors



The SDA and SDL are connected to GPIO 3 and GPIO 5 respectively and VCC and GND pins are connected to GPIO 1 and GPIO 9 respectively.*3 such sensors are attached to the robot for the measurement of length from front back right and left

Test Code

Retrieves sensor data and prints it

Can be used for:
- Checking distance from wall for placement
- Checking if I2C communication of sensor id functional
- etc
"""

"""
============================== IMPORT REQUIRED LIBRARIES ==============================
"""
import VL53L1X # type: ignore
import RPi.GPIO as GPIO # type: ignore
import time

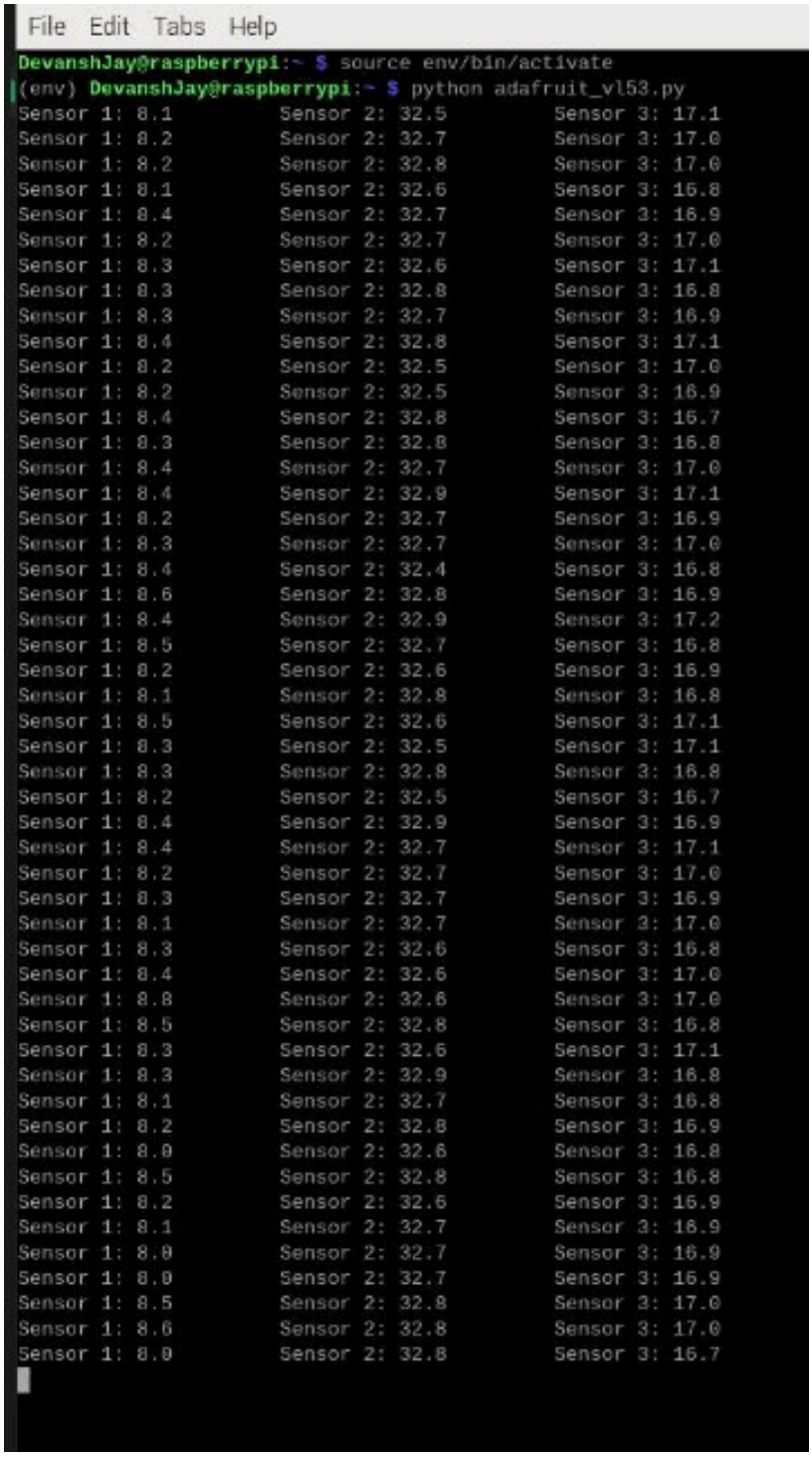
"""
================================ SETUP VL53L1X SENSOR ================================
"""
tof1 = VL53L1X.VL53L1X(i2c_bus=1, i2c_address=0x29)
tof1.open()
tof1.set_timing(66000, 70)
time.sleep(5)

"""
============================ FETCH AND PRINT SENSOR DATA ============================
"""
while True:
 tof1.start_ranging(0) # Start ranging, 0 = If custom timing budget is used, 1 = Short Range, 2 = Medium Range, 3 = Long Range
 time.sleep(0.05)

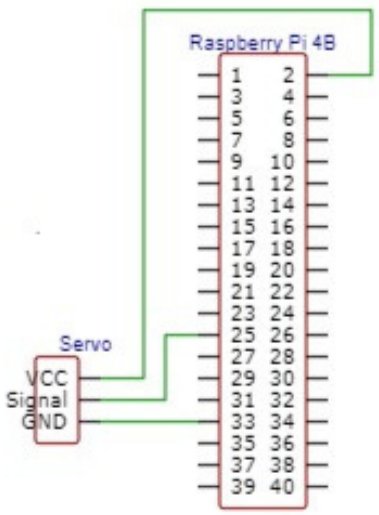
```
distance_in_mm = tof1.get_distance() # Grab the range in mm
 time.sleep(0.05)
 print('Sensor:1',distance_in_mm)
 tof1.stop_ranging()
 time.sleep(0.05)

GPIO.cleanup()
```

How To Use-

Simply prints values of distance sensors that are connected to it. 3 Sensor values (1- left side sensor, 2-right side sensor, 3- front sensor) are retrieved from the I2C bus at the same time.



Servo Motor



The servo has only 3 pins that are to be connected - VCC - GPIO 2 , GND-GPIO 33 and Signal-GPIO 25

Test Code

```
"""
============================== IMPORT REQUIRED LIBRARIES ==============================
"""
import RPi.GPIO as GPIO # type: ignore
import time


"""
================================== SETUP GPIO PINS ==================================
"""
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(13,GPIO.OUT)
# Setup PWM
servo1 = GPIO.PWM(13,50)


"""
================================== TURN ON SERVO ==================================
"""
servo1.start(0)
print ("Waiting for 2 seconds")
time.sleep(2)


"""
========================== TURN SERVO TO GIVEN INPUT ANGLE ==========================
"""
while True:
 try:
  x = float(input("Angle:"))

  if True:
   servo1.ChangeDutyCycle((float(x)/18))
   time.sleep(0.05)
   servo1.ChangeDutyCycle(0)
   except Exception as e:
   print(e)
   break
```

```
""
==================================== CLEANUP ====================================
"""
servo1.stop()
GPIO.cleanup()
print ("Goodbye")
```

How To Use-

Input a number from 0 to 180
to control the servo motor`s angle of turning
(10 to 120 is recommended otherwise the servo axis might jam)

Obstacle Management

Blocks Placed On Path

This work is solely done by the camera using the library opencv , a trackbar is created around the green and red blocks , and their distance and block count along with estimated turn angle required to fulfill the direction required is obtained is calculated using the following test code.-

```
"""
Trying to use shape detection to locate red or green block, show the closest one and print() the no. of pillars detected while autosaving the hsv ranges
(P.S. This version of the code is for RaspberryPi, not PC)
"""

"""
============================== IMPORT REQUIRED LIBRARIES ==============================
"""
import cv2
import math
import numpy as np
from picamera2 import Picamera2 # type: ignore
import serial

"""
================================== PI CAMERA SETUP ==================================
"""
p = Picamera2()

WIDTH = 1280
HEIGHT = 720

p.preview_configuration.main.size = (WIDTH, HEIGHT)
p.preview_configuration.main.format = 'RGB888'
p.preview_configuration.align()
p.configure('preview')

p.start()
"""
================================= DEFINE FUNCTIONS ==================================
"""

#* empty function
def empty(a):
    pass

#* draws shape and bounding box around a contour of sufficient area
def getContours(img, imgContour, clr):
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    count = 0
    pillars = []

    # draw vert. lines
    cv2.line(imgContour, (int(WIDTH / 3), 0), (int(WIDTH / 3), HEIGHT), (0, 255, 255), 1)
    cv2.line(imgContour, (int(2 * WIDTH / 3), 0), (int(2 * WIDTH / 3), HEIGHT), (0, 255, 255), 1)

    for cnt in contours:
        area = cv2.contourArea(cnt)
        area_thresh = cv2.getTrackbarPos("AreaThresh", "Parameters")
        if area >= area_thresh:
            count += 1
```

```
cv2.drawContours(imgContour, cnt, -1, (255, 0, 255), 2)

peri = cv2.arcLength(cnt, True)
appx = cv2.approxPolyDP(cnt, 0.02 * peri, True)

x, y, w, h = cv2.boundingRect(appx)
coords = [(x, y), (x + w, y), (x, y + h), (x + w, y + h)]
pillars.append(coords)
        drawn = False

        for i, j in coords:
            if i < (2 * WIDTH / 3) and clr == 'green':
                cv2.rectangle(imgContour, (x, y), (x + w, y + h), (0, 0, 255), 3)
                drawn = True
            elif i > (WIDTH / 3) and clr == 'red':
                cv2.rectangle(imgContour, (x, y), (x + w, y + h), (0, 0, 255), 3)
                drawn = True
        if not drawn:
            cv2.rectangle(imgContour, (x, y), (x + w, y + h), (0, 255, 0), 3)

        # area is inversely proportional to square of dist
        dist = 30 / math.sqrt(area / 32100)

        cv2.putText(imgContour, "x: " + str(x) + ', y: ' + str(y) + ', w: ' + str(w) + ', h: ' + str(h), (x, y - 70),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 255, 0), 2)
        cv2.putText(imgContour, "Area" + str(int(area)), (x, y - 45), cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 255, 0), 2)
        cv2.putText(imgContour, "Dist" + str(int(dist)), (x, y - 20), cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 255, 0), 2)

    return count, pillars

"""
================================== LOAD MASK DATA ==================================
"""
with open('green_mask.txt', 'r') as gm:
    g_hmin, g_hmax, g_smin, g_smax, g_vmin, g_vmax = gm.read().split(' ')

with open('red_mask.txt', 'r') as rm:
    r_hmin, r_hmax, r_smin, r_smax, r_vmin, r_vmax = rm.read().split(' ')
"""
TRACKBARS SETUP
"""
cv2.namedWindow("Green HSV dials")
cv2.resizeWindow("Green HSV dials", 640, 270)
cv2.createTrackbar("HUE Min", "Green HSV dials", int(g_hmin), 179, empty)
cv2.createTrackbar("HUE Max", "Green HSV dials", int(g_hmax), 179, empty)
cv2.createTrackbar("SAT Min", "Green HSV dials", int(g_smin), 255, empty)
cv2.createTrackbar("SAT Max", "Green HSV dials", int(g_smax), 255, empty)
cv2.createTrackbar("VAL Min", "Green HSV dials", int(g_vmin), 255, empty)
cv2.createTrackbar("VAL Max", "Green HSV dials", int(g_vmax), 255, empty)
cv2.createTrackbar("Start detection", "Green HSV dials", 0, 1, empty)

cv2.namedWindow("Red HSV dials")
cv2.resizeWindow("Red HSV dials", 640, 270)
cv2.createTrackbar("HUE Min", "Red HSV dials", int(r_hmin), 179, empty)
cv2.createTrackbar("HUE Max", "Red HSV dials", int(r_hmax), 179, empty)
cv2.createTrackbar("SAT Min", "Red HSV dials", int(r_smin), 255, empty)
cv2.createTrackbar("SAT Max", "Red HSV dials", int(r_smax), 255, empty)
cv2.createTrackbar("VAL Min", "Red HSV dials", int(r_vmin), 255, empty)
cv2.createTrackbar("VAL Max", "Red HSV dials", int(r_vmax), 255, empty)
```

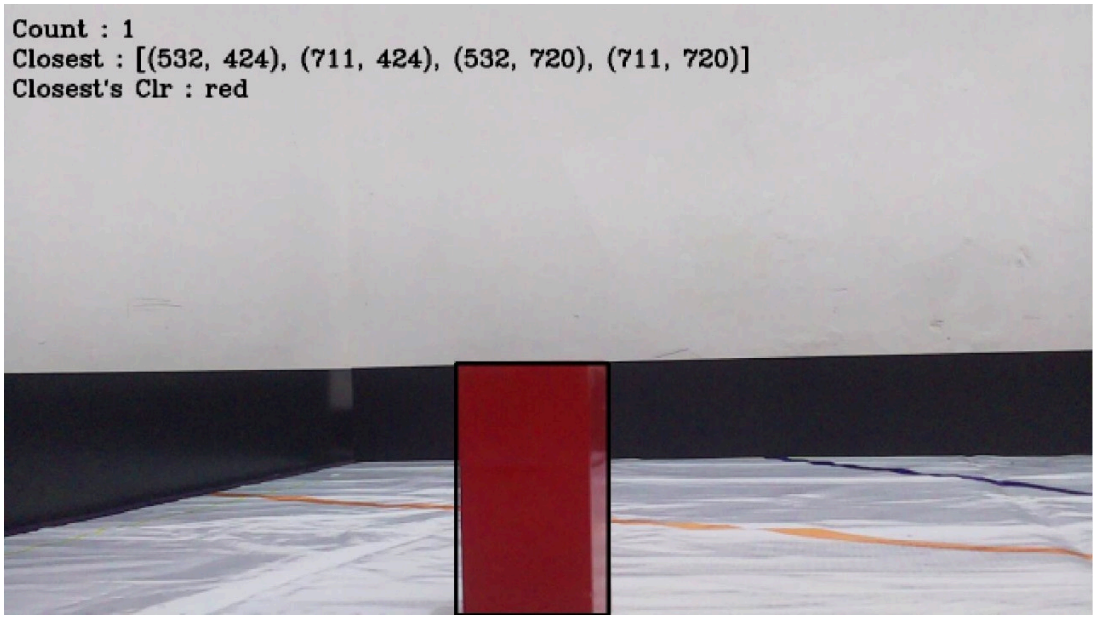
```
cv2.namedWindow("Parameters")
cv2.resizeWindow("Parameters", 640, 120)
cv2.createTrackbar("THRESH_2", "Parameters", 150, 255, empty)
cv2.createTrackbar("THRESH_1", "Parameters", 255, 255, empty)
cv2.createTrackbar("AreaThresh", "Parameters", 5000, 30000, empty)

"""
================================== RUNTIME CODE ==================================
"""
runtime = True
while runtime:
    pillar_cnt = 0
    img = p.capture_array()
    img_hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

    g_h_min = cv2.getTrackbarPos("HUE Min", "Green HSV dials")
    g_h_max = cv2.getTrackbarPos("HUE Max", "Green HSV dials")
    g_s_min = cv2.getTrackbarPos("SAT Min", "Green HSV dials")
    g_s_max = cv2.getTrackbarPos("SAT Max", "Green HSV dials")
    g_v_min = cv2.getTrackbarPos("VAL Min", "Green HSV dials")
    g_v_max = cv2.getTrackbarPos("VAL Max", "Green HSV dials")

    r_h_min = cv2.getTrackbarPos("HUE Min", "Red HSV dials")
    r_h_max = cv2.getTrackbarPos("HUE Max", "Red HSV dials")
    r_s_min = cv2.getTrackbarPos("SAT Min", "Red HSV dials")
    r_s_max = cv2.getTrackbarPos("SAT Max", "Red HSV dials")
    r_v_min = cv2.getTrackbarPos("VAL Min", "Red HSV dials")
    r_v_max = cv2.getTrackbarPos("VAL Max", "Red HSV dials")

    detect = cv2.getTrackbarPos("Start detection", "Green HSV dials")

    g_lower = np.array([g_h_min, g_s_min, g_v_min])
    g_upper = np.array([g_h_max, g_s_max, g_v_max])

    r_lower = np.array([r_h_min, r_s_min, r_v_min])
    r_upper = np.array([r_h_max, r_s_max, r_v_max])

    g_mask = cv2.inRange(img_hsv, g_lower, g_upper)
    g_result = cv2.bitwise_and(img, img, mask=g_mask)

   r_mask = cv2.inRange(img_hsv, r_lower, r_upper)
    r_result = cv2.bitwise_and(img, img, mask=r_mask)
    if detect == 1:
    g_img_Contour = g_result.copy()
    r_img_Contour = r_result.copy()

    g_img_blur = cv2.GaussianBlur(g_result, (7, 7), 1)
    g_img_gray_with_blur = cv2.cvtColor(g_img_blur, cv2.COLOR_BGR2GRAY)

    r_img_blur = cv2.GaussianBlur(r_result, (7, 7), 1)
    r_img_gray_with_blur = cv2.cvtColor(r_img_blur, cv2.COLOR_BGR2GRAY)

    thresh1 = cv2.getTrackbarPos("THRESH_1", "Parameters")
    thresh2 = cv2.getTrackbarPos("THRESH_2", "Parameters")
    kernel = np.ones((5, 5))
```

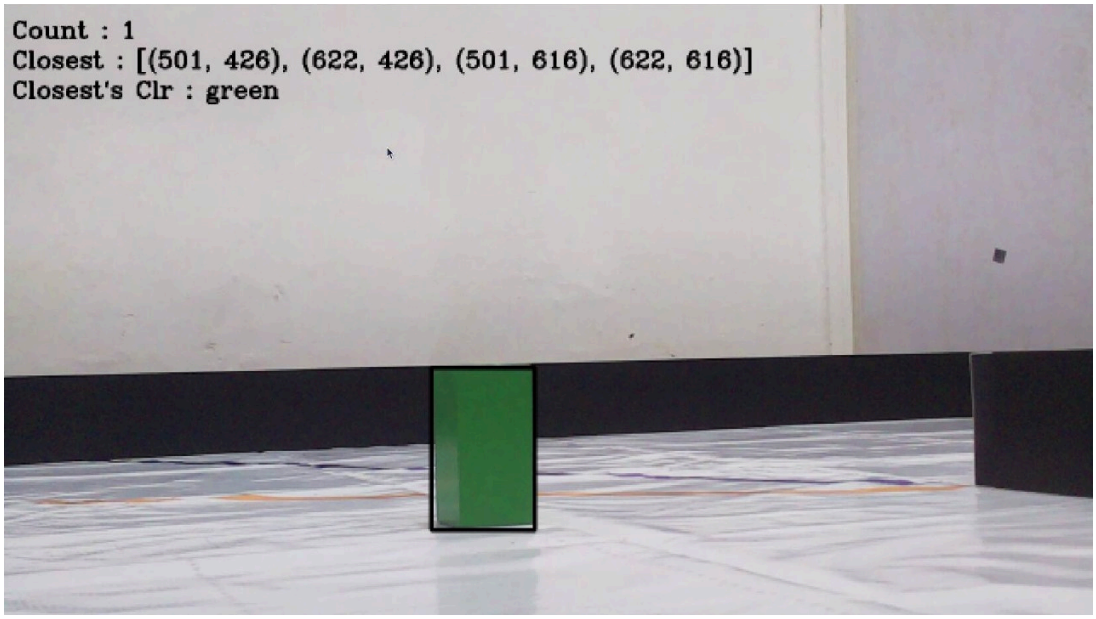
```
        g_imgCanny = cv2.Canny(g_img_gray_with_blur, thresh1, thresh2)
        g_imgDil = cv2.dilate(g_imgCanny, kernel, iterations=1)

        r_imgCanny = cv2.Canny(r_img_gray_with_blur, thresh1, thresh2)
        r_imgDil = cv2.dilate(r_imgCanny, kernel, iterations=1)

        cnt1, pillars_g = getContours(g_imgDil, g_img_Contour, 'green')
        cnt2, pillars_r = getContours(r_imgDil, r_img_Contour, 'red')

        g_contour_display = cv2.resize(g_img_Contour, None, None, 0.5, 0.5)
        r_contour_display = cv2.resize(r_img_Contour, None, None, 0.5, 0.5)

        y_min = 0
        closest_p = None
        closest_p_clr = 'None'

        for p in pillars_g + pillars_r:
            if p[3][1] > y_min:
                y_min = p[3][1]
                closest_p = p
                closest_p_clr = 'green' if p in pillars_g else 'red'

        cv2.imshow('g contour', g_contour_display)
        cv2.imshow('r contour', r_contour_display)

        cv2.putText(img, "Count : " + str(int(cnt1 + cnt2)), (10, 40), cv2.FONT_HERSHEY_COMPLEX, 0.9, (0, 0, 0), 2)
        cv2.putText(img, "Closest : " + str(closest_p), (10, 75), cv2.FONT_HERSHEY_COMPLEX, 0.9, (0, 0, 0), 2)
        cv2.putText(img, "Closest's Clr : " + str(closest_p_clr), (10, 110), cv2.FONT_HERSHEY_COMPLEX, 0.9, (0, 0, 0), 2)
        if closest_p is not None:
            cv2.rectangle(img, closest_p[0], closest_p[3], (0, 0, 0), 3)

    img = cv2.resize(img, None, None, 0.5, 0.5)
    g_mask = cv2.resize(g_mask, None, None, 0.35, 0.35)
    r_mask = cv2.resize(r_mask, None, None, 0.35, 0.35)
    g_result = cv2.resize(g_result, None, None, 0.35, 0.35)
    r_result = cv2.resize(r_result, None, None, 0.35, 0.35)
    cv2.imshow('img', img)
    cv2.imshow('g isolated', g_result)
    cv2.imshow('r isolated', r_result)

    if cv2.waitKey(1) == ord('q'):
        runtime = False

"""
==================================== CLEANUP ====================================
"""
cv2.destroyAllWindows()

"""
================================ SAVE MASK DATA ================================
"""
with open('green_mask.txt', 'w') as gm:
    gm.write(f'{g_h_min} {g_h_max} {g_s_min} {g_s_max} {g_v_min} {g_v_max}')

with open('red_mask.txt', 'w') as rm:
    rm.write(f'{r_h_min} {r_h_max} {r_s_min} {r_s_max} {r_v_min} {r_vmax}')
```

- Virtual Environment is used here

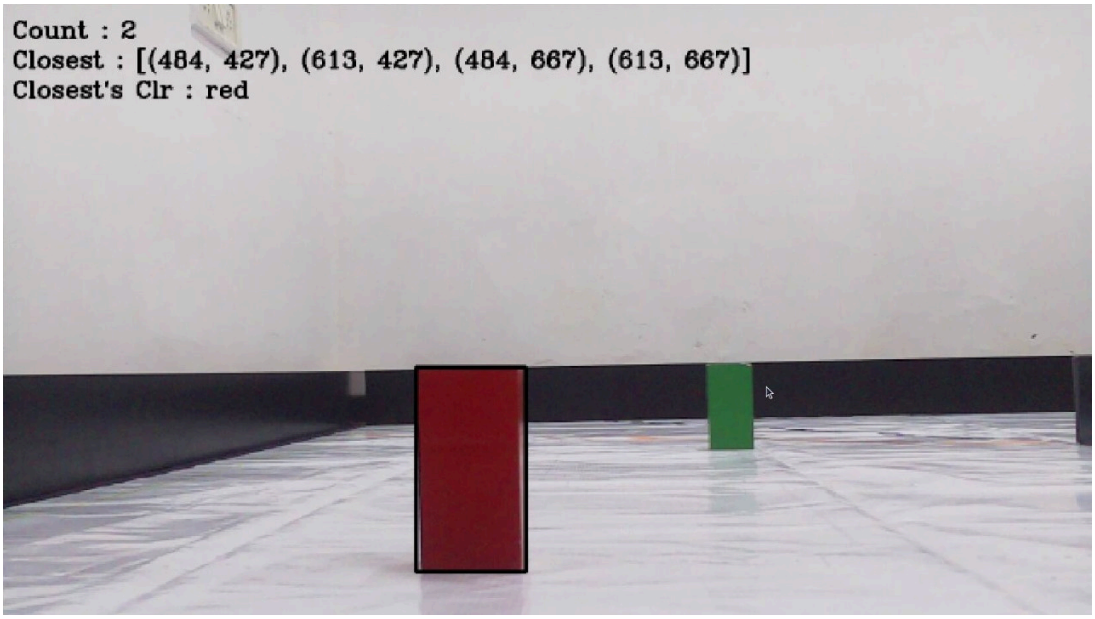
- This code requires 2 text documents in the same folder as the code to function , they are : red_mask.txt and green_mask.txt with items as 4 172 145 247 105 255 and
  56 85 82 239 71 255 respectively

Items Obtained From Code

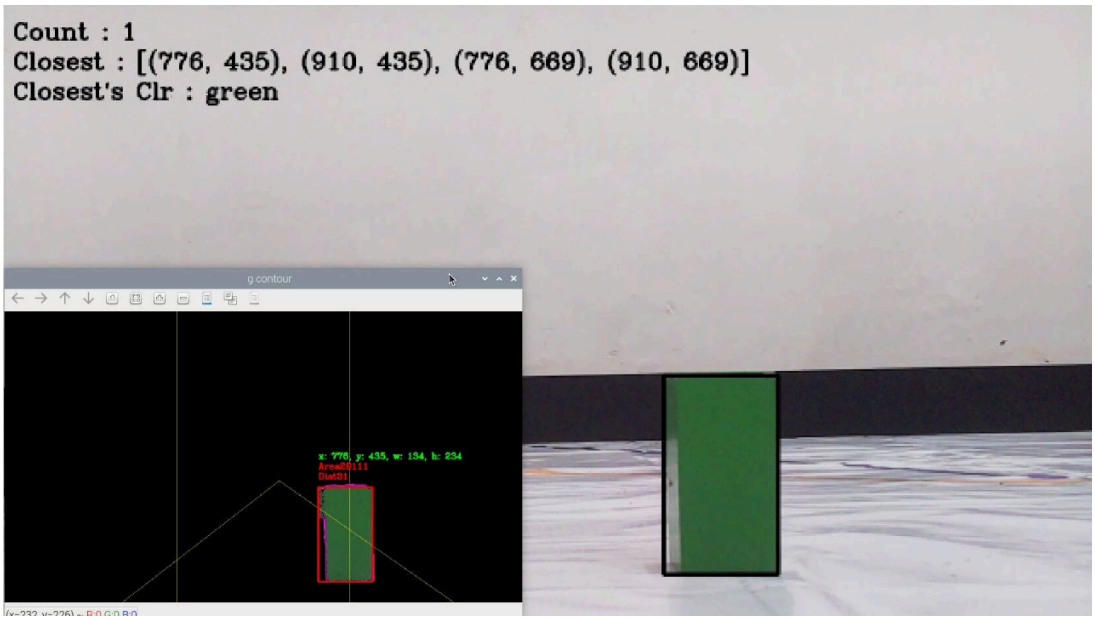




Red And Green Blocks Detected By The Code With Trackbars and Closest Colour



Block count calculated by the code , only the closest block has a trackbar around it , also done by the code

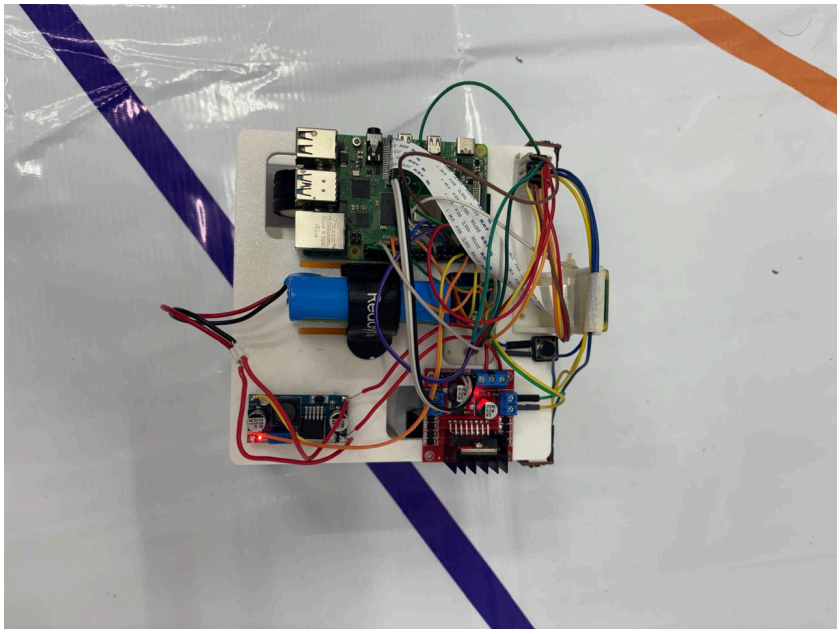


Red colour of text indicates that if the robot goes straight, it will move from the wrong side of the block or it will hit it and triangular line on the small window is the projection of the robot's wheels if the robot moves in a straight line , also done by this code.

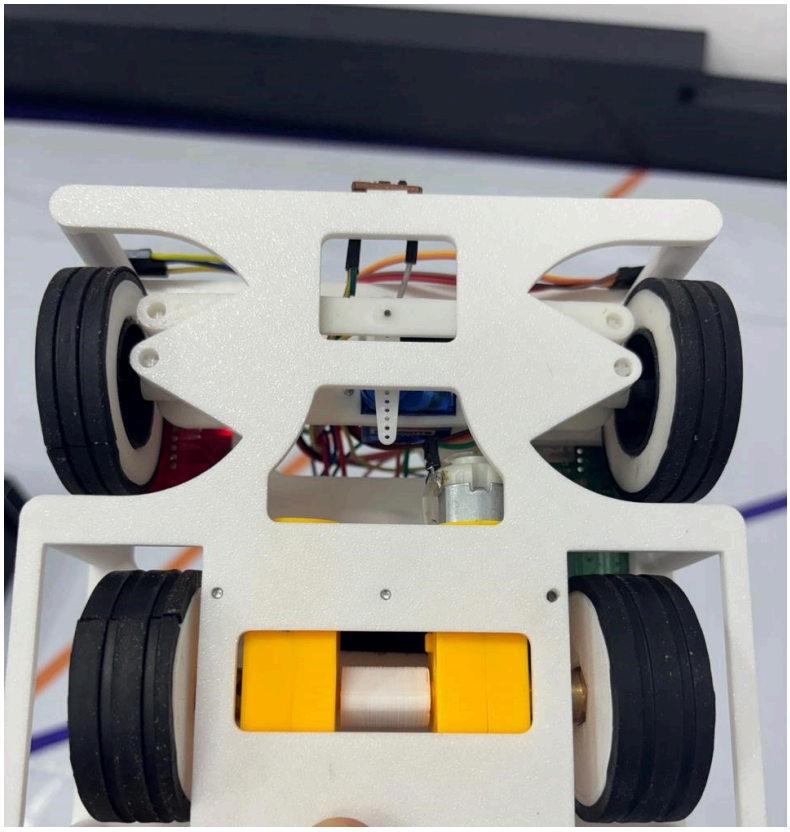
- Also to be taken into note that the camera is mounted exactly 10cm from the ground so that the block and the wall approximately are seen as the same height.

Specifications Of The Robot

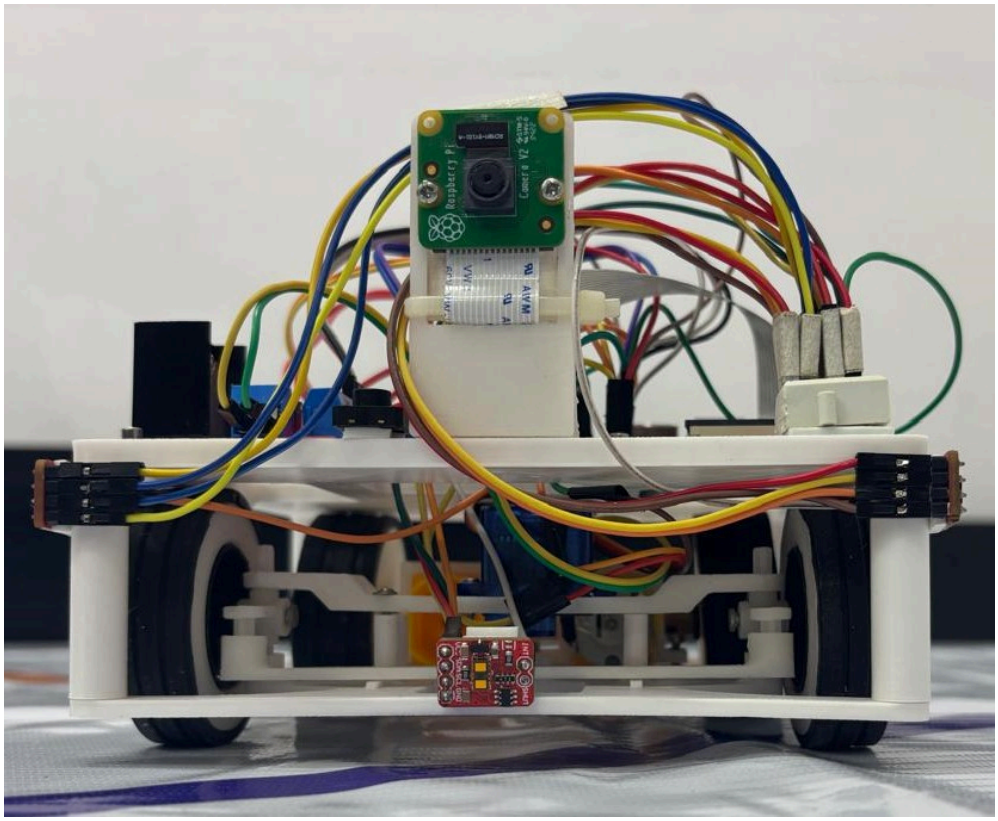
Mass - 606 grams
Dimensions - 16 cm(length) by 16 cm(breadth) by 13cm (height with cables , without cables is around 11cm)



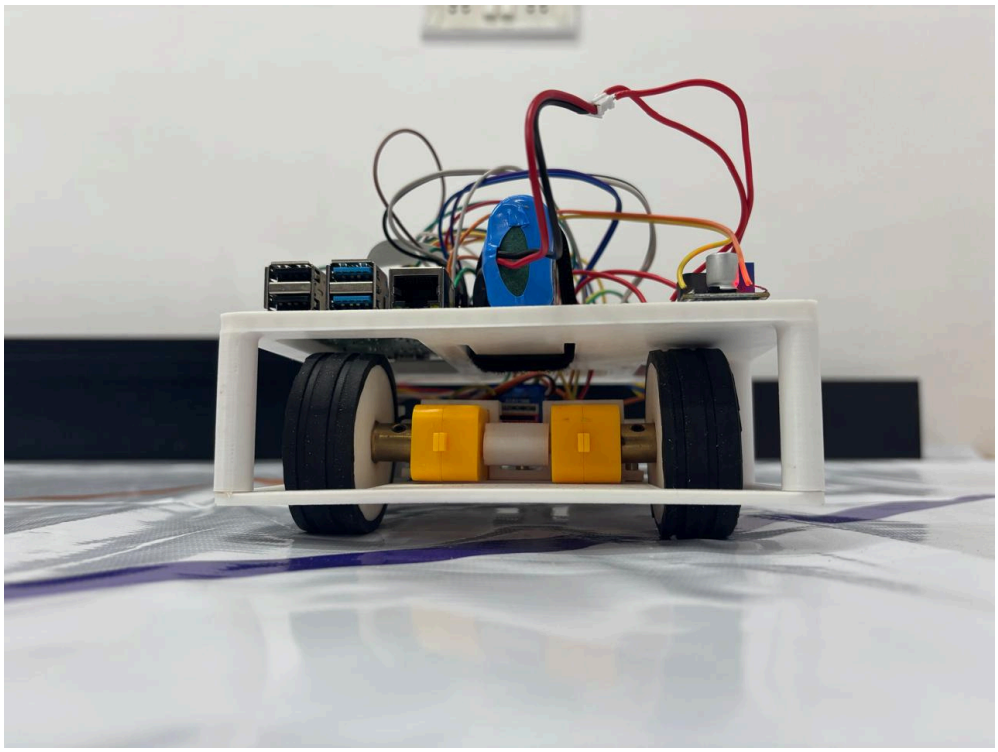
Top View



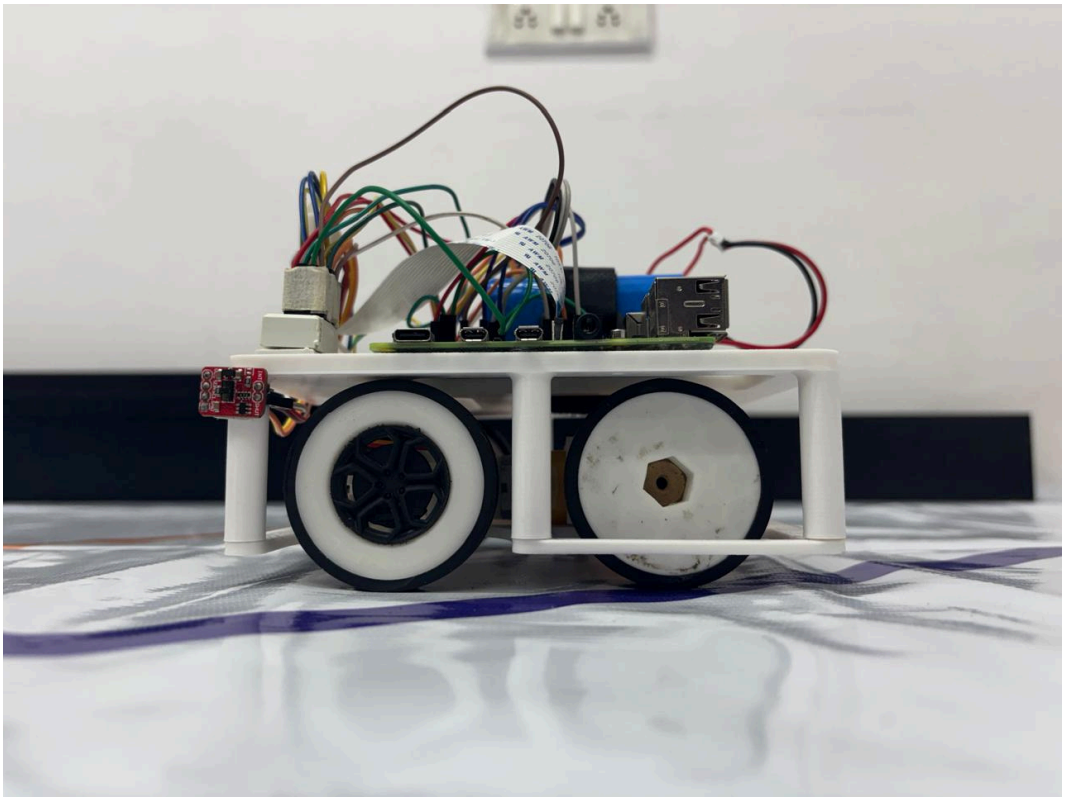
Bottom View



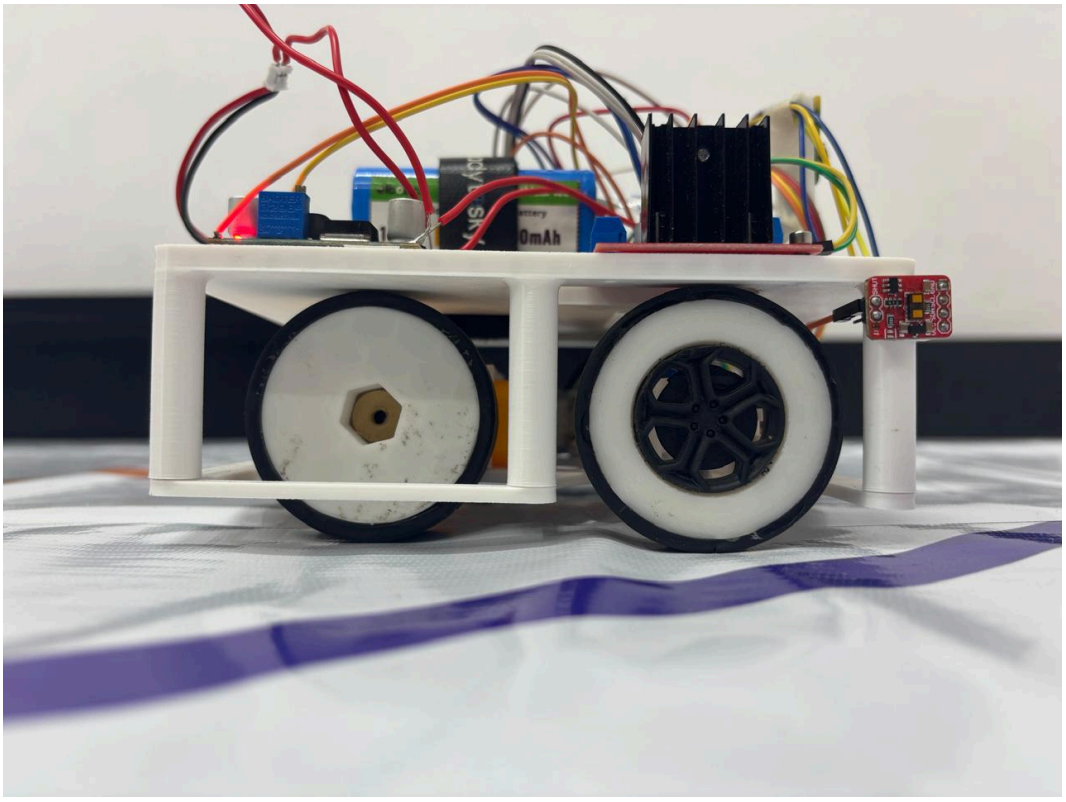
Front View



Back View



Left View



Right View

- Engineering to be taken into note , the front wheels are spread wider apart than the backwheels for higher turning efficiency as seen is race cars.

Final Codes

Open Challenge

```
import RPi.GPIO as GPIO
import time
import board
import digitalio
import adafruit_vl53l1x

"""
GPIO PINS
"""
btn       = 27
en        = 12
in1       = 23
in2       = 24
servo     = 13

"""
DATA VARIABLES
"""
cur       = 0
lap_counter = 0
turn_dir    = 0
cmd_given   = True

"""
GPIO SETUP
"""
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(in1   ,GPIO.OUT)
GPIO.setup(in2   ,GPIO.OUT)
GPIO.setup(en    ,GPIO.OUT)
GPIO.setup(servo ,GPIO.OUT)
GPIO.output(in1  ,GPIO.LOW)
GPIO.output(in2  ,GPIO.LOW)
GPIO.setup(btn   , GPIO.IN)
"""
PWM SETUP
"""
mpower = GPIO.PWM(en   ,1000)
servo1 = GPIO.PWM(servo,  50)

"""
I2C SETUP
"""
i2c = board.I2C()

"""
ToF SHUT PIN SETUP
"""
shut_1 = digitalio.DigitalInOut(board.D5 )
shut_2 = digitalio.DigitalInOut(board.D6 )
shut_3 = digitalio.DigitalInOut(board.D26)
shut_1.switch_to_output(value=False)
shut_2.switch_to_output(value=False)
shut_3.switch_to_output(value=False)
```

```
"""
ToF SETUP and ADDRESS REASSIGNMENT
"""
#sensor 1
shut_1.value       = True
tof_1              = adafruit_vl53l1x.VL53L1X(i2c)
tof_1.distance_mode = 2
tof_1.timing_budget = 100
tof_1.set_address(0x28)

#sensor 2
shut_2.value       = True
tof_2              = adafruit_vl53l1x.VL53L1X(i2c)
tof_2.distance_mode = 2
tof_2.timing_budget = 100
tof_2.set_address(0x38)

#sensor 3
shut_3.value       = True
tof_3              = adafruit_vl53l1x.VL53L1X(i2c)
tof_3.distance_mode = 2
tof_3.timing_budget = 100
tof_3.set_address(0x48)            """
WAIT FOR BTN ACTIVATION
"""
start= time.time()
print("Awaiting command")
while not cmd_given:
    if GPIO.input(btn) == GPIO.HIGH:
        cur = time.time() - start
    else:
        cur = 0
        start = time.time()
    if cur > 0.5:
        cmd_given = True
time.sleep(1)

"""
BTN ACTIVATED
"""
"""
START DETECTING
"""
tof_1.start_ranging()
tof_2.start_ranging()
tof_3.start_ranging()
time.sleep(0.1)
d1, d2, d3 = None, None, None
while (d1 is None) or (d2 is None) or (d3 is None):
    time.sleep(0.1)
    if tof_1.data_ready:
        d1 = tof_1.distance
        print("Sensor 1: ", str(d1), end="        ")
        tof_1.clear_interrupt()
    if tof_2.data_ready:
        d2 = tof_2.distance
        print("Sensor 2: ", str(d2), end="        ")
        tof_2.clear_interrupt()
```

```
 if tof_3.data_ready:
 d3 = tof_3.distance
 print("Sensor 3: ", str(d3), end=" ")
 tof_3.clear_interrupt()
 print("")

"""
BEGIN PWM
"""
servo1.start(0)
mpower.start(100)
mpower.ChangeDutyCycle(50)

SET SERVO ANGLE
"""
d = "57"
servo1.ChangeDutyCycle((float(d)/18))
time.sleep(0.5)

if tof_1.data_ready:
    c1 = tof_1.distance
    print("Sensor 1: ", str(c1), end="        ")
    tof_1.clear_interrupt()
if tof_2.data_ready:
    c2 = tof_2.distance
    print("Sensor 2: ", str(c2))
    tof_2.clear_interrupt()
time.sleep(0.1)

"""
TURN ON MOTOR
"""
GPIO.output(in1, GPIO.LOW)
GPIO.output(in2, GPIO.HIGH)

while turn_dir == 0:
    time.sleep(0.1)
    if tof_1.data_ready:
        d1 = tof_1.distance
        print("Sensor 1: ", str(d1), end="        ")
        tof_1.clear_interrupt()
    if tof_2.data_ready:
        d2 = tof_2.distance
        print("Sensor 2: ", str(d2), end="        ")
        tof_2.clear_interrupt()
    print("")
    if d1 is not None and d1 - c1 > 15:
        turn_dir = 1
        sensor = tof_1
        sleeptime = (1/24)*(c1-25) if c1 > 25 else 0
    elif d2 is not None and d2 - c2 > 15:
        turn_dir = -1
        sensor = tof_2
        sleeptime = (1/24)*(c2-25) if c2 > 25 else 0
```

```
servo1.ChangeDutyCycle(0)
GPIO.output(in1, GPIO.LOW)
GPIO.output(in2, GPIO.LOW)
if turn_dir == -1:
    d = 80
elif turn_dir == 1:
    d = 30
servo1.ChangeDutyCycle((float(d)/18))
time.sleep(0.5)
servo1.ChangeDutyCycle(0)
mpower.ChangeDutyCycle(70)
time.sleep(0.5)
GPIO.output(in2, GPIO.HIGH)
time.sleep(1.55 + (turn_dir / 20))
GPIO.output(in2, GPIO.LOW)
lap_counter += 1

servo1.ChangeDutyCycle((56/18))
time.sleep(0.3)
servo1.ChangeDutyCycle(0)
mpower.ChangeDutyCycle(60)
time.sleep(0.5)
GPIO.output(in2, GPIO.HIGH)
time.sleep(sleeptime)
GPIO.output(in2, GPIO.LOW)

turning = False
sensor.start_ranging()
time.sleep(0.1)
GPIO.output(in2, GPIO.HIGH)
prev = sensor.distance
sensor.clear_interrupt()
time.sleep(0.1)
while lap_counter < 13:
    time.sleep(0.1)
    if sensor.data_ready:
        dist = sensor.distance
        print("Sensor Of Detected Dir: ", str(dist), end='      ')
        sensor.clear_interrupt()
        if dist is None:
            time.sleep(1)
            print("NONE")
            servo1.ChangeDutyCycle((d)/18)
            time.sleep(0.1)
            servo1.ChangeDutyCycle(0)
        elif dist > 36 and prev > 36:
            if not turning:
                mpower.ChangeDutyCycle(60)
                turning = True
            servo1.ChangeDutyCycle((d)/18)
            time.sleep(0.1)
            servo1.ChangeDutyCycle(0)
            print("Turning")
            prev = dist
        else:
            if turning:
                turning = False
                mpower.ChangeDutyCycle(60)
                lap_counter += 1
```

```
turning = False
 a = 56 + (((32 - dist) / 4) * 6 * turn_dir)
 a = min(max(a, 44), 68) # Clamp a between 44 and 68
 servo1.ChangeDutyCycle((a)/18)
 time.sleep(0.1)
 servo1.ChangeDutyCycle(0)
 print("")
 prev = dist
 else:
 print("No data . . . . . .. . . . . ")
 time.sleep(1)

time.sleep(1.5)
GPIO.output(in2, GPIO.LOW)
```

Challenger Round Code

```
"""
Trying to use shape detection to locate red or green block, show the closest one and print() the no. of pillars detected while autosaving the hsv ranges
(P.S. This version of the code is for RaspberryPi not PC)
"""

"""
IMPORTS
"""
import cv2
import math
import numpy as np
from picamera2 import Picamera2
import RPi.GPIO as GPIO
import time
import board
import digitalio
import adafruit_vl53l1x

in1 = 23
in2 = 24
en = 12
servo = 13
SHUT1 = 5
SHUT2 = 6
SHUT3 = 26

"""
GPIO SETUP
"""
GPIO.setwarnings(False)
GPIO.setmode(GPIO.BCM)
GPIO.setup(in1    , GPIO.OUT)
GPIO.setup(in2    , GPIO.OUT)
GPIO.setup(en     , GPIO.OUT)
GPIO.setup(servo  , GPIO.OUT)
GPIO.output(in1   , GPIO.LOW)
GPIO.output(in2   , GPIO.LOW)

motor  = GPIO.PWM(   en, 1000)
servo1 = GPIO.PWM(servo,   50)
motor.start(100)
motor.ChangeDutyCycle(50)
servo1.start(0)

i2c = board.I2C()
"""
ToF SHUT PIN SETUP
"""
shut_1 = digitalio.DigitalInOut(board.D5 )
shut_2 = digitalio.DigitalInOut(board.D6 )
shut_3 = digitalio.DigitalInOut(board.D26)
shut_1.switch_to_output(value=False)
shut_2.switch_to_output(value=False)
shut_3.switch_to_output(value=False)
```

```
"""
ToF SETUP and ADDRESS REASSIGNMENT
"""
#sensor 1
shut_1.value       = True
tof_1              = adafruit_vl53l1x.VL53L1X(i2c)
tof_1.distance_mode = 2
tof_1.timing_budget = 100
tof_1.set_address(0x28)
tof_1.start_ranging()
#sensor 2
shut_2.value       = True
tof_2              = adafruit_vl53l1x.VL53L1X(i2c)
tof_2.distance_mode = 2
tof_2.timing_budget = 100
tof_2.set_address(0x38)
tof_2.start_ranging()
#sensor 3
shut_3.value       = True
tof_3              = adafruit_vl53l1x.VL53L1X(i2c)
tof_3.distance_mode = 2
tof_3.timing_budget = 100
tof_3.set_address(0x48)
tof_3.start_ranging()
"""
Pi Cam Setup
"""
p=Picamera2()

WIDTH = 1280
HEIGHT = 720

p.preview_configuration.main.size=(WIDTH,HEIGHT)
p.preview_configuration.main.format='RGB888'
p.preview_configuration.align()
p.configure('preview')

p.start()
"""
FUNCTIONS
"""


#* empty function
def empty(a):
    pass

#* draws shape and bounding box around a countour of sufficient area
def getContours(img, imgContour, clr):
    contours, hierarchy = cv2.findContours(img, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_NONE)

    count = 0
    pillars = []
    correct_directions = []

    # draw vert. lines
    cv2.line(imgContour, (int(WIDTH/3), 0), (int(WIDTH/3), HEIGHT), (0, 255, 255), 1)
    cv2.line(imgContour, (int(2*WIDTH/3), 0), (int(2*WIDTH/3), HEIGHT), (0, 255, 255), 1)

    cv2.line(imgContour, ( 291, 720), (679, 419), (0, 255, 255), 1)
    cv2.line(imgContour, (1110, 720), (679, 419), (0, 255, 255), 1)
```

```
 for cnt in contours:
 area = cv2.contourArea(cnt)
 area_thresh = cv2.getTrackbarPos("AreaThresh", "Parameters")
 if area >= area_thresh:
 count += 1

 cv2.drawContours(imgContour, cnt, -1, (255, 0, 255), 2)

 peri = cv2.arcLength(cnt, True)
 appx = cv2.approxPolyDP(cnt, 0.02*peri, True)

 x, y, w, h = cv2.boundingRect(appx) # x: x-coordinate of top left corner of bb, y: y-coordinate of top left corner
of bb, w: width, h: height
        if y > 355:
            coords = [(x, y), (x+w, y), (x, y+h), (x+w, y+h)]
            pillars.append(coords)
            drawn = False

            for i,j in coords:
                if i < (2*WIDTH/3) and clr == 'green':
                    cv2.rectangle(imgContour, (x, y), (x+w, y+h), (0, 0, 255), 3)
                    drawn = True
                elif i > (WIDTH/3) and clr == 'red':
                    cv2.rectangle(imgContour, (x, y), (x+w, y+h), (0, 0, 255), 3)
                    drawn = True
            if drawn == False:
                cv2.rectangle(imgContour, (x, y), (x+w, y+h), (0, 255, 0), 3)

            # area is inversely proportional to square of dist
            # at 30cm dist -> area = 64000
            dist = 30/math.sqrt(area/32100)

            cv2.putText(imgContour, "x: " + str(x) + ', y: ' + str(y) + ', w: ' + str(w) + ', h: ' + str(h), (x, y - 70),
cv2.FONT_HERSHEY_COMPLEX, 0.7, (0, 255, 0), 2)

            asdfg = (0, 0, 255) if (((((137*(x+w))+(177*(y+h))-167362>0) or (x+w > 679.6))and(clr == 'red')) or (((-192*
(x+w))+(275*(y+h))+15120>0) or (x+w < 679.6))and(clr=='green'))) else (0, 255, 0)
            correct_dir = True if asdfg == (0, 255, 0) else False
            correct_directions.append(correct_dir)
            cv2.putText(imgContour, "Area" + str(int(area)), (x, y - 45), cv2.FONT_HERSHEY_COMPLEX, 0.7, asdfg, 2)
            cv2.putText(imgContour, "Dist" + str(int(dist)), (x, y - 20), cv2.FONT_HERSHEY_COMPLEX, 0.7, asdfg, 2)

    return count, pillars, correct_directions
"""
LOAD MASK DATA
"""
with open('green_mask.txt', 'r') as gm:
    g_hmin, g_hmax, g_smin, g_smax, g_vmin, g_vmax = gm.read().split(' ')

with open('red_mask.txt', 'r') as rm:
    r_hmin, r_hmax, r_smin, r_smax, r_vmin, r_vmax = rm.read().split(' ')

first_round = True
"""
TRACKBARS SETUP
"""
cv2.namedWindow("Green HSV dials")
cv2.resizeWindow("Green HSV dials", 640, 270)
cv2.createTrackbar("HUE Min", "Green HSV dials", int(g_hmin), 179, empty)
cv2.createTrackbar("HUE Max", "Green HSV dials", int(g_hmax), 179, empty)
cv2.createTrackbar("SAT Min", "Green HSV dials", int(g_smin), 255, empty)
```

```
cv2.createTrackbar("SAT Max", "Green HSV dials", int(g_smax), 255, empty)
cv2.createTrackbar("VAL Min", "Green HSV dials", int(g_vmin), 255, empty)
cv2.createTrackbar("VAL Max", "Green HSV dials", int(g_vmax), 255, empty)
cv2.createTrackbar("Start detection", "Green HSV dials", 1, 1, empty)

cv2.namedWindow("Red HSV dials")
cv2.resizeWindow("Red HSV dials", 640, 270)
cv2.createTrackbar("HUE Min", "Red HSV dials", int(r_hmin), 179, empty)
cv2.createTrackbar("HUE Max", "Red HSV dials", int(r_hmax), 179, empty)
cv2.createTrackbar("SAT Min", "Red HSV dials", int(r_smin), 255, empty)
cv2.createTrackbar("SAT Max", "Red HSV dials", int(r_smax), 255, empty)
cv2.createTrackbar("VAL Min", "Red HSV dials", int(r_vmin), 255, empty)
cv2.createTrackbar("VAL Max", "Red HSV dials", int(r_vmax), 255, empty)
# Initial time
last_time = time.time()

# Loop
while True:
    # Image
    img = p.capture_array()

    # Converting the color from BGR to HSV
    imgHSV = cv2.cvtColor(img, cv2.COLOR_RGB2HSV)

    # Creating mask
    g_hmin = cv2.getTrackbarPos("HUE Min", "Green HSV dials")
    g_hmax = cv2.getTrackbarPos("HUE Max", "Green HSV dials")
    g_smin = cv2.getTrackbarPos("SAT Min", "Green HSV dials")
    g_smax = cv2.getTrackbarPos("SAT Max", "Green HSV dials")
    g_vmin = cv2.getTrackbarPos("VAL Min", "Green HSV dials")
    g_vmax = cv2.getTrackbarPos("VAL Max", "Green HSV dials")

    r_hmin = cv2.getTrackbarPos("HUE Min", "Red HSV dials")
    r_hmax = cv2.getTrackbarPos("HUE Max", "Red HSV dials")
    r_smin = cv2.getTrackbarPos("SAT Min", "Red HSV dials")
    r_smax = cv2.getTrackbarPos("SAT Max", "Red HSV dials")
    r_vmin = cv2.getTrackbarPos("VAL Min", "Red HSV dials")
    r_vmax = cv2.getTrackbarPos("VAL Max", "Red HSV dials")

    lower_green = np.array([g_hmin, g_smin, g_vmin])
    upper_green = np.array([g_hmax, g_smax, g_vmax])

    lower_red = np.array([r_hmin, r_smin, r_vmin])
    upper_red = np.array([r_hmax, r_smax, r_vmax])
    # Create mask and perform bitwise operations
    mask_green = cv2.inRange(imgHSV, lower_green, upper_green)
    mask_red = cv2.inRange(imgHSV, lower_red, upper_red)

    imgContour = img.copy()

    # Get contours
    green_count, green_pillars, green_directions = getContours(mask_green, imgContour,
'green')
    red_count, red_pillars, red_directions = getContours(mask_red, imgContour, 'red')

    total_pillars = green_count + red_count
    print(f'Pillars detected: {total_pillars}')

    cv2.imshow("Contours", imgContour)
    key = cv2.waitKey(1)
```

```
if key == 27: # Escape key
break

# Cleanup
cv2.destroyAllWindows()
GPIO.cleanup()
```

- This code too requires 2 text documents in the same folder for the code to function , they are : red_mask.txt and green_mask.txt with items as 4 172 145 247 105 255 and 56 85 82 239 71 255 respectively.

- Virtual Environment is employed here for the code to function

References

- https://learn.adafruit.com/adafruit-vl53l1x/python-circuitpython
- https://learn.sparkfun.com/tutorials/qwiic-distance-sensor-vl53l1x-hookup-guide/python-examples
- https://docs.sunfounder.com/projects/raphael-kit/en/latest/appendix/install_the_libraries.html#create-virtual
- https://docs.sunfounder.com/projects/raphael-kit/en/latest/appendix/i2c_configuration.html#i2c-config
- https://www.youtube.com/watch?v=2bganVdLg5Q&t=42s
- https://github.com/World-Robot-Olympiad-Association/wro2022-fe-template
- https://www.youtube.com/playlist?list=PLGs0VKk2DiYyXlbJVaE8y1qr24YldYNDm