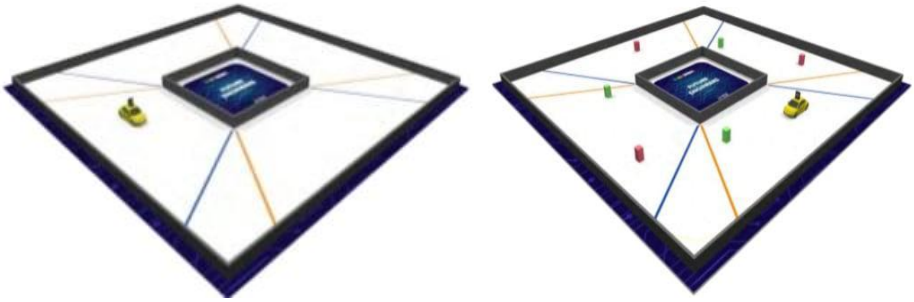
The Strategy

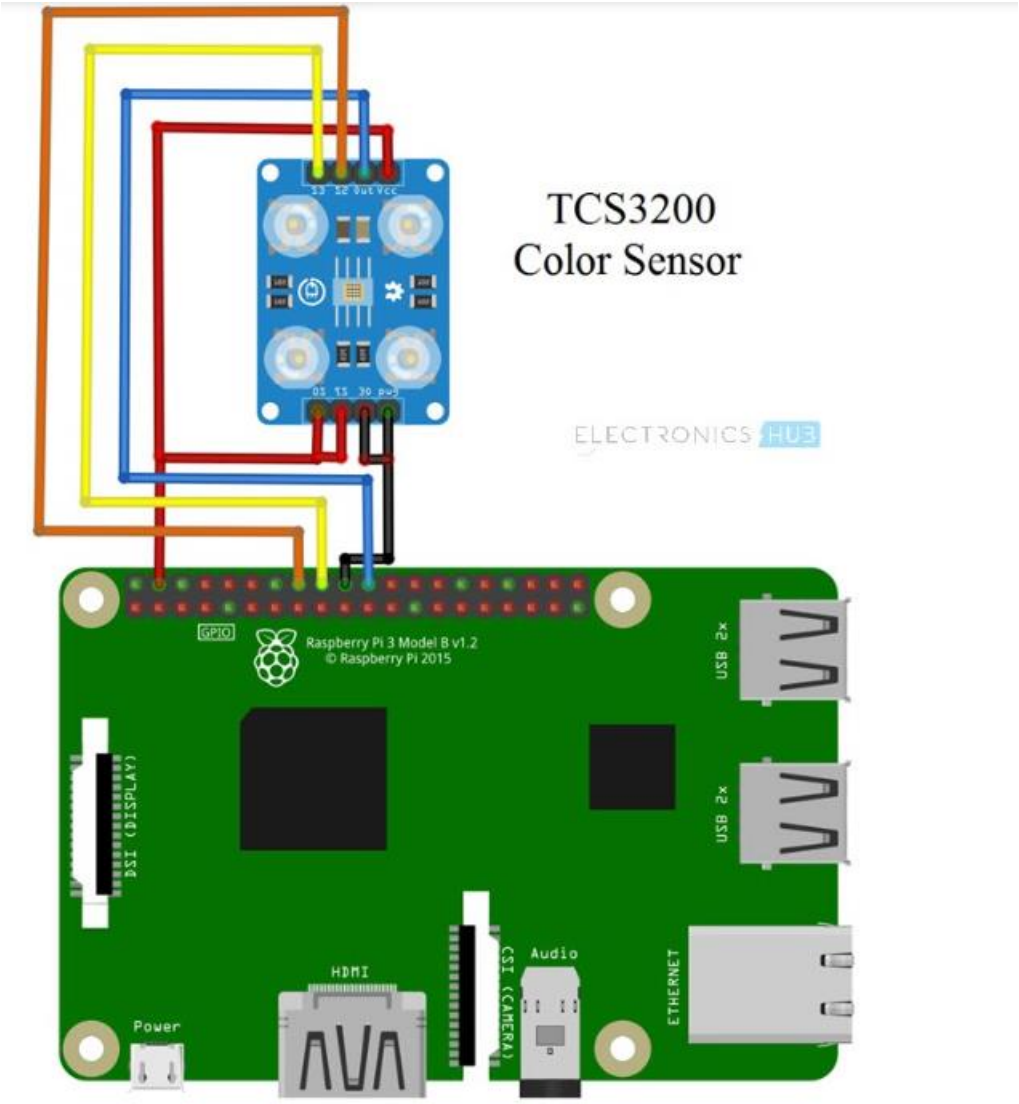
We, as the Future Saviors team, have built and programmed a self-driving car capable of completing two stages. In the first stage, it starts driving from a specific location, completes three laps, and stops at the same point where it started. We achieved this by using a color sensor, which we programmed. Based on the track, which has blue lines, the car completes three full laps after crossing 12 blue lines.

As for the second stage, the car also completes three laps, but there are cubes with red and green colors on the track. The car must pass the green cube on the left and the red one on the right. We utilized a camera sensor and programmed it using computer vision. Additionally, we used software libraries and algorithms to support these functionalities. In both stages, the robot needs to avoid obstacles, and to achieve this, we used an ultrasonic
sensor



(1)Stop_The_Car_after_3turn strategy :

Circuit :



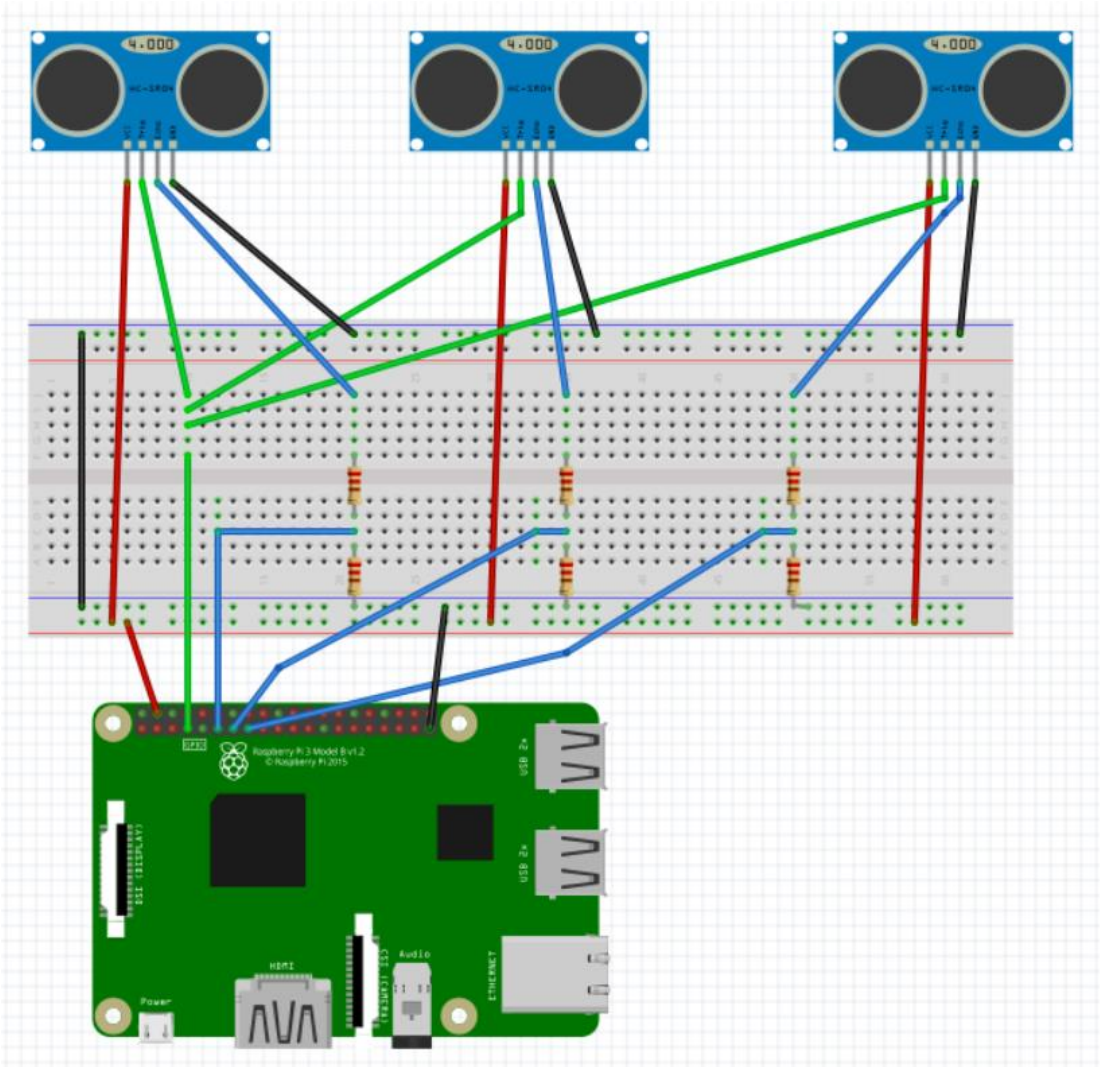
Code:

```
# color sensor
# Define the GPIO pins for  S2, S3, and OUT

S2 = 24
S3 = 25
OUT = 23

# Set up GPIO mode and configure pins
GPIO.setmode(GPIO.BCM)
GPIO.setup(S2, GPIO.OUT)
GPIO.setup(S3, GPIO.OUT)
GPIO.setup(OUT, GPIO.IN)

# Configure scaling (adjust as needed)
# GPIO.output(S0, GPIO.LOW)
# GPIO.output(S1, GPIO.HIGH)
```

```
blue_count = 0

def read_color():
    # Set S2 and S3 to select the color sensitivity (adjust as needed)
    GPIO.output(S2, GPIO.LOW)
    GPIO.output(S3, GPIO.LOW)

    # Wait for stabilization (adjust as needed)
    time.sleep(0.1)

    # Read the frequency
    frequency = 0
    for _ in range(5):  # Read 5 times and average
        frequency += GPIO.input(OUT)
        time.sleep(0.1)
    frequency /= 5.0

    # Determine the color based on the frequency (calibration may be
required)
    if 22 < frequency < 30:  # Adjust these values for your specific
setup
        return "Red"
    elif 31 < frequency < 40:
        return "Green"
    elif 41 < frequency < 50:
        return "Blue"
    else:
        return "Unknown"

try:
    while True:
        color = read_color()
        if color == "Blue":
            blue_count += 1
            print("Blue detected! Count:", blue_count)
            if blue_count==12:
                break
        time.sleep(1)

except KeyboardInterrupt:
    pass

# Cleanup GPIO
GPIO.cleanup()
```

(2) Avoid obstacles strategy :

Circuit :

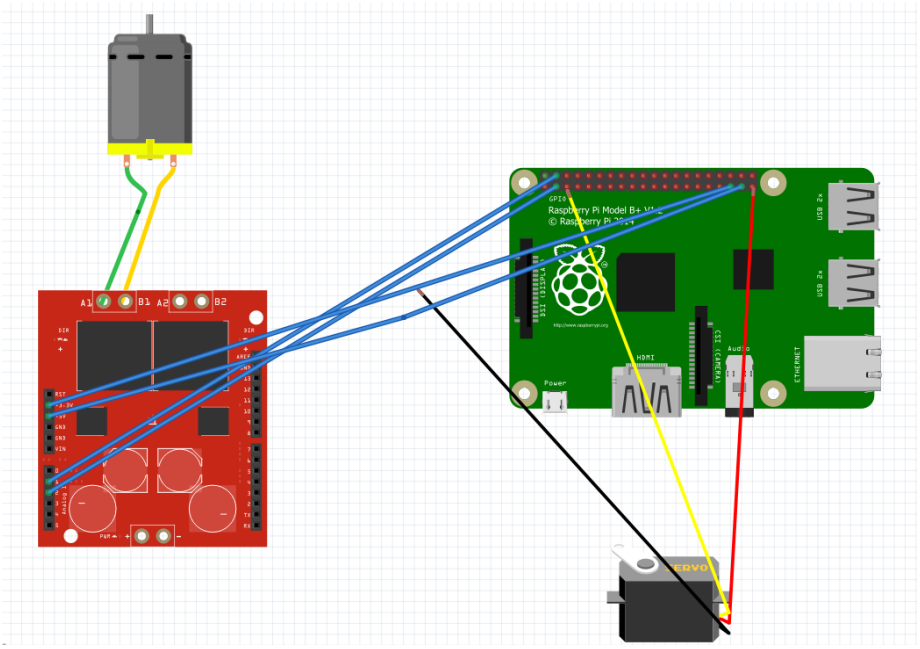


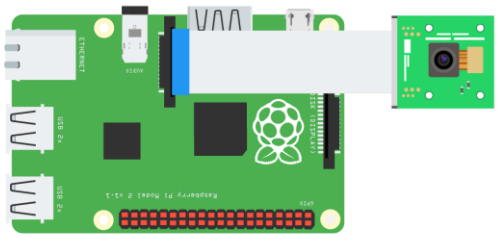
Code:

```
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO_TRIGGER = 4
GPIO.setup(GPIO_TRIGGER, GPIO.OUT)

ECHOS = {"echo_right": 17, "echo_forward": 27, "echo_left": 22}
for e in ECHOS:
  GPIO.setup(ECHOS[e], GPIO.IN)

def distance(e):
  new_reading = False
  counter = 0
  GPIO.output(GPIO_TRIGGER, True)
  time.sleep(0.00001)
```

```
  GPIO.output(GPIO_TRIGGER, False)

  while GPIO.input(e) == 0:
    pass
    counter += 1
    if counter == 5000:
      new_reading = True
      break
  start_time = time.time()

  if new_reading:
    return False

  while GPIO.input(e) == 1:
    pass
  end_time = time.time()

  time_elapsed = end_time - start_time
  dist = (time_elapsed * 34300) / 2
  return dist


def run_distance():
    sensors = {}
    for e in ECHOS:
        dist = distance(ECHOS[e])
        sensors[e] = dist
        time.sleep(1)

    if sensors["echo_left"] < DISTANCE_THRESHOLD:
        turn_rigth()
    elif sensors["echo_right"] < DISTANCE_THRESHOLD:
        turn_left()
    elif sensors["echo_forward"] < DISTANCE_THRESHOLD:
        turn_left()
    else:
        move_forward()
    return sensors

if __name__ == "__main__":
    DISTANCE_THRESHOLD = 20
```

(1)Avoid_colored_cube strategy :
Circuit :





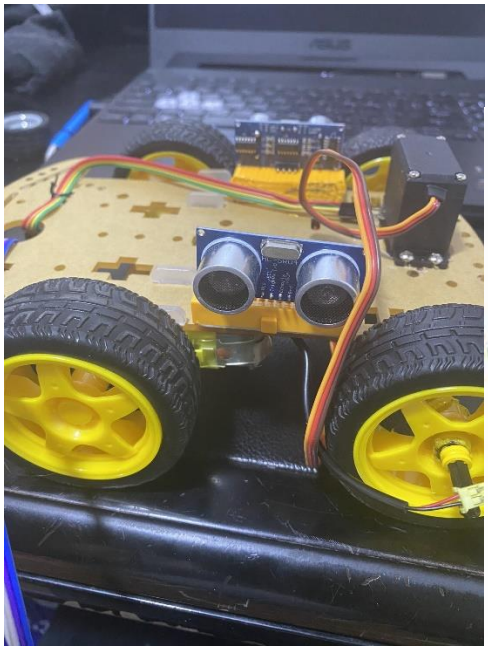
Code :

```
import sys
import time
import RPi.GPIO as GPIO
import cv2
import numpy as np
from time import sleep

# control car movement
# ###################################

# GPIO Pins for controlling the car
motor_pin1 = 26
motor_pin2 = 20
servo_pin=16

# Set up GPIO
GPIO.setmode(GPIO.BCM)
```

```
GPIO.setup(motor_pin1, GPIO.OUT)
GPIO.setup(motor_pin2, GPIO.OUT)
GPIO.setup(servo_pin, GPIO.OUT)


# Create a function to control the car's movement
def move_forward():
    GPIO.output(motor_pin1, GPIO.HIGH)
    GPIO.output(motor_pin2, GPIO.HIGH)

def stop():
    GPIO.output(motor_pin1, GPIO.LOW)
    GPIO.output(motor_pin2, GPIO.LOW)

#Right and Left control
# ###################################

# Set the duty cycle for left and right positions
left_position = 2.5  # Adjust this value for your servo
right_position = 12.5  # Adjust this value for your servo
 # Set the PWM frequency (Hz)
pwm_frequency = 50
# Create a PWM instance with the specified frequency
pwm = GPIO.PWM(servo_pin, pwm_frequency)
# Start the PWM signal with 0 duty cycle (center position)
pwm.start(7.5)

def turn_rigth():

     # Turn the servo to the right position
    pwm.ChangeDutyCycle(right_position)
    time.sleep(1)  # Wait for 1 second
    pwm.stop()
    GPIO.cleanup()

def turn_left():

    # Turn the servo to the left position
    pwm.ChangeDutyCycle(left_position)
    time.sleep(1)  # Wait for 1 second
    pwm.stop()
    GPIO.cleanup()


# camera code to avoid colored cube
# ###################################
# Create a VideoCapture object for the Pi Camera
cap = cv2.VideoCapture(0)
```

```
while True:
    ret, frame = cap.read()

    if not ret:
        break

    # Convert the frame to HSV color space
    hsv_frame = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Define the range of green color in HSV
    lower_green = np.array([40, 40, 40])
    upper_green = np.array([80, 255, 255])

    # Define the range of red color in HSV
    lower_red = np.array([0, 100, 100])
    upper_red = np.array([10, 255, 255])

    # Threshold the frame to get only green and red colors
    green_mask = cv2.inRange(hsv_frame, lower_green, upper_green)
    red_mask = cv2.inRange(hsv_frame, lower_red, upper_red)

    # Find contours in the masks
    green_contours, _ = cv2.findContours(green_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    red_contours, _ = cv2.findContours(red_mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    # Check if there are green and red objects
    if len(green_contours) > 0:
        turn_left()
    elif len(red_contours) > 0:
        turn_rigth
    else:
        # Continue moving forward if neither green nor red is detected
        move_forward()

    # Display the frame
    cv2.imshow("Car Camera", frame)

    # Break the loop if 'q' is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the VideoCapture and GPIO
cap.release()
cv2.destroyAllWindows()
GPIO.cleanup()
```

Ultrasonic sensor:

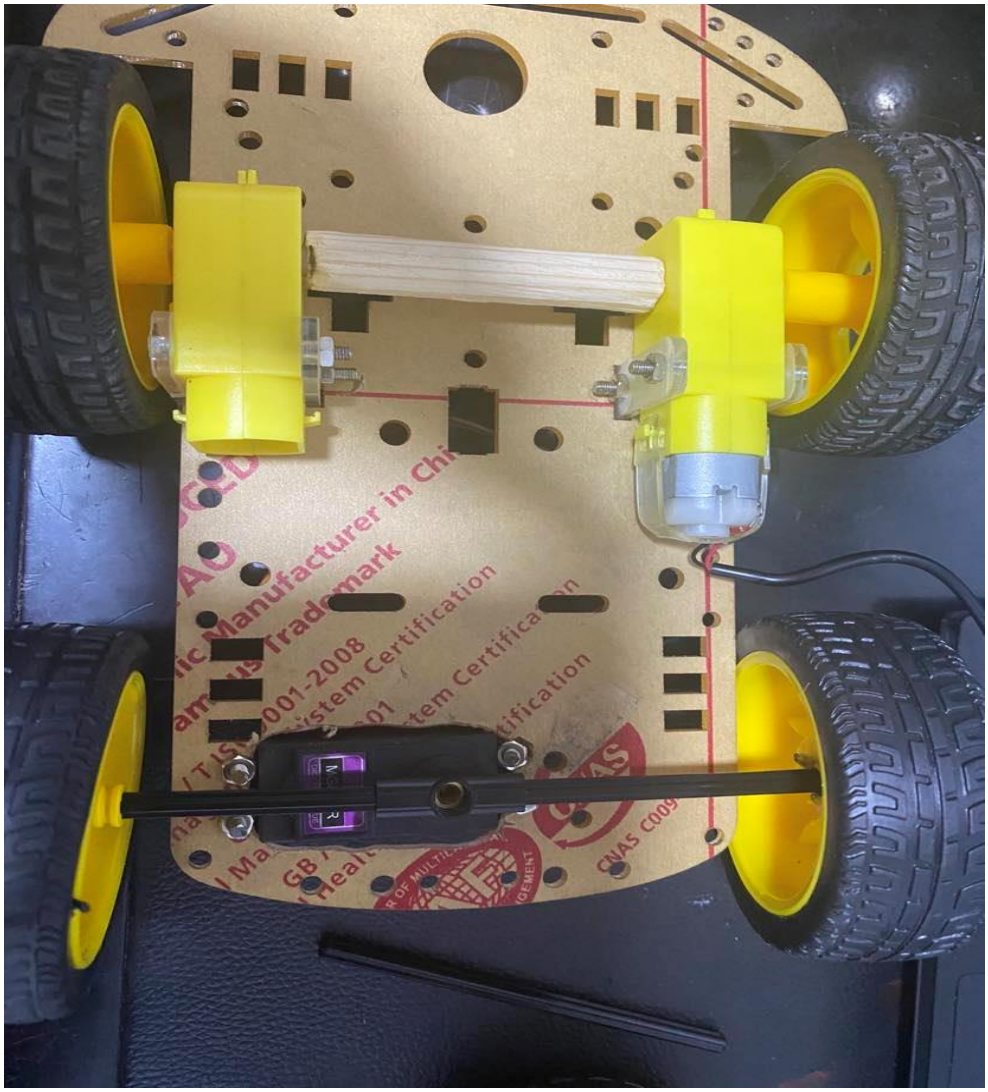


We used an ultrasonic sensor of the type HC-SR04 and connected it to the Raspberry Pi using the bread board .
We connected the sensors to the GPIO pins.

The controller used:

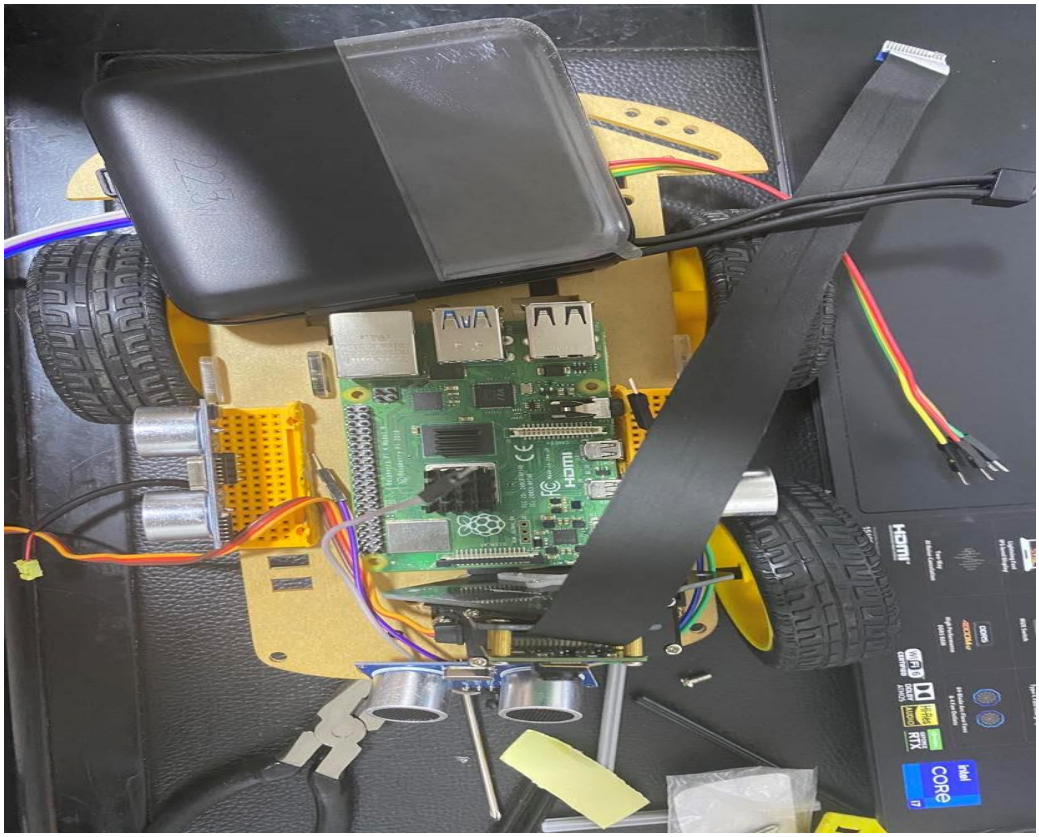
We used a Raspberry Pi controller and connected it to the power bank and took a voltage of 5 volts from it.

The motor used:



We used one DC motor.

Baseus opow power
Bank:



It has a small design, weighing 190 grams, and its battery capacity is very high and it charges
quickly

.

.The exterior is made of a composite material of ABS and PC and its wire is hidden inside the battery and is made of durable
plastic

.

It has 4 charging ports, which helps charge 4 different devices at the same
time
-١usb(٢)
-٢c (port)
-٣c (wire)
-٤lightning (wire)

It has an LED indicator that shows the percentage of remaining battery, and there is a smart charging chip and a temperature reducer when charging, which extends the life of the battey and maintains the safety of the
user

The current value is high, equivalent to 3 A, and it contains a 2-in-1 charging cable
It has a large capacity equivalent to 10,000 mAh, and it has a digital display screen to show t
the remaining charge
percentage