# Term Project – Robot Car - Wayno

Kiran Poudel

Nhan Nguyen

Fernando Abel Malca Luque

Yuvraj Gupta

**Github: *nhannguyensf***

## Task Description:

This project involves designing and building a robot car powered by a **Raspberry Pi**. The goal is to implement **line-following functionality** using line sensors and integrate **obstacle detection and avoidance** using ultrasonic echo sensors and RGB sensors to stop on the red line. The car operates autonomously, detecting lines on the ground to navigate paths and avoiding obstacles in its path.

## Approach / What We Did:

During our first meeting, we met with each other to decide the structure and design of our robot car. We decided on the components we would use for each scenario, the number of components, and the algorithm approach for each component to complete a task. We originally had a triple-layered car, however, the weight of every component added to each layer would significantly lower the car's speed as the motors weren't fast enough to ignore the overall weight of the car. Due to this problem, we decided to go with a two-layered car which made the car go at a much faster speed.

Our intention for the robot car was that it would follow a certain order of operations. Using the sensors at the front, it would check and follow a black line when detected using the PID algorithm to calibrate itself during "line-following" mode. There were 5 sensors at the front of the car, each with a 2 cm distance from each other, for a more precise error calibration. We prioritized (P) to have a more accurate position adjustment while on the line, and occasionally adjusted (I) and (D) for more fine-tuning and stability during turns.

After following the line, it would then detect if there was an obstacle in front of it using an ultrasonic sensor. We used five sensors, three at the front and one on each side of the car, heavily prioritizing the front-center sensor, to detect an obstacle in front of us at a distance of 30cm. We would then turn on "obstacle-maneuvering" mode to go around the obstacle through a series of steps: turning counterclockwise and going forward for a certain amount of time and speed, then turning clockwise and going forward, and finally, it would turn clockwise and go forward again.

After it was done with this process, it would try to detect a black line again to continue

following it. The intention for the RGB sensor was to stop the loop the main process was running when it detected something "red".

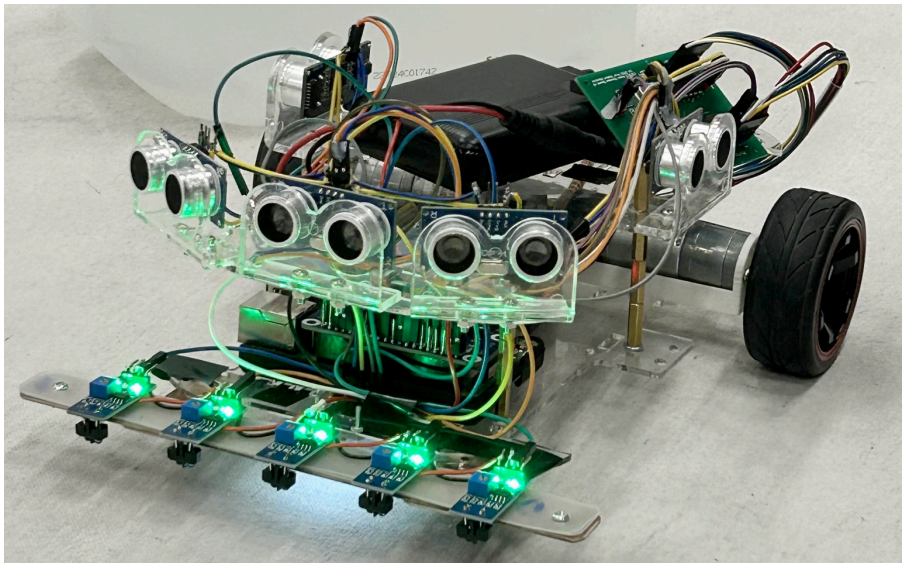
## Building the Robot (include photos)

The robot was constructed step by step:

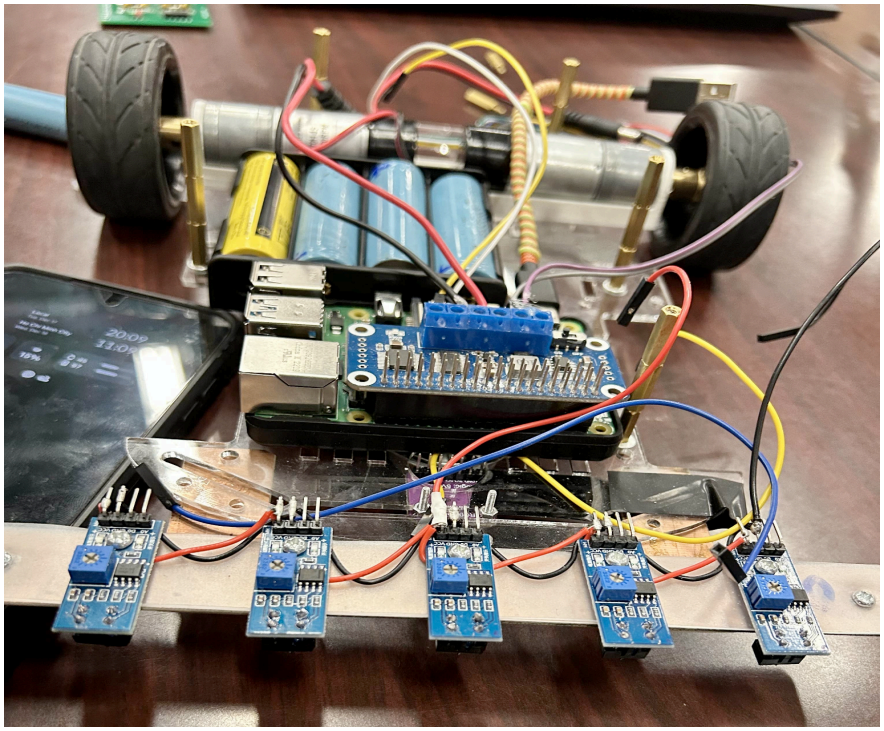
1. **Chassis Assembly**: Mounted the motors and wheels to the robot chassis.
2. **Motor Encoder**: Connected the motors to the LS7336 Quadrature Encode chip to record the speed.
3. **Line Sensors**: Attached line sensors at the front base of the robot to detect lines.
4. **Echo Sensors**: Positioned ultrasonic sensors to detect obstacles (3 in the front, 1 on the left, 1 on the right).
5. **Raspberry Pi Integration**: Mounted the Raspberry Pi securely on the chassis and wired the sensors/motors.
6. **Motor Hat:** Connects the motors with power and is mounted on Pi.
7. **Battery:** 4 batteries of the 3.76V were used to power up the motors.
8. **PowerBank:** A designated power supply to motors was used to make sure Pi was on.

**Photos**:

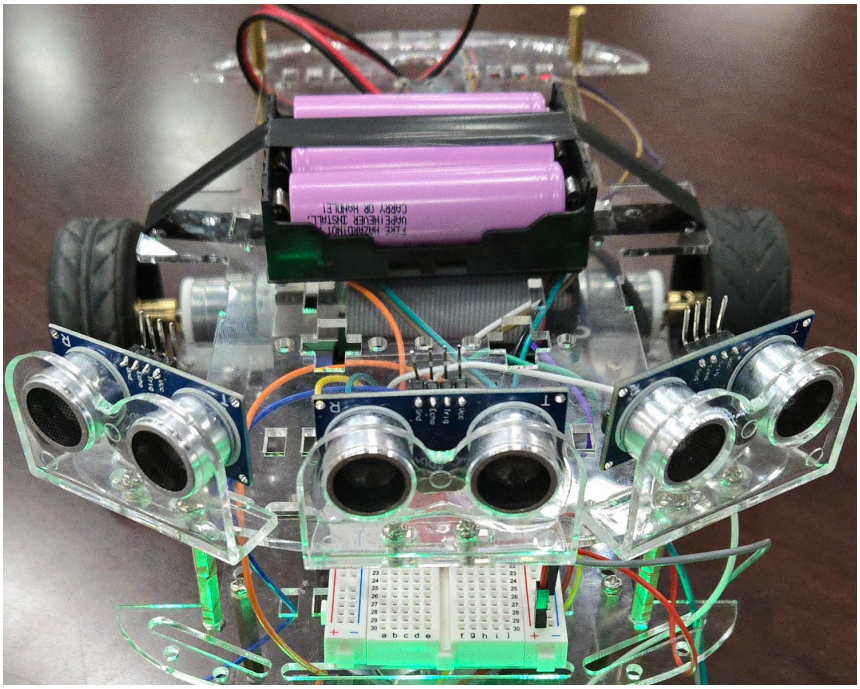
1. *Assembled Bot:*



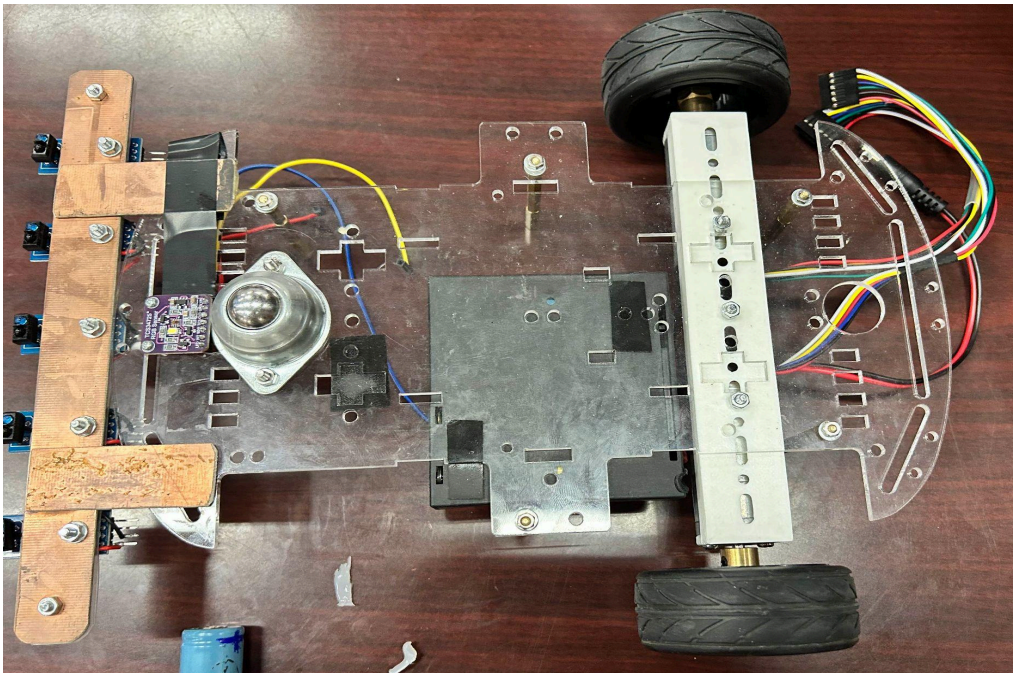
2. *Line Sensors*:



3. *Ultrasonic Sensors:*



4. *RGB Sensors:*



**Components Used (*include photo, and part numbers where applicable, such as HC-SR04 for the sonic echo sensor*):**

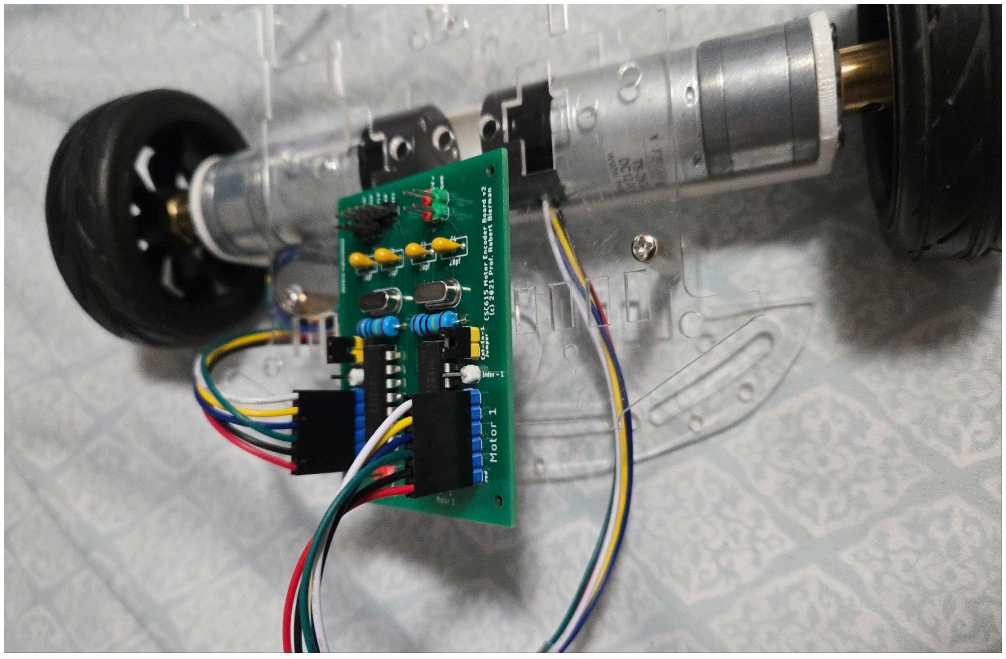
| Component | Part Number | Quantity | Purpose |
|---|---|---|---|
| Brushless Geared Motor | TS-25GA370H-4 | 2 | To drive the wheels and provide motion to the robot |
| EchoSensor | HC-SR04 | 5 | To measure the distance of obstacles for navigation and obstacle avoidance. |
| Line Sensor | TCRT5000 IR Reflective Sensor | 5 | To detect lines or edges for line-following |
| Encoder Board | Motor Encoder Board v2 | 1 | To read the RPM of the motors for speed and position control |
| Raspberry Pi | Raspberry Pi 4 | 1 | Main controller for processing data and managing the robot car |
| Motor HAT | WaveShare Motor | 1 | To control the motors by |

| | | | |
|---|---|---|---|
| | HAT | | providing power and direction signals |
| Battery (with holder) | 4-Cell 18650 Battery Holder with Wired Ends | 1 | To provide power to the robot using 4 18650 batteries |
| PowerBank | # | 1 | To power the Raspberry Pi |
| RGB Color Sensor | TCS34725 | 1 | To detect the current color the robot car was under for certain tasks |

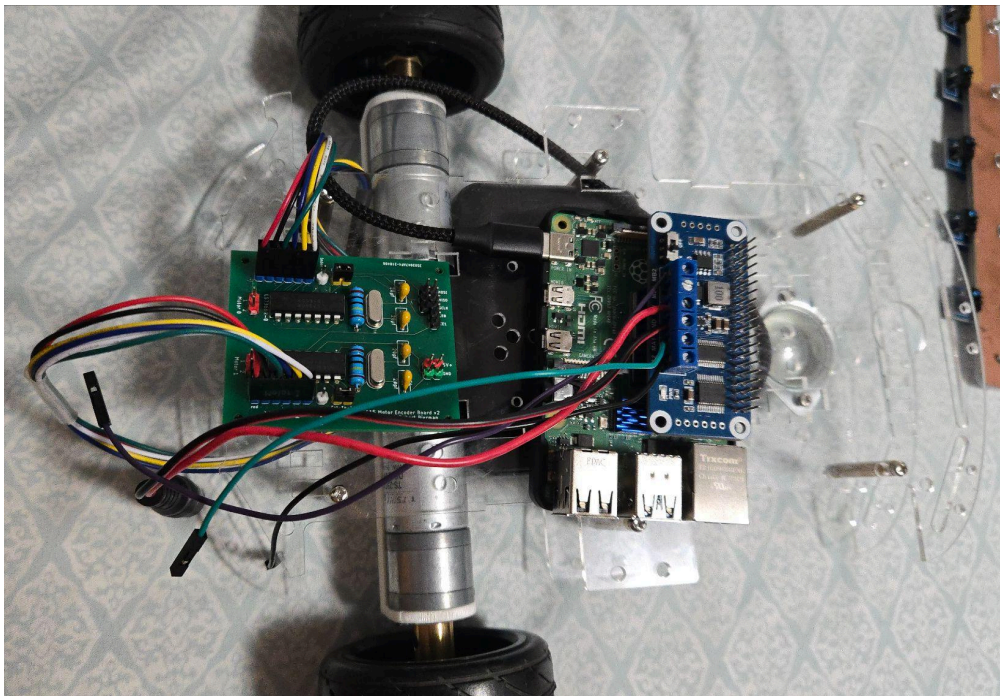
## How was the bot built (photos good to include)

The bot was built in the following steps:

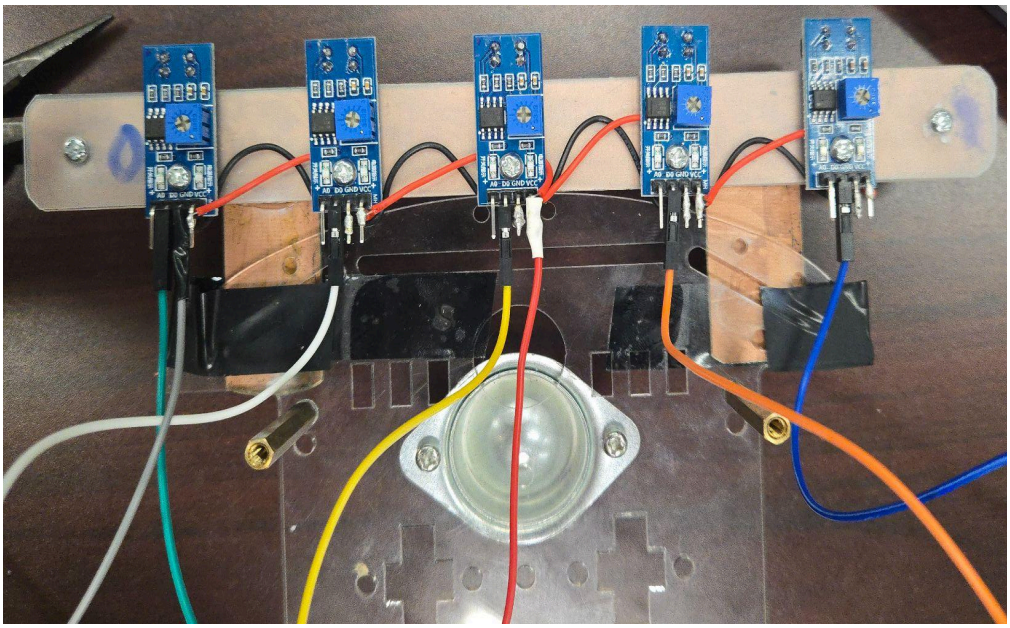
1. **Hardware Assembly**:
    - Attached the motors and wheels to the chassis.




    - Mounted the Raspberry Pi and Motor Driver HAT.

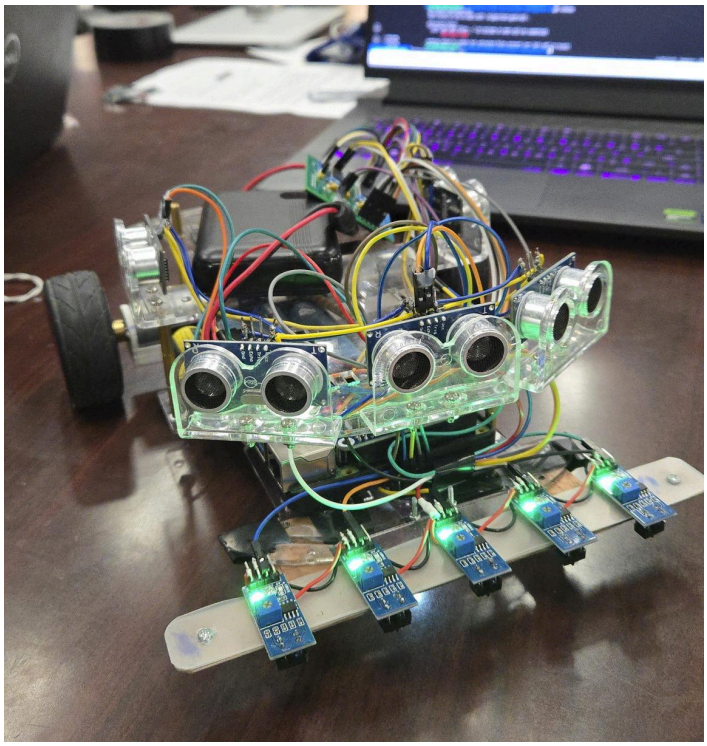


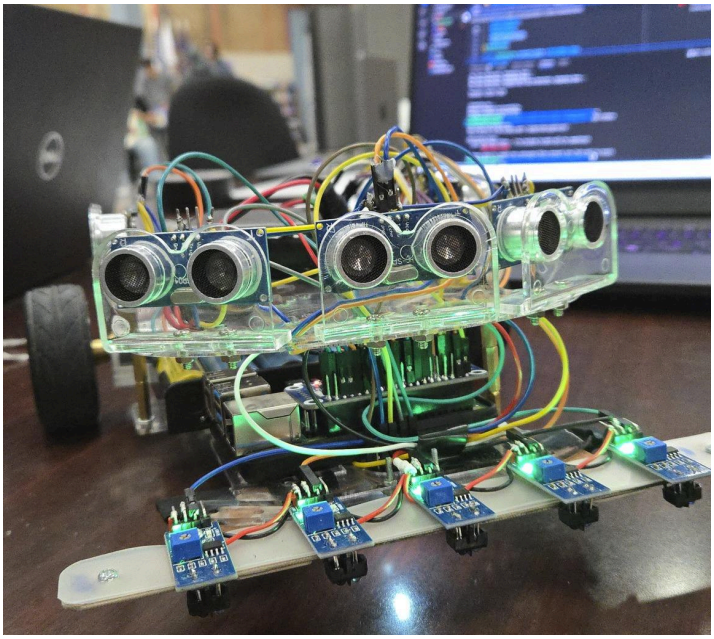
- Installed line sensors at the front and echo sensors at the appropriate positions.



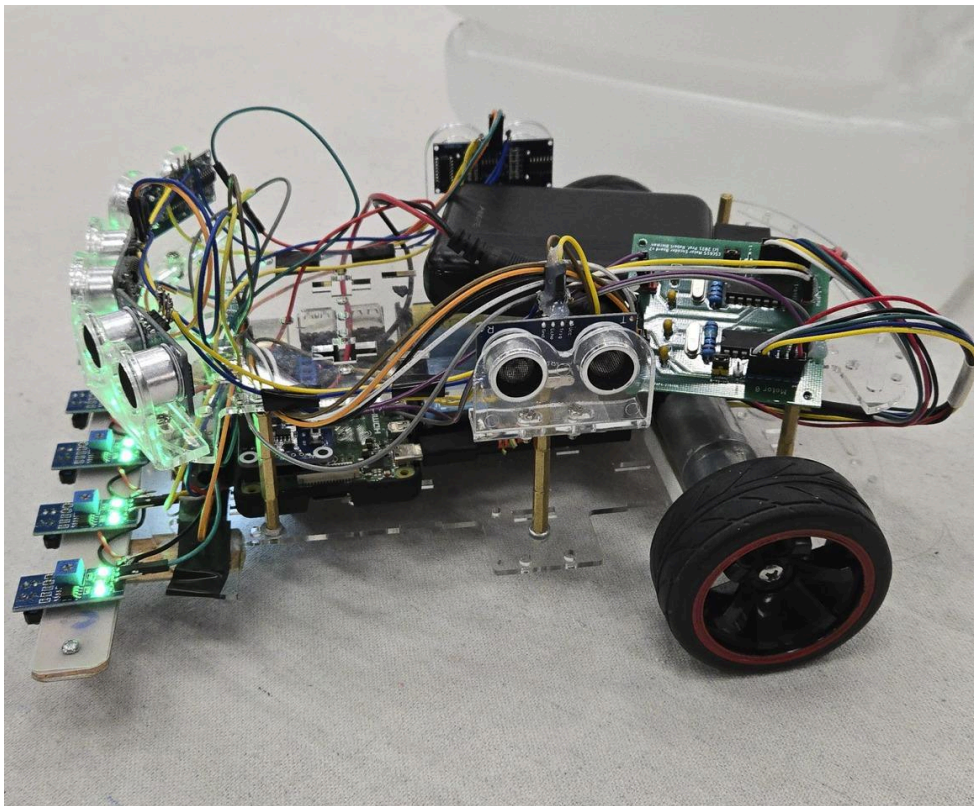
## Photos of the Car:

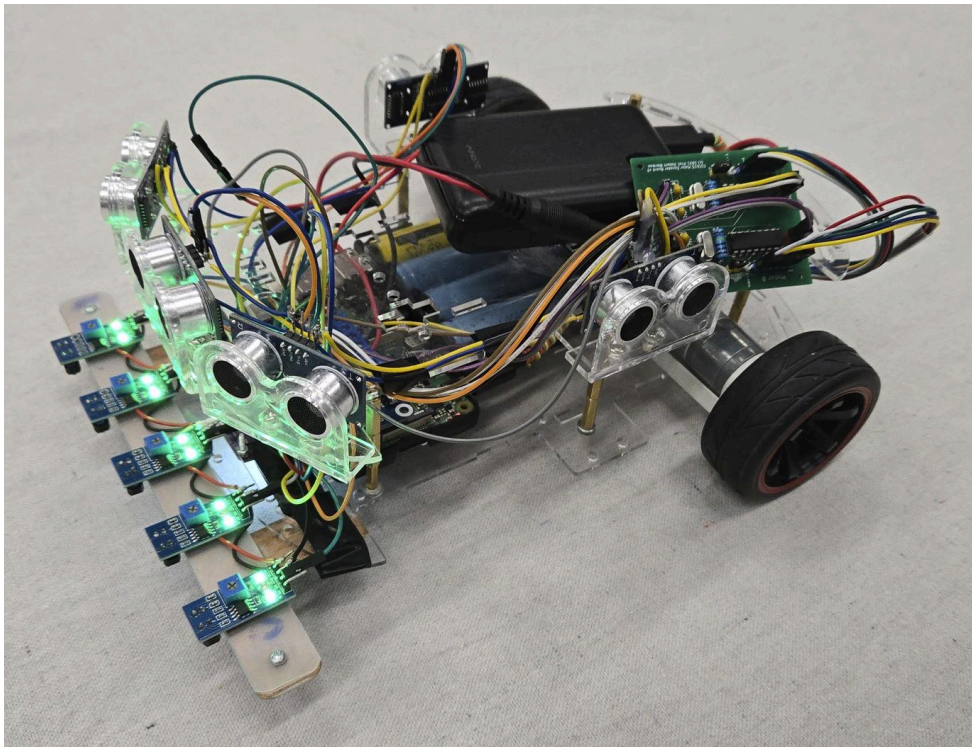
1. *Front-View:*



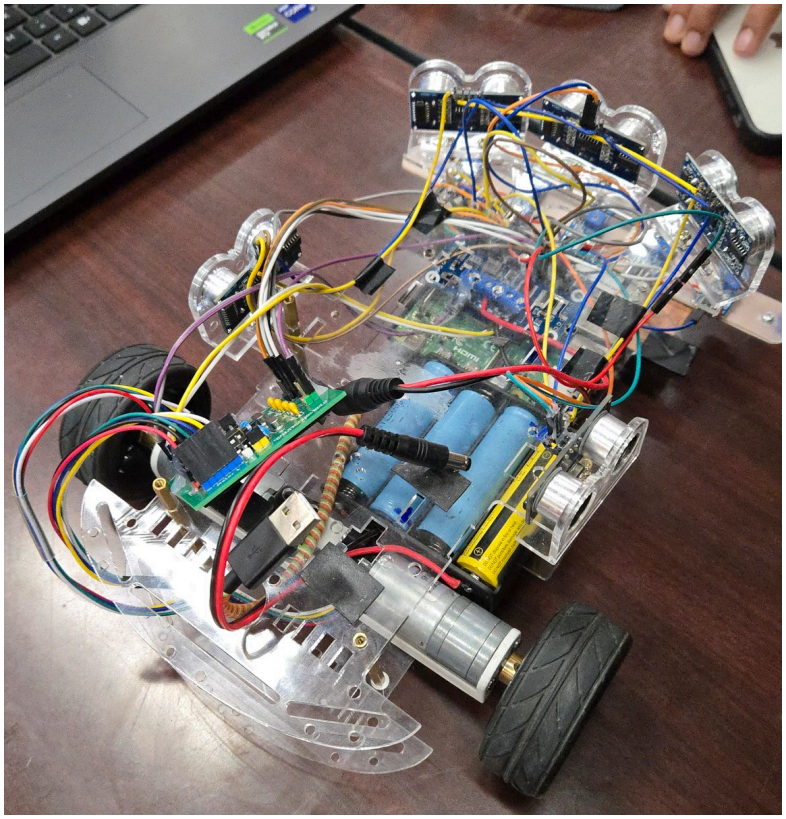


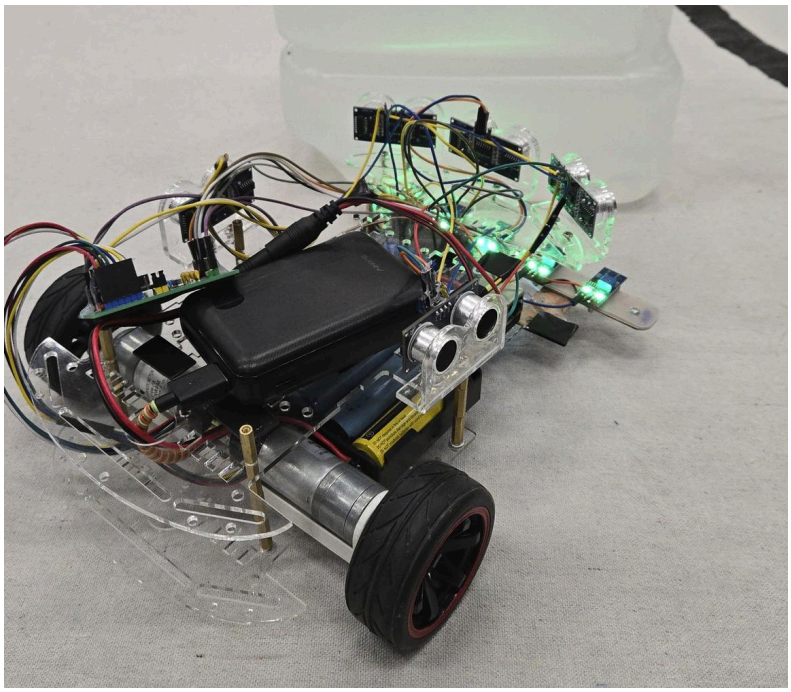
2. *Side-View:*





3. *Top-View:*





**What libraries/software did you use in your code (include full reference)**

## Libraries

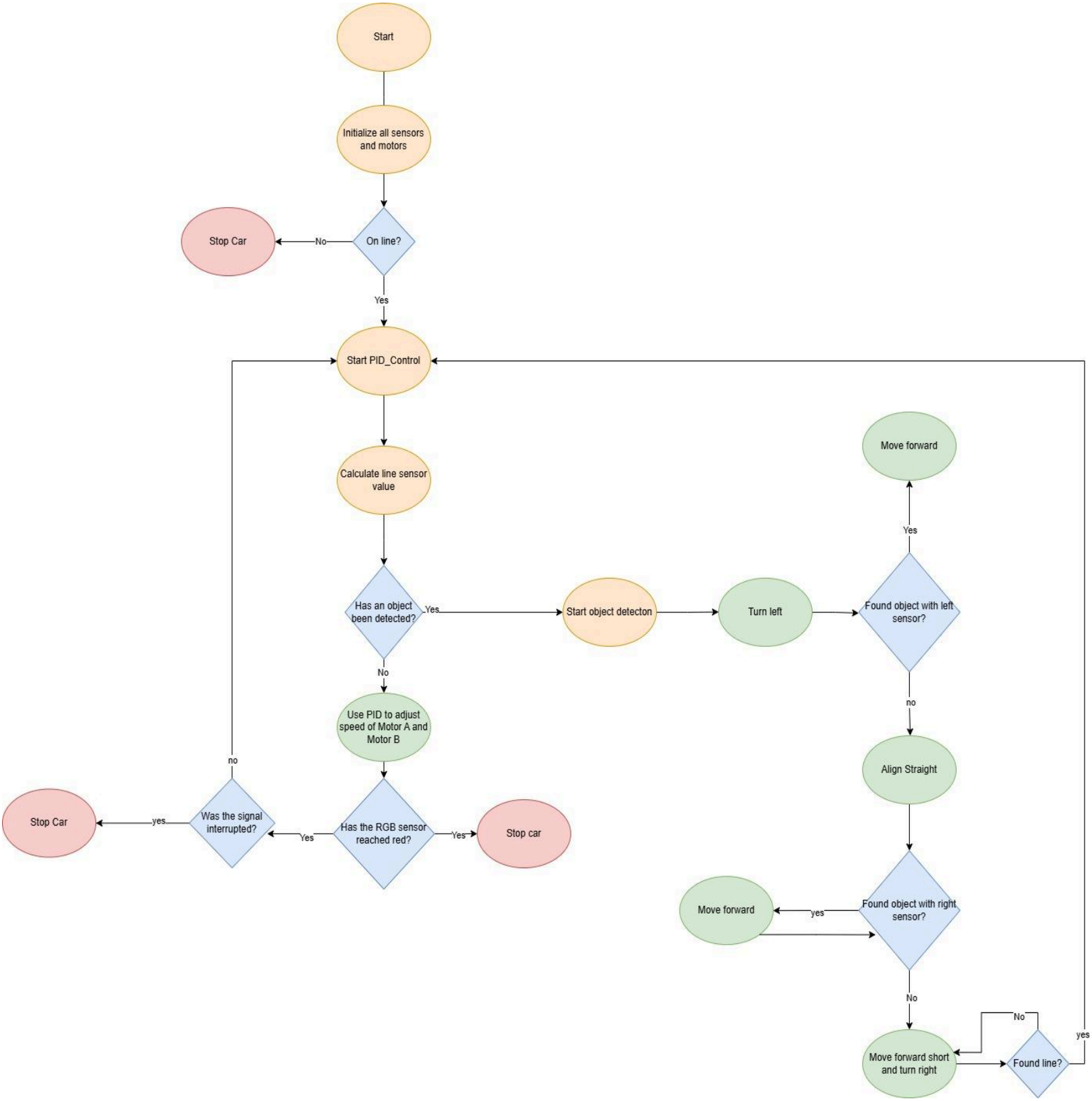
- **bcm2835**: GPIO control for the Motors in the Raspberry Pi
    - Reference: http://www.airspayce.com/mikem/bcm2835/
- **PCA9685 Library**: PWM control for the motor driver.
- **Standard C Libraries**: stdio.h, unistd.h, signal.h, time.h, etc.
- **Encoder**: lsr3776.c

  Reference: Bierman
- **pigpio**: GPIO control for the RGB sensors in the Raspberry Pi

## Software Tools

- **Raspberry Pi OS** (Linux environment).
- **GCC Compiler**: Compiling C programs.
- **Visual Studio Code**: IDE for writing and debugging code.

## Flowchart of our code:



**Wiring Table Section (with the cable colors, pins used, sensor #, etc.):**

- **Line Sensors:**

| **LINE SENSORS** | | | | | |
|---|---|---|---|---|---|
| Cable Color: | Green | White | Yellow | Orange | Blue |
| GPIO | 17 | 27 | 22 | 23 | 24 |
| #Sensor | 0 | 1 | 2 | 3 | 4 |

- **Encoder Motors:**

| **MOTOR (with Encoder)** | | | | | |
|---|---|---|---|---|---|
| Cable Color: | Orange | Brown | White | Black | White |
| GPIO: | 10 | 9 | 11 | 8 | 7 |
| Functionality: | MOSI | MISO | SCLK | CE0 | CE1 |

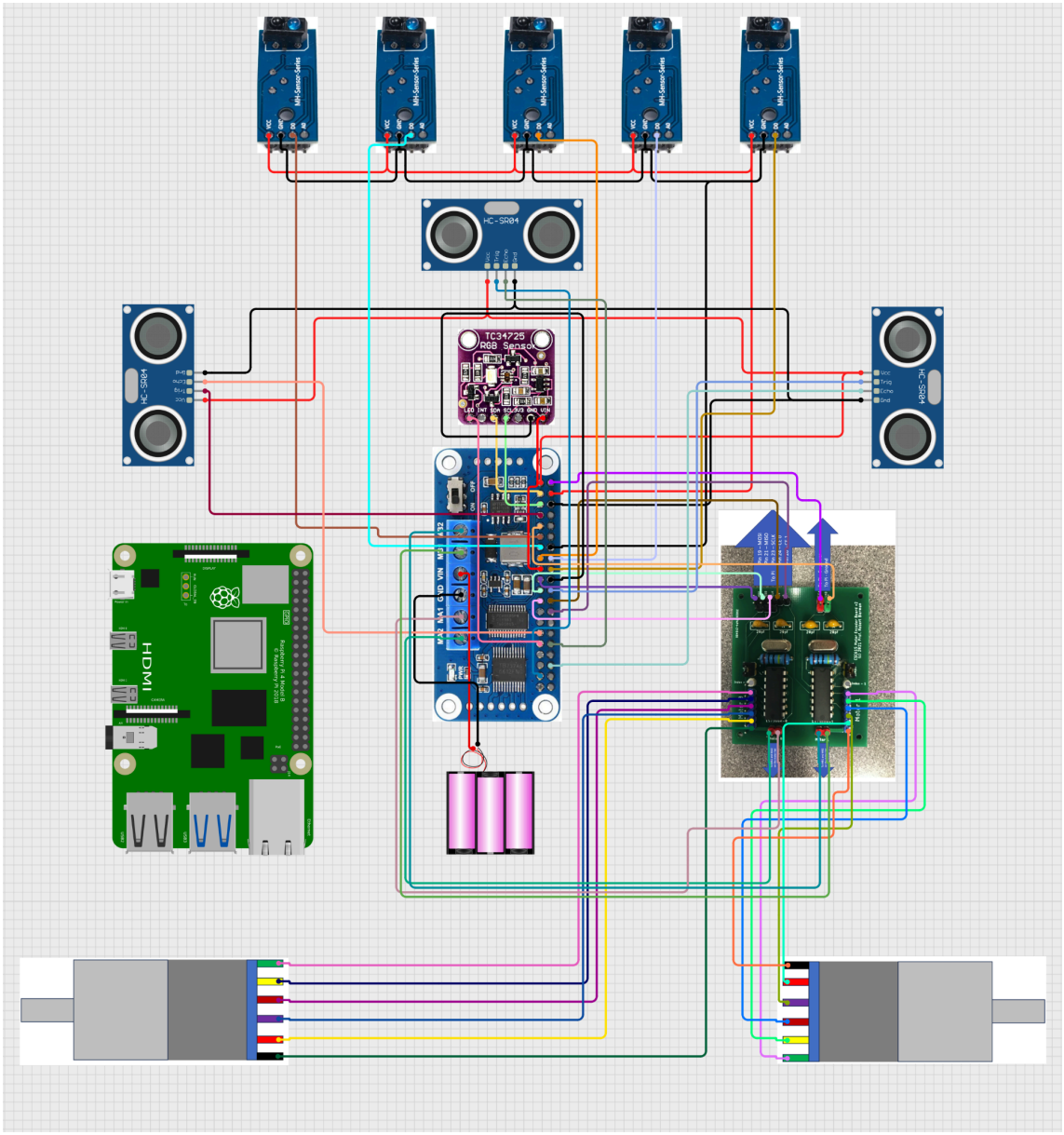
- **Echo Sensors:**

| **ULTRASONIC SENSORS** - FRONT (*TRIG* \| *ECHO*) | | | | | | |
|---|---|---|---|---|---|---|
| Cable Color: | Yellow | Brown | Blue | Orange | Orange | Green |
| GPIO | 4 | 5 | 26 | 12 | 25 | 16 |
| #Sonar | 0 | | 1 | | 2 | |
| BACK | | | | | | |

- **RGB Color Sensor:**

| **RGB SENSOR** - PINS | | | | | |
|---|---|---|---|---|---|
| Cable Color: | Green | Grey | Blue | Yellow | Brown |
| GPIO: | 3v3 | GND | 2 | 3 | 6 |
| Functionality: | VIN | GND | SDA | SCL | LED |

## Hardware Diagram:



Please find an interactive version here to see all the pin numbers:
**https://app.cirkitdesigner.com/project/9b556b1d-acc6-450d-80c5-113a936d3a9b**

***Key Connections***:

- Motors connected to the encoder board and then PCA9685 PWM outputs for motor speed and direction control.
- Line sensors (5 total) connected to Raspberry Pi GPIO pins for line detection.
- Ultrasonic sensors (3 total) connected to GPIO pins for trigger and echo operations.

## What worked well:

**Line-Following Logic:**

- Using the PID algorithm to control the motors' turning speed when detecting turns and automatically adjusting its positioning to stay on the line.

**Obstacle Detection:**

- Using the HC-SR04 sensors to identify if there's an object in front of it when in range, and do object-avoidance mode to go around an object

**Mode-Switching Between Algorithms:**

- A seamless transition between line-following and obstacle avoidance, allowed the car to easily maneuver around an obstacle and immediately start tracking the line after it was past the obstacle.

## What were issues

**Sensor Noise/Calibration**:

- The ultrasonic sensors occasionally returned noisy values, requiring filtering and careful handling of the code.
- Using three echo sensors at the front would occasionally result in faulty calibration when avoiding the obstacle, so we resolved the issue by only using one sensor at the front during final testing.

**Power Management**:

- While managing the motors and the Raspberry Pi, it was a challenge to ensure that both components had an appropriate amount of power while being under heavy load.
- We resolved this issue by using the battery pack to power the Raspberry Pi and the 4-slot battery pack to power the motors. This allowed us to separate the power usage so each component had enough power to work.

**PID Calibration**:

- Ensuring that the values defined in the PID algorithm would allow for the robot car to follow the line accurately, required a lot of adjustment but still lacked precision

**Motor Selection**:

- The gray geared motors with the hall effect sensors were considerably slower than the yellow motors, however, the yellow motors would occasionally have issues such as the wheels coming off when reaching a certain speed.
- To resolve this issue, we decided to sacrifice speed for precision and just used the gray-gated motors.

**I2C Library Calls (*bcm2835 vs. pigpio*):**

- Both libraries attempted to control the I2C peripheral, leading to conflicts when initializing or accessing the I2C bus
- Unable to initialize the RGB sensor due to it using Pigpio's I2C calls when Motors used bcm2835's I2C calls.
- To resolve this issue, we decided to not use the RGB sensor to check the red-line as we didn't have enough time to convert the motor functions into a pigpio initialization, or the RGB sensor library into a bcm2835 initialization.

**High CPU Usage**

- With five sensors operating simultaneously or frequently, the CPU becomes overloaded because The Raspberry Pi was not optimized for high-frequency real-time processing.
- To resolve this we changed the design to use 3 echo sensors.

**Conclusion:**

The robot car project has significantly enriched our understanding of embedded Linux systems and the integration of hardware and software in robotics. Through this hands-on experience, we enhanced our skills in problem-solving, team collaboration, and the practical application of theoretical concepts. By addressing real-world challenges such as sensor calibration, power management, and algorithm optimization, we gained valuable insights into designing and developing autonomous systems.

This project not only deepened our knowledge of embedded systems and robotics but also prepared us for future endeavors in innovation and technology.