Kuwait University

College of Computer Science & Engineering

Computer Engineering Department

CpE 434

Robotics

Phas1 Report

Lane-Following Robot with Autonomous Obstacle Avoidance

Prepared by Team KAYO

Zahraa Mohammad Alrashidi 2191118389
Hala Almutairi 2211112873

15/4/2025

Table of Contents

List of Figures

0. Purpose of the report

This report illustrates, in detail, the process of implementing our 434-Robotics course project named "Lane-Following Robot with Autonomous Obstacle Avoidance," which is divided into two phases. Phase 1 focuses on the assembly and remote operation of the robot, while Phase 2 involves implementing autonomous navigation and obstacle avoidance. This report covers Phase 1, starting with a summary of the project's goals. It then describes the project's background, including its technical, environmental, and economic impacts. After that, it explains the solution for the project, detailing the components, tools, and skills required for its implementation. Following that, it provides a step-by-step explanation of the robot's operation, supported by diagrams. Finally, the report concludes with the challenges encountered, lessons learned, and suggestions for improvement, along with appendices containing the glossary and references.

1. Project scope

Problem:

Schools in Kuwait face a dilemma with their transport systems, as many parents are concerned about the safety of their children when relying on school bus drivers, especially in terms of driving skills, punctuality, and the trustworthiness of strangers. The school aims to reduce traffic congestion during student pick-up and drop-off, but many parents, especially at girls' schools in Kuwait, are hesitant to trust the traditional bus system. The aim of this project is to develop a prototype for an autonomous, obstacle-avoiding bus system that addresses these concerns. The solution promises to reduce traffic congestion around school zones, decrease reliance on traditional bus systems, and improve overall time efficiency for student transportation.

Goals:

1. Assemble a rolling robot chassis with integrated motors, wheels, and sensors.
2. Establish functional remote-control operations using wireless interfaces.
3. Ensure sensors are properly working to detect obstacles effectively during remote operation.
4. Provide a foundation for autonomous navigation implementation in Phase 2.

2. Background

Prototype of Autonomous Electric Vehicle (AEV):

This prototype's design (Aisha Abdul Mohammed, 2021) incorporates an obstacle avoidance system. This research utilizes ultrasonic sensors to detect obstacles in front, left, and right of the vehicle, which are connected to an ATmega32 microcontroller for processing. The vehicle uses GPS navigation to autonomously navigate to its destination. The range of obstacle detection is from 2 cm to 400 cm, with an accuracy difference between measured and calculated distances of about 5.4%. The system is powered by rechargeable lithium-ion batteries that can be charged via an external power source or a solar panel.

Advantages:

1- Autonomous Movement: The vehicle operates autonomously after the initial programming, with no further human intervention required.

2- Obstacle Avoidance: The ultrasonic sensors help the vehicle avoid obstacles with high accuracy.

3- Energy Efficiency: The use of solar power alongside rechargeable batteries makes it more energy efficient.

Disadvantages:

1- Limited Navigation Distance: The vehicle can only move about 20 meters back and forth, which restricts its practical use to short-distance applications or small environments.

2- Battery Charging: Although the vehicle uses solar panels as a secondary power source, the system still relies on an external power grid to charge the lithium-ion batteries. In environments with insufficient sunlight, the solar charging may not be reliable, making it dependent on the electrical grid.

3- Battery Life: The vehicle's range is still constrained by the battery capacity, and excessive reliance on battery power can limit its long-term operational capability, especially when the solar panel cannot keep up with the charging needs.

Conclusion:

The system demonstrates how autonomous vehicles can be developed with a focus on obstacle avoidance, using affordable and reliable components such as ultrasonic sensors, GPS, and microcontrollers. The vehicle is effective in avoiding obstacles and performing basic autonomous navigation tasks, making it suitable for environments that might be unsafe for humans. This system can help build a prototype for the goal project, but more complex and fast reading/acting sensors must be used for the final bus system that would operate in the human environment. The vehicle is effective in avoiding obstacles and performing basic autonomous navigation tasks which shows its potential for improving traffic flow, reducing congestion, and promoting sustainability.

3. Project Specifications

a. The type of Agent

- In Phase 1, the robot functions as a **simple agent**, recording sensor inputs like obstacle detection without maintaining memory or modeling its environment. It moves on simple commands input by the user. This ensures efficient and straightforward operation for remote control.
- In Phase 2, the robot evolves into an **Autonomous-based agent**, using an internal representation of its environment to navigate autonomously, adapt to lane changes, and avoid dynamic and static obstacles with greater complexity.

b. Robot Environment description

1- Operating Environment

The robot is designed to operate indoors, specifically within a classroom setting. The environment includes a flat maze representing a road with a white surface bordered by black lines. The maze simulates real-world road scenarios, incorporating both static and dynamic obstacles. Therefore, implementing obstacle detection and avoidance mechanisms is essential.

2-Objects of Interaction

- The black lines at the side of the road would simulate the pavement on each side of the road.
- Dynamic obstacles moving in front of the robot that would simulate moving cars and pedestrians crossing the road, especially in school zones where children and parents might cross the road suddenly.
- Static obstacles on the side of the lanes, that would simulate parked cars and road barriers that the robot would need to avoid.

3- Operating Temperature Range

The robot is expected to function in indoor classroom temperatures ranging from 21°C to 25°C, consistent with Kuwait University's indoor climate settings. Hardware such as the Raspberry Pi 4 and Arduino Uno can operate up to 85°C and 85°C respectively, although optimal performance is achieved at lower temperatures (GPU/CPU temperature limit for RPI3, 2017), (What is the operating tempreture range for Arduino Boards?, 2024). Ultrasonic Sensors (e.g., HC-SR04) Operating Temperature Range: -15°C to +70°C (Ultrasonic Distance Sensor (HC-SR04), n.d.) Infrared Sensors (e.g., Sharp GP2Y0A21YK0F or similar) Operating Temperature Range: -10°C to +60°C (GP2Y0A02YK0F data sheet, n.d.).

4- Operating Humidity Range

The robot will operate in low humidity environments, as Kuwait University classrooms are climate-controlled to protect sensitive electronic equipment.

5- Terrain Type

The robot is designed for flat indoor surfaces with minimal unevenness, resembling typical classroom floors.

6- Ambient Light Intensity

The robot will operate in standard indoor lighting conditions, averaging around 390-420 lux (measured using Light Meter app on iPhone), similar to laboratory and classroom environments.

7- Obstacle Dimensions

- **Height:** Any ground level obstacle (that simulates parked cars and road barriers) should be detected. Obstacles that reach up to 10 cm (that simulate bridges) can be detected by the robot. The height represents the position of the robot's ultrasonic sensor fixed on the robot, enabling detection of ground-level and low-hanging objects.

- **Width:** Obstacles may vary in width. The robot should be capable of navigating around any obstruction that fully or partially blocks the simulated road.

8- Environment Characteristics

- The environment is discrete because the robot will operate in a structured road system where it interacts with a limited set of objects.
- The environment is episodic to some degree, as the robot will encounter individual events, such as stopping for dynamic pedestrians or avoiding a stationary car. However, its actions might affect subsequent states, meaning it's somewhat sequential as well.
- The environment is mostly static, with road structures of the lanes and static obstacles being fixed. However, dynamic elements like simulating pedestrians and vehicles make the environment partially dynamic.
- The environment can be considered fully observable to the robot at phase 2 when applying appropriate sensors, such as ultrasonic sensors, autonomous algorithms, and infrared sensors to detect all obstacles in its vicinity.
- The robot operates in a mostly deterministic environment but faces unpredictability due to dynamic obstacles like simulated pedestrians.

c. Environmental impacts of the system

1- Human and Environmental Impact

Positive Impacts:

- Reduction in Traffic Accidents: Autonomous vehicles can reduce accidents caused by human error, thereby improving public health and safety.
- Improved Traffic Flow: Autonomous vehicles can optimize traffic flow, reducing congestion due to excess cars on the road.

- This robot would give a better understanding of autonomous vehicles and introduce problem solving thinking. Ultimately producing improvements in the future of autonomous driving technology.

Negative Impacts:

Electronic Waste: The batteries and electronics used in autonomous vehicles may contribute to e-waste once they reach the end of their useful life. This can have environmental impacts if not disposed of properly.

2- Environmental Influence on Robot

Extremely high temperatures in Kuwait's summer can reach up to 50 degrees Celsius, which could cause overheating of the robot's components like batteries and internal electronics. High humidity can also affect the vehicle's sensors and can cause short-circuits or reducing effectiveness. Poor road conditions in outside environments like potholes and uneven dirt roads can negatively affect the robot's smooth navigation ability.

d. Technical impacts of the system

This section introduces system technical impacts as follows:

Sensors Latency:

The ultrasonic sensors detect obstacles within 2 ms per reading. The Arduino processes sensor data and sends motor commands within 5–10 ms.

Communication Latency:

The Raspberry Pi 4 communicates with the Arduino Uno via UART serial at 9600 baud rate, introducing a delay of 10–20 ms for command transmission due to the lower data rate.

Remote Control Latency:

The website updates sensor data every 500 ms and relays user commands within 100 ms of input.

Transactions per Second:

The system processes 10–15 sensor updates per second (combined for ultrasonic and IR sensors), limited by the 9600 baud rate.

Job Completion Time:

During testing, the robot navigated a 5 m simulated straight lane in 30–45 seconds under remote control, consistent with the original report. It also navigated a simulated maze with obstacles in 60 seconds under remote control.

Mean Time to Repair & Rate of Occurrence of Failure:

Sensor/Motor Replacement: 5 to 10 minutes.

Microcontroller Replacement: 10 to 30 minutes.

Observed failure during testing due to IR misalignment.

e. Economical impacts of the system

- **Positive Economic Impacts**

  Power Efficiency:

  - Low Power System Consumption:

    Arduino run on a 3.7V, 3200mAh lithium battery

    Most commonly Arduino boards consume 0.5~2W.

    Raspberry Pi uses 22.5 W power bank

  Total system consumption ≈ 0.5~2W + 22.5W = 23~24.5W

  Low Maintenance Costs:

  - Affordable Repairs:

    Battery replacement: 2–5 kd

    Power bank replacement: 10–20 kd

    Sensor/component replacement: 0.1–20 kd
  - Easy to repair or upgrade without replacing the entire system.

  Low Manufacturing Costs:

  - Total Prototype cost: 83.75 KD
  - Affordable for target users (schools, transport companies).

- **Negative Economic Impacts**

  Power Limitations:

  - The lithium battery loses capacity over time requiring replacement.

  Maintenance Challenges:

  - If critical parts fail the system will be unusable until repaired or replaced.

  Specialized Design:

  - Built for school bus obstacle detection following rode not easily adaptable for other uses without costly redesigns.

4. Solution Formulation

a. Introduction

In this section we are explaining the system solution in addition to its advantages and disadvantages

b. Proposed Solution

I. components

Car built using the following components:

1. Power Bank:

   Provides power to Raspberry Pi 4.

2. Raspberry Pi 4:

   Main processing unit controlling logic and communication.

3. Arduino Uno:

   Secondary controller. Handles sensor data and motor controller.

4. Arduino Expansion Board:

   Extends the Arduino Uno pins.

5. L298N Motor Driver Module:

   Connect between the Arduino and DC motors enabling motors control.

6. DC Motors with Tires (x4):

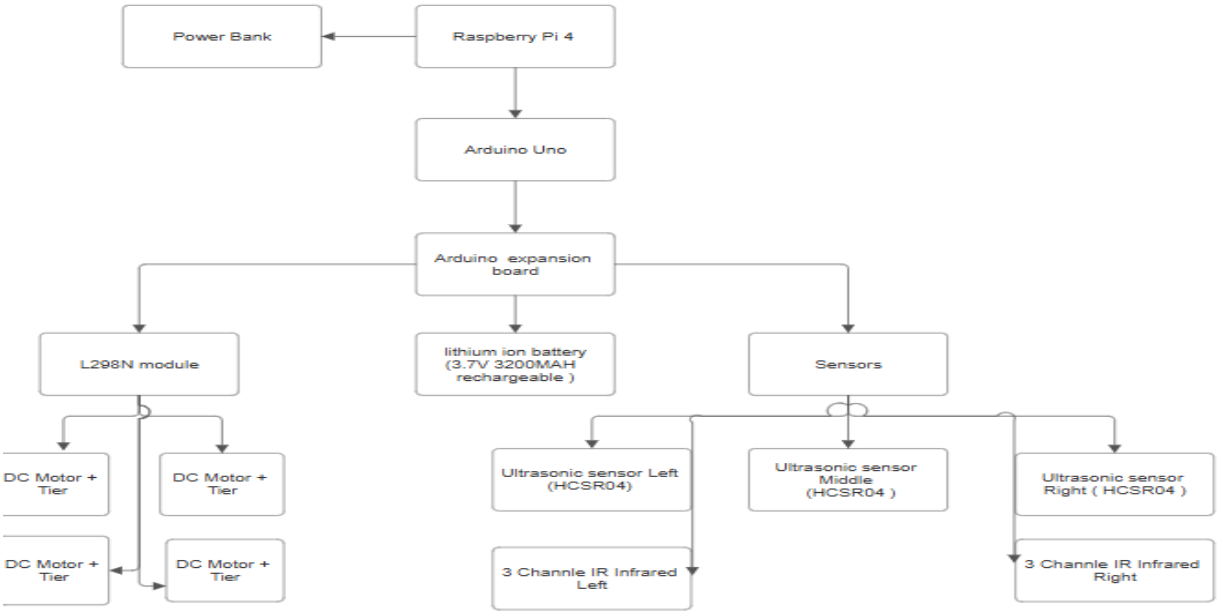
   For Movement of car.

7. 3x Ultrasonic Sensors (HCSR04):

   For obstacle detection (Left, Right and Middle).

8. 2x 3-Channel IR Infrared Sensors (2):

   For Detecting black/white lines (Left and Right).



: Components Connection1 Figure

Car components are interconnected as follows:

The Lithium-ion Battery powers the Arduino Uno system consists of sensors and L298N Module. The Arduino reads data from three Ultrasonic Sensors (left, right, middle) for obstacle detection and two 3-Channel IR Sensors (left, right) for line detection. then sends motor control signals to the L298N Module. The L298N drives four DC Motors using power from the Li-ion battery. The Arduino Expansion Board extends I/O pins to manage wiring easily. The Power Bank power Raspberry Pi 4. Communication between the Raspberry Pi 4 and Arduino is done by using UART Serial (USB).

II. Algorithm

The system consists of a Raspberry Pi 4 and Arduino working together, as illustrated in the flowchart. The Raspberry Pi 4 acts as the central controller, running a website that allows users to read sensors and control car by sending commands to Arduino. The Arduino controls hardware such as motor and sensor and responding according to instructions from the Raspberry Pi.

How It Works:

The process begins with both devices initializing the Raspberry Pi starts its web server while the Arduino prepares the motors and sensors. The system then checks the serial connection between them, automatically retrying if needed. Once connected users can send commands through the website which the Raspberry Pi forwards to the Arduino. The Arduino first verifies it’s in the correct operating mode before executing any movements. Meanwhile sensor data continuously flows back to the Raspberry Pi updating the website sensor data.

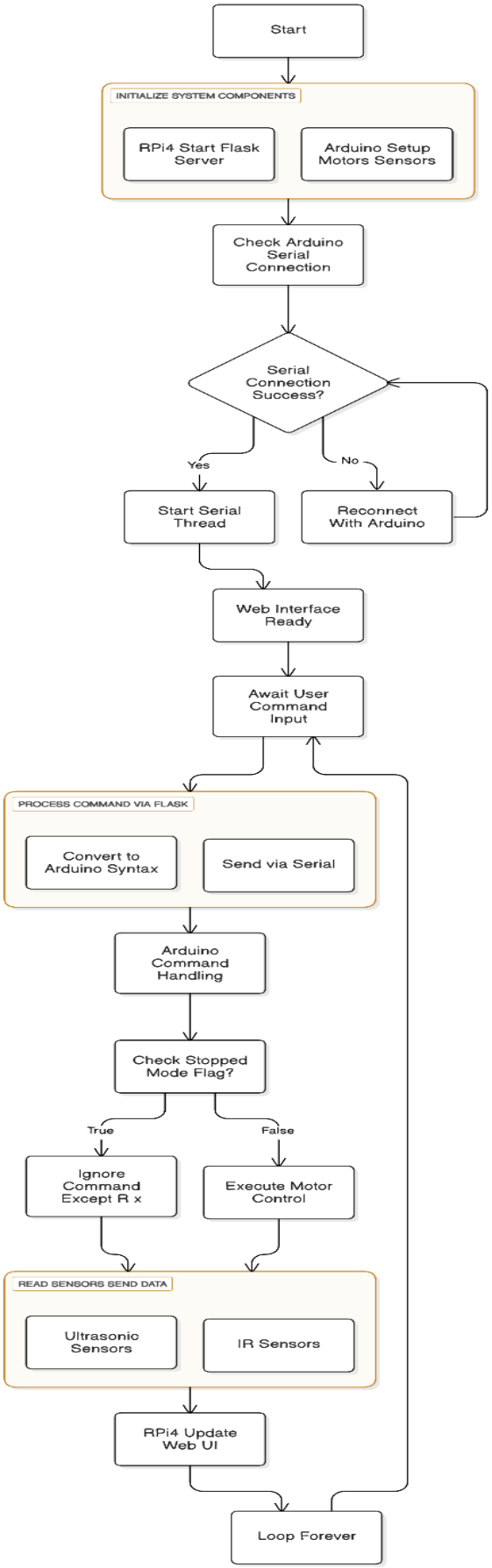


Figure 2: Flow Chart for System

III. Required Implementation tools

- Hardware Equipment:

  Laptop/PC: For programming and connecting to Arduino.

  Serial Cable (USB): For downloading code to the Arduino and connecting to Raspberry Pi4.

  MicroSD: For storing the Raspberry Pi files NOOBS OS.

  Workshop Tools: Screwdrivers ,drill , Multimeter

  USB Type C Power Supply (Model B 5v 3a): for powering Raspberry Pi 4 .

  Monitor Screen&Micro-HDMI: for connecting monitor to Raspberry Pi4.

  Keyboard: For typing on Raspberry Pi 4.

  Mouse: For navigating on Raspberry Pi 4.

  18650 lithium battery charger: for charging lithium battery.

- Software Applications:

  Arduino IDE: For programming the Arduino.

  NOOBS: For setting up the Raspberry Pi4.

  Python: For programming the Raspberry Pi4.

  Thonny IDE: For Programming Raspberry Pi4.

  PiTunnel: Remote access to Raspberry Pi4.

  GitHub: For version control and collaboration.

IV. Advantages

The system is easy to set up because it uses common parts and common hardware programming. Also, the kit saved us time by providing most of the parts we needed and making assembly quicker. Plus, it was easy to add new components and replace broken components.

V. Disadvantages

Finding the right batteries for our system took weeks due to their limited availability and Learning Python while building the system was hard as we struggled with syntax and debugging. HTML website also slowed us down even though we are familiar with html and JavaScript before we had to relearn a lot of things. We also encountered difficulties with the Raspberry Pi 4 as we needed to have a correct MicroSD card with NOOBS installed and all required packages on it. Additionally, we had to place the Raspberry Pi on an upper layer due to lack of space on the middle layer which required us to acquire new skills in hardware drilling and using screws. Moreover, there was no dedicated workspace for the team to

work on making it hard to work as we always have transport equipment.

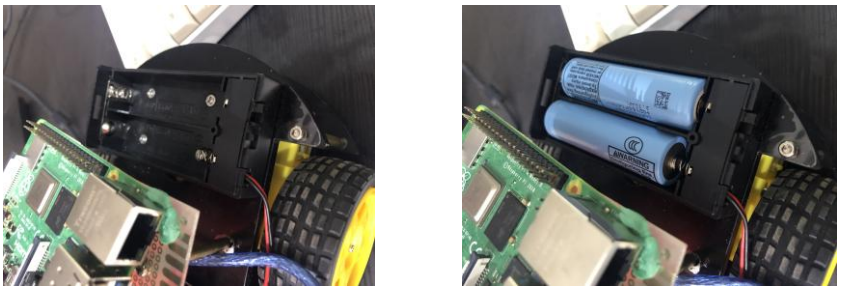
5. Final System User Manual

This section explains how the system works.

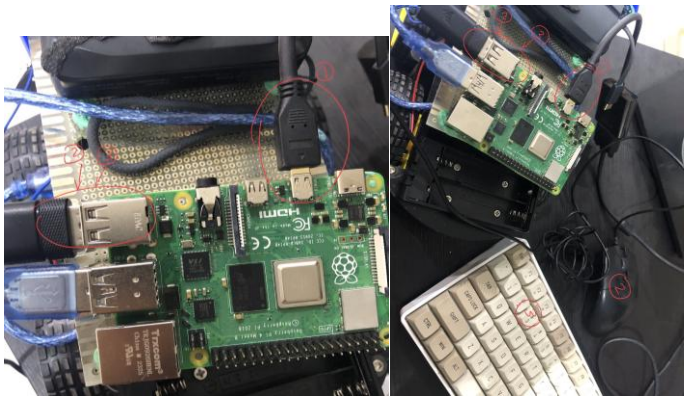
a. Robot Setup Guide:

1- Make sure the power bank and the motor batteries are fully charged.

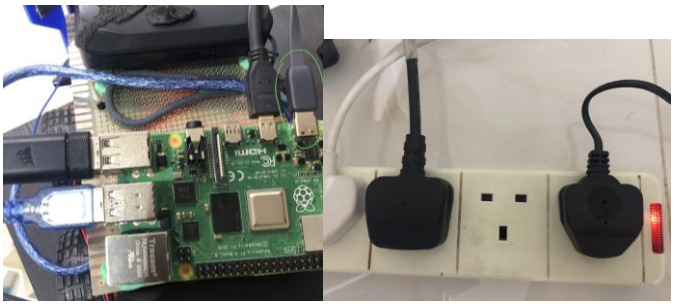
2- Set the batteries in the battery case of the robot.



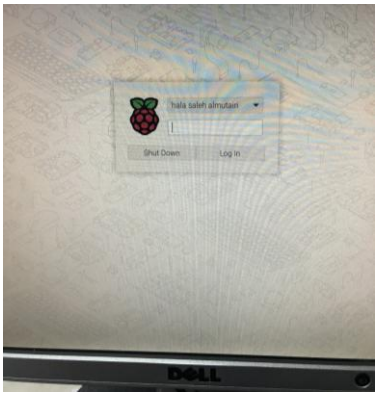
3- Connect screen monitor (using D-Sub to micro-HDMI cable), mouse and keyboard (using the two USB ports) to raspberry pi.



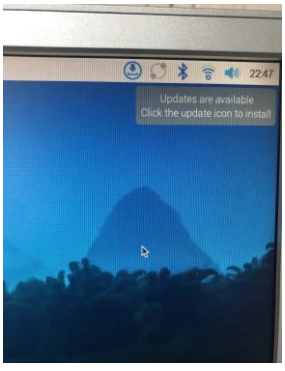
4- Connect the power bank to raspberry pi on the C-port and turn the screen power on.

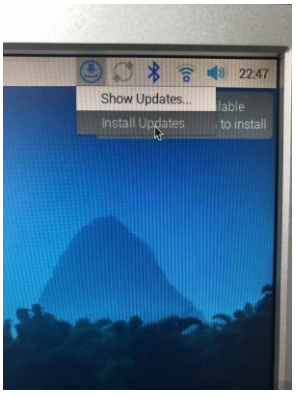


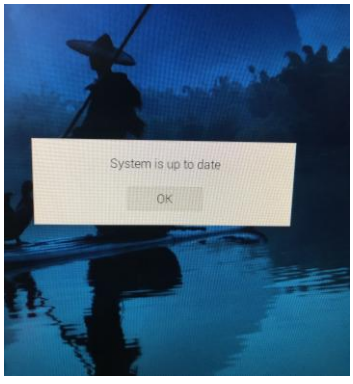
5- Login to account on raspberry pi.



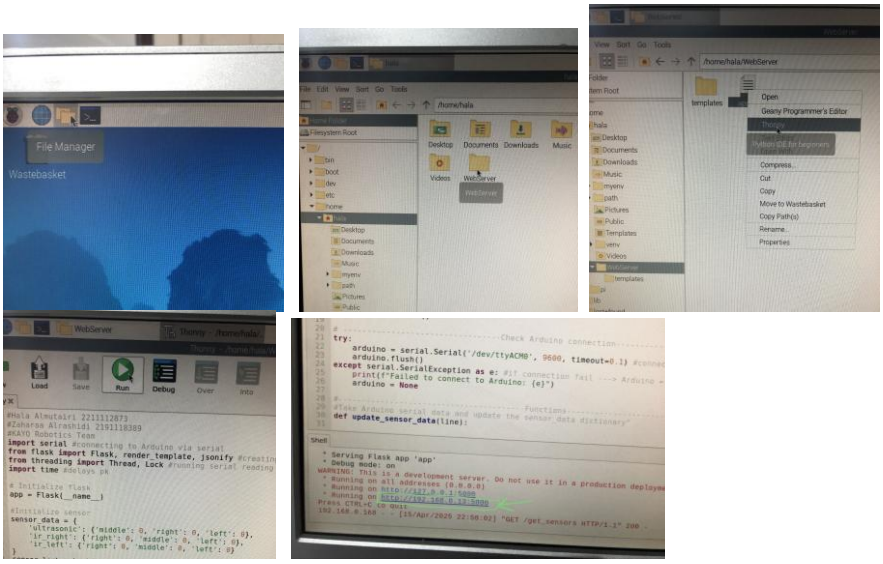
6- Make sure any updates are installed.





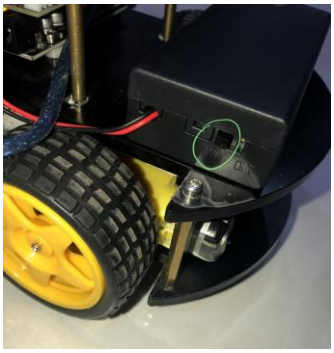


7- Run the raspberry pi code “app.py” stored in WebServer folder, run the code using Thonny. Memorize the local IP address you need to connect to that appears on the shell channel.



8- Disconnect raspberry pi from screen, mouse and keyboard. Careful to NOT disconnect the raspberry-to-Arduino cable.

9- Turn on motors battery.



10- Connect your phone to a 5G network.

11- Open Safari or chrome browser and type the previous IP address with the port number 5000. Example:

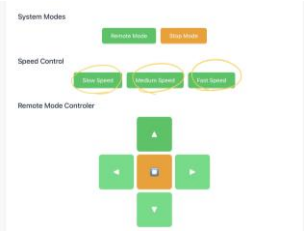
Example: 192.168.0.13:5000

b. Robot Controls Guide:

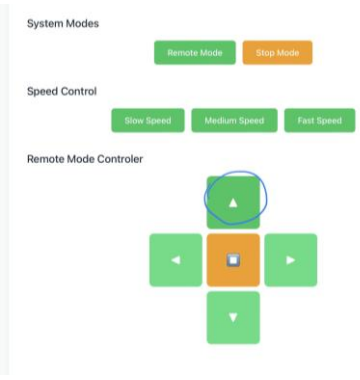
1- Place the robot on the ground.

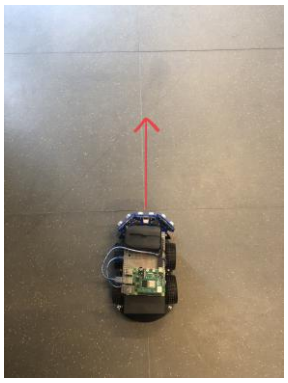
2- Press “Remote Mode” to move the robot with user commands.

3- Press the intended button for the desired speed; slow, medium, or fast. Automatically the speed is set to medium.

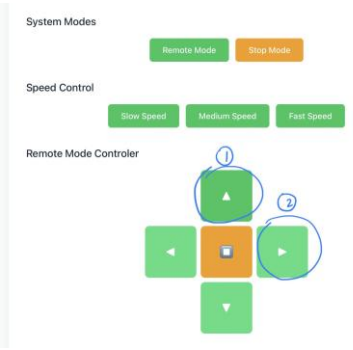


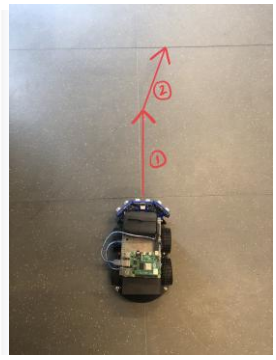
4- Press the upward facing arrow to move the robot forward



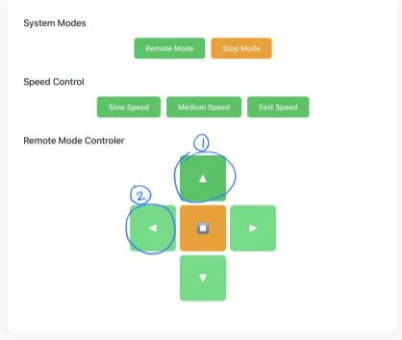


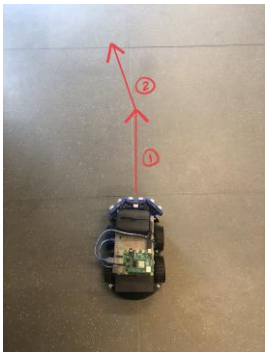
5- While moving forward, press the right arrow button to move the robot 20 degrees in the right direction then keep moving forward.



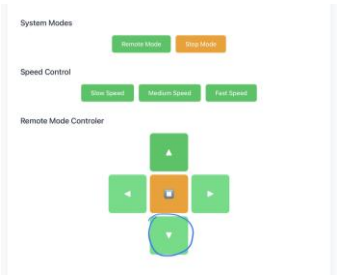


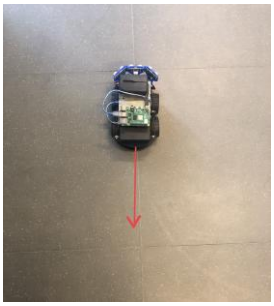
6- While moving forward, press the left arrow button to move the robot 20 degrees in the left direction then keep moving forward.



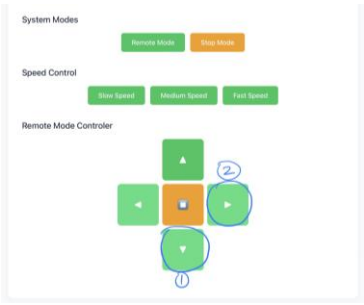


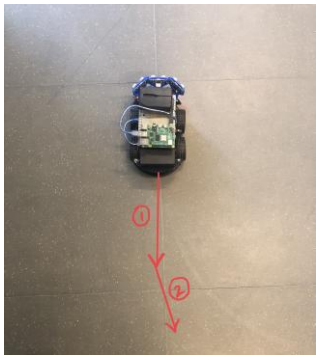
7- Press the down arrow button to move the robot backward.



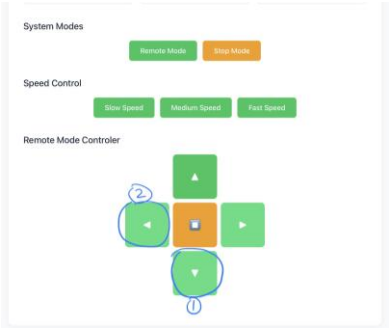


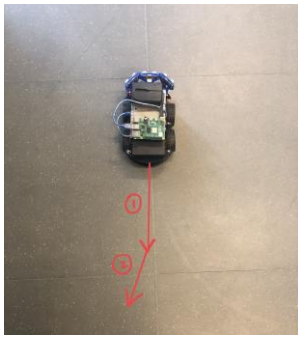
8- Press the right arrow button while moving backward to move the robot 20 degrees in the southeast direction then keep moving backward.



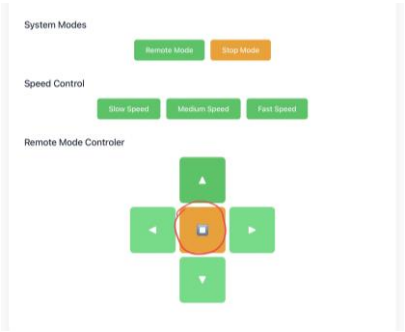


9- Press the left arrow button while moving backward to move the robot 20 degrees in the southwest direction then keep moving backward.

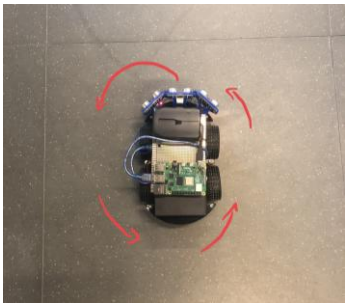




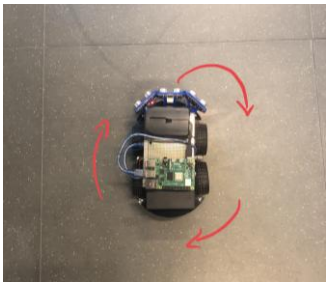
10- Press the middle button to stop the robot's movement.



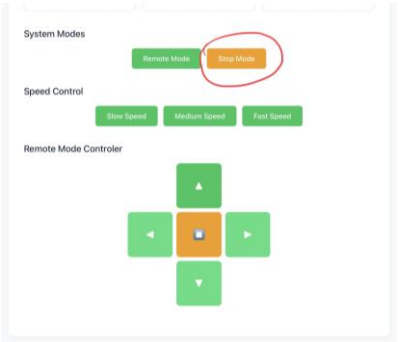
11- While the robot is stopped press the left arrow button to keep moving the robot 360 degrees to the left. Until the desired angle is achieved then continue with selecting the next command button (stop, forward, backward).



12- While the robot is stopped press the right arrow button to keep moving the robot 360 degrees to the right. Until the desired angle is achieved then continue with selecting the next command button (stop, forward, backward)

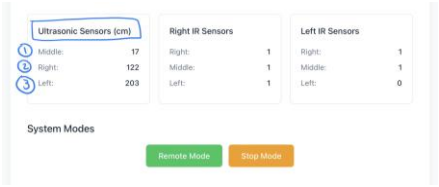


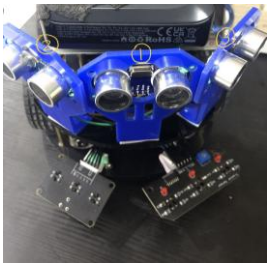
13- Press stop mode button for emergency exiting all modes (remote mode and future Auto drive mode in phase 2)



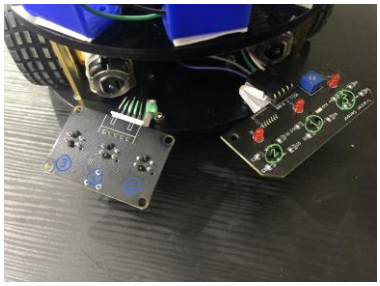
14- Sensor data will be displayed in their respective slots. The readings would be displayed with a delay of 300 milliseconds for better reading ability.

- Ultrasonic sensor readings are shown in cm distance.



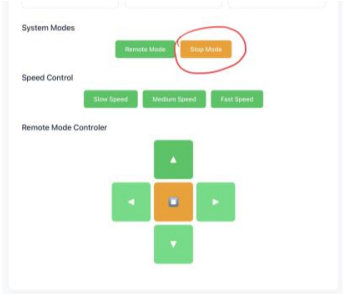


- Right and left infrared sensors readings will also be displayed, with 1 meaning a black road line border is being detected, and 0 reading meaning white road is being detected.



c. Stop the System Safely:

1- Press the stop mode button on the website.



2- Turn off the battery supply from the motors.

3- Disconnect the power bank from the raspberry pi.

Preconditions That Must be Achieved:

1- Robot Setup must be executed first, in order of the steps, before robot control.

2- Remote Mode must be selected before trying to manually command the robot to move.

Deficiencies Discovered During Testing:

1- Calibrating the servo motors is tricky and still not up to the team's standards.

6. Conclusions and suggestions

a. Problems encountered and solutions

- Battery:

  Finding the right batteries took weeks due to limited availability. We solved this by researching alternative battery to buy in Kuwait instead of original battery we are looking for.

- Python & HTML:

  Connecting Python with HTML website caused errors like data not displaying correctly and accessing Raspberry Pi4 website after discounting it. We solved this after lap7 on Robotics Lap as we learned about Flask for Python and how to connect Raspberry Pi4 website step by step.

- MicroSD Card Setup:

  Setting up Raspberry Pi 4 required a correct MicroSD card with NOOBS and all necessary packages. We solved this by following setup guides and ensuring all required software was installed.

- Limited Space on Car:

  The Raspberry Pi4 had to be placed on an additional upper layer due to lack of space on the middle layer. We solved this by asking help from Engineer Tareek AL-Melhem and he provided us with a new layer and helped us setting it up layer.

- Limited Workspace:

  There was no dedicated workspace to work on as team made it hard to work consistently requiring regular transportation of equipment. We haven't solved this problem yet. We work by scheduling equipment between team members to work on home the other time if our schedule align we work on campus and on Engineer Tareek AL-Melhem Lab.

b. Skills and concepts learned from the project

Both team members learned new skills and knowledge on the following:

environment Setup:

- Learned how to Setup Raspberry Pi.
- Learned how to Setup Arduino.

Resource Searching:

- Improved ability to search for resources and solutions via the Internet, including forums, tutorials, and documentation.

Teamwork:

- This project helped us appreciate the value of good team members. Our communication skills improved significantly throughout Phase 1. We support each other in understanding concepts and learning new skills enhancing our overall teamwork.

Hardware:

- Assembling System components.
- Learn how components interact with each other and connect.
- Where to buy and price of each component.
- Debugging components to see if they work properly.

Software:

- Learned Python for Raspberry Pi.
- Arduino programing language C++.
- Implementing a website using Flask, HTML, and JavaScript
- How to connect Raspberry Pi4 to Arduino
- How to connect PiTunnel to Raspberry Pi4
- Leaned Terminal command to install packages
- Learned how to Program sensors (ultrasonic, infrared line module sensor)

c. Suggestions

To improve the final system of our team, we suggest the following:

- New battery:
  Search for a new battery alternative to fit our system not the other way around.

- Optimizing space.:
  Instead of buying kit we should build Chassis layer and design it to fit our system and component.

- Improving Website UI:
  Enhance the user interface of the website to make it more user-friendly.

- Enhancing Performance:
  Optimize the software and hardware setup to improve the overall system performance.

- Reducing Cost:
  Explore inexpensive alternatives for components and tools to reduce the overall system cost.

7. Copyright and intellectual properties

Appendix A:

Aisha Abdul Mohammed, A. A. (2021, November). Development of aPrototype Autonomous Electric Vehicle. *Journal of Robotics and Control (JRC), 2*(6), [559, 564]. doi:10.18196/jrc.26137

GP2Y0A02YK0F data sheet. (n.d.). Retrieved from Sharp Global: https://global.sharp/products/device/lineup/data/pdf/datasheet/gp2y0a02yk_e.pdf

GPU/CPU temperature limit for RPI3. (2017, April 29). Retrieved from Rasberry Pi Forums: https://forums.raspberrypi.com/viewtopic.php?t=181882#:~:text=Re%3A%20GPU%2FCPU%20temperature%20limit%20for%20RPI3&text=Limit%20is%2085%C2%B0C,back%20until%20the%20temperature%20drops.

Ultrasonic Distance Sensor (HC-SR04). (n.d.). Retrieved from PiBorg.: https://www.piborg.org/sensors-1136/hc-sr04

What is the operating tempreture range for Arduino Boards? (2024, January 29). Retrieved from Arduino Help Center: https://support.arduino.cc/hc/en-us/articles/360016076980-What-is-the-operating-temperature-range-for-Arduino-boards#:~:text=Operating%20Temperature%20Range%3A%20%2D40%C2%B0,if%20this%20range%20is%20surpassed.

Appendix B:

1- Arduino Code:

```
// Hala Almutairi 2211112873
// Zaharaa Alrashidi 2191118389
// KAYO Robotics Team

#include <Servo.h> //Servo library to control motor
Servo myservo; // servo object



//Right Line Tracking
#define RLT_R !digitalRead(10)
#define RLT_M !digitalRead(4)
#define RLT_L !digitalRead(2)

//Left Line Tracking
#define LLT_R digitalRead(12)
#define LLT_M digitalRead(13)
#define LLT_L digitalRead(3) //took a LED pin to not mess with the motors

// Define L298N motor pins
#define ENA 5  // Enable Left Motor speed controller
#define ENB 6  // Enable Right Motor speed controller
#define IN1 7  // Left Motor pin1
#define IN2 8  // Left Motor pin2
#define IN3 9  // Right Motor pin3
#define IN4 11 // Right Motor pin4
int carSpeed = 150; // car speed

bool stoppedMode = false; // Flag for Stop mode
char command;  //to store command received from website
String state = "stop"; //keep track of prev command ---> initial stop

// Define Middle ultrasonic pins
int Echo = A4;
int Trig = A5;
// Define Right ultrasonic pins
int EchoR = A0;
int TrigR = A1;
// Define Left ultrasonic pins
int EchoL = A3;
int TrigL = A2;

// initialize ultrasonic
int rightDistance = 0;
int leftDistance = 0;
```

```
int middleDistance = 0;

// Function to STOP car
void stop(){
  digitalWrite(ENA, LOW);
  digitalWrite(ENB, LOW);
  Serial.println("Stop");
}

// Function to move car forward with calibrated speeds
void forward(){
  int leftSpeed = carSpeed-10; // servo wheels calibration
  analogWrite(ENA, leftSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN4, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN1, HIGH);




  Serial.println("Forward");
}

// Function to move car back with calibrated speeds
void back(){
  int rightSpeed = carSpeed-8; // servo wheels calibration
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, rightSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN4, LOW);
  Serial.println("Back");
}

// Function to move car left with calibrated speeds
void left(){
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN4, HIGH);
  Serial.println("Left");
}

// Function to move car right with calibrated speeds
void right(){
```

```
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN2, LOW);
  digitalWrite(IN4, LOW);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN3, HIGH);
  Serial.println("Right");
}

// Function to calc distance via ultrasonic sensor
int Distance_test(int Trig, int Echo) {
  digitalWrite(Trig, LOW);    //Initialize Trig
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);  //Send Trig pulses
  delayMicroseconds(20);
  digitalWrite(Trig, LOW);   //Stop Trig pulses
  float Fdistance = pulseIn(Echo, HIGH);  // Calc pulse distance
  Fdistance = Fdistance / 58;       // Convert distance to cm
  return (int)Fdistance; //Return Distance
}
int Distance_test2(int Trig, int Echo) {
  digitalWrite(Trig, LOW);    //Initialize Trig
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);  //Send Trig pulses
  delayMicroseconds(20);
  digitalWrite(Trig, LOW);   //Stop Trig pulses
  float Fdistance = pulseIn(Echo, HIGH);  // Calc pulse distance
  Fdistance = Fdistance / 58;       // Convert distance to cm
  return (int)Fdistance; //Return Distance
}
void setup() {
  myservo.attach(3); // Attach servo obj to pin3
  Serial.begin(9600);
  pinMode(Echo, INPUT);
  pinMode(Trig, OUTPUT);
  pinMode(EchoR, INPUT);
  pinMode(TrigR, OUTPUT);
  pinMode(EchoL, INPUT);
  pinMode(TrigL, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(RLT_R, INPUT);
  pinMode(RLT_M, INPUT);
  pinMode(RLT_L, INPUT);
  pinMode(LLT_R, INPUT);
  pinMode(LLT_M, INPUT);
```

```
  pinMode(LLT_L, INPUT);
  state = "stop";
  stop(); // Initialize car at stop mode
}

void loop() {
  if (Serial.available() > 0) { // Check if there is command from Serial
communication
    command = Serial.read();

    if (command == 'R') {
      stoppedMode = false;
    } else if (command == 'x') {
      stoppedMode = true;
      stop();
    } else if (command == 'L') { // Slow
      carSpeed = 90;
    } else if (command == 'M') { // Medium
      carSpeed = 150;
    } else if (command == 'H') { // Fast
      carSpeed = 220;
    } else if (!stoppedMode) { // if is not any of the Mode flag then its Remote Mode
      if (command == 'f') {
        state = "forward";
        forward();
      } else if (command == 'b') {
        state = "back";
        back();
      } else if (command == 'l') {
        if(state == "forward") {
          left();
          delay(100);
          forward();
        } else if (state == "back") {
          left();
          delay(100);
          back();
        } else if (state == "stop") {
          left();
        }
      } else if (command == 'r') {
        if(state == "forward") {
          right();
          delay(100);
          forward();
        } else if (state == "back") {
          right();
          delay(100);
          back();
        } else if (state == "stop") {
```

```
          right();
        }
      } else if (command == 's') {
        state = "stop";
        stop();
      }
    }
  }

  //send ultrasonic sensors data to Raspberry Pi4 via serial
  Serial.print("US_M:");
  Serial.println(Distance_test(Trig, Echo));
  Serial.print("US_R:");
  Serial.println(Distance_test2(TrigR, EchoR));
  Serial.print("US_L:");
  Serial.println(Distance_test(TrigL, EchoL));

  //send right IR sensors data to Raspberry Pi4 via serial
  Serial.print("IR_R_R:"); Serial.println(RLT_R);
  Serial.print("IR_R_M:"); Serial.println(RLT_M);
  Serial.print("IR_R_L:"); Serial.println(RLT_L);

  //send left IR sensors data to Raspberry Pi4 via serial
  Serial.print("IR_L_R:"); Serial.println(LLT_R);
  Serial.print("IR_L_M:"); Serial.println(LLT_M);
  Serial.print("IR_L_L:"); Serial.println(LLT_L);
 }
```

2- Raspberry Pi “app.py” Code:

```
#Hala Almutairi 2211112873
#Zaharaa Alrashidi 2191118389
#KAYO Robotics Team
import serial #connecting to Arduino via serial
from flask import Flask, render_template, jsonify #creating web application ---> FLAsK
from threading import Thread, Lock #running serial reading in a separate thread --> threading
import time #delays pk

# Initialize flask
app = Flask(__name__)

#Initialize sensor
sensor_data = {
    'ultrasonic': {'middle': 0, 'right': 0, 'left': 0},
    'ir_right': {'right': 0, 'middle': 0, 'left': 0},
```

```
    'ir_left': {'right': 0, 'middle': 0, 'left': 0}
}
sensor_lock = Lock()

# ---------------------------------Check Arduino connection------------------------------
try:
    arduino = serial.Serial('/dev/ttyACM0', 9600, timeout=0.1) #connecting to
Arduino via serial
    arduino.flush()
except serial.SerialException as e: #if connection fail ---> Arduino = none
    print(f"Failed to connect to Arduino: {e}")
    arduino = None

#---------------------------------------Functions-----------------------------------------
#Take Arduino serial data and update the sensor_data dictionary"
def update_sensor_data(line):

    try:
        if not line or ':' not in line:
            return False

        key, value = line.split(':', 1) #split line into (key-value)
        value = int(value) #convert String to int
        with sensor_lock:  #update sensors value based on key
            if key == 'US_M':
                sensor_data['ultrasonic']['middle'] = value
            elif key == 'US_R':
                sensor_data['ultrasonic']['right'] = value
            elif key == 'US_L':
                sensor_data['ultrasonic']['left'] = value
            elif key == 'IR_R_R':
                sensor_data['ir_right']['right'] = value
            elif key == 'IR_R_M':
                sensor_data['ir_right']['middle'] = value
            elif key == 'IR_R_L':
                sensor_data['ir_right']['left'] = value
            elif key == 'IR_L_R':
                sensor_data['ir_left']['right'] = value
            elif key == 'IR_L_M':
                sensor_data['ir_left']['middle'] = value
            elif key == 'IR_L_L':
                sensor_data['ir_left']['left'] = value
            else:
                return False #if it isnt sensor data ---> false
        return True
    except ValueError: #if converting String to int failed ---> handel error
        return False

#continuously reads Arduino serial data
def read_serial_data():
```

```
    while True:
        if arduino and arduino.in_waiting > 0:
            try:
                line = arduino.readline().decode('utf-8', errors='ignore').strip() #read from
serial
                if line: #check if there is reading
                    update_sensor_data(line)  #yes -->  update sensors with new data
            except Exception as e: #no ---> handel error
                print(f"Serial read error: {e}")
        time.sleep(0.01)

#---------------------------------Flask Website Endpoints----------------------------------
------
#endpoint to get current (now) sensor data
@app.route('/get_sensors')
def get_sensors():
    with sensor_lock:
        return jsonify(sensor_data)

#endpoint to send commands to Arduino
@app.route('/send_command/<cmd>')
def send_command(cmd):
    if arduino is None:
        return jsonify({'status': 'error', 'message': 'Arduino not connected'}), 500
    command_map = {
        'forward': 'f',
        'back': 'b',
        'left': 'l',
        'right': 'r',
        'stop': 's',
        'remote mode': 'R',
        'stop mode': 'x',
        'slow':      'L',
        'medium':  'M',
        'fast':      'H'
    }
    if cmd in command_map:
        arduino.write(command_map[cmd].encode())  #send command to Arduino
    return jsonify({'status': 'success'})

#Main app page endpoint --> Render HTML file
@app.route('/')
def index():
    return render_template('index.html')
#---------------------------------start serial reading thread and run Flask----------------
-------
if __name__ == '__main__':
    if arduino is not None:
        serial_thread = Thread(target=read_serial_data)
        serial_thread.daemon = True
```

```
        serial_thread.start()

    # Run Website
    app.run(debug=True, use_reloader=False, port=5000,
host='0.0.0.0',threaded=True)
```

3- Raspberry Pi Website “index.html” Code:

```
<!--
Hala Almutairi 2211112873
Zaharaa Alrashidi 2191118389
KAYO Robotics Team
CSS (Style) Generated By Chatgpt
-->
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <meta name="viewport" content="width=device-width, initial-scale=1.0">
   <title>KAYO</title>
   <style>
      :root {
         --bg: #f8fafc;
         --card: #ffffff;
         --text: #1e293b;
         --text-light: #64748b;


         --green-300: #86efac;
         --green-400: #4ade80;
         --green-500: #22c55e;
         --green-600: #16a34a;
         --green-700: #15803d;


         --amber-300: #fcd34d;
         --amber-400: #fbbf24;
         --amber-500: #f59e0b;
         --amber-600: #d97706;
         --amber-700: #b45309;


         --gray-100: #f1f5f9;
         --gray-200: #e2e8f0;
```

```
    --gray-300: #cbd5e1;

    --border: var(--gray-200);
    --highlight: var(--gray-100);
    --shadow: 0 2px 12px rgba(0,0,0,0.05);
}

* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: 'Inter', -apple-system, BlinkMacSystemFont, sans-serif;
    line-height: 1.6;
    color: var(--text);
    background-color: var(--bg);
    padding: 1rem;
    min-height: 100vh;
    display: flex;
    flex-direction: column;
    align-items: center;
}

.control-panel {
    background: var(--card);
    border-radius: 12px;
    padding: 2rem;
    width: 100%;
    max-width: 900px;
    box-shadow: var(--shadow);
}

.header {
    margin-bottom: 2rem;
    text-align: center;
}

.header h1 {
    font-weight: 600;
    font-size: 1.8rem;
    margin-bottom: 0.5rem;
    color: var(--text);
}

.sensor-grid {
    display: grid;
    grid-template-columns: repeat(auto-fit, minmax(200px, 1fr));
    gap: 1rem;
```

```
    margin-bottom: 2rem;
}

.sensor-card {
    background: var(--card);
    border-radius: 8px;
    padding: 1.25rem;
    border: 1px solid var(--border);
    transition: transform 0.2s, box-shadow 0.2s;
}

.sensor-card:hover {
    transform: translateY(-2px);
    box-shadow: var(--shadow);
}

.sensor-card h3 {
    font-size: 1rem;
    font-weight: 500;
    margin-bottom: 1rem;
    color: var(--text);
}

.sensor-row {
    display: flex;
    justify-content: space-between;
    margin-bottom: 0.5rem;
    font-size: 0.9rem;
}

.sensor-label {
    color: var(--text-light);
}

.sensor-value {
    font-weight: 500;
}

.control-section {
    margin-bottom: 2rem;
}

.control-section h2 {
    font-size: 1.2rem;
    font-weight: 500;
    margin-bottom: 1rem;
}

.directional-pad {
    display: grid;
```

```
    grid-template-columns: repeat(3, 1fr);
    gap: 0.5rem;
    max-width: 300px;
    margin: 0 auto 1.5rem;
}

.dpad-button {
    aspect-ratio: 1;
    display: flex;
    align-items: center;
    justify-content: center;
    font-size: 1.5rem;
    padding: 0;
    border: none;
    border-radius: 8px;
    cursor: pointer;
    transition: all 0.2s;
}

.dpad-forward {
    background-color: var(--green-500);
    color: white;
}

.dpad-forward:hover {
    background-color: var(--green-600);
    transform: translateY(-2px);
}

.dpad-left, .dpad-right, .dpad-back {
    background-color: var(--green-400);
    color: white;
}

.dpad-left:hover, .dpad-right:hover, .dpad-back:hover {
    background-color: var(--green-500);
    transform: translateY(-2px);
}

.dpad-stop {
    background-color: var(--amber-500);
    color: white;
}

.dpad-stop:hover {
    background-color: var(--amber-600);
    transform: translateY(-2px);
}

.system-button {
```

```
            padding: 0.75rem 1.5rem;
            border-radius: 6px;
            border: none;
            font-weight: 500;
            font-size: 0.95rem;
            cursor: pointer;
            transition: all 0.2s;
        }

        .button-mode {
            background-color: var(--green-500);
            color: white;
        }

        .button-mode:hover {
            background-color: var(--green-600);
            transform: translateY(-1px);
        }

        .button-emergency {
            background-color: var(--amber-500);
            color: white;
        }

        .button-emergency:hover {
            background-color: var(--amber-600);
            transform: translateY(-1px);
        }

        .button-group {
            display: flex;
            gap: 0.75rem;
            justify-content: center;
        }

        @media (max-width: 600px) {
            .button-group {
                flex-direction: column;
                align-items: center;
            }
        }
    </style>
</head>
<body>
    <div class="control-panel">
        <div class="header">
            <h1>Sensor Data</h1>
        </div>

            <!-- Sensor data grid -->
```

```
<div class="sensor-grid">

  <!-- Ultrasonic sensor card -->
  <div class="sensor-card">
    <h3>Ultrasonic Sensors (cm)</h3>
    <div class="sensor-row">
      <span class="sensor-label">Middle:</span>
      <span class="sensor-value" id="us-middle">0</span>
    </div>
    <div class="sensor-row">
      <span class="sensor-label">Right:</span>
      <span class="sensor-value" id="us-right">0</span>
    </div>
    <div class="sensor-row">
      <span class="sensor-label">Left:</span>
      <span class="sensor-value" id="us-left">0</span>
    </div>
  </div>

  <div class="sensor-card">
    <h3>Right IR Sensors</h3>
    <div class="sensor-row">
      <span class="sensor-label">Right:</span>
      <span class="sensor-value" id="ir-r-right">0</span>
    </div>
    <div class="sensor-row">
      <span class="sensor-label">Middle:</span>
      <span class="sensor-value" id="ir-r-middle">0</span>
    </div>
    <div class="sensor-row">
      <span class="sensor-label">Left:</span>
      <span class="sensor-value" id="ir-r-left">0</span>
    </div>
  </div>

  <div class="sensor-card">
    <h3>Left IR Sensors</h3>
    <div class="sensor-row">
      <span class="sensor-label">Right:</span>
      <span class="sensor-value" id="ir-l-right">0</span>
    </div>
    <div class="sensor-row">
      <span class="sensor-label">Middle:</span>
      <span class="sensor-value" id="ir-l-middle">0</span>
    </div>
    <div class="sensor-row">
      <span class="sensor-label">Left:</span>
      <span class="sensor-value" id="ir-l-left">0</span>
    </div>
  </div>
```

```
    </div>

    <div class="control-section">
      <h2>System Modes</h2>
      <div class="button-group">
        <button onclick="sendCommand('remote mode')" class="system-button
button-mode">
          Remote Mode
        </button>
        <button onclick="sendCommand('stop mode')" class="system-button
button-emergency">
          Stop Mode
        </button>
      </div>
    </div>

      <div class="control-section">
      <h2>Speed Control</h2>
      <div class="button-group">
        <button onclick="sendCommand('slow')" class="system-button button-
mode">
          Slow Speed
        </button>
        <button onclick="sendCommand('medium')" class="system-button button-
mode">
          Medium Speed
        </button>
            <button onclick="sendCommand('fast')" class="system-button button-
mode">
          Fast Speed
        </button>
      </div>
    </div>

    <div class="control-section">
      <h2>Remote Mode Controler</h2>
      <div class="directional-pad">
        <button onclick="sendCommand('forward')" class="dpad-button dpad-
forward" style="grid-column: 2; grid-row: 1;">
          ▲
        </button>
        <button onclick="sendCommand('left')" class="dpad-button dpad-left"
style="grid-column: 1; grid-row: 2;">
          ◀
        </button>
        <button onclick="sendCommand('stop')" class="dpad-button dpad-stop"
style="grid-column: 2; grid-row: 2;">
          ■
        </button>
```

```
            <button onclick="sendCommand('right')" class="dpad-button dpad-right"
style="grid-column: 3; grid-row: 2;">
                ▶
            </button>
            <button onclick="sendCommand('back')" class="dpad-button dpad-back"
style="grid-column: 2; grid-row: 3;">
                ▼
            </button>
        </div>
    </div>
</div>

<script>
    //Function to send commands to Arduino
    function sendCommand(cmd) {
        fetch(`/send_command/${cmd}`)
            .then(response => response.json())
            .then(data => {
                if (data.status === 'success') {
                    // Command successful
                }
            })
            .catch(error => console.error('Error:', error));
    }

    //Function to update sensors
    function updateSensors() {
        fetch('/get_sensors')
            .then(response => response.json())
            .then(data => {
                // Update ultrasonic sensors
                document.getElementById('us-middle').textContent =
data.ultrasonic.middle;
                document.getElementById('us-right').textContent = data.ultrasonic.right;
                document.getElementById('us-left').textContent = data.ultrasonic.left;

                // Update right IR sensors
                document.getElementById('ir-r-right').textContent = data.ir_right.right;
                document.getElementById('ir-r-middle').textContent =
data.ir_right.middle;
                document.getElementById('ir-r-left').textContent = data.ir_right.left;

                // Update left IR sensors
                document.getElementById('ir-l-right').textContent = data.ir_left.right;
                document.getElementById('ir-l-middle').textContent =
data.ir_left.middle;
                document.getElementById('ir-l-left').textContent = data.ir_left.left;
            })
            .catch(error => console.error('Error:', error));
    }
```

```
        //Update sensors ---> 300ms
        setInterval(updateSensors, 300);

        //Initial update when page loads
        updateSensors();
    </script>
</body>
</html>
```