Kuwait University

College of Computer Science & Engineering

Computer Engineering Department

CpE-434

Robotics

Final Report for Lane-Following Robot with Autonomous Obstacle Avoidance

Prepared by Team KAYO

Hala Almutairi
2211112873
Zahra'a Mohammad Alrashidi
2191118389

Instructor Name: Dr. Abdullah Alshaibani

TA Name: Eng. Maryam Aljame

Date: 15/5/2025

Table of Contents

List of Figures

List of Tables

1. 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.

2. 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.

Stakeholders of the System:

Table 1: Stakeholders of the Autonomous Bus System.

Name	Description	Responsibility	User
Students	Children or adolescents enrolled in elementary, middle, or high schools. Their ages typically range between 5 to 18 years old.	They are the main motivation for this project. Their responsibility is to get on the Autonomous bus system and wear their seatbelt to get to school and return home safely.	Yes.
Parents	The primary caregivers and guardians responsible for the upbringing and support of their children during their school years.	Give permission and trust for the Autonomous bus system to deliver their children to school and return them home safely.	No.
Schools	Formal education institutions where students are given guidance and support. They serve as foundational spaces for students to learn, socialize, and develop.	-Purchase the bus system from suppliers and set it up to use for student pickup and drop off. -Use the website to start the students pick up journey from their homes to school at the start of the school day and to start the drop off journey at the end of the school day.	Yes.
Bus supply companies	Companies that specialize in supplying new or old buses for public transportation, schools, or private companies. They are considered businesses that provide vehicles, parts, equipment related to buses.	-Invest in the autonomous bus system project. -Purchase stock of the system and rights to re-sell.	No.
Developers	Kayo team consisting of two students at the College of Engineering and Petroleum at Kuwait University	-Develop and test the system to ensure safety of the users. -Obtain hardware needed for building the system. -Install the system.	No.
Dr. Abdullah Alshaibani and Eng. Maryam Aljame	Computer Engineering faculty members with specialty in Robotics and AI systems, at the College of Engineering and Petroleum at Kuwait University.	Supervising the work of the autonomous bus system project and give feedback/suggestions.	No.

3. 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 and Comparison:

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 which shows its potential for improving traffic flow, reducing congestion, and promoting sustainability.

Comparison with the Kayo team system prototype:

Table 2: Kayo System vs AEV System Comparison.

Comparison	Kayo Autonomous System	AEV System
Road following Infrared sensors module	✓	
Ultrasonic sensors	✓	✓
Q-learning AI module	✓	
Website	✓	
Manual control mode	✓	
Uses batteries	✓	✓
DC motors	✓	✓

The Kayo team's autonomous robot system introduces several advanced features that distinguish it from the AEV system. One of the main differences is Kayo's use of infrared (IR) sensors for road following, allowing the robot to detect and avoid black borders on either side of the path, which is a functionality not present in the AEV system. Both systems use ultrasonic sensors for obstacle detection. Unlike the AEV system which utilizes GPS for navigation, Kayo's system uses a more advanced Q-learning AI model that enables dynamic, experience-based path optimization for navigating the road. Kayo's robot includes a web-based interface that allows users to control the robot in two modes: Start Manual Mode for manual control of the robots movements and Autonomous Mode where the trained AI handles navigation. The AEV system lacks a website interface and only supports a single autonomous driving mode. Despite these differences, both systems are powered by batteries and use DC motors to control wheel movement.

4. Project Specifications

a. The type of the 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.

5. 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. 3x One-Channel IR Infrared Sensors (3):

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

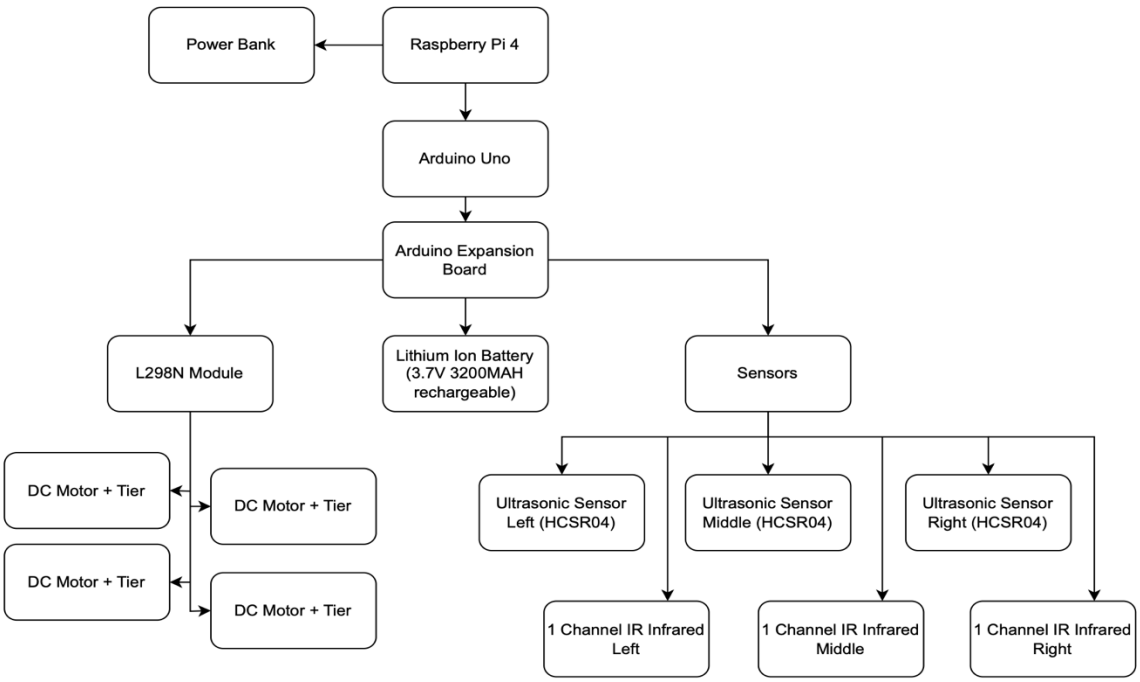


Figure 1: Components Connection.

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, and middle) for obstacle detection and three One-Channel IR Sensors (left, middle, and 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 an Arduino Uno working together to enable both manual (Phase 1) and autonomous control of a robot car as illustrated in Phase 2 flowchart. The Raspberry Pi 4 acts as the central controller, running a Flask-based web server that allows users to control the car, initiate training sessions, and monitor sensor data. The Arduino handles low-level motor control and real-time sensor readings.

Phase 2 System Flowchart :
https://ku365-my.sharepoint.com/:u:/g/personal/s2211112873_ku_edu_kw/EVyD_RTeKuxClmaIzbdWa5kBwkY7B1cqdZF6JeTK-BJIRQ

Phase 1 System Flowchart :

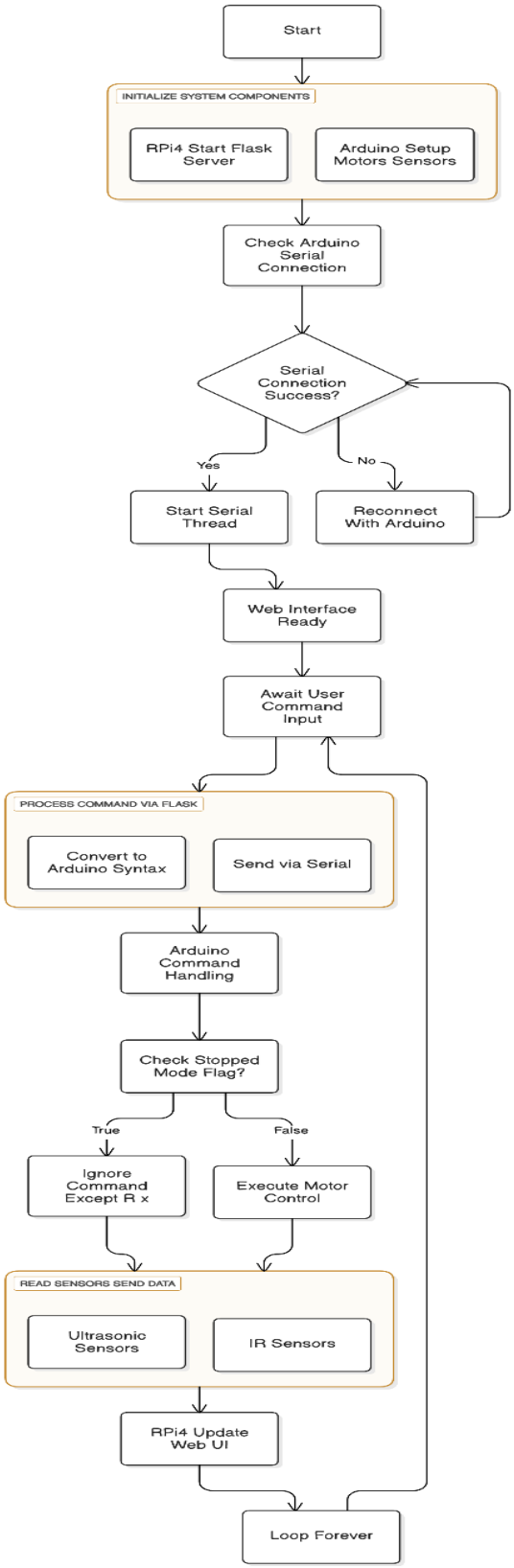


Figure 2: Flow Chart for System.

How It Works:

1. **Initialization:**
    - The Raspberry Pi boots Flask web server.
    - The Arduino initializes the motors and sensors.
    - A serial connection is established between the Raspberry Pi and Arduino, with retries if the connection fails.
    - If available, a previously saved Q-table is loaded for use in autonomous navigation.

2. **Web Interface Interaction:**
    - The web interface provides users with three features:
        - **Manual Control Mode:** Send directional commands like forward, back, left, and right.
        - **Sensor Monitoring**: View real-time line sensor states from the Arduino.
        - **Q-Learning Mode:** Start/stop/pause training and view the current reward that is received.
    - Commands issued from the web interface are processed by the Flask backend and forwarded to the Arduino over serial communication.

3. **Manual Mode:**
    - In manual mode, users can directly control the robot through on-screen buttons Forward/Back/Stop/Right/Left.
    - The Raspberry Pi sends Manual ('M') command strings to the Arduino to move the car.

4. **Q-Learning**

   The Q-learning module enables the robot to learn how to follow a marked path by interacting with its environment using rewards. This is implemented in App.py which include Model Agent backend that
   is running on the Raspberry Pi.

How the Q-Learning Module Works

Initialization:

- A Q-table is initialized with zeros for all valid states and actions.
- An R-table (Reward Table) is predefined with rewards based on sensor readings and desired behavior.

Hyperparameters:

- **Learning Rate (α)**: 0.1
- **Discount Factor (γ)**: 0.9
- **Exploration Rate (ε)**: 0.1

Training Execution:

- Initiated via the web interface ("Start Training" button).
- The robot repeatedly:
  - Reads the current state from the 3 IR sensors.
  - Selects an action using the ε-greedy policy.
  - Executes the action via the Arduino.
  - Calculates a reward from the R-table.
  - Updates the Q-table using:
  - Q[state][action] = Q[state][action] + α * (reward + γ * max(Q[next_state]) - Q[state][action])
- Episodes continue until Pause is pressed (Saved to model.pth) or stop to discard sessions and not save it into model.pth.

Saving Training Sessions:

- After training, the Q-table is saved to model.pth for loading across sessions.
- On startup, the Flask backend attempts to reload the saved model.pth if it exists.

States

Each state is represented by a binary string indicating IR sensor readings:

Table 3: Q-Learning States Table.

Sensor Reading	Meaning
000 , 101	Centered on the path (ideal)
001 , 100	Right/Left off
000 , 111	Robot lost or on boundary
010	End Of road there Left/Right turn

Actions

ACTIONS = ['forward', 'left', 'right', 'scan_left', 'scan_right', 'strongLeft', 'strongRight']

Table 4: Q-Learning Actions Table.

Action	Description
forward	Move straight
left	left turn
right	right turn
scan_left	Pause and scan Left to read new IR readings
scan_right	Pause and scan Right to read new IR readings
strongLeft	Sharp left
strongRight	Sharp right

Reward Table (R Table)

Each index in the R-table corresponds to the reward for each action taken in a specific sensor state:

```
R = {
   '000': [50, -25, -25, -25, -25, -25, -25],  # All white
   '001': [-1, 10, -10, -10, -10, -10, -10],   # Right black
   '010': [-20, -1, -1, -1, -1, 8, 8],             # Middle black
   '011': [-1, -1, -1, -20, 15, -1, -1],    # Middle+Right black
   '100': [-1, -10, 10, -10, -10, -10, -10],   # Left black
   '101': [12, -20, -20, -20, -20, -20, -20],  # Left+Right black
   '110': [-1, -1, -1, -20, 15, -1, -1],     # Left+Middle black
   '111': [-10, -10, -10, -10, -10, 10, 10]    # All black
```

5. Arduino Algorithm

The Arduino manage motors, sensors, and obstacle navigation based on commands from the Raspberry Pi and real-time sensor feedback.

Initialization:

- The Arduino initializes motor driver pins, three line tracking sensors (left, middle, right), and three ultrasonic sensors (left, middle, right) for distance measurement.
- Initial sensor states and distances are read to establish baseline conditions.
- The car starts in a stopped state.

Command Handling and Movement Control:

- The Arduino listens to serial commands sent from the Raspberry Pi.
- Supported commands include:

    - F (forward) with speed calibrated to 130 (PWM) and minor wheel adjustment (left wheel speed reduced by 10).
    - B (backward) with speed 150 and right wheel speed reduced by 8.
    - L (left turn) and R (right turn) both run motors at full speed (150 PWM).
    - S (stop) stops all motors immediately.
    - SCAN_LEFT and SCAN_RIGHT commands rotate the robot for 1 second (1000 ms).
    - STRONG_LEFT and STRONG_RIGHT perform longer turns lasting 2 seconds (2000 ms).
- Manual mode (M) allows for Car control until an exit command (X) is received.

Sensor Reading and Obstacle Detection:

- Ultrasonic sensors detect obstacles with a threshold of 20 cm for obstacle presence.
- Distances above 60 cm indicate a clear path.
- The robot maintains a safe following distance of 15 cm and considers distances above 30 cm as lost objects.
- Stable sensor readings are obtained by taking 3 consecutive samples with 10 ms delays each.
- Timeout cases in ultrasonic distance measurement return 70 cm (clear path assumption).
- The robot uses line tracking sensors to avoid leaving the path during obstacle navigation.
- After maneuvering, the robot moves forward for 300 ms to update sensor data.

Obstacle Avoidance Flow Chart:

https://ku365-my.sharepoint.com/:b:/g/personal/zaharaa_alrashidi_eng_ku_edu_kw/EdRrxlFsPiFPrzv_CIX8-6wBCNG-JSszai3f_ZTqsokfgw?e=EMQC6U

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. Furthermore implementing Q-Learning and machine learning algorithms proved to be challenging. The learning process was slow, and tuning the algorithm to work in real-time was difficult which made it face limited success during tests.

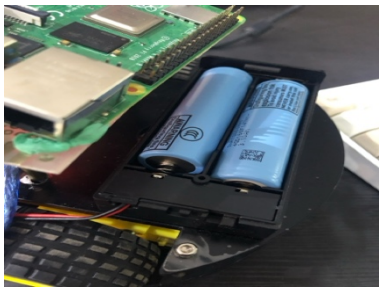
6. Final System User Manual

This section explains how the system works.

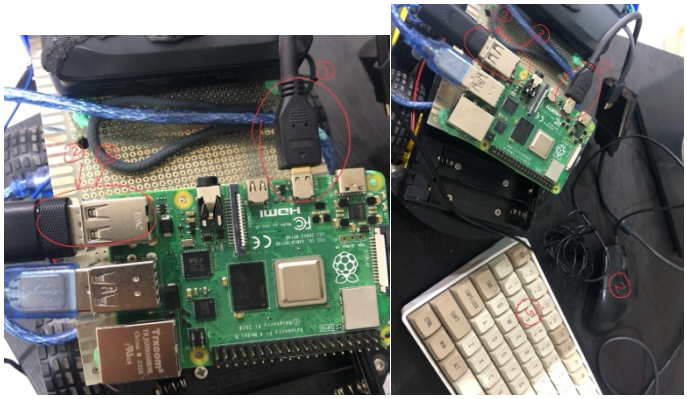
a. Robot Setup Guide:

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

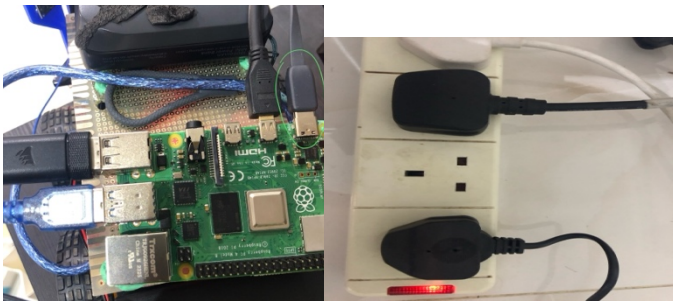
b. Set the batteries in the battery case of the robot.



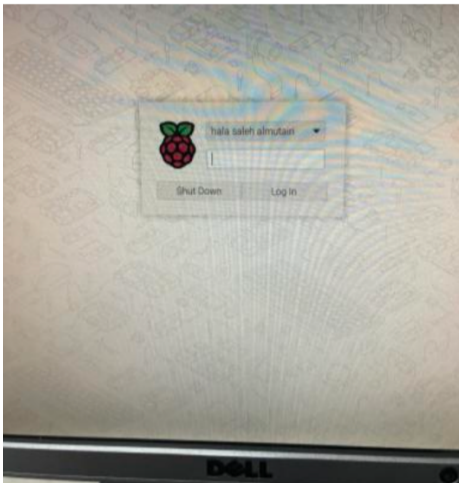
c. Connect screen monitor (using D-Sub to micro-HDMI cable), mouse and keyboard (using the two USB ports) to raspberry pi.



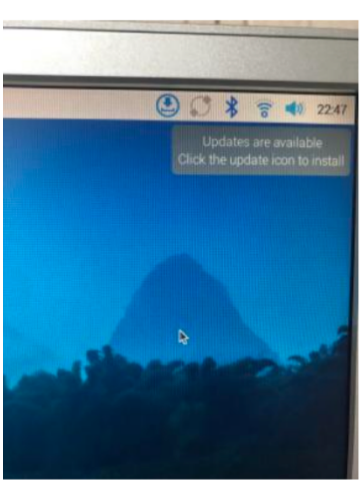
d. Connect the power bank to raspberry pi on the C-port and turn the screen power on.

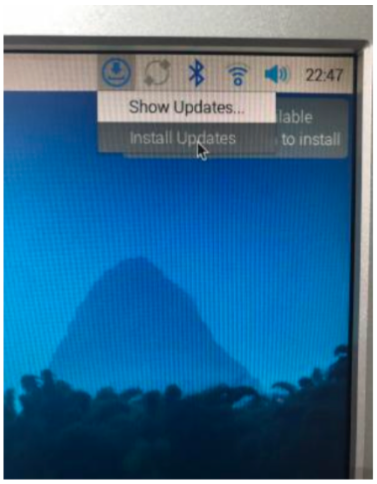


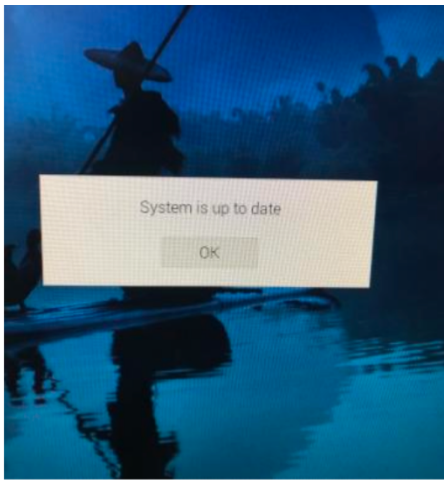
e. Login to account on raspberry pi.



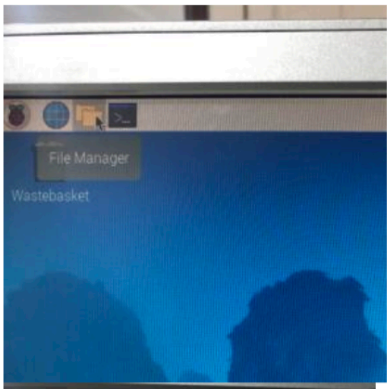
f. Make sure any updates are installed.

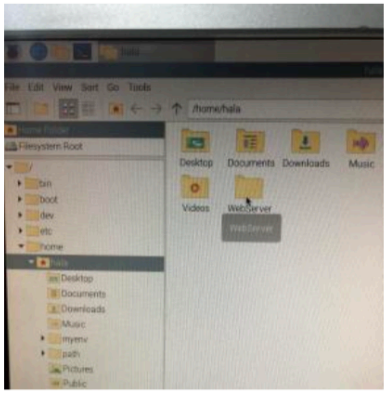


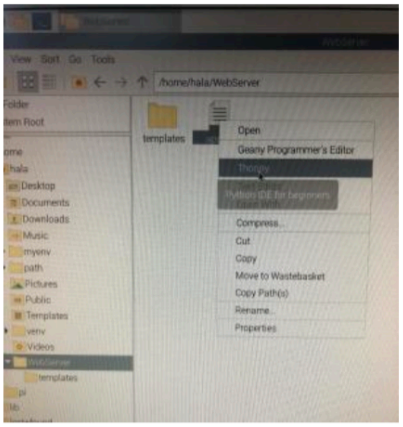


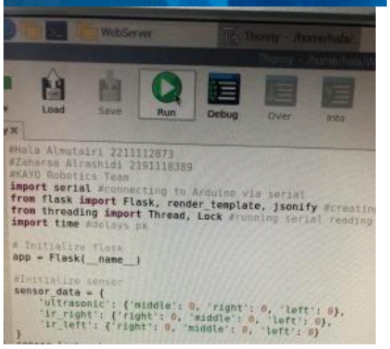


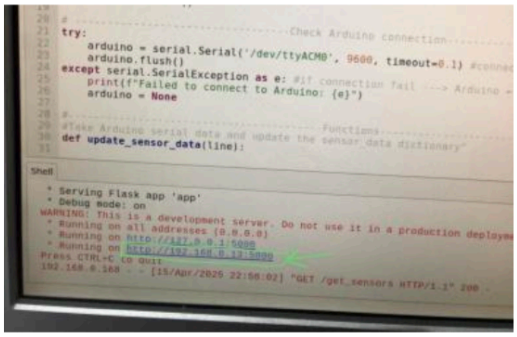
g. Run the raspberry pi code “app.py” stored in WebServer folder, run the code using Thonny. Memorize the local IP address you need to connect to that appears on the shell channel.



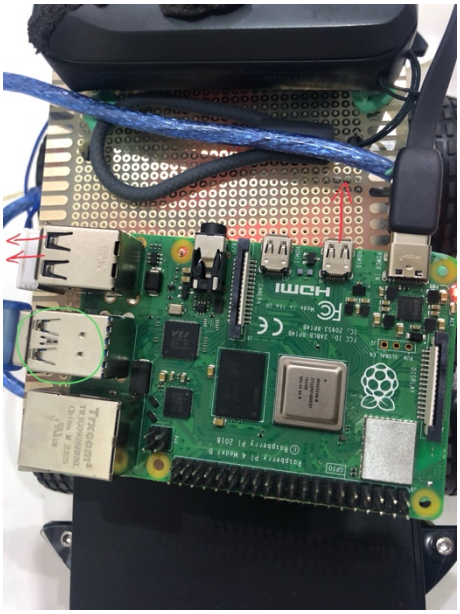




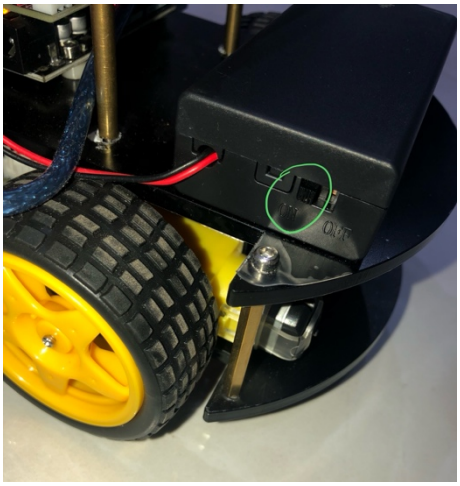




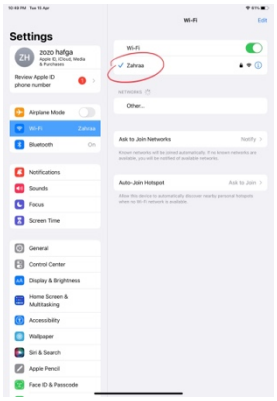
h. Disconnect raspberry pi from screen, mouse and keyboard. Careful to NOT disconnect the raspberry-to-Arduino cable.



i. Turn on motors battery.

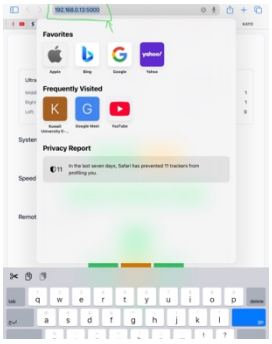


j. Connect your phone to a 5G network.



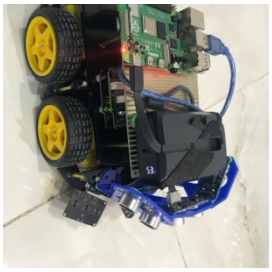
k. Open Safari or chrome browser and type the previous IP address with the port number 5000. Example:

Example: 192.168.0.13:5000

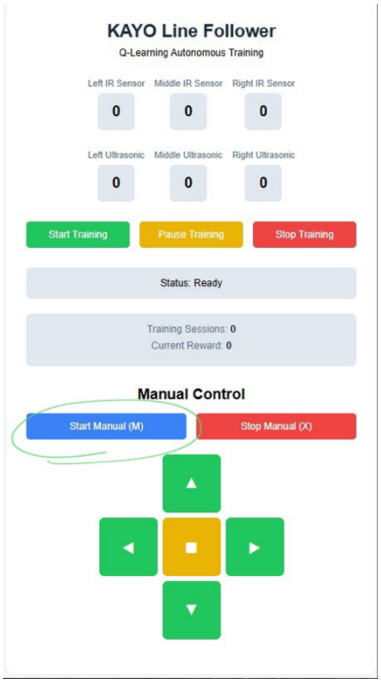


b. Robot Manual Controls Guide:

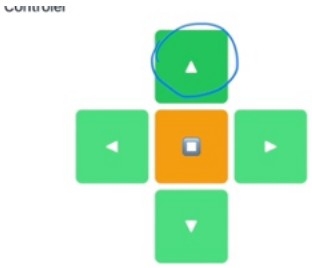
a. Place the robot on the ground.

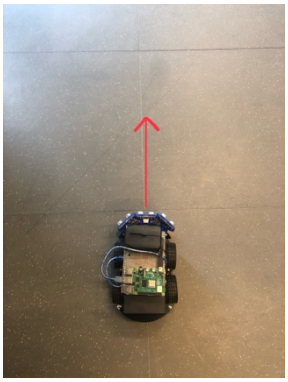


b. Press “Start Manual” to move the robot with user commands.

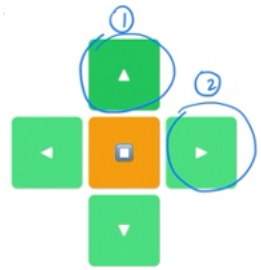


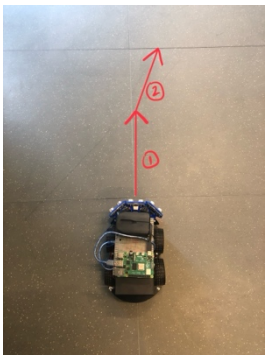
c. Press the upward facing arrow to move the robot forward



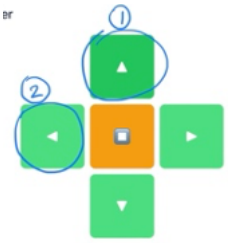


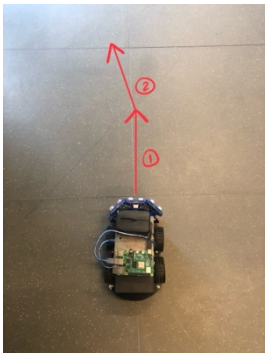
d. While moving forward, press the right arrow button to move the robot 20 degrees in the right direction then keep moving forward.



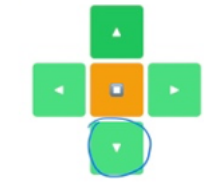


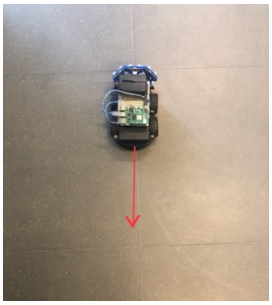
e. While moving forward, press the left arrow button to move the robot 20 degrees in the left direction then keep moving forward.



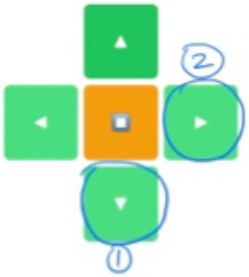
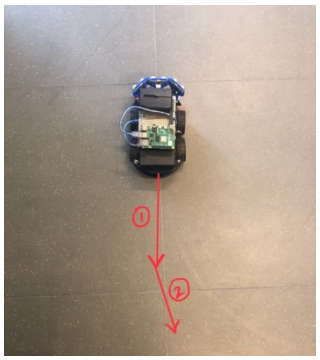


f. Press the down arrow button to move the robot backward.

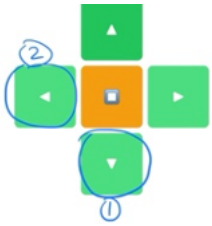
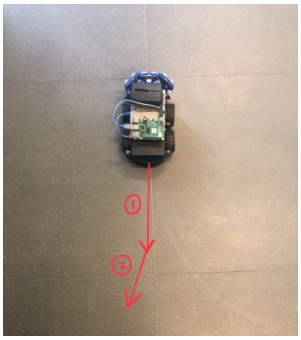




g. Press the right arrow button while moving backward to move the robot 20 degrees in the southeast direction then keep moving backward.

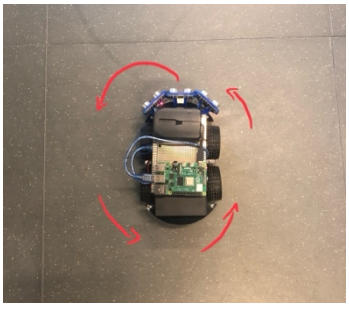



h. Press the left arrow button while moving backward to move the robot 20 degrees in the southwest direction then keep moving backward.

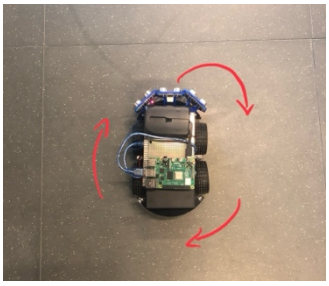



i. Press the middle button to stop the robot's movement.

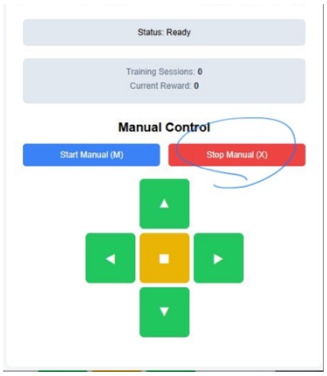
j. While the robot is stopped press the left arrow button to keep moving the robot 360 degrees to the left. Until the desired angle is achieved then continue with selecting the next command button (stop, forward, backward).



k. 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)

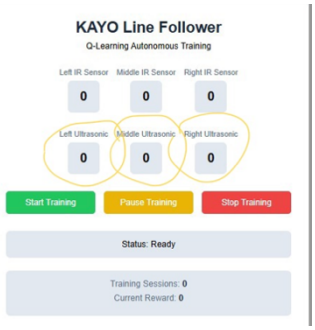


l. Press stop mode button for emergency exiting Manual mode

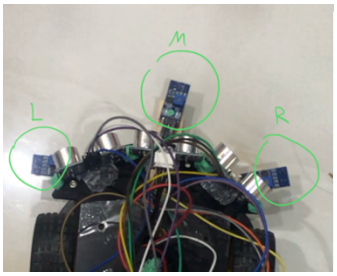


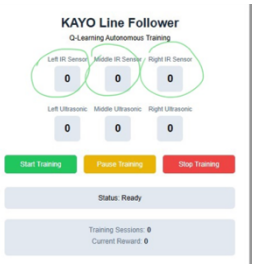
m. Sensor data will be displayed in their respective slots. The readings would be displayed with a delay of 300 milliseconds for better reading ability.

- 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.

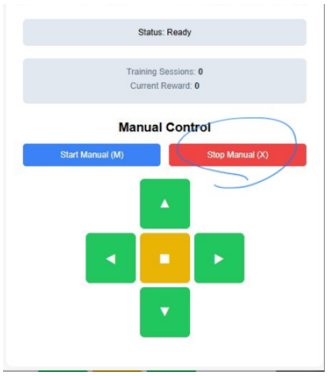




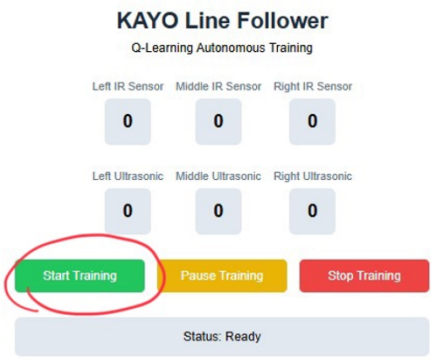
c. **Robot Autonomous Driving Mode Guide:**

Path Autonomous Model was Trained On:

1. To begin Autonomous driving with the trained model. First select Stop Manual to exit the manual mode.

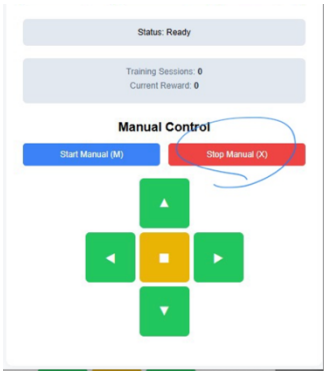


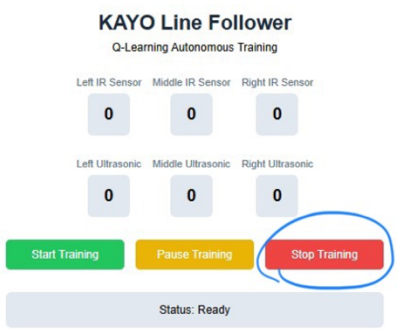
2. Select Start Training to begin the Autonomous Driving Mode.



d. Stop the System Safely:

a. Press the Stop Manual mode button on the website if the system was in Manual mode. Else if it were in Autonomous driving mode then press Stop Training button.





b. Turn off the battery supply from the motors.

c. 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- Start Manual Mode must be selected before trying to manually command the robot to move.

3- Must exit Start Manual Mode before selecting Start Training Mode (Autonomous Driving Mode).

Deficiencies Discovered During Testing:

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

2- The Autonomous Driving Mode ability to make choices is not up to the teams standards, still makes minor mistakes.

7. 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.

- Q-Learning Model:

  There were challenges in implementing the reward and penalty system in a way that allowed the robot to learn without becoming hesitant to make decisions. A balance was eventually achieved through multiple tests, trials, and errors.

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)
- Learned to implement Autonomous driving through Q-learning.

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. Optimizing the reward model to get better performance from the autonomous driving system.

- Reducing Cost:

  Explore inexpensive alternatives for components and tools to reduce the overall system cost.

8. 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
//********************************************** Library / Pins Define
*****************************************************//
// Right Line Tracking Sensors
#define LTR 4
#define LTM 3
#define LTL 2

// 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

//*************************************** Variable Initialization
*****************************************************//
int carSpeed = 135; // Car speed
bool stoppedMode = false; // Flag for Stop mode
String command;  // Command received from Raspberry Pi 4
String state = "stop"; // Track previous command (initial stop)

int LT_R = 0;
int LT_M = 0;
int LT_L = 0;

int LeftIR = LT_L;
int middelIR = LT_M;
int RightIR = LT_R;



// 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;

int MiddleState = 0;
int LeftState = 0;
int RightState = 0;

// Distance thresholds for obstacle detection and following
#define OBSTACLE_THRESHOLD 20     // Distance in cm to detect obstacle
#define FOLLOW_DISTANCE 15        // Distance to maintain when following
object
#define PATH_CLEAR_DISTANCE 60    // Distance considered "clear path"
#define MAX_FOLLOW_DISTANCE 30    // Maximum distance before considering
object lost

//****************************************************** Motion
Functions ******************************************************//
void stop(){
  digitalWrite(ENA, LOW);
  digitalWrite(ENB, LOW);
  Serial.println("Stop");
}

void forward(){
  int leftSpeed = carSpeed - 10; // Wheel calibration
  analogWrite(ENA, leftSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN4, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN1, HIGH);
}

void back(){
  int rightSpeed = carSpeed - 8; // Wheel calibration
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, rightSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN4, LOW);
}

void left(){
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN3, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN4, HIGH);
}
```

```
void right(){
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN2, LOW);
  digitalWrite(IN4, LOW);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN3, HIGH);
}

//********************************************** Strong Left/Right
Turns **********************************************//
void strongLeft() {
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  delay(2000);  // Longer turn
  stop();
}

void strongRight() {
  analogWrite(ENA, carSpeed);
  analogWrite(ENB, carSpeed);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  delay(2000);  // Longer turn
  stop();
}

//****************************************** Sensor Reading Stability
Function ******************************************//
int stableRead(int pin) {
  int lastState = digitalRead(pin);
  int count = 0;

  for (int i = 0; i < 3; i++) { // Take 3 consecutive readings
    if (digitalRead(pin) == lastState) count++;
    delay(10); // Short delay between readings
  }

  return (count >= 2) ? lastState : !lastState; // Majority filtering
}

//***************************Function to calc distance via ultrasonic sensor
with improved reliability*********************//
```

```
int Distance_test(int Trig, int Echo) {
  digitalWrite(Trig, LOW);
  delayMicroseconds(2);
  digitalWrite(Trig, HIGH);
  delayMicroseconds(20);
  digitalWrite(Trig, LOW);

  float Fdistance = pulseIn(Echo, HIGH); // Added timeout to prevent hanging

  // Handle timeout case
  if (Fdistance == 0) {
    return PATH_CLEAR_DISTANCE + 10; // Return a value indicating clear path if
timeout
  }

  Fdistance = Fdistance / 58;
  return (int)Fdistance;
}



//***************************** Survival mode - more intelligent border
following to find a way out*********************//
void Survive() {
  Serial.println("SURVIVAL MODE ACTIVATED");
  Serial.print("Left=");
  Serial.print(LeftState);
  Serial.print(", Middle=");
  Serial.print(MiddleState);
  Serial.print(", Right=");
  Serial.println(RightState);

  // Try to find the clearest direction
  if (LeftState > RightState) {
    // Left is clearer, turn left and try to follow something on the right
    Serial.println("Survival: trying left direction");
    left();
    delay(400);

  } else {
    // Right is clearer, turn right and try to follow something on the left
    Serial.println("Survival: trying right direction");
    right();
    delay(400);

  }


}
```

```
//************************ Functions to check for a path around obstacles and initiate following behavior*********************//
void checkForPath() {
  Serial.println("Checking for path around obstacle");
  int counterR =0;
  int counterM =0;
  int counterL =0;
  // Get the latest sensor readings
  MiddleState = Distance_test(Trig, Echo);
  LeftState = Distance_test(TrigL, EchoL);
  RightState = Distance_test(TrigR, EchoR);

  // Print all sensor readings for debugging
  Serial.print("Distances - Left: ");
  Serial.print(LeftState);
  Serial.print("cm, Middle: ");
  Serial.print(MiddleState);
  Serial.print("cm, Right: ");
  Serial.print(RightState);
  Serial.println("cm");

  // First check if middle is blocked
  if (MiddleState < OBSTACLE_THRESHOLD) {
    // Middle is blocked, check left and right paths
     Serial.println("Check path to left - turning left");
      left();
      delay(500);
      stop();
      delay(100);

      // Get the latest sensor readings
      MiddleState = Distance_test(Trig, Echo);
      LeftState = Distance_test(TrigL, EchoL);
      RightState = Distance_test(TrigR, EchoR);

    if (MiddleState > PATH_CLEAR_DISTANCE) {
      // Clear path on left, obstacle must be on right

      Serial.println("Clear path on left - turning left");
      while(RightState < OBSTACLE_THRESHOLD){
        forward();
        delay(700);
        stop();
        delay(100);
        RightState = Distance_test(TrigR, EchoR);
        if (LT_R){
          counterR++;
        }
        if (LT_M){
```

```
      counterM++;
     }
     if (LT_L){
      counterL++;
     }
    }

    while ((counterR < 5) && (counterM < 5) && (counterL < 5) && (RightState >
PATH_CLEAR_DISTANCE)){
     forward();
     delay(400);
     right();
     delay(200);
     stop();
     delay(100);
     if (LT_R){
      counterR++;
     }
    if (LT_M){
      counterM++;
     }
    if (LT_L){
      counterL++;
     }

    }




  }
  else if (MiddleState < PATH_CLEAR_DISTANCE) {

   Serial.println("Check path to right - turning right");
    right();
    delay(1000);
    stop();
    delay(100);
    // Clear path on right, obstacle must be on left
    // Get the latest sensor readings
    MiddleState = Distance_test(Trig, Echo);
    LeftState = Distance_test(TrigL, EchoL);
    RightState = Distance_test(TrigR, EchoR);

  if (MiddleState > PATH_CLEAR_DISTANCE) {
    // Clear path on left, obstacle must be on right

    Serial.println("Clear path on left - turning left");
    while(LeftState < OBSTACLE_THRESHOLD){
```

```
      forward();
      delay(700);
      stop();
      delay(100);
      LeftState = Distance_test(TrigL, EchoL);
      if (LT_R){
        counterR++;
      }
      if (LT_M){
        counterM++;
      }
      if (LT_L){
        counterL++;
      }
    }

    while ((counterR < 5) && (counterM < 5) && (counterL < 5) && (LeftState >
PATH_CLEAR_DISTANCE)){
      forward();
      delay(400);
      left();
      delay(200);
      stop();
      delay(100);
      if (LT_R){
        counterR++;
      }
     if (LT_M){
        counterM++;
      }
     if (LT_L){
        counterL++;
      }

    }

  }
  else {
    // No clear path, enter survival mode to find a way out
    Serial.println("No clear path - entering survival mode");
    Survive();
  }
 }
 // If middle is clear but objects on sides
 else if (LeftState < OBSTACLE_THRESHOLD) {
  // Object on left, follow its border
  Serial.println("Object detected on left - will follow left border");

 }
 else if (RightState < OBSTACLE_THRESHOLD) {
```

```
    // Object on right, follow its border
    Serial.println("Object detected on right - will follow right border");

  }
  else {
    // No obstacles detected, continue normal operation

    Serial.println("Path clear, resuming normal operation");
  }
}


  // Move forward a bit to get new sensor readings
  forward();
  delay(300);
  stop();
}

//****************************************** Scanning Left and Right with
Correction ******************************************//
void scanLeft() {
  left();
  delay(1000);
  stop();
}

void scanRight() {
  right();
  delay(1000);
  stop();

  // Check if center remains active after scanning
}



//************************************************ Execution Code
*************************************************************//
void setup() {
  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(LTR, INPUT);
  pinMode(LTM, INPUT);
  pinMode(LTL, INPUT);

  LT_R = digitalRead(LTR);
  LT_M = digitalRead(LTM);
  LT_L = digitalRead(LTL);

  // Take initial distance readings
  MiddleState = Distance_test(Trig, Echo);
  LeftState = Distance_test(TrigL, EchoL);
  RightState = Distance_test(TrigR, EchoR);

  stop();
  state = "stop";
}

void loop() {

  carSpeed = 150;
  // Hold previous state unless multiple confirmed changes
  LeftIR = stableRead(2);
  middelIR = stableRead(3);
  RightIR = stableRead(4);

  if (Serial.available() > 0) {
    String cmd = Serial.readStringUntil('\n');
    cmd.trim();
     if (cmd == "M") {
      handleManual();
    }else if (cmd == "F") {
      carSpeed = 130; // Car speed
      while ((LeftIR == LT_L) && (middelIR == LT_M) && (RightIR == LT_R) &&
!((RightState <= OBSTACLE_THRESHOLD) || (MiddleState <=
OBSTACLE_THRESHOLD) || (LeftState <= OBSTACLE_THRESHOLD))) {
        forward();
        delay(50); // Short delay for stability
        LeftIR = stableRead(2);
        middelIR = stableRead(3);
        RightIR = stableRead(4);
        MiddleState = Distance_test(Trig, Echo);
        RightState = Distance_test(TrigR, EchoR);
        LeftState = Distance_test(TrigL, EchoL);

        // Break loop if stop command is received
        if (Serial.available() > 0) {
          String newCmd = Serial.readStringUntil('\n');
          newCmd.trim();
```

```
        if (newCmd == "S") {
          stop();
          break;
        }
      }
    }
    back();
    delay(300);
  }
  else if (cmd == "B") back();
  else if (cmd == "L") left();
  else if (cmd == "R") right();
  else if (cmd == "S") stop();
  else if (cmd == "SCAN_LEFT") scanLeft();
  else if (cmd == "SCAN_RIGHT") scanRight();
  else if (cmd == "STRONG_LEFT") strongLeft();
  else if (cmd == "STRONG_RIGHT") strongRight();
}


// Get latest sensor readings
  MiddleState = Distance_test(Trig, Echo);
  RightState = Distance_test(TrigR, EchoR);
  LeftState = Distance_test(TrigL, EchoL);

  // Print sensor readings for debugging
  Serial.print("US_L:");
  Serial.print(LeftState);
  Serial.print(" US_M:");
  Serial.print(MiddleState);
  Serial.print(" US_R:");
  Serial.println(RightState);

  // Normal operation - no object following
  // Check if any sensor detects an obstacle
  if ((RightState <= OBSTACLE_THRESHOLD) || (MiddleState <=
OBSTACLE_THRESHOLD) || (LeftState <= OBSTACLE_THRESHOLD)) {
    stop();
    delay(100); // Pause to get stable readings
    checkForPath(); // Determine how to navigate around obstacle
  }


  // Send sensor data to Raspberry Pi 4
  Serial.print("IR_R:"); Serial.println(LT_R);
  Serial.print("IR_M:"); Serial.println(LT_M);
  Serial.print("IR_L:"); Serial.println(LT_L);
}
```

```
void handleManual() {
  bool manualMode = true;
  while (manualMode) {
    if (Serial.available() > 0) {
      String cmd = Serial.readStringUntil('\n');
      cmd.trim();
      cmd.toUpperCase(); // Make command case-insensitive

      if (cmd == "F") {
        forward();
      }
      else if (cmd == "B") {
        back();
      }
      else if (cmd == "L") {
        left();
      }
      else if (cmd == "R") {
        right();
      }
      else if (cmd == "S") {
        stop();
      }
      else if (cmd == "X") {
        stop();
        manualMode = false;
      }
    }
  }
}
```

2- Raspberry Pi “app.py” Code:

```
#Hala Almutairi 2211112873
#Zaharaa Alrashidi 2191118389
#KAYO Robotics Team
sys.path.insert(0,'/home/hala/path/to/venv/lib/python3.11/site-packages')
import serial
from flask import Flask, render_template, jsonify
from threading import Thread, Lock
import time
import random
import numpy as np
from collections import deque
import torch
import torch.nn as nn
import torch.optim as optim
import torch.nn.functional as F
```

```
import os

# Initialize Flask app
app = Flask(__name__)

# Sensor data
sensor_data = {
    'IR_R': 0,
    'IR_M': 0,
    'IR_L': 0,
    'training_sessions': 0,
    'current_reward': 0
}
sensor_lock = Lock()

# Training control
training_thread = None
training_active = False
training_lock = Lock()
paused = False
stop_session = False

# Q-Learning constants
MAX_MEMORY = 100_000
BATCH_SIZE = 1000
LR = 0.001

# Action definitions
ACTIONS = ['forward', 'left', 'right', 'scan_left', 'scan_right','strongLeft','strongRight']
ACTION_MAP = {
    'forward': 'F',
    'left': 'L',
    'right': 'R',
    'strongLeft':'SL',
    'strongRight':'SR',
    'scan_left': 'SCAN_LEFT',
    'scan_right': 'SCAN_RIGHT',
    'stop': 'S',
    'back': 'B',
    'M': 'M',
    'X': 'X',
}

# Neural Network
class Linear_QNet(nn.Module):
    def __init__(self, input_size, hidden_size, output_size):
        super().__init__()
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.linear2 = nn.Linear(hidden_size, output_size)
```

```
    def forward(self, x):
        x = F.relu(self.linear1(x))
        x = self.linear2(x)
        return x

    def save(self, file_name='model.pth'):
        model_folder_path = '/home/hala/WebServer'
        file_name = os.path.join(model_folder_path, file_name)
        torch.save(self.state_dict(), file_name)

class QTrainer:
    def __init__(self, model, lr, gamma):
        self.lr = lr
        self.gamma = 0.9
        self.model = model
        self.optimer = optim.Adam(model.parameters(), lr=self.lr)
        self.criterion = nn.MSELoss()

    def train_step(self, state, action, reward, next_state, done):
        state = torch.tensor(state, dtype=torch.float)
        next_state = torch.tensor(next_state, dtype=torch.float)
        action = torch.tensor(action, dtype=torch.long)
        reward = torch.tensor(reward, dtype=torch.float)

        if len(state.shape) == 1:
            state = torch.unsqueeze(state, 0)
            next_state = torch.unsqueeze(next_state, 0)
            action = torch.unsqueeze(action, 0)
            reward = torch.unsqueeze(reward, 0)
            done = (done, )

        pred = self.model(state)
        target = pred.clone()
        for idx in range(len(done)):
            Q_new = reward[idx]
            if not done[idx]:
                Q_new = reward[idx] + self.gamma *
torch.max(self.model(next_state[idx]))
            target[idx][torch.argmax(action[idx]).item()] = Q_new

        self.optimer.zero_grad()
        loss = self.criterion(target, pred)
        loss.backward()
        self.optimer.step()

class Agent:
    def __init__(self):
        self.n_games = 0
        self.epsilon = 0
        self.gamma = 0.9
```

```
        self.memory = deque(maxlen=MAX_MEMORY)
        self.state_memory = deque(maxlen=3)
        self.model = Linear_QNet(3, 256, len(ACTIONS))
        self.trainer = QTrainer(self.model, lr=LR, gamma=self.gamma)
        self._safe_load_model()

    def _safe_load_model(self, file_name='model.pth'):
        model_folder_path = '/home/hala/WebServer'
        file_name = os.path.join(model_folder_path, file_name)
        if not os.path.exists(file_name):
            print("No saved model found. Starting fresh.")
            self.n_games = 0
            self.epsilon = 0
            return
        try:
            checkpoint = torch.load(file_name)
            self.model.load_state_dict(checkpoint['model_state_dict'])
            self.trainer.optimer.load_state_dict(checkpoint['optimizer_state_dict'])
            self.n_games = checkpoint.get('n_games', 0)
            self.epsilon = checkpoint.get('epsilon', 0)
            print(f"Loaded model with {self.n_games} games")
        except Exception as e:
            print(f"Error loading model: {e}")

    def save_model(self, file_name='model.pth'):
        model_folder_path = '/home/hala/WebServer'
        os.makedirs(model_folder_path, exist_ok=True)
        file_name = os.path.join(model_folder_path, file_name)
        torch.save({
            'model_state_dict': self.model.state_dict(),
            'optimizer_state_dict': self.trainer.optimer.state_dict(),
            'n_games': self.n_games,
            'epsilon': self.epsilon,
        }, file_name)
        print(f"Saved model with {self.n_games} games")

    def get_state(self):
        with sensor_lock:
            raw_state = np.array([sensor_data['IR_L'], sensor_data['IR_M'],
sensor_data['IR_R']])
            self.state_memory.append(raw_state)
            if len(self.state_memory) >= 3:
                filtered_state = np.round(np.mean(self.state_memory, axis=0)).astype(int)
            else:
                filtered_state = raw_state
            return filtered_state

    def get_action(self, state):
        self.epsilon = max(20, 80 - self.n_games)
        if random.randint(0, 100) < self.epsilon:
```

```
            move = random.randint(0, len(ACTIONS) - 1)
        else:
            state_tensor = torch.tensor(state, dtype=torch.float)
            prediction = self.model(state_tensor)
            move = torch.argmax(prediction).item()
        final_move = [0] * len(ACTIONS)
        final_move[move] = 1
        return final_move

    def remember(self, state, action, reward, next_state, done):
        self.memory.append((state, action, reward, next_state, done))

    def train_short_memory(self, state, action, reward, next_state, done):
        self.trainer.train_step(state, action, reward, next_state, done)

    def train_long_memory(self):
        if len(self.memory) > BATCH_SIZE:
            mini_sample = random.sample(self.memory, BATCH_SIZE)
        else:
            mini_sample = self.memory
        states, actions, rewards, next_states, dones = zip(*mini_sample)
        self.trainer.train_step(states, actions, rewards, next_states, dones)

# Initialize agent
agent = Agent()

# Arduino setup
try:
    arduino = serial.Serial('/dev/ttyACM0', 9600, timeout=0.1)
    arduino.flush()
    print("Arduino connected successfully")
except serial.SerialException as e:
    print(f"Failed to connect to Arduino: {e}")
    arduino = None

def update_sensor_data(line):
    try:
        if not line or ':' not in line:
            return False
        key, value = line.split(':', 1)
        value = int(value.strip())
        with sensor_lock:
            if key in sensor_data:
                sensor_data[key] = value
        return True
    except ValueError:
        return False

def read_serial_data():
    while True:
```

```
        try:
            if arduino and arduino.in_waiting > 0:
                try:
                    line = arduino.readline().decode('utf-8').strip()
                except UnicodeDecodeError:
                    line = arduino.readline().decode('utf-8', errors='ignore').strip()
                if line:
                    update_sensor_data(line)
            time.sleep(0.001)
        except Exception as e:
            print(f"Serial read error: {e}")
            time.sleep(1)

def calculate_reward(old_state, action, new_state):
    reward = 0
    L, M, R = new_state

    # 1) All white – lost the path
    if (L, M, R) == (0, 0, 0):
        if action == 'forward':
            reward += 50
        else:
            reward -= 25

    # 2) Drifted left – only left sensor sees black
    elif (L, M, R) == (1, 0, 0):
        if action == 'right':
            reward += 10
        elif action == 'left':
            reward -= 10

    # 3) Drifted right – only right sensor sees black
    elif (L, M, R) == (0, 0, 1):
        if action == 'left':
            reward += 10
        elif action == 'right':
            reward -= 10

    # 4) Centered on the line – middle sensor sees black
    elif (L, M, R) == (0, 1, 0):
        if action in ('scan_left', 'scan_right'):
            reward += 8
        elif action == 'forward':
            reward -= 20

    # 5) Too far left – left + middle sensors see black
    elif (L, M, R) == (1, 1, 0):
        if action == 'strongRight':
            reward += 15
        elif action == 'strongLeft':
```

```
            reward -= 20

    # 6) Too far right – middle + right sensors see black
    elif (L, M, R) == (0, 1, 1):
        if action == 'strongLeft':
            reward += 15
        elif action == 'strongRight':
            reward -= 20

    # 7) Both edges black (intersection) – go straight through
    elif (L, M, R) == (1, 0, 1):
        if action == 'forward':
            reward += 12
        else:
            reward -= 20

    # 8) All black – completely off-track
    elif (L, M, R) == (1, 1, 1):
        if action in ('scan_left', 'scan_right'):
            reward += 10
        else:
            reward -= 10

    return reward



def train():
    global agent, training_active, paused, stop_session
    print("Training thread started")
    total_reward = 0
    iteration = 0

    try:
        while training_active:
            iteration += 1
            print(f"Training iteration {iteration}, Sessions: {agent.n_games}")

            if stop_session:
                print("Session stopped and discarded.")
                stop_session = False
                total_reward = 0
                continue

            state_old = agent.get_state()

            if paused:
                # Get current action before processing pause
                final_move = agent.get_action(state_old)
                action_idx = np.argmax(final_move)
```

```
        action = ACTIONS[action_idx]

        print(f"Pausing session. Current n_games: {agent.n_games}")
        reward = -40
        next_state = agent.get_state()

        agent.train_short_memory(state_old, final_move, reward, next_state, True)
        agent.remember(state_old, final_move, reward, next_state, True)

        total_reward += reward
        agent.n_games += 1
        agent.train_long_memory()

        with sensor_lock:
            sensor_data.update({
                'training_sessions': agent.n_games,
                'current_reward': total_reward
            })

        total_reward = 0
        training_active = False
        print("Training deactivated (paused)")
        paused = False
        agent.save_model()
        continue

    # Normal training
    final_move = agent.get_action(state_old)
    action_idx = np.argmax(final_move)
    action = ACTIONS[action_idx]

    if arduino:
        try:
            arduino.write((ACTION_MAP[action] + '\n').encode())
        except Exception as e:
            print(f"Error sending command: {e}")

    time.sleep(0.1)
    state_new = agent.get_state()
    reward = calculate_reward(state_old, action, state_new)
    total_reward += reward

    agent.train_short_memory(state_old, final_move, reward, state_new, False)
    agent.remember(state_old, final_move, reward, state_new, False)

    with sensor_lock:
        sensor_data.update({
            'training_sessions': agent.n_games,
            'current_reward': total_reward,
        })
```

```
            if not training_active:
                print("Training deactivated")
                break

    except Exception as e:
        print(f"Exception in training loop: {e}")
        import traceback
        traceback.print_exc()



# Flask routes
@app.route('/')
def index():
    return render_template('index.html')

@app.route('/get_sensors')
def get_sensors():
    with sensor_lock:
        return jsonify(sensor_data)

@app.route('/send_command/<cmd>')
def send_command(cmd):
    if arduino and cmd in ACTION_MAP:
        arduino.write((ACTION_MAP[cmd] + '\n').encode())
        return jsonify({'status': 'success'})
    return jsonify({'status': 'error'})

@app.route('/pause_training')
def pause_training():
    global paused
    print("Pause request received")
    paused = True
    if arduino:
        try:
            arduino.write((ACTION_MAP['stop'] + '\n').encode())
        except Exception as e:
            print(f"Error sending stop command: {e}")
    return jsonify({
        'status': 'success',
        'message': 'Training paused',
        'n_games': agent.n_games
    })

@app.route('/stop_training')
def stop_session_route():
    global stop_session, training_active
    stop_session = True
    training_active = False
```

```
    if arduino:
        try:
            arduino.write((ACTION_MAP['stop'] + '\n').encode())
        except Exception as e:
            print(f"Error sending stop command: {e}")
    return jsonify({
        'status': 'success',
        'message': 'Session stopped',
        'n_games': agent.n_games
    })

@app.route('/start_training')
def start_training():
    global training_thread, training_active, paused
    paused = False
    training_active = True

    if training_thread and training_thread.is_alive():
        return jsonify({'status': 'error', 'message': 'Training already running'})

    training_thread = Thread(target=train)
    training_thread.daemon = True
    training_thread.start()

    return jsonify({
        'status': 'success',
        'message': 'Training started',
        'n_games': agent.n_games
    })

if __name__ == '__main__':
    if arduino:
        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:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0" />
    <title>KAYO Robotics - Q Learning</title>
    <style>
      :root {
        --bg: #f8fafc;
        --card: #ffffff;
        --text: #1e293b;
        --text-light: #64748b;

        --green-500: #22c55e;
        --green-600: #16a34a;

        --red-500: #ef4444;
        --red-600: #dc2626;

        --blue-500: #3b82f6;
        --blue-600: #2563eb;

        --yellow-500: #eab308;
        --yellow-600: #ca8a04;

        --gray-200: #e2e8f0;
        --gray-300: #cbd5e1;

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

      * {
        box-sizing: border-box;
        margin: 0;
        padding: 0;
        font-family: "Inter", -apple-system, BlinkMacSystemFont, sans-serif;
      }

      body {
        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: 600px;
  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-display {
  display: flex;
  justify-content: center;
  gap: 1rem;
  margin-bottom: 2rem;
}

.sensor-item {
  display: flex;
  flex-direction: column;
  align-items: center;
  gap: 0.5rem;
}

.sensor-label {
  font-size: 0.9rem;
  color: var(--text-light);
}

.sensor-value {
  width: 60px;
  height: 60px;
  background-color: var(--gray-200);
  border-radius: 8px;
  display: flex;
  align-items: center;
  justify-content: center;
```

```
  font-weight: bold;
  font-size: 1.5rem;
}

.control-buttons {
  display: grid;
  grid-template-columns: repeat(3, 1fr);
  gap: 1rem;
  margin-bottom: 2rem;
}

.control-button {
  padding: 0.75rem 1.5rem;
  border-radius: 6px;
  border: none;
  font-weight: 500;
  font-size: 1rem;
  cursor: pointer;
  transition: all 0.2s;
  color: white;
}

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

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

.stop-button {
  background-color: var(--red-500);
}

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

.pause-button {
  background-color: var(--yellow-500);
}

.pause-button:hover {
  background-color: var(--yellow-600);
  transform: translateY(-2px);
}

.status-display {
```

```
  text-align: center;
  padding: 1rem;
  border-radius: 8px;
  background-color: var(--gray-200);
  margin-top: 1rem;
  font-weight: 500;
}

.training-info {
  margin-top: 1.5rem;
  padding: 1rem;
  background-color: var(--gray-200);
  border-radius: 8px;
  text-align: center;
}

.training-info p {
  margin-bottom: 0.5rem;
  color: var(--text-light);
}

.training-info span {
  font-weight: 600;
  color: var(--text);
}

/* Manual Control Styles */
.manual-control {
  margin-top: 2rem;
  width: 100%;
}

.manual-control-buttons {
  display: grid;
  grid-template-columns: 1fr 1fr;
  gap: 1rem;
  margin-bottom: 1.5rem;
}

.manual-button {
  padding: 0.75rem 1.5rem;
  border-radius: 6px;
  border: none;
  font-weight: 500;
  font-size: 1rem;
  cursor: pointer;
  transition: all 0.2s;
  color: white;
}
```

```
.start-manual-button {
  background-color: var(--blue-500);
}

.start-manual-button:hover {
  background-color: var(--blue-600);
  transform: translateY(-2px);
}

.stop-manual-button {
  background-color: var(--red-500);
}

.stop-manual-button:hover {
  background-color: var(--red-600);
  transform: translateY(-2px);
}

.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(--yellow-500);
      color: white;
    }

    .dpad-stop:hover {
      background-color: var(--yellow-600);
      transform: translateY(-2px);
    }
  </style>
</head>
<body>
  <div class="control-panel">
    <div class="header">
      <h1>KAYO Line Follower</h1>
      <p>Q-Learning Autonomous Training</p>
    </div>

    <!-- Sensor display -->
    <div class="sensor-display">
      <div class="sensor-item">
        <span class="sensor-label">Left IR Sensor</span>
        <div class="sensor-value" id="ir-left">0</div>
      </div>
      <div class="sensor-item">
        <span class="sensor-label">Middle IR Sensor</span>
        <div class="sensor-value" id="ir-middle">0</div>
      </div>
      <div class="sensor-item">
        <span class="sensor-label">Right IR Sensor</span>
        <div class="sensor-value" id="ir-right">0</div>
      </div>
    </div>

    <!-- Ultrasonic Sensor display -->
    <div class="sensor-display" style="margin-top: 1rem">
      <div class="sensor-item">
```

```
    <span class="sensor-label">Left Ultrasonic</span>
    <div class="sensor-value" id="ultrasonic-left">0</div>
  </div>
  <div class="sensor-item">
    <span class="sensor-label">Middle Ultrasonic</span>
    <div class="sensor-value" id="ultrasonic-middle">0</div>
  </div>
  <div class="sensor-item">
    <span class="sensor-label">Right Ultrasonic</span>
    <div class="sensor-value" id="ultrasonic-right">0</div>
  </div>
</div>

<!-- Control buttons -->
<div class="control-buttons">
  <button onclick="startTraining()" class="control-button start-button">
    Start Training
  </button>
  <button onclick="pauseTraining()" class="control-button pause-button">
    Pause Training
  </button>
  <button onclick="stopTraining()" class="control-button stop-button">
    Stop Training
  </button>
</div>

<!-- Status display -->
<div class="status-display" id="status-display">Status: Ready</div>

<!-- Training info -->
<div class="training-info">
  <p>Training Sessions: <span id="training-sessions">0</span></p>
  <p>Current Reward: <span id="current-reward">0</span></p>
</div>

<!-- Manual Control Section -->
<div class="manual-control">
  <h2 style="text-align: center; margin-bottom: 1rem">Manual Control</h2>

  <div class="manual-control-buttons">
    <button
      onclick="sendCommand('M')"
      class="manual-button start-manual-button"
    >
      Start Manual (M)
    </button>
    <button
      onclick="sendCommand('X')"
      class="manual-button stop-manual-button"
    >
```

```
        Stop Manual (X)
      </button>
    </div>

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

<script>
  // Function to update sensor display
  function updateSensors() {
    fetch("/get_sensors")
      .then((response) => response.json())
      .then((data) => {
```

```
      // Update sensor values
      document.getElementById("ir-left").textContent = data.IR_L;
      document.getElementById("ir-middle").textContent = data.IR_M;
      document.getElementById("ir-right").textContent = data.IR_R;

      // Update ultrasonic values if they exist in the response
      if (data.ULTRASONIC_L !== undefined) {
        document.getElementById("ultrasonic-left").textContent =
          data.ULTRASONIC_L;
      }
      if (data.ULTRASONIC_M !== undefined) {
        document.getElementById("ultrasonic-middle").textContent =
          data.ULTRASONIC_M;
      }
      if (data.ULTRASONIC_R !== undefined) {
        document.getElementById("ultrasonic-right").textContent =
          data.ULTRASONIC_R;
      }

      // Update sensor colors based on value
      updateSensorColor("ir-left", data.IR_L);
      updateSensorColor("ir-middle", data.IR_M);
      updateSensorColor("ir-right", data.IR_R);

      // Update training info
      document.getElementById("training-sessions").textContent =
        data.training_sessions || 0;
      document.getElementById("current-reward").textContent =
        data.current_reward || 0;
    })
    .catch((error) => console.error("Error:", error));
}

function updateSensorColor(elementId, value) {
  const element = document.getElementById(elementId);
  if (value === 1) {
    element.style.backgroundColor = "#000000";
    element.style.color = "#ffffff";
  } else {
    element.style.backgroundColor = "#e2e8f0";
    element.style.color = "#000000";
  }
}

// Training control functions
function startTraining() {
  fetch("/start_training")
    .then((response) => response.json())
    .then((data) => {
      updateStatus(data.message);
```

```
    })
    .catch((error) => {
      console.error("Error:", error);
      updateStatus("Error starting training");
    });
}

function pauseTraining() {
  fetch("/pause_training")
    .then((response) => response.json())
    .then((data) => {
      updateStatus(data.message);
    })
    .catch((error) => {
      console.error("Error:", error);
      updateStatus("Error pausing training");
    });
}

function stopTraining() {
  fetch("/stop_training")
    .then((response) => response.json())
    .then((data) => {
      updateStatus(data.message);
    })
    .catch((error) => {
      console.error("Error:", error);
      updateStatus("Error stopping training");
    });
}

// Manual control function
function sendCommand(cmd) {
  fetch(`/send_command/${cmd}`)
    .then((response) => response.json())
    .then((data) => {
      if (data.status === "success") {
        // Command successful
        if (cmd === "M") {
          updateStatus("Manual Mode Started");
        } else if (cmd === "X") {
          updateStatus("Manual Mode Stopped");
        }
      }
    })
    .catch((error) => console.error("Error:", error));
}

function updateStatus(message) {
  const statusDisplay = document.getElementById("status-display");
```

```
        if (statusDisplay) {
          statusDisplay.textContent = `Status: ${message}`;

          // Change color based on status
          if (
            message.includes("started") ||
            message.includes("Manual Mode Started")
          ) {
            statusDisplay.style.backgroundColor = "#dcfce7";
            statusDisplay.style.color = "#166534";
          } else if (message.includes("paused")) {
            statusDisplay.style.backgroundColor = "#fef9c3";
            statusDisplay.style.color = "#854d0e";
          } else if (
            message.includes("stopped") ||
            message.includes("Manual Mode Stopped")
          ) {
            statusDisplay.style.backgroundColor = "#fee2e2";
            statusDisplay.style.color = "#991b1b";
          } else {
            statusDisplay.style.backgroundColor = "#e2e8f0";
            statusDisplay.style.color = "#1e293b";
          }
        }
      }

      // Update sensors every 300ms
      setInterval(updateSensors, 300);

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