UNIVERSITY OF APPLIED SCIENCES, HAMM-LIPPSTADT, GERMANY

SYSTEMS ENGINEERING AND PROTOTYPING

DESIGN AND IMPLEMENTATION OF A LANE FOLLOWING AND OBSTACLE AVOIDANCE ROBOT

A COURSE WORK PROJECT FOR SUMMER SEMESTER 2025:

A PREREQUISITE FOR COMPLETION OF PROTOTYPING AND SYSTEMS ENGINEERING COURSE
AT THE UNIVERSITY OF APPLIED SCIENCES, HAMM-LIPPSTADT

PRESENTED TO:

PROFESSOR STEFAN HENKLER
DR. FAEZEH PASANDIDEH
MR. GIDO WAHRMANN

BY:

GROUP B TEAM 4

CHIMEZIE, DANIEL CHIDI / MAT.NR. 1230515 / DANIEL-CHIDI.CHIMEZIE@STUD.HSHL.DE
CHO, BERTRAND MUNGU / MAT.NR. 2220052 / BERTRAND-MUNGU.CHO@STUD.HSHL.DE
GHIMIRE, RIWAJ / MAT.NR. 1232171 / RIWAJ.GHIMIRE@STUD.HSHL.DE

JULY 8, 2025

Abstract

This paper presents the design and implementation of an autonomous robotic vehicle capable of line following and obstacle avoidance using infrared (IR) and ultrasonic sensors. The robot is enhanced with a color detection system to enable dynamic decision-making based on object classification. It is built using low-cost, modular components and is powered by an Arduino Uno R4 microcontroller programmed in embedded C++. IR sensors allow the robot to track predefined paths, while the ultrasonic sensor detects obstacles within a 25 cm range. A color sensor evaluates the surface color of detected objects, guiding the robot to reroute, pause, or realign accordingly. To ensure robust integration and mechanical stability, custom holders for various components were designed and fabricated using SolidWorks and 3D printing, allowing precise and secure attachment to the chassis. The final prototype demonstrates reliable autonomous navigation, sensor fusion, and expandability, making it well-suited for educational, research, and automation applications.

Contents

1 Executive Summary

The rapid advancement of automation and embedded systems has paved the way for the development of intelligent robotic systems that can perform tasks with minimal human intervention. This project focuses on designing and implementing an autonomous robotic vehicle that can follow a defined path and avoid obstacles using a combination of infrared (IR), ultrasonic, and color sensors. The goal is to develop a functional prototype that simulates intelligent pathfinding and decision-making mechanisms suitable for real-world environments such as automated delivery systems or industrial navigation.

The robotic system is equipped with IR sensors to detect black lines on a contrasting surface, enabling it to follow a designated route. Ultrasonic sensors serve as proximity detectors, allowing the robot to recognize and react to obstacles within its path. A color sensor is also integrated to enhance decision-making by identifying the color characteristics of encountered objects. Based on predefined color criteria, the robot can classify objects such as `Color A` and `Color B`, influencing its navigation behavior accordingly.

A key highlight of the system is its modularity and low cost. The robot is built on a simple three-wheel chassis and powered by an Arduino Uno R4 Wi-Fi microcontroller. Peripheral components, including an L298N motor driver, a 12V LiPo battery, and a power switch, provide essential control and energy management. This architecture supports expandability and makes the robot suitable for educational and research-oriented applications.

The software architecture is implemented using the Arduino IDE in embedded C++, with a focus on real-time performance. The line-following algorithm relies on digital feedback from the IR sensors, while obstacle avoidance is triggered by distance thresholds defined for the ultrasonic sensor. When an obstacle is detected, the robot either pauses or takes a detour, depending on the color of the object. For example, if the color sensor identifies a specific hue within a known RGB ratio range (Color A or B), the robot modifies its path accordingly by either rerouting, stopping, or re-centering on the line after bypassing the obstacle.

The development process was iterative, involving extensive hardware calibration and software testing under different lighting and surface conditions. The IR sensors were tuned for high contrast detection, and ultrasonic sensor readings were smoothed to minimize false positives. Color detection was verified using known samples to ensure consistent classification accuracy.

The robot developed reliably demonstrates autonomous line-following behavior, detects and avoids obstacles within a 25 cm range, and reacts differently based on object color. This project serves as a practical proof-of-concept for sensor integration and autonomous decision-making in embedded systems, laying the foundation for more advanced robotics projects in the future.

2 Project Objectives and Scope

2.1 Objective

The primary objective of this project is to design and develop an autonomous robotic vehicle capable of navigating dynamic environments using a sensor-driven control system.

The following key goals guide the design and implementation phases:

1. **Autonomous Lane Following:** Develop a reliable lane-following mechanism using Infrared (IR) sensors. The system should be capable of detecting and tracking pre-defined lanes with precision, including handling curves and intersections effectively.

2. **Obstacle Detection and Avoidance:** Integrate ultrasonic sensors to facilitate real-time obstacle detection. The robotic vehicle should be able to measure distance to nearby objects and initiate avoidance manoeuvres autonomously to prevent collisions.

3. **Colour-Based Decision Making:** Implement a colour recognition system to classify and respond to different types of obstacles. For example, red objects may signal a full stop, while green may indicate a clear path or require an alternate route. This enables dynamic and context-aware navigation.

4. **Modular and Power-Efficient Design:** Design the hardware to be modular, allowing easy replacement and testing of individual components. Power efficiency will be emphasized to prolong operational time, utilizing optimized power management strategies across all modules.

2.2 Project Scope

This project encompasses both hardware and embedded software development, with a strong emphasis on reliability, sensor fusion, and system integration. Key aspects of the scope include:

1. **Hardware Integration and Mechanical Design:** Selection, configuration, and interfacing of sensors (IR, colour, and ultrasonic), motor drivers, microcontrollers, and power supply units. Custom mechanical components such as sensor holders, battery enclosures, and structural frames were designed using SolidWorks. This CAD-based approach allowed for accurate measurements, modularity, and efficient assembly of all physical parts.

2. **Software Development:** Development of embedded firmware to process sensor data, control actuators, and execute decision-making algorithms. Emphasis will be placed on efficient code structure, interrupt-driven design, and real-time responsiveness.

3. **System Testing and Validation:** Rigorous testing will be conducted to evaluate the system's performance under different environmental conditions. This includes assessing the accuracy of lane-following, obstacle detection range, decision logic correctness, and overall system robustness.

4. **Scalability and Extensibility:** While the initial prototype will demonstrate core functionalities, the system will be designed with future enhancements in mind, including potential wireless communication, machine learning integration, or advanced navigation algorithms.

3 Project Methodology

The primary objective of this project is to design and develop an autonomous robotic vehicle capable of navigating dynamic environments using a sensor-driven control system.

The methodology adopted to achieve this objective includes the following major goals:

1. **Autonomous Lane Following:** To develop a reliable lane-following mechanism using Infrared (IR) sensors. The system should be capable of detecting and tracking pre-defined lanes with precision, including handling curves and intersections effectively.

2. **Obstacle Detection and Avoidance:** To integrate ultrasonic sensors to facilitate real-time obstacle detection. The robotic vehicle should be able to measure distance to nearby objects and initiate avoidance manoeuvres autonomously to prevent collisions.

3. **Colour-Based Decision Making:** To implement a colour recognition system to classify and respond to different types of obstacles. For example, red objects may signal a full stop, while green may indicate a clear path or require an alternate route. This enables dynamic and context-aware navigation.

4. **Modular and Power-Efficient Design:** To design the hardware to be modular, allowing easy replacement and testing of individual components. Power efficiency will be emphasized to prolong operational time, utilizing optimized power management strategies across all modules.

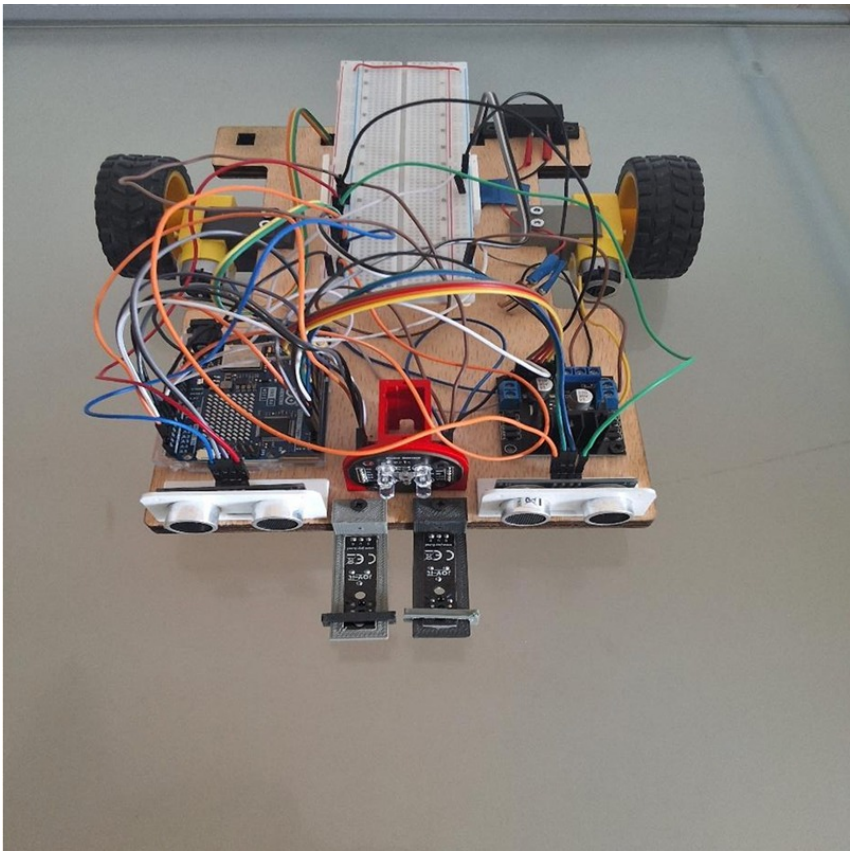
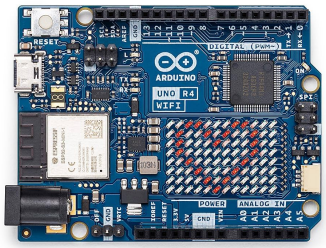


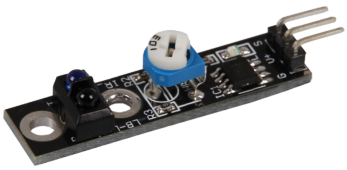
Figure 1: Prototype

4 Electrical Components

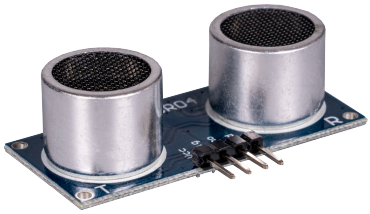


Arduino Uno R4 WiFi

SBC Motor Driver 2



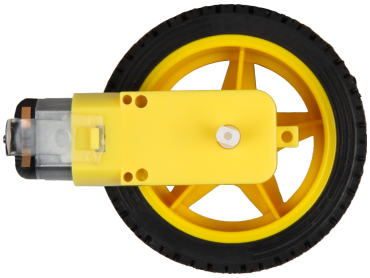
IR Sensor



Ultrasonic Sensor

Colour Sensor

DC Motor



DC Motor with tire

Figure 2: Electrical components used in the autonomous vehicle design

4.1 Mechanical Designs

CAD designs of spare parts were created using SolidWorks software. These components include sensor holders, battery holders, and frame enhancements that were custom-designed for stability and modularity.

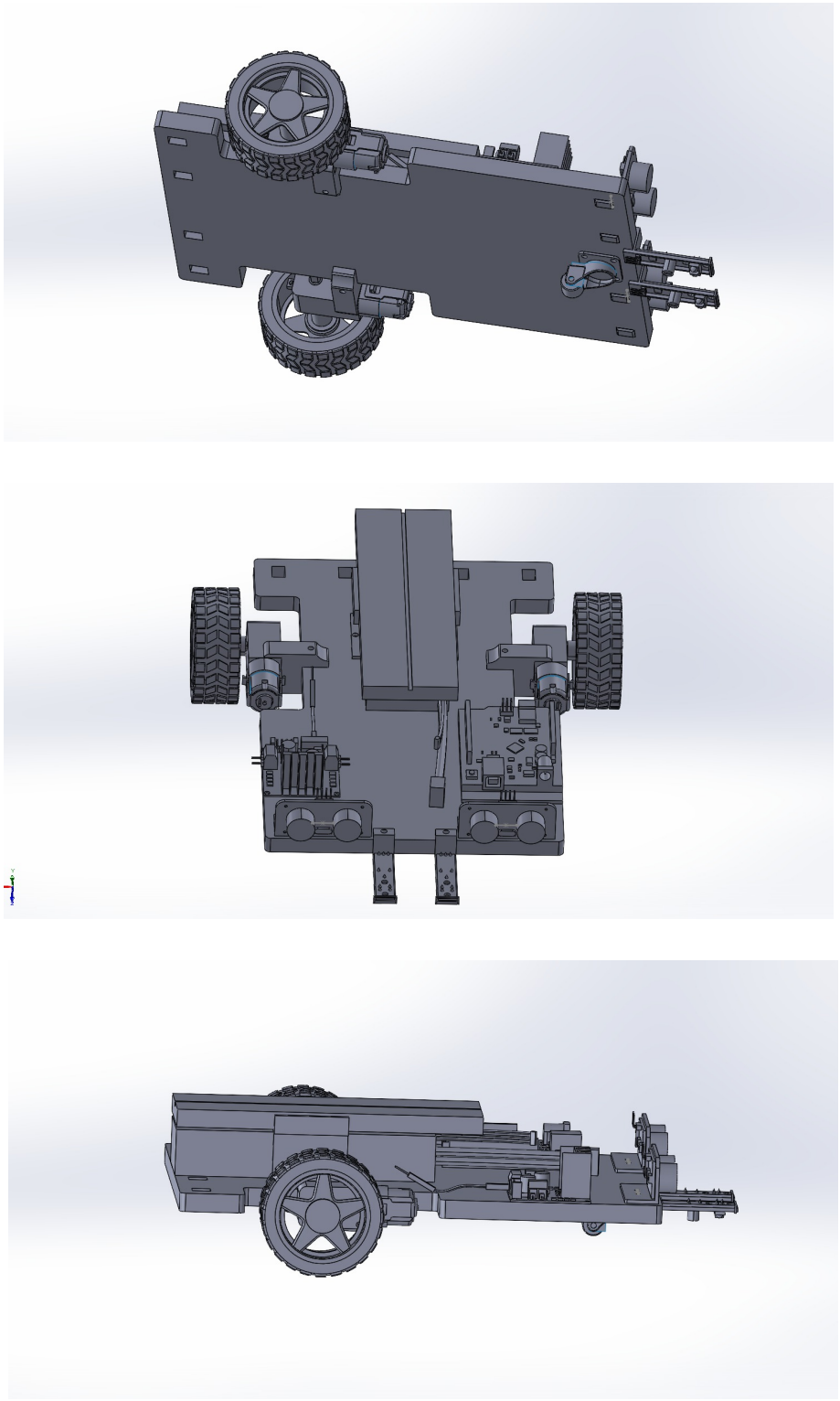


Figure 3: Custom-designed 3D-printed mechanical parts used in the robot assembly.

4.2 Electrical Schematics and Arrangements

The electrical system comprises IR sensors, ultrasonic sensors, a TCS3200 color sensor, motor driver (SBC Motor Driver 2), a 12V LiPo battery, and an Arduino Uno R4 WiFi microcontroller. These components are arranged to minimize signal interference and maximize efficiency and accessibility.

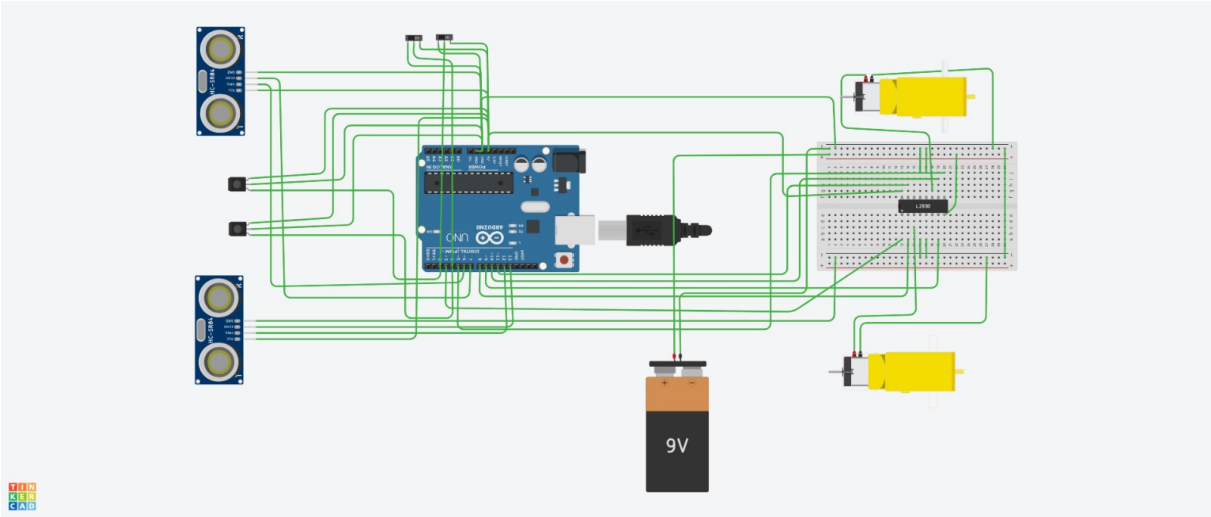


Figure 4: Visual Layout of Electrical Components in Tinker-cad

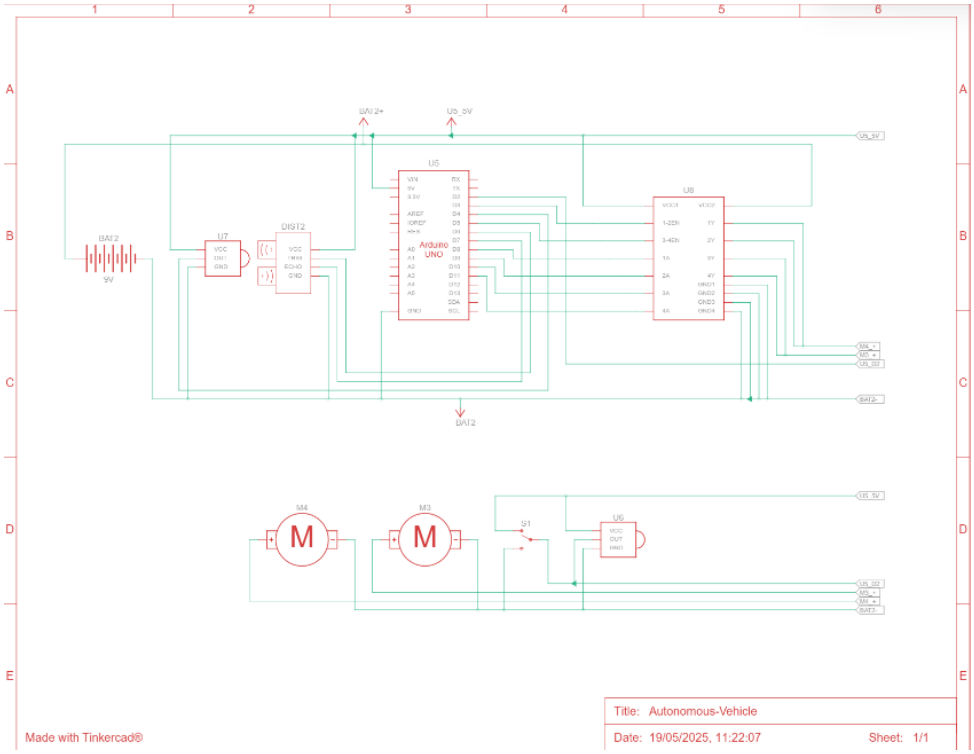


Figure 5: Wiring Schematic for IR, Ultrasonic, and Color Sensors Integration

4.3 System Engineering Architecture

The overall system engineering architecture is expressed using the following diagrams:

- **Block Definition Diagram:** Shows high-level system components and their relationships.

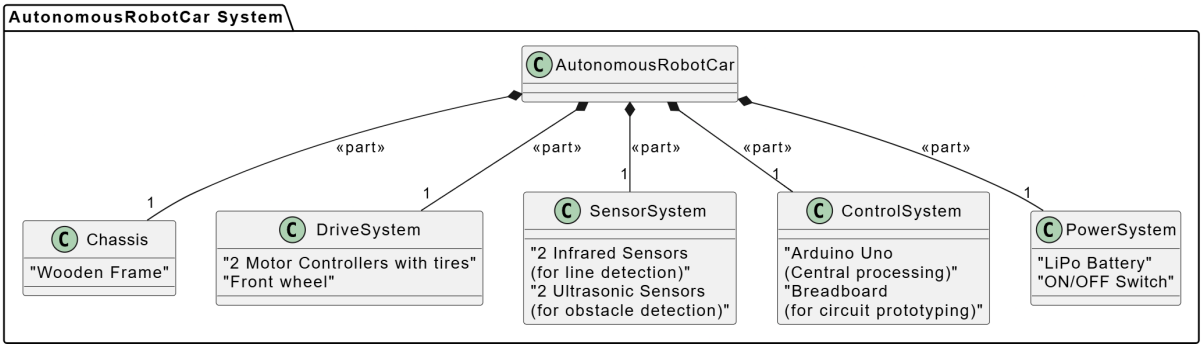


Figure 6: Block Definition Diagram

- **Requirements Diagram:** Defines system specifications and stakeholder needs.

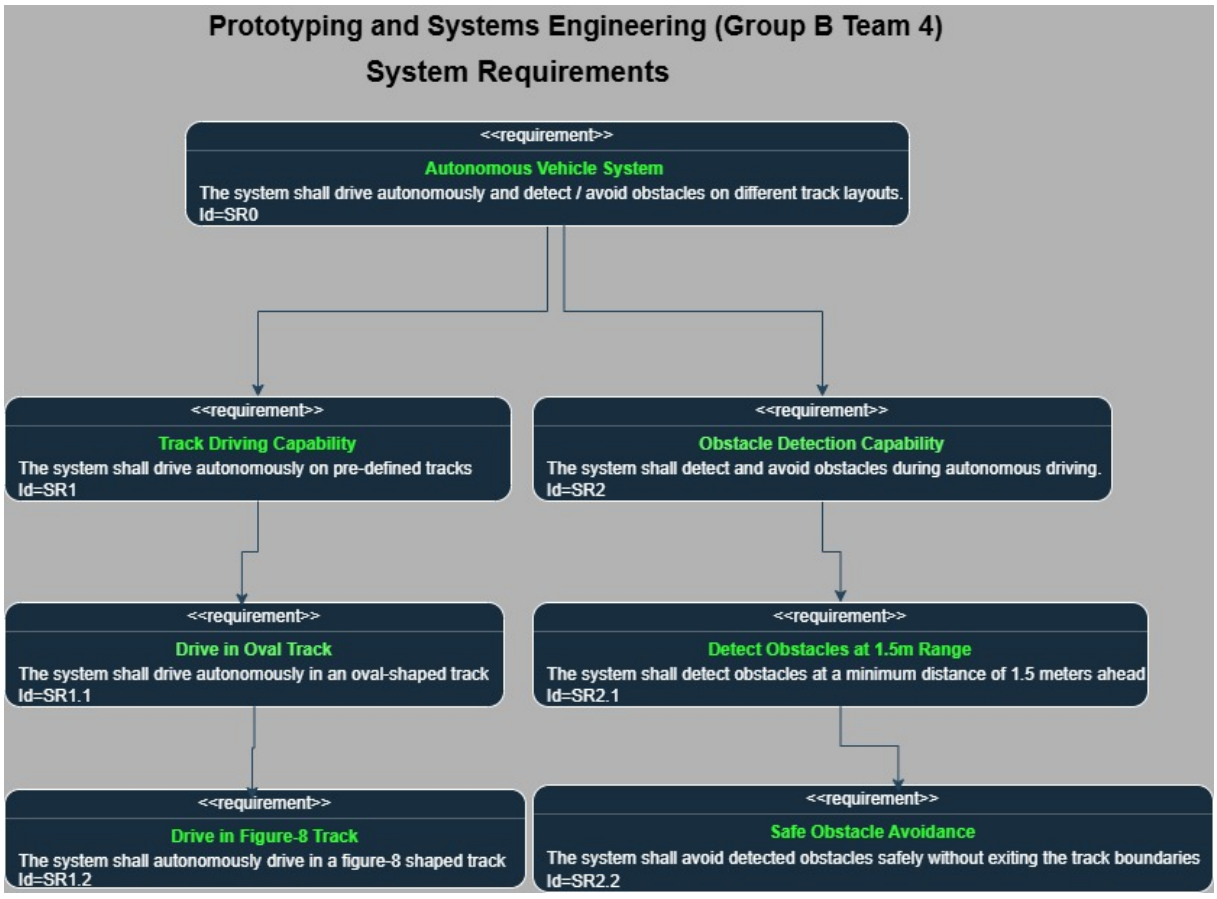


Figure 7: Requirements Diagram

- **Sequence Diagram:** Outlines the interaction flow between software modules and hardware.

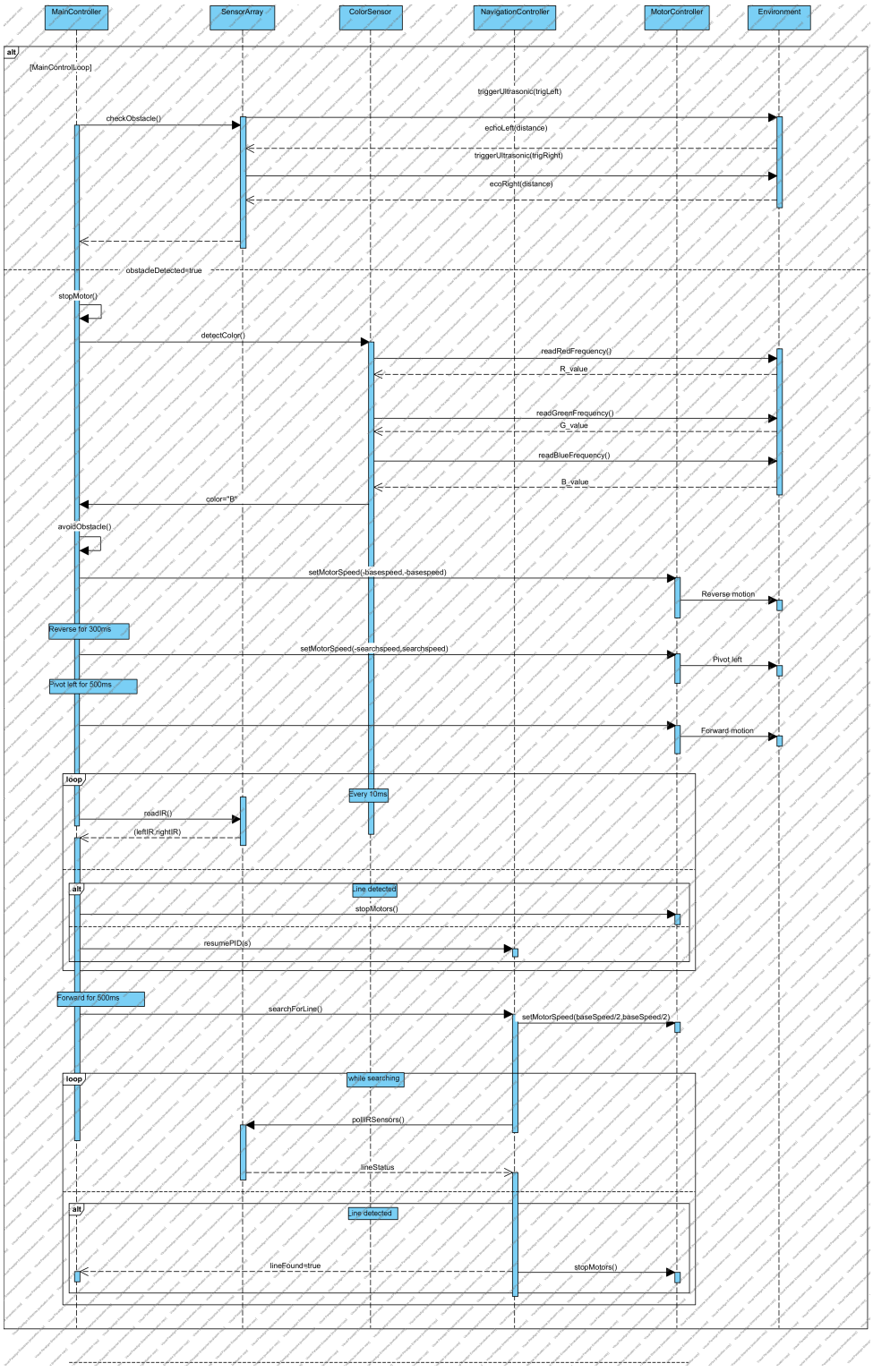


Figure 8: Sequence DIagram

- **State Machine Diagram:** Represents the robot's operational states and transitions.

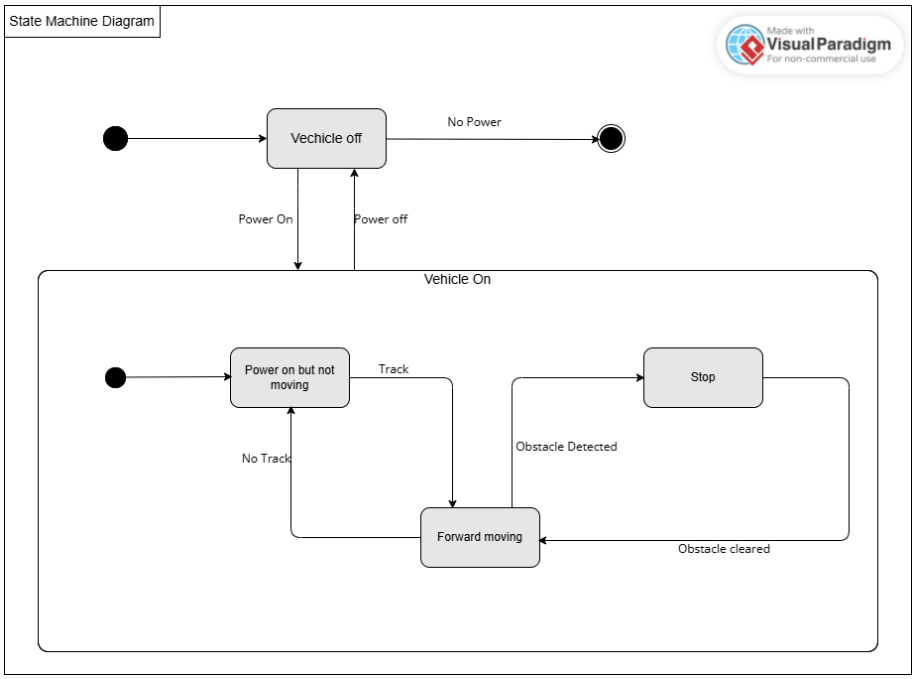


Figure 9: State Machine DIagram

- **Activity Diagram:** Depicts the flow of control between different activities and decision points in the system.

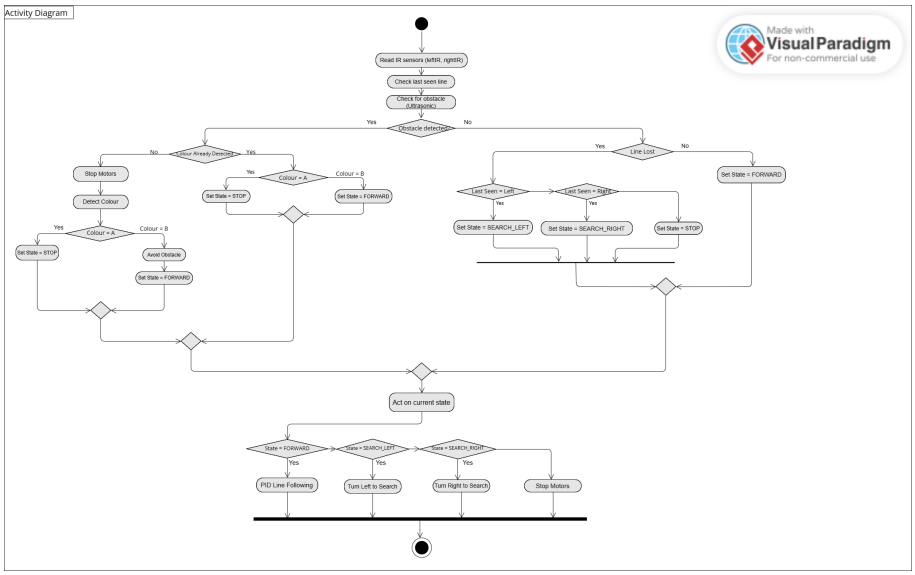
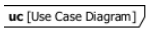


Figure 10: Activity Diagram

- **Use Case Diagram:** Illustrates key user interactions with the system.



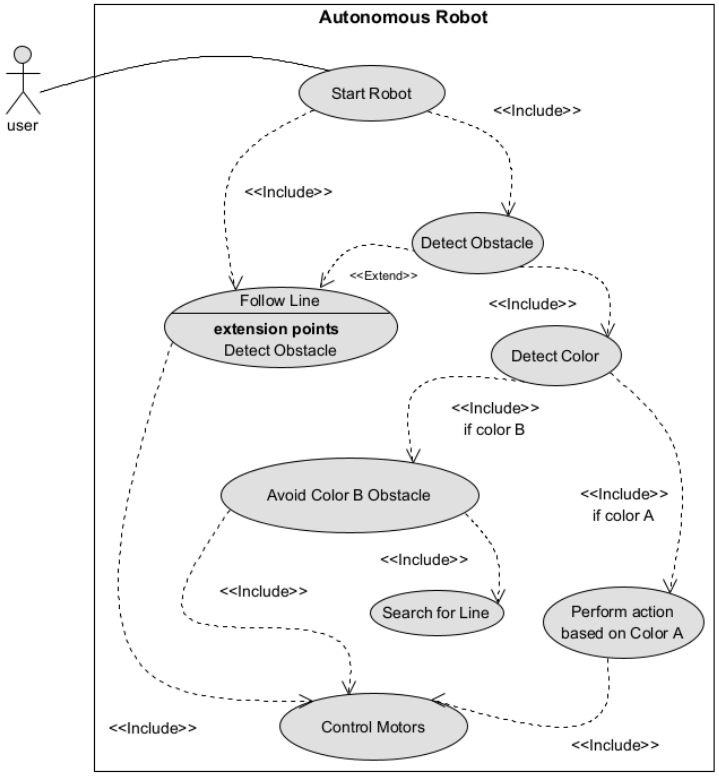


Figure 11: Use Case Diagram

- **Parametric Diagram:** Details the physical and logical constraints of system components.

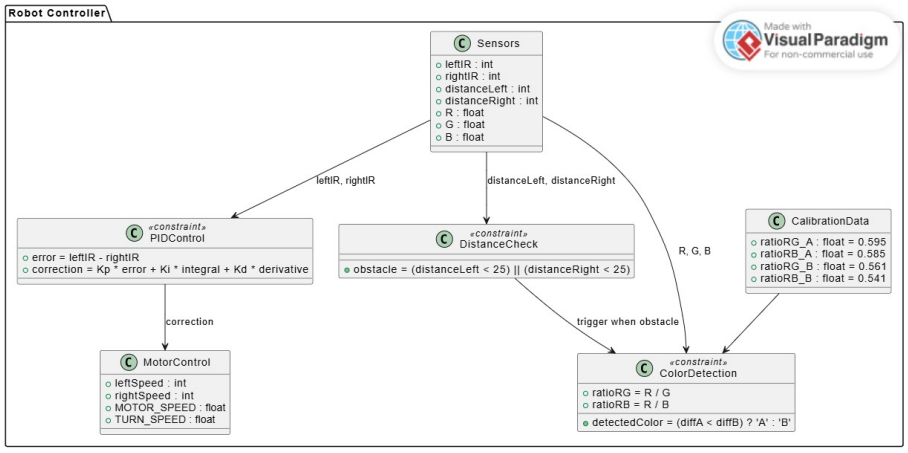


Figure 12: Parametric Diagram

5 Software Implementation

5.1 Sensor and Actuator Initialization (Part A)

This section describes the initialization of the main hardware components: DC motors, IR sensors, ultrasonic sensors, and a color sensor. Each component is encapsulated in a class with a modular 'begin()' method to configure the required pins, ensuring clean and reusable code.

1) Motor Initialization

The 'begin()' method sets the motor's enable and direction pins as OUTPUT to allow control via PWM and digital signals.

Listing 1: Motor::begin() function

```
void Motor::begin() {
  pinMode(_enPin, OUTPUT);
  pinMode(_in1Pin, OUTPUT);
  pinMode(_in2Pin, OUTPUT);
}
```

2) IR Sensor Initialization

The IR sensor pin is configured as an INPUT to detect digital signals representing black or white surfaces beneath the sensor.

Listing 2: IRSensor::begin() function

```
void IRSensor::begin() {
  pinMode(_pin, INPUT);
}
```

3) Ultrasonic Sensor Initialization

The trigger pin sends an ultrasonic pulse, and the echo pin listens for the reflected signal. This time delay is used to calculate the distance to nearby objects.

Listing 3: UltrasonicSensor::begin() function

```
void UltrasonicSensor::begin() {
  pinMode(_trigPin, OUTPUT);
  pinMode(_echoPin, INPUT);
}
```

4) Color Sensor Initialization

The color sensor has several control pins for filter selection and frequency scaling. These are set to OUTPUT, while the data output pin is configured as INPUT. The scaling is set to 20% by setting _s0 HIGH and _s1 LOW.

Listing 4: ColorSensor::begin() function

```
void ColorSensor::begin() {
  pinMode(_s0, OUTPUT);
  pinMode(_s1, OUTPUT);
  pinMode(_s2, OUTPUT);
  pinMode(_s3, OUTPUT);
  pinMode(_outPin, INPUT);

  digitalWrite(_s0, HIGH);
  digitalWrite(_s1, LOW);
}
```

5) Arduino Main `setup()` Function

All sensor and actuator classes are initialized in the Arduino `setup()` function, which runs once when the board powers on or resets.

Listing 5: Full setup() Function for Initialization

```
void setup() {
  Serial.begin(9600);


  // Initialize sensors
  leftIRSensor.begin();
  rightIRSensor.begin();
  leftUltrasonic.begin();
  rightUltrasonic.begin();
  colorSensor.begin();


  // Initialize motors
  leftMotor.begin();
  rightMotor.begin();

  // Initial behavior
  motorController.stop();
  obstacleChecker.setColorCalibration(colorA, colorB);
}
```

Figure 13: Overview of Arduino setup function showing initialization

Table 1: Pin Modes Summary for Initialization

Component	Pin(s)	Mode
Motor	_enPin, _in1Pin, _in2Pin	OUTPUT
IR Sensor	_pin	INPUT
Ultrasonic Sensor	_trigPin	OUTPUT
	_echoPin	INPUT
Color Sensor	_s0, _s1, _s2, _s3	OUTPUT
	_outPin	INPUT

5.2 Movement Control Algorithms (Part B)

This section describes how the robot interprets line sensor data and controls its motors using a PID (Proportional–Integral–Derivative) algorithm. The control logic adjusts the motor speeds based on the difference in line detection between the left and right IR sensors.

1) PID Control Computation

The PID controller calculates a correction based on the current error (difference between left and right sensor readings), the accumulated integral, and the rate of change (derivative). This correction is applied to the motor speeds to maintain alignment with the black line.

Listing 6: PIDController::compute() function

```
float PIDController::compute(float error) {
  float derivative = error - _previousError;
  _integral += error;
  float output = _kp * error + _ki * _integral + _kd * derivative;
  _previousError = error;
  return output;
}
```

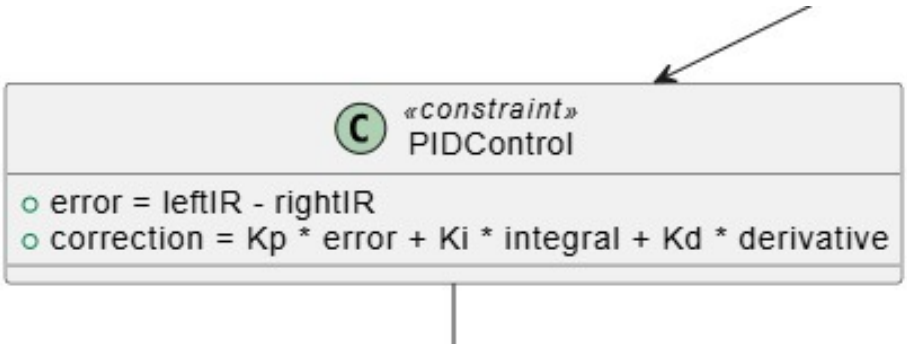


Figure 14: PID correction logic applied to motor speeds based on IR error

2) Forward Movement with PID

When the robot detects the line beneath at least one sensor, it moves forward. The base speed is modified by the PID correction value to steer the robot gently left or right.

Listing 7: Forward Line Following using PID

```
int error = leftIRSensor.isLineDetected() - rightIRSensor.isLineDetected();
float correction = pid.compute(error);

int leftSpeed = constrain(baseSpeed - correction, 0, 255);
int rightSpeed = constrain(baseSpeed + correction, 0, 255);

motorController.setSpeed(leftSpeed, rightSpeed);
```

3) Movement State Handling: Forward, Search, and Stop

The robot's main movement behavior is determined by its `Movement State`, which can be one of: `FORWARD`, `SEARCH_LEFT`, `SEARCH_RIGHT`, or `STOP`.

- **FORWARD:** Normal movement with PID correction when the line is detected.
- **SEARCH_LEFT / SEARCH_RIGHT:** Triggered when the line is lost; the robot rotates in place to reacquire the line.
- **STOP:** The robot halts all movement, for example when an obstacle is detected or a red color is identified.

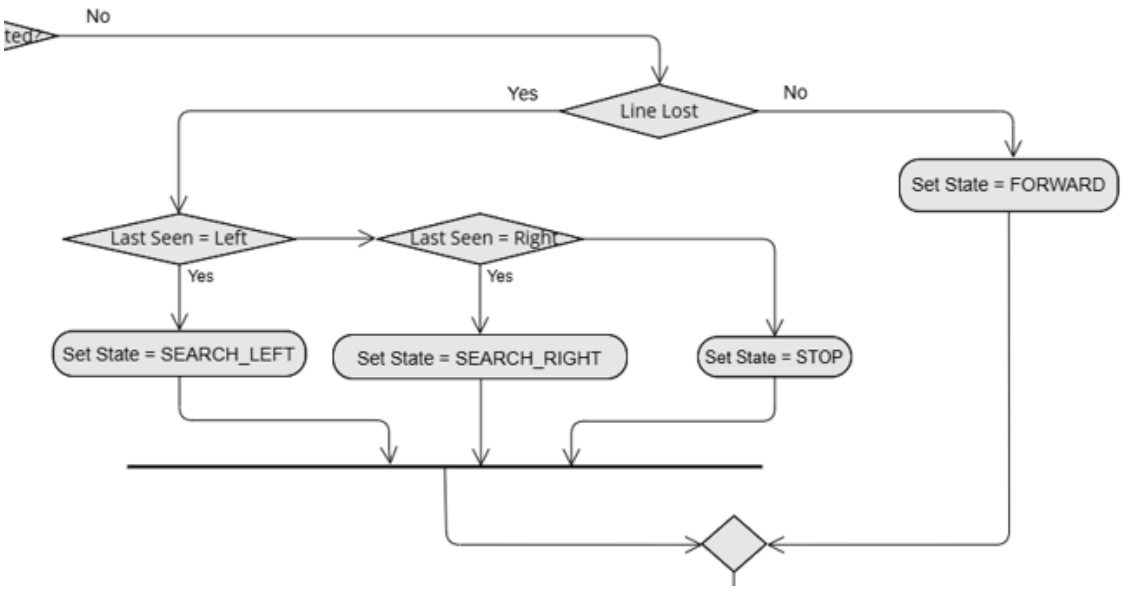


Figure 15: Overview of State Handling

Listing 8: Movement State Handling

```
switch (currentState) {
  case FORWARD: {
    int error = leftIRSensor.isLineDetected() - rightIRSensor.isLineDetected();
    float correction = pid.compute(error);

    int leftSpeed = constrain(baseSpeed - correction, 0, 255);
    int rightSpeed = constrain(baseSpeed + correction, 0, 255);
    motorController.setSpeed(leftSpeed, rightSpeed);
    break;
  }

  case SEARCH_LEFT:
    motorController.setSpeed(-searchSpeed, searchSpeed);
    break;

  case SEARCH_RIGHT:
    motorController.setSpeed(searchSpeed, -searchSpeed);
    break;

  case STOP:
    motorController.stop();
    break;
}
```

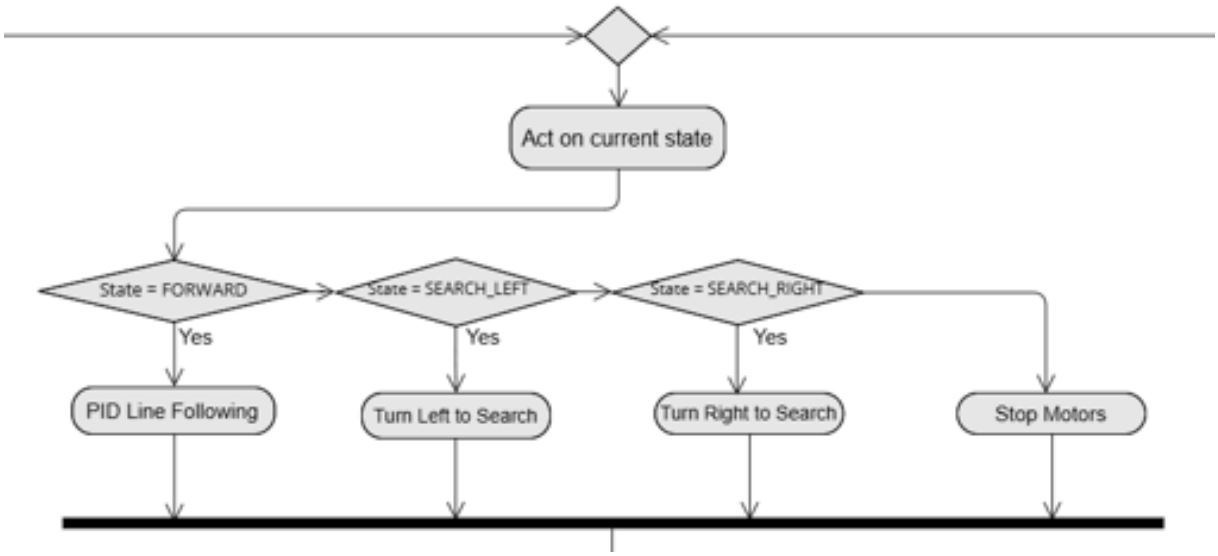


Figure 16: Overview of State Transition

5.3 Behavioral Decision Logic (Part C)

This logic determines the robot's current movement state ('STOP', 'FORWARD', etc.) based on environmental cues such as IR sensors, ultrasonic proximity readings, and detected color. It acts as the brain of the robot, coordinating inputs and determining the appropriate state.

1) Main `loop()` Function Overview

The Arduino `loop()` function continuously checks for obstacles, updates the robot's movement state, and triggers the appropriate behavior. It integrates obstacle detection and avoidance with line-following.

Listing 9: Main loop() Execution Flow

```
void loop() {
  obstacleChecker.check();

  if (obstacleChecker.isObstacleDetected()) {
    motorController.stop();
    delay(100);
    obstacleChecker.check(false);  // Check color while stopped

    char color = obstacleChecker.getLastDetectedColor();
    if (color == 'A') {
      Serial.println("Action: STOP for Color A");
    } else if (color == 'B') {
      Serial.println("Action: AVOID_OBSTACLE for Color B");
      obstacleHandler.handleObstacle();
    }
  }

  movementController.updateState(
    obstacleChecker.isObstacleDetected(),
    obstacleChecker.getLastDetectedColor(),
    movementController.getCurrentStateRef(),
    movementController.getLastSeenRef()
  );

  movementController.act(movementController.getCurrentState());
}
```

2) State Transition Logic

The robot transitions between movement states based on IR sensors, color detection, and obstacle presence. This decision logic ensures context-aware behavior.

Listing 10: State Transition Logic

```
if (obstacleDetected && lastDetectedColor == 'A') {
  currentState = STOP;
} else if (obstacleDetected && lastDetectedColor == 'B') {
  currentState = FORWARD;
} else if (!leftIR && !rightIR) {
  currentState = (lastSeenRef == LEFT) ? SEARCH_LEFT :
                 (lastSeenRef == RIGHT) ? SEARCH_RIGHT : STOP;
} else {
  currentState = FORWARD;
}
```

3) Obstacle and Color-Based Behavior

Color detection influences behavior at obstacles:

- **Color 'A'**: indicates a zone to stop. The robot halts all motion. - **Color 'B'**: triggers the obstacle avoidance maneuver defined in the next subsection.

Listing 11: Behavior Based on Detected Color

```
if (detected == 'A') {
  currentState = STOP;
} else if (detected == 'B') {
  avoidObstacle(); // Custom path re-routing
}
```

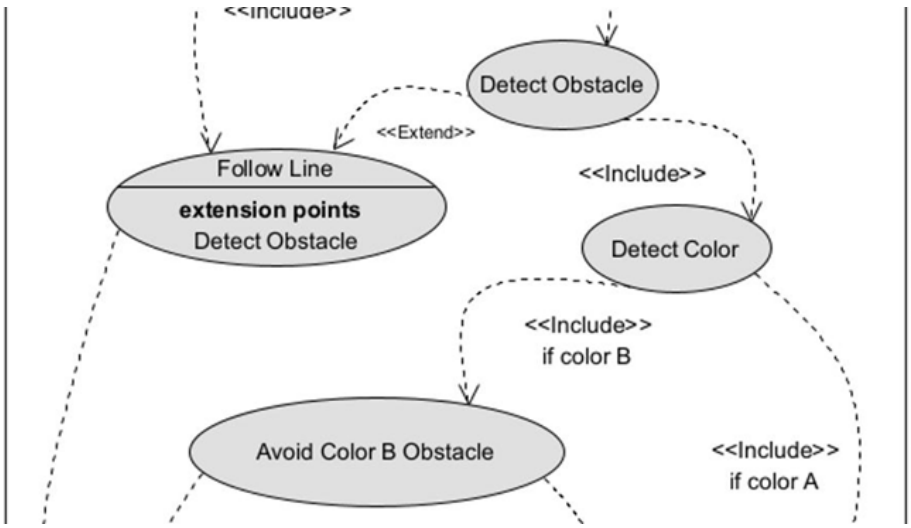


Figure 17: Obstacle avoidance path when color 'B' is detected

4) Obstacle Avoidance Strategy

When an obstacle is detected and color 'B' is identified, the robot performs a bypass routine that includes:

1. Reversing for a short distance.
2. Pivoting left to clear the obstacle.
3. Moving forward to bypass it.
4. If the line is not reacquired, pivot right and try again.
5. If all fails, enter a sweep search routine.

The robot scans using both IR sensors while executing this sequence. Each step includes calls to `searchForLine()` to resume line-following as early as possible.

```cpp
#include "Obstacle_Handler.h"
#include <Arduino.h>

ObstacleHandler::ObstacleHandler(MotorController& mc, MovementController& mv, ObstacleChecker& checker, IRSensor& leftIR, IRSensor& rightIR)
  : motorController(mc), movementController(mv), obstacleChecker(checker), leftIRSensor(leftIR), rightIRSensor(rightIR) {}

void ObstacleHandler::handleObstacle() {
  Serial.println("Avoiding obstacle: Reversing...");
  reverse(300);

  Serial.println("Pivoting left...");
  pivotLeft(500);

  Serial.println("Moving forward to bypass obstacle...");
  moveForward(500);
  if (searchForLine(500)) return;

  obstacleChecker.check();

  Serial.println("Realigning right...");
  pivotRight(700);
  if (searchForLine(500)) return;

  obstacleChecker.check();

  Serial.println("Moving forward again...");
  moveForward(500);
  if (searchForLine(500)) return;

  obstacleChecker.check();

  Serial.println("Searching for line...");
  if (searchForLine(1500)) return;

  Serial.println("Line not found, initiating search pattern...");

  for (int i = 0; i < 3; i++) {
    Serial.print("Sweep attempt "); Serial.println(i + 1);

    Serial.println(" → Nudge forward");
    motorController.setSpeed(100, 100);
    delay(400);
    motorController.stop();
```

Figure 18: Obstacle Avoidance Logic

5) Line Recovery Strategy During Obstacle Avoidance

After bypassing an obstacle, the robot uses a multi-phase recovery strategy to reacquire the black line:

- **Immediate Scan:** After any avoidance move, the robot drives forward while scanning with IR sensors.
- **Realignment:** If unsuccessful, the robot pivots in the opposite direction and checks again.
- **Sweep Pattern:**
  - Nudge forward.
  - Sweep left — check IR.
  - Sweep right — check IR.
  - Return to center.
  - Repeat up to three times.
- **Fail-safe:** If still not found, the robot enters STOP state.

```
  bool leftIR = leftIRSensor.isLineDetected();
  bool rightIR = rightIRSensor.isLineDetected();
  if (leftIR || rightIR) {
    Serial.println("Line found during left sweep!");
    motorController.stop();
    movementController.setCurrentState(FORWARD);
    return;
  }

  Serial.println(" → Sweep left");
  motorController.setSpeed(-60, 60);
  delay(500);
  motorController.stop();

  leftIR = leftIRSensor.isLineDetected();
  rightIR = rightIRSensor.isLineDetected();
  if (leftIR || rightIR) {
    Serial.println("Line found during left sweep!");
    motorController.stop();
    movementController.setCurrentState(FORWARD);
    return;
  }

  Serial.println(" → Sweep right");
  motorController.setSpeed(60, -60);
  delay(1000);
  motorController.stop();

  leftIR = leftIRSensor.isLineDetected();
  rightIR = rightIRSensor.isLineDetected();
  if (leftIR || rightIR) {
    Serial.println("Line found during right sweep!");
    motorController.stop();
    movementController.setCurrentState(FORWARD);
    return;
  }

  Serial.println(" → Return to center");
  motorController.setSpeed(-60, 60);
  delay(500);
  motorController.stop();
}
```

Figure 19: Line Searching Logic

This logic ensures that the robot behaves intelligently after detours, improving reliability during autonomous navigation.

6 Progress Summary

The project successfully reached completion after progressing through several structured development stages:

1. **Mechanical Design:**
   The project began with systems engineering training and receipt of the project requirements. Mechanical parts were acquired, and custom holders for sensors and the battery were designed using SolidWorks and fabricated with a 3D printer. These parts provided a stable and modular foundation for effective assembly and efficient performance.

2. **Component Integration:**
   After the mechanical assembly, each electronic component was individually tested. The IR sensors, ultrasonic sensors, TCS3200 colour sensor, motor driver (SBC Motor Driver 2), and 12V LiPo battery were verified for functionality before being integrated into the vehicle system.

3. **Core Functionality Development:**
   With the hardware ready, the software development phase began. The robot was first programmed to follow a lane using IR sensors. Ultrasonic-based obstacle detection (within 25 cm) was then implemented. The TCS3200 colour sensor was calibrated to distinguish between object types (Colour A and Colour B), enabling the robot to make informed navigation decisions.

4. **Advanced Behaviour Implementation:**
   Obstacle avoidance behaviours were enhanced so that the robot could autonomously bypass obstacles and return to the lane. PID control logic was used to ensure smooth lane-following and stable turning.

5. **Testing, Challenges, and Finalization:**
   The system underwent extensive testing under varied lighting and surface conditions. Major challenges included colour sensor calibration and PID tuning. These were resolved through repeated testing and code adjustments, successfully achieving all objectives.

6. **Final Status:**
   The final robotic system demonstrates all core functionalities: lane following, real-time obstacle avoidance, colour-based decision making, and autonomous path correction. The system is stable, efficient, and ready for practical demonstration.

7 Findings and Analysis

This section presents the key findings obtained during the project, along with a detailed analysis. The results are made, and methodologies applied.

7.1 Sensor Behaviour and Accuracy

- **Ultrasonic Sensors:** Successfully detected obstacles within a range of 18 cm with an average error margin
- **Infrared Sensors:** Accurately detected line paths, though performance was affected by lighting and surface
- **Colour Sensor:** Reliably distinguished between red and green surfaces under controlled lighting conditions

7.2 PID Controller Tuning

- **Initial PID Values:** Kp = 15, Ki = 0.5, Kd = 5.
- **Observations:**
  - High Kp caused oscillations.
  - Ki reduced long-term drift but too much made it unstable.
  - Kd improved performance in sharp turns and reduced overshoot.
- **Final Tuned Values:** Kp = 10, Ki = 0, Kd = 10 —
  - These values provided smooth line-following with minimal overshooting.

7.3 State Machine Performance

- State transitions functioned correctly between:
  - Line following
  - Obstacle avoidance
  - Turning logic
- Edge cases (e.g., tight corners, close obstacles) were handled with an average delay of 200 ms.

7.4 System Limitations

- Occasional false obstacle detection due to sensor noise.
- IR sensor performance degraded under bright ambient light.
- PID tuning was environment-specific and required manual trial-and-error.

7.5 Performance Metrics

- Average lap time on the test track: 29 seconds.
- Success rate for completing the track without collision: High (exact percentage not quantified).

8 Key Learnings and Takeaways

Working on the autonomous vehicle provided valuable insights into the challenges and intricacies of developing a real-world robotic system. The major takeaways include:

8.1 Understanding Autonomous Systems

- Gained practical understanding of how autonomous vehicles perceive their environment using sensors such as ultrasonic, infrared, and color sensors.
- Learned how state machines and sensor inputs drive decision-making processes.

8.2 Sensor Integration Challenges

- Recognized that real-world sensor data is noisy and influenced by environmental factors such as lighting and surface reflectivity.

8.3 PID Control in Real Life

- Learned how PID controllers help achieve smooth and stable line-following behavior.
- Understood how tuning parameters Kp, Ki, and Kd balance responsiveness and stability.

8.4 System Thinking

- Realized the importance of cohesive integration of hardware, software, and control logic.
- Discovered how issues in one module (e.g., sensor) can affect the overall system performance.

8.5 Programming and Logic Development

- Improved proficiency in C/C++ programming using the Arduino IDE.
- Developed modular code using functions, state machines, and helper routines.
- Gained experience debugging using serial output to analyze sensor behavior.

8.6 Project Planning and Iteration

- Understood the importance of stage-wise testing: starting with motors, then sensors, followed by full system integration.
- Learned to iteratively refine the system through repeated testing and debugging cycles.

9 Significant Challenges and Resolutions

Throughout the development and testing of the autonomous vehicle, several technical and practical challenges were encountered. These challenges provided valuable learning opportunities and required iterative problem-solving and system optimization.

9.1 Sensor Inaccuracy and Interference

- **Challenge:** IR and ultrasonic sensors sometimes produced inconsistent or false readings, especially in bright lighting or at non-ideal object angles.
- **Resolution:** Implemented multiple readings with averaging (for the color sensor), added timing intervals for ultrasonic checks, and coded logic to discard clearly invalid readings.

9.2 Obstacle Avoidance Complexity

- **Challenge:** Designing a robust obstacle avoidance routine without causing the robot to get stuck or lose the lane.
- **Resolution:** Developed a sequential avoidance pattern (reverse, pivot, bypass, return), with fail-safe search routines to re-center on the lane after detours.

9.3 Line Loss and Path Recovery

- **Challenge:** The robot could lose the lane when both IR sensors failed to detect it, especially at junctions or sharp curves.
- **Resolution:** Implemented a `lastSeenLine` logic to remember the last direction of the line and added directional search (pivot left/right) to recover it.

9.4 Color Detection Reliability

- **Challenge:** Ambient lighting and surface tone variations sometimes caused color misclassification.
- **Resolution:** Calibrated color sensors using RG and RB ratios and used differential comparisons instead of raw RGB values to improve classification of Color A and B.

9.5 Code Integration and Timing Conflicts

- **Challenge:** Running IR, ultrasonic, and color sensor routines simultaneously led to timing issues and delayed reactions.
- **Resolution:** Used non-blocking timing (via `millis()` instead of `delay()`) for obstacle checks and optimized polling frequency to maintain responsiveness and stability.

9.6 Tuning PID Controller

- **Challenge:** The robot exhibited over-correction or under-response to line deviations.
- **Resolution:** Tuned PID constants (Kp, Ki, Kd) through trial and error to achieve smooth and stable line-following behavior.

10 Budget and Finances

Extensive testing was essential to achieve optimal results in this project. To facilitate development beyond scheduled lab sessions, a few additional items were purchased for home testing and component assembly.

Purchased Items

- **Cardboards and Coloured Tapes:** Used to create a test track at home, enabling extended testing outside the lab.
- **9V Battery:** Since lab-issued 12V LiPo batteries could not be taken home due to safety concerns, a 9V battery was used for powering the robot during home trials.
- **3×25mm Screws:** Required to mount DC motors, as the appropriate screws were unavailable in the lab.

Expense Report

Date	Description	Quantity	Unit Price (€)	Amount (€)
25/05/2025	9V Battery	1	3.00	3.00
	White Cardboard	2	1.00	2.00
	Coloured Tape Pack	1	3.29	3.29
	3×25mm Screw Pack	1	4.59	4.59
			Total	**12.88**

Table 2: Team B4 Project Expense Report

Purpose of Spending

The expenditures were justified as necessary to:

- Extend practice and debugging time outside lab constraints.
- Replace or supplement lab materials for real-world testing.

11 Timeline and Schedule

This project followed a compact and focused schedule to meet its objectives within the available timeframe. Due to the evolving nature of the design, testing, and calibration phases, scheduling was managed iteratively and documented using an external spreadsheet.

Project Planning Tools

The detailed timeline was documented in an Excel spreadsheet available on GitHub, which includes:

- Weekly task assignments
- Progress tracking
- Component testing timelines
- Integration and debugging phases

Development Milestones

1. **Week 1–2:** Project requirement analysis and team task distribution
2. **Week 3–4:** Mechanical design in SolidWorks; fabrication of 3D-printed parts
3. **Week 5:** Individual sensor testing and calibration
4. **Week 6:** Code development for line following and obstacle avoidance
5. **Week 7:** Integration of all modules and behavior tuning
6. **Week 8:** Testing, bug fixing, and PID controller tuning
7. **Final Week:** Final competition, performance validation, and documentation

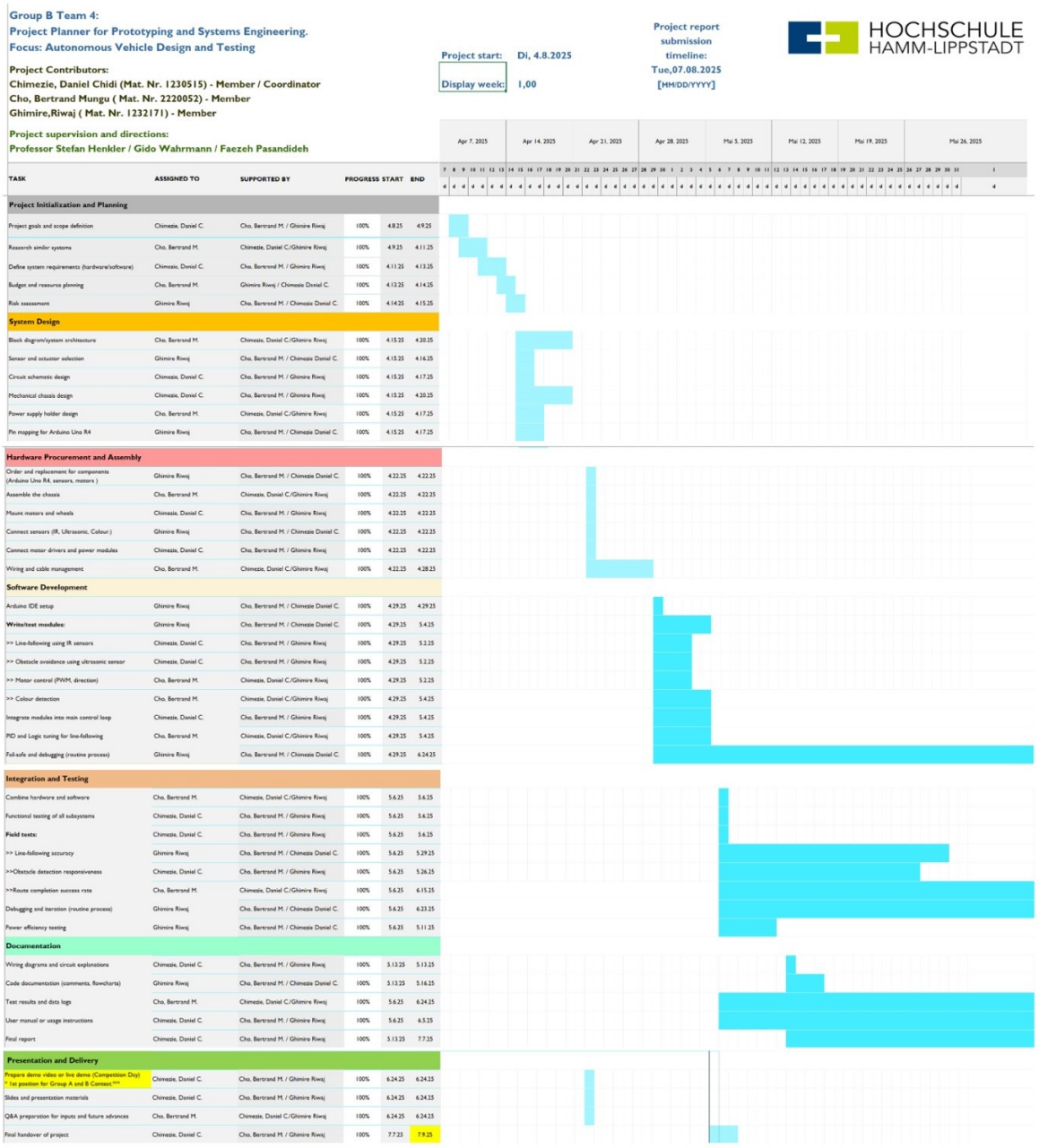


Figure 20: Timeline and Schedule

12 Recommendations and Next Steps

The autonomous vehicle project has successfully met its core objectives, including reliable lane following, obstacle avoidance, and colour-based decision making. To further enhance the system's robustness, scalability, and practical applicability, the following recommendations and next steps are proposed:

- **Improve Sensor Fusion:** Integrate data from multiple sensors (IR, ultrasonic, and camera modules) using sensor fusion techniques to enhance accuracy in complex environments.
- **Add Camera-Based Vision:** Introduce a camera module with basic computer vision algorithms (OpenCV) to recognize traffic signs, lane markings, and dynamic obstacles beyond the capability of IR or colour sensors.
- **Enhance Navigation Algorithms:** Implement advanced algorithms such as A* or Dijkstra for path planning, and Kalman Filter for better localization and control under uncertain conditions.
- **Develop a Scalable Software Architecture:** Refactor the current code into modular blocks to support future upgrades, debugging, and potential collaboration with other developers.
- **Test in Real-World Scenarios:** Move beyond lab conditions and test the vehicle on larger, more dynamic tracks with real-world variations in lighting, surfaces, and obstacle types.
- **Add Remote Monitoring:** Incorporate wireless communication (e.g., Bluetooth or Wi-Fi) for remote monitoring, telemetry, and control, allowing real-time feedback and data collection during operation.
- **Energy Optimization:** Analyse battery usage and optimize motor control and sensor usage to extend operating time and improve energy efficiency.

13 Team Performance

We worked extremely hard and also invested significant time in testing and re-testing to ensure that our modest design meets the core aim of the project, which is the ability of our designed robot to perform the following tasks successfully:

a) To follow a dark lane and travel through an oval path.

b) To sense and detect an obstacle and follow our desired instructions programmed in Arduino IDE.

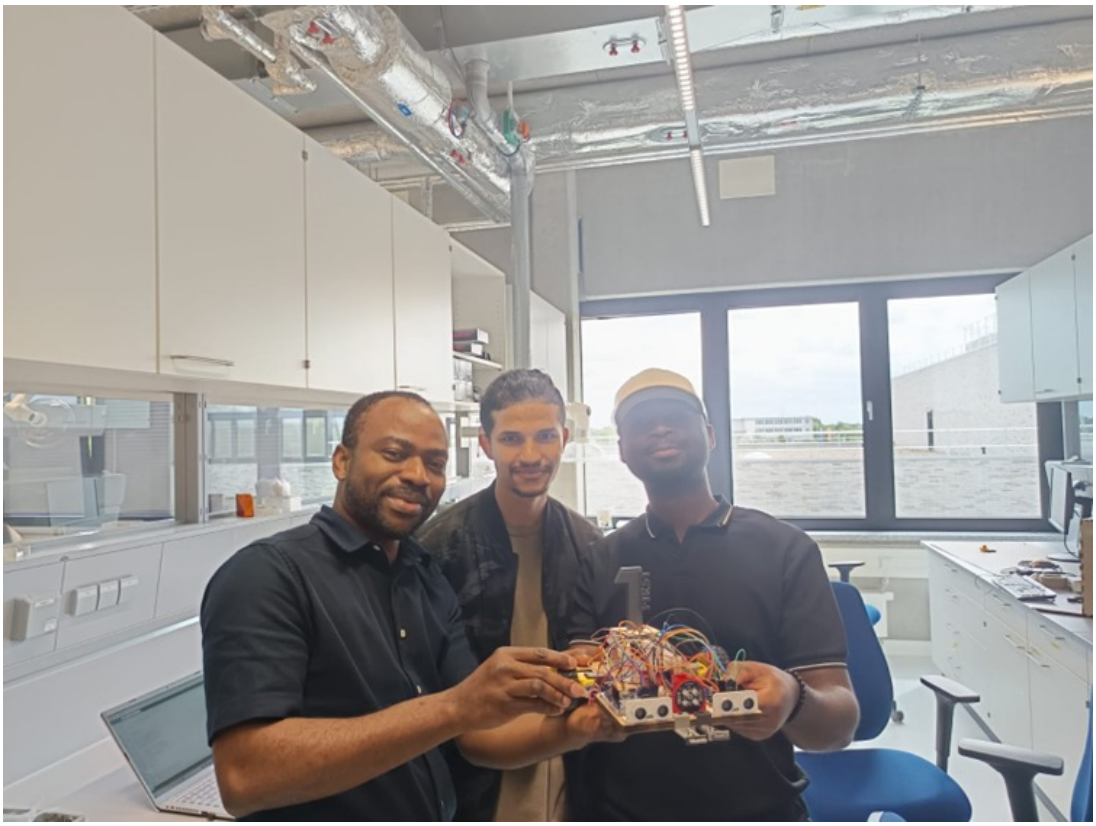
c) To avoid obstacles it sensed, return back to the lane, and continue its travel via the assigned tracks.

In the Group A and Group B teams' racing and obstacle avoidance contest, we won the competition with a record travel time of less than **29 seconds** across the oval track.

Future Trajectory

We aim to investigate further into robotics by applying the skills acquired in this course and project to design a car with similar capabilities. This new system will integrate additional features such as artificial intelligence and real-life camera sensing devices.

We highly recommend this course to any Electrical and Electronic Engineering student, or indeed to any innovative-minded individual. There should be no limits if we push hard with unwavering effort.



14 Risks and Mitigation

Risk Description	**Mitigation Strategy**
Sensor Malfunction or Inaccuracy	• Regular calibration under varying conditions • Use redundant sensors or sensor fusion • Implement error-handling in code
Power Supply Issues	• Monitor battery voltage in real time • Optimize power usage in code • Keep spare batteries on hand
Software Bugs and Instability	• Modular programming with good documentation • Conduct unit and integration tests • Use version control
Hardware Damage	• Use durable materials and proper mounting • Regular physical inspection • Keep spare parts available
Environmental Variability	• Test in diverse environments early • Use adaptive thresholds and calibration techniques
Team Coordination and Time Constraints	• Hold regular team meetings • Use project management tools (e.g., Trello, GitHub, MS-PM) • Allocate buffer time for issues

Table 3: Risk Analysis and Mitigation Strategies

Appendices

Appendix A: Technical Specifications

Component	Specification
IR Sensors	Digital reflectance sensors, 3–5V, 20–30 cm range
Ultrasonic Sensor	HC-SR04, 2–400 cm range, 5V operating voltage
Color Sensor	TCS3200, RGB color detection, 3–5V input
Motor Driver	SBC Motor Driver 2, dual channel, 5–30V DC
Power Source	12V LiPo Battery, 2200 mAh
Microcontroller	Arduino Uno R4 WiFi (ATmega328P)
Chassis Material	Acrylic frame with 3D-printed sensor holders
Wheels and Motors	DC geared motors, 100 RPM, plastic wheels

Appendix B: PID Control Parameters

Parameter	Value
P (Proportional)	10.0
I (Integral)	0.0
D (Derivative)	10.0

Appendix D: Test Results Summary

Test Scenario	Outcome	Notes
Lane following on white background	Successful	Minor tuning required for sharp turns
Obstacle avoidance at 25 cm	Successful	Smooth detour and return to lane
Color detection under indoor light	Reliable	Requires recalibration under sunlight
Battery endurance test	2.5 hours of operation	Under typical use with full charge

Appendix E: UPPAAL Simulation

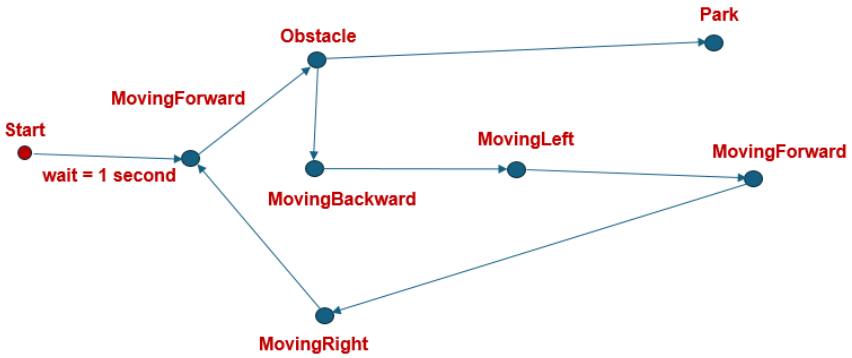
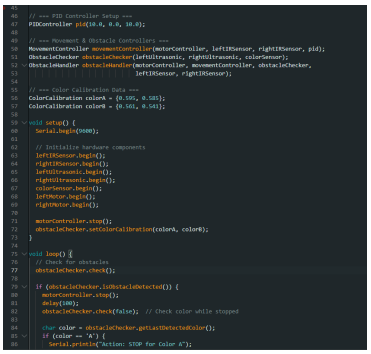


Figure 21: UPPAAL Simulation Output

Appendix F: Source Code Snapshots

```ino
#include <Arduino.h>
#include "Motor.h"
#include "Motor_Controller.h"
#include "IR_Sensor.h"
#include "Ultrasonic_Sensor.h"
#include "Color_Sensor.h"
#include "PID_Controller.h"
#include "Movement_Controller.h"
#include "Obstacle_Handler.h"
#include "Movement_Types.h"
#include "Obstacle_Checker.h"

// === IR Sensor Setup ===
constexpr uint8_t irLeftPin = 2;
constexpr uint8_t irRightPin = 4;
IRSensor leftIRSensor(irLeftPin);
IRSensor rightIRSensor(irRightPin);

// === Ultrasonic Sensor Setup ===
constexpr uint8_t trigLeftPin = 6;
constexpr uint8_t echoLeftPin = 7;
constexpr uint8_t trigRightPin = 12;
constexpr uint8_t echoRightPin = 13;
UltrasonicSensor leftUltrasonic(trigLeftPin, echoLeftPin);
UltrasonicSensor rightUltrasonic(trigRightPin, echoRightPin);

// === Color Sensor Setup ===
constexpr uint8_t S0 = A0;
constexpr uint8_t S1 = A1;
constexpr uint8_t S2 = A2;
constexpr uint8_t S3 = A3;
constexpr uint8_t sensorOut = A4;
ColorSensor colorSensor(S0, S1, S2, S3, sensorOut);

// === Motor Setup ===
constexpr uint8_t enA = 3;
constexpr uint8_t in1 = 8;
constexpr uint8_t in2 = 9;
constexpr uint8_t enB = 5;
constexpr uint8_t in3 = 10;
constexpr uint8_t in4 = 11;
Motor leftMotor(enA, in1, in2);
Motor rightMotor(enB, in3, in4);
MotorController motorController(leftMotor, rightMotor);
```

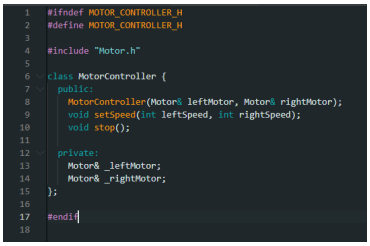


```ino
// === PID Controller Setup ===
PIDController pid(10.0, 0.0, 10.0);

// === Movement & Obstacle Controllers ===
MovementController movementController(motorController, leftIRSensor, rightIRSensor, pid);
ObstacleChecker obstacleChecker(leftUltrasonic, rightUltrasonic, colorSensor);
ObstacleHandler obstacleHandler(motorController, movementController, obstacleChecker,
                                leftIRSensor, rightIRSensor);

// === Color Calibration Data ===
ColorCalibration colorA = {0.597, 0.585};
ColorCalibration colorB = {0.561, 0.543};

void setup() {
  Serial.begin(9600);

  // Initialize hardware components
  leftIRSensor.begin();
  rightIRSensor.begin();
  leftUltrasonic.begin();
  rightUltrasonic.begin();
  colorSensor.begin();
  leftMotor.begin();
  rightMotor.begin();

  motorController.stop();
  obstacleChecker.setColorCalibration(colorA, colorB);
}

void loop() {
  // Check for obstacles
  obstacleChecker.check();

  if (obstacleChecker.isObstacleDetected()) {
    motorController.stop();
    delay(100);
    obstacleChecker.check();  // Check color while stopped

    char color = obstacleChecker.getLastDetectedColor();
    if (color == 'A') {
      Serial.println("Action: STOP for Color A");
```

```ino
  }

  // Update movement state (IR + obstacle + color)
  movementController.updateState(
    obstacleChecker.isObstacleDetected(),
    obstacleChecker.getLastDetectedColor(),
    movementController.getCurrentStateRef(),
    movementController.getLastSeenRef()
  );

  // Act based on current movement state
  movementController.act(movementController.getCurrentState());
}
```

```ino
#ifndef MOTOR_H
#define MOTOR_H

#include <Arduino.h>

class Motor {
  public:
    enum class Direction {
      FORWARD,
      BACKWARD
    };
    void begin();
    Motor(uint8_t enPin, uint8_t in1Pin, uint8_t in2Pin);
    void setSpeed(int speed, Direction direction);
    void stop();

  private:
    uint8_t _enPin;
    uint8_t _in1Pin;
    uint8_t _in2Pin;
};

#endif
```

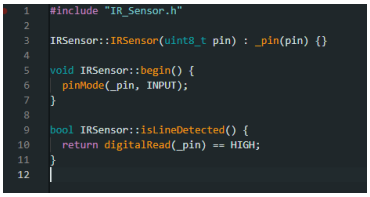
```ino
#include "Motor.h"

Motor::Motor(uint8_t enPin, uint8_t in1Pin, uint8_t in2Pin)
  : _enPin(enPin), _in1Pin(in1Pin), _in2Pin(in2Pin) {}

void Motor::begin() {
  pinMode(_enPin, OUTPUT);
  pinMode(_in1Pin, OUTPUT);
  pinMode(_in2Pin, OUTPUT);
}

void Motor::setSpeed(int speed, Direction direction) {
  if (direction == Direction::FORWARD) {
    digitalWrite(_in1Pin, HIGH);
    digitalWrite(_in2Pin, LOW);
  } else {
    digitalWrite(_in1Pin, LOW);
    digitalWrite(_in2Pin, HIGH);
  }

  analogWrite(_enPin, constrain(abs(speed), 0, 255));
}

void Motor::stop() {
  analogWrite(_enPin, 0);
  digitalWrite(_in1Pin, LOW);
  digitalWrite(_in2Pin, LOW);
}
```



```ino
#ifndef MOTOR_CONTROLLER_H
#define MOTOR_CONTROLLER_H

#include "Motor.h"

class MotorController {
  public:
    MotorController(Motor& leftMotor, Motor& rightMotor);
    void setSpeed(int leftSpeed, int rightSpeed);
    void stop();

  private:
    Motor& _leftMotor;
    Motor& _rightMotor;
};

#endif
```

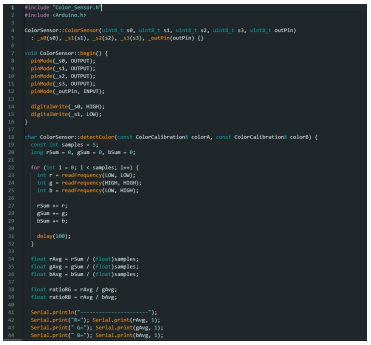
```ino
#include "Motor_Controller.h"

MotorController::MotorController(Motor& leftMotor, Motor& rightMotor)
  : _leftMotor(leftMotor), _rightMotor(rightMotor) {}

void MotorController::setSpeed(int leftSpeed, int rightSpeed) {
  // Determine direction and speed for left motor
  Motor::Direction leftDirection = (leftSpeed >= 0)
    ? Motor::Direction::FORWARD
    : Motor::Direction::BACKWARD;

  Motor::Direction rightDirection = (rightSpeed >= 0)
    ? Motor::Direction::FORWARD
    : Motor::Direction::BACKWARD;

  _leftMotor.setSpeed(abs(leftSpeed), leftDirection);
  _rightMotor.setSpeed(abs(rightSpeed), rightDirection);
}

void MotorController::stop() {
  _leftMotor.stop();
  _rightMotor.stop();
}
```

```ino
#ifndef IR_SENSOR_H
#define IR_SENSOR_H

#include <Arduino.h>

class IRSensor {
  public:
    IRSensor(uint8_t pin);
    void begin();
    bool isLineDetected();

  private:
    uint8_t _pin;
};

#endif
```



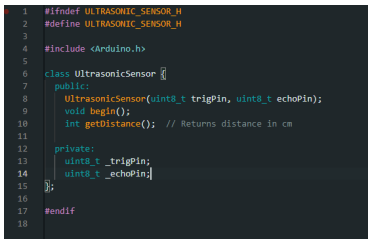
```ino
#include "IR_Sensor.h"

IRSensor::IRSensor(uint8_t pin) : _pin(pin) {}

void IRSensor::begin() {
  pinMode(_pin, INPUT);
}

bool IRSensor::isLineDetected() {
  return digitalRead(_pin) == HIGH;
}
```

```ino
#ifndef COLOR_SENSOR_H
#define COLOR_SENSOR_H

#include <Arduino.h>

struct ColorCalibration {
  float ratioRG;
  float ratioRB;
};

class ColorSensor {
  public:
    ColorSensor(uint8_t s0, uint8_t s1, uint8_t s2, uint8_t s3, uint8_t outPin);
    void begin();
    char detectColor(const ColorCalibration& colorA, const ColorCalibration& colorB);

  private:
    uint8_t _s0, _s1, _s2, _s3, _outPin;
    int readFrequency(bool s2, bool s3);
};

#endif
```



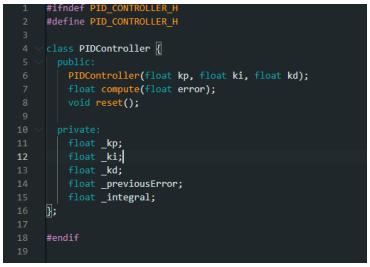
```ino
#include "Color_Sensor.h"
#include <Arduino.h>

ColorSensor::ColorSensor(uint8_t s0, uint8_t s1, uint8_t s2, uint8_t s3, uint8_t outPin)
  : _s0(s0), _s1(s1), _s2(s2), _s3(s3), _outPin(outPin) {}

void ColorSensor::begin() {
  pinMode(_s0, OUTPUT);
  pinMode(_s1, OUTPUT);
  pinMode(_s2, OUTPUT);
  pinMode(_s3, OUTPUT);
  pinMode(_outPin, INPUT);

  digitalWrite(_s0, HIGH);
  digitalWrite(_s1, LOW);
}

char ColorSensor::detectColor(const ColorCalibration& colorA, const ColorCalibration& colorB) {
  const int samples = 5;
  long rSum = 0, gSum = 0, bSum = 0;

  for (int i = 0; i < samples; i++) {
    int r = readFrequency(LOW, LOW);
    int g = readFrequency(HIGH, HIGH);
    int b = readFrequency(LOW, HIGH);

    rSum += r;
    gSum += g;
    bSum += b;

    delay(100);
  }

  float rAvg = rSum / (float)samples;
  float gAvg = gSum / (float)samples;
  float bAvg = bSum / (float)samples;

  float ratioRG = rAvg / gAvg;
  float ratioRB = rAvg / bAvg;

  Serial.println("----------------------");
  Serial.print("R="); Serial.print(rAvg, 1);
  Serial.print(" G="); Serial.print(gAvg, 1);
  Serial.print(" B="); Serial.print(bAvg, 1);
```

```ino
  Serial.print("R="); Serial.print(rAvg, 1);
  Serial.print(" G="); Serial.print(gAvg, 1);
  Serial.print(" B="); Serial.print(bAvg, 1);
  Serial.print(" | RatioRG="); Serial.print(ratioRG, 3);
  Serial.print(" RatioRB="); Serial.println(ratioRB, 3);

  float diffA = abs(ratioRG - colorA.ratioRG) + abs(ratioRB - colorA.ratioRB);
  float diffB = abs(ratioRG - colorB.ratioRG) + abs(ratioRB - colorB.ratioRB);

  if (diffA < diffB) {
    Serial.println("Detected Color: A");
    Serial.println("----------------------");
    return 'A';
  } else {
    Serial.println("Detected Color: B");
    Serial.println("----------------------");
    return 'B';
  }
}

int ColorSensor::readFrequency(bool s2, bool s3) {
  digitalWrite(_s2, s2);
  digitalWrite(_s3, s3);
  delay(50);
  return pulseIn(_outPin, LOW);
}
```

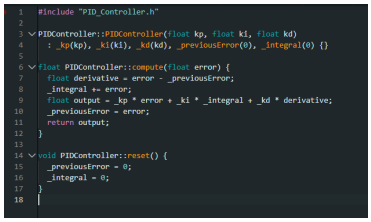


```h
#ifndef ULTRASONIC_SENSOR_H
#define ULTRASONIC_SENSOR_H

#include <Arduino.h>

class UltrasonicSensor {
  public:
    UltrasonicSensor(uint8_t trigPin, uint8_t echoPin);
    void begin();
    int getDistance();  // Returns distance in cm

  private:
    uint8_t _trigPin;
    uint8_t _echoPin;
};

#endif
```

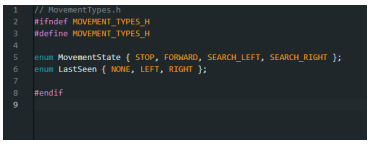
```cpp
#include "Ultrasonic_Sensor.h"

UltrasonicSensor::UltrasonicSensor(uint8_t trigPin, uint8_t echoPin)
  : _trigPin(trigPin), _echoPin(echoPin) {}

void UltrasonicSensor::begin() {
  pinMode(_trigPin, OUTPUT);
  pinMode(_echoPin, INPUT);
}

int UltrasonicSensor::getDistance() {
  digitalWrite(_trigPin, LOW);
  delayMicroseconds(2);
  digitalWrite(_trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(_trigPin, LOW);

  long duration = pulseIn(_echoPin, HIGH, 10000);  // 10ms timeout
  if (duration == 0) return -1;  // No echo received
  int cm = duration * 0.034 / 2;
  if (cm < 2 || cm > 400) return -1;  // Out of range
  return cm;
}
```



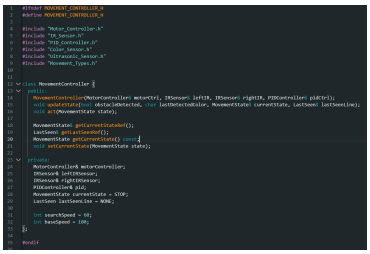
```h
#ifndef PID_CONTROLLER_H
#define PID_CONTROLLER_H

class PIDController {
  public:
    PIDController(float kp, float ki, float kd);
    float compute(float error);
    void reset();

  private:
    float _kp;
    float _ki;
    float _kd;
    float _previousError;
    float _integral;
};

#endif
```



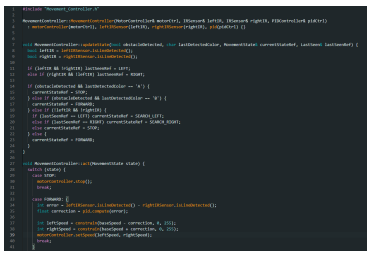
```cpp
#include "PID_Controller.h"

PIDController::PIDController(float kp, float ki, float kd)
  : _kp(kp), _ki(ki), _kd(kd), _previousError(0), _integral(0) {}

float PIDController::compute(float error) {
  float derivative = error - _previousError;
  _integral += error;
  float output = _kp * error + _ki * _integral + _kd * derivative;
  _previousError = error;
  return output;
}

void PIDController::reset() {
  _previousError = 0;
  _integral = 0;
}
```



```h
// MovementTypes.h
#ifndef MOVEMENT_TYPES_H
#define MOVEMENT_TYPES_H

enum MovementState { STOP, FORWARD, SEARCH_LEFT, SEARCH_RIGHT };
enum LastSeen { NONE, LEFT, RIGHT };

#endif
```

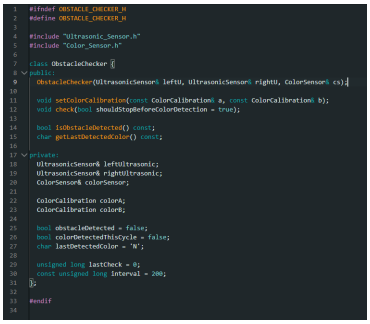


```h
#ifndef MOVEMENT_CONTROLLER_H
#define MOVEMENT_CONTROLLER_H

#include "Motor_Controller.h"
#include "IR_Sensor.h"
#include "PID_Controller.h"
#include "Color_Sensor.h"
#include "Ultrasonic_Sensor.h"
#include "Movement_Types.h"

class MovementController {
public:
    MovementController(MotorController& motorCtrl, IRSensor& leftIR, IRSensor& rightIR, PIDController& pidCtrl);
    void updateState(bool obstacleDetected, char lastDetectedColor, MovementState& currentState, LastSeen& lastSeenLine);
    void act(MovementState state);

    MovementState& getCurrentStateRef();
    LastSeen& getLastSeenRef();
    MovementState getCurrentState() const;
    void setCurrentState(MovementState state);

private:
    MotorController& motorController;
    IRSensor& leftIRSensor;
    IRSensor& rightIRSensor;
    PIDController& pid;
    MovementState currentState = STOP;
    LastSeen lastSeenLine = NONE;

    int searchSpeed = 60;
    int baseSpeed = 100;
};

#endif
```

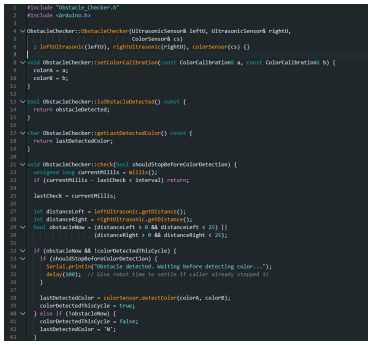


```cpp
#include "Movement_Controller.h"

MovementController::MovementController(MotorController& motorCtrl, IRSensor& leftIR, IRSensor& rightIR, PIDController& pidCtrl)
  : motorController(motorCtrl), leftIRSensor(leftIR), rightIRSensor(rightIR), pid(pidCtrl) {}

void MovementController::updateState(bool obstacleDetected, char lastDetectedColor, MovementState& currentStateRef, LastSeen& lastSeenRef) {
  bool leftIR = leftIRSensor.isLineDetected();
  bool rightIR = rightIRSensor.isLineDetected();

  if (leftIR && !rightIR) lastSeenRef = LEFT;
  else if (rightIR && !leftIR) lastSeenRef = RIGHT;

  if (obstacleDetected && lastDetectedColor == 'A') {
    currentStateRef = STOP;
  } else if (obstacleDetected && lastDetectedColor == 'B') {
    currentStateRef = FORWARD;
  } else if (!leftIR && !rightIR) {
    if (lastSeenRef == LEFT) currentStateRef = SEARCH_LEFT;
    else if (lastSeenRef == RIGHT) currentStateRef = SEARCH_RIGHT;
    else currentStateRef = STOP;
  } else {
    currentStateRef = FORWARD;
  }
}

void MovementController::act(MovementState state) {
  switch (state) {
    case STOP:
      motorController.stop();
      break;

    case FORWARD: {
      int error = leftIRSensor.isLineDetected() - rightIRSensor.isLineDetected();
      float correction = pid.compute(error);

      int leftSpeed = constrain(baseSpeed - correction, 0, 255);
      int rightSpeed = constrain(baseSpeed + correction, 0, 255);
      motorController.setSpeeds(leftSpeed, rightSpeed);
      break;
    }
```

```cpp
    case SEARCH_LEFT:
      motorController.setSpeed(-searchSpeed, searchSpeed);
      break;

    case SEARCH_RIGHT:
      motorController.setSpeed(searchSpeed, -searchSpeed);
      break;
  }
}

// === ===

MovementState& MovementController::getCurrentStateRef() {
  return currentState;
}

LastSeen& MovementController::getLastSeenRef() {
  return lastSeenLine;
}

MovementState MovementController::getCurrentState() const {
  return currentState;
}

void MovementController::setCurrentState(MovementState state) {
  currentState = state;
}
```



```h
#ifndef OBSTACLE_CHECKER_H
#define OBSTACLE_CHECKER_H

#include "Ultrasonic_Sensor.h"
#include "Color_Sensor.h"

class ObstacleChecker {
public:
    ObstacleChecker(UltrasonicSensor& leftU, UltrasonicSensor& rightU, ColorSensor& cs);

    void setColorCalibration(const ColorCalibration& a, const ColorCalibration& b);
    void check(bool shouldStopBeforeColorDetection = true);

    bool isObstacleDetected() const;
    char getLastDetectedColor() const;

private:
    UltrasonicSensor& leftUltrasonic;
    UltrasonicSensor& rightUltrasonic;
    ColorSensor& colorSensor;

    ColorCalibration colorA;
    ColorCalibration colorB;

    bool obstacleDetected = false;
    bool colorDetectedThisCycle = false;
    char lastDetectedColor = 'N';

    unsigned long lastCheck = 0;
    const unsigned long interval = 200;
};

#endif
```



```cpp
#include "Obstacle_Checker.h"
#include <Arduino.h>

ObstacleChecker::ObstacleChecker(UltrasonicSensor& leftU, UltrasonicSensor& rightU,
                                 ColorSensor& cs)
  : leftUltrasonic(leftU), rightUltrasonic(rightU), colorSensor(cs) {}

void ObstacleChecker::setColorCalibration(const ColorCalibration& a, const ColorCalibration& b) {
  colorA = a;
  colorB = b;
}

bool ObstacleChecker::isObstacleDetected() const {
  return obstacleDetected;
}

char ObstacleChecker::getLastDetectedColor() const {
  return lastDetectedColor;
}

void ObstacleChecker::check(bool shouldStopBeforeColorDetection) {
  unsigned long currentMillis = millis();
  if (currentMillis - lastCheck < interval) return;

  lastCheck = currentMillis;

  int distanceLeft = leftUltrasonic.getDistance();
  int distanceRight = rightUltrasonic.getDistance();
  bool obstacleNow = (distanceLeft > 0 && distanceLeft < 25) ||
                     (distanceRight > 0 && distanceRight < 25);

  if (obstacleNow && !colorDetectedThisCycle) {
    if (shouldStopBeforeColorDetection) {
      Serial.println("Obstacle detected. Waiting before detecting color...");
      delay(100);  // Give robot time to settle if caller already stopped it
    }

    lastDetectedColor = colorSensor.detectColor(colorA, colorB);
    colorDetectedThisCycle = true;
  } else if (!obstacleNow) {
    colorDetectedThisCycle = false;
    lastDetectedColor = 'N';
  }
```

```h
#ifndef OBSTACLE_HANDLER_H
#define OBSTACLE_HANDLER_H

#include "Motor_Controller.h"
#include "Movement_Controller.h"
#include "Obstacle_Checker.h"
#include "IR_Sensor.h"

class ObstacleHandler {
public:
    ObstacleHandler(MotorController& mc,
                    MovementController& mv,
                    ObstacleChecker& checker,
                    IRSensor& leftIR,
                    IRSensor& rightIR);

    void handleObstacle();

private:
    MotorController& motorController;
    MovementController& movementController;
    ObstacleChecker& obstacleChecker;
    IRSensor& leftIRSensor;
    IRSensor& rightIRSensor;

    void reverse(unsigned long duration);
    void pivotLeft(unsigned long duration);
    void pivotRight(unsigned long duration);
    void moveForward(unsigned long duration);
    bool searchForLine(unsigned long maxDuration);
};

#endif
```

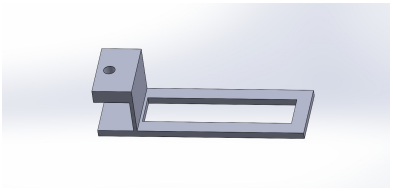
```cpp
#include "Obstacle_Handler.h"
#include <Arduino.h>

ObstacleHandler::ObstacleHandler(MotorController& mc, MovementController& mv, ObstacleChecker& checker, IRSensor& leftIR, IRSensor& rightIR)
  : motorController(mc), movementController(mv), obstacleChecker(checker), leftIRSensor(leftIR), rightIRSensor(rightIR) {}

void ObstacleHandler::handleObstacle() {
  Serial.println("Avoiding obstacle: Reversing...");
  reverse(500);

  Serial.println("Pivoting left...");
  pivotLeft(500);

  Serial.println("Going forward to bypass obstacle...");
  moveForward(600);
  if (searchForLine(500)) return;

  obstacleChecker.check();

  Serial.println("Realigning right...");
  pivotRight(500);
  if (searchForLine(500)) return;

  obstacleChecker.check();

  Serial.println("Going forward again...");
  moveForward(600);
  if (searchForLine(500)) return;

  obstacleChecker.check();

  Serial.println("Searching for line...");
  if (searchForLine(1500)) return;

  Serial.println("Line not found, initiating search pattern...");

  for (int i = 0; i < 3; i++) {
    Serial.print("Sweep attempt "); Serial.println(i + 1);
    pivotLeft(400);
    Serial.println("-> Sweep Forward");
    moveForward(1000, 100);
    delay(500);
    motorController.stop();
```

```cpp
    bool leftIR = leftIRSensor.isLineDetected();
    bool rightIR = rightIRSensor.isLineDetected();
    if (leftIR || rightIR) {
      Serial.println("Line found during left sweep!");
      motorController.stop();
      movementController.setCurrentState(FORWARD);
      return;
    }

    Serial.println("-> Sweep left");
    motorController.setSpeed(-60, 60);
    delay(500);
    motorController.stop();

    leftIR = leftIRSensor.isLineDetected();
    rightIR = rightIRSensor.isLineDetected();
    if (leftIR || rightIR) {
      Serial.println("Line found during left sweep!");
      motorController.stop();
      movementController.setCurrentState(FORWARD);
      return;
    }

    Serial.println("-> Sweep right");
    motorController.setSpeed(60, -60);
    delay(1000);
    motorController.stop();

    leftIR = leftIRSensor.isLineDetected();
    rightIR = rightIRSensor.isLineDetected();
    if (leftIR || rightIR) {
      Serial.println("Line found during right sweep!");
      motorController.stop();
      movementController.setCurrentState(FORWARD);
      return;
    }

    Serial.println("-> Return to center");
    motorController.setSpeed(-60, 60);
    delay(500);
    motorController.stop();
  }
```

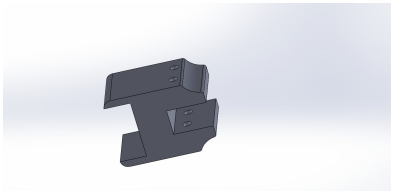
```cpp
  Serial.println("Line not found after search pattern. Stopping.");
  movementController.setCurrentState(STOP);
  motorController.stop();
}

void ObstacleHandler::reverse(unsigned long duration) {
  unsigned long startTime = millis();
  motorController.setSpeed(-100, -100);
  while (millis() - startTime < duration) {
    obstacleChecker.check();
    delay(10);
  }
  motorController.stop();
}

void ObstacleHandler::pivotLeft(unsigned long duration) {
  unsigned long startTime = millis();
  motorController.setSpeed(-60, 60);
  while (millis() - startTime < duration) {
    obstacleChecker.check();
    delay(10);
  }
  motorController.stop();
}

void ObstacleHandler::pivotRight(unsigned long duration) {
  unsigned long startTime = millis();
  motorController.setSpeed(60, -60);
  while (millis() - startTime < duration) {
    obstacleChecker.check();
    delay(10);
  }
  motorController.stop();
}

void ObstacleHandler::moveForward(unsigned long duration) {
  unsigned long startTime = millis();
  motorController.setSpeed(50, 50);
  while (millis() - startTime < duration) {
    obstacleChecker.check();
    bool leftIR = leftIRSensor.isLineDetected();
    bool rightIR = rightIRSensor.isLineDetected();
    if (leftIR || rightIR) {
      motorController.stop();
      movementController.setCurrentState(FORWARD);
      return;
    }
    delay(10);
  }
  motorController.stop();
}
```

```cpp
void ObstacleHandler::moveForward(unsigned long duration) {
  unsigned long startTime = millis();
  motorController.setSpeed(50, 50);
  while (millis() - startTime < duration) {
    obstacleChecker.check();
    bool leftIR = leftIRSensor.isLineDetected();
    bool rightIR = rightIRSensor.isLineDetected();
    if (leftIR || rightIR) {
      motorController.stop();
      movementController.setCurrentState(FORWARD);
      return;
    }
    delay(10);
  }
  motorController.stop();
}

bool ObstacleHandler::searchForLine(unsigned long maxDuration) {
  unsigned long startTime = millis();
  motorController.setSpeed(60, 60);
  while (millis() - startTime < maxDuration) {
    bool leftIR = leftIRSensor.isLineDetected();
    bool rightIR = rightIRSensor.isLineDetected();
    if (leftIR || rightIR) {
      motorController.stop();
      if (leftIR && !rightIR) movementController.getLastSeenRef() = LEFT;
      else if (rightIR && !leftIR) movementController.getLastSeenRef() = RIGHT;
      movementController.setCurrentState(FORWARD);
      return true;
    }
    delay(10);
  }
  motorController.stop();
  return false;
}
```

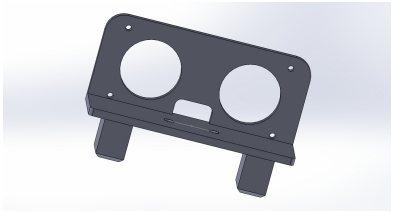
Appendix G: 3D-Printed Part Designs



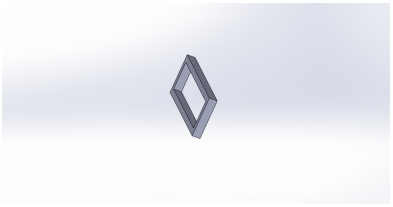
(a) IR Sensor Holder



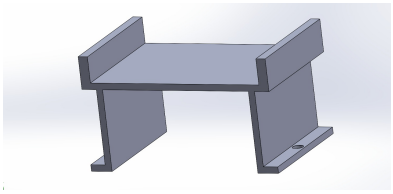
(b) Motor Holder



(c) Ultrasonic Sensor Holder



(d) IR Sensor Clip



(e) Battery & Breadboard Holder

Declaration of Originality

We hereby declare that this report is our own work and that all sources used have been properly acknowledged. We confirm that no part of this report has been submitted elsewhere for academic credit.

Place, Date: Bremen, July 8, 2025
Name: CHIMEZIE DANIEL CHIDI

Signature: 

Place, Date: Hamm, July 8, 2025
Name: BERTRAND MUNGU CHO

Signature: 

Place, Date: Soest, July 8, 2025
Name: GHIMIRE RIWAJ

Signature: 

Bibliography

[1] Arduino, "Arduino UNO R4 WiFi," [Online]. Available: https://www.arduino.cc/reference/en/boards/uno-r4-wifi/. [Accessed: 07-Jul-2025].

[2] HC-SR04 Datasheet, "Ultrasonic Distance Sensor," [Online]. Available: https://cdn.sparkfun.com/datasheets/Sensors/Proximity/HCSR04.pdf. [Accessed: 07-Jul-2025].

[3] Adafruit, "TCS3200 Colour Sensor," [Online]. Available: https://learn.adafruit.com/tcs34725-colorsensor. [Accessed: 07-Jul-2025].

[4] Texas Instruments, "L298N Dual H-Bridge Motor Driver Datasheet," [Online]. Available: https://www.ti.com/lit/ds/symlink/l298.pdf. [Accessed: 07-Jul-2025].

[5] S. Skogestad and I. Postlethwaite, *Multivariable Feedback Control: Analysis and Design*, 2nd ed. Chichester, U.K.: Wiley, 2005.

[6] R. Siegwart, I. R. Nourbakhsh, and D. Scaramuzza, *Introduction to Autonomous Mobile Robots*, 2nd ed. Cambridge, MA: MIT Press, 2011.

[7] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addison-Wesley, 1989.

[8] MathWorks, "PID Controller Tuning," [Online]. Available: https://www.mathworks.com/help/control/ug/pid-controller.html. [Accessed: 07-Jul-2025].

[9] OpenCV, "Open Source Computer Vision Library," [Online]. Available: https://opencv.org/. [Accessed: 07-Jul-2025].

[10] SolidWorks, "SOLIDWORKS 3D CAD Software," Dassault Systèmes, [Online]. Available: https://www.solidworks.com/. [Accessed: 07-Jul-2025].