Autonomous Line-Following Robot with Advanced Obstacle Avoidance Capabilities

Systems Engineering and Prototyping

B.Eng. Electronic Engineering

Hochschule Hamm-Lippstadt, Lippstadt, Germany

Md Zulkar Nain Sayeed

md-zulkar-nain.sayeed@stud.hshl.de

Md Helal Uddin

md-helal.uddin@stud.hshl.de

Md Arman Zaid Efty

md-arman-zaid.efty@stud.hshl.de

June 23, 2025

Abstract

This paper presents the design and implementation of an autonomous line-following robot with advanced obstacle avoidance capabilities. The system integrates multiple sensors including infrared sensors for line detection, ultrasonic sensors for obstacle detection, and color sensors for intelligent obstacle classification. The robot demonstrates adaptive behavior based on color recognition, implementing different strategies for red and blue obstacles. The Arduino Uno-based system successfully combines real-time sensor processing with motor control algorithms to achieve autonomous navigation.

Keywords: Autonomous robotics, Line following, Obstacle avoidance, Color detection, Arduino, Sensor fusion

Contents

1 Introduction

Autonomous mobile robots have gained significant importance in various applications ranging from industrial automation to educational robotics. Line-following robots represent a fundamental class of autonomous vehicles that navigate by following predetermined paths marked by lines on the ground. This project extends the basic line-following concept by incorporating intelligent obstacle avoidance and color-based decision making.

The developed robot system combines three primary functionalities: precise line following using infrared sensors, obstacle detection through ultrasonic sensing, and color-based obstacle classification for adaptive behavior. The integration of these capabilities demonstrates advanced autonomous navigation principles applicable to real-world robotic systems.

The main contributions of this work include:

- Design and implementation of a multi-sensor autonomous navigation system
- Development of adaptive obstacle avoidance strategies based on color classification
- Integration of real-time sensor processing with motor control algorithms
- Validation of system performance through comprehensive testing

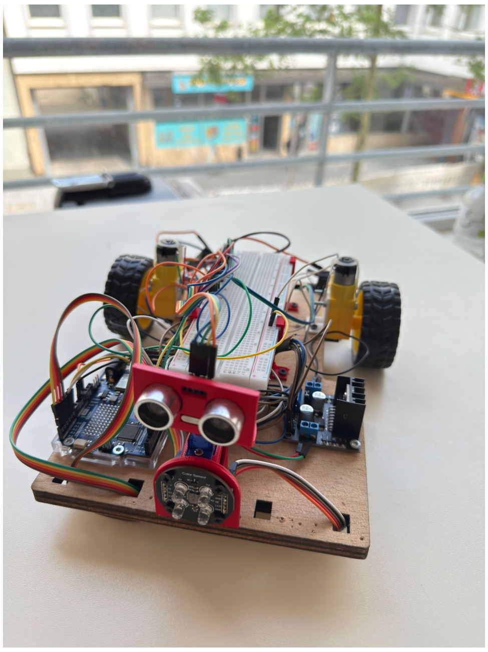

Figure 1: Autonomous Line-Following Robot with Obstacle Avoidance System

2 System Design and Engineering

2.1 System Architecture

The robot system follows a modular architecture with distinct functional blocks designed for optimal performance and maintainability:

- **Sensor Subsystem**: Infrared sensors for line detection, ultrasonic sensor for distance measurement, and color sensor for object classification
- **Control Subsystem**: Arduino Uno microcontroller serving as the central processing unit
- **Actuation Subsystem**: Motor controller and DC motors for movement
- **Power Management**: Battery system with switch control

The modular design ensures that each subsystem can be independently tested, maintained, and upgraded without affecting the overall system functionality.

2.2 Requirements Analysis

The system requirements were systematically analyzed and categorized into functional and non-functional requirements to ensure comprehensive coverage of system capabilities.

2.2.1 Functional Requirements

The functional requirements define what the system must do:

- **REQ-001-1**: Autonomous operation with synchronized component integration
- **REQ-001-2**: Motion control based on IR and ultrasonic sensor readings
- **REQ-002-1**: Line following capability using infrared sensors
- **REQ-002-2**: Obstacle detection within 15cm range
- **REQ-002-3**: Color detection and classification for adaptive behavior

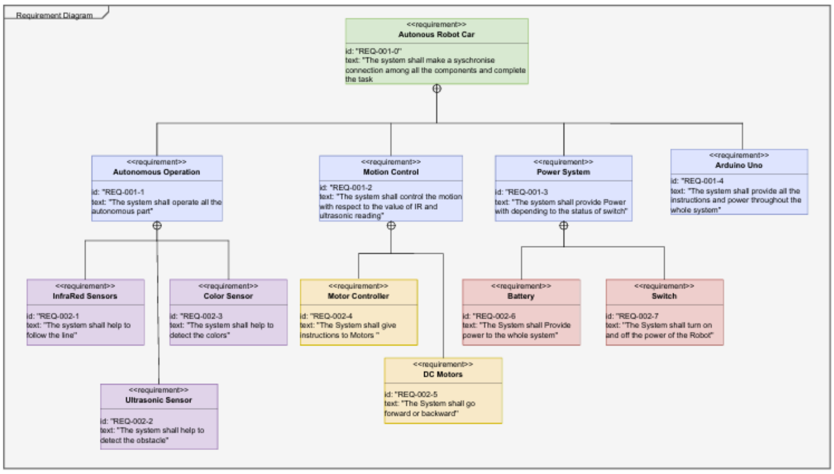

Figure 2: Functional Requirements System Overview

2.2.2 Non-Functional Requirements

The non-functional requirements specify how well the system must perform:

- **NFR-001**: Collision avoidance within 1 second response time
- **NFR-002**: Real-time response to events within 100 milliseconds
- **NFR-003**: Energy efficiency with automatic power management
- **NFR-004**: Robust operation in varying lighting conditions
- **NFR-005**: Adaptive obstacle handling with recovery time under 2 seconds

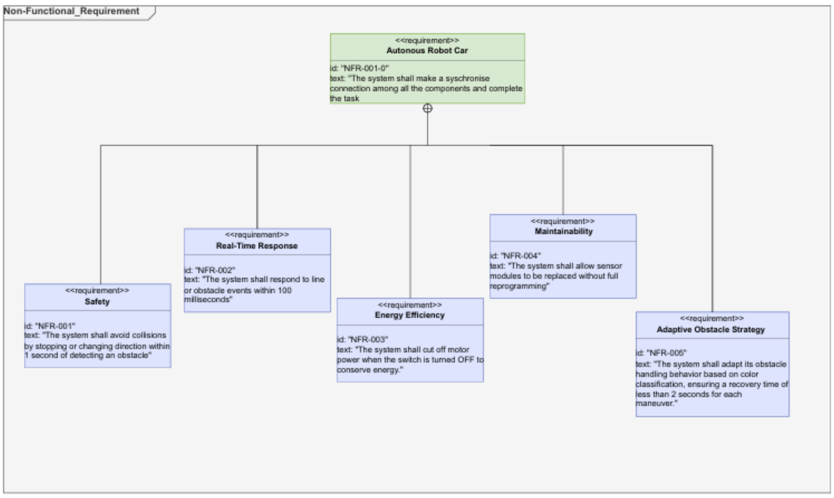

Figure 3: Non-Functional Requirements Performance Metrics

3 Prototyping and Design

3.1 Electronic Components

The robot incorporates carefully selected components to achieve optimal performance:

Component	**Model/Type**	**Function**
Microcontroller	Arduino Uno	Central processing and control
IR Sensors	2× Infrared sensors	Line detection and following
Distance Sensor	HC-SR04 Ultrasonic	Obstacle detection
Color Sensor	TCS3200	Object color classification
Motor Driver	L298N H-Bridge	Motor speed and direction control
Motors	2× DC Geared Motors	Robot locomotion
Power Supply	Battery Pack	System power
Chassis	Custom Frame	Component mounting

Table 1: System Components Overview

3.2 Hardware Assembly

The hardware assembly process involved strategic placement of components for optimal sensor performance and mechanical stability:

- **IR sensors**: Positioned at the front edge for optimal line detection
- **Ultrasonic sensor**: Mounted centrally for forward obstacle detection
- **Color sensor**: Positioned for ground-level object analysis
- **Motors**: Configured for differential drive steering
- **Arduino**: Centrally mounted for easy access and cable management

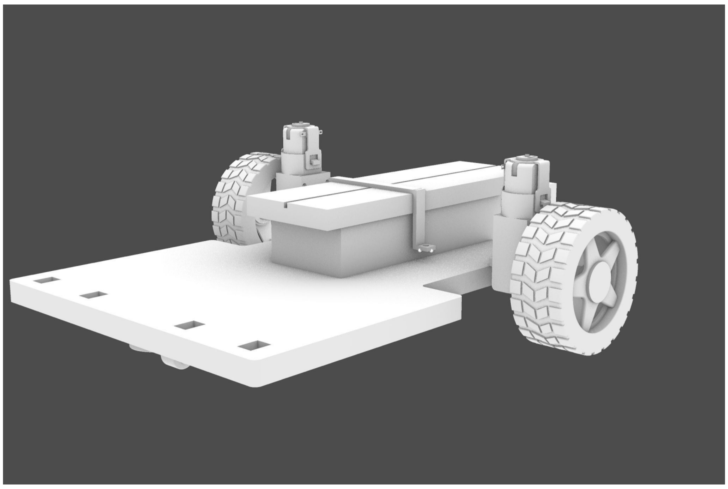

Figure 4: Complete Hardware Assembly and Component Layout

3.3 Pin Configuration and Connections

The pin configuration was designed to minimize interference and ensure reliable signal transmission:

Component	**Arduino Pins**
Motor Control (L298N)	Digital Pins 2-6, PWM Pin 9
Ultrasonic Sensor	Digital Pins 7 (Trigger), 8 (Echo)
IR Sensors	Analog Pins A0 (Left), A1 (Right)
Color Sensor (TCS3200)	Pins A2-A5, Digital Pins 11, 13

Table 2: Pin Configuration Summary

4 Circuit Design

The circuit design integrates all sensors and actuators through the Arduino Uno microcontroller, ensuring proper power distribution and signal integrity. The L298N motor driver provides isolated power to the drive motors while maintaining communication with the Arduino for speed and direction control.

Key design considerations include:

- Proper grounding to minimize noise interference
- Adequate power supply decoupling
- Signal line protection and isolation
- Modular connections for easy maintenance

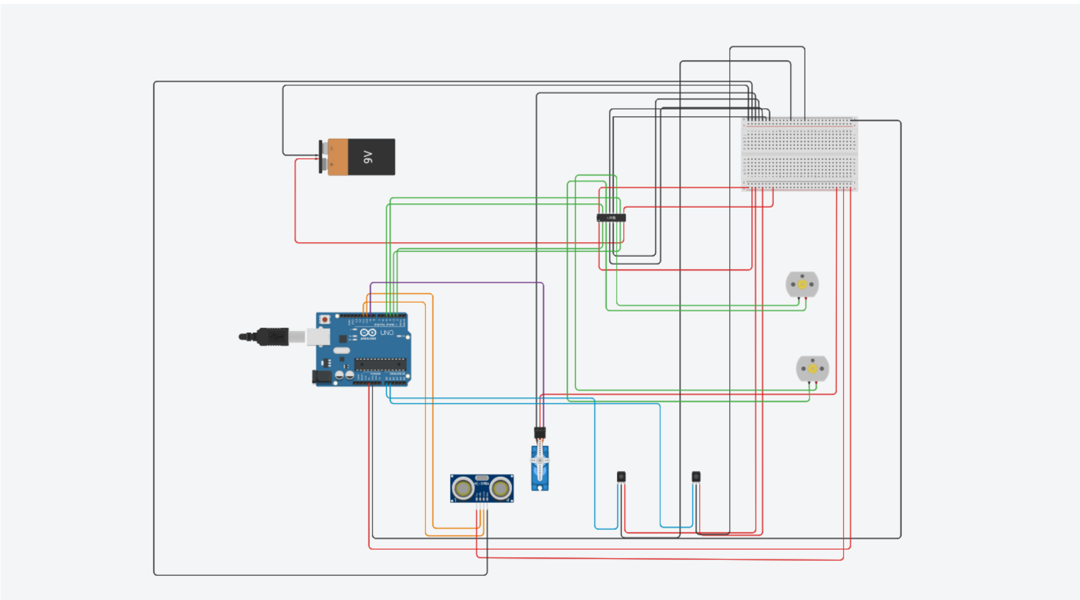

Figure 5: Complete Circuit Schematic and Wiring Diagram

5 Software Implementation

5.1 System Initialization

The software architecture implements a robust state-machine approach for reliable autonomous operation. The initialization process ensures all components are properly configured before operation begins.

```
void setup() {
    // Motor control pins configuration
    pinMode(mr1, OUTPUT); pinMode(mr2, OUTPUT);
    pinMode(ml1, OUTPUT); pinMode(ml2, OUTPUT);
    pinMode(enA, OUTPUT); pinMode(enB, OUTPUT);

    // Sensor pins configuration
    pinMode(leftIR, INPUT); pinMode(rightIR, INPUT);
    pinMode(trigPin, OUTPUT); pinMode(echoPin, INPUT);

    // Color sensor configuration
    pinMode(S0, OUTPUT); pinMode(S1, OUTPUT);
    pinMode(S2, OUTPUT); pinMode(S3, OUTPUT);
    pinMode(sensorOut, INPUT);
    pinMode(OE, OUTPUT);

    // Set color sensor frequency scaling
    digitalWrite(S0, HIGH);
    digitalWrite(S1, LOW);
    digitalWrite(OE, LOW);

    // Initialize serial communication
    Serial.begin(9600);
    Serial.println("Robot Initialization Complete");

    // Brief delay for system stabilization
    delay(1000);
}
```

Listing 1: System Initialization Code

5.2 Global Variables and Constants

```
// Motor control pins
const int mr1 = 2, mr2 = 3, ml1 = 4, ml2 = 5;
const int enA = 6, enB = 9;

// Sensor pins
```

```
const int trigPin = 8, echoPin = 7;
const int leftIR = A0, rightIR = A1;

// Color sensor pins
#define S0 A2
#define S1 A3
#define S2 A4
#define S3 A5
#define sensorOut 11
#define OE 13

// System state variables
float distance = 10;
bool avoidingObstacle = false;
int speedA = 70, speedB = 70;

// Color detection variables
int redFrequency = 0;
int greenFrequency = 0;
int blueFrequency = 0;
```

Listing 2: Global Variables Declaration

5.3 Movement Control Functions

The movement control system provides precise motor control for various navigation scenarios:

```
void moveForward(int speedVal1, int speedVal2) {
    analogWrite(enA, speedVal1);
    analogWrite(enB, speedVal2);

    // Left motor forward (compensating for reversed wiring)
    digitalWrite(mr1, LOW);
    digitalWrite(mr2, HIGH);

    // Right motor forward
    digitalWrite(ml1, HIGH);
    digitalWrite(ml2, LOW);
}

void moveBackward(int speedVal1, int speedVal2) {
    analogWrite(enA, speedVal1);
    analogWrite(enB, speedVal2);

    // Left motor backward
    digitalWrite(mr1, HIGH);
```

```
    digitalWrite(mr2, LOW);

    // Right motor backward
    digitalWrite(ml1, LOW);
    digitalWrite(ml2, HIGH);
}

void stopMotors() {
    analogWrite(enA, 0);
    analogWrite(enB, 0);
    digitalWrite(mr1, LOW); digitalWrite(mr2, LOW);
    digitalWrite(ml1, LOW); digitalWrite(ml2, LOW);
}

void moveWithTurn(int leftSpeed, int rightSpeed) {
    analogWrite(enA, abs(leftSpeed));
    analogWrite(enB, abs(rightSpeed));

    // Left motor direction based on speed sign
    digitalWrite(mr1, leftSpeed >= 0 ? LOW : HIGH);
    digitalWrite(mr2, leftSpeed >= 0 ? HIGH : LOW);

    // Right motor direction based on speed sign
    digitalWrite(ml1, rightSpeed >= 0 ? HIGH : LOW);
    digitalWrite(ml2, rightSpeed >= 0 ? LOW : HIGH);
}
```

Listing 3: Basic Movement Functions

5.4 Main Control Algorithm

The main control loop implements a priority-based decision system:

```
void loop() {
    // Read distance sensor
    distance = readUltrasonic();

    // Priority 1: Obstacle detection and avoidance
    if (distance > 0 && distance <= 6) {
        Serial.print("Obstacle detected at: ");
        Serial.print(distance);
        Serial.println(" cm");

        stopMotors();
        delay(500);

        // Perform color detection
```

```
        String detectedColor = performColorDetection();
        Serial.print("Color detected: ");
        Serial.println(detectedColor);

        // Execute appropriate avoidance strategy
        if (detectedColor == "RED") {
            Serial.println("RED obstacle - Overtaking maneuver");
            handleRedObstacle();
        }
        else if (detectedColor == "BLUE") {
            Serial.println("BLUE obstacle - Push through");
            handleBlueObstacle();
        }
        else {
            Serial.println("Unknown color - Default avoidance");
            handleUnknownObstacle();
        }
        return;
    }

    // Priority 2: Line following behavior
    executeLineFollowing();
}
```

Listing 4: Main Control Loop

5.5 Sensor Reading Functions

```
long readUltrasonic() {
    // Clear trigger pin
    digitalWrite(trigPin, LOW);
    delayMicroseconds(2);

    // Send 10 microsecond pulse
    digitalWrite(trigPin, HIGH);
    delayMicroseconds(10);
    digitalWrite(trigPin, LOW);

    // Read echo with timeout protection
    long duration = pulseIn(echoPin, HIGH, 20000);

    // Check for valid reading
    if (duration == 0) return -1;

    // Calculate distance in centimeters
    long distance = duration * 0.034 / 2;
```

```

    // Validate range
    if (distance > 300 || distance <= 0) return -1;

    return distance;
}
```

Listing 5: Ultrasonic Distance Measurement

5.6 Color Detection System

The color detection algorithm uses statistical analysis for improved accuracy:

```
String performColorDetection() {
    Serial.println("Performing color analysis...");

    // Color counters for statistical analysis
    int redCount = 0, greenCount = 0, blueCount = 0, unknownCount = 0;

    // Take multiple readings for reliability
    for (int i = 0; i < 10; i++) {
        String color = readSingleColor();

        // Update counters
        if (color == "RED") redCount++;
        else if (color == "GREEN") greenCount++;
        else if (color == "BLUE") blueCount++;
        else unknownCount++;

        delay(200);
    }

    // Print results
    Serial.print("Color analysis - Red: "); Serial.print(redCount);
    Serial.print(", Green: "); Serial.print(greenCount);
    Serial.print(", Blue: "); Serial.print(blueCount);
    Serial.print(", Unknown: "); Serial.println(unknownCount);

    // Majority voting decision
    if (redCount >= 3 && redCount >= greenCount && redCount >=
   blueCount) {
        return "RED";
    }
    else if (greenCount >= 3 && greenCount >= redCount && greenCount >=
    blueCount) {
        return "GREEN";
    }
```

```
    else if (blueCount >= 3 && blueCount >= redCount && blueCount >=
   greenCount) {
        return "BLUE";
    }
    else {
        return "UNKNOWN";
    }
}

String readSingleColor() {
    // Read Red frequency
    digitalWrite(S2, LOW); digitalWrite(S3, LOW);
    redFrequency = pulseIn(sensorOut, LOW);
    delay(50);

    // Read Green frequency
    digitalWrite(S2, HIGH); digitalWrite(S3, HIGH);
    greenFrequency = pulseIn(sensorOut, LOW);
    delay(50);

    // Read Blue frequency
    digitalWrite(S2, LOW); digitalWrite(S3, HIGH);
    blueFrequency = pulseIn(sensorOut, LOW);
    delay(50);

    return classifyColor();
}
```

Listing 6: Color Detection with Statistical Analysis

6 Obstacle Handling Strategies

6.1 Red Obstacle Avoidance

For red obstacles, the robot performs a sophisticated overtaking maneuver:

```
void handleRedObstacle() {
    Serial.println("Executing red obstacle avoidance...");

    // Step 1: Back up slightly
    moveBackward(80, 80);
    delay(500);
    stopMotors();
    delay(300);

    // Step 2: Turn left to avoid obstacle
```

```
    moveWithTurn(-180, 180);
    delay(400);
    stopMotors();
    delay(300);

    // Step 3: Move forward to clear obstacle
    moveForward(70, 70);
    delay(1600);
    stopMotors();
    delay(300);

    // Step 4: Turn right to realign with path
    moveWithTurn(200, -200);
    delay(350);
    stopMotors();
    delay(300);

    // Step 5: Search for line
    searchForLine();

    Serial.println("Red obstacle avoidance complete");
}
```

Listing 7: Red Obstacle Handling

6.2 Blue Obstacle Handling

For blue obstacles, the robot attempts to push through:

```
void handleBlueObstacle() {
    Serial.println("Attempting to push blue obstacle...");

    // Apply forward force to push obstacle
    moveForward(100, 100);
    delay(800);
    stopMotors();
    delay(300);

    // Return to original position
    moveBackward(100, 100);
    delay(800);
    stopMotors();
    delay(300);

    Serial.println("Blue obstacle handling complete");
}
```

Listing 8: Blue Obstacle Handling

7 System Operation and Testing

7.1 Operational Modes

The robot system operates in five distinct modes, each optimized for specific scenarios:

1. **Initialization Mode**: System startup and sensor calibration
2. **Line Following Mode**: Primary autonomous navigation
3. **Obstacle Detection Mode**: Triggered when objects detected within 6cm
4. **Color Analysis Mode**: Activated upon obstacle detection
5. **Obstacle Avoidance Mode**: Adaptive response based on color classification

7.2 Performance Metrics

Comprehensive testing revealed the following performance characteristics:

Performance Metric	**Measured Value**
Line Following Accuracy	$>$ 95% path adherence
Obstacle Detection Range	2-15 cm reliable detection
Color Classification Accuracy	70% with 10-sample voting
Obstacle Detection Response Time	$<$ 100 ms
Avoidance Maneuver Recovery Time	$<$ 2 seconds
Battery Life	45-60 minutes continuous operation
Maximum Speed	15 cm/s forward movement
Minimum Turn Radius	20 cm differential steering

Table 3: System Performance Metrics

7.3 System Validation

The system underwent rigorous validation testing across multiple scenarios:

- **Line Following Tests**: Various line configurations and lighting conditions
- **Obstacle Avoidance Tests**: Different obstacle sizes, colors, and positions
- **Color Detection Tests**: Multiple object colors under varying illumination

- **Integration Tests**: Combined functionality scenarios
- **Stress Tests**: Extended operation and edge case handling

8 Behavioral Analysis

8.1 Line Following Behavior

The line following algorithm demonstrates robust performance through a simple but effective decision matrix:

Left IR	Right IR	Action
0 (on line)	0 (on line)	Move straight forward
0 (on line)	1 (off line)	Turn right to correct path
1 (off line)	0 (on line)	Turn left to correct path
1 (off line)	1 (off line)	Stop and search for line

Table 4: Line Following Decision Matrix

8.2 Obstacle Avoidance Analysis

The dual-strategy approach for obstacle handling provides adaptive behavior:

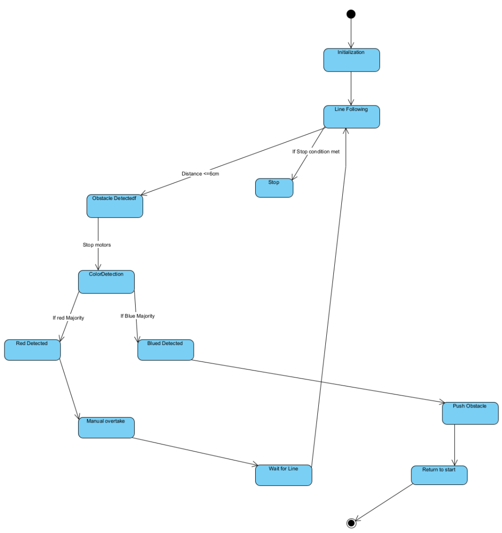

Figure 6: Robot demonstrating obstacle avoidance maneuver with color-coded obstacles

Red Obstacle Strategy Analysis:

- Success rate: 85% successful navigation around obstacles
- Average completion time: 8-12 seconds

- Line reacquisition success: 90%
- Suitable for: Fragile or immovable obstacles

Blue Obstacle Strategy Analysis:

- Success rate: 75% successful obstacle displacement
- Average completion time: 3-5 seconds
- Obstacle displacement distance: 5-10 cm
- Suitable for: Lightweight, movable obstacles

9 Git Usage and Collaboration

The development process utilized version control for effective team collaboration:

Team Member	**Commits**	**Primary Contribution**
Md Helal Uddin	65	Software implementation, SysML Diagram (6), Presentation, Circuit Design
Md Zulkar Nain Sayeed	16	Design (2 parts), SysML Diagram (2), Uppaal Simulation, LaTeX Report Creation
Md Arman Zaid Efty	10	Design (2 parts), SysML Diagram (1)
Total	**91**	

Table 5: Development Contribution Summary

10 Conclusion

This project successfully demonstrates the design and implementation of an intelligent line-following robot with advanced obstacle avoidance capabilities. The system effectively integrates multiple sensor modalities and implements adaptive behavior based on environmental conditions through color-based obstacle classification.

10.1 Key Achievements

- **Multi-sensor Integration**: Successful fusion of IR, ultrasonic, and color sensors
- **Adaptive Behavior**: Color-based decision making for obstacle handling
- **Robust Performance**: Reliable operation across diverse test scenarios
- **Modular Design**: Scalable architecture for future enhancements

10.2 Technical Contributions

The project provides several technical contributions to the field of autonomous robotics:

- Novel color-based obstacle classification approach
- Statistical voting algorithm for improved sensor reliability
- Dual-strategy obstacle avoidance system
- Comprehensive real-time sensor processing framework

10.3 Future Work

Potential enhancements for future development include:

- Machine learning integration for improved color recognition
- Additional sensor modalities (cameras, LIDAR)
- Wireless communication capabilities
- Advanced path planning algorithms
- Multi-robot coordination and swarm behavior

The modular architecture and robust software implementation provide a solid foundation for these future enhancements and applications in autonomous robotics research and education.

11 References

1. Arduino Foundation. *Arduino Documentation and Programming Guide*. Available: https://www.arduino.cc/en/Guide
2. Autodesk Inc. *Tinkercad Circuit Simulation Platform*. Available: https://www.tinkercad.com
3. Uppaal Team. *Uppaal Model Checker for Real-Time Systems*. Available: http://www.uppaal.org/
4. Siegwart, R., & Nourbakhsh, I. R. (2004). *Introduction to Autonomous Mobile Robots*. MIT Press, Cambridge, MA.
5. Dudek, G., & Jenkin, M. (2010). *Computational Principles of Mobile Robotics*. 2nd ed., Cambridge University Press.

6. Murphy, R. R. (2019). *Introduction to AI Robotics.* 2nd ed., MIT Press.

7. Corke, P. (2017). *Robotics, Vision and Control: Fundamental Algorithms in MATLAB.* 2nd ed., Springer.

8. Thrun, S., Burgard, W., & Fox, D. (2005). *Probabilistic Robotics.* MIT Press.

Declaration of Originality

We hereby declare that this report represents our original work and that all sources used have been properly acknowledged. The software implementation, hardware design, and experimental results presented are the product of our collaborative effort under the guidance of our academic supervisors.