Project Title

End-Term Project Report

For

Practical Robotics Projects with Arduino (CSE 4571)

Submitted by

Dinanath Dash	**2241004161**
Himanshu Sekhar Dash	**2241004163**
Priyansu Nayak	**2241014108**
Ashutosh Raj	**2241018015**
Swarnabha Roy	**2241018192**
Subhojeet Sarkar	**2241019321**

B. Tech. 7th Semester, CSE

Project Supervisor

Dr. Rahul Priyadarshi **(Assistant Professor/Associate Professor)**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Institute of Technical Education and Research

SIKSHA 'O' ANUSANDHAN DEEMED TO BE UNIVERSITY

Bhubaneswar, Odisha, India.

(January, 2026)

DECLARATION

We certify that

a. The work contained in this report is original and has been done by us under the guidance of our supervisor.

b. The work has not been submitted to any other Institute for any degree or diploma.

c. We have followed the guidelines provided by the Department in preparing the report.

d. Whenever we have used materials (data, theoretical analysis, figures, and text) from other sources, we have given due credit to them by citing them in the text of the report and giving their details in the reference.

Name of the Student	Registration Number	Signature
Dinanath Dash	2241004161	
Himanshu Sekhar Dash	2241004163	
Priyansu Nayak	2241014108	
Ashutosh Raj	2241018015	
Swarnabha Roy	2241018192	
Subhojeet Sarkar	2241019321	

REPORT APPROVAL

The report entitled

Line Following Robot

by

Dinanath Dash	**2241004161**
Himanshu Sekhar Dash	**2241004163**
Priyansu Nayak	**2241014108**
Ashutosh Raj	**2241018015**
Swarnabha Roy	**2241018192**
Subhojeet Sarkar	**2241019321**

is approved for

Internet of Things Projects using Python (CSE 4110)

in

Computer Science and Engineering

_____

Supervisor's Name

_____

Date: _____

Place: _____

CONTENTS

ABSTRACT

This project presents the design and development of an autonomous Line Following Robot using an Arduino-based embedded control system. The robot detects and follows a predefined track using infrared reflective sensors that differentiate between black and white surfaces. Sensor data is processed by the Arduino UNO, which generates precise motor-control signals through an L298N motor driver to maintain accurate path tracking.

The objective is to implement a low-cost, reliable navigation system suitable for academic learning and foundational industrial automation concepts such as Automated Guided Vehicles (AGVs). The system integrates core robotics principles-sensor acquisition, real-time decision-making, and motor actuation-while maintaining mechanical stability through a lightweight chassis, geared DC motors, and caster support.

Testing demonstrates consistent line detection, stable maneuvering on curves, and effective response to surface variations after calibration. The project provides a practical understanding of embedded systems, robotics, and control algorithms, and serves as a scalable baseline for advanced autonomous navigation applications.

Chapter 1: Introduction

1.1. Motivation

Automation is becoming the backbone of modern manufacturing, logistics, and service systems. Line-following robots represent the simplest yet most practical form of autonomous navigation used in industries such as warehouse transport, assembly-line material handling, and guided mobility platforms. For students, building such a robot is a direct way to understand embedded systems, sensor integration, motion control, and real-time decision-making without unnecessary complexity. This project offers a hands-on approach to robotics fundamentals while keeping the system low-cost, scalable, and educationally valuable.

1.2. Design Goals

1.2.1 Purpose

The purpose of this project is to design and implement an autonomous mobile robot capable of detecting and following a continuous line path using IR sensors and an Arduino-based control system.

1.2.2 Scope

The system covers sensor-based path detection, microcontroller-based decision logic, motor control, chassis design, power management, and performance testing on different surface conditions. The scope remains limited to 2-wheel differential drive navigation without advanced path-planning or wireless communication.

1.2.3 Applicability

The concepts implemented here apply to Automated Guided Vehicles (AGVs), robotic process automation, academic robotics labs, and small-scale industrial training setups. The design serves as a foundation for more complex robots such as maze solvers, obstacle-avoiding robots, and autonomous delivery units.

1.3. Problem Statement

Traditional manual material movement is error-prone, inconsistent, and inefficient. Fully automated navigation systems are expensive and complex to deploy. A low-cost, Arduino-based line-following robot provides an accessible solution by autonomously tracking a predefined path using simple reflective sensors, enabling reliable movement in controlled environments. The challenge is to ensure accurate line detection, stable motor control, and consistent performance across varying track patterns.

1.4. Organization of the Report

- Chapter 2 reviews existing work related to line-following robots and sensor-based navigation.
- Chapter 3 explains the system design, architecture, hardware components, and implementation steps.
- Chapter 4 outlines testing procedures and evaluates performance metrics.
- Chapter 5 discusses cost factors, safety considerations, socio-economic impact, and global relevance.
- Chapter 6 lists engineering tools, standards, and development practices applied in the project.
- Chapter 7 documents the encountered problems, bugs, faults, and technical challenges.
- Chapter 8 presents teamwork contributions and evaluation.
- Chapter 9 concludes the project with key findings and potential future improvements.

Chapter 2: Literature Survey

2.1. Background - work done so far

2.1.1. Early analog and threshold-based systems

- Early line followers used simple phototransistors or LDRs with analog comparators. They detect contrast and switch motors left/right.
- Strengths: very cheap, simple to implement.
- Weaknesses: brittle - fails under varying lighting, surface reflections, and on thin/curved lines.

2.1.2. IR reflectance sensor arrays (digital + analog)

- Modern hobby and educational robots commonly use IR reflective sensors (QRE1113, TCRT5000) in 3-8 sensor arrays. Raw analog voltages or digital thresholds feed into a microcontroller.
- Strengths: compact, fast response, low cost, easy to interface with Arduino.
- Weaknesses: sensitivity to surface color/texture and ambient IR; requires calibration and filtering.

2.1.3. Control strategies: Bang-bang (on/off), Proportional, PID

- Bang-bang control (if-left then turn-left) is simplest but oscillatory.
- Proportional control (adjust turn proportional to error from sensor centroid) smooths motion and increases speed capability.
- Full PID controllers are commonly used in research/competitions for tight tracking and to compensate dynamic lag from motors. PID reduces overshoot on curves and helps maintain stable high speed.
- Practical note: PID works, but tuning (especially derivative term) must be done carefully because noisy IR signals amplify derivative action.

2.1.4. Sensor fusion and filtering

- Median filters, moving averages, and simple debouncing across sensors greatly reduce false readings. Centroid calculation from weighted sensor values gives a continuous error signal for proportional/PID control.
- Some systems add a light sensor to adapt thresholds under changing illumination.

2.1.5. Advanced approaches: vision, fuzzy logic, ML

- Camera-based line detection (OpenCV) provides robustness for complex tracks but increases computational cost and power requirements. Used in advanced or research platforms.
- Fuzzy logic controllers have been applied to handle uncertain sensor inputs and provide smoother transitions; they can outperform naive PID in highly variable conditions.
- Machine learning methods (CNNs) exist for robust track following and generalization but require datasets, heavier hardware (Raspberry Pi/Jetson), and more development time.

2.1.6. Motor drivers and mechanical considerations

- H-bridge drivers (L298N, L293D) are standard for small DC motors. L298N handles higher currents but is less efficient and generates heat; modern MOSFET-based drivers are more efficient.

- Geared motors with encoders improve repeatability and enable closed-loop velocity control, but add cost/complexity. For basic projects, geared DC motors with proper gearing and wheels are adequate.

2.1.7. Industrial AGV practices (applied lessons)

- Industrial systems use multiple sensors, redundancy, and robust calibration routines; they often use differential odometry plus line guidance for error correction.
- Safety, fail-safe stopping, and predictable behavior at junctions are emphasized - useful design points even for a student project.

2.2 Gaps and practical limitations observed in prior work

- **Ambient sensitivity:** IR reflectance modules need calibration; direct sunlight or glossy surfaces break many hobby setups.
- **Speed vs. accuracy trade-off:** Higher speeds increase tracking error unless advanced control (tuned PID, encoders) is used.
- **Intersections and decision-making:** Simple systems struggle at forks or when the line is broken; added logic or more sensors required.
- **Hardware inefficiencies:** Cheap H-bridge modules like L293D waste power and heat; they can limit motor performance under load.
- **Over-engineering risk:** Vision and ML are powerful but unnecessary for the learning objective and increase power/complexity.

2.3 How this project positions itself

- Use an **IR sensor array** (3-5 sensors) with centroid computation for a continuous error signal - it's the sweet spot between simplicity and performance.
- Implement **proportional control** first, then add integral (PI) if steady-state offset appears; add derivative only if you can filter noise well. Don't throw a full PID at noisy raw readings without filtering.
- Include a simple **calibration routine** (read sensors on line and off line) to set thresholds before each run. That fixes 60% of failure cases.
- **Keep the mechanics solid:** matched geared motors, good wheel traction, low-friction caster—poor chassis design will drown any software improvements.
- If you want to scale later, design for an optional encoder or a more efficient motor driver; but don't add them now unless you have extra time.

2.4 Key references

- Practical tutorials on IR sensor arrays and centroid error computation (numerous maker/hobbyist guides).
- PID tuning resources for small mobile robots (practical guides rather than dense theory).
- OpenCV-based line-following examples if you plan to migrate to camera-based systems later.

Chapter 3: Design Scheme

3.1. System Design

The system is designed around a closed-loop control model where the robot continuously senses the path, processes the readings, and actuates the motors accordingly. The IR sensor array detects the contrast between the line and the background surface. These signals are fed into the Arduino UNO, which processes the data and computes the required motor actions. The motor driver (L298N) serves as an interface between the microcontroller and the DC motors, providing the necessary current and voltage for movement. The overall system relies on differential drive control to achieve smooth line tracking and turning.

Key Subsystems:

- **Sensing Subsystem:** IR reflective sensors for line detection.
- **Processing Subsystem:** Arduino UNO for decision-making and control logic.
- **Actuation Subsystem:** DC geared motors controlled via L298N driver.
- **Power Subsystem:** AA battery pack powering motors and microcontroller.
- **Mechanical Subsystem:** Lightweight chassis with wheels and caster support.

3.2. Architecture

The architecture follows a modular, layered approach:

Layer 1 — Input Layer (Sensors)

- Multiple IR sensors measure reflected infrared light to determine whether the robot is on or off the line.
- Sensors generate either digital (HIGH/LOW) or analog signals.

Layer 2 — Control Layer (Arduino)

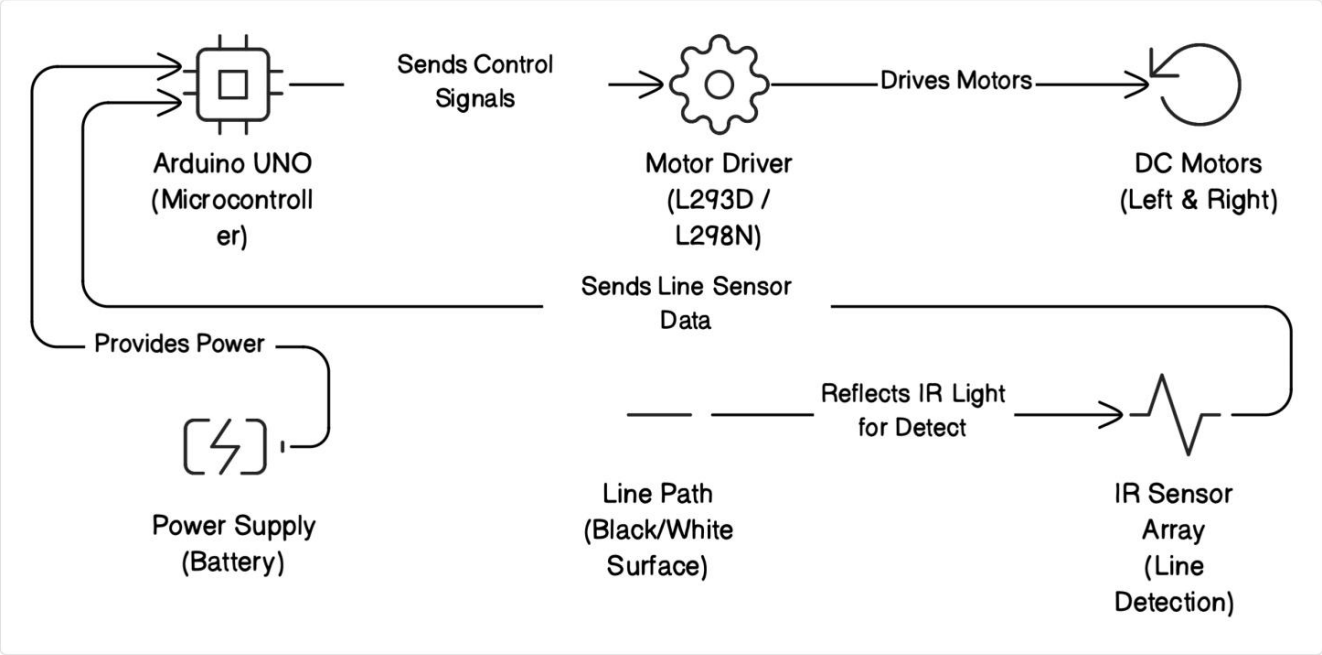
- Arduino reads sensor values.
- Line position error is calculated (e.g., left deviation, right deviation).
- Control logic (Proportional or basic conditional logic) maps sensor patterns to motor commands.

Layer 3 — Actuation Layer (Motor Driver + Motors)

- L298N receives PWM and direction signals from Arduino.
- It drives the two BO geared motors independently.
- Differential speeds allow turning, correction, and straight-line tracking.

Layer 4 — Mechanical Layer (Chassis)

- Two rear motors provide movement.
- Front castor support ensures stability.
- Chassis houses electronics, wiring, and power module.



3.3. Component Design

3.3.1. IR Sensor Array

- Uses multiple IR reflective sensors (typically 3–5).
- Each sensor outputs a digital signal based on surface reflectivity.
- Black line = low reflectance → sensor LOW
- White background = high reflectance → sensor HIGH
- Array placement ensures coverage across line width for accurate lateral detection.

3.3.2. Arduino UNO

- ATmega328P microcontroller.
- Reads sensor states at high frequency.
- Executes the control algorithm to determine motor speed and direction.
- Outputs PWM signals for precise speed control.

3.3.3. L298N Motor Driver

- Dual H-Bridge circuit.
- Drives two DC motors independently.
- Supports direction control and PWM-based speed regulation.
- Necessary because Arduino pins cannot supply adequate current.

3.3.4. BO Geared Motors + Wheels

- 100 RPM geared motors provide balanced torque and speed.
- Rubber wheels ensure grip and reduce slippage on track.

3.3.5. Power Supply

- AA battery pack delivering 6–9V.

- Separate 5V line regulated for Arduino (either onboard regulator or L298N's 5V output when safe).

3.3.6. Chassis

- 3D printed or acrylic body.
- Mounting points for sensors, Arduino, driver, battery, and motors.

3.4. Implementation

Steps followed in implementation:

3.4.1. Sensor Calibration

- Sensors tested on black line and white surface.
- Threshold values determined for reliable detection.

3.4.2. Hardware Assembly

- Motors mounted on chassis.
- IR sensor array fixed at appropriate distance from ground.
- Arduino and L298N placed centrally for balanced wiring.
- Battery pack and switch installed.

3.4.3. Electrical Connections

- IR sensors → Arduino digital pins.
- Arduino → L298N (PWM + direction pins).
- L298N → Motors.
- Common ground established.

3.4.4. Software Logic

- Read sensor values continuously.
- Detect line position error (left, right, center).
- Map error to motor adjustments using conditional or proportional logic.
- Apply PWM signals for smooth speed control.

3.4.5. Testing & Tuning

- Robot tested on straight and curved paths.
- Adjusted speed, PWM values, and sensor sensitivity to minimize oscillation.
- Ensured stability at moderate speeds.

3.5. Design Evolution

3.5.1. Initial Prototype

- Simple two-sensor design with on/off control.
- Led to oscillations and unstable tracking on curves.

3.5.2. Upgraded Sensor Array

- Moved to a 3–5 sensor configuration for better resolution.

- Enabled proportional error measurement.

3.5.3. Improved Control Strategy

- Added speed tuning using PWM.
- Reduced jerky motion and improved curvature handling.

3.5.4. Mechanical Adjustments

- Sensor height optimized to increase contrast sensitivity.
- Wheel alignment corrected for straight-line motion.

3.5.5. Final System

- Stable tracking on varying curves.
- More reliable performance under moderately changing lighting conditions.

Chapter 4: Testing, Analysis, and Evaluation

4.1. Testing Criterions

To evaluate the performance of the Line Following Robot, several testing criteria were defined. Each criterion focuses on functional accuracy, responsiveness, and stability of the system under different operating conditions.

4.1.1. Relevance

This criterion checks whether the robot performs the core task it was designed for: *detecting and following a continuous line accurately*.

Tests conducted:

4.1.1.1. Line Detection Test

- Sensors placed on black-and-white surfaces.
- Verified correct HIGH/LOW transitions for each sensor.

4.1.1.2. Straight Path Test

- Robot placed on a straight black line.
- Checked if motors maintain alignment and minimal lateral drift.

4.1.1.3. Curve Tracking Test

- Robot tested on shallow and moderate curves.
- Assessed how quickly the control algorithm corrects deviations.

Outcome:

The robot reliably detected the line and maintained accurate tracking on straight paths and moderate curves. Occasional deviations occurred on very sharp turns but were corrected automatically.

4.1.2. Effectiveness

This evaluates how well the robot follows the line under various conditions.

Metrics used:

4.1.2.1. Response Time

- Time taken for the robot to correct its trajectory when sensors detect an off-line condition.
- Observed response: fast correction with no significant overshoot for normal speeds.

4.1.2.2. Stability

- Measured oscillation (left-right wobble).
- With proper PWM tuning, oscillation was minimized.

4.1.2.3. Robustness Test

- Tested on surfaces with different reflectivity (matte vs. slightly glossy).
- Verified performance under moderate ambient lighting variation.

4.1.2.4. Error Recovery

- Robot intentionally pushed partially off the line.

- Observation: fast recovery when at least one sensor still detected the line.

Outcome:

The robot showed solid effectiveness. It could handle typical reflections, maintain stability, and recover quickly from small disturbances.

4.1.3. Efficiency

Efficiency focuses on power usage, motor performance, and overall smoothness of operation.

4.1.2.1. Power Efficiency

- With AA batteries, the robot operated for a reasonable duration without voltage drops severely affecting performance.
- Motors consumed the bulk of power; the Arduino remained stable throughout tests.

4.1.2.2. Motor & Control Efficiency

- PWM control ensured efficient motor speed regulation.
- Smooth acceleration and reduced jerky movements were observed after tuning.

4.1.2.3. Track Completion Time

- Robot completed a standard test track without major pauses or stalls.
- Faster speeds were possible but increased oscillation; optimal speed was chosen for balanced performance.

4.1.2.4. Thermal Behavior

- L298N driver warmed up at higher speeds but remained within acceptable operating limits.

Outcome:

The system maintained efficient power and motion control, achieving reliable track completion times without performance degradation.

Chapter 5: Socio-Economic Issues associated with the Project

5.1. Detailed Cost Analysis

5.1.1. Cost Analysis

The line-following robot is designed to be a low-cost educational and prototyping platform.

Its affordability makes it accessible for students, hobbyists, and small research labs.

Cost factors include:

- Electronic components (Arduino, sensors, driver)
- Mechanical components (motors, chassis)
- Power system (battery pack)
- Miscellaneous items (wires, switches, fasteners)

The overall cost is significantly lower than industrial AGV systems, demonstrating how inexpensive hardware can still produce meaningful autonomous behavior.

5.1.2. Bill of Materials

Sl. No.	Component	Specification	Qty	Approx. Cost (INR)
1	Arduino UNO	ATmega328P, 16 MHz	1	250–600
2	L298N Motor Driver	Dual H-Bridge	1	150–200
3	IR Sensor Modules	Reflective IR, digital	2	150–250
4	BO Geared Motors + Wheels	100 RPM, 65mm wheels	2	250–300
5	Castor Wheel	Ball-type	1	40–60
6	AA Battery Pack + Holder	6–9V output	1	120–180
7	ON/OFF Switch	SPST	1	10–20
8	Jumper Wires	Male–Male, Male–Female	-	40–50
9	3D Printed/Acrylic Chassis	Lightweight	1	300–600

Estimated Total Cost: ~ INR 1400 – 1900

This price point demonstrates excellent cost-efficiency for an autonomous robotics platform.

5.2. Safety issues

Even though the system operates at low power and voltage, several safety considerations must be acknowledged:

5.2.1. Electrical Safety

- Ensure correct wiring to avoid short circuits.
- Batteries should not be reverse-polarized.

- Motor driver heat should be monitored during extended runs.

5.2.2. Mechanical Safety

- Moving wheels can cause minor pinching hazards.
- Chassis parts must be firmly mounted to avoid detachment during motion.

5.2.3. Component Overheating

- L298N may heat at higher motor loads; adequate ventilation is required.
- Motors can heat slightly after continuous operation.

5.2.4. Power Management

- Use fresh batteries; low voltage can cause erratic behavior.
- Always switch OFF the system before modifying the circuit.

Overall, the project is safe for laboratory and educational use when standard precautions are followed.

5.3. Global Impact

This project contributes positively in several global contexts:

5.3.1. Educational Impact

- Provides hands-on learning for students worldwide.
- Helps bridge the gap between theoretical robotics and practical implementation.

5.3.2. Technological Awareness

- Encourages understanding of automation, which is essential in modern economies.
- Prepares learners for advanced robotics, AI, and embedded systems.

5.3.3. Sustainable Development

- Promotes low-cost automation.
- Demonstrates how inexpensive components can solve real-world problems.
- Reduces need for large-scale industrial hardware in early prototyping.

5.3.4. Industry Relevance

- Line-following principles form the basis of AGVs used globally.
- Knowledge gained from this project is directly applicable to logistics, warehousing, and manufacturing automation.

5.4. Lifelong Learning

Developing this project fosters several lifelong learning attributes:

- **Technical Curiosity:** Encourages continuous exploration of sensors, control systems, and robotics.
- **Problem-Solving Mindset:** Challenges students to debug hardware and software issues.
- **Adaptability:** Teaches how to modify designs based on testing and real-world constraints.
- **Collaboration Skills:** Reinforces teamwork, communication, and project management.
- **Foundation for Advanced Learning:** Serves as a stepping stone toward machine vision, AI-driven robots, path planning, and autonomous navigation research.

Chapter 6: Engineering Tools and Standards used in the Project

This chapter outlines the tools, platforms, and engineering standards applied during the development of the Arduino-based Line Following Robot.

6.1 Engineering Tools Used

6.1.1. Arduino IDE

- Used for writing, compiling, and uploading the control code to the Arduino UNO.
- Provides serial monitor capabilities for debugging sensor values, motor control signals, and calibration routines.

6.1.2. Circuit Simulation Tools

- Platforms like Proteus, TinkerCAD, or Fritzing were used to simulate wiring diagrams and validate circuit behavior before hardware implementation.
- Helpful for verifying sensor connections, motor driver logic, and power distribution.

6.1.3. PlantUML / Diagramming Tools

- Used to generate system architecture and block diagrams.
- Ensures clear visual representation of subsystem interactions.

6.1.4. Soldering & Hardware Tools

- Basic tools such as soldering iron, wire strippers, and multimeter were used for connection integrity, continuity checks, and hardware testing.

6.1.5. 3D Modeling Tools (If chassis is 3D printed)

- Tools like Fusion 360, SolidWorks, or TinkerCAD for designing the robot chassis.
- Ensures proper mounting of sensors, motors, and electronics.

6.2 Software Libraries and Frameworks

Arduino Core Libraries

- digitalRead(), analogRead(), analogWrite(), pinMode() used for sensor interfaces and PWM control.

Motor Control Libraries

- Simple motor control logic implemented directly.
- Additional libraries for PID or smoothing could be used if extended functionality is required.

6.3 Engineering Standards Followed

6.3.1. IEEE Coding and Documentation Practices

- Clear variable naming, modular code, and inline explanation where needed.
- Ensures readability and maintainability.

6.3.2. Safety Standards in Low-Voltage Electronic Systems

- Followed basic guidelines for handling DC circuits, ensuring proper insulation, grounding, and current limits.

6.3.3. PCB/Prototyping Standards

- Proper layout for breadboard and sensor placement.
- Organized wiring to avoid interference and reduce electrical noise.

6.3.4. Mechanical Design Standards

- Stable center of gravity for robot chassis.
- Correct alignment of wheels and sensors to ensure accurate motion.

6.3.5. Testing Standards

- Consistent test conditions for sensor calibration, motor functionality, and line-following behavior.
- Repeated runs to ensure reliability and reproducibility.

Chapter 7: Problems, faults, bugs, challenges

7.1. Problems

7.1.1. Inconsistent Line Detection

- At early stages, the IR sensors produced fluctuating readings due to ambient lighting and reflective floor surfaces. This caused the robot to drift or lose the line.

7.1.2. Motor Speed Imbalance

- Even identical BO motors do not rotate at exactly the same speed. This caused the robot to veer even when sensor readings indicated a centered line.

7.1.3. Power Drops During Operation

- Using AA batteries caused noticeable voltage drops under load, especially when motors accelerated. This affected sensor accuracy and Arduino stability.

7.1.4. Mechanical Alignment Issues

- The initial sensor height and wheel alignment were not optimal, leading to uneven detection and reduced stability.

7.2. Faults

7.2.1. Loose Wiring Connections

- During testing, jumper wires occasionally loosened from vibration, causing random sensor or motor failures.

7.2.2. Incorrect Motor Polarity

- Initial wiring caused reversed motor directions, making the robot turn opposite to the intended direction.

7.2.3. Overheating of L298N Driver

- Extended operation at high speeds caused the motor driver to heat up, leading to reduced performance until cooled.

7.2.4. Poor Surface Contrast

- Certain floor surfaces had insufficient contrast between black and white, leading to sensor misreads.

7.3. Bugs

7.3.1. Incorrect Conditional Logic

- Early code versions had overlapping or conflicting if–else conditions, causing unpredictable turns.

7.3.2. Threshold Miscalculation

- Improper sensor calibration led to inaccurate HIGH/LOW thresholds, causing the robot to misinterpret line boundaries.

7.3.3. PWM Mismatch

- Different PWM values produced non-linear speed changes due to motor characteristics; required fine tuning.

7.3.4. Sensor Noise

- Fast reading loops produced noisy values from sensors, causing jittery movement until delays and filters were added.

7.4. Challenges

7.4.1. Achieving Smooth Turns

- The robot initially overshot curves. Tuning the PWM and sensor spacing was necessary to achieve smoother trajectory correction.

7.4.2. Handling Sharp Corners

- Sharp turns or intersections were difficult for the basic sensor array. Required adjustments to mounting angle and sensor spacing.

7.4.3. Balancing Speed and Stability

- Increasing speed introduced oscillation. Balancing between responsiveness and stability was a continuous tuning challenge.

7.4.4. Calibration Under Different Lighting

- IR sensors behave differently under sunlight, tube light, or LED light. Achieving robust performance under varying illumination required repeated calibration.

7.4.5. Weight Distribution

- Incorrect placement of batteries initially made the robot front-heavy, causing inconsistent sensor contact with the surface.

Chapter 8: Teamwork

8.1. Summary of team work

The project was completed through coordinated collaboration among all team members. Each member contributed to specific tasks while ensuring continuous communication and synchronized progress. The teamwork approach focused on dividing responsibilities based on technical strengths and maintaining collective ownership of the final outcome.

Key Contributions

I. System Design & Architecture

- Members collaborated to plan the hardware layout, sensor arrangement, and wiring structure. Joint decision-making ensured a stable, balanced, and practical design.

II. Hardware Assembly & Circuit Integration

- Tasks such as motor installation, sensor mounting, chassis construction, and soldering were shared among members. Regular cross-checking helped eliminate wiring faults.

III. Software Development

- Code development, debugging, and logic refinement were handled collaboratively. Members tested various control strategies and helped optimize PWM values and sensor thresholds.

IV. Testing & Calibration

- The team collectively performed repeated test runs to evaluate performance under different light and surface conditions. Each iteration involved adjustments based on observations from all members.

V. Documentation & Reporting

- Responsibilities for preparing chapters, diagrams, cost analysis, and reference compilation were split evenly, ensuring consistency and accuracy throughout the report.

Overall, teamwork played a critical role in the successful completion of this project. The combination of individual responsibilities and group-based problem-solving resulted in a dependable and thoroughly tested line-following robot.

8.1.1 Attributes

1	Attends group meetings regularly and arrives on time.
2	Contributes meaningfully to group discussions.
3	Completes group assignments on time.
4	Prepares work in a quality manner.
5	Demonstrates a cooperative and supportive attitude.
6	Contributes significantly to the success of the project.

8.1.2 Score

1=strongly disagree; **2**=disagree; **3**=agree; **4**=strongly agree

Student 1 _____ **Student 2** _____

Student 3 _____ **Student 4** _____

Student 5 _____ **Student 6** _____

Student 1	Evaluated by	
	Attributes	Student 1
	1	
	2	
	3	
	4	
	5	
	6	
	Grand Total	

Student 2	Evaluated by	
	Attributes	Student 1
	1	
	2	
	3	
	4	
	5	
	6	
	Grand Total	

_____

Signature of Student 1

_____

Signature of Student 2

Chapter 9: Conclusion

This project successfully demonstrated the design and development of an Arduino-based Line Following Robot capable of autonomously tracking a predefined path using infrared sensor feedback. Through systematic hardware integration, software implementation, and iterative testing, the robot achieved stable navigation on straight and curved tracks, validating the effectiveness of the chosen control strategy and mechanical design.

The work highlighted several core engineering concepts: real-time sensor acquisition, embedded decision-making, differential motor control, and performance tuning under variable environmental conditions. Challenges such as sensor noise, motor imbalance, and surface reflectivity were addressed through calibration, PWM adjustments, and mechanical refinement. These improvements contributed to reliable system behavior and consistent line-following performance.

Beyond technical results, the project reinforced essential skills in problem-solving, teamwork, time management, and documentation. The low-cost nature of the system makes it a practical platform for education and experimentation, and the modular architecture allows future enhancements such as PID control, obstacle detection, wireless telemetry, or camera-based vision.

In conclusion, the Line Following Robot achieved its intended objectives and provided a solid foundation for exploring more advanced autonomous robotic systems and intelligent navigation techniques.

References

1. Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., & Bengio, Y.

Generative Adversarial Nets.

In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., & Weinberger, K. Q. (Eds.), Advances in Neural Information Processing Systems 27, pp. 2672–2680, 2014.

Link: *https://proceedings.neurips.cc/paper_files/paper/2014/file/f033ed80deb0234979a61f95710dbe25-Paper.pdf*

2. Arduino Documentation.

"Line Following Robot"

Link: *https://projecthub.arduino.cc/lightthedreams/line-following-robot-34b1d3*

3. CircuitDigest.

" Line Follower Robot using Arduino UNO: How to Build (Step-by-Step Guide)"

Link: *https://circuitdigest.com/microcontroller-projects/arduino-uno-line-follower-robot*

4. YouTube Tutorials.

Various creators explaining Arduino line follower logic, motor driver wiring, and sensor calibration.

Link: *https://youtu.be/88nKA6nWmqw?si=lENPBcpj4-GW2y8O*

5. OpenAI.

"ChatGPT: Large Language Model for Text Assistance and Technical Documentation."

Accessed: 2025.

6. Generative AI Tools.

Used for generating diagrams (PlantUML), refining explanations, and improving documentation quality.

7. CirkitDesign Circuits.

For virtual simulation of Arduino + sensors.

Link: *https://app.cirkitdesigner.com/project/6aa4bf68-fe0a-4e69-939a-1217ca46beb7*

Appendix

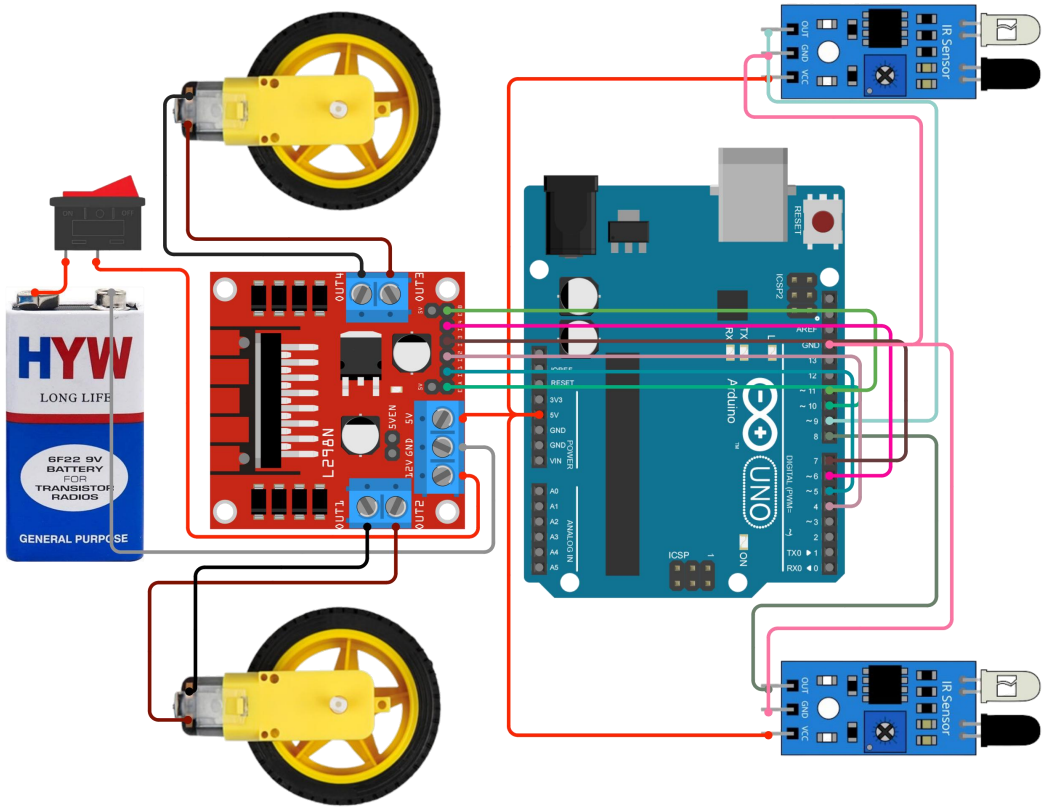
The appendix contains supplementary materials that support the main content of the project report. These items include circuit diagrams, code listings, test results, calibration tables, and any additional resources referenced during development.

A. Arduino Code

```
#define IR_SENSOR_RIGHT 8
#define IR_SENSOR_LEFT 9
#define MOTOR_SPEED 255

//Right motor
int enableRightMotor=10;
int rightMotorPin2=4;
int rightMotorPin1=5;

//Left motor
int enableLeftMotor=11;
int leftMotorPin2=6;
int leftMotorPin1=7;

void setup() {
  TCCR0B = TCCR0B & B11111000 | B00000010 ; //This sets PWM frequecny as 7812.5 hz.
  pinMode(enableRightMotor, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);
  pinMode(enableLeftMotor, OUTPUT);
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);

  pinMode(IR_SENSOR_RIGHT, INPUT);
  pinMode(IR_SENSOR_LEFT, INPUT);
  rotateMotor(0,0);
}


void loop() {

  int rightIRSensorValue = digitalRead(IR_SENSOR_RIGHT);
  int leftIRSensorValue = digitalRead(IR_SENSOR_LEFT);

  //If none of the sensors detects black line, then go straight
  if (rightIRSensorValue == LOW && leftIRSensorValue == LOW) {
    rotateMotor(MOTOR_SPEED, MOTOR_SPEED);
  }
  //If right sensor detects black line, then turn right
  else if (rightIRSensorValue == HIGH && leftIRSensorValue == LOW ) {
    rotateMotor(-MOTOR_SPEED, MOTOR_SPEED);
  }
  //If left sensor detects black line, then turn left
  else if (rightIRSensorValue == LOW && leftIRSensorValue == HIGH ) {
    rotateMotor(MOTOR_SPEED, -MOTOR_SPEED);
  }
  //If both the sensors detect black line, then stop
  else {
    rotateMotor(0, 0);
  }
}
```

```
void rotateMotor(int rightMotorSpeed, int leftMotorSpeed) {
  if (rightMotorSpeed < 0) {
    digitalWrite(rightMotorPin1,LOW);
    digitalWrite(rightMotorPin2,HIGH);
  }
  else if (rightMotorSpeed > 0) {
    digitalWrite(rightMotorPin1,HIGH);
    digitalWrite(rightMotorPin2,LOW);
  }
  else {
    digitalWrite(rightMotorPin1,LOW);
    digitalWrite(rightMotorPin2,LOW);
  }

  if (leftMotorSpeed < 0) {
    digitalWrite(leftMotorPin1,LOW);
    digitalWrite(leftMotorPin2,HIGH);
  }
  else if (leftMotorSpeed > 0) {
    digitalWrite(leftMotorPin1,HIGH);
    digitalWrite(leftMotorPin2,LOW);
  }
  else {
    digitalWrite(leftMotorPin1,LOW);
    digitalWrite(leftMotorPin2,LOW);
  }
  analogWrite(enableRightMotor, abs(rightMotorSpeed));
  analogWrite(enableLeftMotor, abs(leftMotorSpeed));
}
```

B. Circuit Diagram / Wiring Schematic



C. Additional Photos (Robot Build)