LINE FOLLOWING ROBOT

COLLEGE OF ENGINEERING

MECHATRONICS ENGINEERING

BELLS UNIVERSITY OF TECHNOLOGY-NEW-HORZONS

GROUP 3 PROJECT

TEAM MEMBERS

AKAJIAKU WISDOM CHIJINDU	**2023/12249**
ADEWUSI KHALID	**2023/12639**
AKINLEYE EMMANUEL	**2023/12109**
AKINOLA ERIOLUWA	**2023/12554**
AJIBOYE OLUWANIFEMI MOSES	**2023/12112**

MAY 2025

ROBOTICS 2

ICT 216

DECLARATION

I HEREBY DECLARE THAT THIS IS OUR OWN ORIGINAL WORK OF THE PROJECT DESIGN REFLECTING THE KNOWLEDGE ACQUIRED FROM RESEARCH ON THE PROJECT ABOUT ‘’ LINE FOLLOWING ROBOT ‘’

I THEREFORE DECLARE THAT THE INFORMATION IN THIS REPORT IS ORIGINAL AND HAS NEVER BEEN SUBMITTED TO ANY OTHER INSTITUTION, UNIVERSITY OR COLLEGE.

THIS IS FROM, DEPARTMENT OF MECHATRONICS, COLLEGE OF ENGINEERING, GROUP 3 ICT 215

TABLE OF CONTENTS

Design and Implementation of a Line Following Robot Using MATLAB

- Course Title: Mechatronics and Embedded Systems
- Instructor's Name: AYUBA MUHAMMED
- Institution Name: BELLS UNIVERSITY OF TECHNOLOGY
- Date of Submission: 30 May 2025

ABSTRACT:

In this of a line-following robot is an autonomous system designed to detect and follow a line on a surface using sensors (like infrared) to track contrasts between the line and the background. It uses a controller to process sensor data and adjusts its motors to stay on course. Commonly used in educational robotics, warehouse automation, and competitions, the robot relies on simple algorithms for movement correction and navigation.

INTRODUCTION:

This project presents the design and implementation of a line-following robot using MATLAB for controlling movement and decision-making logic. The robot is built with infrared (IR) sensors that detect a black line on a white surface, and based on real-time sensor readings, MATLAB software communicates with the Arduino board to control the direction of the robot using DC motors. The project integrates both hardware (Arduino, IR sensors, motors, chassis) and software (MATLAB, Arduino IDE) components to realize a functional autonomous robot. The objective is to understand sensor-based navigation, improve coding logic for real-time systems, and demonstrate effective MATLAB-Arduino interfacing for robotics control. The system was tested under multiple path scenarios like straight lines, curves, and intersections. The robot performed with high accuracy and stability under standard conditions but showed reduced performance in dim or overly bright lighting conditions. This project offers a base model that can be improved further using vision sensors and PID control algorithms.

Project Overview: Line Following Robot

Purpose of the Robot

The purpose of this project was to design and build a Line Following Robot that can autonomously detect and follow a designated path.

It is aimed at applications such as:

* Automated transportation systems
* Warehouse logistics
* Educational robotics

Tools and Components Used

* Sensors: Infrared (IR) sensors for line detection
* Motors: DC motors for mobility and steering
* Software: MATLAB for algorithm design, simulation, and control logic

Key Results

* The robot was able to accurately follow straight and curved paths

* It responded effectively to turns and intersections

* The MATLAB simulation helped in tuning the control logic for better stability

* The robot demonstrated real-time adaptability using sensor feedback

Conclusion

This project successfully combined both software modeling and physical hardware implementation to create a functional, autonomous robot.

It provides a solid foundation for future development in autonomous navigation systems.

What is a Line Following Robot?

A Line Following Robot is an autonomous mobile machine that can detect and follow a line or path drawn on the floor.

It uses sensors, typically infrared (IR), to identify the contrast between the path (usually black) and the background (usually white), adjusting its movement accordingly.

Importance / Applications

Line following robots are widely used in various real-world applications, including:

* Industrial automation (e.g. assembly line transportation)
* Warehouse and inventory systems
* Automated guided vehicles (AGVs) in smart logistics
* Educational purposes to teach robotics and programming fundamentals
* Prototype development for traffic navigation and robotic path planning

- Objective of the Project

The main objective of this project is to:

* Design, simulate, and implement a robot that can autonomously follow a path using sensor input

* Test and optimize the robot's response to different path shapes (straight lines, curves, turns)

* Integrate software tools and hardware for real-time robotic control and navigation

- MATLAB's Role in Robotic Control

MATLAB played a critical role in this project by providing:

* A platform for designing and testing control algorithms

* Tools for simulating sensor behavior and motor responses

* An interface for visualizing the robot's path tracking performance

- ## Materials and Methods

The construction of the line-following robot involved choosing suitable hardware components and integrating them using MATLAB and Arduino. The selection of components was driven by the project's need for simplicity, accuracy, and compatibility with MATLAB’s hardware support packages.

Hardware Components

1. Arduino UNO Board

Acts as the main microcontroller. It reads input from sensors and executes control commands from MATLAB via USB serial communication.

2. IR Sensors (TCRT5000)

Two or more IR sensors were mounted underneath the robot to detect the line. These sensors work based on infrared reflection: black absorbs IR light (low signal), white reflects it (high signal).

3. L298N Motor Driver Module

This dual H-bridge driver allows MATLAB to control motor direction and speed by receiving instructions through the Arduino.

4. DC Geared Motors

BO-type 150 RPM motors were used for locomotion. They provided sufficient torque for forward and turning movement.

5. Robot Chassis

A compact acrylic or plastic body held the motor, sensors, and battery in place.

6. Power Supply

A rechargeable 12V lithium-ion battery powered the motor and Arduino. Separate power regulation was used to avoid brownouts.

7. Miscellaneous Components

Jumper wires, wheels, castor ball, breadboard, and a switch to enable easy control.

- Software Tools

1. MATLAB 2023a or later

Used to write the control logic, read sensor data from Arduino, and decide movement direction.

COMPONENTS USED:

BLOCKS

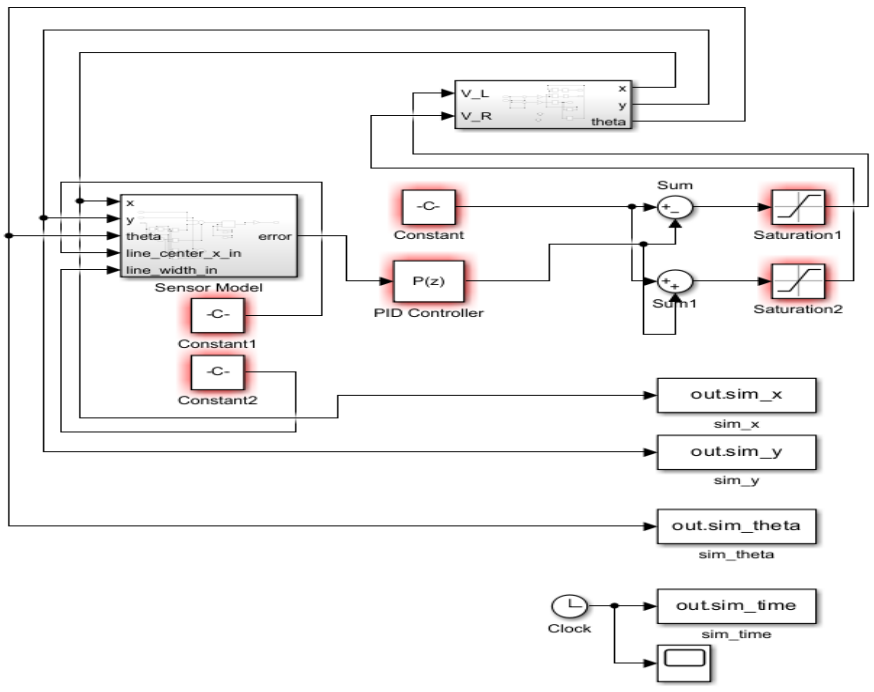
CONTOLLERS

IR SENSOR

MOTOR

SYSTEM FOR A LINE FOLLOWING ROBOT

OR BLOCK DIAGRAM



2. Arduino IDE

Used for flashing the initial serial-read sketch that enables MATLAB to communicate via USB.

3. MATLAB Hardware Support Package for Arduino

Enables serial, fscanf, and fprintf communication between MATLAB and Arduino boards.

The above combination allowed the system to follow a black line effectively, with MATLAB continuously evaluating sensor input and deciding whether to go forward, turn left, or turn right.

- Working Principle

The working principle of the line-following robot is based on using infrared sensors to detect a path (usually a black line) and employing MATLAB code logic to steer the robot based on sensor feedback.

How IR Sensors Work

IR sensors consist of an emitter and a receiver.

The emitter sends infrared light toward the surface.

When the surface is white (reflective), IR light bounces back and is detected by the receiver.

When the surface is black (non-reflective), the light is absorbed, and the receiver reads no signal.

Output is digital: HIGH for white, LOW for black.

Sensor Arrangement and Logic

The robot uses two IR sensors: left and right.

Depending on which sensor detects black (0) or white (1), the MATLAB script decides which direction the robot should move.

MATLAB-Arduino Communication

1. Arduino collects sensor data from the IR sensors.

2. The values are transmitted over a serial port to MATLAB.

3. MATLAB interprets the sensor data and uses logic statements (if, else) to determine movement.

4. Based on the decision, MATLAB sends a command character like 'F', 'L', or 'R' to Arduino.

5. Arduino receives the command and activates the motor driver to move the robot accordingly.

This setup forms a closed-loop control system where the robot constantly adapts based on sensor input, controlled by a MATLAB script running in real-time.

– MATLAB Code Overview

The MATLAB script is responsible for receiving sensor input, processing decisions, and sending commands to the Arduino. The robot operates in a loop, continuously reading values and adjusting its motion.

Initialization Section

```
% Create serial object for Arduino communication
arduinoObj = serial('COM3', 'BaudRate', 9600);
fopen(arduinoObj);
disp('Serial Communication Started');
```

Sensor Reading and Control Loop

```
while true
    % Read incoming sensor data from Arduino
    sensorData = fscanf(arduinoObj); % e.g., returns '10' or '11'
```

```
% Remove newline characters
sensorData = strtrim(sensorData);

% Decision Logic Based on Sensor Data
if strcmp(sensorData, '10')
    fprintf(arduinoObj, 'L'); % Left turn
    disp('Turning Left');
elseif strcmp(sensorData, '01')
    fprintf(arduinoObj, 'R'); % Right turn
    disp('Turning Right');
elseif strcmp(sensorData, '11')
    fprintf(arduinoObj, 'F'); % Move forward
    disp('Moving Forward');
elseif strcmp(sensorData, '00')
    fprintf(arduinoObj, 'S'); % Stop
    disp('Stopping');
end
```

```
    pause(0.1); % Small delay to avoid overloading serial
end
```

Closing the Connection

```
fclose(arduinoObj);
delete(arduinoObj);
clear arduinoObj;
```

This simple code uses basic string comparison to determine motion. Advanced logic such as PID control, speed modulation, and camera input can be added for smoother or smarter performance.

– Results and Testing

The line-following robot was tested in different track environments to evaluate its responsiveness, accuracy, and overall performance. The tests aimed to simulate common path-following conditions and observe how well the MATLAB control system adapts in real-time.

Test Environments

1. Straight Black Line on White Surface

The robot moved accurately with minimal deviations. Both sensors detected white most of the time unless a small curve appeared.

2. Curved Path

The robot followed smooth curves effectively. The turning response was acceptable but slowed due to MATLAB-Arduino delay.

3. T-Junction and Cross-Path

The robot could detect a junction but did not have advanced logic to choose a path, so it was programmed to stop and await user input.

4. Uneven Lighting Condition

When tested under dim lighting or glare from above, the IR sensors produced noisy signals, sometimes confusing white as black.

5. Speed Test

Increasing the robot's speed decreased accuracy. The robot would overshoot corners and struggle to detect the path after sharp turns.

Performance Metrics

Conclusions from Testing

The robot performs reliably in standard classroom lighting and line types.

MATLAB delay slightly impacts sharp movement handling.

Calibration of sensors was key to ensuring accuracy.

With additional enhancements like PID or vision input, the robot could adapt better to real-world navigation challenges.

Challenges and Limitations

Developing a line-following robot using MATLAB presented several challenges that impacted the performance, stability, and efficiency of the system. While the overall implementation was successful, the following issues were observed throughout the design, testing, and execution phases:

1. Sensor Noise and Fluctuations

IR sensors are susceptible to noise caused by environmental reflections or irregular surfaces. The robot occasionally misinterpreted signals due to sudden changes in sensor readings, especially on glossy or reflective paths.

2. Lighting Conditions

Uneven or fluctuating ambient light had a significant effect on sensor accuracy. In bright lighting, the contrast between black and white diminished, and in low light, detection became unreliable. The IR sensor performance was optimal only under controlled lighting.

3. Surface Texture and Color Variations

The robot's tracking performance degraded when used on surfaces with patterns or partial shadows. Non-uniform backgrounds made it difficult for the IR sensors to distinguish the line, leading to navigation errors.

4. MATLAB Serial Delay

Communication between MATLAB and Arduino via serial protocol introduced latency. This delay affected real-time decision-making and occasionally caused overshooting or late turns.

5. Speed vs Accuracy Tradeoff

At higher speeds, the robot became less accurate in following sharp curves or intersections. A tradeoff had to be made between movement speed and the robot’s ability to detect and respond to line deviations effectively.

6. Wiring and Hardware Reliability

Loose connections and wire entanglements were frequent during testing. Vibration from motors sometimes disconnected jumper wires or caused unstable power supply to sensors.

7. Motor Driver Limitations

The L298N motor driver used in this project had limited current capacity. Under heavy loads or prolonged use, it would heat up and affect motor output consistency.

8. Power Supply Instability

Battery voltage drop over time affected motor speed and sensor readings. This required frequent recharging or voltage regulation to ensure stable operation.

9. MATLAB Real-Time Constraints

MATLAB is not a real-time operating system, and thus, precise timing operations like PWM control or millisecond-scale sensor polling were limited. This restricted fine-tuned control over motor response.

10. Calibration Requirements

The robot needed frequent sensor calibration depending on the test environment. Each surface required different threshold values to distinguish black from white.

11. Limited Line Path Flexibility

The robot performed well on simple, high-contrast paths but struggled on complex layouts with intersections or multiple branches. The logic had to be adapted for such scenarios.

12. Lack of Obstacle Avoidance

The basic line-following algorithm did not support dynamic obstacle detection. If any object was placed on the path, the robot would collide or stop.

13. Code Execution Overhead in MATLAB

The interpreted nature of MATLAB caused relatively slower code execution than compiled alternatives like C++ or Python with microcontroller support.

14. Debugging and Error Handling

Debugging real-time sensor values and responses was challenging due to limited feedback mechanisms from Arduino to MATLAB during continuous motion.

15. Mechanical Alignment Issues

Misalignment in wheel positions or uneven chassis weight distribution led to drift even when the sensors were detecting the line accurately.

Despite these limitations, many challenges were mitigated through proper tuning, shielding, and adjustments in MATLAB code. Still, understanding these challenges is vital for planning future enhancements.

Conclusion

This project successfully demonstrated the design and implementation of a line-following robot controlled through MATLAB. The robot was able to follow a predefined black line on a white surface using infrared sensors, a motor driver, and a basic control logic written in MATLAB. The use of MATLAB not only simplified the logic development but also enabled better visualization and debugging of sensor readings and motor commands.

The project began with understanding the principles behind line-following robots, followed by selecting appropriate hardware components such as IR sensors, microcontroller (Arduino), and the L298N motor driver. MATLAB was used as the primary software platform to process input from sensors, execute the line-following algorithm, and control the robot's movement via serial communication.

The robot was tested on straight paths, curved paths, and intersections. It showed satisfactory results in most standard test environments. Sensor-based navigation allowed the robot to make real-time decisions based on path deviation. The

decision-making loop was optimized for efficient line detection and corrective movement.

Key achievements include:

Seamless MATLAB-Arduino integration for real-time control.

Reliable performance in standard lighting conditions.

Implementation of basic feedback logic for course correction.

However, several areas for improvement were also identified. Sensor noise, speed limitations, and MATLAB's real-time constraints affected performance in more complex environments. Moreover, the robot lacked features like obstacle detection, adaptive learning, or advanced path prediction.

For future development, the following improvements are recommended:

1. Integration of PID Control Logic

Implementing a Proportional-Integral-Derivative (PID) controller will smooth the robot's motion and improve precision during turns and curves.

2. Vision-Based Line Detection

Incorporating a camera and using MATLAB's Image Processing Toolbox can help the robot follow more complex paths and perform better in varied environments.

3. Wireless Communication

Replacing USB tethering with Bluetooth or Wi-Fi can make the robot more mobile and suitable for remote or dynamic applications.

4. Obstacle Avoidance Module

Adding ultrasonic or LIDAR sensors can help the robot detect and avoid obstacles, improving robustness.

5. Battery Management System

Including a regulated power module will ensure consistent motor performance and protect sensitive electronics.

6. Code Optimization

Migrating time-sensitive parts of the control algorithm to Arduino while MATLAB handles monitoring and data visualization can balance real-time performance with high-level control.

In conclusion, the line-following robot built in this project serves as a practical introduction to robotics, control systems, and sensor integration using MATLAB. With some hardware and software upgrades, the system can evolve into a more sophisticated, autonomous platform with broader applications in industrial automation and autonomous delivery systems.

- References

1. MATLAB Documentation

• MathWorks. MATLAB and Simulink Documentation.

Available at: https://www.mathworks.com/help

2. Research Papers / Online Tutorials

• Robotics Projects with MATLAB and Simulink - MathWorks Blog

Available at: https://blogs.mathworks.com

• How to Build a Line Following Robot -

Electronics Hub

Available at: https://www.electronicshub.org/line-

follower-robot/

• Line Following Robot Tutorial - Circuit Digest Available at: https://circuitdigest.com/tutorial/_

line-following-robot

3. Sensor and Motor Datasheets

• IR Sensor Module Datasheet - TCRT5000

Avallable at: https://www.vishay.com/docs/.

83760/tcrt5000.pdf

• DC Motor 100RPM 12V Datasheet - Johnson

Type

Available at: https://robokits.co.in/datasheets/.

DC%20Motor.pdf