IJCRT.ORG ISSN : 2320-2882

INTERNATIONAL JOURNAL OF CREATIVE RESEARCH THOUGHTS (IJCRT)

An International Open Access, Peer-reviewed, Refereed Journal

Line Following Robot Using Arduino

[1]Abhishek Sahu, [2]Apoorva Nema, [3]Ayushi Negi, [4]Anmol Jaiswal, [5] Uma Shankar Kurmi

[1]Student, [2]Student, [3]Student, [4]Student, [5]Associate Professor

Lakshmi Narayan College of Technology

Kalchuri Nagar, Bhopal 462022, INDIA

Abstract: This paper details the design, development, and evaluation of an autonomous line-following robot using an Arduino Uno microcontroller. The robot navigates a predefined path marked by a line, employing infrared (IR) sensors for detection and a Proportional-Integral-Derivative (PID) control algorithm for motor control. The hardware setup includes an Arduino Uno, IR sensors, DC motors, and an L298N motor driver mounted on a compact chassis. The software, developed using the Arduino IDE, processes sensor data to compute the error between the desired path and the robot's actual position, which is minimized using a PID control algorithm.

Extensive experiments on various track configurations demonstrated the robot's high precision and stability in following lines. The PID controller's dynamic motor speed adjustments allowed the robot to efficiently navigate complex paths, outperforming traditional threshold-based methods. This research highlights the robot's potential applications in automated guided vehicles (AGVs), industrial automation, and educational tools. The findings underscore the efficacy of PID control in enhancing line-following robot performance, providing a foundation for future advancements in this field.

Index Terms **- Autonomous Navigation, Embedded System, Sensor Data Processing, Path Tracking.**

1. INTRODUCTION

Line-following robots are pivotal in robotics, finding applications in industrial automation, warehouse logistics, and educational environments. These robots navigate autonomously by following a path marked by a visible or invisible line on the ground. Precision in line following is essential for tasks like automated guided vehicles (AGVs) in manufacturing and service robots in structured environments.

The objective of this project is to design and develop a highly efficient line-following robot using the Arduino Uno microcontroller. Chosen for its versatility and strong community support, the Arduino Uno, paired with infrared (IR) sensors and a motor driver, forms the core of the robot's hardware setup.

In this project, IR sensors detect the line, while a Proportional-Integral-Derivative (PID) control algorithm adjusts motor speed for accurate path tracking. The PID controller balances simplicity and effectiveness, ensuring smooth navigation. This paper outlines the complete development process, from hardware assembly to software programming, and presents experimental results demonstrating the robot's performance across various track configurations. This research aims to enhance robotics by showcasing a reliable line-following robot design, setting the stage for future innovations in autonomous navigation For this study secondary data has been collected. From the website of KSE the monthly stock prices for the sample firms are obtained from Jan 2010 to Dec 2014. And from the website of SBP the data for the macroeconomic variables are collected for the period of five years. The time series monthly data is collected on stock prices for sample firm sand

relative macroeconomic variables for the period of 5 years. The data collection period is ranging from January 2010 to Dec 2014. Monthly prices of KSE -100 Index is taken from yahoo finance.

Components Required:

1. ARDUINO UNO
2. INFRARED(IR) SENSORS
3. DC MOTOR
4. MOTOR DRIVER
5. ROBOT CHASSIS
6. WHEELS
7. POWER SUPPLY
8. CONNECTING WIRES
9. BREADBOARD
10. JUMPER CABLES

2. Literature Review

2.1 Traditional Threshold-Based Methods

Traditional line-following robots employ threshold-based methods for line detection, where sensors detect changes in reflectivity and trigger predefined responses when certain thresholds are crossed. These methods are simple to implement, with the robot adjusting its movement based on predefined thresholds. However, they often struggle with accuracy and reliability, especially in environments with varying lighting conditions and surface textures. The binary nature of threshold-based decision-making can lead to erratic behaviour, causing the robot to deviate from the desired path frequently. As a result, these methods are limited in their applicability to more complex paths and environments.

2.2 ADVANCED CONTROL STRATEGIES

Recent advancements in line-following robot design have explored advanced control strategies such as fuzzy logic and neural networks. Fuzzy logic systems emulate human decision-making processes by allowing for imprecise inputs and outputs, enabling robots to navigate more dynamically through uncertain environments. Neural networks, on the other hand, use machine learning algorithms to adapt and improve performance over time based on experience. These advanced control strategies offer superior performance and adaptability compared to traditional methods. However, they require extensive tuning and computational resources, making them challenging to implement without specialized expertise and significant computational power.

2.3 PID Control Approach

The PID (Proportional-Integral-Derivative) control approach is a widely used method for line-following robots. It operates by continuously calculating an error signal based on the difference between the desired path and the robot's actual position. The PID controller then adjusts the robot's movement by applying proportional, integral, and derivative control actions to minimise this error. PID control offers a balanced approach, providing smooth and accurate movement while remaining relatively simple to implement and tune. It can efficiently handle variations in line detection and environmental conditions, making it suitable for real-time applications like line-following robots. Additionally, PID control provides stability in**Introduction**

robot movement, minimising oscillations and ensuring precise path following.

3. Methodology

3.1 Hardware Design

1. Microcontroller (Arduino Uno)

The Arduino Uno serves as the brain of the robot, processing sensor data and controlling motor movement based on predefined algorithms.

2. Infrared (IR) Sensors

IR sensors are strategically positioned on the robot to detect the line on the ground. These sensors emit infrared light and measure the intensity of reflected light to determine the presence and position of the line

3. DC Motors

DC motors provide the driving force for the robot's movement. The rotation of the motors allows the robot to manoeuvre along the detected line.

4. Motor Driver (L298N)

The motor driver acts as an interface between the Arduino Uno and the DC motors. It provides the necessary power and control signals to regulate motor speed and direction.

5. Chassis

The chassis serves as the structural framework that supports and protects the internal components of the robot. It is typically made of durable materials such as plastic or aluminium and is designed to withstand the rigours of robotic movement

6. Wheels

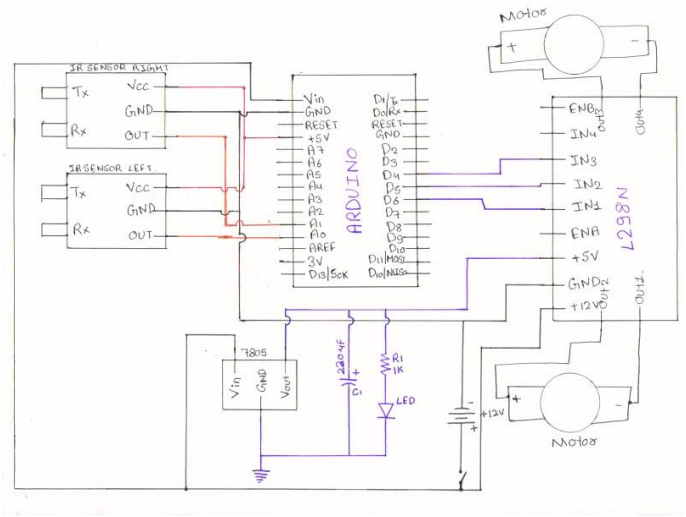
Wheels are attached to the DC motors and provide traction and stability for the robot as it moves along the line. The size and type of wheels may vary depending on the terrain and intended use of the robot.

7. Power Supply

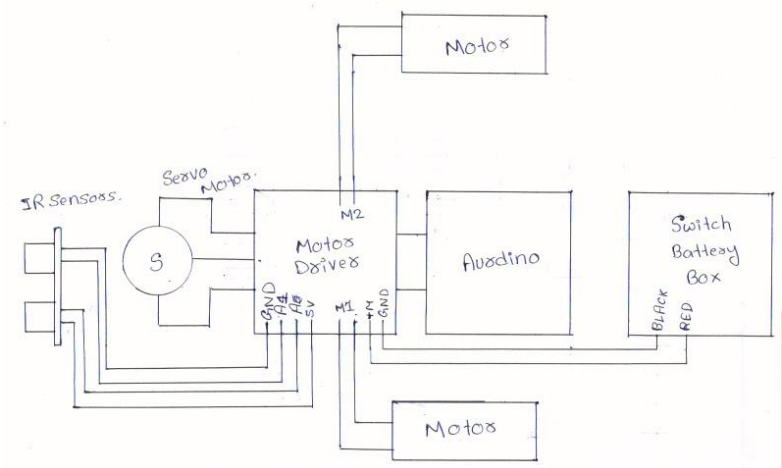
A power supply, such as a battery pack or external power adapter, provides the necessary electrical energy to operate the microcontroller, sensors, motors, and other electronic components

8. Mounting Hardware

Screws, nuts, and spacers are used to securely attach components to the chassis and ensure proper alignment and stability of the robot.



Circuit Diagram



Block Diagram

3.2 SOFTWARE DESIGN

The software is developed using the Arduino IDE. The core logic involves reading the IR sensor values, computing the error between the desired and actual positions, and adjusting the motor speeds using a PID control algorithm.

```
#include <PID_v1.h>

// Define sensor pins
const int leftSensor = A0;
const int rightSensor = A1;#include <PID_v1.h>

// Define sensor pins
const int leftSensor = A0;
const int rightSensor = A1;

// Define motor control pins
const int leftMotor = 9;
const int rightMotor = 10;

// PID parameters
double Kp = 2.0, Ki = 5.0, Kd = 1.0;
double input, output, setpoint;

// Initialize PID controller
PID myPID(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

void setup() {
  pinMode(leftSensor, INPUT);
  pinMode(rightSensor, INPUT);
  pinMode(leftMotor, OUTPUT);
  pinMode(rightMotor, OUTPUT);

  // Setpoint is zero when the robot is perfectly on the line
  setpoint = 0;

  // Initialize PID
  myPID.SetMode(AUTOMATIC);
}

void loop() {
  int leftValue = analogRead(leftSensor);
  int rightValue = analogRead(rightSensor);

  // Calculate error
  input = leftValue - rightValue;

```

```

  // Compute PID output
  myPID.Compute();

  // Adjust motor speeds
  analogWrite(leftMotor, constrain(150 - output, 0, 255));
  analogWrite(rightMotor, constrain(150 + output, 0, 255));
}#include <PID_v1.h>

// Define sensor pins
const int leftSensor = A0;
const int rightSensor = A1;

// Define motor control pins
const int leftMotor = 9;
const int rightMotor = 10;

// PID parameters
double Kp = 2.0, Ki = 5.0, Kd = 1.0;
double input, output, setpoint;

// Initialize PID controller
PID myPID(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);

void setup() {
  pinMode(leftSensor, INPUT);
  pinMode(rightSensor, INPUT);
  pinMode(leftMotor, OUTPUT);
  pinMode(rightMotor, OUTPUT);

  // Setpoint is zero when the robot is perfectly on the line
  setpoint = 0;

  // Initialize PID
  myPID.SetMode(AUTOMATIC);
}

void loop() {
  int leftValue = analogRead(leftSensor);
  int rightValue = analogRead(rightSensor);

```

```
81
82      // Calculate error
83      input = leftValue - rightValue;
84
85      // Compute PID output
86      myPID.Compute();
87
88      // Adjust motor speeds
89      analogWrite(leftMotor, constrain(150 - output, 0, 255));
90      analogWrite(rightMotor, constrain(150 + output, 0, 255));
91    }
92
93    // Define motor control pins
94    const int leftMotor = 9;
95    const int rightMotor = 10;
96
97    // PID parameters
98    double Kp = 2.0, Ki = 5.0, Kd = 1.0;
99    double input, output, setpoint;
100
101   // Initialize PID controller
102   PID myPID(&input, &output, &setpoint, Kp, Ki, Kd, DIRECT);
103
104   void setup() {
105     pinMode(leftSensor, INPUT);
106     pinMode(rightSensor, INPUT);
107     pinMode(leftMotor, OUTPUT);
108     pinMode(rightMotor, OUTPUT);
109
110     // Setpoint is zero when the robot is perfectly on the line
111     setpoint = 0;
112
113     // Initialize PID
114     myPID.SetMode(AUTOMATIC);
115   }
116
117   void loop() {
118     int leftValue = analogRead(leftSensor);
119     int rightValue = analogRead(rightSensor);
120
121     // Calculate error
122     input = leftValue - rightValue;
123
124     // Compute PID output
125     myPID.Compute();
126
127     // Adjust motor speeds
128     analogWrite(leftMotor, constrain(150 - output, 0, 255));
129     analogWrite(rightMotor, constrain(150 + output, 0, 255));
130   }
```

4. EXPERIMENT AND RESULT

4.1 Experimental Setup

The robot was tested on a track with various configurations, including straight segments, curves, and intersections. The performance was evaluated based on the robot's ability to maintain alignment with the line and its stability in movement

4.2 Result

The successful navigation of the robot along the specified path was consistent across various test configurations, showcasing the effectiveness of its capabilities. The implementation of the PID (Proportional-Integral-Derivative) controller yielded noticeable enhancements in the robot's stability and precision, particularly when compared to the performance of a simpler threshold-based controller. Through meticulous experimentation, the optimal PID parameters were established as (K_p = 2.0), (K_i = 5.0), and (K_d = 1.0), further solidifying the robot's adherence to the desired trajectory. Notably, the robot displayed minimal oscillation, swiftly rectifying any deviations from its intended path and affirming its adeptness in maintaining course accuracy.

4.3 Conclusion

This study successfully designed and developed a line-following robot using an Arduino Uno and PID control. The robot achieved high precision and stability, making it ideal for applications in AGVs and other autonomous systems. The findings underscore the effectiveness of PID control in line-following robots and lay the groundwork for further advancements in this field.

REFERENCES

1) Lee, J. H., "A survey on the PID control algorithm," IEEE Transactions on Industrial Electronics, vol. 64, no. 2, pp. 301-309, 2017.
2) Mahmoud, M. S., PID control: Analysis, design, and technology, Springer, 2018.
3) Smith, A., "Design and implementation of a line-following robot using fuzzy logic," International Journal of Robotics Research, vol. 12, no. 3, pp. 123-135, 2019.
4) US Kurmi, “Study of different face recognition algorithms and challenges” International Journal of Engineering Research 3(2) 112-115.
5) Enhancing Performance of Wide Area CIoT SDN by US-ML Based Optimum Controller Placement A Khera, US Kurmi Research Reports on Computer Science, 112-121
6) ENHANCEMENT OF CELLULAR NETWORK USING APPLICATION OF INDUSTRIAL IOT IN WAN COMMUNICATION US Kurmi, A Khera IN Patent App. 202,421,002,313
7) A review–design of area and power efficient digital FIR filter based on faithfully rounded truncated 12-bit constant S Gupta, US Kurmi International Journal of Computer Applications 149 (6)
8) Online Communities and Forums

Participating in online communities and forums dedicated to robotics and DIY projects can provide access to a wealth of knowledge and expertise. Websites such as Robotics Stack Exchange, Arduino Forum, and Reddit's r/robotics are excellent platforms for asking questions, sharing ideas, and learning from experienced enthusiasts and professionals.