SPECTULATION DOCUMENT

Line following Robot using PID Controller

AIM:

The aim of this project is to design and build a line following robot that utilizes a PID (Proportional-Integral-Derivative) controller to accurately track and follow a line. The robot will be able to navigate autonomously along a predefined path by detecting and analysing the line using sensors and adjusting its movement based on the PID control algorithm.

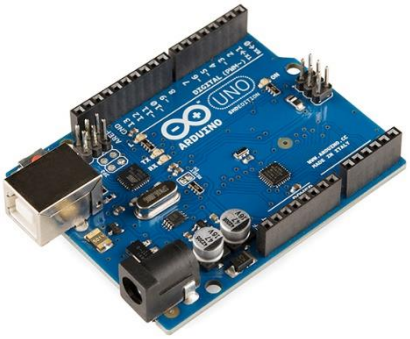
Setting Up Arduino IDE:

Arduino IDE Setup:

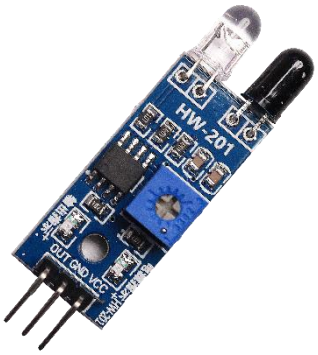
1. Go to this link https://www.arduino.cc/en/Main/Software and download Arduino IDE according to your system.
2. Extract the zip file.
3. Open the extracted folder and run Arduino.exe.
4. Your Arduino IDE is ready to use.

COMPONENTS REQUIRED:

1)Arduino UNO



2) 5 x IR sensor

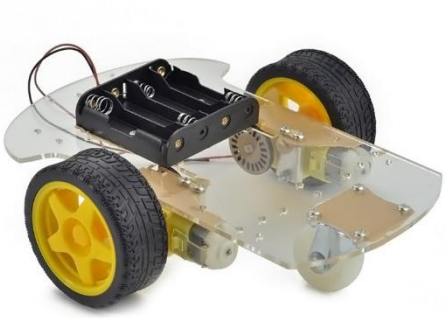


3)L298N Motor driver module

4) DC motor

5) Wheels

6) Car chassis

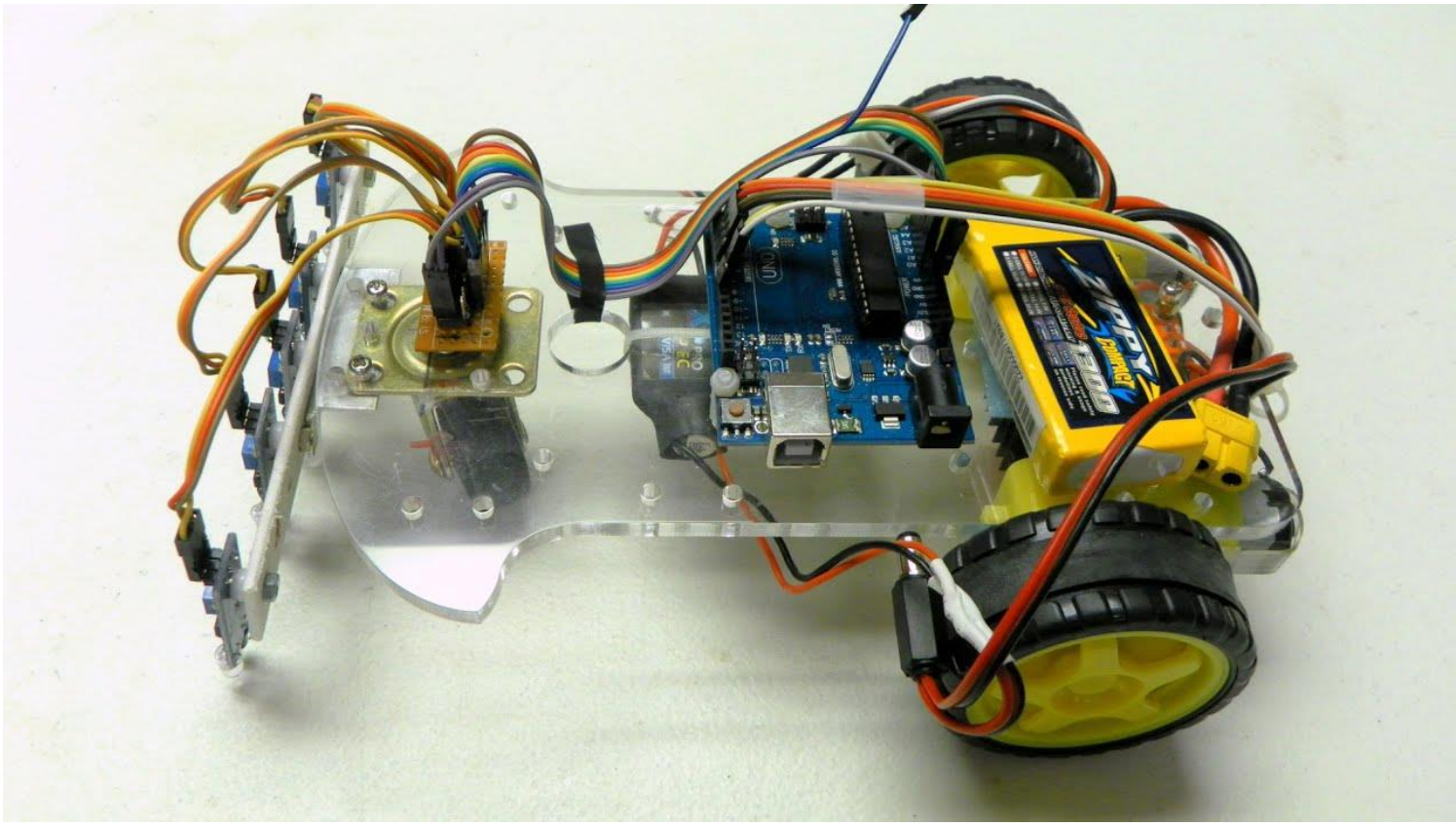


7) Battery

THEORY:

The line following robot will use infrared (IR) sensors to detect the line on the ground. These sensors will generate analog signals based on the reflection of infrared light from the surface. The PID controller will receive these sensor readings as input and compute the appropriate control signals to guide the robot along the line. The PID algorithm combines proportional, integral, and derivative terms to provide precise control and minimize errors in following the line.

The proportional term adjusts the robot's steering in proportion to the difference between the current position and the desired position on the line. The integral term accounts for accumulated errors over time and helps eliminate steady-state errors. The derivative term predicts future errors based on the rate of change of the current error and helps in stabilizing the robot's movement.



Next, we will see the implementation of our code:

```
// Line Following Robot using PID Control with L298N Motor Driver

// Pin connections
const int leftMotorPin1 = 2;
const int leftMotorPin2 = 3;
const int rightMotorPin1 = 4;
const int rightMotorPin2 = 5;
const int irSensorPins[] = {A0, A1, A2, A3, A4}; // IR sensor analog input pins

// PID Constants
const float Kp = 1.0;  // Proportional constant
```

```
const float Ki = 0.1;  // Integral constant
const float Kd = 0.05; // Derivative constant

// Setpoint (desired position on the line)
const int setpoint = 500;

// Variables
int sensorValues[5];
float error, lastError, totalError, correction;

void setup() {
  // Initialize motor control pins as output
  pinMode(leftMotorPin1, OUTPUT);
  pinMode(leftMotorPin2, OUTPUT);
  pinMode(rightMotorPin1, OUTPUT);
  pinMode(rightMotorPin2, OUTPUT);

  // Start serial communication for debugging
  Serial.begin(9600);
}

void loop() {
  // Read sensor values
  for (int i = 0; i < 5; i++) {
    sensorValues[i] = analogRead(irSensorPins[i]);
  }

  // Calculate error
  int weightedSum = 0;
  int totalWeight = 0;
  for (int i = 0; i < 5; i++) {
    weightedSum += sensorValues[i] * (i - 2);
    totalWeight += sensorValues[i];
  }
  error = setpoint - weightedSum / totalWeight;

  // PID calculations
  correction = Kp * error + Ki * totalError + Kd * (error - lastError);

  // Update last error and total error
  lastError = error;
```

```
totalError += error;

// Motor control
int leftSpeed = 200;  // Base speed for left motor
int rightSpeed = 200; // Base speed for right motor

// Apply correction to motor speeds
leftSpeed += correction;
rightSpeed -= correction;

// Limit motor speeds to avoid exceeding maximum values
leftSpeed = constrain(leftSpeed, 0, 255);
rightSpeed = constrain(rightSpeed, 0, 255);

// Set motor directions
if (leftSpeed > 0) {
  digitalWrite(leftMotorPin1, HIGH);
  digitalWrite(leftMotorPin2, LOW);
} else {
  digitalWrite(leftMotorPin1, LOW);
  digitalWrite(leftMotorPin2, HIGH);
}

if (rightSpeed > 0) {
  digitalWrite(rightMotorPin1, HIGH);
  digitalWrite(rightMotorPin2, LOW);
} else {
  digitalWrite(rightMotorPin1, LOW);
  digitalWrite(rightMotorPin2, HIGH);
}

// Set motor speeds
analogWrite(leftMotorPin1, abs(leftSpeed));
analogWrite(rightMotorPin1, abs(rightSpeed));

// Print sensor values and correction for debugging
for (int i = 0; i < 5; i++) {
  Serial.print("Sensor ");
  Serial.print(i);
  Serial.print(": ");
  Serial.print(sensorValues[i]);
```

```
  Serial.print(" | ");
 }
 Serial.print("Correction: ");
 Serial.println(correction);

 delay(10);
```

After the connections, as we upload the code from the Arduino IDE to the Arduino board, we will now see the working principle of the code.

1. The robot is equipped with five IR sensors placed in a line across its width. These sensors detect the reflected infrared light from the surface.

2. The sensor values are used to calculate the weighted sum and the total weight. The weighted sum takes into account the position of each sensor in the line and its corresponding intensity. This calculation provides an estimate of the robot's position with respect to the line.

3. The desired position on the line, called the setpoint, is predefined. The error is then computed as the difference between the setpoint and the estimated position calculated in the previous step.

4. Using the PID control algorithm, the error is processed to generate a correction value. The PID algorithm combines proportional, integral, and derivative terms to adjust the robot's movement and minimize tracking errors.

5. The correction value is applied to the base speed of the left and right motors. The correction can increase or decrease the motor speeds depending on the error. This adjustment allows the robot to steer and follow the line accurately.

6. The adjusted motor speeds are applied to the motor driver connected to the respective motor control pins. The motor driver amplifies and regulates the control signals to drive the motors.

7. The robot continues to read the sensor values, calculate the error, and adjust the motor speeds based on the PID control algorithm in a continuous loop.

8. The sensor values, along with the calculated correction, can be printed or monitored for debugging purposes. This allows for fine-tuning of the PID constants or troubleshooting if needed.

9. The line following robot moves along the line, making real-time adjustments to its motor speeds based on the line position detected by the IR sensors and the PID control algorithm. It continuously strives to minimize tracking errors and maintain its position on the line.

Challenges Faced:

1) Line Contrast and Width: I struggled with detecting lines that had low contrast or were too thin. It was important to optimize sensor placement, adjust sensor sensitivity, or employ line enhancement techniques to overcome this challenge.
2) PID Tuning: Finding the right values for the PID constants (Kp, Ki, Kd) proved to be a challenging task. Incorrect values led to issues like overshooting, instability, or slow response. I had to perform iterative tuning and experimentation to strike a balance between responsiveness and stability.

Overcoming these challenges demanded a combination of careful design, sensor calibration, algorithm optimization, and iterative testing. Each challenge provided valuable learning experiences and opportunities for improvement, contributing to the successful development of a robust line following robot.