Vidyalankar Institute of Technology

Electronics and Computer Science Department

P.I.D Line follower bot
Project Report

November 2025

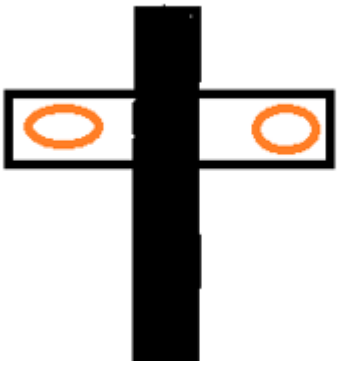
Prepared by:

Shubham Ralkar
TE EXCS A

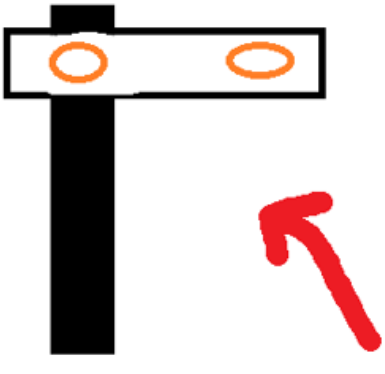
Index

Section 1: Working of Basic line follower

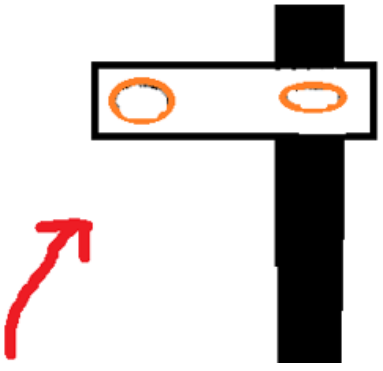
A simple line follower bot, having only two I.R sensors works on very simple feedback control. This control is successful at keeping the bot on track most of the yimes with some disturbances.



The above figure displays the ideal situation in which a line follower bot must be. Both sensors on white surface and black surface on middle.



Here, one of the two sensors is on the black line. So the bot orders the motor on opposite motor to spin at higher rate, Which corrects the bot, trajectory and brings it in ideal state.



Similarly, when right sensor goes on black line, the motor on left is spun faster than right motor so as to bring bot in idea state.

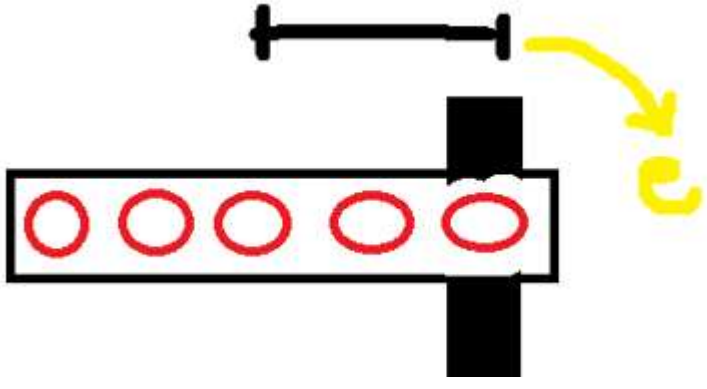
This is how the basic line follower with two sensors work. From next section onwards we are going to se how a l.f.r with multiple sensors work.

Section 2: Adding more sensors to LFR

From previous section we are clear that, The LFR works by correcting its direction in which it moves, so that it stays on the line.

This happens by calculating which sensor is on line (is the left or right) and then giving necessary signals to the motors.

When we tend to add more sensors, we tend to get a scalar, meaning which sensor/s is/are on line and thus a ‘error’ which must be corrected. As illustrated below



Here, the error is of distance between two sensors. (error is just distance from ideal case and current case which must be correcyed to keep the bot on track)

Section 3: What is P.I.D?

P.I.D algorithm which expands to Proportional Integral Derivative, refers to a algorithm that basically does is **"Pushes the system to the desired state, by optimally providing correcting signal, modifying those correcting signals, reducing the overshoots and oscillations, thus stabilising the system at desired setpoint."**

This was the idea which PID works on. To expand over each term in the algorithm. What each term does is:

1. Proportional term: It provides a correcting signal proportional to error we have.

```
P = Kp*error
```

2. Integral term: It is very essential in stabilsing the system at desired setpoint, without it, the system can be stabilsed but at different setpoint, thus it is used to elimate the steady state error.

```
I = (I + error)*Ki
```

3. Derivative term: It is used to reduce the correcting signal as the system nears the desired setpoint. Otherwise this term, the system may overshoot, not become stabilising and just oscillating.

```
D = Kd*(current_error- prev_error)
```

```
Correction signal =
Kp*error + (I + Ki)*error + Kd*(current_error-
prev_error)
```

Section 4: P.I.D in Line follower

In section 2, we came across the error term which should be eliminated to keep the bot on the line, P.I.D algorithm application plays a very pivotal role here in generating the correct signals to corresponding motors, so as to keep the bot on the line.

Here is the explanation of how the P.I.D is implemented block by block in case of line follower bot.

Block I:

```
/**
 * PID Line Follower for Arduino
 * Uses Cytron 5 IR array sensor and TB6612FNG motor driver
 */

// Pin definitions for TB6612FNG motor driver
// Motor A - Left Motor
#define AIN1 5
#define AIN2 4
#define PWMA 3

// Motor B - Right Motor
#define BIN1 7
#define BIN2 6
#define PWMB 8

// Pin definitions for Cytron 5 IR array sensor (digital inputs)
#define IR1 18  // Leftmost sensor
#define IR2 19  // Left sensor
#define IR3 20  // Middle sensor
#define IR4 21  // Right sensor
#define IR5 22  // Rightmost sensor

// PID Constants - adjust these based on your robot's characteristics
#define KP 30  // Proportional gain   prev=28
#define KI 0   // Integral gain
#define KD 40  // Derivative gain      prev= 25
```

defines the connections and Kp,Ki and Kd values.

Block II:

```
// Motor speed constants
#define BASE_SPEED 80    // Base speed (0-255)
#define MAX_SPEED 255     // Maximum speed (0-255)

// Variables for PID calculation
float lastError = 0;
float integral = 0;
```

Additional terms used in PID calclations initialised.

Block III:

```
void setup() {
  // Initialize serial communication
  Serial.begin(9600);

  // Set motor driver pins as outputs
  pinMode(AIN1, OUTPUT);
  pinMode(AIN2, OUTPUT);
  pinMode(PWMA, OUTPUT);
  pinMode(BIN1, OUTPUT);
  pinMode(BIN2, OUTPUT);
  pinMode(PWMB, OUTPUT);

  // Set IR sensor pins as inputs
  pinMode(IR1, INPUT);
  pinMode(IR2, INPUT);
  pinMode(IR3, INPUT);
  pinMode(IR4, INPUT);
  pinMode(IR5, INPUT);

  // Wait for 2 seconds before starting
  Serial.println("PID Line Follower Starting...");
  delay(2000);
}
```

Definition of I/O pin connections, used for getting signals from sensor and giving command to motors.

Block IV:

```
void readSensors(int sensorValues[]) {
  // Read the IR sensors (0 for white, 1 for black)
  sensorValues[0] = digitalRead(IR1);
  sensorValues[1] = digitalRead(IR2);
  sensorValues[2] = digitalRead(IR3);
  sensorValues[3] = digitalRead(IR4);
  sensorValues[4] = digitalRead(IR5);
}

float calculateError(int sensorValues[]) {
  float position = 0;
  int sensorsSum = sensorValues[0] + sensorValues[1] + sensorValues[2] + sensorValues[3] + sensorValues[4];

  // If all sensors read white (0), maintain last error
  if (sensorsSum == 0) {
    return lastError;
  }

  // Calculate weighted position
  float weightedSum = (sensorValues[0] * -3.5 +
                       sensorValues[1] * -1.5 +
                       sensorValues[2] * 0.0 +
                       sensorValues[3] * 1.5 +
                       sensorValues[4] * 3.5);

  // Calculate normalized position
  position = weightedSum / sensorsSum;

  return position;
}
```

This block, gets input signal from sensors and calculates an error value based on placement of sensor on the line at the instant.

Block V:

This block sets the motor speeds, that is sends command to motor driver by PWM pulses to provide speed instructions to the motors.

```
void setMotors(int leftSpeed, int rightSpeed) {
  // Constrain motor speeds
  leftSpeed = constrain(leftSpeed, -MAX_SPEED, MAX_SPEED);
  rightSpeed = constrain(rightSpeed, -MAX_SPEED, MAX_SPEED);

  // Set left motor direction and speed
  if (leftSpeed >= 0) {
    digitalWrite(AIN1, HIGH);  // Forward
    digitalWrite(AIN2, LOW);
    analogWrite(PWMA, leftSpeed);
  } else {
    digitalWrite(AIN1, LOW);   // Reverse
    digitalWrite(AIN2, HIGH);
    analogWrite(PWMA, -leftSpeed);
  }

  // Set right motor direction and speed
  if (rightSpeed >= 0) {
    digitalWrite(BIN1, HIGH);  // Forward
    digitalWrite(BIN2, LOW);
    analogWrite(PWMB, rightSpeed);
  } else {
    digitalWrite(BIN1, LOW);   // Reverse
    digitalWrite(BIN2, HIGH);
    analogWrite(PWMB, -rightSpeed);
  }
}
```

Block VI:

```
void loop() {
 // Read sensor values
 int sensorValues[5];
 readSensors(sensorValues);

 // Calculate error and apply PID control
 float error = calculateError(sensorValues);

 // Calculate PID components
 float proportional = error;
 integral += error;
 float derivative = error - lastError;

 // Apply anti-windup to integral term
 integral = constrain(integral, -100, 100);

 // Calculate PID output
 float pidOutput = (KP * proportional) + (KI * integral) + (KD * derivative);

 // Update lastError for next iteration
 lastError = error;

 // Calculate motor speeds
 int leftSpeed = BASE_SPEED - pidOutput;
 int rightSpeed = BASE_SPEED + pidOutput;

 // Set motor speeds
 setMotors(leftSpeed, rightSpeed);
```

This is the main block, Which controls the bot by calling all the functions described above and running them for as frequent as 4 million times per second! (case for Atmeg 328p, using delay() can reduce this speed).

Section 5: Components used

1. Cytron IR array sensor: Contains 5 ir arrays and provides the output in analog as well as digital values.

2. TB6612FNG motor driver: Can manipulate speed of motor based on PWM signals provided. Can handle max of 3.2 A of peak current per channel.

3. Pico microcontroller: RP2040 based dual core Cortex M0+ microcontroller.

4. N20 Motors: low torque, high rpm motors, critical for speed.

5. 7.4v 850mah lipo battery: Very essential for delivering high currents to motors

5. Other miscellaneous components:
7805 voltage regulator, switches, castor ball, wires, motor brackets ,capacitors,etc.