Indian Institute of Technology, Madras

A Report on

Design and Development of a 3-wheeled mini-AGV

Submitted by

Kiruthees G	ME23B246
Selvakkumaran S V	ME23B199
T. Ashish Kumar Reddy	ME23B241
T Pruthvi Raj	ME23B207
Deviprasad V K	ME23B164

Submitted to

Dr. Charchit Kumar

Assistant Professor in the Department of Mechanical Engineering

Indian Institute of Technology, Madras

LINE FOLLOWER

The line follower robot is a self navigating car driven by using infrared sensors for path detection and dc batteries for local power supply.

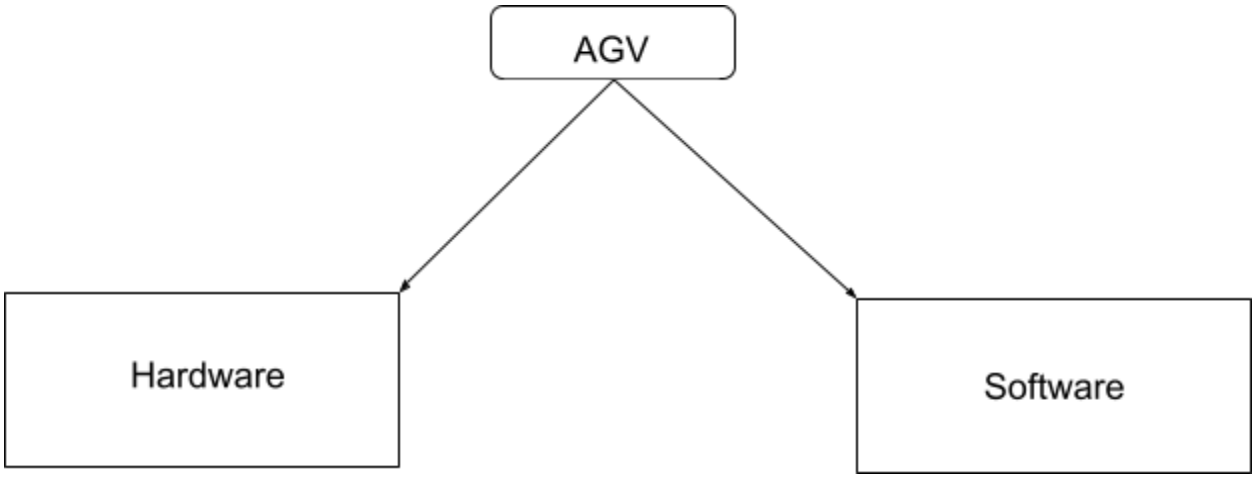
PLANNING PROCESS:

We had a group discussion and did some research on line follower robots using numerous sources from youtube and the internet.

The two reference videos from youtube are

1. https://youtu.be/QoNkpnpvEqc?si=9iK-v7xmn5zC5BiH
2. https://youtu.be/88nKA6nWmqw?si=epJvhtzjFkABgPqj

After our brainstorming session together we divided the project into 2 divisions,



From our research, we found that the main hardware components were:

a. Motors (DC)

b. Microcontroller

c. Motor driver

d. Batteries

e. Infrared Sensors

f. Wheels

So far, we knew what kind of components we needed, but we were yet to figure out what model/specifications each of these needed to have.

MICROCONTROLLER:

From our team nobody had any prior electronics knowledge.

The microcontroller is the integration of a microprocessor with memory and input/output interfaces, and other peripherals such as timers, on a single chip.

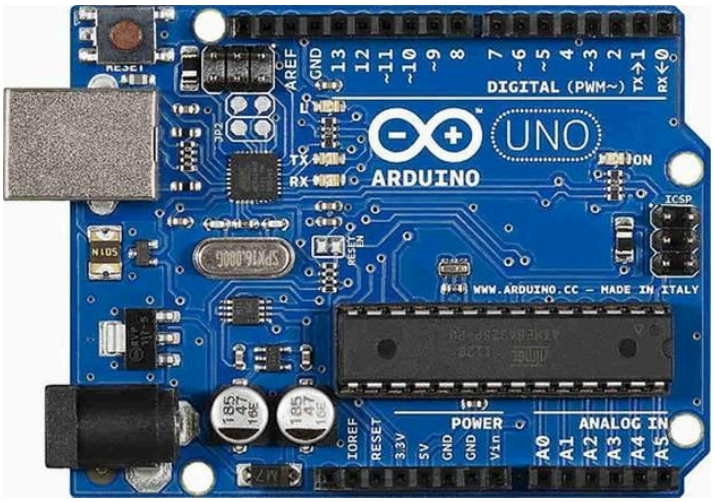
From our research we found some microcontrollers like Arduino, Raspberry Pi, ESP32.

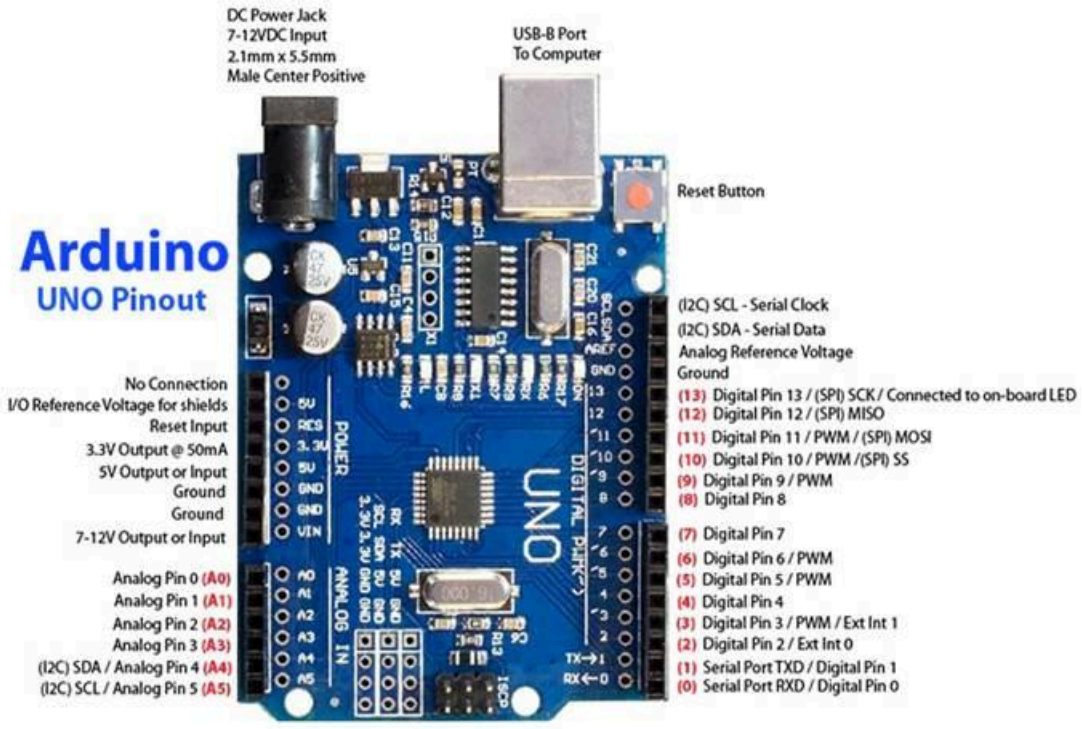
We didn’t really have any solid idea as to which one to use and which would be more compatible with the other components we were going to use. So we finally decided to use Arduino because many of the projects we came across used them.

In Arduino, we decided to use the **uno**, even though many other versions like the Arduino Mega, Arduino nano were all available.

We came to this consideration after looking into factors like:

- Amount of current it would need to handle
- No. of slots available for the required connections with other components.
- The size
- The cost





We went online to buy Arduino Uno, but the original one was too expensive.

So after consulting with some of our electronics friends we decided to get an inauthentic cheaper version of it after confirming it has almost no major downsides compared to the original one, so we could cut down on cost without compromising on performance.

MOTORS:

We wanted motors which would have a high RPM. Some of the motors we came across were mini servo motors, Brushless DC motors. The main factors we looked into were:

- RPM
- Size
- Voltage needed

After all these considerations, we finally decided to use N20 DC motors. Its features include:

a. 300 RPM

b. Small size and lightweight

c. 12V maximum operating voltage

WHEELS:

After choosing the motor we needed, we were able to find online the wheels which are custom made for N20 DC motors.

The axle of the motor fits perfectly into the wheels so the torque can be transmitted perfectly to the wheel without any slippage between the axle and the wheel.

The wheel also had rubber tyres with grooves for a good grip onto the ground. We used 2 of these in the back and a castor wheel in the front center.

The castor wheel we ordered had a plastic wheel and stainless steel body and it had smooth free rotation about the vertical axis due to ball-bearings mechanism.

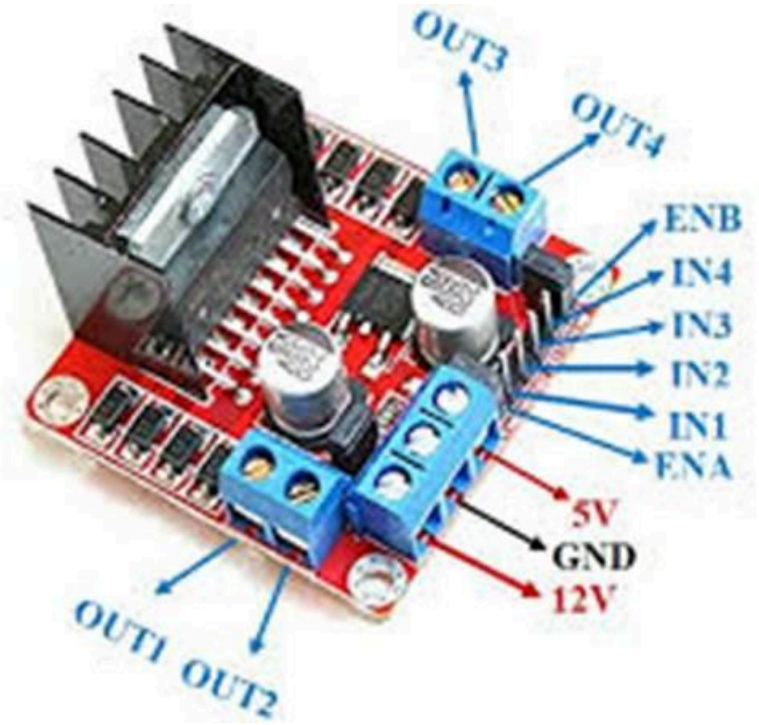
MOTOR DRIVER:

Our main focus was that it should be able to handle the high currents needed for the 2 motors and be able to handle the runtime without overheating.

We decided to use the **L298N** Motor driver. The L298N motor driver can handle a maximum supply voltage of 46V. The operating voltage range is typically 5 to 35V. Additionally, the logic voltage for controlling the driver is typically 5V.

It has PWM channels to control the speed of the motors and 4 logic channels for direction control.

It also has a 5V output channel which can be connected to the Arduino uno which needs 5V operating voltage.



But it has a drawback of overheating over continued use. But since the runtime is going to be low, we decided to go on with it considering the other pros as they outweigh the cons.

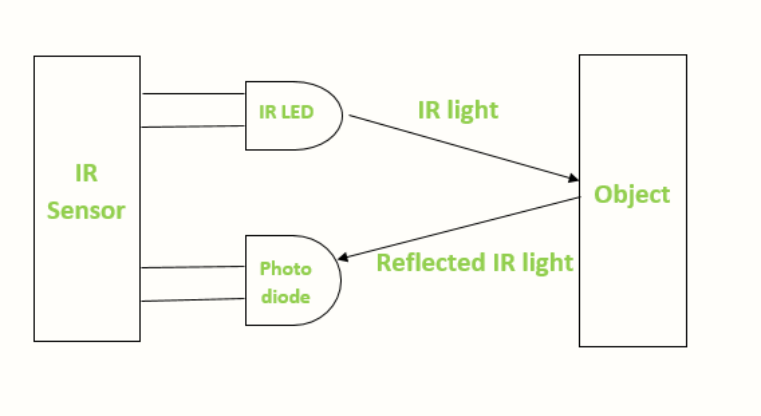
Infrared sensor

IR sensor is an electronic device, that emits the light in order to sense some object of the surroundings(active one to be specific)

An infrared sensor can detect a black line by measuring the amount of reflected infrared light

We decided to go with five infrared sensors so that the vehicle can detect the black line if it is on either of the extreme sides or even in the center.

The output from the IR is analogue with a value of 39 when it detects the black line and more than 700 when it does not detect the line.



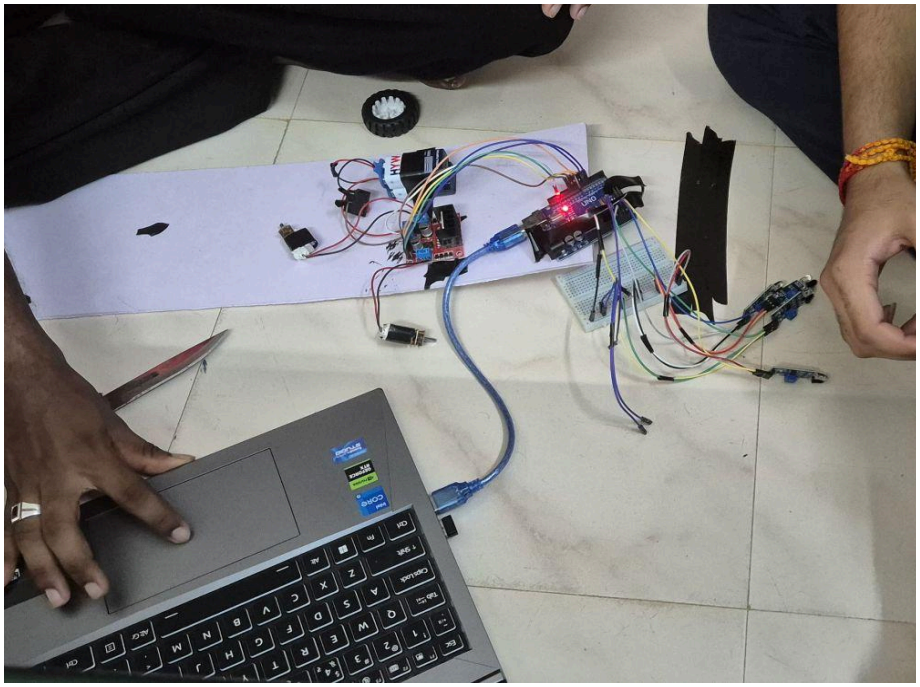
Batteries

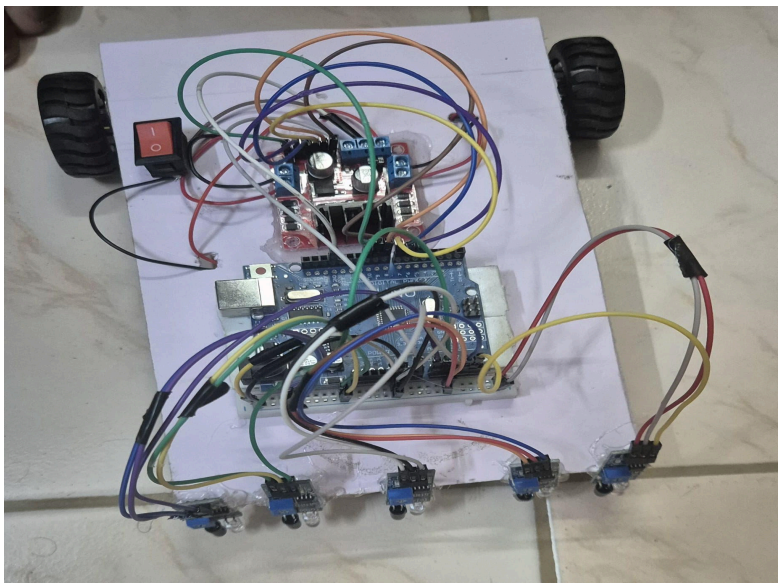
The specifications of the battery depends on the type of motor used, we used a 12 V motor hence we decided to use three 3.7 V rechargeable batteries so we can reuse them a number of times. Ultra fire BRC 18650 lithium ion batteries were used to power the motor each with a capacity 6800 mAh

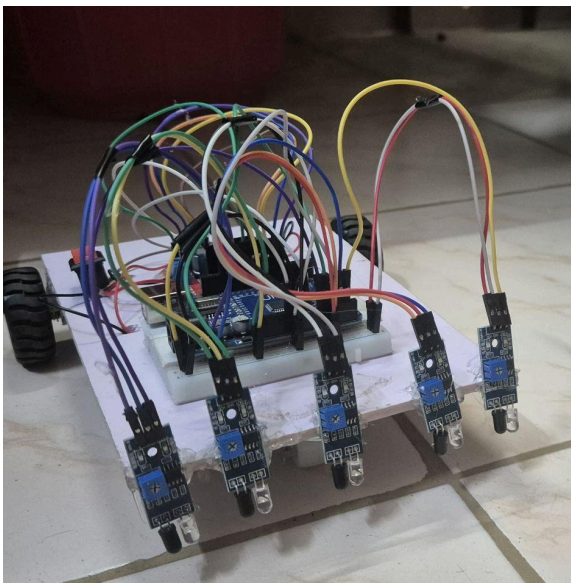
Other hardware components used were:

- Foam board which was borrowed from other group as they had excess of it
- Jumper wires to make connections
- A switch
- A breadboard to connect the IR sensors to the Ardiuno
- Glue Gun is used to integrate all the components on the foam board

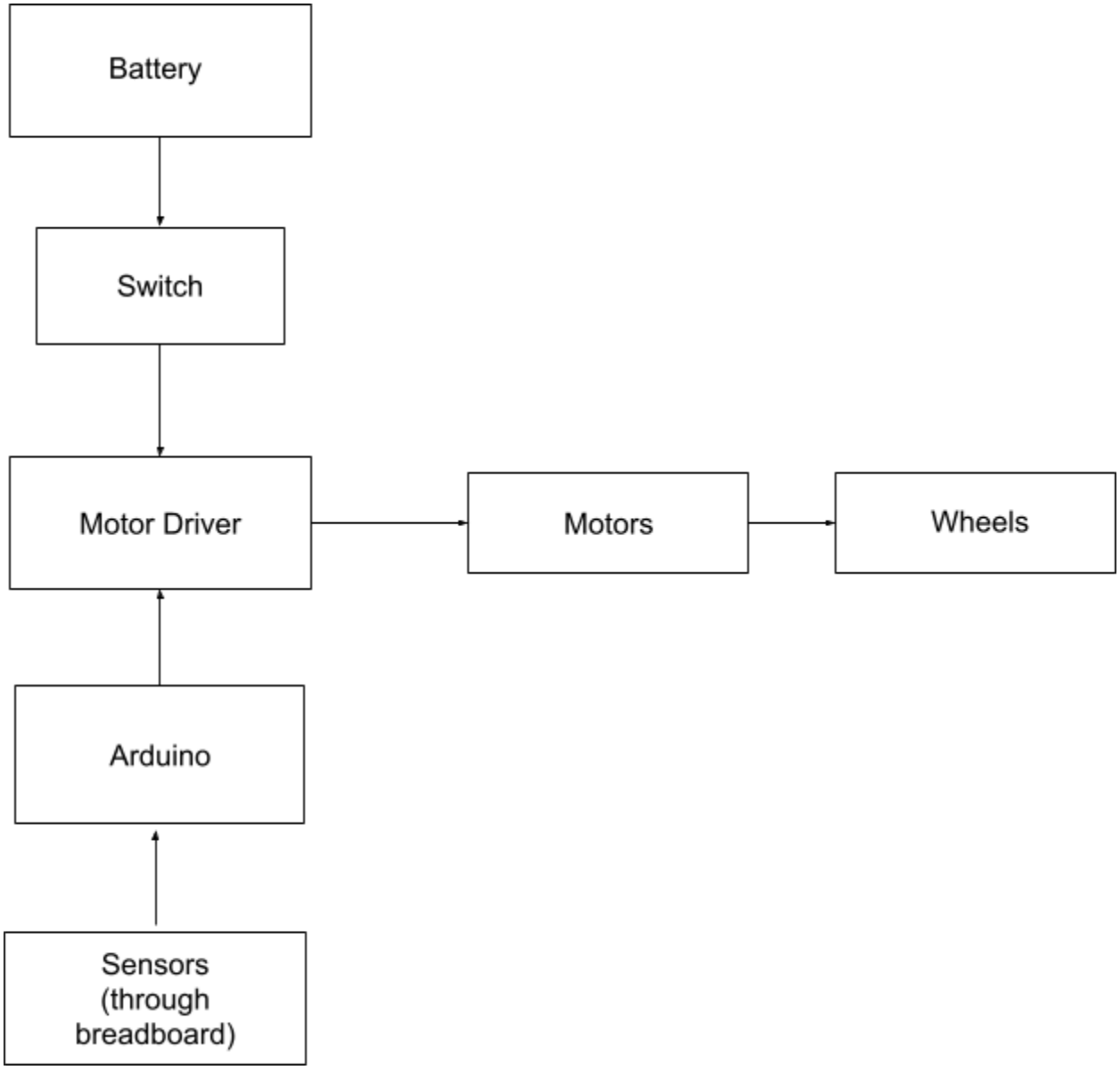
Assembling







Connections



1. Battery to Switch

The battery serves as the primary power source for the vehicle. A switch is connected in-line to manually control the power supply to the system. When the switch is turned on, power flows to the rest of the components.

2. Switch to Motor Driver

The motor driver receives power from the battery via the switch. This module is crucial for controlling the motors, as it can handle the higher current requirements that the Arduino alone cannot.

3. Motor Driver to Motors to Wheels

The motor driver is connected to the DC motors, which are in turn mechanically connected to the wheels. It sends appropriate voltage signals to drive the motors forward, backward, or to turn, based on inputs from the Arduino.

4. Arduino to Motor Driver

The Arduino controls the motor driver by sending digital signals to it. Based on the sensor inputs, it determines which motors should be activated and in what direction, enabling the robot to follow the line.

5. Sensors (Connected via Breadboard) to Arduino

A set of IR sensors is connected to the Arduino through a breadboard. These sensors detect the line on the ground (typically black on white). The Arduino reads this data and makes decisions accordingly to steer the robot.

Overall Flow of Operation

1. Power is supplied from the battery through a switch.
2. Sensors detect the path and send data to the Arduino.
3. The Arduino processes this input and sends control signals to the motor driver.
4. The Motor Driver powers the motors, which drive the wheels, allowing the robot to follow the line.

Testing

1. Checking the output from the sensors

```
const int irSensor1 = A0;
const int irSensor2 = A1;
const int irSensor3 = A2;
const int irSensor4 = A3;
const int irSensor5 = A4;

void setup() {
    // Start Serial Monitor
    Serial.begin(9600);
}

void loop() {
    // Read analog sensor values
    int sensor1 = analogRead(irSensor1);
    int sensor2 = analogRead(irSensor2);
    int sensor3 = analogRead(irSensor3);
    int sensor4 = analogRead(irSensor4);
    int sensor5 = analogRead(irSensor5);

    // Print the sensor values
    Serial.print("IR1: ");
    Serial.print(sensor1);
    Serial.print(" | IR2: ");
    Serial.print(sensor2);
    Serial.print(" | IR3: ");
    Serial.print(sensor3);
    Serial.print(" | IR4: ");
    Serial.print(sensor4);
    Serial.print(" | IR5: ");
    Serial.println(sensor5);
    delay(200); // Small delay for readability
}
```

Once the assembling process is complete, we test whether the sensors are able to detect the black line when they are on top of it by connecting the sensors to a computer and checking their output by running the above code

We found that one of the sensors was not working properly, so we had to replace it with a new one.

2. Moving in a straight line

After calibrating the sensors to measure the black line at the same time, we fixed both the motors at the same speed and let it run on a smooth surface but we encountered a problem, one of the wheels was internally rotating and was not aligned properly, This lead to an undesired steering of the vehicle instead of going in a straight line. Since we used a Glue gun it was not a difficult task to remove the motor. So we had to disassemble one of the motors and reattach in such a way that when the speed of both the motors is set the same it moves in a straight line.

3. Tracking the Black line

Now comes the the logic, since we are using 5 sensors we want the sensor in the centre to stay on top of the black line and if the black line is sensed by this sensor but the other sensor(say right) then the torque in the corresponding wheel(right) is decreased and torque in the opposite motor is increased leading to a steering action in which the vehicle rotates towards the sensor which is sensing the line(right turn)

This steering action is controlled using a PID controller algorithm and setting the rate of change in the torque of the motor depends on 3 factors mainly the Proportional(K_p), Integral(K_i) and Differential gain(K_d).

PID Control Overview

PID control continuously calculates an error value as the difference between a desired setpoint and a measured process variable, and applies a correction based on proportional, integral, and derivative terms.

PID Formula
The general form of the PID control output u(t) is:

$$u(t) = K_p \cdot e(t) + K_i \cdot \int_0^t e(\tau)\, d\tau + K_d \cdot \frac{de(t)}{dt}$$

Where,
- e(t): Error at time t (difference between setpoint and current value)
- K_p: Proportional gain
- K_i : Integral gain z
- K_d: Derivative gain

How Each Term Works :

Proportional Term (K_p * e(t))
This term produces an output value that is proportional to the current error. A larger error results in a stronger correction.

Integral Term (K_i *∫e(t)dt)
This term accounts for the accumulation of past errors. It helps eliminate small steady-state errors that the proportional term

alone cannot correct. In some systems, it may be omitted (i.e., set Ki = 0) to avoid overshooting.

Derivative Term (K_d * de(t)/dt)
This term predicts the future trend of the error by measuring its rate of change. It adds damping and helps reduce overshooting and oscillations.

Application in Line Following Robot
In the robot:
• The error e(t) is calculated as the weighted average position of the detected line relative to the center.
• The correction u(t) is added/subtracted from a baseSpeed to control left and right motor speeds.
• The robot turns left or right depending on the sign and magnitude of the error.

Discrete Implementation in Arduino

In code, the continuous PID equation is approximated in discrete form:

correction = K_p * e + K_i * integral + K_d * (e − previous Error)
leftSpeed = baseSpeed + correction
rightSpeed = baseSpeed − correction

Where,
• e: current error
• previous Error : error from previous loop
• integral : accumulated sum of errors

Tuning PID Parameters

PID gains (K_p, K_i , K_d) must be tuned manually.
For a line-following robot:
• Start with $K_i = 0$ to avoid slow oscillations.
• Increase K_p until the robot responds quickly.
• Add K_d to reduce overshoot and smooth turns.

Conclusion

PID control is a powerful and simple technique for smooth line-following behavior. By adjusting motor speeds based on real-time error calculations, it allows the robot to correct its course continuously and handle curves effectively.

Feedback Mechanism

In control systems, feedback is the mechanism using the output, like the robot's position or motion here, to change commands for input so the system works well. A closed-loop system is made within a line follower robot.

1. Sensing the Line (Input):
   The IR sensors are those which can detect drifting off, and being on the line. For indicating the current position relative to the line, the Arduino receives input from each of these sensors.

2. Error Calculation (Comparison):
   The Arduino serves as one comparator since it compares the actual position to the desired position. It calculates one error from each of the IR sensors that are centered upon the line.

   Error=Desired Position−Actual Position

3. Generating a Correction (Control Signal):
   Based on the error, the Arduino decides how much and in which direction to adjust the motors.
   It sends a corrective signal (PWM or logic high/low) to the motor driver.

4. Correcting the Path (Output):
   The motor driver adjusts the speed/direction of the motors, turning the wheels left or right to bring the robot back on the line.

5. Feedback Loop (New Input):
   As the robot moves, the IR sensors detect its new position. This updated position is again sent to the Arduino, closing the loop.

Code embedded

```
// === Motor Pins ===
const int ENA = 9;      // PWM - Left motor
const int IN1 = 8;
const int IN2 = 7;

const int ENB = 10;     // PWM - Right motor
const int IN3 = 6;
const int IN4 = 5;

// === IR Sensor Pins ===
const int irPins[5] = {A0, A1, A2, A3, A4};
int irValues[5];

// === PID Variables ===
float Kp = 60 ;    // Proportional
float Ki = 0.005;  // Integral
float Kd = 10;    // Derivative

float error = 0, previousError = 0, integral = 0, derivative = 0;
int baseSpeed = 150; // Adjust as needed (0-255)

// === Sensor Weights for Position Calculation ===
int weights[5] = {2.5, 1.5, 0, -1.5, -2.5}; // For position error
calculation

void setup() {
  // Motor pins
  pinMode(ENA, OUTPUT); pinMode(IN1, OUTPUT); pinMode(IN2, OUTPUT);
  pinMode(ENB, OUTPUT); pinMode(IN3, OUTPUT); pinMode(IN4, OUTPUT);

  // IR sensor pins
  for (int i = 0; i < 5; i++) pinMode(irPins[i], INPUT);
```

```
  Serial.begin(9600);
}

void loop() {
  readIRSensors();

  // Calculate weighted position error
  int position = 0;
  int totalActive = 0;
  for (int i = 0; i < 5; i++) {
    if (irValues[i] == 1) {
      position += weights[i];
      totalActive++;
    }
  }

  if (totalActive > 0) {
    error = (float)position / totalActive;
  } else {
    // Line lost: use previous error direction
    if (previousError > 0) error = 2.5;
    else error = -2.5;
  }

  // PID calculations
  integral += error;
  derivative = error - previousError;
  float correction = Kp * error + Ki * integral + Kd * derivative;

  // Motor speeds
  int leftSpeed = baseSpeed + correction;
  int rightSpeed = baseSpeed - correction;

  // Clamp speeds to 0-255
  leftSpeed = constrain(leftSpeed, 0, 255);
  rightSpeed = constrain(rightSpeed, 0, 255);

  // Drive motors
  driveMotors(leftSpeed, rightSpeed);

  previousError = error;
```

```
  //delay(1); // Small delay for stability
}

void readIRSensors() {
  for (int i = 0; i < 5; i++) {
    int sensorReading = analogRead(irPins[i]);

    if (sensorReading < 300) {
      irValues[i] = 0; // WHITE line detected
    } else {
      irValues[i] = 1; // BLACK surface
    }
  }
}



void driveMotors(int leftSpeed, int rightSpeed) {
  // Left motor forward
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENA, leftSpeed);

  // Right motor forward
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  analogWrite(ENB, rightSpeed);
}
```

Code Explanation

1. Introduction

This Arduino program controls a line-following robot that uses five infrared (IR) sensors to detect a black line on a white surface. It applies a PID (Proportional-Integral-Derivative) control algorithm to steer the robot by adjusting the speeds of two DC motors through an L298N motor driver.

2. Hardware Components

- ❖ IR Sensors (x5) – Detect line position.
- ❖ L298N Motor Driver – Drives two DC motors.
- ❖ DC Motors (x2) – Left and right motors for movement.
- ❖ Arduino Uno/Nano – Main controller.
- ❖ Pin Assignments:
- ❖ Left Motor: ENA (PWM), IN1, IN2
- ❖ Right Motor: ENB (PWM), IN3, IN4
- ❖ IR Sensors: Analog pins A0–A4

3. Working Principle

IR Sensor Reading:
analogRead() gets the sensor values.
If value > 300 → black line detected (1), else white (0).

Error Calculation:
Each sensor has a weight: {2.5, 1.5, 0, -1.5, -2.5} from left to right.
The position error is the weighted average of active sensors.
If no line is detected, the robot assumes the last known direction.

PID Controller:
Proportional (P): Reacts to how far the robot is from the center of the line.
Integral (I): Accounts for past errors.
Derivative (D): Reacts to the rate of change of error (adds stability).

- ➢ **Correction** = Kp × error + Ki × integral + Kd × derivative

Motor Control:
Left motor speed = baseSpeed + correction
Right motor speed = baseSpeed - correction
Speeds are clamped between 0–255 using constrain().

Driving:
Motors are always driven forward.
Speed difference causes turns (to follow the line).

4. PID Tuning
The values:
Kp = 120: Strong response to error.
Kd = 40: Smooths quick changes.
Ki = 0.005

These values were tuned based on several parameters such as
Speed of the robot
Sharpness of curves
Sensor accuracy

5. Conclusion
This code enables a robot to autonomously follow a black line using a simple and efficient PID control loop. By dynamically adjusting motor speeds based on real-time sensor feedback, the robot can smoothly follow curves and correct its path when deviations occur. The use of PD (Proportional-Derivative) control ensures a balance between responsiveness and stability.