PIE Mini Project 3:
DC Motor Control

Addison Fisher, Matthew Kwon, Kuhu Jayaswal

October 17, 2025

1 Introduction

The goal of this project was to design and build a two-wheeled, autonomous robot capable of following a tape-based track laid out on the floor. The robot uses infrared (IR) reflectance sensors to detect the contrast between the tape and surrounding surface, allowing it to stay aligned with the path. The system is powered by an Arduino Uno R3 and controlled through a closed-loop feedback algorithm that continuously adjusts motor speeds based on sensor input.

To achieve this, we integrated sensing and control with a mechanical platform consisting of DC gear motors, a motor shield, and a custom chassis. The mechanical design included 3D-printed mounts for the sensors, breadboard, and Arduino to ensure secure placement and modularity. On the electrical side, we soldered and wired four IR sensors to improve tracking precision and reduce oscillation.

A key feature of our implementation is the ability to tune the robot's performance in real time via the Arduino's serial interface. This allows users to adjust motor speeds and control parameters without recompiling or reloading the code, enabling rapid experimentation and optimization. By combining mechanical design, sensor calibration, and software control, our robot successfully navigates a non-circular loop and can be tuned to complete laps as efficiently as possible.

2 Project Breakdown

2.1 System Diagram & Energy Flows

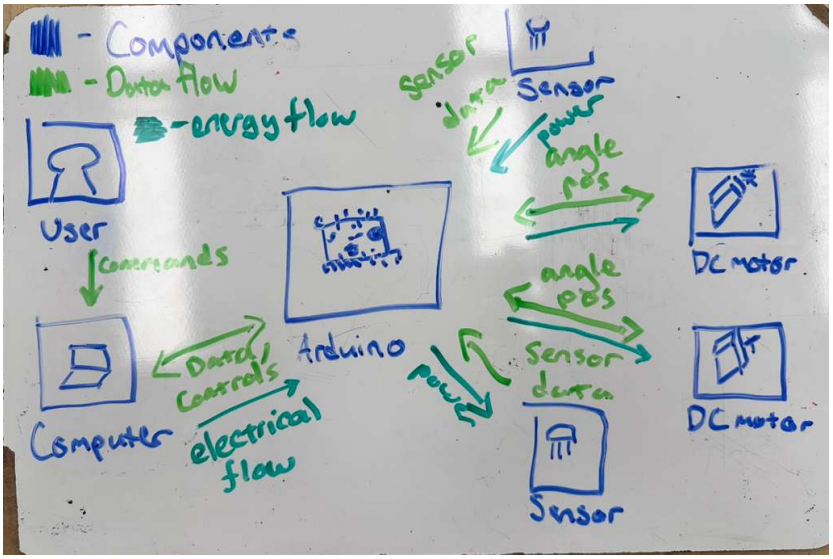

Figure 1: Initial system diagram portraying data and energy flows for this project.

2.2 Mechanical Design

Although we were provided with a basic chassis, it only included mountings for the two DC motors and lacked any structural support for the remaining components. In alignment with the principles of PIE and sound engineering practice, we committed to a no-adhesive fabrication approach to ensure mechanical integrity and modularity. To achieve this, we designed and 3D printed several custom PLA components: a sensor mount capable of holding all four IR sensors, a breadboard mount, and an Arduino mount. These additions allowed for secure placement and easy maintenance. A key design decision involved determining the optimal sensor layout. After collaborative discussions between mechanical and software team members, we chose an arc formation at the front of the robot. This configuration provided consistent spacing and orientation for the sensors, improving line detection accuracy and simplifying software logic.

2.3 Electrical Design

The electrical system was built around four Sharp GP2Y0A02YK0F IR distance sensors, which were used to detect the contrast between tape and floor surfaces. While the original design called for only two sensors, we observed that this configuration led to erratic behavior and poor line tracking. To improve performance, we added two additional sensors, enabling more precise detection of the robot's position relative to the tape. This upgrade significantly reduced oscillations and improved stability during navigation. All sensors were soldered following best practices, including consistent wire color coding and minimizing wire length to reduce noise and clutter. Power was supplied via a regulated 12V source, and motor control was handled by the Adafruit Motor Shield v2.3. The system was assembled on a breadboard for flexibility and ease of debugging.

Hardware Used:

- Arduino Uno R3
- 4 IR reflectance sensors
- 12V power supply
- Adafruit Motor Shield v2.3
- 2 gear motors
- Breadboard and jumper wires

2.4 Software Aspect

The robot's control system was programmed in the Arduino IDE and structured to support modular movement, real-time sensor feedback, and live tuning through serial commands. It continuously reads analog values from four IR reflectance sensors, which are then converted into binary states using a calibrated threshold. In this binary scheme, `1` represents tape detection and `0` represents carpet. These sensor states are used to determine the robot's position relative to the track.

Based on observed patterns during testing, we identified key binary combinations that correspond to specific positional scenarios. For instance, patterns like `1100` or `0011` indicate misalignment and trigger sharp spin corrections, while `0000` signals centered alignment and initiates forward motion. Each case is handled by a dedicated movement function that adjusts motor speeds relative to a tunable base value, allowing the robot to respond dynamically to its environment.

To support debugging and performance optimization, the Serial Monitor outputs a six-value stream: four raw sensor readings followed by the current left and right motor speeds. This live data helped us visualize sensor behavior before thresholding and monitor motor responses for plotting and analysis. Additionally, we implemented a simple command interface—typing `set base {value}` into the Serial Monitor updates the base motor speed instantly, enabling quick tuning without the need to recompile or reload the code.

2.5 Sensor Calibration

To calibrate the IR sensors, we followed a structured process informed by the datasheet for the Sharp GP2Y0A02YK0F. Our objective was to select a pull-down resistor that would allow the sensor output to span the full 0–1023 range of the Arduino's analog-to-digital converter. We began by calculating a theoretical value of 4.5kΩ, which produced readings up to approximately 500. While this was functional, it did not provide sufficient resolution to reliably distinguish tape from carpet.

We then tested an 8kΩ resistor, which extended the readings to around 800, improving contrast but still leaving some ambiguity. After further experimentation, we settled on a 10kΩ resistor, which consistently produced readings across the full range and gave a clear distinction between tape and carpet. This allowed us to set a reliable threshold for binary conversion and ensured that the sensors responded accurately to changes in surface reflectivity.

Each of the four sensors underwent this calibration process individually to ensure they operated with equal sensitivity and consistency. We verified that all sensors produced similar outputs under identical conditions, minimizing the risk of bias or drift in the robot's navigation logic. Once calibrated, the sensors were mounted in an arc formation at the front of the robot, providing optimal coverage and alignment for line detection across a variety of track layouts.

3 Implementation

3.1 Circuit Diagram

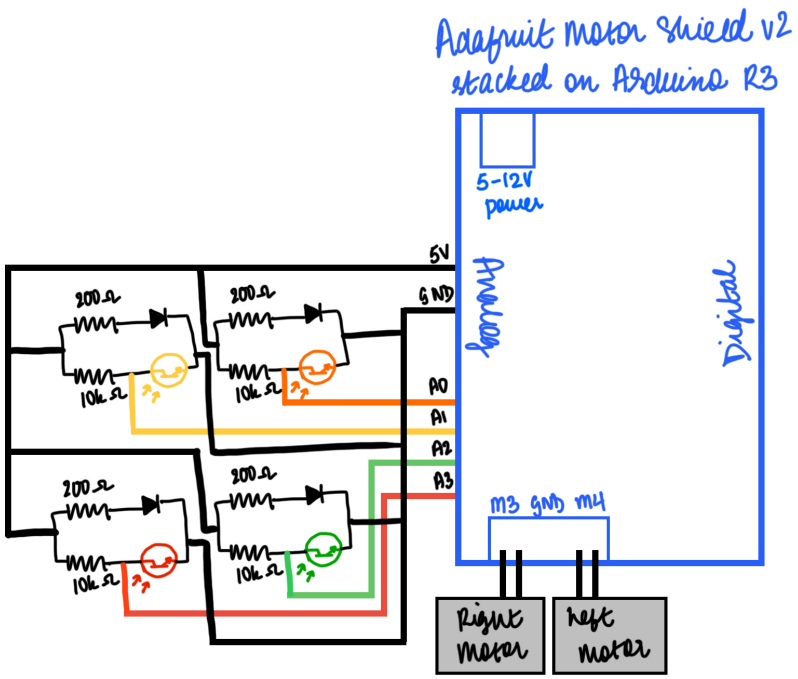

Figure 2: Circuit diagram of Motor Shield stacked on Arduino, connected to sensors and gearmotors.

3.2 Sensor/Motor Data

Figure 3: Sensor data overlayed with motor speeds over a 12 second data collection period.

3.3 Arduino IDE Code

Listing 1: Line-following robot with live motor and sensor speed reporting (Arduino IDE code)

```
#include <Wire.h>
#include <Adafruit_MotorShield.h>

Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// motors
Adafruit_DCMotor *leftMotor = AFMS.getMotor(4);
Adafruit_DCMotor *rightMotor = AFMS.getMotor(3);

// sensors: right to left = A3, A2, A1, A0
const int S1 = A0;  // left outer
const int S2 = A1;  // left middle
const int S3 = A2;  // right middle
const int S4 = A3;  // right outer

const int threshold = 200;

// base speed (can be updated live)
int baseSpeed = 35;

// live motor speeds
int currentLeftSpeed = baseSpeed;
int currentRightSpeed = baseSpeed;

void setup() {
  Serial.begin(9600);
  AFMS.begin();
  leftMotor->setSpeed(baseSpeed);
  rightMotor->setSpeed(baseSpeed);
}

void loop() {
  checkSerialCommands();

  // raw sensor readings
  int rawS1 = analogRead(S1);
  int rawS2 = analogRead(S2);
  int rawS3 = analogRead(S3);
  int rawS4 = analogRead(S4);

  // binary conversion
  int s1 = (rawS1 < threshold) ? 1 : 0;
  int s2 = (rawS2 < threshold) ? 1 : 0;
  int s3 = (rawS3 < threshold) ? 1 : 0;
  int s4 = (rawS4 < threshold) ? 1 : 0;

  // print 6 values: 4 sensors + 2 motor speeds
  Serial.print(rawS1); Serial.print(" ");
  Serial.print(rawS2); Serial.print(" ");
  Serial.print(rawS3); Serial.print(" ");
  Serial.print(rawS4); Serial.print(" ");
  Serial.print(currentLeftSpeed); Serial.print(" ");
  Serial.println(currentRightSpeed);
  delay(100);

  // MAIN LOGIC
  if ((s1 == 1 && s2 == 1 && s3 == 1 && s4 == 0) || (s1 == 1 && s2 == 1 && s3 == 0 &&
      s4 == 0)) {
    while (true) {
      sharpRightSpin();
      rawS1 = analogRead(S1);
      rawS2 = analogRead(S2);
      s1 = (rawS1 < threshold) ? 1 : 0;
      s2 = (rawS2 < threshold) ? 1 : 0;
      if (s1 == 0 && s2 == 0) break;
    }
```

```
  } else if ((s1 == 0 && s2 == 1 && s3 == 1 && s4 == 1) || (s1 == 0 && s2 == 0 && s3
    == 1 && s4 == 1)) {
    while (true) {
      sharpLeftSpin();
      rawS3 = analogRead(S3);
      rawS4 = analogRead(S4);
      s3 = (rawS3 < threshold) ? 1 : 0;
      s4 = (rawS4 < threshold) ? 1 : 0;
      if (s3 == 0 && s4 == 0) break;
    }
  } else if (s2 == 0 && s3 == 0) {
    goForward();
  } else if (s2 == 1 && s3 == 0) {
    softLeftTurn();
  } else if (s2 == 0 && s3 == 1) {
    softRightTurn();
  } else {
    searchForLine();
  }
}

// MOVEMENT FUNCTIONS

void goForward() {
  currentLeftSpeed = baseSpeed;
  currentRightSpeed = baseSpeed;
  leftMotor->setSpeed(currentLeftSpeed);
  rightMotor->setSpeed(currentRightSpeed);
  leftMotor->run(BACKWARD);
  rightMotor->run(BACKWARD);
}

void softLeftTurn() {
  currentLeftSpeed = baseSpeed - 15;
  currentRightSpeed = baseSpeed + 15;
  leftMotor->setSpeed(currentLeftSpeed);
  rightMotor->setSpeed(currentRightSpeed);
  leftMotor->run(BACKWARD);
  rightMotor->run(BACKWARD);
}

void softRightTurn() {
  currentLeftSpeed = baseSpeed + 15;
  currentRightSpeed = baseSpeed - 15;
  leftMotor->setSpeed(currentLeftSpeed);
  rightMotor->setSpeed(currentRightSpeed);
  leftMotor->run(BACKWARD);
  rightMotor->run(BACKWARD);
}

void sharpRightSpin() {
  for (int i = 0; i < 5; i++) {
    currentLeftSpeed = baseSpeed - 5;
    currentRightSpeed = baseSpeed + 50;
    leftMotor->setSpeed(currentLeftSpeed);
    rightMotor->setSpeed(currentRightSpeed);
    leftMotor->run(BACKWARD);
    rightMotor->run(FORWARD);
    delay(75);
  }
}

void sharpLeftSpin() {
  for (int i = 0; i < 5; i++) {
    currentLeftSpeed = baseSpeed + 50;
    currentRightSpeed = baseSpeed - 5;
    leftMotor->setSpeed(currentLeftSpeed);
    rightMotor->setSpeed(currentRightSpeed);
    leftMotor->run(FORWARD);
    rightMotor->run(BACKWARD);
    delay(75);
```

```
  }
}

void searchForLine() {
  currentLeftSpeed = baseSpeed - 10;
  currentRightSpeed = baseSpeed - 10;
  leftMotor->setSpeed(currentLeftSpeed);
  rightMotor->setSpeed(currentRightSpeed);
  leftMotor->run(BACKWARD);
  rightMotor->run(BACKWARD);
}

// SERIAL COMMAND HANDLER

void checkSerialCommands() {
  if (Serial.available()) {
    String input = Serial.readStringUntil('\n');
    input.trim();

    if (input.startsWith("set base ")) {
      baseSpeed = input.substring(9).toInt();
      Serial.println("Updated baseSpeed to " + String(baseSpeed));
    }
  }
}
```

The code shown above represents the complete logic for our line-following robot. It is organized into several key sections:

- **Initialization:** The `setup()` function initializes serial communication and motor control using the Adafruit Motor Shield. It sets the initial motor speeds based on a tunable base value.
- **Sensor Reading and Feedback:** In the `loop()`, the robot continuously reads analog values from four IR sensors. These readings are converted into binary states to determine whether each sensor detects tape (`1`) or carpet (`0`). The raw sensor values and current motor speeds are printed to the Serial Monitor for live feedback and plotting.
- **Main Logic:** The robot uses a series of conditional statements to interpret the binary sensor pattern and decide how to move. Specific combinations trigger movement functions such as forward motion, soft turns, or sharp spins to reorient the robot when it veers off track.
- **Movement Functions:** Each movement function sets motor speeds relative to the base speed and defines the direction of rotation for each motor. This modular structure allows for easy tuning and consistent behavior across different track scenarios.
- **Serial Command Handler:** The `checkSerialCommands()` function listens for user input via the Serial Monitor. If the user enters a command like `set base 45`, the robot updates its base speed in real time without needing to recompile or reload the code.

3.4 CAD Model / Mechanical Assembly

To support the electrical components and ensure consistent sensor placement, we designed and 3D printed a custom arc-shaped sensor panel mounted at the front underside of the robot chassis. This panel securely holds all four IR reflectance sensors in a curved formation, allowing them to evenly span the width of the robot's path. The arc geometry was chosen to maximize coverage and minimize blind spots, enabling smoother transitions during turns and better line detection. The sensor panel was mounted approximately one centimeter above the ground to optimize reflectivity and ensure consistent readings across tape and carpet surfaces. Additional mounts were designed for the Arduino and breadboard to keep the wiring organized and the assembly compact.

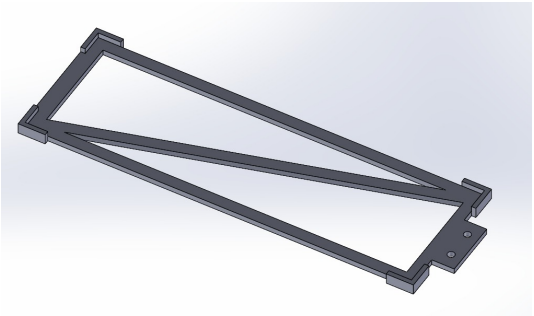

Figure 4: CAD model of the breadboard mount.

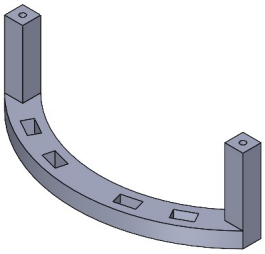

Figure 5: CAD model of the robot with arc-shaped sensor panel.

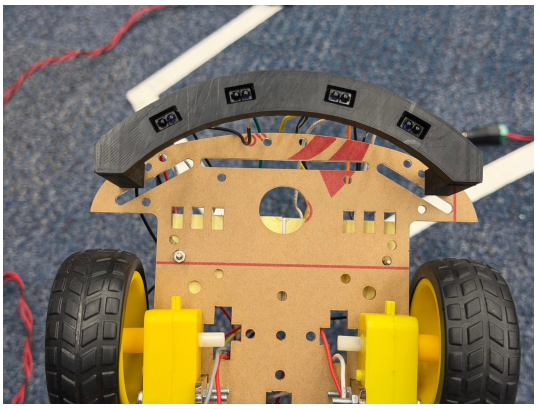

Figure 6: Bottom view of the assembled robot showing sensor placement.

4 Results

The robot was successfully able to follow a non-circular tape loop using its closed-loop control logic and sensor feedback. The modular design allowed for live tuning of motor speeds, and the arc sensor configuration provided reliable detection across a variety of track layouts. A demonstration video of the robot in action can be viewed at the following link:

Demo Video: https://drive.google.com/file/d/13Ff4ENiHnv1-uZZuqKoeQ0IPsJ7xy56_/view?usp=sharing

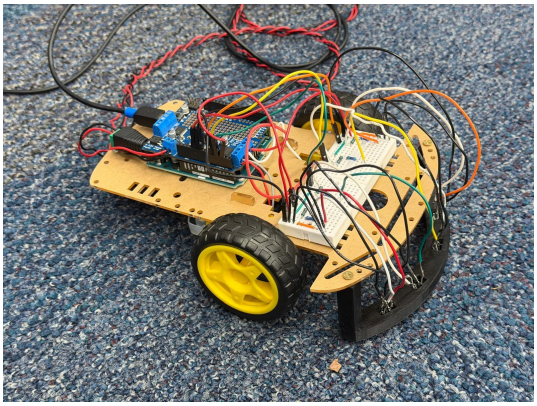

Figure 7: Final assembled robot.

5 Reflection

Reflecting on the mechanical aspect of this project, our car design was relatively simple but effectively met our goals. Starting with the provided chassis and motor mounts gave us a solid foundation, allowing us to focus on refining smaller design details. Designing and 3D printing the mounts for the breadboard and Arduino helped us integrate the electrical and mechanical systems in a clean and organized manner. Creating the arc-shaped mount for the IR sensors emphasized the importance of precise positioning—keeping them about one centimeter above the ground was key to ensuring consistent readings. Overall, the process reinforced that even straightforward mechanical designs demand thoughtful planning and careful execution. As a team, we would like to experiment with alternative IR sensor geometries in future iterations, such as a straight-line configuration, which appeared to be a common and potentially effective approach among other teams.

Electrically, the sensor wiring and calibration process proved to be more challenging than expected due to the variety of hardware issues that could arise. Many hours could have been saved had we referred to the datasheets earlier in the process. For example, we encountered persistent issues with analog pins A4 and A5 shorting unexpectedly—this was due to their shared power and ground lines with the I^2C communication bus used by the motor shield. It took considerable time to identify and resolve this issue, which was not immediately obvious. Additionally, during testing, the robot would occasionally stop mid-run and require a manual nudge to resume motion. While the root cause remains unclear, we suspect it may be related to motor stability or inconsistent power delivery. Smoother motor choices or improved power regulation could help mitigate this behavior in future builds.

Looking ahead, most of our proposed improvements focus on mechanical refinement. We plan to explore a straight IR sensor configuration to compare its performance with the arc layout. A smaller, more mobile chassis design could improve maneuverability and reduce weight. We also aim to find ways to stabilize motor behavior, possibly by selecting motors with better torque consistency or integrating motor encoders. On the software side, we experimented with PID control logic but ultimately shifted to a binary threshold-based approach for simplicity. In future iterations, we would like to revisit PID control to achieve smoother and more adaptive line-following behavior, especially on complex or high-speed tracks.

6 References

- Pigmice Robotics. *Intro to Control Theory*. Retrieved from https://www.pigmice.com/post/intro-to-control-theory
- Beauregard, Brett. *Improving the Beginner's PID Introduction*. Retrieved from http://brettbeauregard.com/blog/2011/04/improving-the-beginners-pid-introduction/
- PID Control Explained — YouTube Video. Retrieved from https://www.youtube.com/watch?v=RZW1PsfgVEI

- Vishay. *TCRT5000 Reflective Optical Sensor Datasheet*. Retrieved from https://www.vishay.com/docs/83760/tcrt5000.pdf
- Adafruit. *Using DC Motors with the Motor Shield v2*. Retrieved from https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino/using-dc-motors
- Digi-Key. *Seeed Technology IR Sensor Product Page*. Retrieved from https://www.digikey.com/en/products/detail/seeed-technology-co-ltd/114090050/10385087
- Adafruit. *Motor Shield v2 for Arduino Overview*. Retrieved from https://learn.adafruit.com/adafruit-motor-shield-v2-for-arduino