Oscar Bao, Peter Sapiro-Mitten, Sam Wisnoski (Group 40)
Principles of Integrated Engineering Section 2

Mini Project 3: Robotic Line Follower

I. Introduction:

Objectives:

1. Integrate your electronic and mechanical hardware with your chassis platform.
2. Implement a controller that uses the optical reflectivity sensors provided to guide your robotic platform along a "track," driving the two DC motors independently to move your robot to follow a black tape line around a loop as fast as possible.
3. Create an interface to your Arduino code that allows you to change the behavior of your controller from your laptop (via the serial connection) without recompiling or reloading your Arduino code.

This project involves building an autonomous robot that follows a track of electrical tape using infrared (IR) sensors and DC motors. The robot, controlled by an Arduino and motor shield, uses three IR sensors to detect the tape's position and adjust its movement accordingly through a closed-loop controller. The goal is to navigate a tape track as quickly as possible. The system allows real-time tuning via a serial interface without recompiling code. Integration of hardware, sensor feedback, and control ensures the robot stays centered on the track.

Our approach to this robot was to use a total of three sensors on the front of our robot. One was directly over the tape, and the other two were to the direct right or left of the tape. When the robot either detected that the middle sensor was off the tape or the left/right sensor was on the tape, it rotated in the appropriate direction in order to stay on course.

Additionally, we used the arduino functions "Serial.readStringUntil", an easy way for our robot to react to serial. If we sent the serial message "start", it would increase our base speed to 50, and sending "slow" would lower it to 30.

II. Design Process: Energy/Data Flow, Circuit Diagram, and Calibration

Requirements:

1. System diagram portraying data and energy flows
2. A circuit diagram. Diagram is clear, complete, and easy to read. Decisions are clearly explained, and any relevant math is included
3. A description of the process you used to test and calibrate your IR sensor

The first step we took in the design process was to sketch out a data flow and energy flow diagram between the power sources, arduino, motors, and sensors. As seen in figure 2.1, the data flows from the sensors to the arduino, which controls the motors. The arduino can also send data to the computer for data storage and plotting. Power flows from the wall to the motor controller, and from the computer to the arduino. The arduino powers the sensors with 5V, while the motor controller controls the motors with 12V.

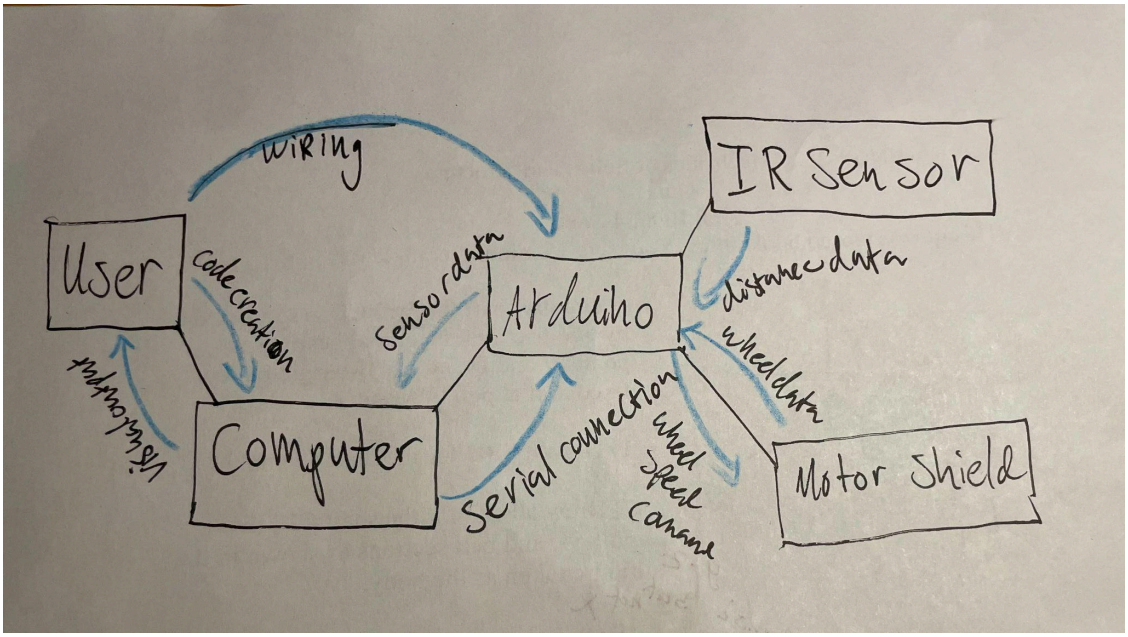

Figure 2.1: Our data/energy flow diagram.

After creating our data/power flow diagram, our next steps were to find appropriate resistance values and create a circuit diagram.

First, let's find resistor values for our sensors:

- For our IR sensors, the typical forward voltage is 1.2V (max 1.5V) at 20 mA. The supply to the sensor is five volts. Using Ohm's law, we can calculate the resistance as:
  $I = V/R \quad -> \quad R = V/I \quad -> \quad R = (5V - 1.2V)/(20mA) = 190\Omega$
  With our calculated resistor being 190Ω, we decided to use a 200Ω resistor.

- For the Phototransistor, a 10kΩ pull-up resistor is a typical choice to ensure reliable switching with the 5V input. After testing several higher ohm resistors, we decided that 10kΩ worked well for our system and provided us with good variation in sensor data.

Using these resistor values, we created our circuit diagram:

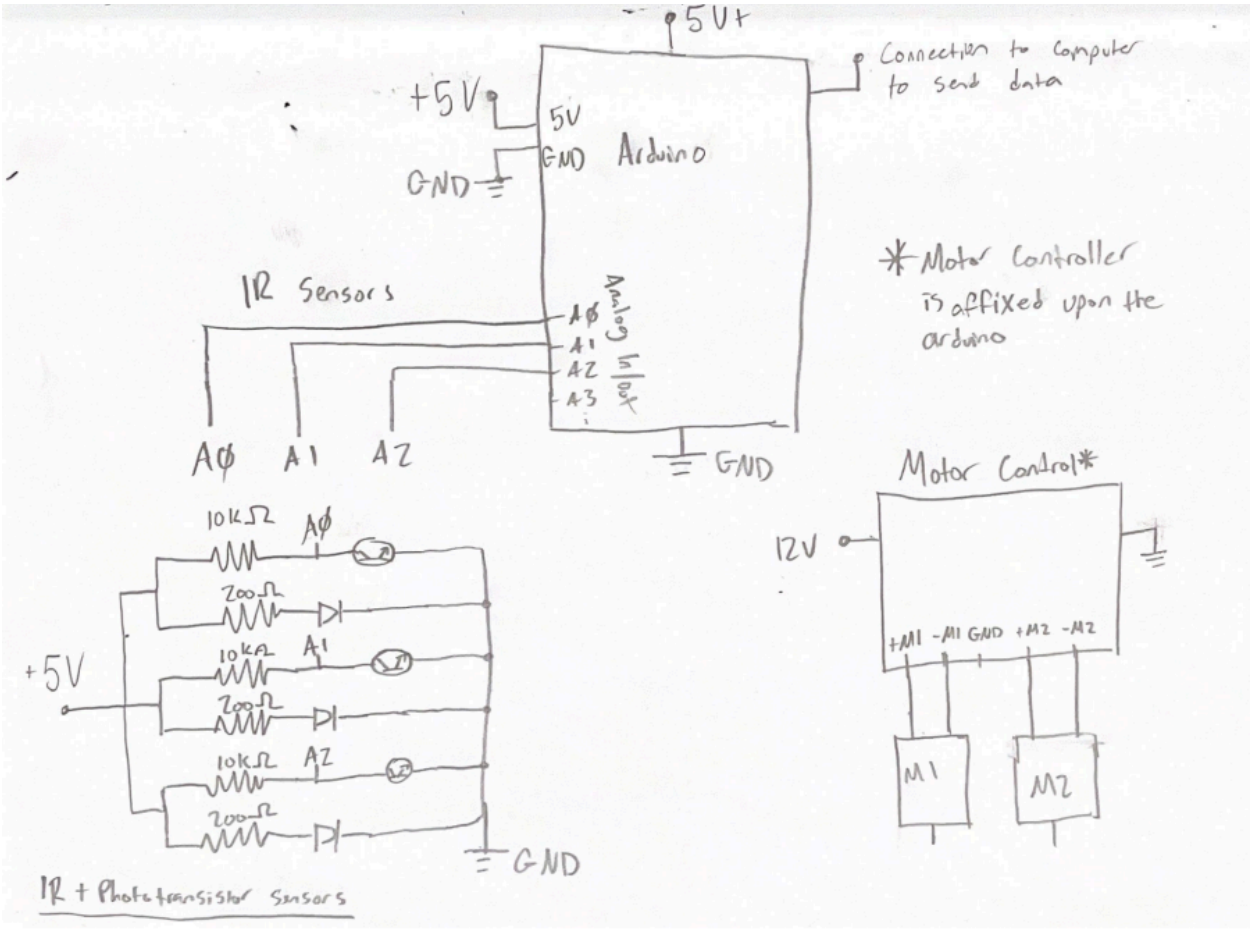

Figure 2.2: Circuit Diagram. Our circuit contained an arduino connected to a motor controller, which powered two motors and three IR/phototransistor sensors.

Once we completed our circuit, our strategy to calibrate the sensors was fairly simple. First, we created mounts (see Figure X.X) in order to hold the sensors at a constant height above the ground. Then, we connected our sensors to the appropriate pins on the Arduino and verified that it worked using the "AnalogInput" example from the Arduino development environment. We used basic arduino commands to read the sensor and print its output for each loop, as seen in figure 2.3.

```
void loop() {    #LOOP OVER read_sensor()
read_sensor()
}

void read_sensor(){
  l_pos = analogRead(SEN_LEFT);  // reads current left output
  m_pos = analogRead(SEN_MIDDLE);  // reads current middle output
```

```
  r_pos = analogRead(SEN_RIGHT);  // reads current right output
  Serial.print("Readings: "); Serial.print(l_pos); Serial.print(",");
Serial.print(m_pos); Serial.print(","); Serial.println(r_pos);
}
```

Figure 2.3: Code for reading and displaying IR sensor results

Next, we moved our robot to the floor and took readings for each sensor. Then, we put a line of black electrical tape on the ground and moved each sensor incrementally closer to the electrical tape line, creating the chart in figure 2.4.

Distance from Electrical Tape	Left Sensor Output	Middle Sensor Output	Right Sensor Output
5 cm	~ 340	~ 590	~ 700
4 cm	~ 350	~ 600	~ 700
3 cm	~ 210	~ 550	~ 690
2 cm	~ 340	~610	~ 710
Edge of Electrical Tape (1 cm)	~ 530	~ 700	~ 800
Directly Above Electrical Tape (0 cm)	~ 800	~ 850	~ 900

Figure 2.4: The output from the left, middle, and right sensors compared to the distance from the tape.

Based on sensor readings, we determined each sensor's proximity to the tape. For this project, we only needed to know if the sensors were or were not directly over the tape. Using the data from Figure 2.4, we set the following cutoff values to identify if a sensor was above the tape:

1. Left Sensor: 700
2. Middle Sensor: 700
3. Right Sensor: 850

A sensor value higher than its cutoff indicated it was over the tape. We could now begin programming our line-following algorithm based on the cutoff values.

III. Design Process: Mechanical Design/Integration

Requirements:

1. Sensor mounts and chassis interface/design is documented either through pictures and CAD. Decisions are clearly explained and relevant math is included.

The sensor mounts allow for easy integration of the infrared sensors into the chassis of our vehicle. The sensor mounts needed to be simple and easily attach and detach from the chassis, to allow for quick modifications and experimentation of sensor placement.

Initially, we anticipated using two sensors, one placed directly in front of each wheel. For this iteration, the sensors were placed in the two oval slots in the center of the chassis. The mount used two parallel bars 3.5mm apart, secured with a perpendicular member, as seen in figure 3.1. This allowed for press fit onto the chassis. A 5.30 mm x 6.60 mm hole was placed 34.25 mm from the inner edge of the lower bar, which allowed for press-fit of the sensor. Unfortunately, this design was ineffective for several reasons. The first was that the sensor was directly flush with the chassis, making it difficult to wire and modify. The second issue was the location of the sensors. We found that placing three sensors in the very front of the vehicle would yield better results.

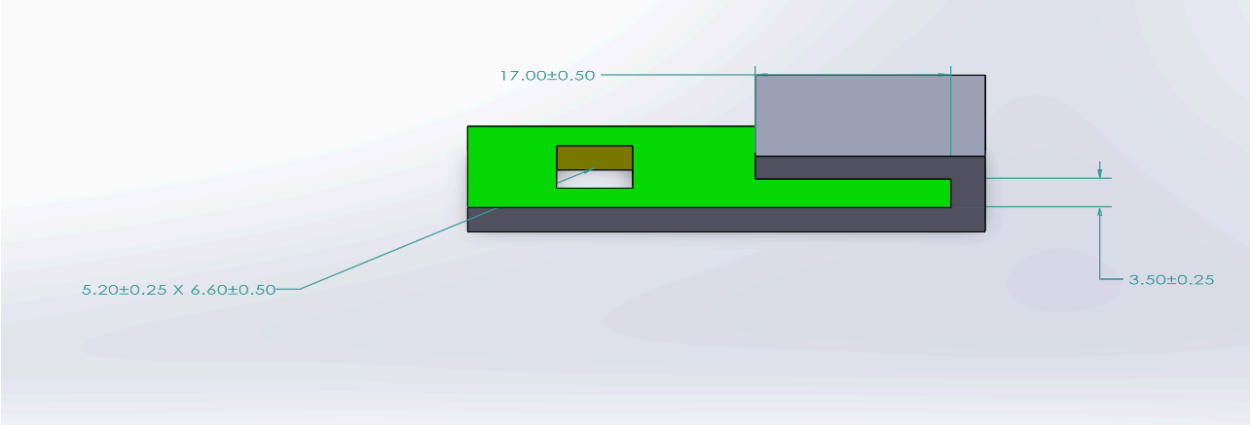

Figure 3.1: The first iteration of our sensor mount

To address these issues, the prior design was modified. The two bars that secured to the chassis remained but a third bar was also added, this is where the hole for the sensor is placed. For the two mounts on the sides, the hole was placed 27 mm from the edge, for the middle one the hole was placed 30.50 mm from the edge. The size of the hole remained the same. This new design allowed for easier access to the wiring and better placement of the sensors. An additional benefit was that the readings from the sensors were more accurate because they were closer to the ground. The goals of the design were met, with a simple and effective mounting system as seen in figure 3.2.

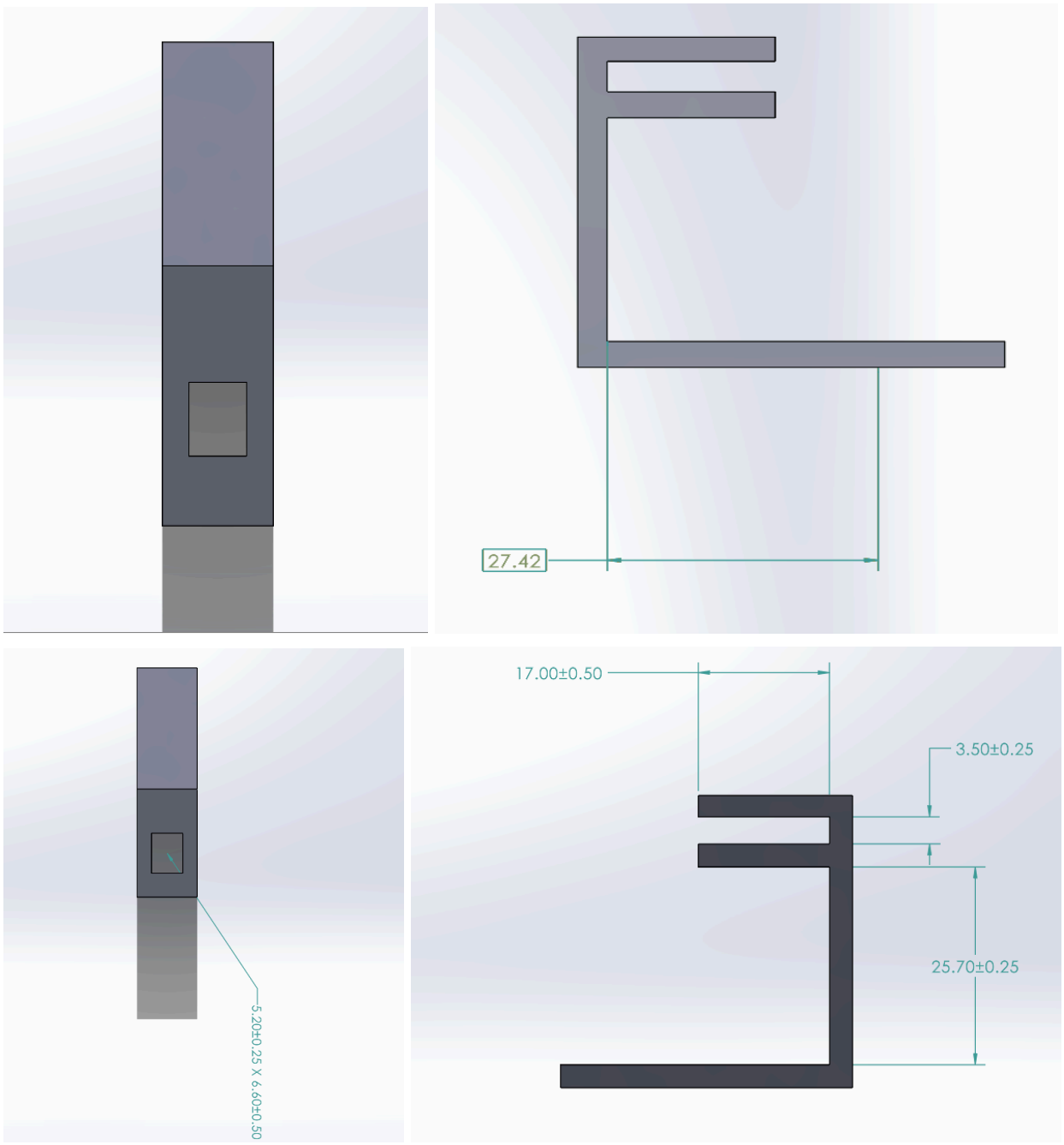

Figure 3.2: Top, left, bottom, and right side views of the second iteration of sensor mount

The second iteration of sensor mounts held the sensors lower to the ground, increasing reliability, and allowed us to move the sensors as needed. The sensor mounts attached to the car are shown in figure 3.3.

When assessing which breadboard to use, we decided to use a breadboard that fit perfectly on the chassis. While the weight of the breadboard caused our car to have trouble moving at low speeds, it was easy to attach and modify as needed. The breadboard can also be seen in figure 3.3.

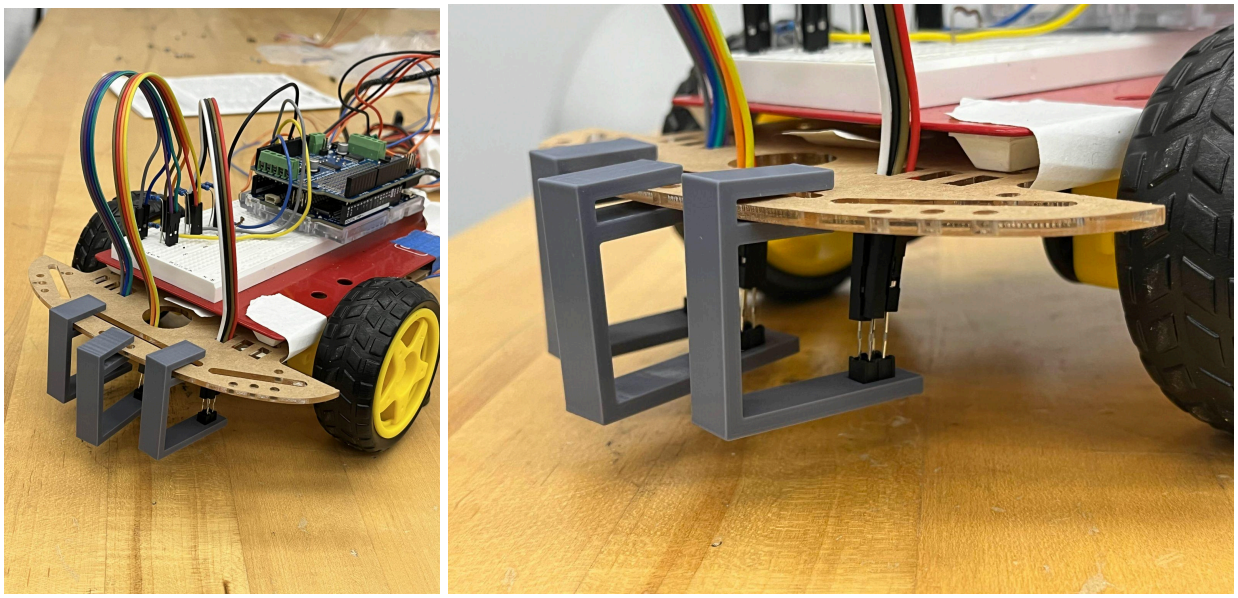

Figure 3.3: Our fully integrated breadboard and sensor mounts

IV. Design Process: Software and Motor Control

Requirements:

1. A description of how your controller works
2. Clear explanations of how these tasks were achieved: Explanation of Control Logic, Serial Interface

Our controller operates on a simple but mostly infallible set of logic. Firstly, the controller takes in the sensor values from the three IR sensors. Then, to determine whether each sensor is on the line, we can check if the sensor value is over the threshold set from our calibration.

After determining if each sensor is on the tape, we use a set of if statements to set the motor speeds. Specifically, if either the left or the right sensor detects the line, we set the opposite motor to move forward while pausing the motor on the side that detected the line. The robot only reverts back to going straight when neither the left nor the right sensor detects the line, while the middle sensor does. Setting our logic this way helps prevent the robot from going off course during sharp turns — when the middle sensor doesn't detect the line, it continuously turns until it reaches the line again. Please reference the appendix for exact code.

As far as the serial interface, we established a serial connection between our computer and the Arduino. In the loop function, we set up code to continuously scan for serial messages using the "Serial.readStringUntil" command. If it receives the serial message "start", we increase the base speed of the motors. Similarly, if the arduino receives the serial message "slow" , we then decrease the base speed. All other messages are ignored. Please reference the appendix for exact code.

V. Results:

1. Plot superimposing sensor data from both your IR sensors and commanded motor speeds (generated by your controller) for a short trial run.

In order to generate this plot, we edited our arduino code to print out the values of each sensor and commanded motor speed over serial connection. We then had matlab read and print the data. Both of these codes can be found in section 7, the appendix. The plot is figure 5.1.

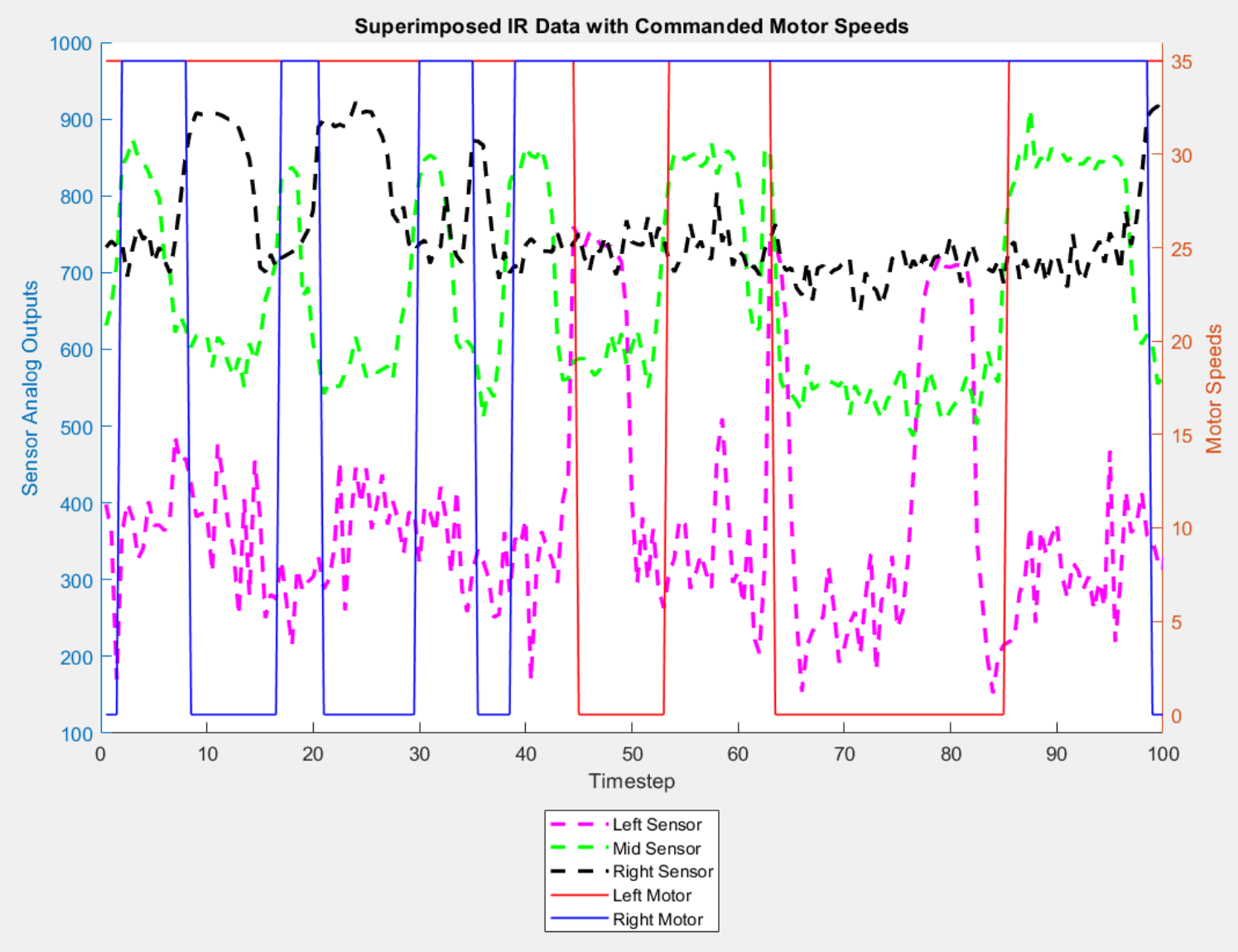

Figure 5.1: A plot of IR Sensor data for our three sensors superimposed with a plot of our motor speeds.

2. A link to a video of your successful completion of the track

   Here is our video of our successful track run. We have our robot follow the course in 309 for an entire lap, then demonstrate increasing and decreasing the speed of the robot on the second lap.

   Long Link:
   https://drive.google.com/file/d/1lK7P3rbPzFtiytiUWzr1FiErMPKT_QmH/view?usp=sharing

VI. Reflection

Requirements:

1. There is some reflection on what went well/what could be improved.

Overall, the project went very well. We had a great team dynamic with Peter working on the mechanical design, Sam doing circuits/electrical work, and Oscar working on the code. We communicated efficiently and made sufficient progress in the first few classes. As a whole, our team worked very effectively on this project. We were able to use class times productively, and delegate tasks so every team member was able to contribute to the project while not being overly burdened. We were all satisfied with the quality of work contributed.

However, during Fall Break, we all became very busy and were unable to work much on the project. While we thought we were in a good spot, we ended up having to work significantly after class in order to complete the project on time. In regards to technical components, if we had more time to work on the project, we would have loved to make a more efficient line-following algorithm. We felt limited by the number and reliability of the sensors, but given more time, we may have been able to use PID or better sensor placement to more reliably navigate the course.

VII. Appendix

Arduino Code with sensor reading and line following algorithm:

```
#include <Adafruit_MotorShield.h>

// Create the motor shield object with the default I2C address
Adafruit_MotorShield AFMS = Adafruit_MotorShield();

// Select which 'port' M1, M2, M3 or M4. In this case, M1
Adafruit_DCMotor *lMotor = AFMS.getMotor(1);
```

```
Adafruit_DCMotor *rMotor = AFMS.getMotor(2);

// Tracks sensor value
int l_pos = 0;
int r_pos = 0;
int m_pos = 0;

// Tracks whether sensor is on the line, if error is a negative value,
// that means the sensor is not on the line, vice versa.
int lError(int pos) {
  return (pos - 700);
}
int rError(int pos) {
  return (pos - 850);
}
int mError(int pos) {
  return (pos - 700);
}

float motorspeed;
int motorspeedl = 0, motorspeedr = 0;
int basespeedl= 35, base speedy = 35;

const uint8_t SEN_LEFT = 2;
const uint8_t SEN_RIGHT = 0;
const uint8_t SEN_MIDDLE = 1;
double current_reading_left;
double current_reading_right;

enum states {
  LEFT, RIGHT, STRAIGHT
};
states state;


void setup() {
  Serial.begin(9600);           // set up Serial library at 9600 bps

  if (!AFMS.begin()) {         // create with the default frequency 1.6KHz
  // if (!AFMS.begin(1000)) {  // OR with a different frequency, say 1KHz
    Serial.println("Could not find Motor Shield. Check wiring.");
    while (1);
  }
  Serial.println("Motor Shield found.");
```

```
  // Set the speed to start, from 0 (off) to 255 (max speed)
  lMotor->setSpeed(0);
  lMotor->run(FORWARD);
  // turn on motor

  rMotor->setSpeed(0);
  rMotor->run(FORWARD);

  // Set up sensor readings
  l_pos = analogRead(SEN_LEFT);
  r_pos = analogRead(SEN_RIGHT);

  state = STRAIGHT;
}

void loop() {

  if (Serial.available() > 0) {
  String serial_msg = Serial.readStringUntil('\n');

    // Check for serial message to increase or decrease speed
    if (serial_msg.equals("start")) {
      basespeedl = 50;
      basespeedr = 50;
    }
    else if (serial_msg.equals("slow")) {
      basespeedl = 30;
      base speedy = 30;
    }
  }

  // Calculate whether sensor is on the line
  read_sensor();
  l_error = lError(l_pos);
  r_error = rError(r_pos);
  m_error = mError(m_pos);



  // If left or right on line, turn other direction
  if (l_error > 0){
    state = LEFT;
  }
  else if (r_error > 0){
    state = RIGHT;
```

```
  }
  // Only revert back to straight
  else if (r_error < 0 && l_error < 0 && m_error > 0) {
    state = STRAIGHT;
  }

  // switch-case system to determine speed
  switch (state) {
    case STRAIGHT:
      motorspeedr = basespeedr;
      motorspeedl = basespeedl;
      break;

    case LEFT:
      motorspeedl = 0;
      motorspeedr = basespeedr;
      break;

    case RIGHT:
      motorspeedl = basespeedl;
      motorspeedr = 0;
      break;
  }


  lMotor->setSpeed(motorspeedl);
  rMotor->setSpeed(motorspeedr);
  delay(20);
}


void read_sensor(){
  l_pos = analogRead(SEN_LEFT);  // reads current left output
  m_pos = analogRead(SEN_MIDDLE);  // reads current right output
  r_pos = analogRead(SEN_RIGHT);  // reads current right output
  Serial.print(l_pos); Serial.print(",");
  Serial.print(m_pos); Serial.print(",");
  Serial.print(r_pos); Serial.print(",");
  Serial.print(motorspeedl); Serial.print(",");
  Serial.println(motorspeedr);
}
```

Matlab Code for receiving data and plotting sensor/motor data:

```
% Clear previous serial connection object
clear s
% Display available serial ports on the system
freeports = serialportlist("available");
% Choose which port to use for Arduino (can be hardcoded)
ports = "COM6"; % Can also use freeports(2) for dynamic selection
baudrate = 9600; % Set the baud rate for serial communication
% Establish a connection to the specified serial port
s = serialport(ports, baudrate);
% Initialize empty arrays to store sensor and motor data
left_sensor = [];
mid_sensor = [];
right_sensor = [];
left_motor_speed = [];
right_motor_speed = [];
time = [];
% Initialize index for storing values
i = 1;
% Main loop to read data from the Arduino, process, and display it
while i<100 % Run 100 times to collect data from loop
   % Check if data is available to read from the Arduino
   while s.NumBytesAvailable > 0
       % Read data from the serial port and convert it to an array of integers
       values = eval(strcat('[', readline(s), ']'));
       % Assign individual values to variables a, b, and c
       a = values(1);
       b = values(2);
       c = values(3);
       d = values(4);
       e = values(5);
       % Print the results for each data set
       fprintf('a,b,c,d,e = %d,%d,%d,%d,%d\n', a, b, c, d, e);
       left_sensor(i) = a; % Store left sensor data
       mid_sensor(i) = b;  % Store middle sensor data
```

```
        right_sensor(i) = c; % Store right sensor data
        left_motor_speed(i) = d; % Store left motor speed
        right_motor_speed(i) = e; % Store right motor speed
        time(i) = i*0.5; % Store current timestep
        i = i + 1; % Increment index for next set of values
    end
    % Pause before next iteration
    pause(0.5);
end
% plot the values on a graph with 2 y axis
figure;
hold on;
yyaxis left; % plot sensor data on the left axis
p1 = plot(time, left_sensor, 'm--', 'LineWidth', 2);
p2 = plot(time, mid_sensor, 'g--', 'LineWidth', 2);
p3 = plot(time, right_sensor, 'k--', 'LineWidth', 2);
ylabel('Sensor Analog Outputs');
yyaxis right; % plot the motor speeds on the right axis
p4 = plot(time, left_motor_speed, 'r-', 'LineWidth', 1);
p5 = plot(time, right_motor_speed, 'b-', 'LineWidth', 1);
ylabel('Motor Speeds');
ylim([-1 36]);
xlim([0 100]);
% plot the xlabel, title, and legend
xlabel('Timestep');
title('Superimposed IR Data with Commanded Motor Speeds');
legend([p1, p2, p3, p4, p5], 'Left Sensor', 'Mid Sensor', 'Right Sensor', 'Left
Motor', 'Right Motor', 'Location', 'southoutside');
```