

# POE Lab Report: DC Motor Control

Dennis Chen & Jennifer Wei

October 2015

## 1 DC Motor Control Description

For this lab, we integrated our Arduino and a circuit with a standard robot chassis. Using a closed-loop controller run on the Arduino, two IR sensors, a motor shield, and a chassis, the robot was able to follow an electrical tape track laid out on the floor. Additionally, we set up serial communication so that we could update the behavior of the control code without restarting the Arduino.

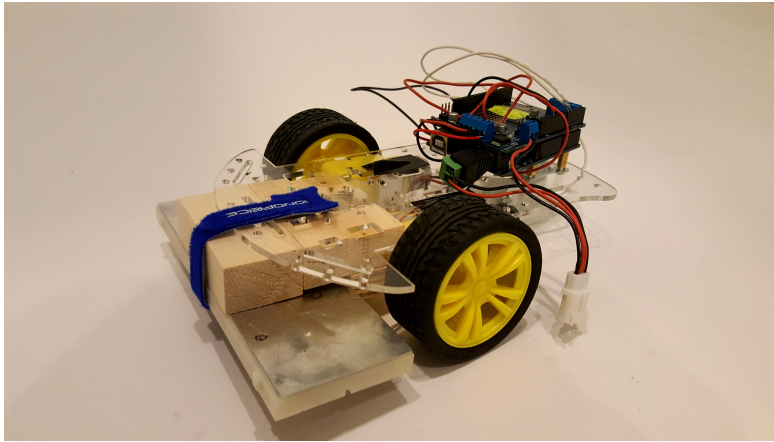


Figure 1: Final mounting setup of the circuit board and Arduino on the chassis.

## 2 IR Sensor Circuit

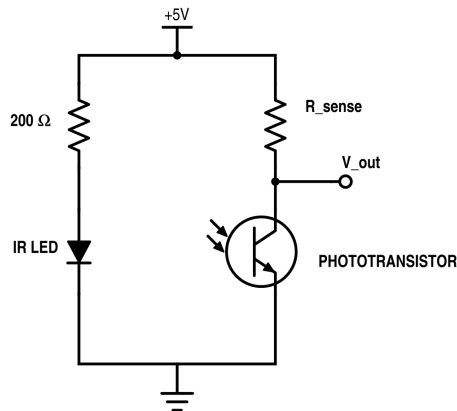


Figure 2: Schematic for the IR circuit used.

Using the schematic provided (as shown in Figure 1) as well as the data sheet for the IR sensors, we found the value for  $R_{sense}$  to complete our circuit.

From the data sheet and tutorial, we know that:

1.  $V = iR$
2.  $I_C$  proportional to  $I_B$ , where  $I_C$  is the current going through the photo transistor (collector current)
3.  $I_C$  proportional to  $I_B$
4.  $I_C$  proportional to  $I_F * \text{Reflectance}$ , where  $I_F$  is the current going through the diode (forward current)

Assuming that  $I_F$  is  $20mA$ ,  $I_C$  is approximately  $2mA$  based off the  $I_F$ - $I_C$  current relationship graph from the datasheet.

Assuming that the photo transistor needs a voltage drop of  $1V$ , we can use  $V = iR$  to find that  $R_{sense}$  should be  $2.0k\Omega$  under the ideal environment (a mirror surface with the sensors at a specific distance from the mirror).

### 3 IR Sensor Calibration And Testing

To better understand the readings from the sensors, we collected data from the sensors when they were on the table, on the dark tape, and on-off (where one sensor is on the table and the other is on the dark tape), and we did this at different distances from the table.

	2K, 1.0 cm	2K, 1.5 cm	2K, 3.0 cm	20K, 1.5 cm
Both Off the Table	-2	0	0	-1
Left On, Right Off	176	180	7	600
Left Off, Right On	-155	-130	-11	-630
Both On Tape	-10	-5	-1	37

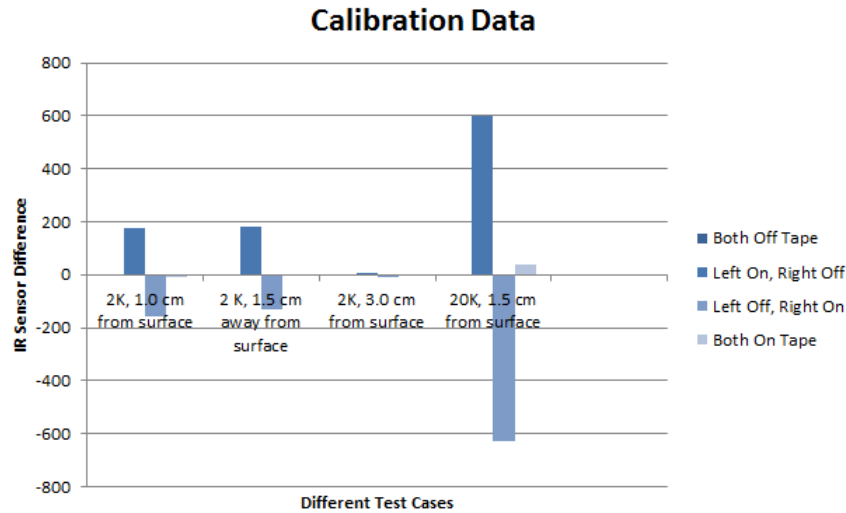


Figure 3: Calibration data collected from the IR sensor.

After doing preliminary testing with the 2K resistor, we noticed that the data collected was not as drastic as we had thought it would be, so we decided to test with a larger resistor, a  $20K\Omega$ , and we ended up with a higher resolution result than before (as shown in Fig. 3) and stuck with that.

### 4 Mechanical Attachment To Mount

Initially we set up our circuit with the IR sensors close together at the far end of a breadboard, which we aligned vertically with the vehicle. This created a front-heavy chassis with a long sensing arm that made the vehicle awkward.

To fix this, we shifted to mounting the breadboard horizontally to the vehicle so that the sensors would be farther apart, lower to the ground, and closer to the center of the chassis, which allowed for tighter turn radii, a more compact vehicle than before, and better sensor readings.

We created a simple interface to attach the breadboard to the vehicle. Inspired by Aaron, we decided to go with the screw and velcro approach, screwing the Arduino to the top of the chassis and velcro-ing the breadboard to the bottom. As mentioned earlier, we attempted to attach the short end such that the length of the breadboard would be perpendicular to the wheelbase of the chassis, and that had more cons than pros. Thus, we rotated our board so that the length of the breadboard would be parallel to the wheelbase. To prevent the breadboard from interfering with the wheels, we created a wooden spacer that kept the breadboard away from the wheels and also helped support the breadboard. Additionally, we sized our spacer to align with tabs on the chassis (on the yellow plate). That way, regardless of which chassis we borrowed, we would be able to easily attach, align, and secure our breadboard with minimal hassle.

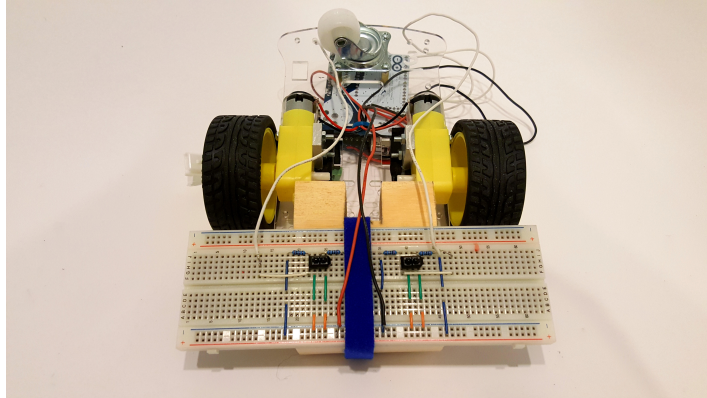


Figure 4: Image of the system used to mount our sensors to the chassis. A wood block and Velcro strips were used to secure our circuit and sensors in place.

## 5 PID Controller

To control our robot, we chose to implement a PID control system, though we found that our line following robot was most effective when only proportional control was used. Given more time with the chassis to experiment with parameters, perhaps we would have found some combination of proportional, integral, and derivative control with better performance, but the robot seemed to do best with just proportional control in the trials we ran, so we chose to leave it as is.

To implement our control algorithm, we had the robot attempt to minimize the differences between the values read by the two IR sensors. Since the sensors are placed so that they are looking at the tile on opposite sides of the black tape when the robot is centered, the difference in sensor readings are small when the robot is centered, and large when the robot starts to veer off the tape. In our program, we measured error as the difference between the two IR sensor readings, and then used the error, the derivative of the error, and the integral of the error multiplied by constants to determine the adjustment to be made to the system. This adjustment value was used to set the difference in speeds of the two motors, causing the robot to turn and correct its path, staying on the black tape.

The implementation of PID control can be seen in lines 61 to 69 of our sample code. First, error is measured by taking the difference between the measured sensor values. Next, we determine the correction to be made using the following equation:

$$\text{Correction} = P * \text{Error} + I * \text{Integral of error} + D * \text{Derivative of the error}$$

where P, I, and D are experimentally determined constants. Then the difference in the motor speeds is set to equal the correction. In order to do this, we set one motor's speed to be some `BASELINE.SPEED` plus the correction divided by 2, and set the speed of the second motor to be the `BASELINE.SPEED` minus the correction divided by 2. We found that PID values of 0.3, 0.0, and 0.0 (respectively) worked well, allowing for the robot to travel smoothly along the track when the baseline motor speed was set to 25.

## 6 Sensor and Motor Speed Plots

To confirm that our control was working in the way we expected, we took plots of the two sensor values and the set motor speeds for a trial run where the robot followed the black tape.

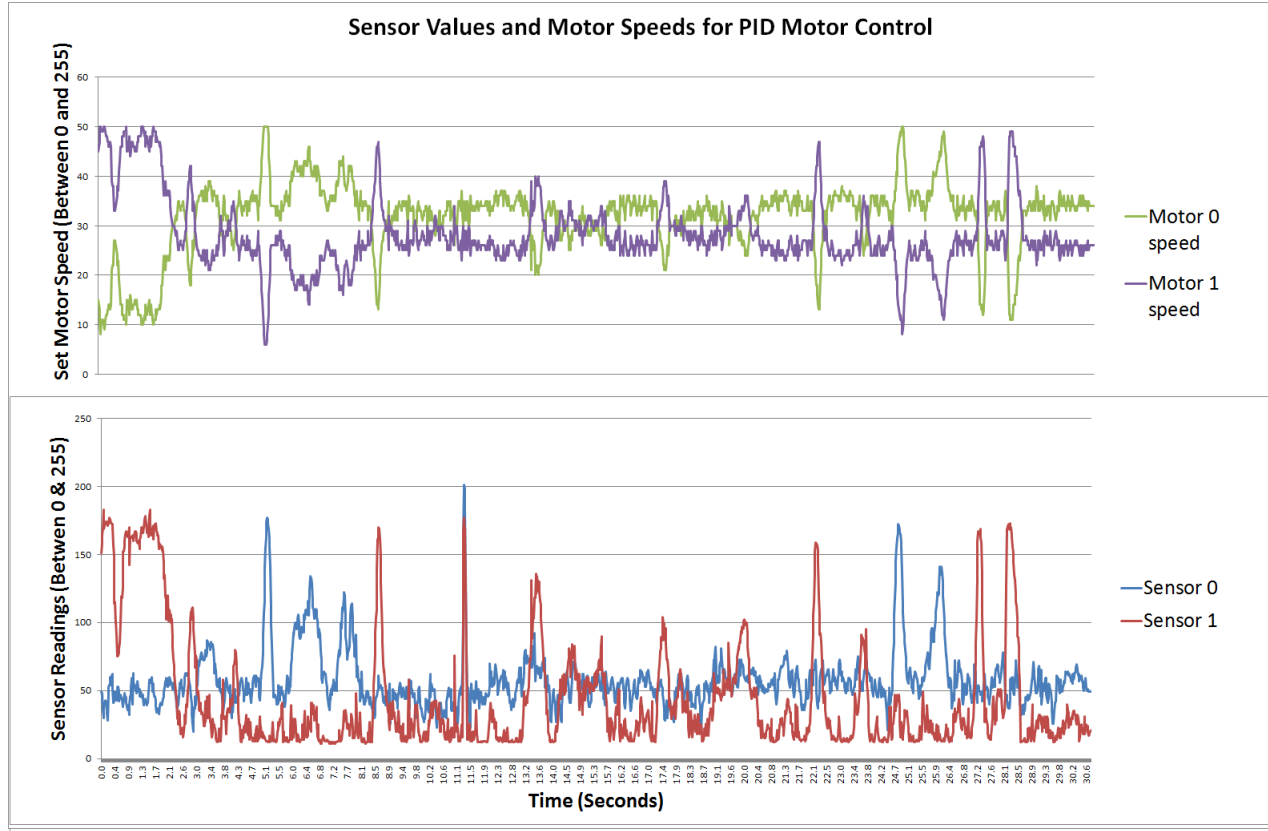
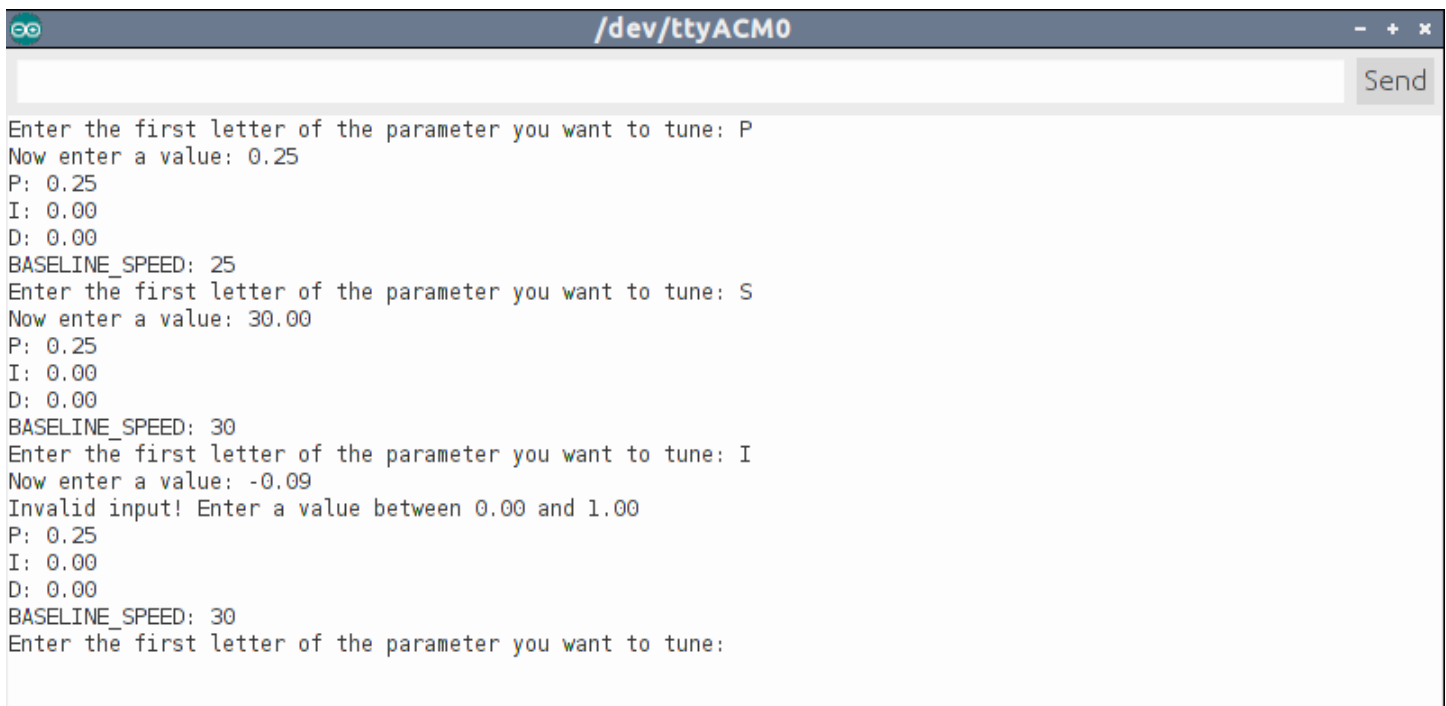


Figure 5: Motor speeds and sensor values for our robot for a short trial run.

The proportional control behavior can be seen in the plots above. Whenever there is a large difference between the measured IR sensor values (the blue and red lines), the control ensures that there is a large difference between the motor speeds (the green and purple lines). We can see that the spikes in motor speeds correspond to spikes in the differences between the measured sensor values. From the plots we can also see the effect of control reigning in the sensor values and ensuring that the difference in values between sensor 0 and sensor 1 stay relatively small. Whenever the difference in sensor values begin to increase, so does the motor speed, which causes a subsequent decrease in the difference in sensor values. In the bottom graph, when there is a large red or blue peak, the left side of the peak is the robot straying away from the line, and the right side of the peak is the control and difference in motor speed bringing the robot back onto the line.

## 7 Serial Control of Arduino

To be able to tune parameters without having to re-upload our code to the Arduino, we included functionality in our code so that we could enter new parameter values from the Serial monitor. We created a basic serial interface that allowed the user to enter the variable they wished to change (P, I, D, or S, the `BASELINE.SPEED`), and to then subsequently enter the float value they wished to set the variable to. We were successfully able to use this to change P and the `BASELINE.SPEED` while the robot went around the course, and saw the expected changes in behavior, which included the robot wiggling due to over-correction as we increased P, and going faster if we increased the `BASELINE.SPEED`. A picture of our serial monitor interface in action is shown below.



```
Enter the first letter of the parameter you want to tune: P
Now enter a value: 0.25
P: 0.25
I: 0.00
D: 0.00
BASELINE_SPEED: 25
Enter the first letter of the parameter you want to tune: S
Now enter a value: 30.00
P: 0.25
I: 0.00
D: 0.00
BASELINE_SPEED: 30
Enter the first letter of the parameter you want to tune: I
Now enter a value: -0.09
Invalid input! Enter a value between 0.00 and 1.00
P: 0.25
I: 0.00
D: 0.00
BASELINE_SPEED: 30
Enter the first letter of the parameter you want to tune:
```

Figure 6: Using our serial monitor to alter settings for the DC motor controller. It does input validation too!

## 8 Reflection

We found this lab to be much more difficult than the last two labs because of the limitations placed on the chassis, which meant that we had to be more careful in our planning and were not always able to work on the lab when we wanted to. It made testing and finding effective work that could be done without the chassis more important. We found ourselves spending a much longer amount of time on mounting than anticipated though we used a simple wood block with a notch in it, which indicated that we should have planned and made more careful measurements beforehand. Additionally, we ran into a lot of problems because we often made the mistake of adjusting more than one variable at a time before testing. We got very confused for a while because we had placed a resistor with the wrong value into our circuit, something that we would have seen had we tested more carefully and thought more critically about what our tests were telling us.

Despite the difficulties of this lab, we weren't afraid to ask for help and got a lot of tips from Maggie, Lucy, and Anisha. Thanks to them and the teaching team, we were finally able to get everything working, and the result was that our robot was able to follow the track beautifully. Hooray!

## 9 Acknowledgements

Thanks to Emily Guthrie for going through the datasheet with us to help us understand the IR circuit and Claire Keum for being the awesome ninja who checked our vehicle off for the lab.

We probably wouldn't have finished on time without numerous suggestions and lots of help from Maggie Jakus, Lucy Wilcox, and Anisha Nakagawa. They helped us debug, provided inspiration for our mounting system, and pointed us in the right direction for figuring out the correct resistor values to use for our circuit. They are the best!

## 10 Source Code

### 10.1 PID Motor Control Code

```
1 #include <Wire.h>
2 #include <Adafruit_MotorShield.h>
3 #include "utility/Adafruit_PWMServoDriver.h"

5 // VARIABLES FOR SERIAL COM
  String validInput = "PIDS"; // string of Inputs (Proportion, Integral, Derivative, Speed)
7 // that can be toggled via Serial
  char paramToSet = '_';
9 boolean readingFirstChar = true;

11 // control parameters that are tunable from serial in
  double P = 0.30;
13 double I = 0;
  double D = 0;
15 byte BASELINE_SPEED = 25;

17 // tested with 0.3 proportion and 25 speed and worked

19 // last time, error accumulator, and last error for I & D
  unsigned long lastTime;
21 double errorSum;
  double lastError;
23
  //pin mappings
25 const byte SENSOR_PIN_0 = A0;
  const byte SENSOR_PIN_1 = A1;
27
  // motorshield initialization
29 Adafruit_MotorShield AFMS = Adafruit_MotorShield();
  Adafruit_DCMotor *motor0 = AFMS.getMotor(1);
31 Adafruit_DCMotor *motor1 = AFMS.getMotor(2);

33 // sensor readings
  byte sensor0Val;
35 byte sensor1Val;

37 void setup() {
  AFMS.begin();
39   Serial.begin(9600);

41   // motor settings (initial state = go forward at baseline speed)
  motor0 -> setSpeed(BASELINE_SPEED);
43   motor1 -> setSpeed(BASELINE_SPEED);
  motor0 -> run(FORWARD);
45   motor1 -> run(FORWARD);

47   Serial.print("Enter the first letter of the parameter you want to tune: ");
  }
49
  void loop() {
51   serialEvent();
  collectSensorValues();
53   setMotorSpeeds();
  }
55
  void collectSensorValues(){
57   sensor0Val = map(analogRead(SENSOR_PIN_0), 0, 1023, 0, 255);
  sensor1Val = map(analogRead(SENSOR_PIN_1), 0, 1023, 0, 255);
```

```

59 }

61 void setMotorSpeeds(){
    int error = sensor0Val - sensor1Val;
63
    int controlCorrection = getPIDCorrection(error);
65
    int motor0Speed = BASELINE_SPEED + (controlCorrection/2);
67 int motor1Speed = BASELINE_SPEED - (controlCorrection/2);

69 motor0 -> setSpeed(motor0Speed);
    motor1 -> setSpeed(motor1Speed);
71 }

73 int getPIDCorrection(int error){
    unsigned long now = millis();
75 double timeChange = (double)(now - lastTime);

77 // calculating the integral
    errorSum += (speedDiff * timeChange);
79
    // calculating the derivative
81 int errorDiff = (speedDiff - lastError) / timeChange;

83 // return the post-PID corrected speed
    return P*speedDiff + I*errorSum + D*errorDiff;
85 }

87 void serialEvent() {
    while (Serial.available()) { // get the new byte
89         if(readingFirstChar){ // if searching for valid input
            char inChar = (char)Serial.read();
91             // check if input is one of our valid inputs (P, I, D, or S)
            if(validInput.indexOf(inChar) != -1){
93                 readingFirstChar = false;
                    paramToSet = inChar;
95                 Serial.println(inChar);
                    Serial.print("Now enter a value: ");
97             }
            } else {
99                 float val = Serial.parseFloat();
                    Serial.println(val);
101                 setParameter(paramToSet, val);
                    readingFirstChar = true;
103                 Serial.print("Enter the first letter of the parameter you want to tune: ");
            }
        }
105     }
107 }

void setParameter(char p, double val){ // set the chosen parameter with the new value inputted
109     switch(p){
        case 'P':
111         if(isValidRange(0,1, val)){
            P = val;
113         }
            break;
115         case 'I':
            if(isValidRange(0,1, val)){
117                 I = val;
            }
            break;
119         case 'D':

```

```

121         if(isValidRange(0,1, val)){
            D = val;
123         }
            break;
125     case 'S':
        if(isValidRange(0,255, val)){
127             BASELINE_SPEED = val;
        }
129         break;
    }
131     Serial.println(P);
    Serial.println(I);
133     Serial.println(D);
    Serial.println(BASELINE_SPEED);
135 }

137 boolean isValidRange(double lower, double upper, double val){
    if(val < lower || val > upper){
139         Serial.print(" Invalid input! Enter a value between ");
        Serial.print(lower);
141         Serial.print(" and ");
        Serial.println(upper);
143         return false;
    }
145     return true;
}

147 byte doubleToByte(double val){
149     return (byte)(int) val;
}

```