Lab 5: PID Controller Design

ECE 564: Fundamentals of Autonomous Robots Lab

Team 1: Jacob Cassady, Chase Crutcher, and Olalekan Olakitan Olowo

September 23, 2019

The group members have worked together and face-to-face at all stages of this project work.
The contributions of members to the report and to the codes are equal.
(Initials of group members)

1. Introduction

The goal of this lab is to carry out a wall following experiment using either the E.T sensor or IR sensor as a wall distance sensor for the Demo-bot. The procedure will include designing an on-off control as done in previous labs and then designing the PID (Proportional Integral Derivative) Controller. The lab activity is also meant to involve the use of stored sensory data extracted from the Wallaby to plot performance graph of our system.

2. Lab Parts

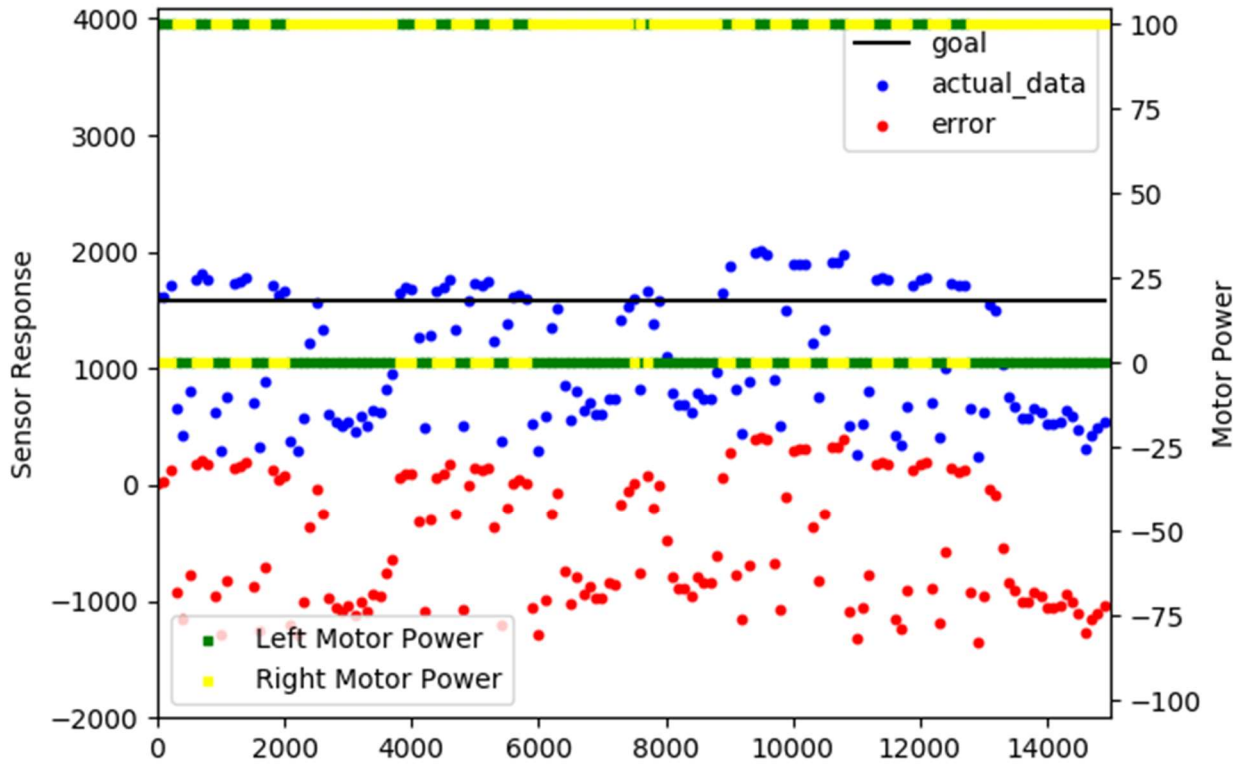
2.1. Primitive Wall Follow

2.1.1. Quick Turn Wall Follow

2.1.1.1. Algorithm

```
void primitiveWallFollow() {
    ix = 0;

    while(ix < 151) {
        // Get data from sensor.
        int wall = analog(LEFT_WALL);
        printf("goal is %d; wall is %d\n", goal, wall);

        if (wall < goal) {
            // Go left
            demoMotor(LEFT_MOTOR, RIGHT_MOTOR, 0, 100);
            // Store motor powers
            leftMotorPowers[ix] = 0;
            rightMotorPowers[ix] = 100;
        } else {
            // Go right
            demoMotor(LEFT_MOTOR, RIGHT_MOTOR, 100, 0);
            // Store motor powers
            leftMotorPowers[ix] = 100;
            rightMotorPowers[ix] = 0;
        }

        responses[ix] = wall;
        ix++;

        // 10 data points a second.
        msleep(100L);
    }
    // Shut off motors.
    ao();
    // Display results.
    printData("response", responses, DATA_ARRAY_SIZE);
    printData("leftMotorPower", leftMotorPowers, DATA_ARRAY_SIZE);
    printData("rightMotorPower", rightMotorPowers, DATA_ARRAY_SIZE);
}
```

2.1.1.2. Experiment



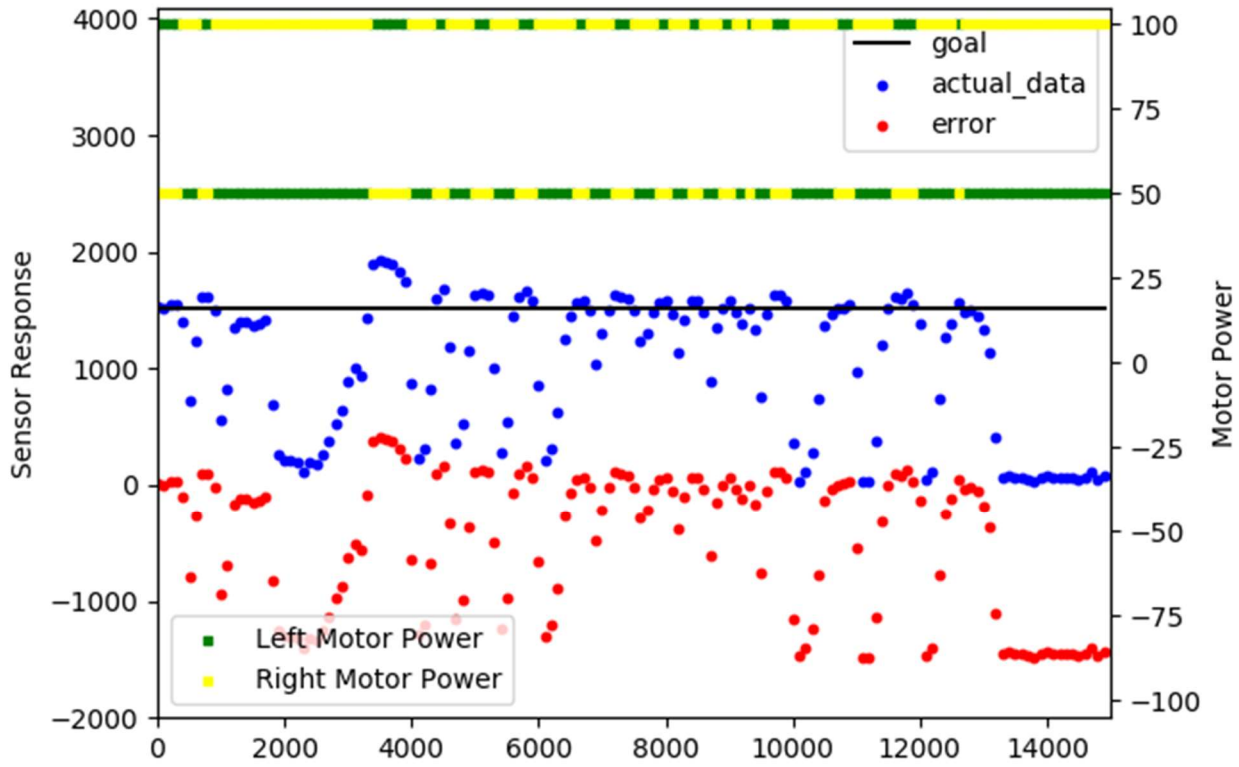
Source	Min	Max	Average	Standard Deviation
Sensor Response	239	2005	1073.48	546.1335
Left Motor Data	0	100	30.66667	46.26545
Right Motor Data	0	100	69.33333	46.26545
Total Error	92474			

2.1.2. Primitive Gentle Turn Wall Follow

2.1.2.1. Algorithm

```
void primitiveGentleWallFollow() {
    ix = 0;

    while(ix < 151) {
        // Get data from sensor.
        int wall = analog(LEFT_WALL);
        printf("goal is %d; wall is %d\n", goal, wall);

        if (wall < goal) {
            // Go left
            demoMotor(LEFT_MOTOR, RIGHT_MOTOR, 50, 100);
            // Store motor powers
            leftMotorPowers[ix] = 50;
            rightMotorPowers[ix] = 100;
        } else {
            // Go right
            demoMotor(LEFT_MOTOR, RIGHT_MOTOR, 100, 50);
            // Store motor powers
            leftMotorPowers[ix] = 100;
            rightMotorPowers[ix] = 50;
        }

        // Store sensor response
        responses[ix] = wall;
        // increment data index.
        ix++;

        // 10 data points a second.
        msleep(100L);
    }
    // Shut off motors.
    ao();
    // Display results.
    printData("response", responses, DATA_ARRAY_SIZE);
    printData("leftMotorPower", leftMotorPowers, DATA_ARRAY_SIZE);
    printData("rightMotorPower", rightMotorPowers, DATA_ARRAY_SIZE);
}
```

2.1.2.2. Experiment



Source	Min	Max	Average	Standard Deviation
Sensor Response	32	1922	1020.153	613.0113
Left Motor Data	50	100	64.33333	22.68597
Right Motor Data	50	100	85.66667	613.0113
Total Error	83843			

2.1.3. Analysis

What are the differences between this figure and figure of part #9?

The figure we produced is less centered around the goal then the figure of part #9.

Which one works better? Quick turn or gentle turn?

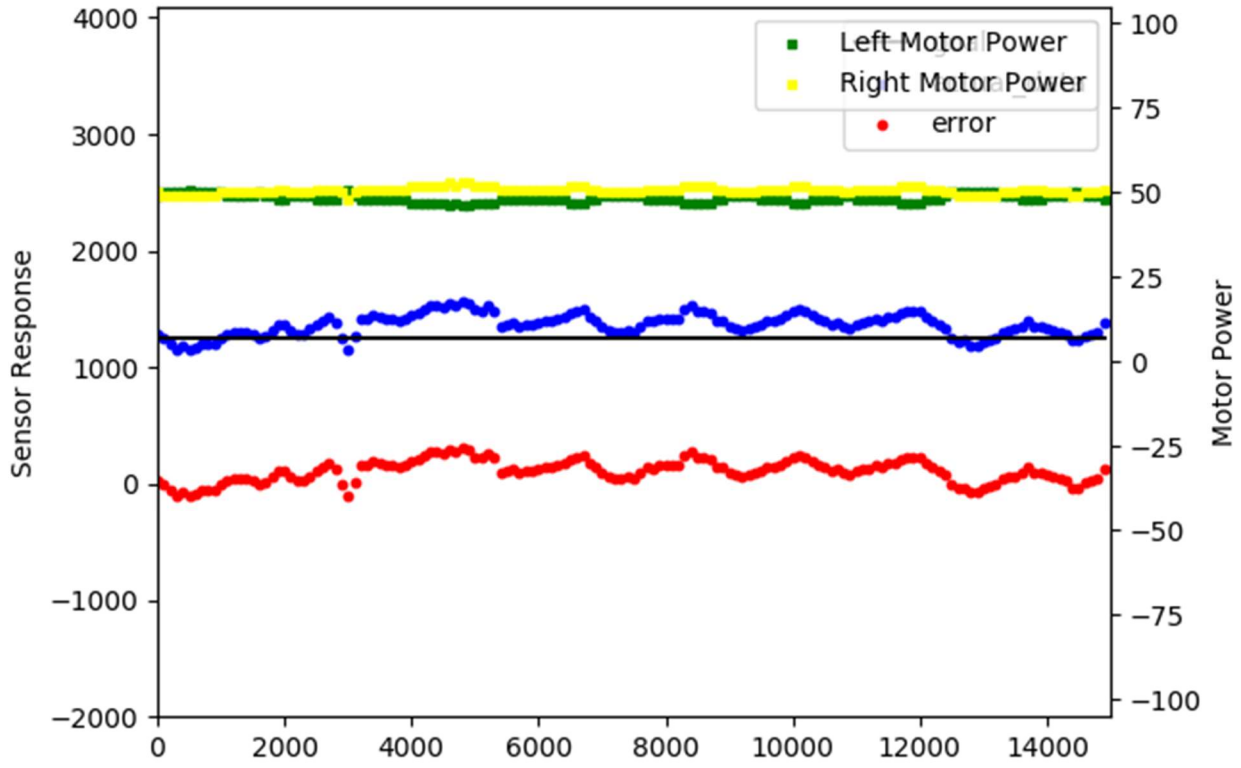
The Gentle Turn algorithm had a lower total error than the quick turn algorithm and therefore works better. It caused more oscillations in the sensor response graph but kept both motors going at a higher rate; consequently, moving forward more.

2.2. Wall Following with P (Proportional) Controller Primitive Wall Follow

2.2.1. Algorithm

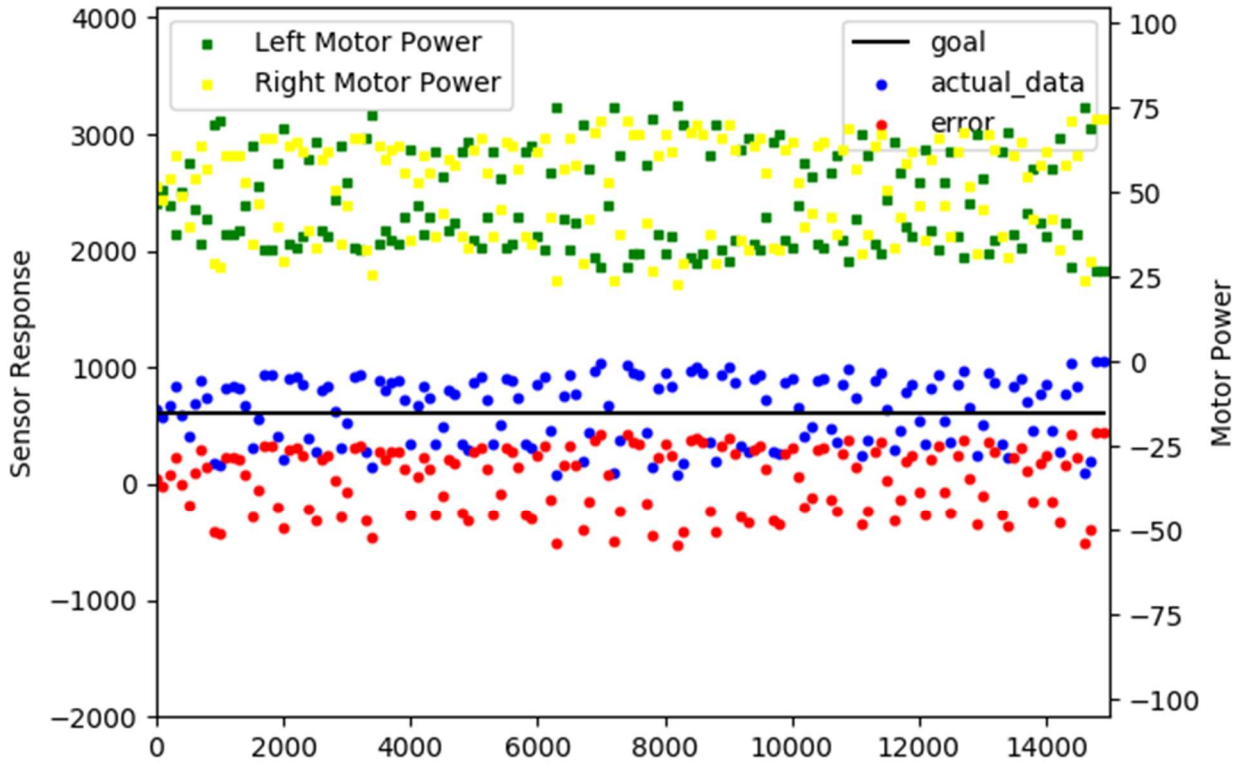
```
void proportionalController(float proportionalGain) {
  ix = 0;
  int error = 0;
  int leftMotorPower = 0;
  int rightMotorPower = 0;

  while(ix < 151) {
      // Get data from sensor.
      int wall = analog(LEFT_WALL);
      printf("goal is %d; wall is %d\n", goal, wall);
      // Calculate error.
      error = goal - wall;

      // Update motor powers considering proportionalGain.
      leftMotorPower = (float) 50 + proportionalGain*(float) error;
      rightMotorPower = (float) 50 - proportionalGain*(float) error;

      // Ensure powers are bound by [-100, 100]
      leftMotorPower = bind(leftMotorPower, -100, 100);
      rightMotorPower = bind(rightMotorPower, -100, 100);

      // Update motor speed.
      demoMotor(LEFT_MOTOR, RIGHT_MOTOR, leftMotorPower, rightMotorPower);

      // Store response and motor powers.  Update indices
      responses[ix] = wall;
      leftMotorPowers[ix] = leftMotorPower;
      rightMotorPowers[ix] = rightMotorPower;
      ix++;

      // 10 data points a second.
      msleep(100L);
  }
  ao();
  printData("response", responses, DATA_ARRAY_SIZE);
  printData("leftMotorPower", leftMotorPowers, DATA_ARRAY_SIZE);
  printData("rightMotorPower", rightMotorPowers, DATA_ARRAY_SIZE);
}
```

2.2.2. Kp = 0.01



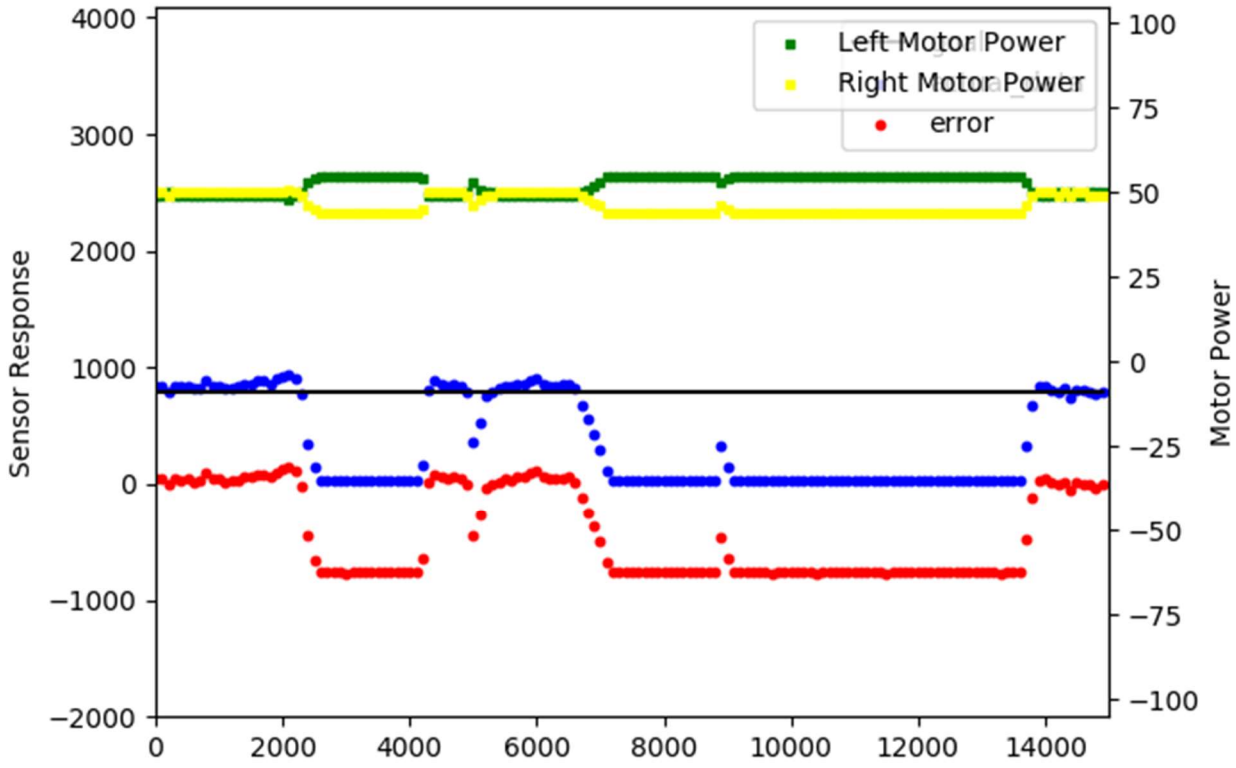
Source	Min	Max	Average	Standard Deviation
Sensor Response	1147	1569	1368.56	95.6671
Left Motor Data	46	51	48.36	1.031582
Right Motor Data	48	53	50.67333	0.986463
Total Error	19126			

2.2.3. Kp = 0.05



Source	Min	Max	Average	Standard Deviation
Sensor Response	77	1051	645.32	279.803
Left Motor Data	27	76	47.38	13.9648
Right Motor Data	23	72	51.66667	13.98497
Total Error	38588			

2.2.4. Kp = 0.0075



Source	Min	Max	Average	Standard Deviation
Sensor Response	30	944	370.4333	385.5259
Left Motor Data	48	55	52.55333	2.803489
Right Motor Data	44	51	46.44667	2.803489
Total Error			69285	

2.2.5. Analysis

Do you see response similar to figures 5.10 through 5.13 given in the book?

Yes the response when kp = 0.05 is similar to the responses in figures 5.10 through 5.13.

Do you think it is always better to have a larger proportional gain P as a controller?

No. It is not always better to have a larger P. When P was too large, the robot had too much overshoot and even stopped moving forward when P was large enough. P was found experimentally to be best when around 0.01.

Does your controller output get into saturation for at least one value of P?

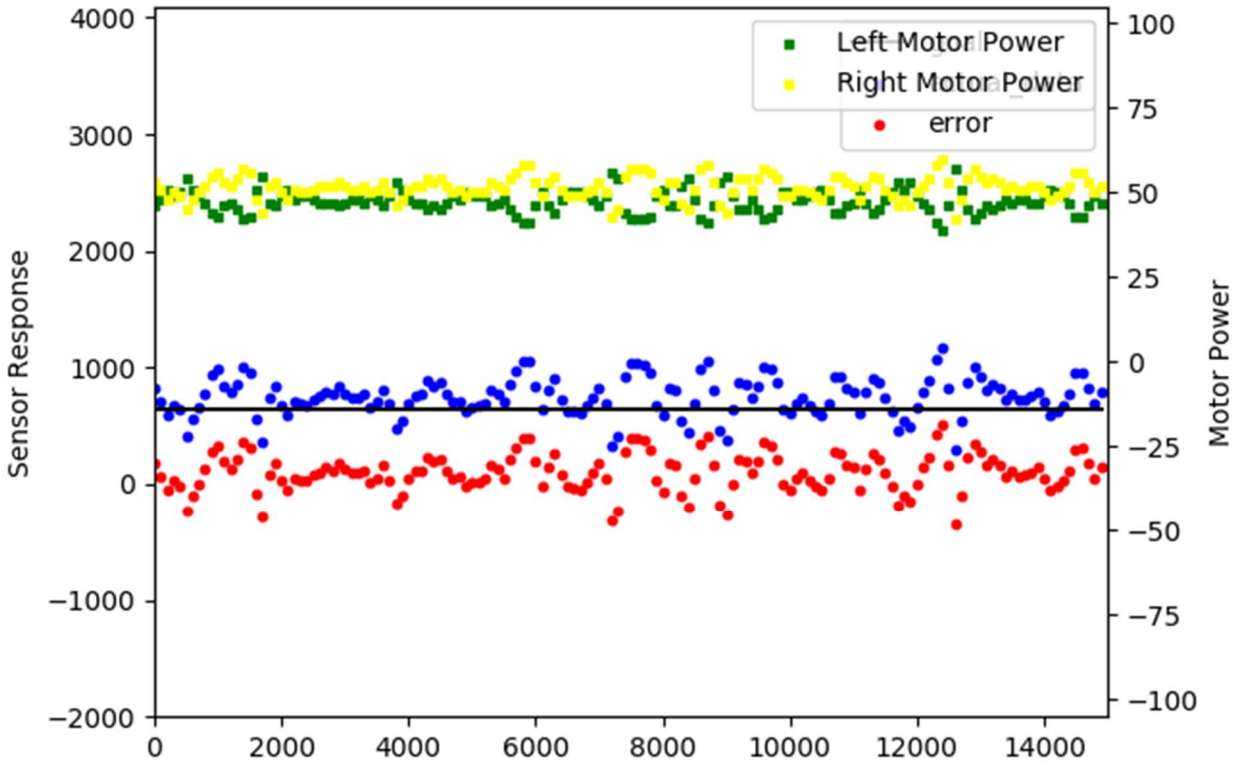
No. The output never had a saturation value for at least one value of P shown; although, when P was large enough, we saw results that would be consistent with a motor power reaching saturation.

2.3. Wall Following with PD (Proportional and Derivative) Controller

2.3.1. Algorithm

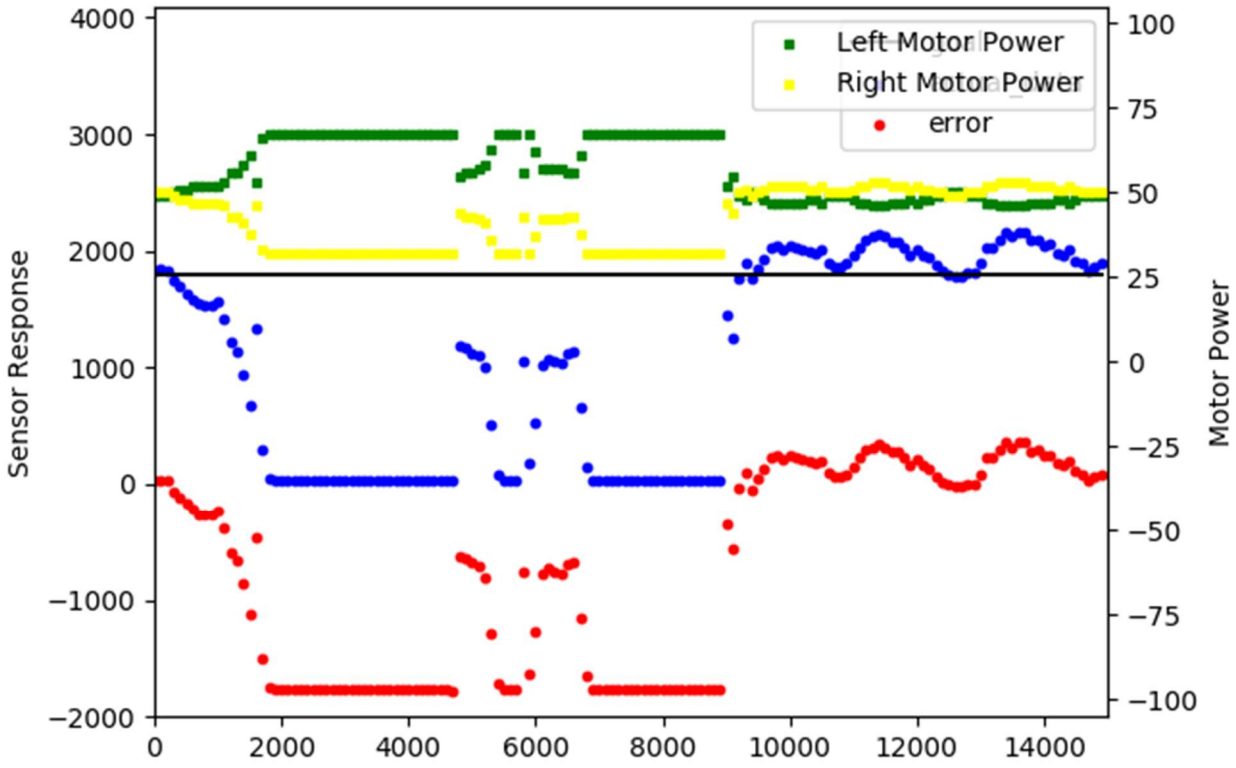
```
void proportionalDerivativeController(int sensorControl, int sensorGoal, float proportionalGain, float derivativeGain) {
  int ix = 0;
  int responses[DATA_ARRAY_SIZE] = {0};
  int leftMotorPowers[DATA_ARRAY_SIZE] = {0};
  int rightMotorPowers[DATA_ARRAY_SIZE] = {0};
  int error = 0;
  int lastError = 0;
  float derivativeError = 0;
  int leftMotorPower = 0;
  int rightMotorPower = 0;

  while(ix < 151) {
      int sensorResponse = analog(sensorControl);
      printf("sensorGoal is %d; sensorResponse is %d\n", sensorGoal, sensorResponse);
      error = sensorGoal - sensorResponse;
      derivativeError = (float) error - (float) lastError) / 0.1

      leftMotorPower = (float) 50 + proportionalGain*(float) error + derivativeGain*derivativeError;
      rightMotorPower = (float) 50 - (proportionalGain*(float) error + derivativeGain*derivativeError);

      // Ensure powers are bound by [-100, 100]
      leftMotorPower = bind(leftMotorPower, -100, 100);
      rightMotorPower = bind(rightMotorPower, -100, 100);

      demoMotor(LEFT_MOTOR, RIGHT_MOTOR, leftMotorPower, rightMotorPower);

      responses[ix] = sensorResponse;
      leftMotorPowers[ix] = leftMotorPower;
      rightMotorPowers[ix] = rightMotorPower;
      ix++;

      // 10 data points a second.
      msleep(100L);
  }
  ao();
  printData("response", responses, DATA_ARRAY_SIZE);
  printData("leftMotorPower", leftMotorPowers, DATA_ARRAY_SIZE);
  printData("rightMotorPower", rightMotorPowers, DATA_ARRAY_SIZE);
}
```

2.3.2. Kp = 0.02, Kd = 0.000076



Source	Min	Max	Average	Standard Deviation
Sensor Response	304	1167	1167	160.4466
Left Motor Data	39	57	47.42667	3.27354
Right Motor Data	42	60	51.57333	3.27354
Total Error	22791			

2.3.3. Kp = 0.01, Kd = 0.0001



Source	Min	Max	Average	Standard Deviation
Sensor Response	28	2166	2166	877.4567
Left Motor Data	46	67	56.82	8.745522
Right Motor Data	32	53	42.18	8.745522
Total Error			131171	

2.3.4. Analysis

Do you see response similar to Figures 5.14 through 5.15 given in the book?

Yes. The response when kp = 0.02 and kd = 0.000076 is similar to Figures 5.14 through 5.15.

What is the role of D gain?

The role of derivative gain is to decrease the overshoot and the setting time.

Does it improve system response?

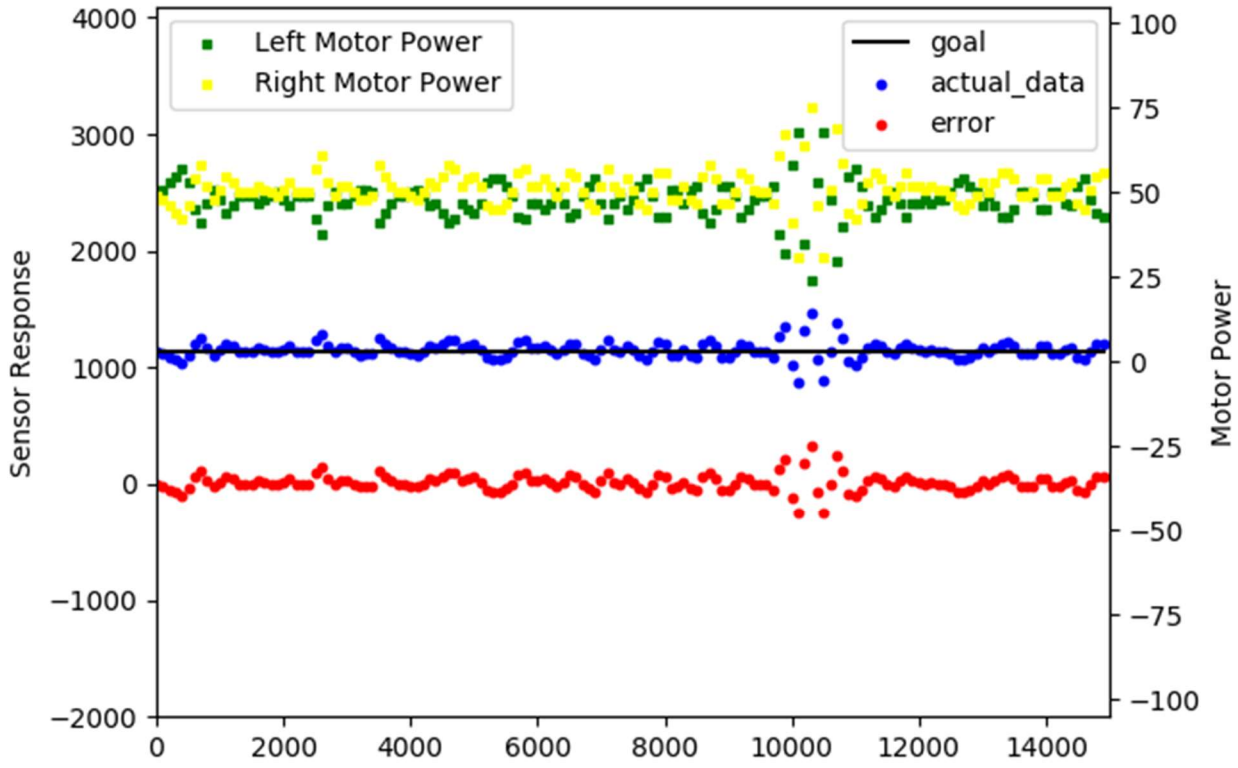
In our case, the derivative gain did not improve the system response although it may be able to be further dialed to improve the system response. There is some overshoot present in the proportional gain graphs; consequently, there is some amount of derivative gain that would likely improve system response.

2.4. Wall Following with PI (Proportional and Integral) Controller

2.4.1. Algorithm

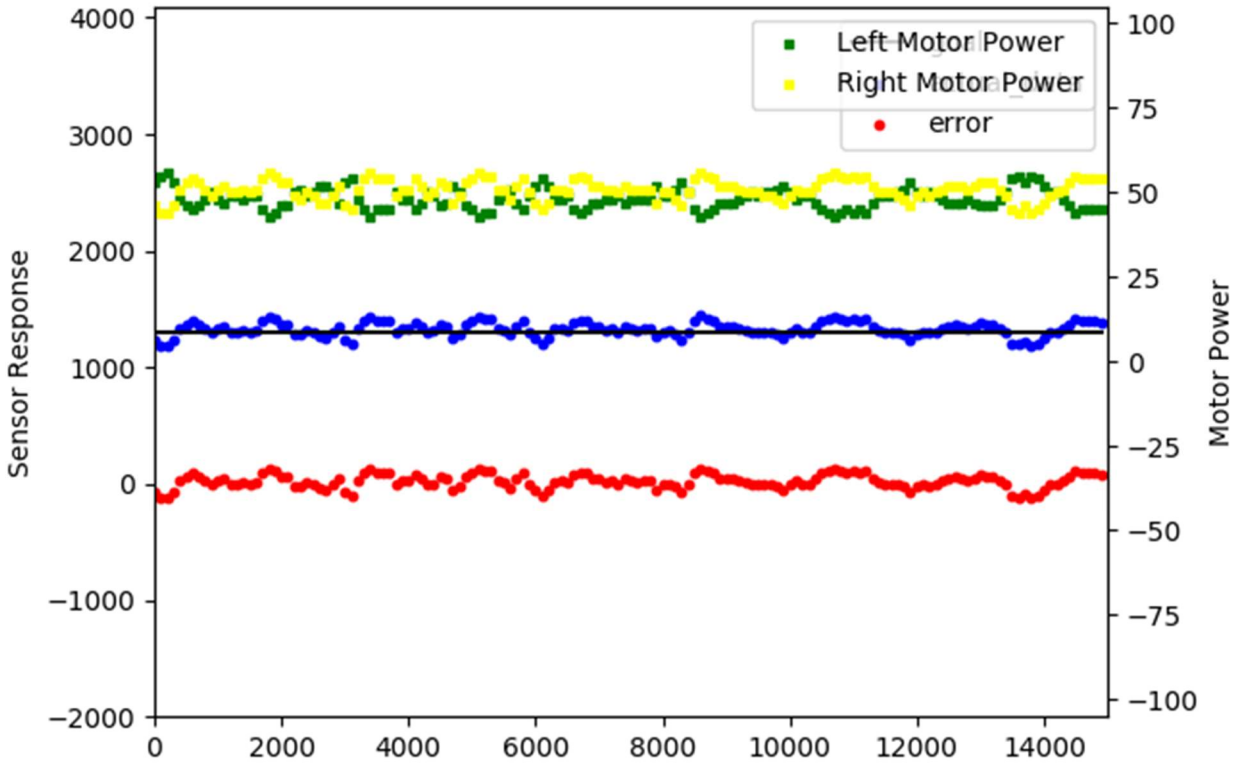
```
void proportionalIntegralController(int sensorControl, int sensorGoal, float proportionalGain, float integralGain) {
  int ix = 0;
  int responses[DATA_ARRAY_SIZE] = {0};
  int leftMotorPowers[DATA_ARRAY_SIZE] = {0};
  int rightMotorPowers[DATA_ARRAY_SIZE] = {0};
  int error = 0;
  int integralError = 0;
  int leftMotorPower = 0;
  int rightMotorPower = 0;

  while(ix < 151) {
      int sensorResponse = analog(sensorControl);
      printf("sensorGoal is %d; sensorResponse is %d\n", sensorGoal, sensorResponse);
      error = sensorGoal - sensorResponse;
      integralError += error;

      leftMotorPower = (float) 50 + proportionalGain*(float) error + integralGain*integralError);
      rightMotorPower = (float) 50 - (proportionalGain*(float) error + integralGain*integralError);

      // Ensure powers are bound by [-100, 100]
      leftMotorPower = bind(leftMotorPower, -100, 100);
      rightMotorPower = bind(rightMotorPower, -100, 100);

      demoMotor(LEFT_MOTOR, RIGHT_MOTOR, leftMotorPower, rightMotorPower);

      responses[ix] = sensorResponse;
      leftMotorPowers[ix] = leftMotorPower;
      rightMotorPowers[ix] = rightMotorPower;
      ix++;

      // 10 data points a second.
      msleep(100L);
  }
  ao();
  printData("response", responses, DATA_ARRAY_SIZE);
  printData("leftMotorPower", leftMotorPowers, DATA_ARRAY_SIZE);
  printData("rightMotorPower", rightMotorPowers, DATA_ARRAY_SIZE);
}
```

2.4.2. Kp = 0.075, Ki = 0.0005



Source	Min	Max	Average	Standard Deviation
Sensor Response	881	1466	1466	69.89801
Left Motor Data	24	68	47.72	5.325789
Right Motor Data	31	75	51.28	5.325789
Total Error			7330	

2.4.3. Kp = 0.05, Ki = 0.0001



Source	Min	Max	Average	Standard Deviation
Sensor Response	1184	1443	1443	59.43443
Left Motor Data	43	56	48.16667	2.941069
Right Motor Data	44	56	50.87333	2.913281
Total Error	7748			

2.4.4. Analysis

What is the role of I gain?

The role of integral gain is to decrease the rise time and eliminate the steady-state error. It also causes the overshoot to increase and the setting time to increase.

Does it improve system response?

The integral gain improved the system response in both of the experiments performed. The steady-state error was near eliminated.

Do you think large values of I is better or small?

From our experiments, very small values of I are better than large values of I. When I was large, the motor responses would get caught in feedback loops such as attempting to turn right when already to the right of the goal value.

Is the response of the system better than the PD controller?

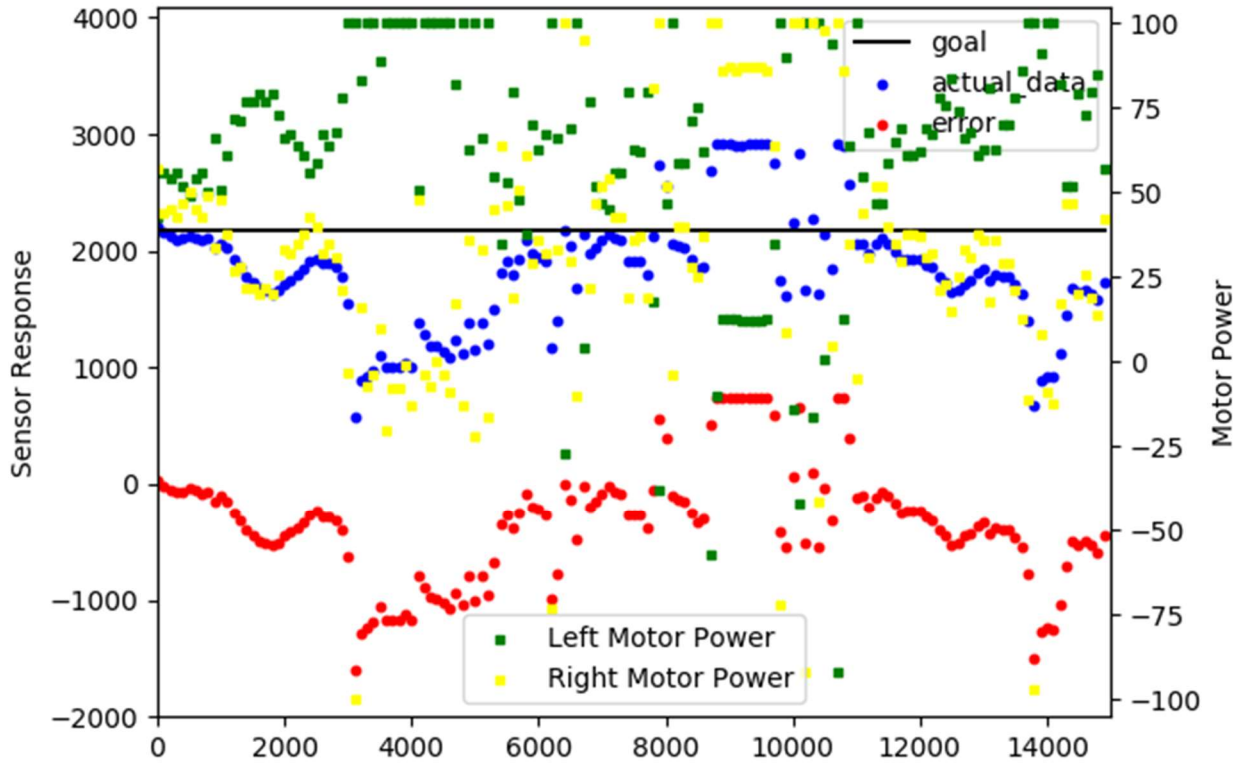
The system response is much better than the PD controller. The total absolute error is larger on all PD experiments when compared to PI experiments.

2.5. Wall Following with PID (Proportional, Integral, and Derivative) Controller

2.5.1. Algorithm

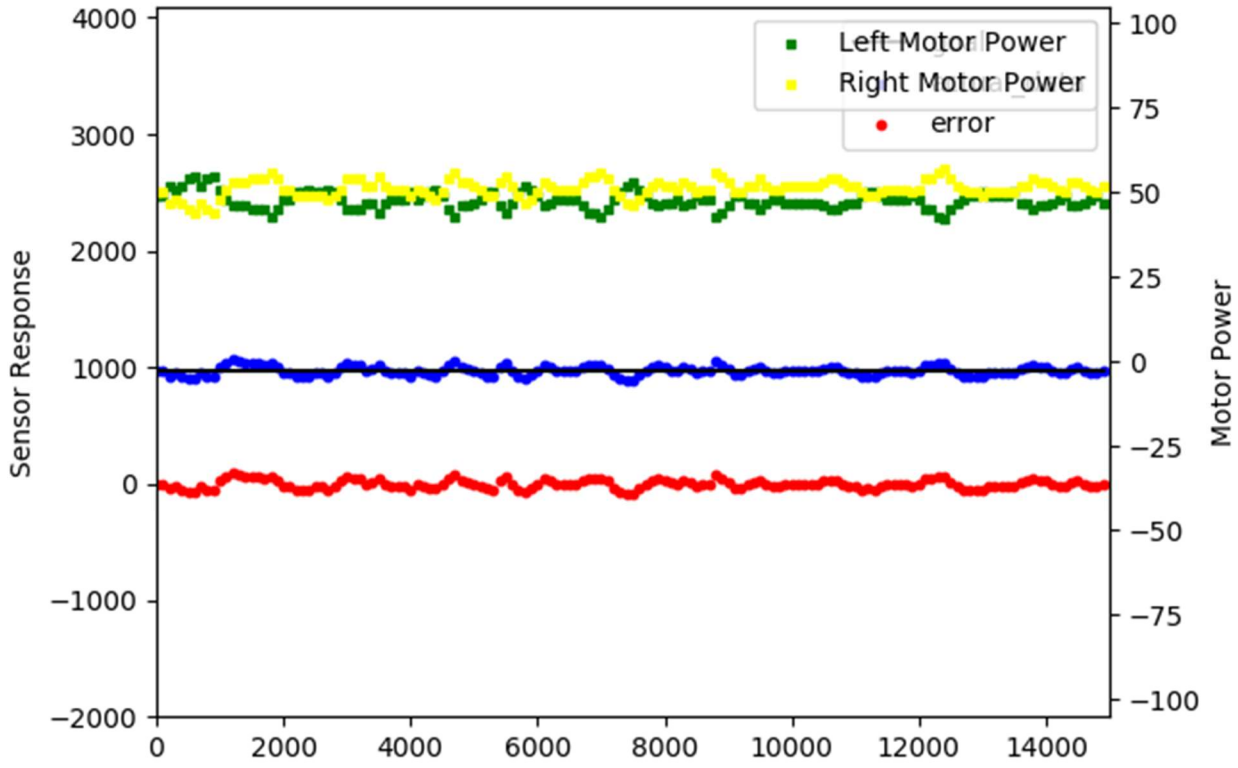
```
void proportionalIntegralDerivativeController(int sensorControl, int sensorGoal, float proportionalGain, float integralGain, float derivativeGain) {
  int ix = 0;
  int responses[DATA_ARRAY_SIZE] = {0};
  int leftMotorPowers[DATA_ARRAY_SIZE] = {0};
  int rightMotorPowers[DATA_ARRAY_SIZE] = {0};
  int error = 0;
  int lastError = 0;
  int integralError = 0;
  float derivativeError = 0;
  int leftMotorPower = 0;
  int rightMotorPower = 0;

  while(ix < 151) {
      int sensorResponse = analog(sensorControl);
      printf("sensorGoal is %d; sensorResponse is %d\n", sensorGoal, sensorResponse);
      error = sensorGoal - sensorResponse;
      derivativeError = (float) error - (float) lastError) / 0.1
      integralError += error;

      leftMotorPower = (float) 50 + proportionalGain*(float) error + derivativeGain*derivativeError + integralGain*integralError);
      rightMotorPower = (float) 50 - (proportionalGain*(float) error + derivativeGain*derivativeError + integralGain*integralError);

      // Ensure powers are bound by [-100, 100]
      leftMotorPower = bind(leftMotorPower, -100, 100);
      rightMotorPower = bind(rightMotorPower, -100, 100);

      demoMotor(LEFT_MOTOR, RIGHT_MOTOR, leftMotorPower, rightMotorPower);

      responses[ix] = sensorResponse;
      leftMotorPowers[ix] = leftMotorPower;
      rightMotorPowers[ix] = rightMotorPower;
      ix++;

      // 10 data points a second.
      msleep(100L);
  }
  ao();
  printData("response", responses, DATA_ARRAY_SIZE);
  printData("leftMotorPower", leftMotorPowers, DATA_ARRAY_SIZE);
  printData("rightMotorPower", rightMotorPowers, DATA_ARRAY_SIZE);
}
```

2.5.2. Kp = 0.05, Ki = 0.00001, Kd = 0.01



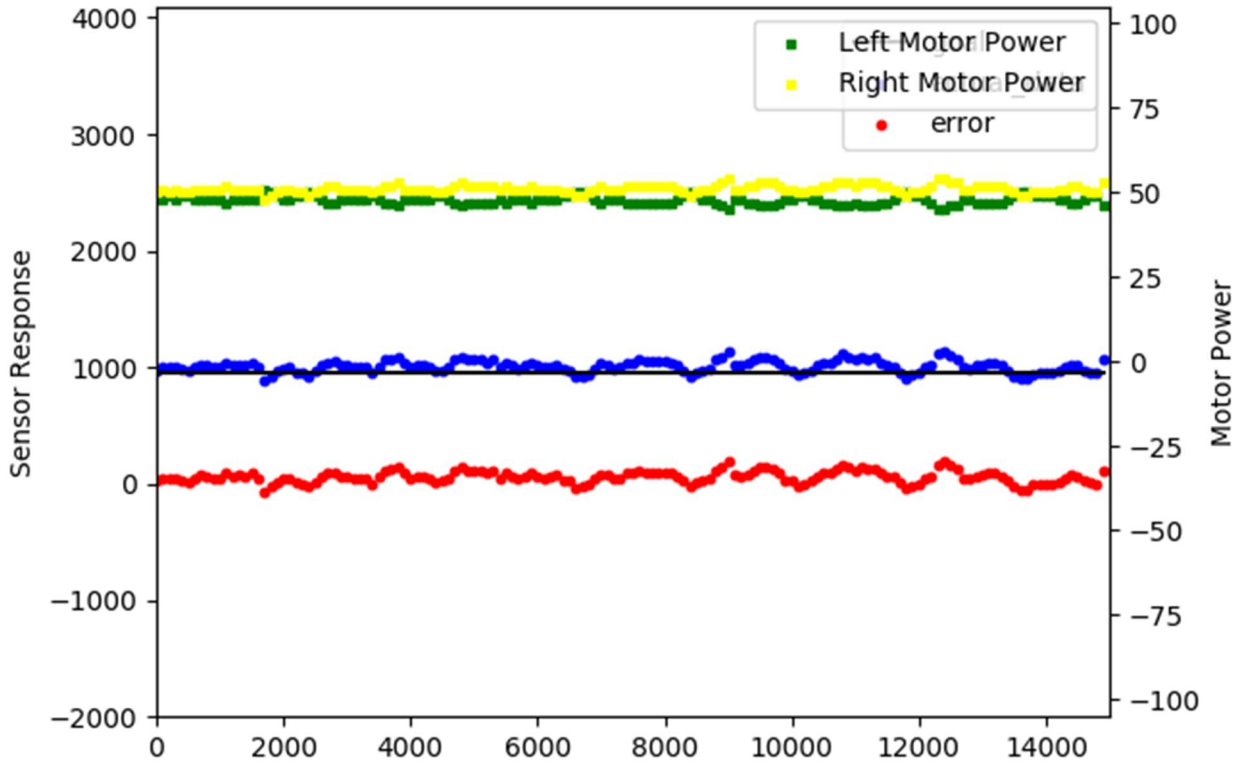
Source	Min	Max	Average	Standard Deviation
Sensor Response	577	2917	2917	509.4527
Left Motor Data	-92	100	61.96	34.02583
Right Motor Data	-100	100	30.92667	37.06426
Total Error	73009			

2.5.3. Kp = 0.002, Ki = 0.00001, Kd = 0.02



Source	Min	Max	Average	Standard Deviation
Sensor Response	882	1067	1067	38.02744
Left Motor Data	42	55	47.74667	2.441769
Right Motor Data	44	57	51.26667	2.429314
Total Error	4535			

2.5.4. Kp = 0.02, Ki = 0.000076, Kd = 0.0002



Source	Min	Max	Average	Standard Deviation
Sensor Response	885	1142	1142	50.42532
Left Motor Data	45	51	47.79333	1.125045
Right Motor Data	48	54	51.20667	1.125045
Total Error	10358			

2.5.5. Analysis

Does combining all three terms improve the response of your system?

Yes, combining all three terms can improve the response of the system. The best performing experiment had values kp = 0.002, ki = 0.00001, kd = 0.02. This experiment had a small overshoot and minimal steady-state error. The total absolute error of this experiment was 4535, about 1800 less than the best proportional integral experiment.

3. Conclusion

In this particular lab experiment and activity, the Demobot was built and programmed implementing the assigned code and formula given for the different combinations of the PID controllers in the different parts of the lab work. As the Demobot moved along the wall it provided sensor data which served as the feedback to adjust and update the system according. The performance graph provided within the report gave the insight of how the Demobot responded to the different configuration of the PID controllers.

4. Suggestions

It is suggested that the E.T sensor be provided alongside the IR sensor to determine which is more effective while implementing the different PID controllers.