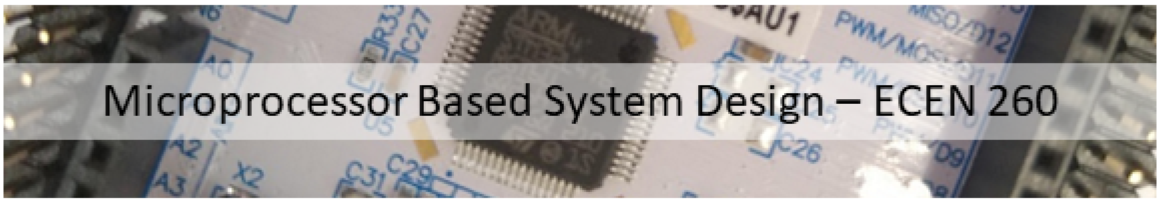

ECEN 260 - Final Project

PID Ball and Beam Controller

Samuel Fowler

Instructor: Brother Allred
December 10, 2024

Contents

List of Tables

List of Figures

1 Project Introduction

This report will explore the details of my final project for my microprocessors class at Brigham Young University–Idaho.

For my final project, I constructed a PID control system that maintains the position of a ball using a beam, servo, and distance sensor. I applied techniques in PWM, I2C protocol, and digital I/O communication with a sensor. In this report, we will explore the functionality and limits of this system. I will go into detail regarding the construction and design. Figure 1 shows the final project in operation.

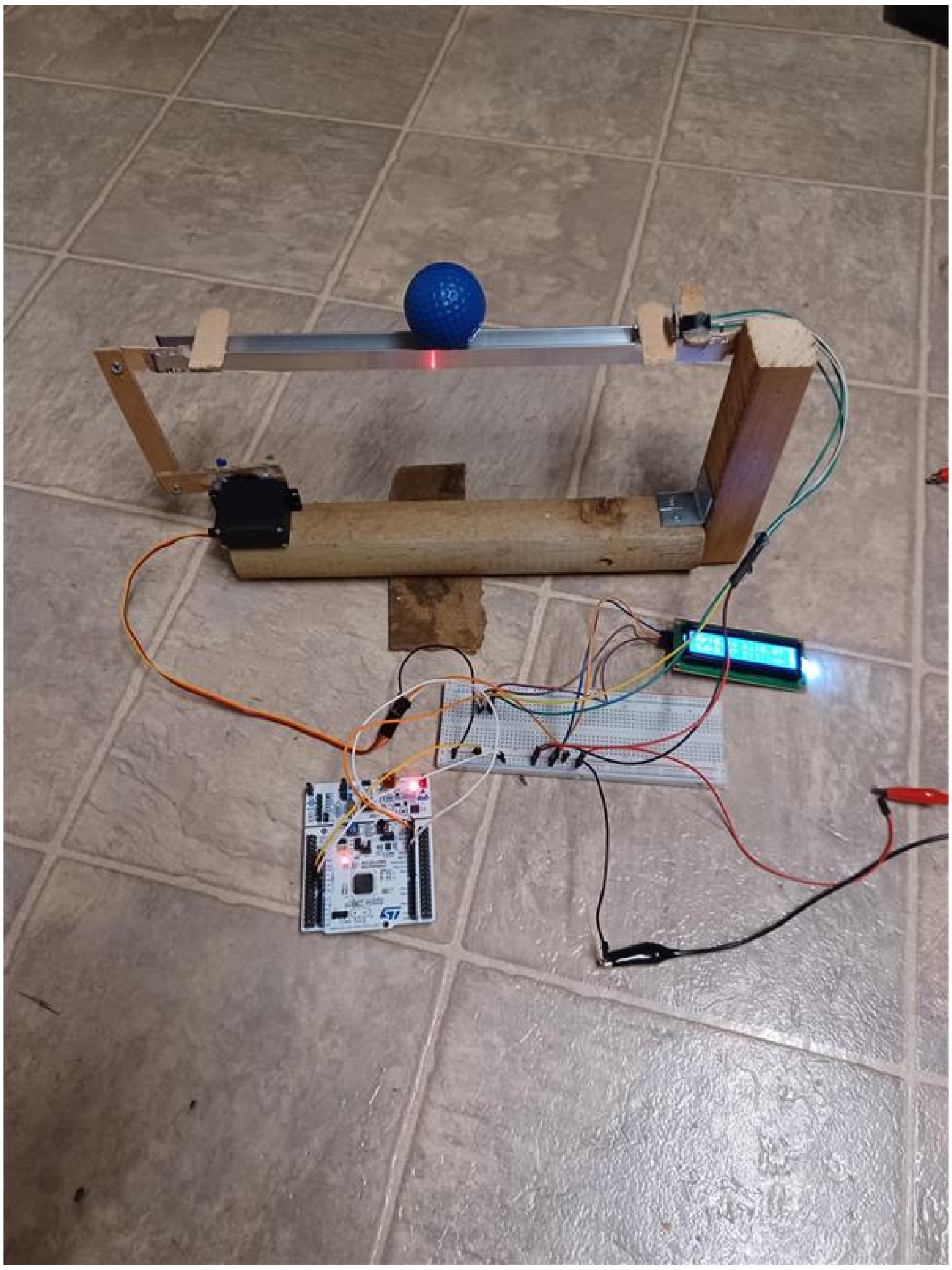

Figure 1: Project in operation

1.1 Design Overview

The PID ball-beam balance project was completed to keep a ball at a certain set point based on readings from an IR distance sensor. This operation is applicable in industries requiring precision testing of equipment or sensors. The concepts of this design are applicable in most basic control systems. Such as cruise in cars, movement in robotic arms, flow control in hydraulic pumps etc. The specific needs that were met in my project focus on the distance application. This device could be used to test the accuracy of various distance sensors and measure their performance against various materials.

Various factors were considered as this project was completed. Economically, this project is quite cost effective. Its construction is easily repeatable with cheap and robust components. The software prohibits the servo from rotating too far. This increases safety and minimizes component failures. There was no social or cultural focus in this project, making it a perfect candidate for multicultural teams and settings. There is no significant impact on global factors, environmental factors, or public health factors.

Implementing an STM32 microcontroller to facilitate a PID control system in this project involved 3 different important parts. We had a PWM signal sent to a servo, an I2C display showing important parameters and data, and a digital I/O sensor to send distance values to the microcontroller over I2C. The servo controls the height of one side of the beam, thus rolling the ball from one side to the other. I had to implement a PWM limit so that the servo didn't rotate too far and damage the setup. The display prints updated sensor readings every second. This allows us to see where the ball is in real time and know how effective the current control parameters are. It also prints the current PID control parameters for reference. The IR distance sensor is on the same I2C bus as the display. It simply returns distance values to the STM32 which then implements the control logic to move the ball.

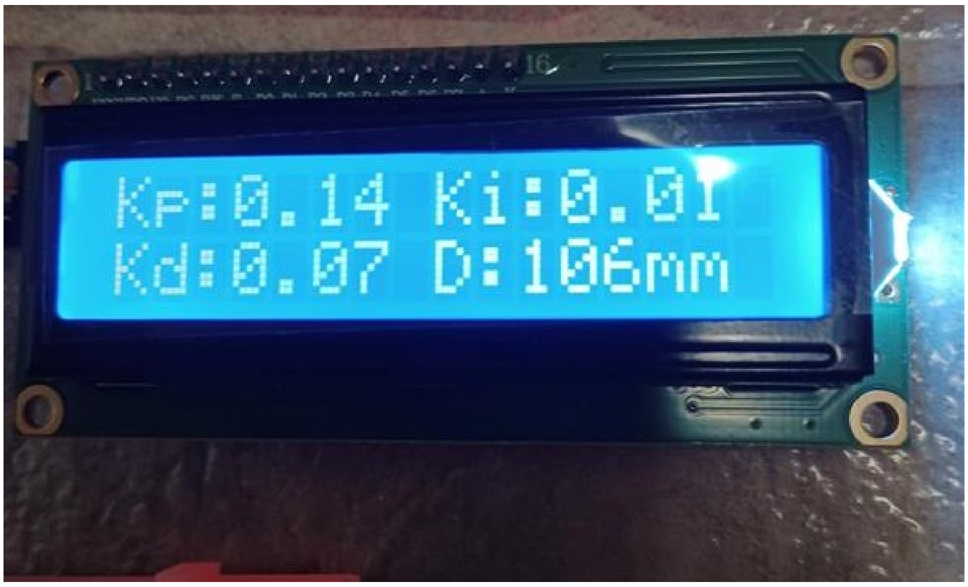
```
Kp:0.14 Ki:0.01
Kd:0.07 D:106mm
```

Figure 2: Display during operation

2 Specifications

The frame for this project was constructed with scrap wood from an old bunk bed. The smaller pieces of wood were from clip boards and popsicle sticks. The main beam was an aluminum rail from the hardware store. I simply used hot glue to hold most of these parts together.

I found a conservative minimum and maximum position for the servo and coded those limits into the program. I also found a suitable wall adapter that outputs 5v at 3.5amps. This was more than adequate to power the servo and other components. I set the timer prescaler and counter period appropriately to get a 50Hz signal. In my case, I chose 1599 for the prescaler and 999 for the counter period. These values worked well and my servo responded with great performance.

The distance sensor runs on I2C and has a different address than the display. Using addressing I didn't have any problem running the two on the same bus. To communicate and get data from the sensor, I used a library created by Ahmet Batuhan Günaltay. Here is a link to his GitHub `https://github.com/Squieler/VL53L0X---STM32-HAL`. After removing some includes not compatible with the NUCLEO-L476RG, I was able to communicate well with the IR sensor.

The display is a common QAPASS I2C display. The code to get that one working came mostly from a lab tutorial we completed in class a few weeks ago. It integrated well with the other components.

Once the desired PID values have been programmed into the micro-controller, it is ready to begin operation. The STM32 gets readings from the distance sensor and adjusts the beam with the servo. The PID values and current distance reading are sent to the display each second (as seen in figure 2 in the design overview). The device does not yet have the functionality to reprogram the PID values without a code update from the computer. However, a future iteration using a membrane keypad could eliminate this drawback.

2.1 Parts List

- NUCLEO-L476RG
- Wall adapter (5V 3.5Amps)
- QAPASS I2C display
- MG996R servo motor
- VL53L0 IR distance sensor
- The breadboard and various jumper wires

'For specifics on the wiring setup, see the schematic in Section 3."

3 Schematic

See Figure 3 for the schematic of the wiring for this Project.

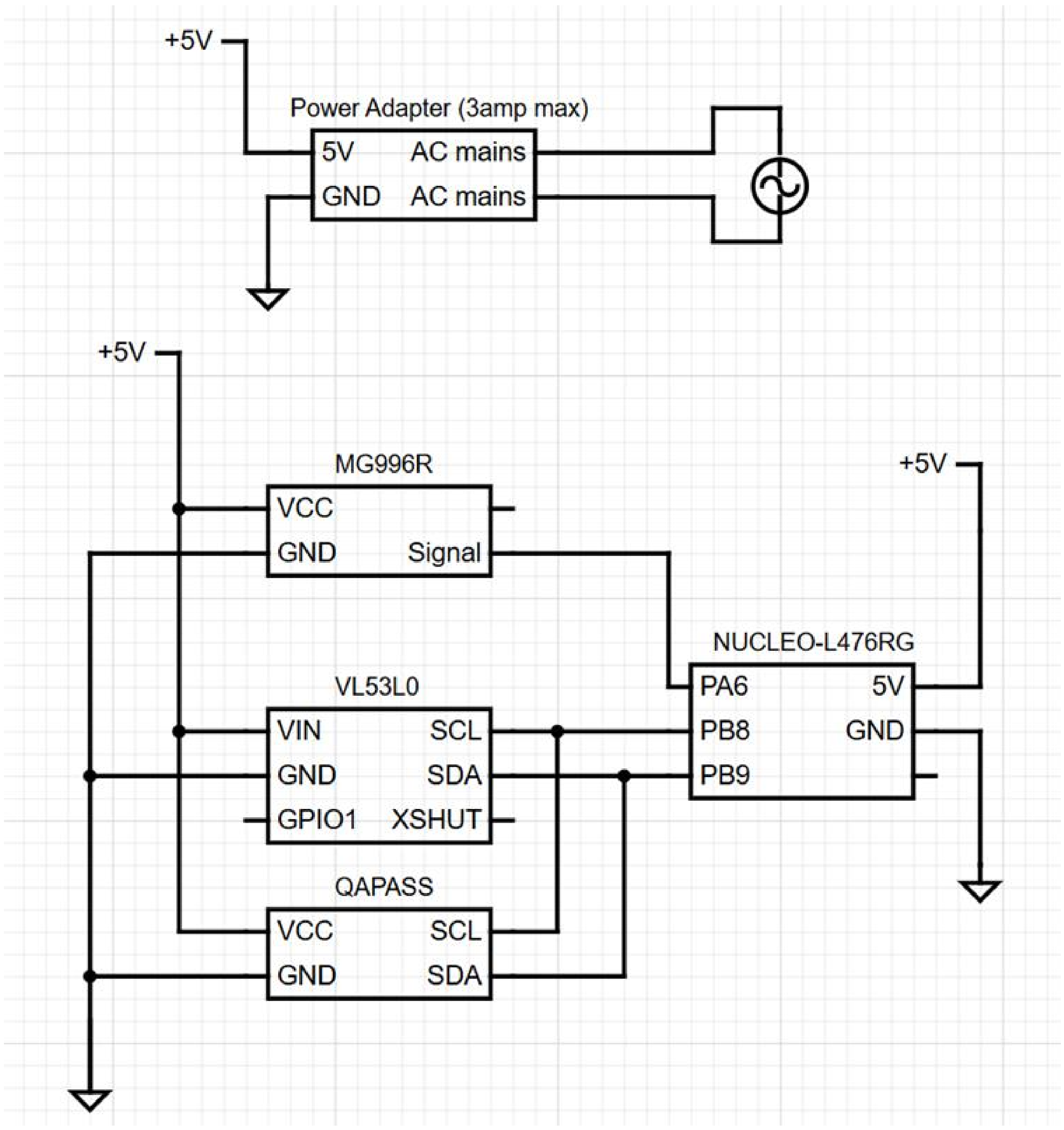

Figure 3: Schematic diagram of the project.

4 Test Plan and Test Results

The majority of the testing that was required on this system consisted of isolating the different components. After that, it was testing the system limits as all the components came together. Here is a compilation of the most important test cases. For further reference this is a link to a video of proper project operation `https://youtu.be/Pk7nl5lWs_w`

4.1 Test Plan Procedure

- Servo Max and Min don't stress the system
  - Send the command to move the servo to its coded "neutral height":
  - Send the command to move the servo to its coded "safe max height":
  - Send the command to move the servo to its coded "neutral height":
  - Send the command to move the servo to its coded "safe min height":
- Distance can be read accurately from the distance sensor
  - Print the sensor readings over UART and view them through putty:
  - Take a ruler and adjust the positioning of an object to ensure accuracy
- Display will show the distance and PID values
  - Adjust an object and ensure the distance changes values as expected on the display.
  - Program the following PID values (P:1.23, I:1.23, D:1.23)
  - Program the following PID values (P:1.239, I:1.239, D:1.239)
- Display, Distance sensor, and servo can operate at the same time
  - Adjust the code so that the servo max and min test occurs.
  - Add the code for the display and distance sensor. Program the following PID values (P:1.23, I:1.23, D:1.23). Have that code running at the same time as the servo test.
- The PID controller will stabilize a ball
  - With all other PID parameters set to zero and the ball on the leftmost side of the beam, adjust the P parameter until you see continuous oscillations (but the ball is not hitting the edges). Ensure you return the ball to the left most side each time you reprogram the parameters.
  - Now adjust the K value until you see sufficient damping in the system (the ball may not make it to the set point yet. The important thing is that its initial movement is dampened quickly and it comes to a relative stop). Ensure you return the ball to the left most side each time you reprogram the parameters.

  - Adjust the I parameter until the ball is able to reach the set point at the speed you wish. Ensure you return the ball to the left most side each time you reprogram the parameters.

4.2 Expected and Observed Results

- Servo Max and Min don't stress the system
  - Expected Result: Servo moves to neutral position
  - Actual Result: Servo moved to neutral position
  - Expected Result: Servo moves within functioning bounds to highest position
  - Actual Result: Servo moved within functioning bounds to highest position
  - Expected Result: Servo moves to neutral position
  - Actual Result: Servo moved to neutral position
  - Expected Result: Servo moves within functioning bounds to lowest position
  - Actual Result: Servo moved within functioning bounds to lowest position
- Distance can be read accurately from the distance sensor
  - Expected Result: Distance readings show up in the Putty terminal
  - Actual Result: Distance readings showed up in the Putty terminal
  - Expected Result: Distance readings show up in the Putty terminal and are relatively accurate as shown by the ruler within 5mm
  - Actual Result: Distance readings showed up and were accurate within 5mm
- Display will show the distance and PID values
  - Expected Result: Distance will change appropriately according to the objects placement. Distance will show in millimeters and have mm after the number.
  - Actual Result: Distance changed appropriately in the display. Was shown in millimeters.
  - Expected Result: Sensor will display Kp:1.23 Ki:1.23 Kd:1.23 D:xxxmm (some distance)
  - Actual Result: Sensor displayed Kp:1.23 Ki:1.23 Kd:1.23 D:xxxmm (some distance)
  - Expected Result: Sensor will display Kp:1.23 Ki:1.23 Kd:1.23 D:xxxmm (some distance) (the numbers should cut off the extra digit that is too long)
  - Actual Result: Sensor displayed Kp:1.23 Ki:1.23 Kd:1.23 D:xxxmm (some distance)
- Display, Distance sensor, and servo can operate at the same time

  - Expected Result: test occurs as expected
  - Actual Result: test occured as expected
  - Expected Result: Servo test occurs as expected alongside the display showing Kp:1.23 Ki:1.23 Kd:1.23 D:xxxmm (some distance)
  - Actual Result: Servo test occured as expected alongside the display showing Kp:1.23 Ki:1.23 Kd:1.23 D:xxxmm (some distance)

- The PID controller will stabilize a ball
  - Expected Result: Ball oscillates back and forth. The Display shows the current programmed values and the read distance each second.
  - Actual Result: Ball oscillated back and forth. The Display showed my current programmed values and the read distance each second.
  - Expected Result: Ball oscillations come to a stop. The Display shows the current programmed values and the read distance each second.
  - Actual Result: Ball oscillations came to a stop. The Display showed the current programmed values and the read distance each second.
  - Expected Result: Ball oscillations come to a stop and correct set point is realized. The Display shows the current programmed values and the read distance each second.
  - Actual Result: Ball oscillations came to a stop and the correct set point was realized. The Display showed the current programmed values and the read distance each second.

5 Code

The code in section 5.1 shows the PID function that takes the setpoint and current measured distance and computes an appropriate command to the servo. The code in section5.2 shows the while loop where I update the servo and LCD screen based on the current time from the HAL Get Tick function. For the complete code, please see my GitHub repository here `https://github.com/fowler557/PID-ball-and-Beam.git`

5.1 PID function

```c
/* USER CODE BEGIN 4 */
// Function to compute PID output with derivative filtering
float computePID(uint16_t setpoint, uint16_t measured_value) {
    uint32_t current_time = HAL_GetTick();  // Get the current time in ms
    float delta_time = (current_time - previous_time) / 1000.0;  // Calculate time difference in seconds
    previous_time = current_time;

    // Calculate the error
    float error = setpoint - measured_value;

    // Check if the error is within the tolerance range
    if (error >= -TOLERANCE && error <= TOLERANCE) {
        // If within tolerance, no need to adjust, set the output to 0 (or any value that makes sense)
        return 0.0;
    }

    // Proportional term
    float Pout = Kp * error;

    // Integral term
    integral += error * delta_time;
    float Iout = Ki * integral;

    // Derivative term with filtering
    float derivative_raw = (error - prev_error) / delta_time;  // Raw derivative
    // Low-pass filter to smooth the derivative term
    derivative_filtered = derivative_filtered + DERIVATIVE_FILTER_CONSTANT * (derivative_raw - derivative_filtered);
    float Dout = Kd * derivative_filtered;

    // Combine terms
    float output = Pout + Iout + Dout;

    // Store the current error for the next derivative calculation
    prev_error = error;

    return output;
}
/* USER CODE END 4 */
```

5.2 Non Blocking Logic to handle continuous updates

```c
 /* USER CODE BEGIN WHILE */
  uint32_t lastTime = HAL_GetTick();
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

      // Get the current time in milliseconds
      uint32_t currentTime = HAL_GetTick();

      // Only update sensor and PID every 20ms
      if (currentTime - lastTime >= updateInterval) {
          lastTime = currentTime;

          // Read the distance from VL53L0X
          distance = readRangeSingleMillimeters(&distanceStr);

          // Calculate the PID output
          float pid_output = computePID(setpoint, distance);

          // Reverse sign of PID output for orientation of servo
          pid_output = -pid_output;

          // Convert PID output to a PWM value (ensure it's within the valid
 range for the servo/motor)
          uint16_t pwm_value = (uint16_t)(75 + pid_output); //75 is PWM
servo midpoint
          if (pwm_value > 100) {
              pwm_value = 100;  // Maximum PWM servo limit
          } else if (pwm_value < 50) {
              pwm_value = 50;   // Minimum PWM servo limit
          }

          // Update the PWM output (control the servo)
          __HAL_TIM_SET_COMPARE(&htim3, TIM_CHANNEL_1, pwm_value);
      }

      // LCD update (every 1000 ms or whatever interval makes sense)
      if (currentTime - lastPrintUpdate >= printUpdateInterval) {
          lastPrintUpdate = currentTime;

          // Prepare LCD buffer
          char lcdBuffer[32];  // Buffer to hold 32 characters for each line

          // Clear the LCD first
          lcd_clear();

          // Display the Kp and Ki values on the first line
          snprintf(lcdBuffer, sizeof(lcdBuffer), "Kp:%d.%02d Ki:%d.%02d",
                   (int)(Kp), (int)(Kp * 100) % 100,
                   (int)(Ki), (int)(Ki * 100) % 100);
```

```
            lcd_set_cursor(0, 0);  // Set cursor to the beginning of the first
    line
            lcd_write_string(lcdBuffer);


            // Display the Kd and Setpoint (SP) on the second line
            snprintf(lcdBuffer, sizeof(lcdBuffer), "Kd:%d.%02d D:%dmm",
                     (int)(Kd), (int)(Kd * 100) % 100,
                     (distance));
            lcd_set_cursor(1, 0);  // Set cursor to the beginning of the
   second line
            lcd_write_string(lcdBuffer);


        }
    }
    /* USER CODE END 3 */
  }

}
```

6 Conclusion

It was so satisfying to see this project come together like it did. I was able to use PWM, I2C and digital logic to control the position of a ball. There were many areas I was able to test and integrate one at a time. It was fun to experience the prototyping process with wood, hot glue, and Popsicle sticks. I am inspired to see what other more complicated control systems can be implemented with the STM32 micro-controller. I think it would be cool to implement a simple Kalman Filter in a future iteration. That would teach me a lot about state estimation and with only 1 degree of freedom it shouldn't be too tricky.

Even though this project was fun,it was certainly not without it's setbacks. My first iteration was using two rulers hot glued together. It was very difficult to get proper control because of inconsistencies in the material widths. I also originally had a lower timing budget set for the IR sensor (200). This allowed the sensor to read faster but sacrificed accuracy. For some reason the servo had a hard time responding to a lower value. When I raised it from 200 to around 350 things started to work much better. I was also able to learn about low pass filters while constructing this project. At first, the derivative term was extremely noisy due to sensor imperfections. However, after implementing a low-pass filter for the derivative term, its performance improved. It was very fun to take elements from my microprocessors class and implement them into a project of my own making. It was also really cool to use online resources to learn about various parts and libraries that would be useful in the final integration. All in all, I feel more prepared to tackle harder problems because of this project.

References