PD control of a ball rolling along a single axis

Thomas Leech and Andrew Hall

June 23, 2018

Abstract

PD controllers are one of the most popular controllers for simple systems and are used in a wide variety of applications. Drawing inspiration from systems that balance and move a ball along a planar surface, we applied PD control to a simplified, one degree of freedom system. We built a model our system's dynamics and our feedback controller. We then built a physical system and used our model to generate appropriate gains.

1 Introduction

In class, we studied PD controllers in the context of the copter levitated arm. In the case of the copter levitated arm, we controlled the voltage into the motor which effectively allowed us to control the thrust generated by the propeller. Since thrust is proportional to acceleration, we quickly found that a proportional controller would not be sufficient to control the arm.

The problem we are looking at is similar in that we can control the angle of the track and therefore the acceleration of the ball. Our system is a ball on a long rail that can freely roll along the rail. A servo motor which allows us to directly control position is attached to the center of the rail. To read the state of the system, we have and infrared range-finder looking down the length of the rail to detect the position of the golf ball.

2 Our Model

2.1 Dynamics

To compute the dynamics of our system we define the following values: Y the position of the ball along the track, Θ the angle of the beam measured from the horizontal, m the mass of the ball, g the acceleration due to gravity, R the radius of the ball, and J the moment of inertia of the ball. Doing a force balance gives the following:

$$0 = (\frac{J}{R^2} + m)\ddot{Y} + mg\sin(\Theta) - mY\dot{\Theta} \tag{1}$$

We can then linearize our system about the point $\Theta = 0$ giving:

$$(\frac{J}{R^2} + m)\ddot{Y} = -mg\Theta \tag{2}$$

By taking the Laplace transform of this equation, we can find the continuous time transform function for our plant. We also let $\gamma = \frac{-mg}{(\frac{J}{R^2}+m)}$ to simplify the expression The transfer function is:

$$\frac{Y(s)}{\Theta(s)} = \frac{-mg}{(\frac{J}{R^2} + m)s^2} = \frac{\gamma}{s^2} \tag{3}$$

Our hardware, however, works in discrete time. To find the discrete time formulation, we use MATLAB's c2d function which gives:

$$\frac{4.5 * 10^-4 * \gamma * (z + 1)}{(z^2 - 2z + 1)} \tag{4}$$

2.2 Controls Model

We initially tried to stabilize our system using PID control. By analyzing the step response of our system, we found we could stabilize it using only proportional and derivative control, and did not need the use of integral control. After settling on a PD controller, we estimated gamma and picked sufficient gains, as discussed in section three. The PD controller had a transfer function of

$$K_p + \frac{K_d(z - 1)}{z\Delta T} \tag{5}$$

Where $K_p = 0.08$, $K_d = 0.06$, and $\Delta T = 0.03$ Implementing our chosen gains yielded a closed loop transfer function of

$$\frac{0.006552z^2 + 0.000252z - 0.0063}{z^3 - 1.993z^2 + z - 0.0063} \tag{6}$$

3 Implementation

For our implementation, we decided use an Arduino Uno micro controller. We chose the Uno because we were more familiar with its pin-out and because we wanted to keep our other lab projects intact. Our other materials included a VS11 servo motor, a SHARP GP2Y0E02A IR sensor, a golf ball, and various LEGO bricks.

3.1 Estimating Gamma

Our ability to model our system accurately relies on our ability to measure an accurate value for γ. While our dynamics model includes an analytical form of γ which was never shown in class, there could easily be other physical parameters that we didn't account for. It is also difficult to precisely measure some the quantities needed to calculate γ. Therefore, we measured γ indirectly by looking at our system in oscillation. By measuring the period of oscillation, we can determine a value for

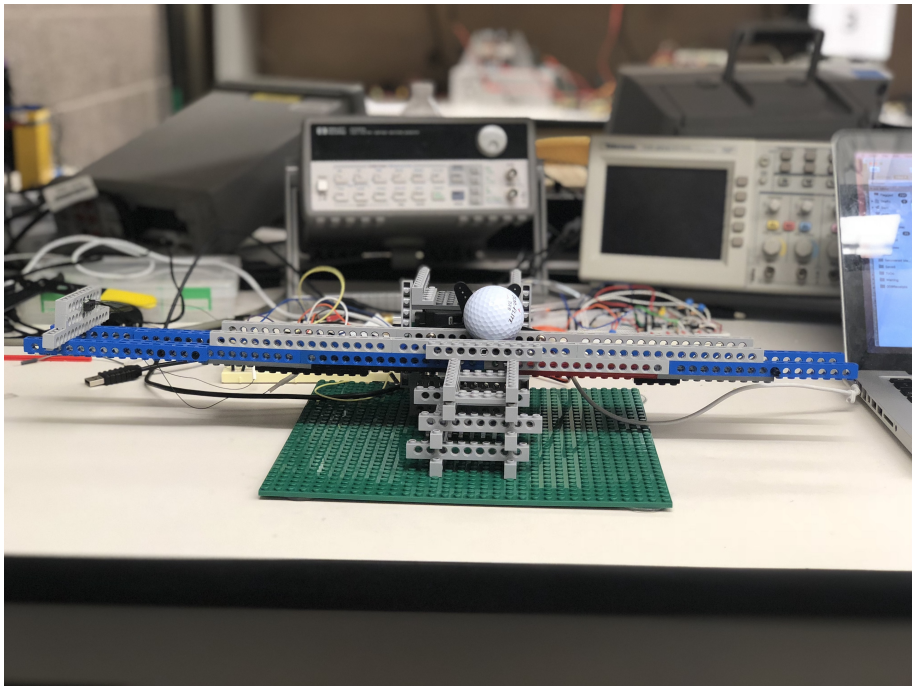

Figure 1: Ball and beam system.

γ more accurately than we could calculate analytically. To find a value for γ, we use the fact that the largest imaginary part of the poles is $\frac{2\pi}{period}$. We can then guess values for γ until we find one that generates the same largest imaginary component. For this system, we found γ to be 7.3.

3.2 Choosing Gains

To ensure the stability of our discrete time system, we of course wanted our poles to fall within the unit circle. We started out choosing our derivative gain based on how much noise we could tolerate. We found noise in our derivative to be the largest problem we faced in designing our system and it was the limiting factor in our systems performance. We chose our derivative gain to be as large as possible while not amplifying the noise so much that it interrupts convergence. We then chose an appropriate proportional gain using our model that was as large as possible without causing too much overshoot. We settled on a $K_d = 0.06$ and a $K_p = 0.08$. The small magnitude of these gains is due to the large values read out by the sensor (in the range from 0 to 400 in about 30cm).

3.3 Working With Noise

As previously mentioned, the most difficult part of designing this system was dealing with noisy sensor readings. At times it was difficult to even have a consistent distance measurement, let alone trying to take the derivative. Figure 1 shows and example of the unfiltered signal we received from our sensor. Throughout the semester, we tried four different sensors; one sonic sensor and three infrared sensors. The final sensor we tried was by far the most reliable and also conveniently had the fastest sample rate. This was another problem, many of the sensors, even the final one, had sample

rates on the order of tens of milliseconds. They implemented a zero-order hold in between samples so that running at a faster sample rate would only be sampling the same point again and would therefore be useless. This limit on our sample rate made the noise problem even more difficult since most low pass filters (implemented in software or hardware) introduce some lag and with a slow sample rate, that lag becomes significant quickly.

To combat this noise, we tried three different approaches and eventually arrived at a combination of a couple of them. The first approach was to make a simple low pass filter with a resistor and a capacitor and filter the signal coming out of the sensor before it got to the arduino. We decided to forgo this method because it did not make a significant impact on the signal until the capacitor was large enough to make the system lag too much. Our second approach was to implement a windowed average in the arduino code. The code for this is very straight forward. We keep an array of previous samples and a running sum of that array. Each time a new sample is taken, we subtract off the oldest value in the array and replace it with the new sample. The filtered reading is then simply the total divided by the size of the window. The code is provided in Listing 1.

Listing 1: Windowed Average

```
window_total = window_total - window[window_index];
window[window_index] = sensor_reading;
window_total += window[window_index];
window_index = (window_index+1) % window_size;
filtered_reading = window_total/float(window_size);
```

The final technique we implemented for trying to reduce the noise is a simple disturbance rejection algorithm. As we read in new samples, we check to see if it is greater than some threshold away from the previous reading. Our premise in this technique is that the ball cannot physically jump more than say 10cm in one timestep (i.e. we set a limit on the balls velocity). We also applied this concept to the derivative term. Thereby setting a limit on the balls velocity as well. This was the most effective technique at eliminating noise since it could effectively handle spikes of any large magnitude and it did not introduce any lag. The windowed average was then able to more effectively handle noise which fell below this threshold without introducing too much lag since it no longer had to worry about averaging out the large spikes.

4 Results

The final control model yielded a slow and oscillatory step response, yet with the noise of the IR, this was the best performance possible.

Analyzing the root locus of the system showed its poles initially fell outside of the unit circle and was thus unstable. Our chosen gains shifted the poles within the unit circle and stabilized the system.

Shown below are also our distance readings and derivatives before and after filtering.

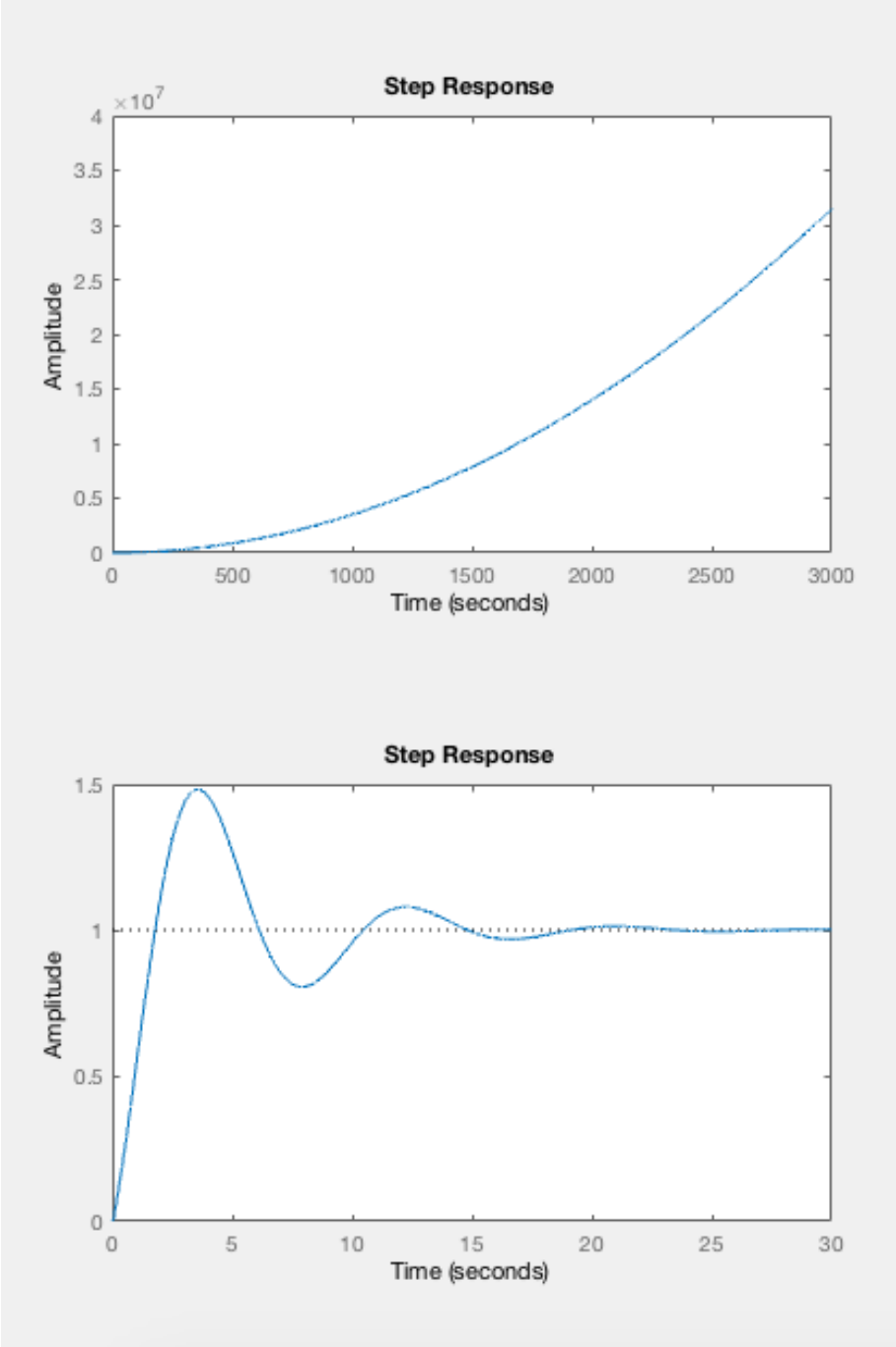

Figure 2: Step response without feedback(above) and with feedback(below).

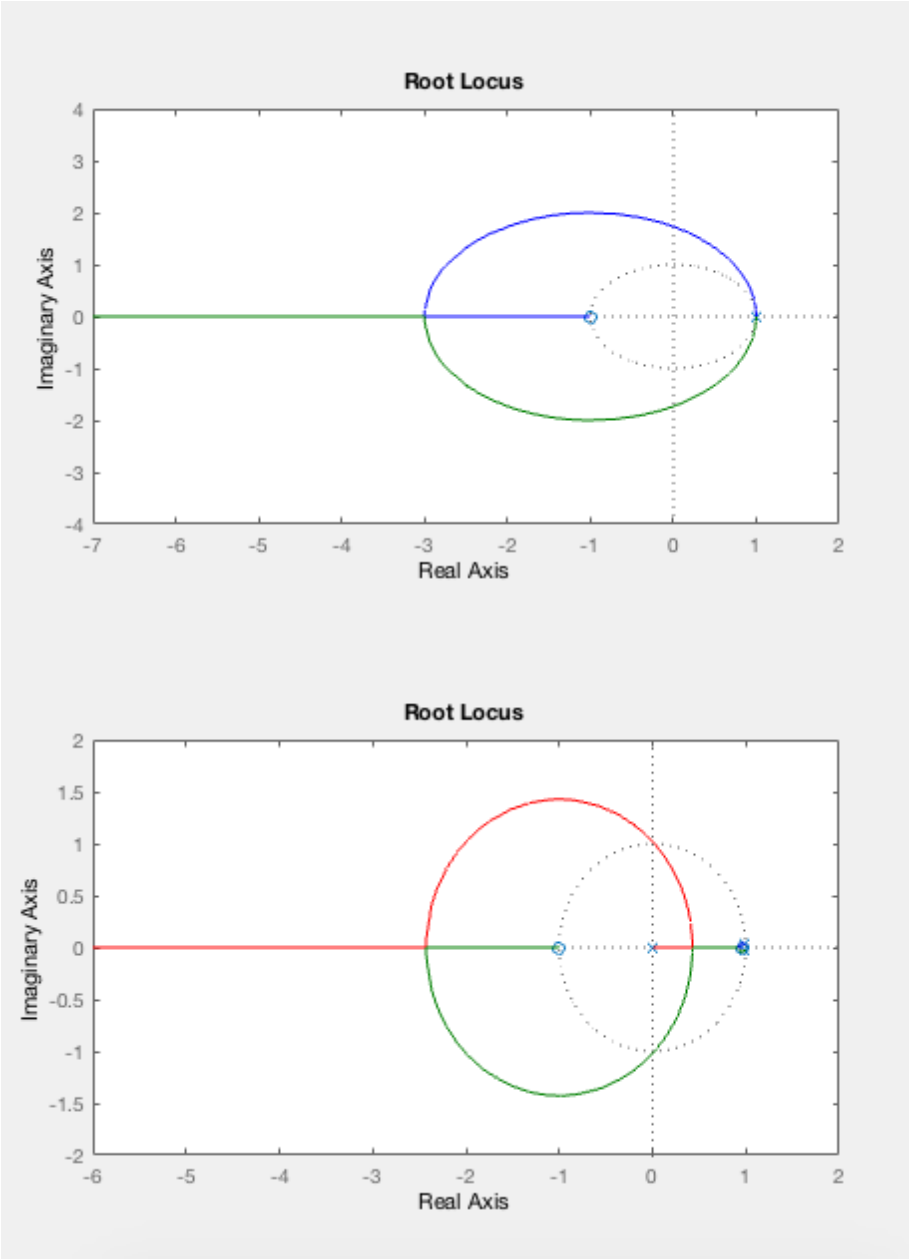

Figure 3: Root locus without feedback(above) and with feedback(below).

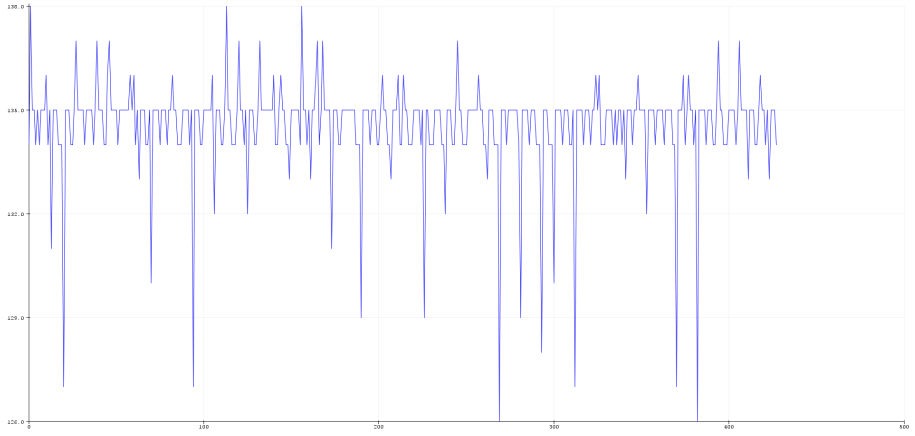

Figure 4: Unfiltered readings from the distance sensor.

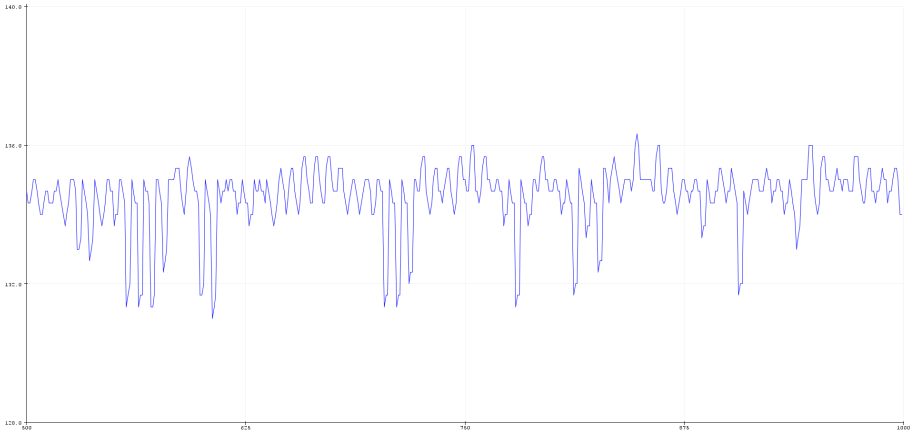

Figure 5: Filtered readings from the distance sensor.

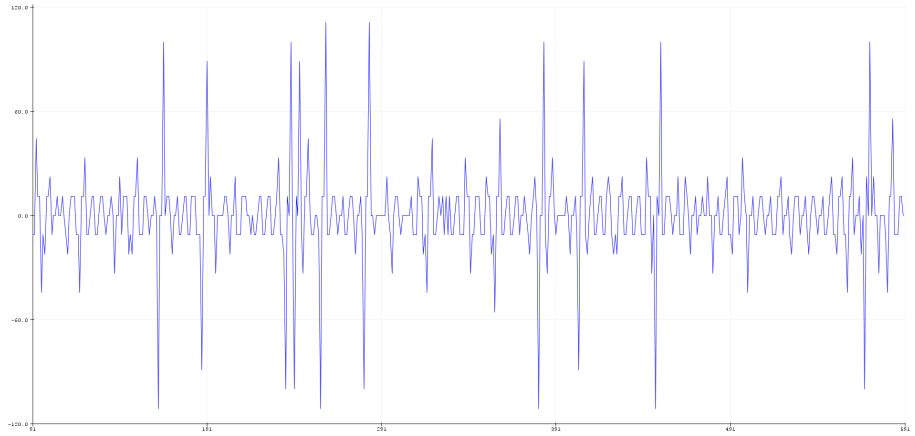

Figure 6: Unfiltered derivative.

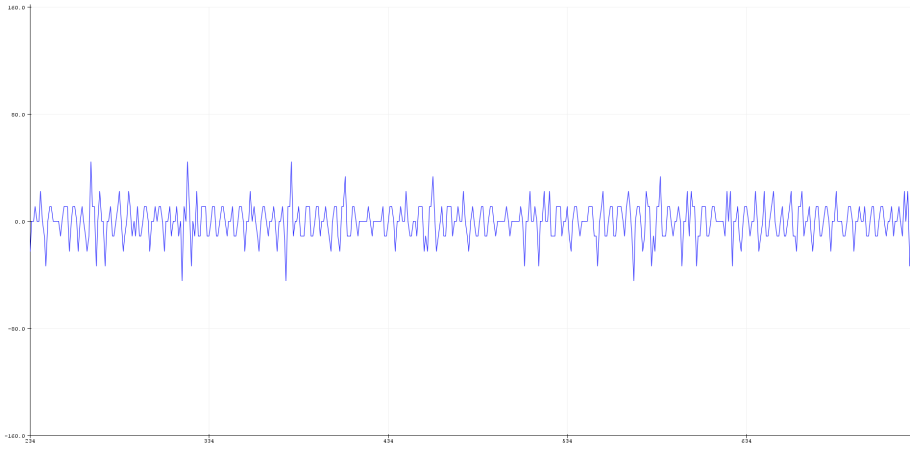

Figure 7: Filtered derivative.

5 Discussion

The system could have potentially been stabilized better by implementing state space control with an observer to compensate for the noise of the system. Adding another IR sensor and removing the correlated noise between both sensors may have also increased signal robustness.

6 Conclusion

Overall, a simple PD controller proved effective in stabilizing the one degree of freedom ball on beam system, despite noisy sensors.