EE572 Course Project

Matt Ruffner
Alex Mueller
Ben Cummins
Galvin Greene

April 27, 2018

1 Objective

A physical control system allows the team to apply class concepts to the real world. These include continuous/discrete time modeling and compensation techniques. With a working system, the team has demonstrated competency in controls theory. For this project the team used a Teensy LC [1] ARM microcontroller to implement a PID compensator. The system manages the speed of a DC motor with a propeller like attachment. Simulations were run in MATLAB to help design the PID compensator.

2 Continuous Time System

In order to fully control our chosen system, the team first had to analyze and decide on a G(s) model that would represent the system. The team had brief discussion and the chosen model can be seen from equation 1.

$$G(s) = \frac{K}{1 + 0.5\tau s} \tag{1}$$

Within this model, K represents the max speed, in RPM, that our motor can physically move, represents the time it took to move from the lowest to the highest. These variables were found by testing and measuring the system. Now that the G(s) has been determined, the team had to create a Zero Order Hold model for our system, which can be seen from equation 2

$$G_{ZOH}(s) = \frac{1}{1 + \frac{sT_s}{2}} \tag{2}$$

This representation for the Zero Order Hold is used to properly model the system in continuous time and simulate the discrete measurements. This model relies on a Ts and was chosen to be .1 seconds. The final model needed for the system is a compensation model to allow the team to alter the system to specific specs. For this the team chose a PID model and can be seen from equation 3

$$G_{PID}(s) = \frac{K_i + sK_p + s^2K_d}{s} \tag{3}$$

This model for the PID compensation requires the team to solve for K_i, K_p, and K_d and is completed by determining an s_1 that meets given transient specs and finding a Z_c and a K_c by computing coefficients. For this model we used equation 4, because the team had no steady state error specs and thus could improve transient specs.

[1] https://www.pjrc.com/teensy/teensyLC.html

$$G_{PID}(s) = s^2 K_c + 2sZ_c K_c + K_c Z_c^2 \tag{4}$$

Where

$$K_d = K_c$$
$$K_p = 2K_c Z_c$$
$$K_i = K_c Z_c^2$$

Once these values were calculated and implemented into the equation, the team could then close the loop for the continuous time model by using equation 5. This is an important part of simulation as it shows how the system should react in real time and whether the transient specs were truly met and if needed, further optimized.

$$G_{closedloop}(s) = \frac{G_{PID}(s)G_{ZOH}(s)G(s)}{1 + G_{PID}(s)G_{ZOH}(s)G(s)} \tag{5}$$

Additionally, if observed from the top level, the system would appear as Figure 1.

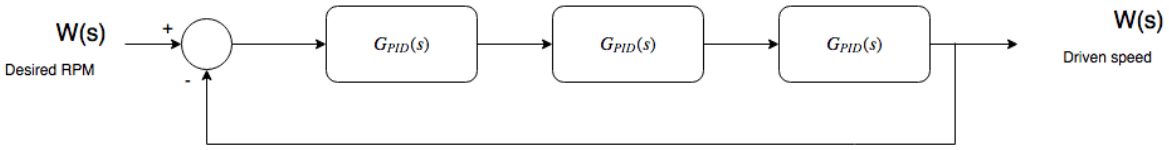

Figure 1: Continuous time system model

For a complete list of our given specs and those values that were solved for/calculated in MATLAB, see Table 1.

Parameter	Value
T_s	0.1 seconds
τ	8 seconds
M_p	4.32%
t_s	1 second
s_1	-4+j3.996
K	810
$\omega_n\zeta$	4
ω_n	5.6566
ζ	0.707
$\angle$ def.	103.314°
Z_c	7.1739
K_c	0.0334
K_i	1.7173
K_p	0.4787
K_d	0.0334

Table 1: System parameters

3 Discrete Time Compensator

The discrete compensator was found by using continuous version and substituting an estimate for s in terms of z. Our estimate was

$$s = \frac{z-1}{zT_s}$$

which is a rectangular approximation from the right side. The bilinear transform was not used because it would have placed a pole at $z = -1$ when substituting in s with the K_d term. Using MATLAB we found

$$H_{PID}(z) = \frac{3.546e17z^2 - 4.129e17z + 1.202e17}{3.603e17z^2 - 3.603e17z}$$

Simplifying the coefficients gives values of $a_0 = 1$, $a_1 = -1$, $a_2 = 0$, $b_0 = 0.9842$, $b_1 = -1.1461$, $b_2 = 0.3337$ for our second order filter to be implemented in microcontroller code.

4 MATLAB Simulation

This section contains plots of simulation data from designing the system in MATLAB.

Figure 2 shows the system's step response before compensation. Figure 3 shows the step response of the compensated system.

Figure 4 shows the system's step response before compensation. Figure 5 shows the step response of the compensated system.

5 Implementation

A simple geared down DC brushed motor is driven by a the Teensy LC microcontroller over USB power. Flyback diodes for reverse EMF protection and noise filtering circuitry are also included.

The second order PID filter was implemented the same way that was taught in EE572 lecture. The general form of the filter is shown in Equation 6.

$$\frac{Y(z)}{W(z)} = \frac{b_0 + b_1 z^{-1} + b_2 z^{-2}}{a_0 + a_1 z^{-1} + a2_z^{-2}} \tag{6}$$

The coefficients were computed in MATLAB, but taking the inverse z transform and rearranging to solve for y_k, our output, gave a new equation that could be written in code:

```
yk=(b0*wk+b1*wk_minus1+b2*wk_minus2-a1*yk_minus1-a2*yk_minus2)/(a0)
```

Update equations were also needed to change the values after each iteration. These were as follows:

```
wk=Setpoint-rpm;
wk_minus2=wk_minus1;
wk_minus1=wk;
yk_minus2=yk_minus1;
yk_minus1=yk;
```

5.1 Feedback

Feedback on the rotation speed of the system is provided by an IR reflectance sensor. When the propeller passes over the sensor the output voltage dips. These dips are monitored with an ADC channel and detected with thresholding logic in software. The interrupt routine responsible for accurately determining the current RPM of the motor has a much high sampling rate than the control system that uses this information.

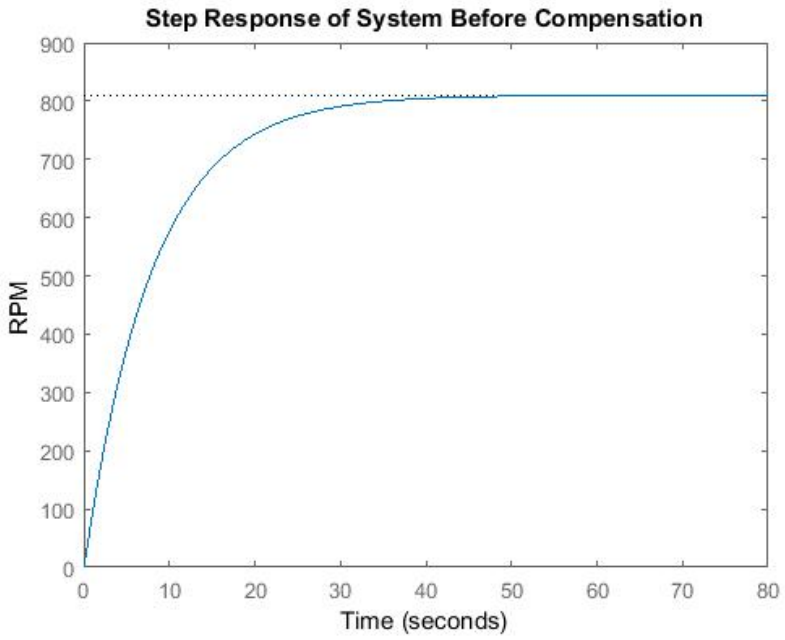

Figure 2: The step response of the system before compensation.

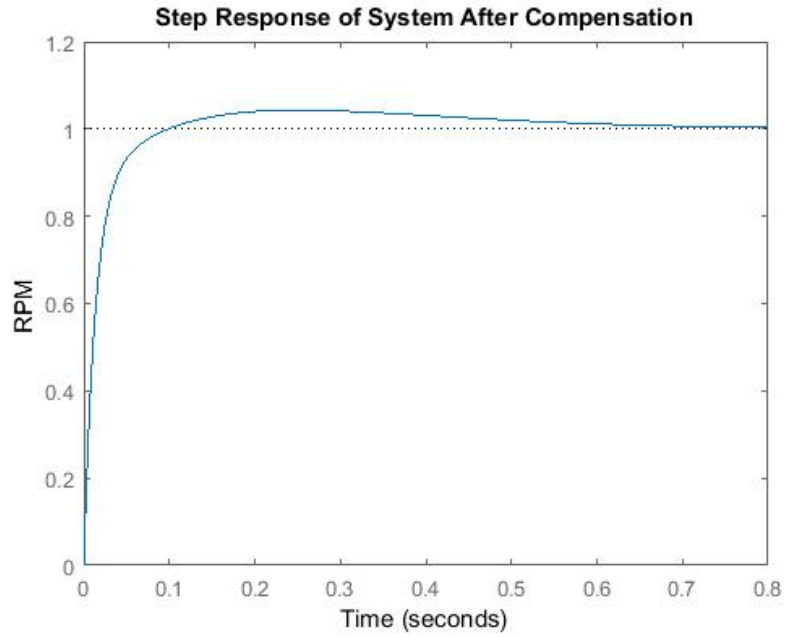

Figure 3: The step response of the system after compensation.

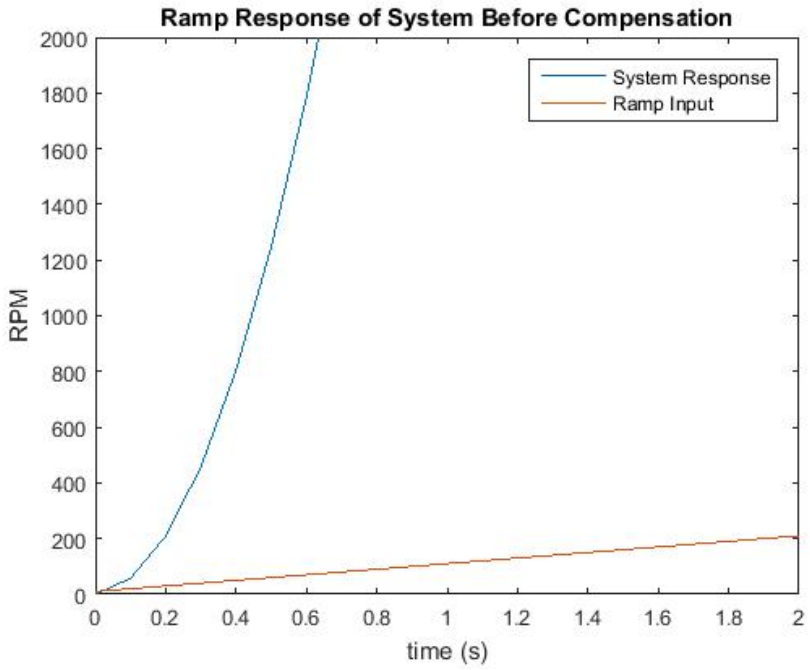

Figure 4: The ramp response of the system before compensation.

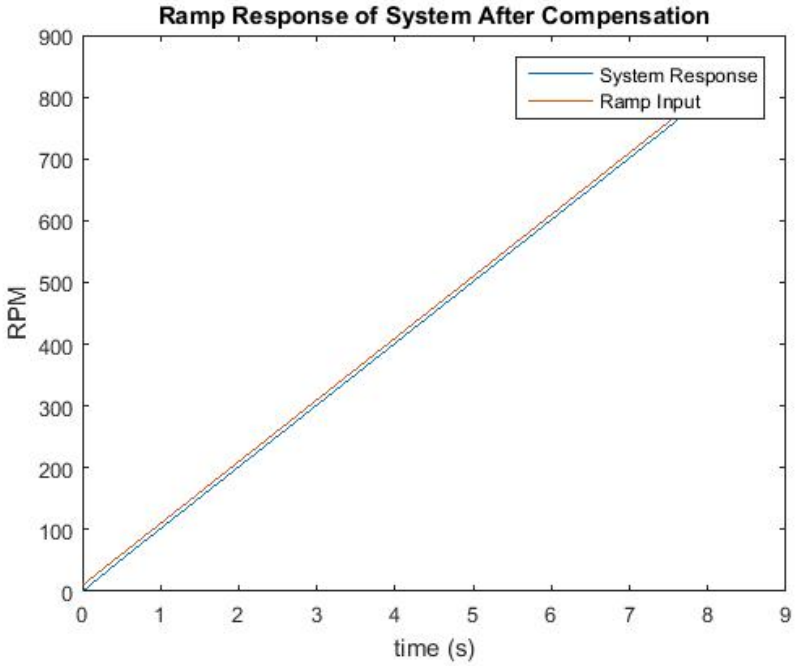

Figure 5: The ramp response of the system after compensation.

The protection circuity was modified from the schematic seen in Figure 6[2] and only includes one MOSFET.

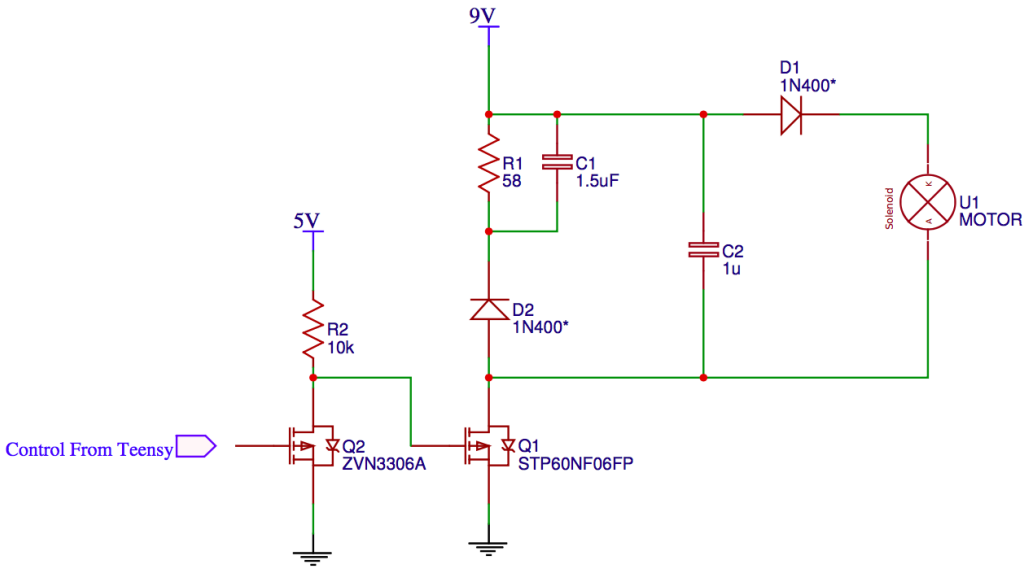

Figure 6: Back EMF protection circuitry.

5.2 Mechanical Design

One key aspect of the code was taking input from the analog reflectance sensor to compute the current RPM of the system. This was achieved by calling the millis() function in conjunction with interrupts from the TimerOne library. RPM values calculated using this method were used as part of the feedback of the system to compare to the setpoint.

5.3 Finished System

5.4 Plot of Compensation during Setpoint Change

The plot in Figure 8 shows a plot of the desired speed in comparison to the set speed with data logged from the device. This verifies the desired 1 second settling time that we improved from the original settling time of over ten seconds.

The plot in Figure 9 shows the W_k and Y_k values during the same run as in Figure 8.

5.5 Plot of Compensation During Loading Down of Motor

Figures 10 and 11 show the response of the device under loading conditions e.g. hand on motor shaft.

The plot in Figure 9 shows the W_k and Y_k values during the same run as in Figure 8.

[2] github.com/ruffner/feedback-strobe

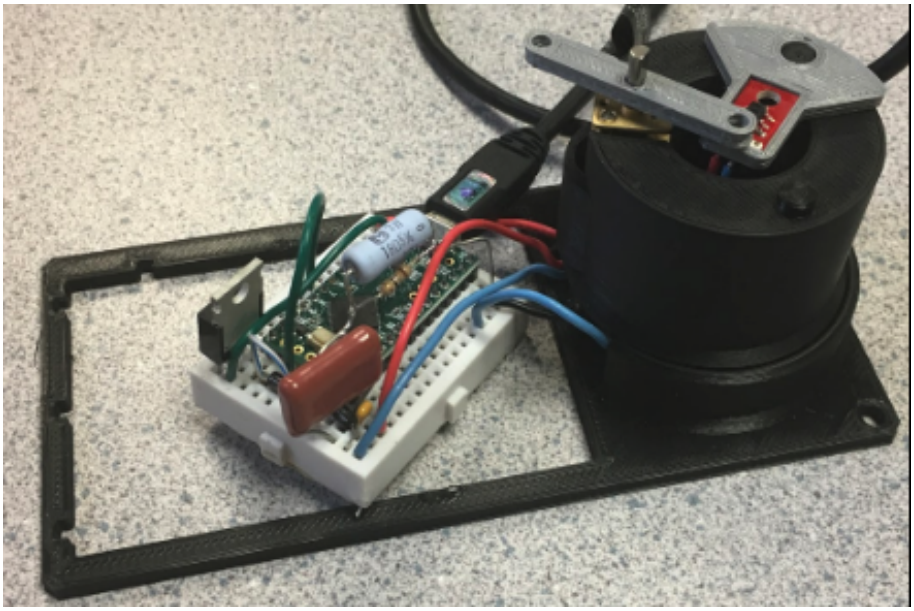

Figure 7: The finished motor + IR sensor setup.

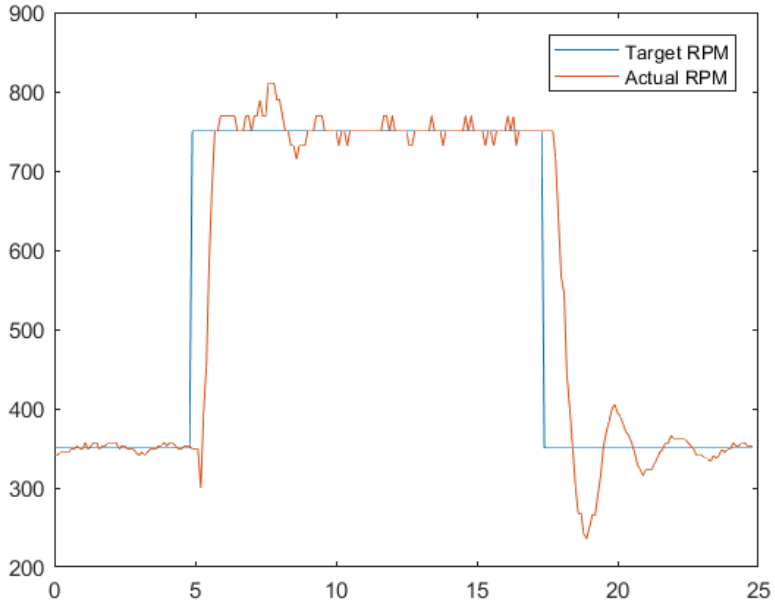

Figure 8: Target RPM vs actual RPM after a setpoint change.

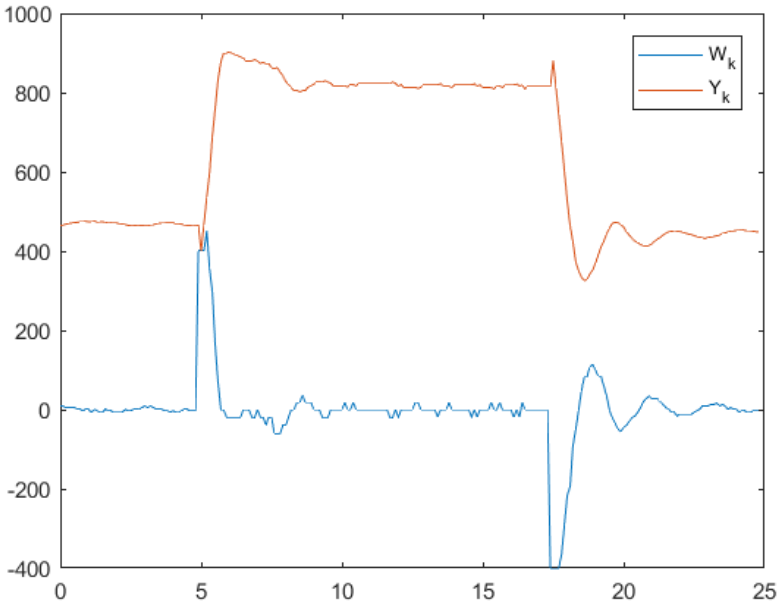

Figure 9: W_k and Y_k values of the device while the setpoint is changed between two values.

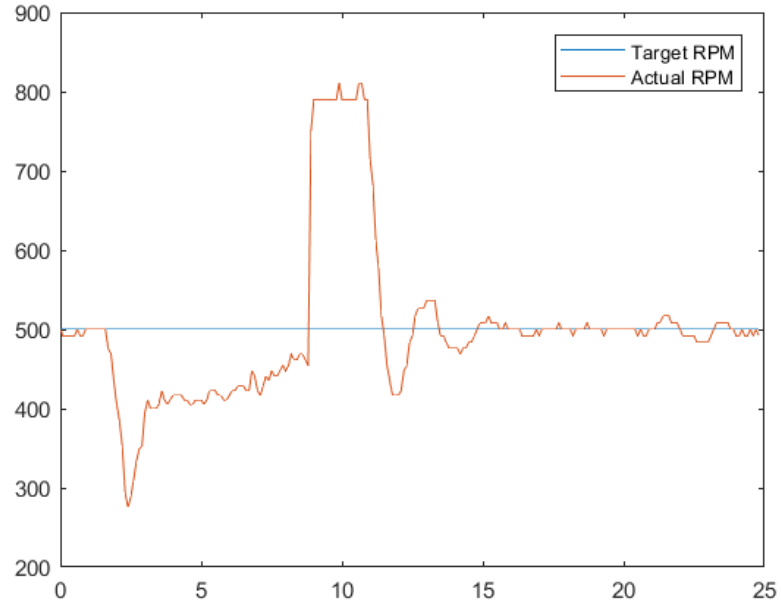

Figure 10: Target RPM vs actual RPM while putting pressure on motor shaft.

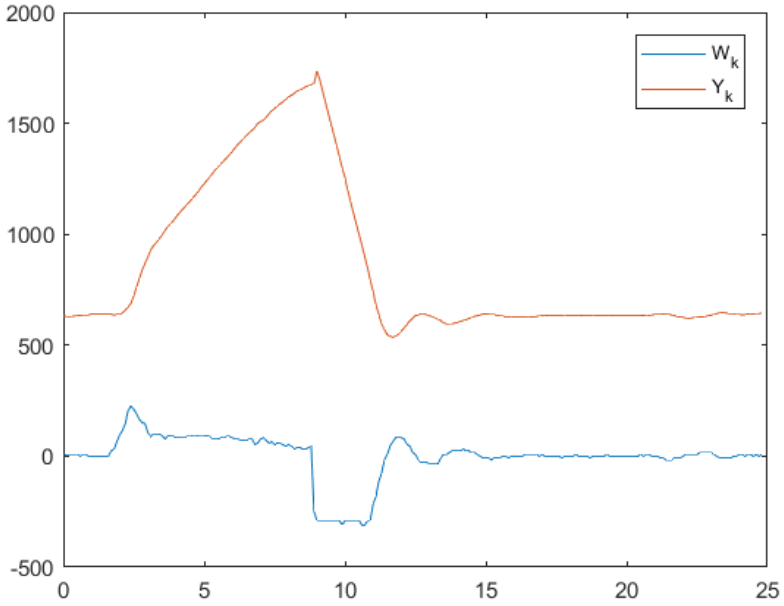

Figure 11: W_k and Y_k values of the device while pressure on motor shaft.

6 Contributions

6.1 Alex Mueller

Preliminary brainstorming and MATLAB Code.

6.2 Matt Ruffner

Mechanical design, Arduino coding.

6.3 Benjamin Cummins

MATLAB coding and brainstorming.

6.4 Galvin Greene

Debugged MATLAB and Arduino code.

7 References

Code can be found at: `https://github.com/ruffner/ee572/tree/master/project`