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.

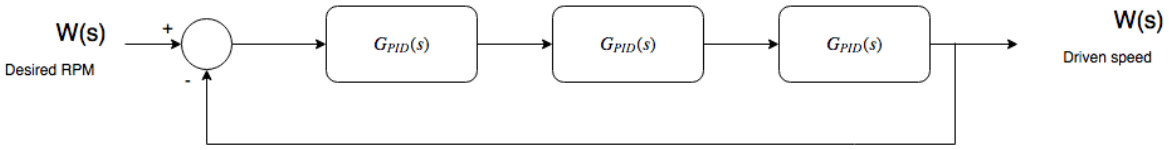

Figure 1: Continuous time system model

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.

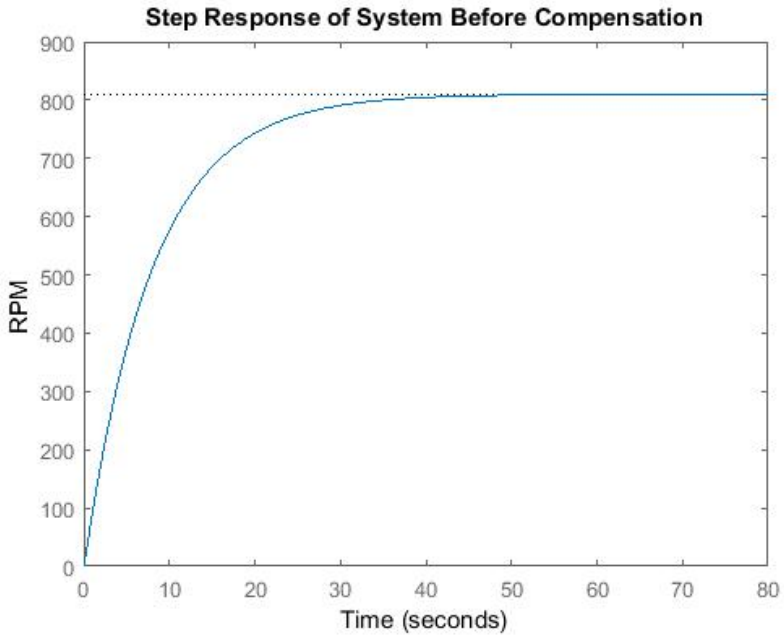

Figure 2: The step response of the system before compensation.

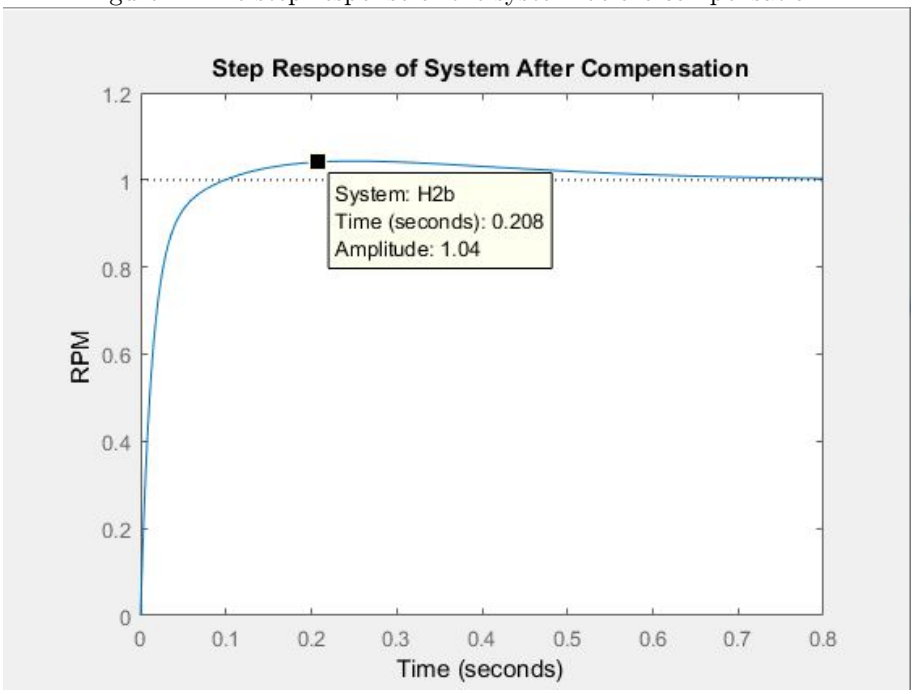

Figure 3: The step response of the system after compensation.

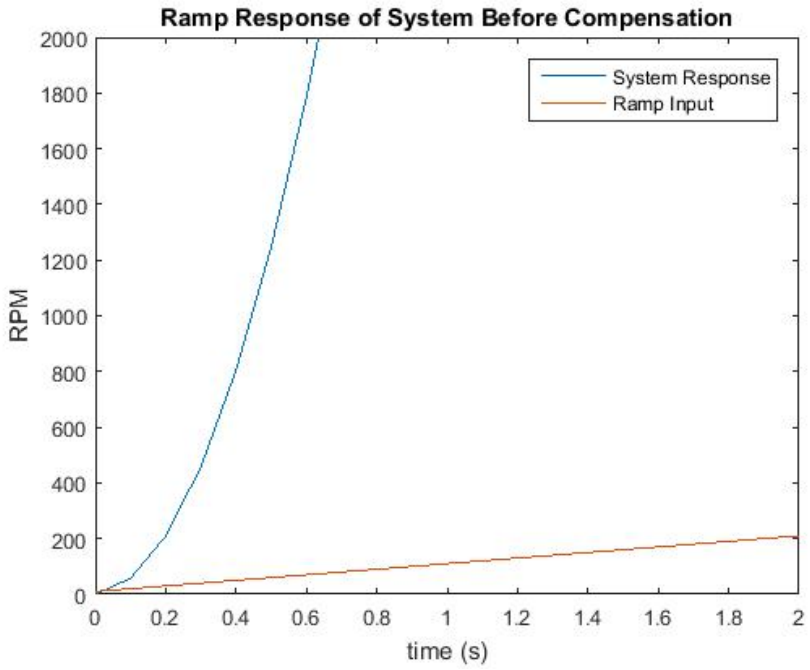

Figure 4: The ramp response of the system before compensation.

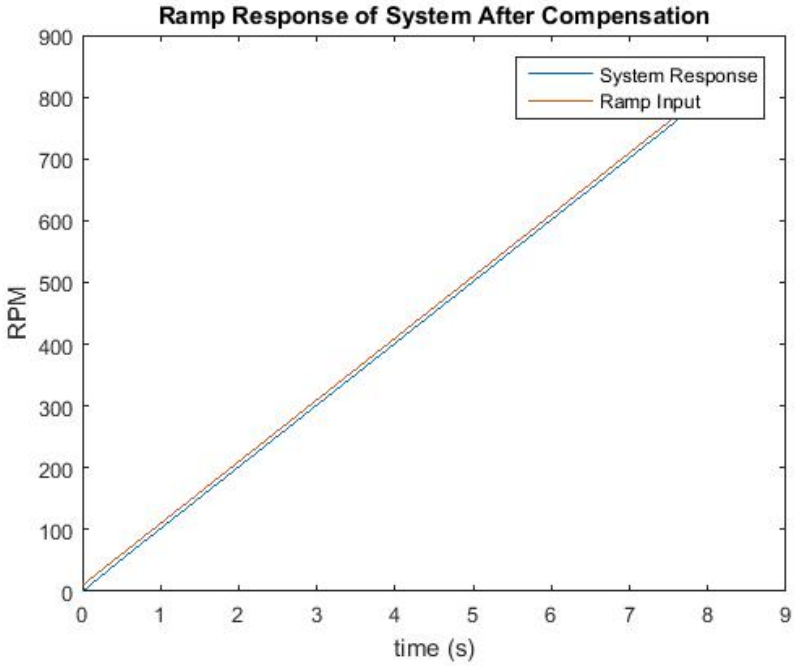

Figure 5: The ramp response of the system after compensation.

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

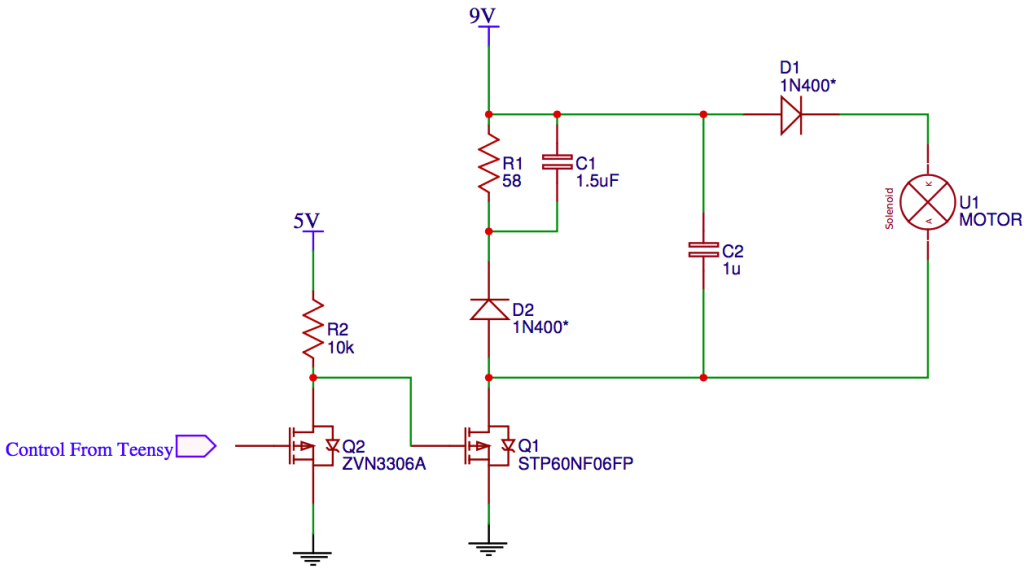

Figure 6: Back EMF protection circuitry.

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.

5.6 Ramp Following

Figure 12 shows the real system following a ramp.

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

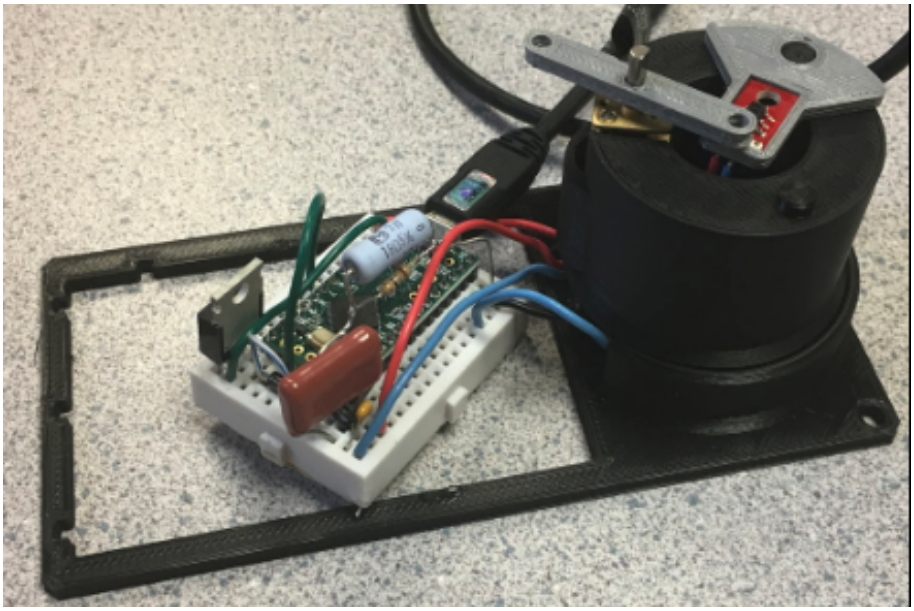

Figure 7: The finished motor + IR sensor setup.

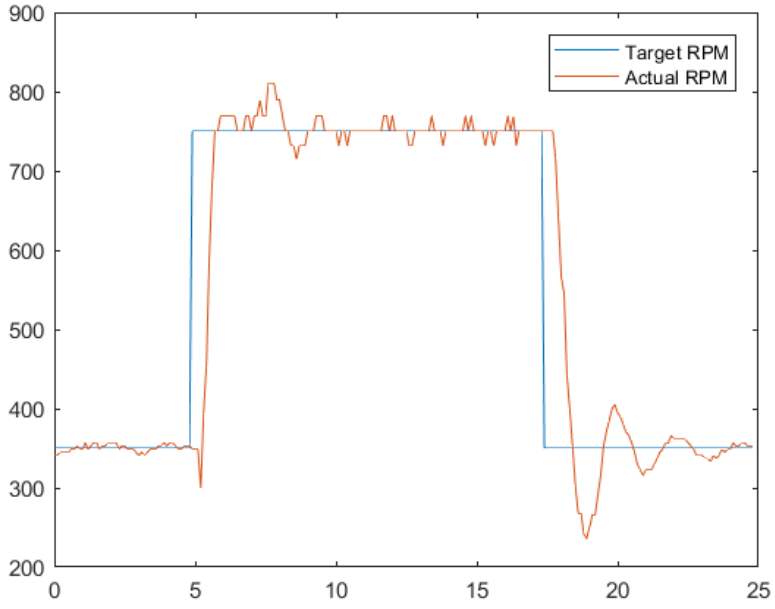

Figure 8: Target RPM vs actual RPM after a setpoint change.

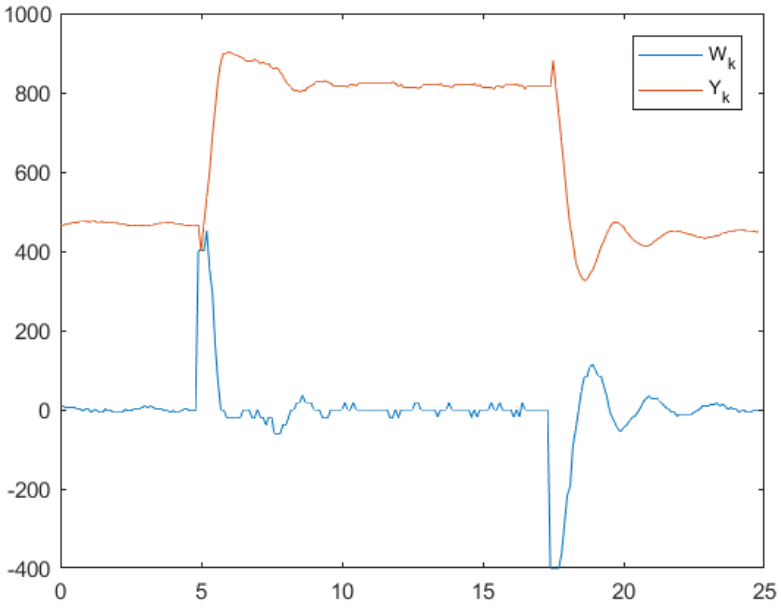

Figure 9: W_k and Y_k values of the device while the setpoint is changed between two values.

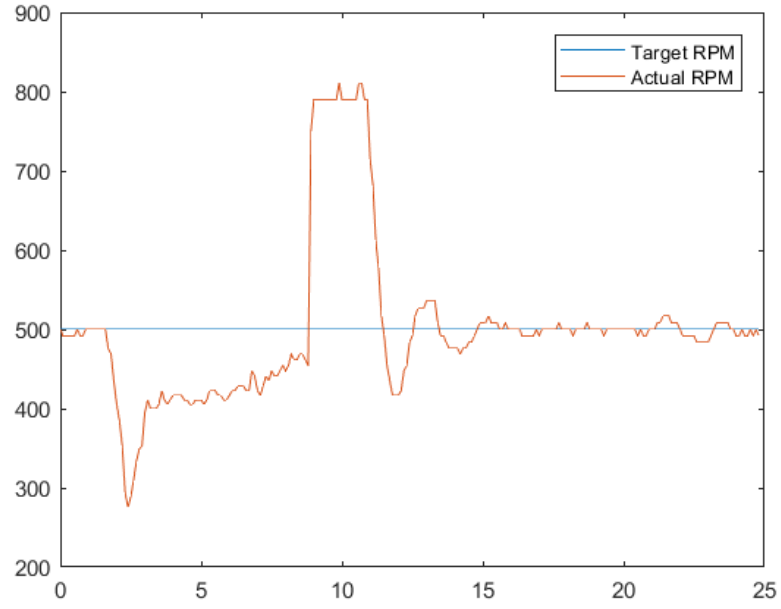

Figure 10: Target RPM vs actual RPM while putting pressure on motor shaft.

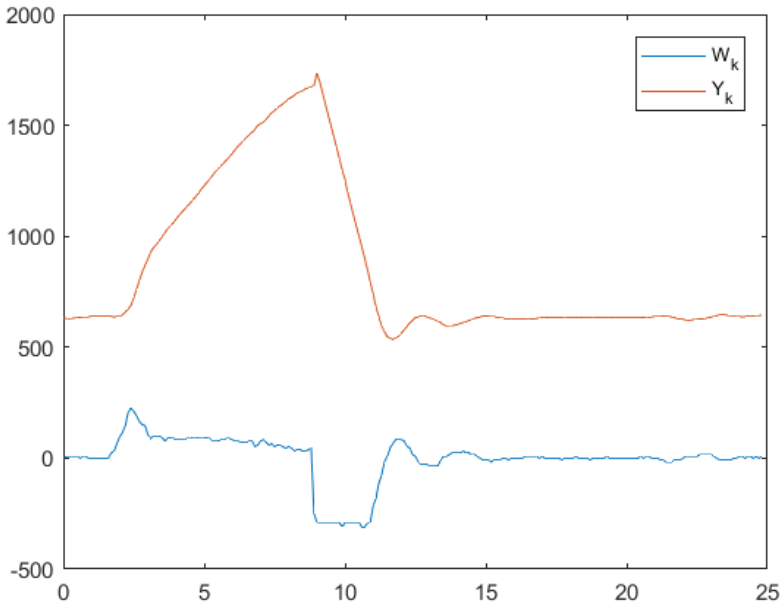

Figure 11: W_k and Y_k values of the device while pressure on motor shaft.

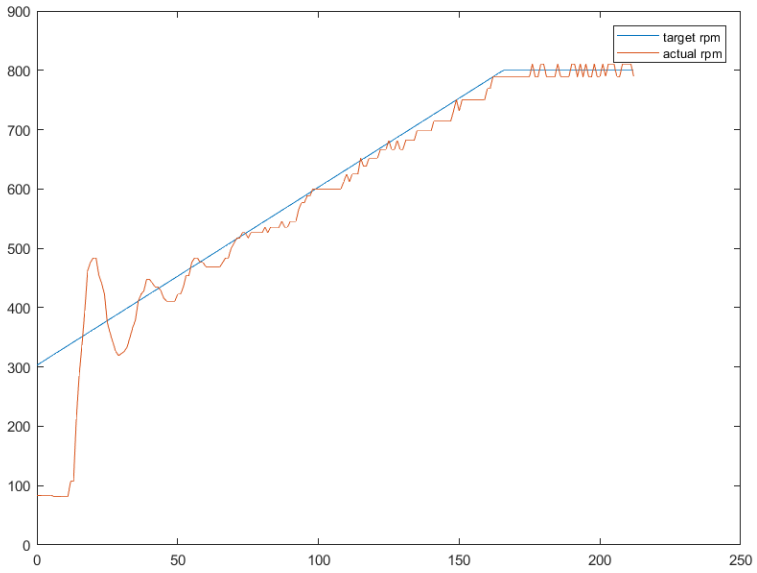

Figure 12: Ramp following in real time.

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`

8 Arduino Code

```
// EE572 control system
// Matt Ruffner
// Alex Mueller
// Ben Cummins
// Galvin Greene

#include <TimerOne.h>

#define IRPIN A9
#define MOTOR 6

// MOTOR PWM VALUES WERE ITERATED THROUGH, WITH THE CORRESPONDING RESULTING
// RPM VALUE LOGGED. THESE VALUES WERE LOADED INTO MATLAB AND FIT WITH A
// POLYNOMIAL. THE FOLLOWING ARRAY IS THE POLYNOMIAL EVALUATED AT VARIOUS
// PWM VALUES TO GET A SMOOTHER SPEED CURVE.
// CODE TO GENERATE THIS TABLE CAN BE FOUND IN speed_fit.m in the
// EE572 REPOSITORY

PROGMEM float RPMMap[175] = {
2.847108e+02,2.918626e+02,2.989418e+02,3.059491e+02,3.128849e+02,
3.197497e+02,3.265440e+02,3.332684e+02,3.399233e+02,3.465093e+02,
3.530267e+02,3.594762e+02,3.658583e+02,3.721733e+02,3.784219e+02,
3.846046e+02,3.907218e+02,3.967740e+02,4.027618e+02,4.086856e+02,
4.145460e+02,4.203434e+02,4.260784e+02,4.317514e+02,4.373630e+02,
4.429136e+02,4.484038e+02,4.538341e+02,4.592049e+02,4.645168e+02,
4.697702e+02,4.749657e+02,4.801038e+02,4.851849e+02,4.902096e+02,
4.951784e+02,5.000917e+02,5.049501e+02,5.097541e+02,5.145042e+02,
```

```
5.192008e+02,5.238446e+02,5.284359e+02,5.329753e+02,5.374632e+02,
5.419003e+02,5.462870e+02,5.506237e+02,5.549110e+02,5.591495e+02,
5.633395e+02,5.674816e+02,5.715763e+02,5.756241e+02,5.796255e+02,
5.835810e+02,5.874911e+02,5.913564e+02,5.951772e+02,5.989541e+02,
6.026877e+02,6.063783e+02,6.100266e+02,6.136330e+02,6.171979e+02,
6.207221e+02,6.242058e+02,6.276496e+02,6.310541e+02,6.344197e+02,
6.377469e+02,6.410363e+02,6.442882e+02,6.475033e+02,6.506821e+02,
6.538249e+02,6.569324e+02,6.600051e+02,6.630433e+02,6.660478e+02,
6.690188e+02,6.719570e+02,6.748628e+02,6.777368e+02,6.805794e+02,
6.833912e+02,6.861726e+02,6.889242e+02,6.916464e+02,6.943398e+02,
6.970049e+02,6.996421e+02,7.022519e+02,7.048349e+02,7.073916e+02,
7.099224e+02,7.124279e+02,7.149086e+02,7.173649e+02,7.197974e+02,
7.222066e+02,7.245930e+02,7.269570e+02,7.292992e+02,7.316201e+02,
7.339201e+02,7.361999e+02,7.384598e+02,7.407004e+02,7.429222e+02,
7.451256e+02,7.473113e+02,7.494796e+02,7.516312e+02,7.537664e+02,
7.558858e+02,7.579900e+02,7.600793e+02,7.621543e+02,7.642155e+02,
7.662634e+02,7.682985e+02,7.703214e+02,7.723324e+02,7.743322e+02,
7.763211e+02,7.782998e+02,7.802686e+02,7.822282e+02,7.841790e+02,
7.861216e+02,7.880563e+02,7.899838e+02,7.919044e+02,7.938189e+02,
7.957275e+02,7.976308e+02,7.995294e+02,8.014237e+02,8.033143e+02,
8.052016e+02,8.070861e+02,8.089683e+02,8.108488e+02,8.127280e+02,
8.146065e+02,8.164847e+02,8.183632e+02,8.202424e+02,8.221228e+02,
8.240050e+02,8.258894e+02,8.277766e+02,8.296671e+02,8.315613e+02,
8.334597e+02,8.353629e+02,8.372714e+02,8.391856e+02,8.411061e+02,
8.430334e+02,8.449679e+02,8.469102e+02,8.488608e+02,8.508201e+02,
8.527887e+02,8.547670e+02,8.567557e+02,8.587551e+02,8.607658e+02,
8.627883e+02,8.648230e+02,8.668706e+02,8.689314e+02,8.710060e+02
};

// VARS FOR DETERMINING CURRENT RPM
unsigned long lastFreqUpdate = 0;
float rpm = 0;
int state = 0;
unsigned long lastLowTime = 0;
volatile unsigned long diffTime = 0;

// CONTROL PARAMETER
double Setpoint;

// PID STATE VARS
float yk=0.0;
float yk_minus1=0.0;
float yk_minus2=0.0;
float wk=0.0;
float wk_minus1=0.0;
float wk_minus2=0.0;

// PID TF COEFFICIENTS
float b0=0.9842;
float b1=-1.1461;
float b2=0.3337;
```

```
float a0=1.0;
float a1=-1.0;
float a2=0.0;

unsigned long startTime = 0;


void setup() {
  // PIN DIRECTION DEFINITIONS
  pinMode(IRPIN, INPUT);
  pinMode(MOTOR, OUTPUT);
  pinMode(13, OUTPUT); // LED

  Serial.begin(115200);
  delay(2000);

  // RPM SENSING SETUP
  // 1 MILLISECOND SAMPLING PERIOD REQUIRED
  // TO GET ACCURATE RPM SENSING
  Timer1.initialize(1000);
  // ISR CALLBACK
  Timer1.attachInterrupt(checkSensor);
  // TIMESTAMP FOR RPM SENSING
  lastLowTime = millis();

  // INITIAL LOW MOTOR SPEED
  analogWrite(MOTOR, 80);

  unsigned long n=millis();
  while(millis()-n < 1000){
    digitalWrite(13, HIGH);
    delay(100);
    digitalWrite(13, LOW);
    delay(100);
  }

  // INITIAL TARGET RPM OF 300
  Setpoint = 300;

  wk=rpm;

  startTime = millis();
}

// 3 STATE THRESHOLD BASED RPM SENSING FOR
// ANALOG REFLECTANCE SENSOR
void checkSensor() {
  int val = analogRead(IRPIN) >> 5;
  if( state == 0 ){
    if( val < (600 >> 5) ){
      state = 1;
```

```
    }
  } else if( state == 1) {
    if( val > (700>>5) ){
      lastLowTime = millis();
      state = 2;
    }
  } else if( state == 2) {
    if( val < (600>>5) ){
      diffTime = millis() - lastLowTime;
      state = 0;
    }
  }
}


void loop() {

  // RUN CODE INSIDE THIS IF EVERY 100 MILLISECONDS (Ts)
  if( millis() - 100 > lastFreqUpdate ){
    if( Serial.available() ){
      Setpoint = Serial.parseInt();
    }

    // UNCOMMENT TO FOLLOW A RAMP
    // Setpoint = Setpoint + 3;
    // if( Setpoint > 800 ) Setpoint = 800;

    // UPDATE TIMESTAMP TO KEEP TRACK OF Ts
    lastFreqUpdate = millis();
    unsigned long temp = 0;
    // ATOMIC READ OF VOLATILE VARIABLE INSIDE ISR
    noInterrupts();
    temp = diffTime;
    interrupts();

    // COMPUTE CURRENT RPM
    rpm = 1.0/(2.0*(float)(diffTime)*0.001) * 60;

    // COMPUTE THE OUTPUT
    yk=(b0*wk+b1*wk_minus1+b2*wk_minus2-a1*yk_minus1-a2*yk_minus2)/(a0);

    // SET THE INPUT
    wk=Setpoint-rpm;

    double targetRPM = yk;

    // TAB DELIMITTED SERIAL LOGGING FOR PLOTTING IN MATLAB
    Serial.print(Setpoint);Serial.print("\t");
    Serial.print(rpm);Serial.print("\t");
    Serial.print(wk);Serial.print("\t");
    Serial.println(yk);
```

```
    // LOOKUP TARGET RPM
    for( int i=0;i<174;i++ ){
      if( RPMMap[i] <= targetRPM && RPMMap[i+1] > targetRPM ){
        // DRIVE MOTOR AT NEW SPEED
        analogWrite(MOTOR, i+81);
      }
    }

    // UPDATE INPUT/OUTPUT STATE VARIABLES
    wk_minus2=wk_minus1;
    wk_minus1=wk;
    yk_minus2=yk_minus1;
    yk_minus1=yk;
  }

}
```

9 Simulation Code

```
% Alex Mueller
% Galvin Greene, Ben Cummins, Matt Ruffner
% EE 572 Final Project Simulation

% Purpose: To simulate our 1st order type zero system when
% subject to ramp and step inputs. Also we wish to compute PID parameters
% and discrete time compensator filter parameters
% to form the compensated system for simulation and programing of the system.

close all

syms s

%motor parameters
k = 810; % dc gain (max of 810 rpm)
tau = 8; % time constant ( time to get to .632 of max speed)

Ts = .1; % sampling time

Mp = 4.32; % max percent overshoot
m = Mp/100;

% compute zeta based on given specs
zeta = -log(m)/sqrt((pi^2)+log(m)^2);

ts = 1; % desired settling time (seconds)
wnzeta = 4/ts;
wn = wnzeta/zeta;

% define system and compensator functions
```

```
gzoh = 1/(1+(Ts/2)*s);
g = k/(1+tau*s);

% desired dominant closed-loop pole
s1 = -wnzeta + 1i*wn*sqrt(1-(zeta^2));

% find transfer function of the system before compensation
H1a = gzoh*g;
H1b = syms2tf(H1a);

% simulate uncompensated system step response
step(H1b)
title('Step Response of System Before Compensation')
ylabel('RPM')

%generate a ramp input
len =85;
u1 = ones(1,len);
for i = 1:len
    u1(i) = 10*i*u1(i);
end
t = 0:Ts:Ts*len-Ts;
x0 = 0;

% simulate uncompensated system with the ramp input
y = lsim(H1b,u1,t,x0,'zoh');

% plot simulation
figure
plot(t,y,t,u1)
ylim([-.1 2000])
xlim([0 2])
legend('System Response','Ramp Input')
title('Ramp Response of System Before Compensation')
ylabel('RPM')

s = s1;
gzohg = double(abs(subs(g*gzoh/s1)));
ang_def = 180-((180*angle(double(subs(g)*subs(gzoh))/s1)/pi));
half_ang = ang_def/2;
Zc = (imag(s1)/tan(half_ang*pi/180))-real(s1);
Kc = 1/(abs(s1+Zc)^2+gzohg);
Kd = Kc;
Ki = Kc*Zc^2;
Kp = 2*Zc*Kc;

syms s

% symbolic pid function
gpid = (Ki+Kp*s+Kd*s^2)/s;
```

```
% find closed-loop transfer function of the compensated system
H2a = (gzoh*g*gpid)/(1+gzoh*g*gpid);
H2b = syms2tf(H2a);

% simulate compensated system step response
figure
step(H2b)
title('Step Response of System After Compensation')
ylabel('RPM')

% simulate compensated system ramp response
y2 = lsim(H2b,u1,t,x0,'zoh');
% plot simulation
figure
plot(t,y2,t,u1)
legend('System Response','Ramp Input')
title('Ramp Response of System After Compensation')
ylabel('RPM')


% discretize system
syms z
s = (z-1)/(z*Ts); % use rectanglar estimation holding from right side
H_z = subs(gpid);
[symNum,symDen] = numden(H_z); %Get num and den of Symbolic TF
TFnum = sym2poly(symNum);    %Convert Symbolic num to polynomial
TFden = sym2poly(symDen);    %Convert Symbolic den to polynomial
H_z_tf =tf(TFnum,TFden,Ts);
[num,den,Ts1] = tfdata(H_z_tf,'v');

factor = den(1);

den = [den 0];

num = num/factor;
den = den/factor;

% assign filter coefficients
b0 = num(1);
b1 = num(2);
b2 = num(3);
a0 = den(1);
a1 = den(2);
a2 = den(3);



% Convert Symbolic Transfer Function to ZPK Transfer Function
% Crystal Nassouri 2009
% Allows for substitution/manipulation that can only be done with syms
%
```

```
% Ex: Gs = syms2tf(G)
% Where G is a symbolic equation and Gs is a zpk transfer function

function[ans] = syms2tf(G)
[symNum,symDen] = numden(G); %Get num and den of Symbolic TF
TFnum = sym2poly(symNum);    %Convert Symbolic num to polynomial
TFden = sym2poly(symDen);    %Convert Symbolic den to polynomial
ans =tf(TFnum,TFden);
```