College of Engineering
School of Aerospace Engineering

AE 6705: Introduction to Mechatronics

LAB10

Professor:
Jonathan Rogers
Gtech AE Professor

Student:
Tomoki Koike
Gtech MS Student

November 28, 2021

Table of Contents

1 Question 1 **2**

2 Question 2 **7**

3 Question 3 **11**

4 Question 4 **14**

5 Appendix **15**

5.1 MATLAB Code 15

Question 1

Pick two motor velocities, ω_1 and ω_2, greater than 90 rpm. Use your closed loop Simulink model to simulate the step response for each velocity. On each plot highlight the rise time and overshoot to show that they meet the specified requirements (you might want to look at Matlab's `stepinfo()` function for this). State the PID gains you used for these simulations.

Solution:
The motor velocities chosen are

$$\omega_1 = 95 \text{ RPM} \qquad \omega_2 = 120 \text{ RPM} .$$

The PID gains for the following model are

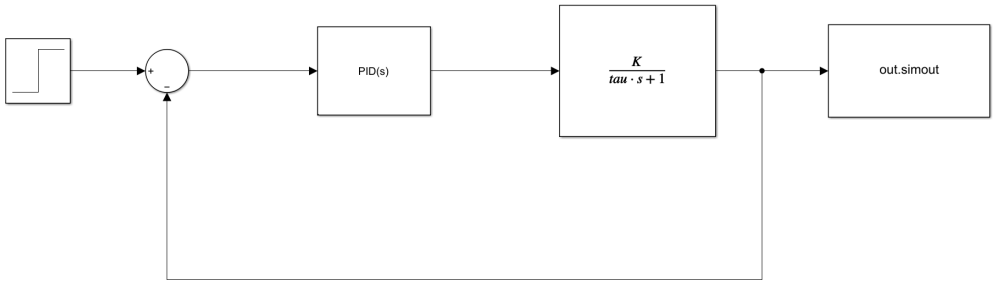

Figure 1: PID controlled feedback model

K_P	K_I	K_D
-0.00122960025714389	0.0382450209614726	0.000132060829813156

Table 1: PID gains tuned in Simulink

The plotted response as well as the characteristics are shown below

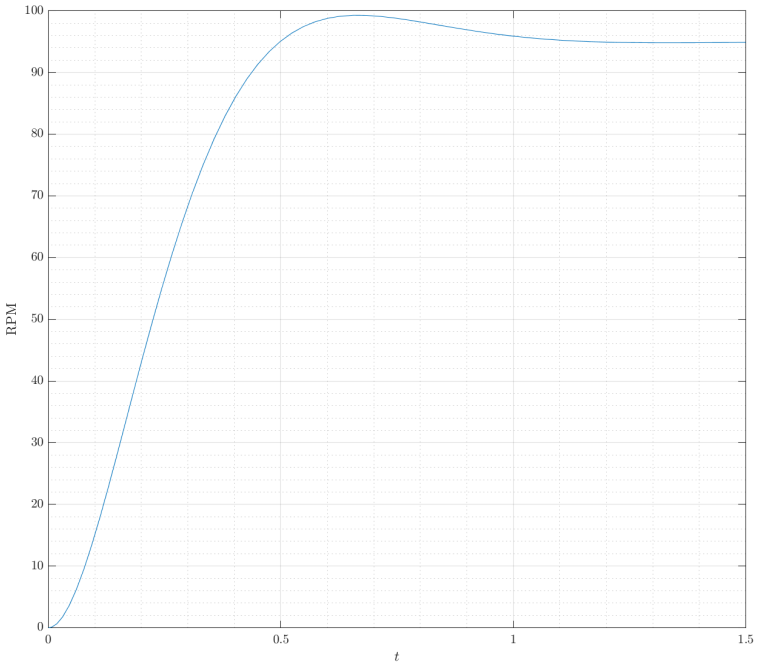

Figure 2: Step response for Simulink model for $\omega_1 = 95$

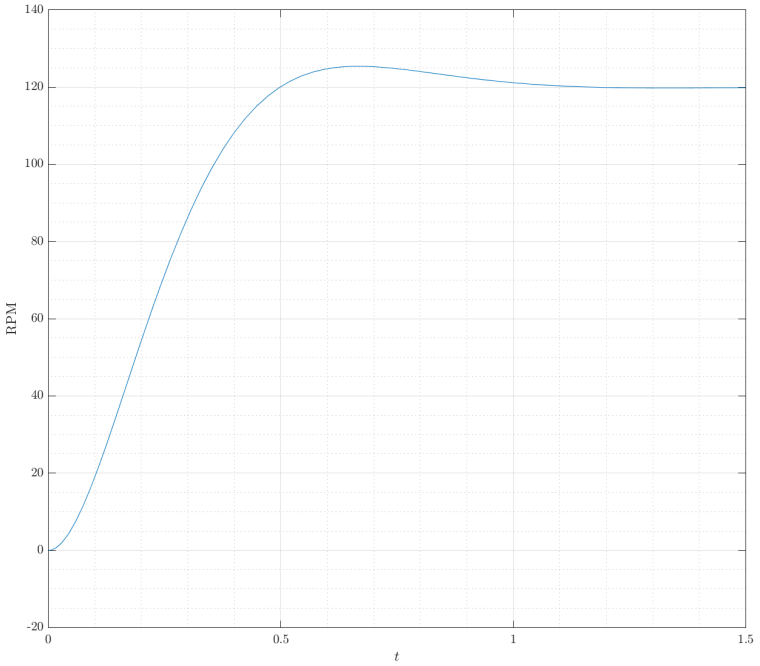

Figure 3: Step response for Simulink model for $\omega_2 = 120$

stepinfo data

```
omega1 = 95 RPM

         RiseTime: 0.3216
    TransientTime: 0.9124
     SettlingTime: 0.9124
      SettlingMin: 86.1496
      SettlingMax: 99.2648
        Overshoot: 4.6256
       Undershoot: 1.5809e-40
             Peak: 99.2648
         PeakTime: 0.6559

omega2 = 120 RPM

         RiseTime: 0.3214
    TransientTime: 0.9123
     SettlingTime: 0.9123
      SettlingMin: 108.1769
      SettlingMax: 125.3781
        Overshoot: 4.6181
       Undershoot: 1.5809e-40
             Peak: 125.3781
         PeakTime: 0.6800
```

We can confirm that both satisfy the requirement of under 5% overshoot and 500 ms rise time.

It is important to note that these results were produced by using the MATLAB Control System Tuner. Where we define a tuning goal for step response tracking with the following performance requirements

$$\text{Max Overshoot } := MOS = 4.5\% \quad \longrightarrow \quad \zeta = \frac{-\ln(MOS/100)}{\sqrt{\pi^2 + \ln^2(MOS/100)}} = 0.7025$$

$$\text{if rise time } := t_r = 0.35$$

$$\because t_r \omega_n = \frac{1}{\sqrt{1-\zeta^2}} \left[\pi - \arctan\left(\frac{\sqrt{1-\zeta^2}}{\zeta} \right) \right] \quad \text{and} \quad \text{time constant } := \tau \approx \frac{1}{\zeta \omega_n}$$

$$\therefore \tau = 0.1509.$$

Then we plug these into the settings for the tuning goals like in the following Figure. Then the app will optimize the PID gains.

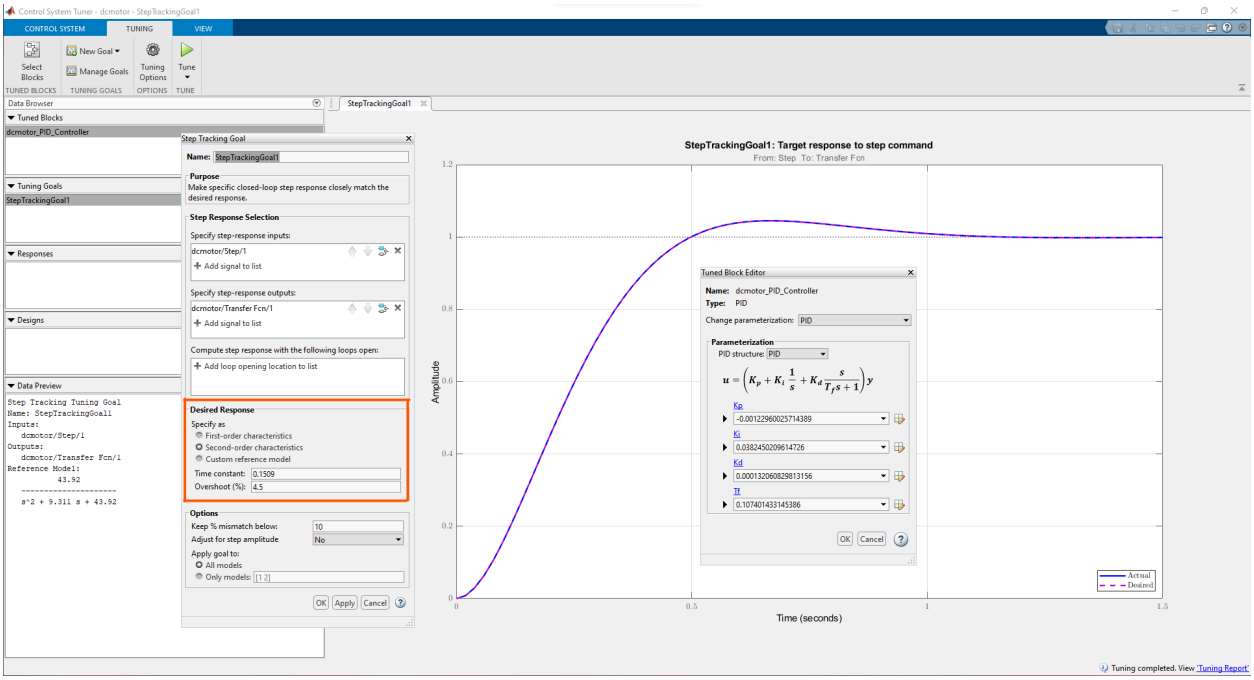

Figure 4: Tuning process with Control System Tuner

Also, we must keep in mind that the PID controller of the Simulink is actually in the form of

$$K_p + \frac{K_i}{s} + K_d \frac{Ns}{s+N}$$

where N is the filter coefficient with a value of

$$N = 9.31086272048467.$$

To implement this in MSP432, we code a high pass filter for the derivative control like in the code of Listing 1. Although, this is in the code, it is not implemented and the first order finite differencing is used instead.

```c
/*
 * Function: derivative_filter
 * ---------------------------
 * Compute output from derivative filter/high pass filter for
 * derivative control.
 *
 * Arguments:
 *   [float] de1: previous error_dot value
 *   [float] de2: current error_dot value
 *   [float] pre_out: previous output value from the filter
 * Returns:
 *   [float] filtered output value
 */
float derivative_filter(float de1, float de2, float pre_out){
    float alpha = (1/N) / ((1/N) + TIME_STEP);
    float yout = alpha * pre_out + alpha * (de2 - de1)*Kd*N;
    return yout;
}
```

Listing 1: Derivative filter c code implementation

Question 2

Now using your *hardware* setup and the same PID gains used in simulation, control the DC motor to each speed of ω_1 and ω_2 starting from zero speed each time. Record the step response for each case. Plot the experimental step response (two plots, one for each case) and highlight the overshoot and the rise time. Does it match the required performance specifications? Now create two new plots where you overlay the simulated and experimental response. How similar is the experimental response to the simulated one? If the simulated and experimental responses look noticeably different, explain why you think that is the case.

Solution:
For $\omega_1 = 95$

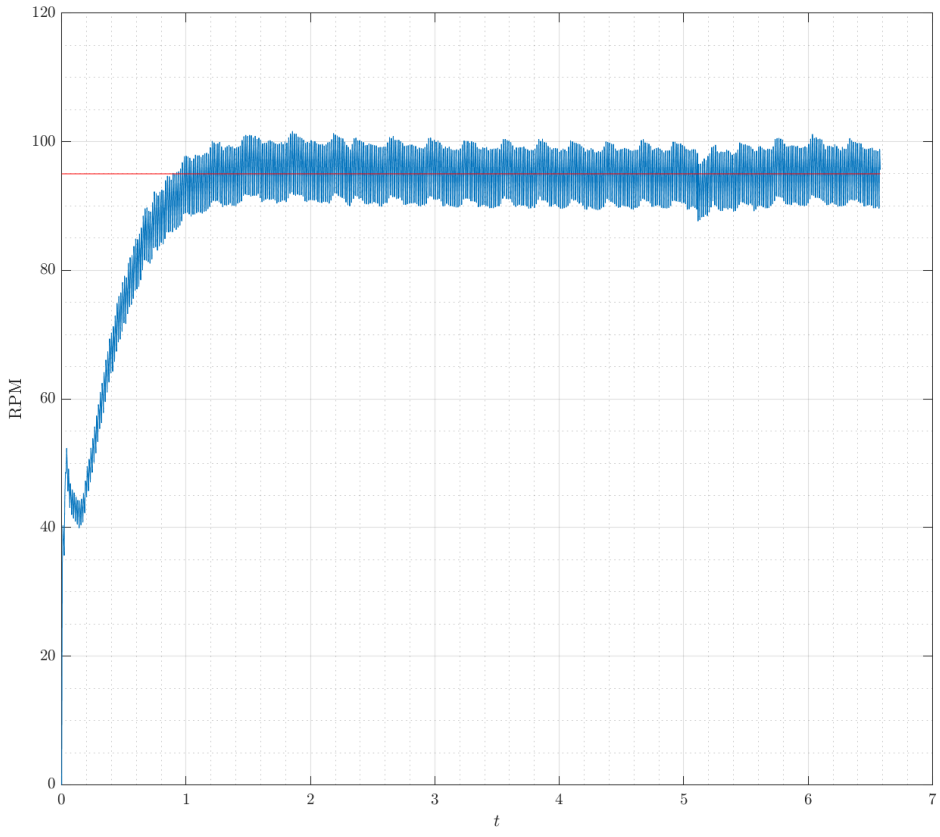

Figure 5: Actual response at $\omega = 95$ RPM with Simulink PID gains

For $\omega_2 = 120$

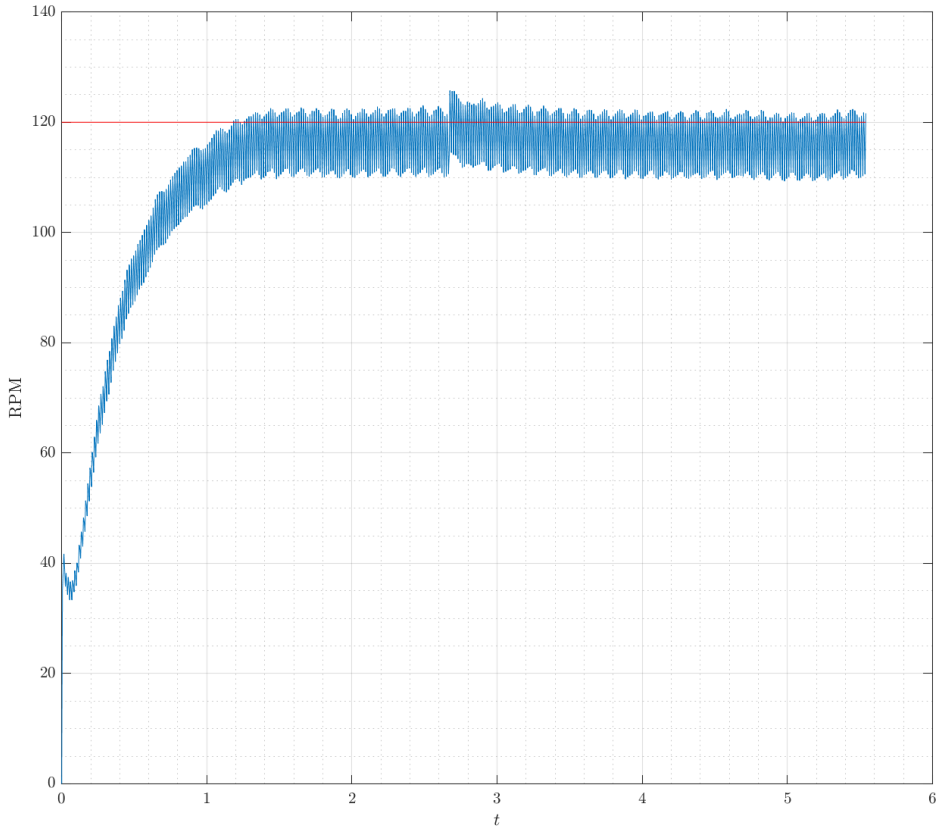

Figure 6: Actual response at $\omega = 120$ RPM with Simulink PID gains

From below we can see that the rise time and overshoot for $\omega_1 = 95$ RPM are not satisfied but for $\omega_2 = 120$ the overshoot is below 5% and the rise time is above 500 ms.

`stepinfo data`

```
omega1 = 95 RPM

         RiseTime: 0.6383
    TransientTime: 6.5770
     SettlingTime: 6.5770
      SettlingMin: 80.0777
      SettlingMax: 101.5857
        Overshoot: 6.2170
       Undershoot: 0
             Peak: 101.5857
         PeakTime: 1.8550
```

```
omega2 = 120 RPM

          RiseTime: 0.7482
    TransientTime: 5.5389
     SettlingTime: 5.5389
      SettlingMin: 100.3019
      SettlingMax: 125.7242
        Overshoot: 3.3694
       Undershoot: 0
             Peak: 125.7242
         PeakTime: 2.6750
```

If we overlay the plots from problem 1 and problem 2 we have

For $\omega_1 = 95$

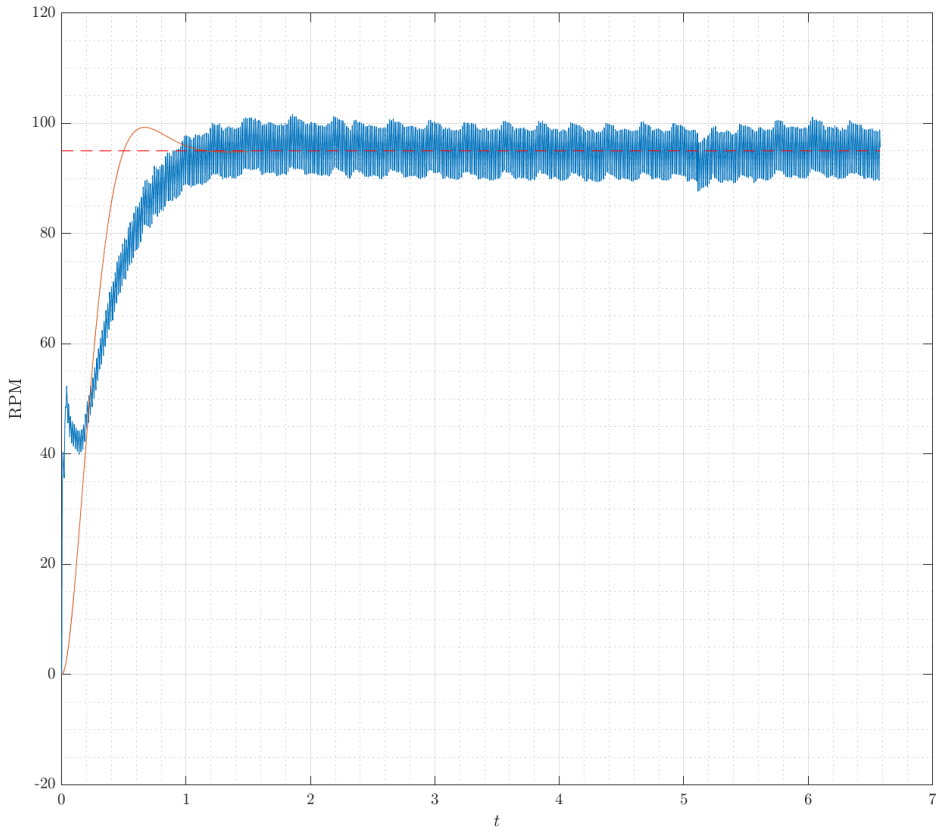

Figure 7: Actual response and simulation response at $\omega = 95$ RPM with Simulink PID gains

For $\omega_2 = 120$

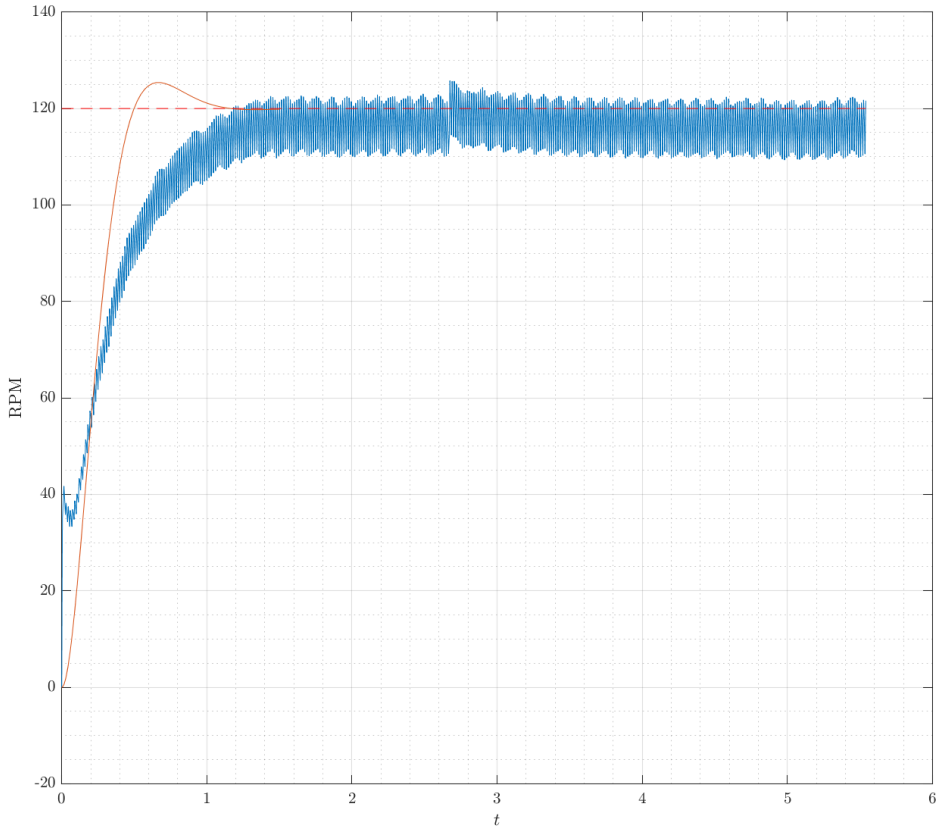

Figure 8: Actual response and simulation response at $\omega = 120$ RPM with Simulink PID gains

Observing these plots we can see that the response seem somewhat close but the overshoot that occurs for a second order system seems to be less obvious for the actual motor response compared to the simulation response. For the higher RPM value we can see that there seems to be a bit of bias at the steady state response and for the both there is a significant amount of noise which could be caused by the amplification of error by the derivative and proportional terms. Additionally, we see a small hump at the beginning of the response for the actual motor.

Question 3

If your experimental response does not meet the required rise time and overshoot specifications, tune your PID gains until it does. You might want to start by setting all gains to 0 and tuning your integral gain first, then your proportional gain, and finally your derivative gain. Note, the final gains might differ significantly from your simulation gains.

Solution:
We employ the following PID gains and redo the analysis

K_P	K_I	K_D
0.00572311362145534	0.120364997080663	-0.000158546538651373

Table 2: New PID gains tuned in Simulink

This produces the following results

`stepinfo data`

```
omega1 = 95 RPM

         RiseTime: 0.1256
    TransientTime: 1.8351
     SettlingTime: 1.5603
      SettlingMin: 86.4440
      SettlingMax: 99.4873
        Overshoot: 4.8281
       Undershoot: 0
             Peak: 99.4873
         PeakTime: 0.4300


omega2 = 120 RPM

         RiseTime: 0.3072
    TransientTime: 1.6532
     SettlingTime: 1.0519
      SettlingMin: 104.0249
      SettlingMax: 115.6861
        Overshoot: 0.0299
       Undershoot: 0
             Peak: 115.6861
         PeakTime: 1.9600
```

For $\omega_1 = 95$

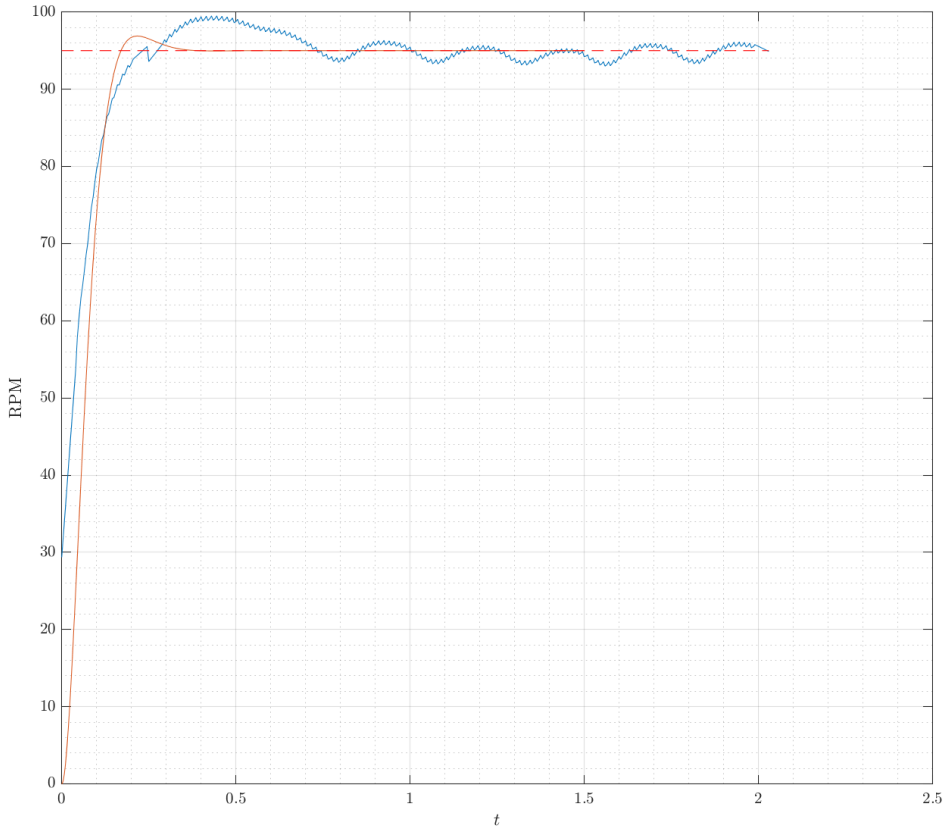

Figure 9: Actual response and simulation response at $\omega = 95$ RPM with new Simulink PID gains

For $\omega_2 = 120$

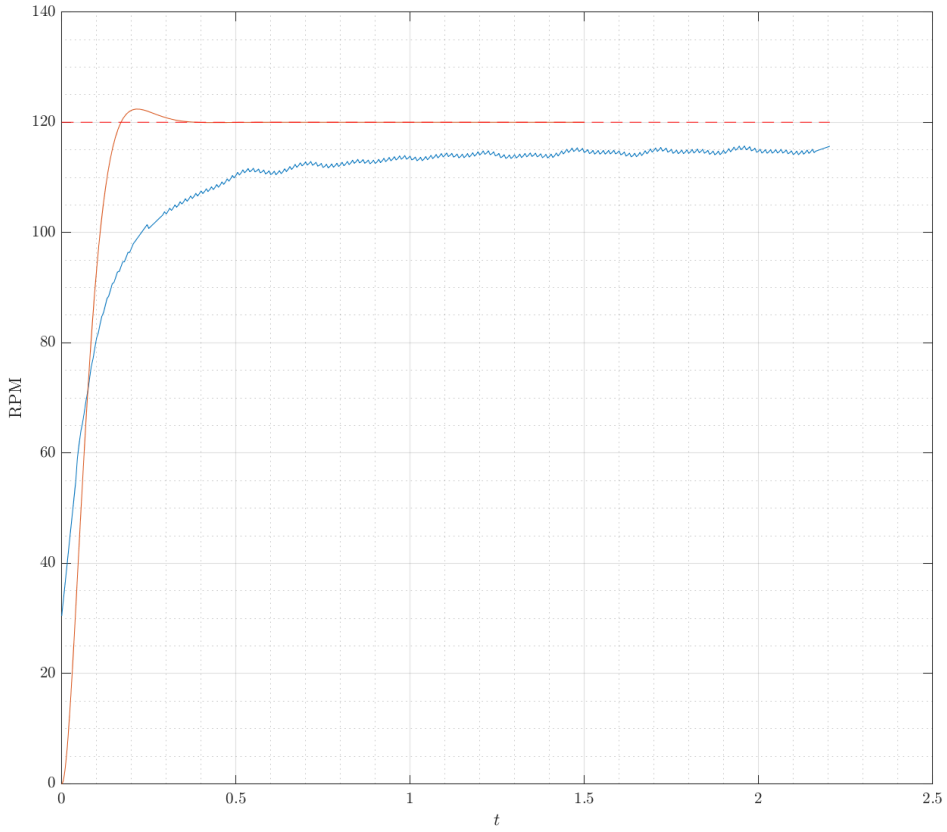

Figure 10: Actual response and simulation response at $\omega = 120$ RPM with new Simulink PID gains

The results show that we have satisfied the requirements. These results were produced by implementing the Savitzky-Golay filtering with `sgolayfilt()` in MATLAB. But still we have a bias for the $\omega = 120$ RPM. This means that if we are to further improve the response of this motor we will have to employ a noise filter and calibrate the RPM for high values.

Question 4

Use the controller gains from Question 3 to control the motor speed to a value lower than 80 rpm. What do you notice in the step response? Explain the reason behind the behavior observed in your response.

Solution:
With $\omega = 60$ we have

$$t_r = 0.0391$$
$$MOS = 45.6143$$

and the overshoot is very high. In the previous lab we have seen that motor plant itself can be computed to be different with different K value and a different time constant when simulating with different duty cycles. From this we can deduce that with a plant computed for a duty cycle of 100% might deviate too much from the actual plant for when the duty cycles are low. That is the gain K value might be too high and that would amplify the response and the noise too much and result in a very high overshoot.

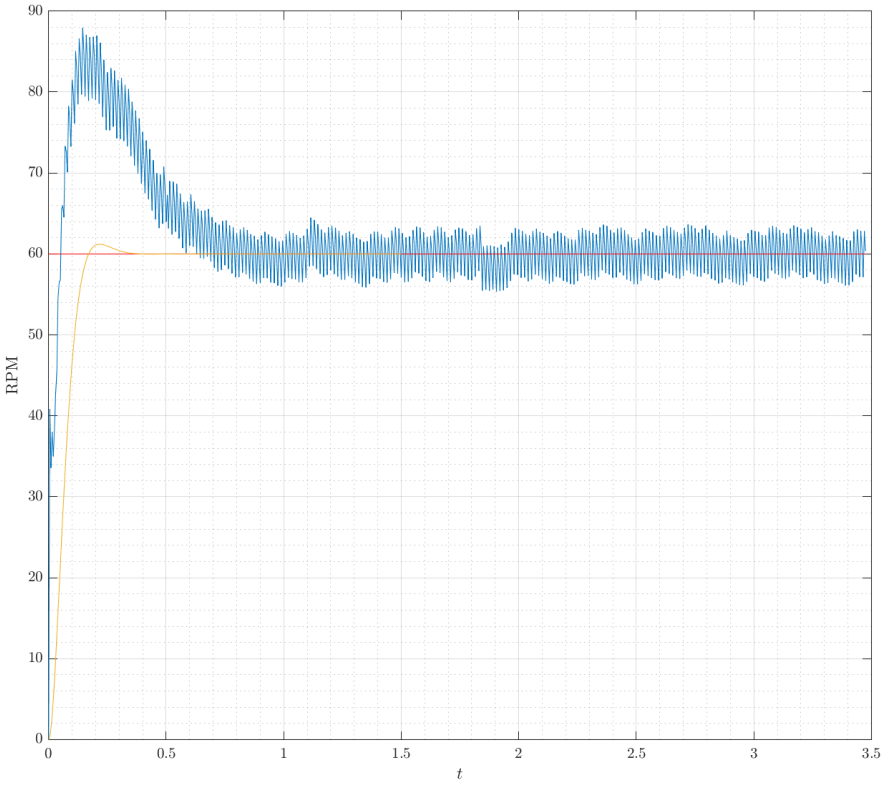

Figure 11: Overlaid results for when $\omega = 60$

Appendix

5.1 MATLAB Code

```
% AE6705 Lab10 MATLAB code
% Tomoki Koike
clear all; close all; clc;  % housekeeping commands
set(groot, 'defaulttextinterpreter','latex');
set(groot, 'defaultAxesTickLabelInterpreter','latex');
set(groot, 'defaultLegendInterpreter','latex');
%%
% Gain and time constant from lab9
K = 1.233265815820895e+02;
tau = 0.075250836120401;
%%
% Max overshoot less than 5% -> 4.5%
MOS = 4.5;
zeta = -log(MOS/100)/sqrt(pi^2 + log(MOS/100)^2);  % 0.702504592260657
% Rise time less than 0.5 seconds -> 0.35 seconds
tr = 0.35;
temp = 1/sqrt(1-zeta^2) * (pi - atan(sqrt(1-zeta^2)/zeta));
time_const = tr / zeta / temp;

% Tuned PID gains
% p2
% Kp = -0.00122960025714389;
% Ki = 0.0382450209614726;
% Kd = 0.000132060829813156;
% N = 9.31086272048467;
%
% p3
Kp = 0.00572311362145534;
Ki = 0.120364997080663;
Kd = -0.000158546538651373;
N  = 36.097373491324220;
%%
% Simulate Simulink Model
omega1 = 95;  % first candidate angular velocity
omega2 = 120;  % second candidate angular velocity
i = 1;
for omega = [omega1 omega2]
    res = sim("dcmotor.slx");
    % Plotting
    fig = figure("Renderer","painters","Position",[60 60 950 800]);
```

```
        plot(res.tout, res.simout.signals.values);
        grid on; grid minor; box on;
        xlabel('$t$')
        ylabel('RPM')
    if omega == omega1
        saveas(fig,'outputs/model_response_p3_w=95.png');
    elseif omega == omega2
        saveas(fig,'outputs/model_response_p3_w=120.png');
    end
    output(i) = res;
    i = i + 1;
end
%%
% Response characteristics - transient and steady state
rc = stepinfo(res.simout.signals.values,res.tout)
%%
% omega = 95
% read data
data1 = csvread("p3_omega=95.csv");
data11 = data1(1:50);
data12 = data1(51:end);
% Filter the data a bit removing outliers
data11 = sgolayfilt(data11,1,17);
data12 = rmoutliers(data12);
data12 = sgolayfilt(data12,1,17);
data1 = [data11; data12];

tspan1 = 0:0.005:0.005*(length(data1)-1);
% Plotting
fig = figure("Renderer","painters","Position",[60 60 950 800]);
    plot(tspan1, data1);
    grid on; grid minor; box on; hold on;
    plot(tspan1, 95 * ones(1, length(data1)), 'r')
    hold off;
    xlabel('$t$')
    ylabel('RPM')
saveas(fig,'outputs/actual_response_p3_w=95.png');
%%
% Response analysis
rc95 = stepinfo(data1, tspan1)
%%
% omega = 120
% read data
data2 = csvread("p3_omega=120.csv");
data21 = data2(1:50);
```

```
data22 = data2(51:end);
% Filter the data a bit removing outliers
data21 = sgolayfilt(data21,1,17);
data22 = rmoutliers(data22);
data22 = sgolayfilt(data22,1,17);
data2 = [data21; data22];

tspan2 = 0:0.005:0.005*(length(data2)-1);
% Plotting
fig = figure("Renderer","painters","Position",[60 60 950 800]);
    plot(tspan2, data2);
    grid on; grid minor; box on; hold on;
    plot(tspan2, 120 * ones(1, length(data2)), 'r')
    hold off;
    xlabel('$t$')
    ylabel('RPM')
saveas(fig,'outputs/actual_response_p3_w=120.png');
%%
% Response analysis
rc120 = stepinfo(data2, tspan2)
%%
% Overlaid plots
res = output(1);
fig = figure("Renderer","painters","Position",[60 60 950 800]);
    plot(tspan1, data1);
    grid on; grid minor; box on; hold on;
    plot(res.tout, res.simout.signals.values)
    plot(tspan1, 95 * ones(1, length(data1)), '--r')
    hold off;
    xlabel('$t$')
    ylabel('RPM')
saveas(fig,'outputs/actual_response_overlaid_p3_w=95.png');
%%
res = output(2);
fig = figure("Renderer","painters","Position",[60 60 950 800]);
    plot(tspan2, data2);
    grid on; grid minor; box on; hold on;
    plot(res.tout, res.simout.signals.values)
    plot(tspan2, 120 * ones(1, length(data2)), '--r')
    hold off;
    xlabel('$t$')
    ylabel('RPM')
saveas(fig,'outputs/actual_response_overlaid_p3_w=120.png');
%%
% P4: omega = 60
```

```
% read data
data4 = csvread("p4_omega=60.csv");
data41 = data4(1:100);
data42 = data4(101:end);
% Filter the data a bit removing outliers
% data41 = sgolayfilt(data41,1,17);
data42 = rmoutliers(data42);
% data42 = sgolayfilt(data42,1,17);
data4 = [data41; data42];

tspan4 = 0:0.005:0.005*(length(data4)-1);
% Plotting
omega = 60;
res = sim("dcmotor.slx");

fig = figure("Renderer","painters","Position",[60 60 950 800]);
    plot(tspan4, data4);
    grid on; grid minor; box on; hold on;
    plot(tspan4, 60 * ones(1, length(data4)), 'r')
    plot(res.tout, res.simout.signals.values);
    hold off;
    xlabel('$t$')
    ylabel('RPM')
saveas(fig,'outputs/actual_response_p4_w=60.png');
%%
% Response analysis
rc60 = stepinfo(data4, tspan4)
```