Subject Code: EEET1368

Assignment Title: Lab 2 Report

Due date: April 18th, 2021

Group Details:

| Name | RMIT ID | Email |
|---|---|---|
| Aditya Prawira | S3859061 | S3859061@student.rmit.edu.au |
| Bryan Yu Chuan Ng | S3628223 | S3628223@student.rmit.edu.au |
| Ruchit Dilipkumar Sheth | S3816430 | S3816430@student.rmit.edu.au |

# Table of Content

## Section 1:

The goal of this section l is to observe the behavior of a PID controller with 3 different implementations/orientations.

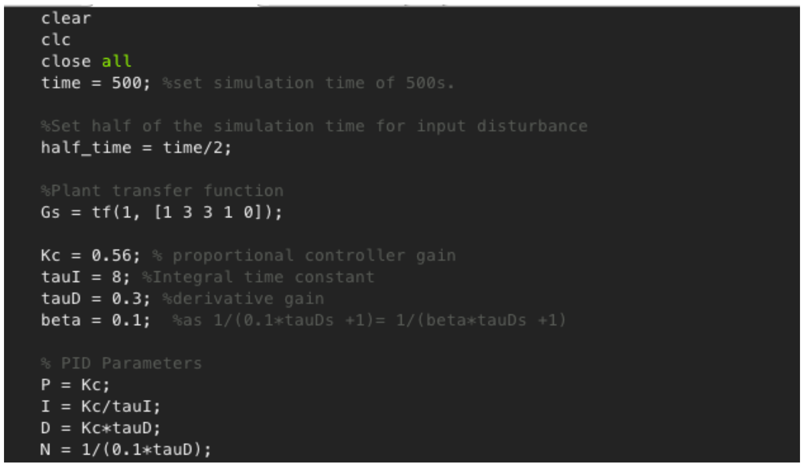
The proposed transfer function for the model's plant is represented as shown below

$$G(s) = \frac{1}{s(s+1)^3} = \frac{1}{(s^4 + 3s^3 + 3s^2 + s)}$$

The model proposed design specifications shown below:

- $K_c = 0.56 \rightarrow$ Proportional controller gain
- $\tau_I = 8 \rightarrow$ Integral time constant
- $\tau_D = 0.3 \rightarrow$ Derivative gain
- $\frac{1}{0.1\tau_D+1} \rightarrow$ a filter of the derivative controller
- With $t_{simulation} = 500s$ time of simulation
- The sample time chosen for this experiment would be $\Delta t = 0.1s$

Presented below is the MATLAB which will declare the required parameters for the PID controller in the Simulink simulation.



```
clear
clc
close all
time = 500; %set simulation time of 500s.

%Set half of the simulation time for input disturbance
half_time = time/2;

%Plant transfer function
Gs = tf(1, [1 3 3 1 0]);

Kc = 0.56; % proportional controller gain
tauI = 8; %Integral time constant
tauD = 0.3; %derivative gain
beta = 0.1;  %as 1/(0.1*tauDs +1)= 1/(beta*tauDs +1)

% PID Parameters
P = Kc;
I = Kc/tauI;
D = Kc*tauD;
N = 1/(0.1*tauD);
```

**Figure 1.** Global variable for Case A-C.

Additionally, sum square errors for each case must be calculated by the algorithm shown below:

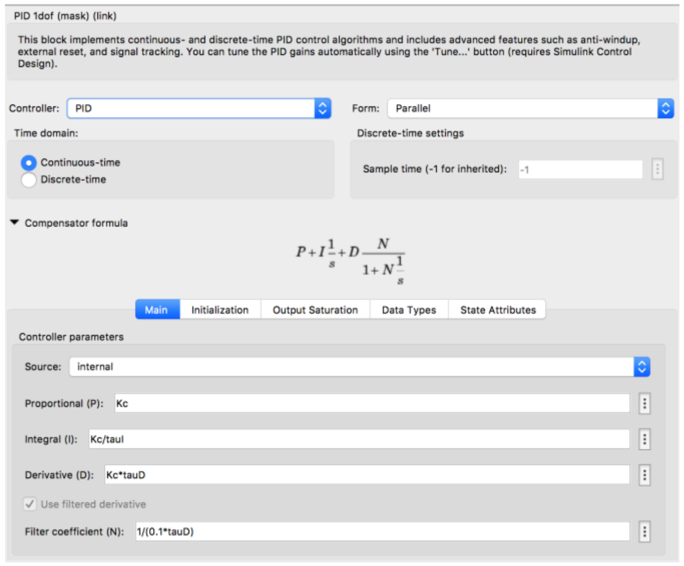
$$\sum_{i=1}^{M} (r(t_i) - y(t_i))^2 = \sum_{i=1}^{M} (e(t_i))^2$$

Finally, for all simulations 3 closed-loop systems conditions below must be applied:

- Reference step input signal of the amplitude of 6.
- Disturbance step input with amplitude of -3.
- Finally, measurement noise starts at $t = 0$ with the team/standard deviation of 0.001.

## Case A:

Firstly, the team would implement the standard/ideal PID controller and applied the aforementioned conditions above. The figures below would represent the implementation of the controller in Simulink.



PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID

Form: Parallel

Time domain: [✓ Continuous-time] Discrete-time

Discrete-time settings

Sample time (-1 for inherited): -1

Compensator formula

$$P+I\frac{1}{s}+D\frac{N}{1+N\frac{1}{s}}$$

Controller parameters

Source: internal

Proportional (P): Kc

Integral (I): Kc/taul

Derivative (D): Kc*tauD

✓ Use filtered derivative

Filter coefficient (N): 1/(0.1*tauD)

**Figure 2.** Configuration for the standard PID controller.

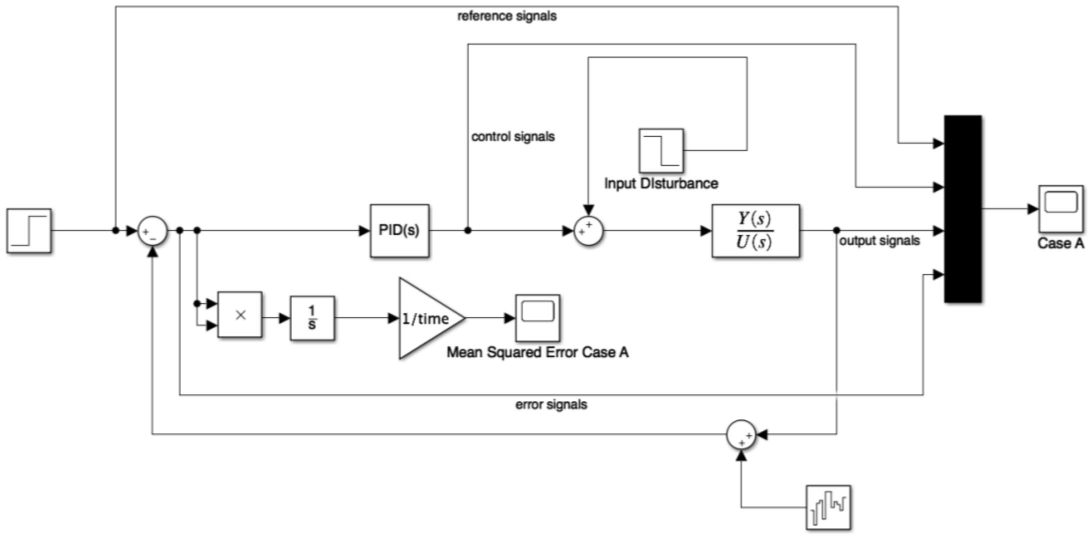


**Figure 3.** Standard PID controller (Simulink closed-loop diagram).

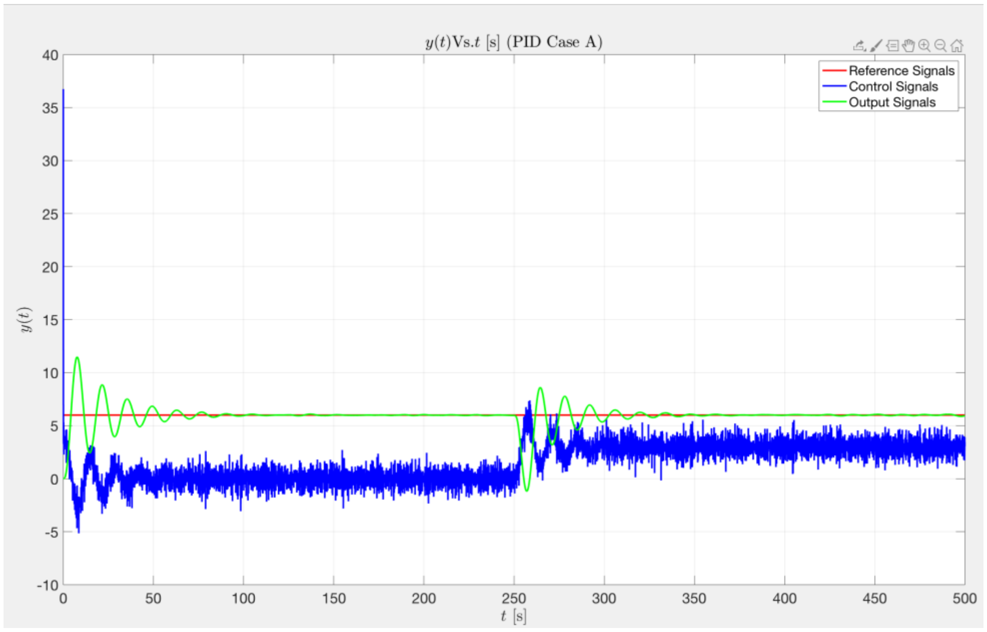


**Figure 4.** Simulink output for reference, control, and output signals for Case A.

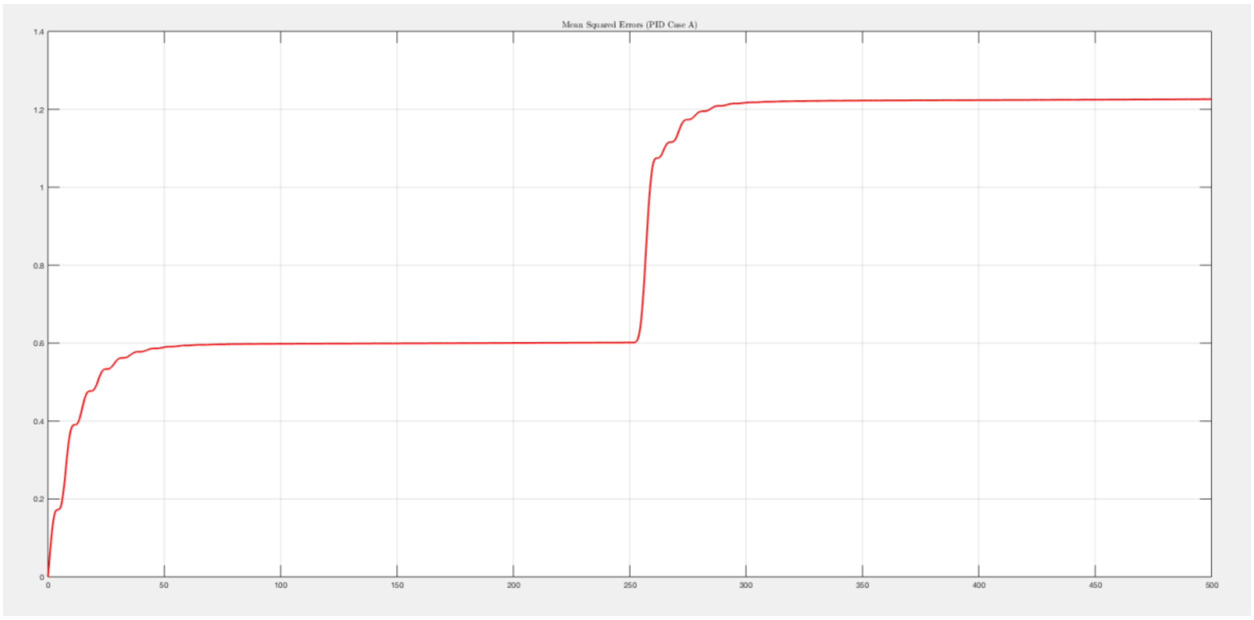
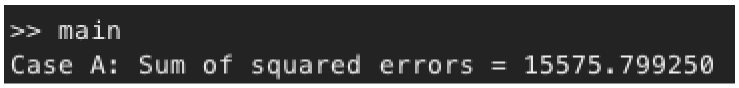


**Figure 5.** Mean Squared Error plot for Case A.

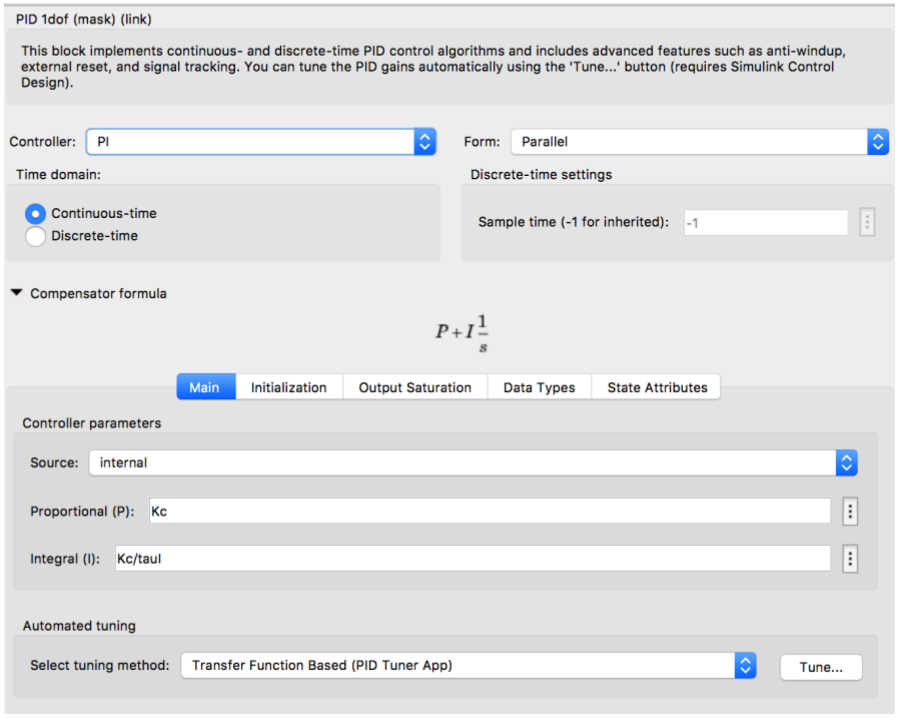


```
>> main
Case A: Sum of squared errors = 15575.799250
```

**Figure 6.** Sum of squared error for Case A (see appendix **A.1.1**).

## Case B:

PI controller on error signals and derivative (D) controller on the output.



PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PI

Form: Parallel

Time domain: [✓ Continuous-time] Discrete-time

Discrete-time settings

Sample time (-1 for inherited): -1

Compensator formula

$$P+I\frac{1}{s}$$

Controller parameters

Source: internal

Proportional (P): Kc

Integral (I): Kc/taul

Automated tuning

Select tuning method: Transfer Function Based (PID Tuner App)

**Figure 7.** Configuration for PI controller.

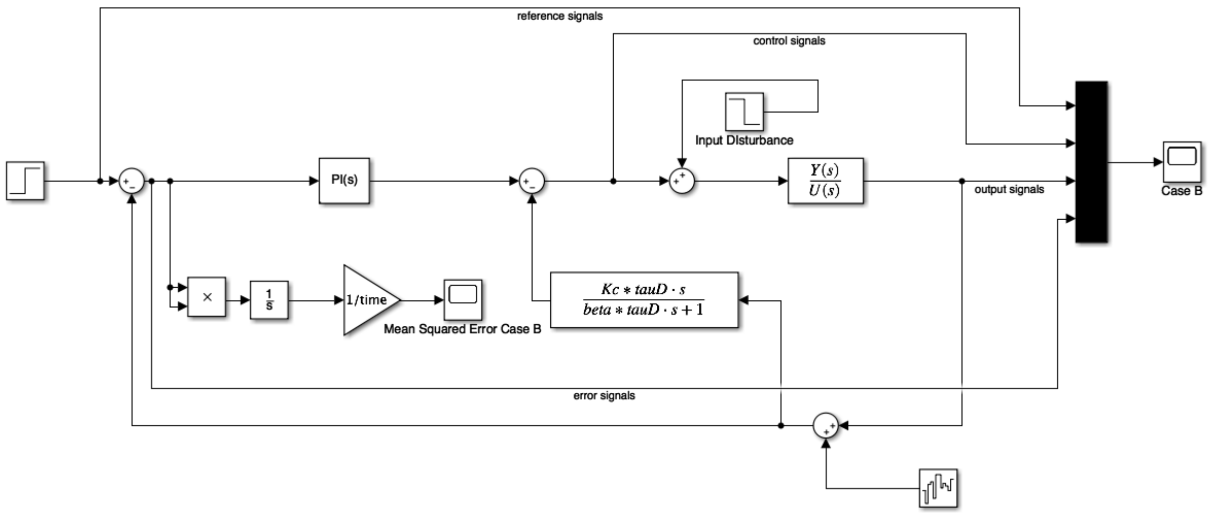


**Figure 8.** PI controller on error signals and derivative controller on the output signal (Simulink closed-loop diagram).

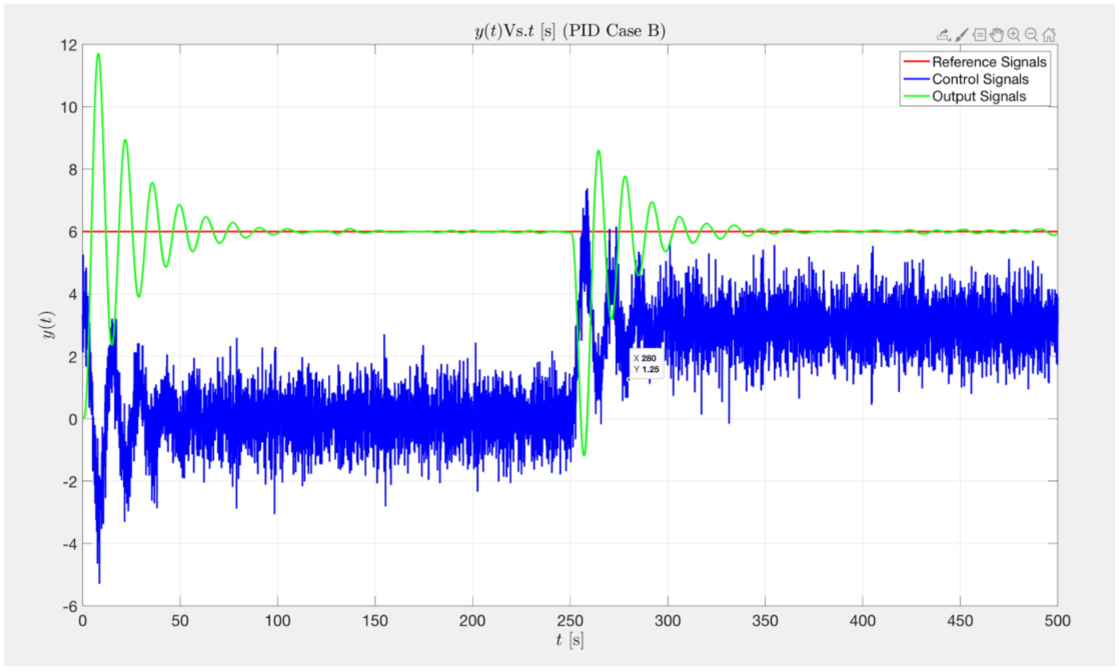


**Figure 9.** Simulink output for reference, control, and output signals for Case B.

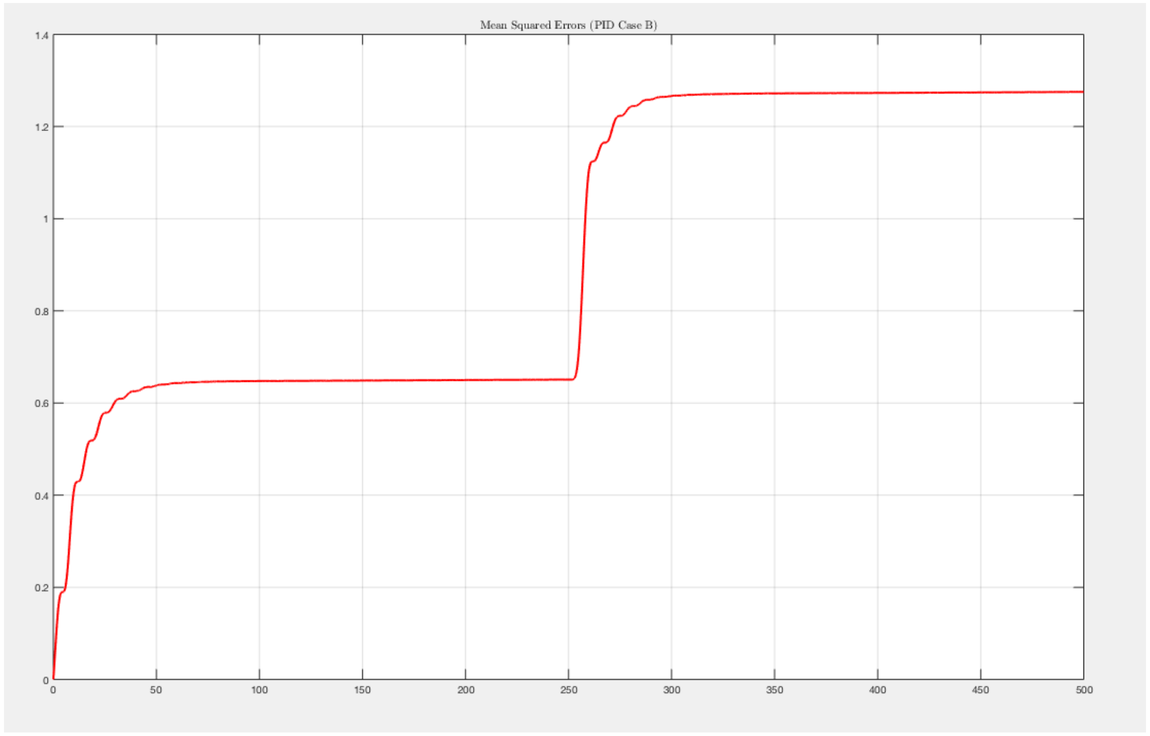
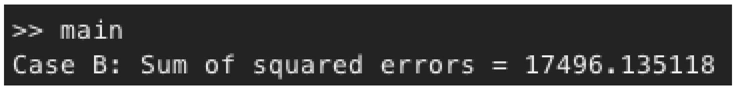


**Figure 10.** Mean squared error for Case B.



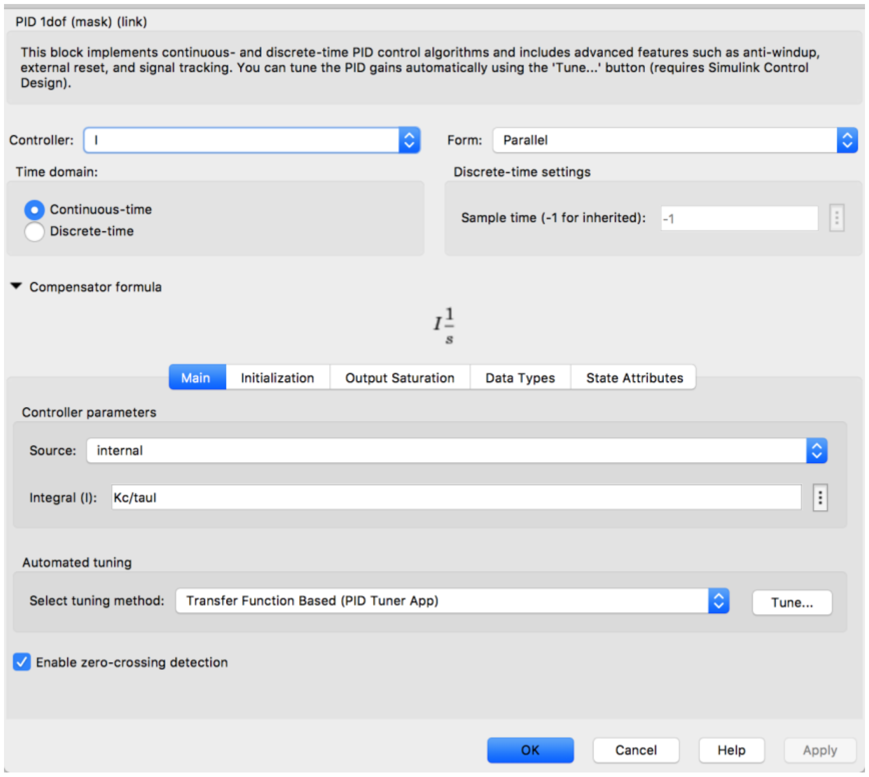
```
>> main
Case B: Sum of squared errors = 17496.135118
```

**Figure 11.** Sum of squared error for Case B (see appendix **A.1.1**).

The output responses for both cases A, and B are similar to each other, where both cases have 2 sets of oscillations. The first oscillation better team the values of 12 and 0, and oscillates to a stabilized value of 6, while the second oscillation occurs at the values of 8 and 0. Finally, as time increases the value reaches the amplitude value of the stop signal. Case A has PID components acting on the error signal, while Case B has PI on the error signals, and D acts on the output signal. The team, the output responses are quite similar to each other. The significant un-similarity between responses is the control signal. Case A's control signal shows that the standard/ideal PID controller is not quite reliable in practical implication as the control signal initially shows amplification of the measurement noise which is produced from the difference of the output signal. Although the Mean squared error curve of B has larger values than A, it shows the teams that the amplification of the measurement noise has been significantly eliminated by assigning a D controller on the output feedback signal.

## Case C:

Integral (I) controller on the error signals and PD controller on the output.



PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: I

Form: Parallel

Time domain:

● Continuous-time
○ Discrete-time

Discrete-time settings

Sample time (-1 for inherited): -1

Compensator formula

$I\frac{1}{s}$

Controller parameters

Source: internal

Integral (I): Kc/taul

Automated tuning

Select tuning method: Transfer Function Based (PID Tuner App)

☑ Enable zero-crossing detection

**Figure 12.** Configuration for I controller.

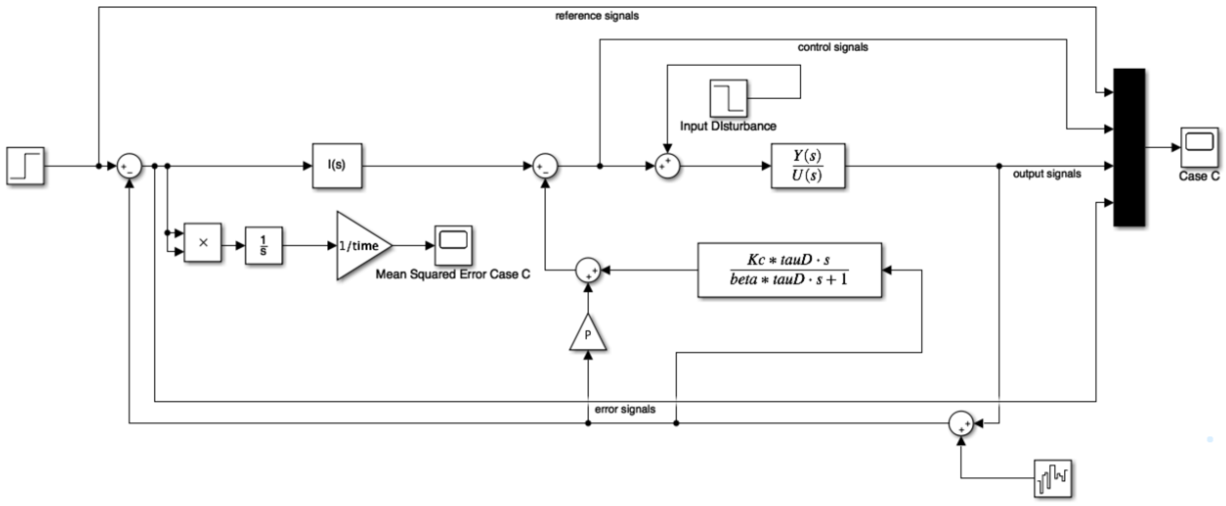


**Figure 13.** I controller on error signals and PD controllers on the output signal (Simulink diagram).

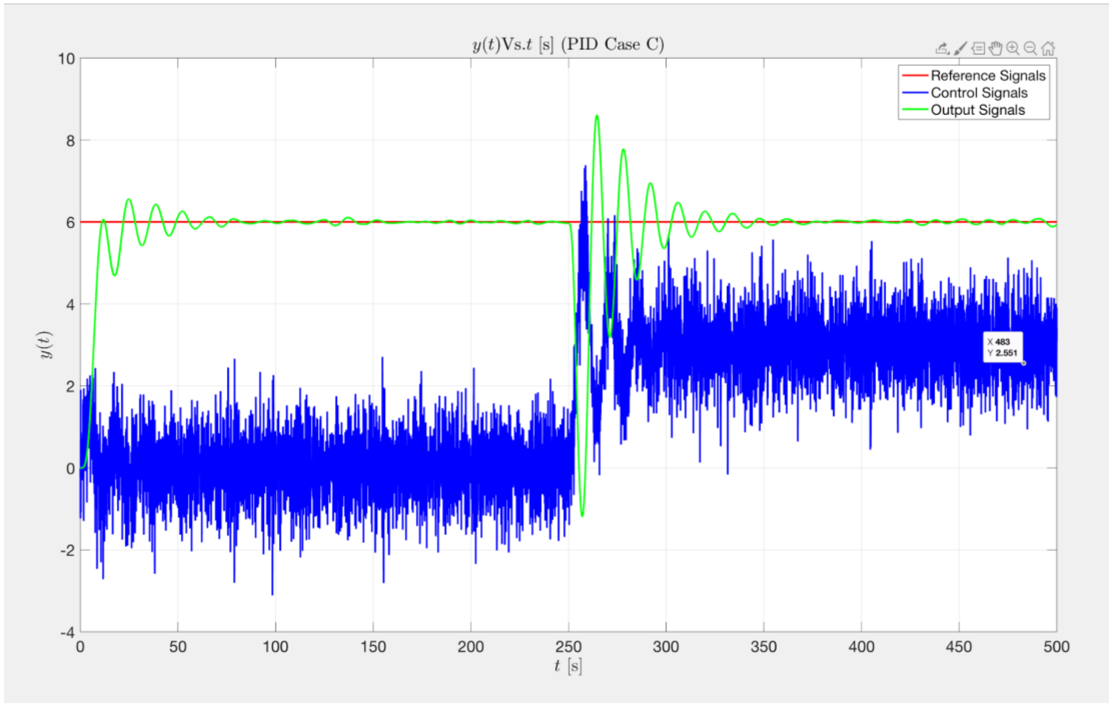


**Figure 14.** Simulink output for reference, control, and output signals for Case C.

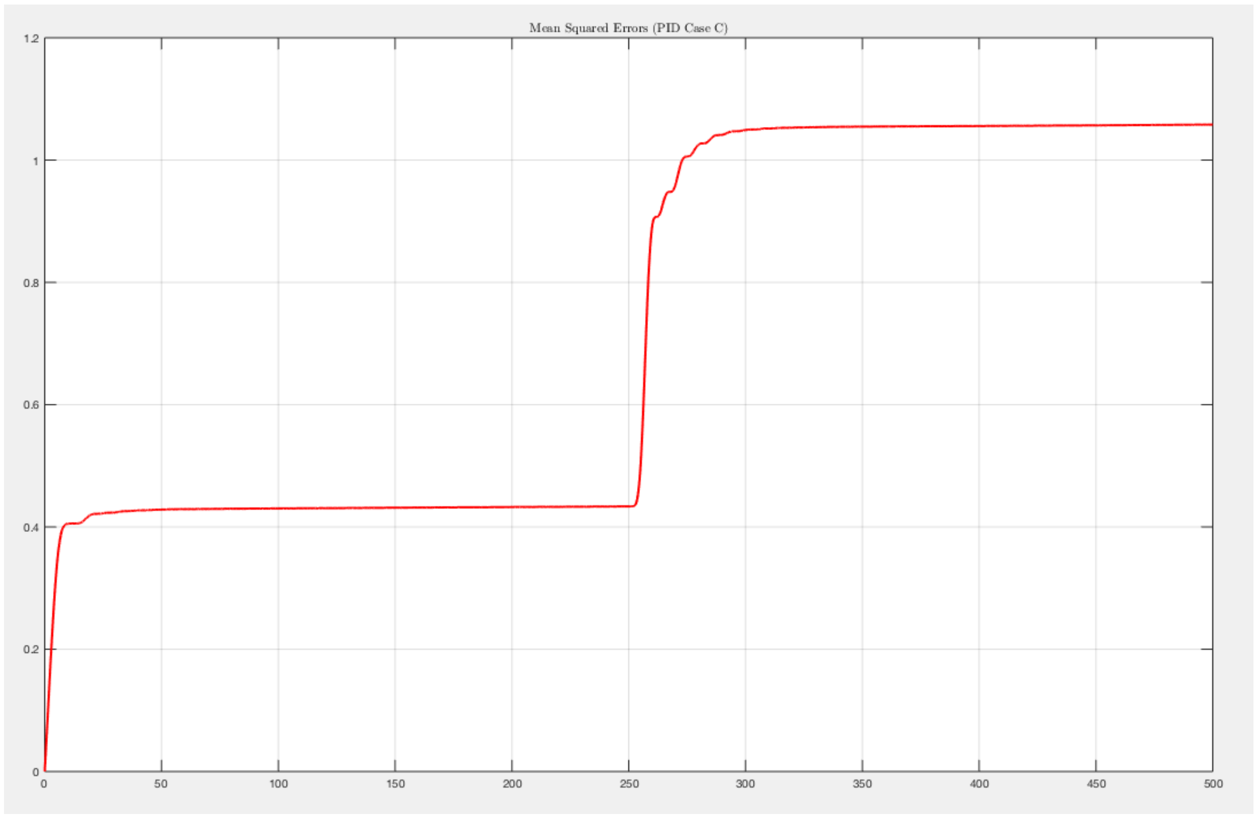
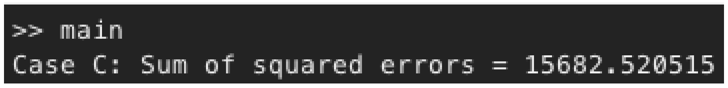


**Figure 15.** Mean squared error for Case C.



```
>> main
Case C: Sum of squared errors = 15682.520515
```

**Figure 16.** Sum of squared error for Case C (see appendix **A.1.1**).

In this case, the I controller is assigned to error signals while applying the PD controller to the output signal. The first oscillation of the response starts at a value of 0 and oscillates to a peak of 7. The first oscillation stabilizes at a value of step signal amplitude of 6. The second oscillation oscillates between the value of -1 and 9 and stabilizes at 6. The difference in the response of case C with case A and B are that the first set of oscillation does not increase to a peak value of 12, for case C the oscillation is rather premature and has a less aggressive response at the start and doesn't reach such a high peak value as the case A and B.

Mean squared error is defined as an estimator average between the estimated value and the actual value. The effect of the sum of squared errors can be observed in the following figures (Fig. 5, Fig. 10, & Fig. 15). The system with the lower mean square error is the best design. Due to the mean squared curves, the best option is case C, as the mean squared curve is far smaller than the other mean squared curves. The smaller mean squared error can be explained from the output signal since it has minimal overshoot compared to other PID implementations. In the end, utilizing the IPD control system leads to the most stable response with minimum overshoot, although the response speed is slower than the other response.

## Section 2.0

In this section, the first order delay plus transfer function for the desired closed-loop PID controller is defined as shown

$$G(s) = \frac{K_c e^{-ds}}{\tau_p s + 1} = \frac{10e^{-5s}}{10s + 1}$$

Where the implementation of the PID controller is designed as shown below.

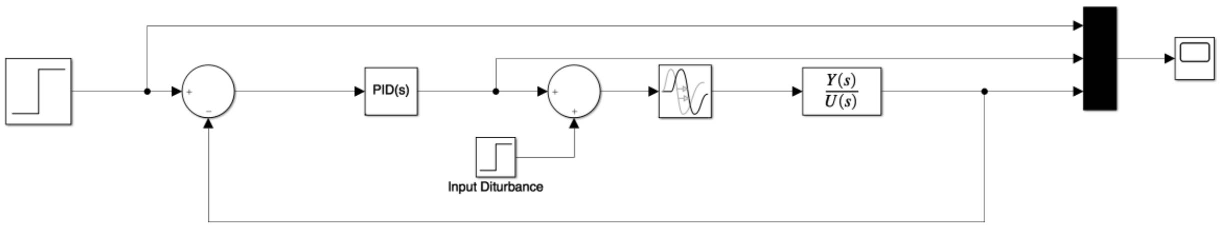


**Figure 17.** Standard PID controller applied for all cases.

Below are the conditions to be applied in the Simulink simulation:

- Considering the extra closed-loop pole at -1 ($\lambda_1$ = 1).
- There is a 5-second delay.
- The reference input signal is a step signal with an amplitude of 3.
- Choosing the sample time of $\Delta t = 0.1s$ and 500s simulation time
- Finally, an input disturbance with an amplitude of 1 at half of the simulation time.

On the other hand, it is known that the parameter for the PID would require the value of $K_c$, $\tau_I$, $\tau_D$, and $\tau_f$, where $\tau_f = \beta\tau_D$. Furthermore, just by setting up/adjusting the value $\omega_n$is enough to change the behavior of the PID controller.

By applying Pade's approximation,

$$G(s) = \frac{K_p e^{-ds}}{\tau_p s + 1} = \frac{K_p(-ds + 2)}{(\tau_p s + 1)(ds + 2)}$$

Then assumed the general form of the transfer function would be in the form of

$$G(s) = \frac{b_1 s + b_0}{(s + \alpha_1)(s + \alpha_2)}$$

Assuming the PID controller transfer function is

$$C(s) = \frac{c_s s^2 + c_1 s + c_0}{s(s + I_0)}$$

Where poles are located at,

$$s_1 = 0, and\ s_2 = -I_0$$

and assuming zeros are located

$$z_1 = -\gamma_1, and\ z_2 = -\alpha_2, as\ -\alpha_2\ is\ one\ of\ G(s)'s\ poles$$

However, choose

$$s_2 = -I_0$$

Now, consider the open loop of PID controller would be

$$L(s) = G(s)C(s) = \frac{b_1 s + b_0}{(s + \alpha_1)(s + \alpha_2)} \frac{c_2(s + \gamma_1)(s + \alpha_2)}{s(s + I_0)}$$

$$\Rightarrow \frac{c_2(b_1 s + b_0)(s + \gamma_1)}{s(s + I_0)(s + \alpha_1)}$$

Then consider the closed-loop transfer function of

$$T(s) = \frac{L(s)}{1 + L(s)} = \frac{c_2(b_1 s + b_0)(s + \gamma_1)}{s(s + I_0)(s + \alpha_1) + c_2(b_1 s + b_0)(s + \gamma_1)}$$

Therefore, it is expected that the desired closed-loop polynomial would be a third-order polynomial.

$$A_{cl}(s) = (s^2 + 2\xi\omega_n s + \omega_n^2)(s + \lambda_1)$$

Then, equate the denominator of $T(s)$ with $A_{cl}(s)$.

$$(s^2 + 2\xi\omega_n s + \omega_n^2)(s + \lambda_1) = s(s + I_0)(s + \alpha_1) + c_2(b_1 s + b_0)(s + \gamma_1)$$

By coefficient:

$$s^2 : \alpha_1 + I_0 + c_2 b_1 = 2\xi\omega_n + \lambda_1$$

$$s : \alpha_1 I_0 + c_2 b_0 + c_2 b_1 \gamma_1 = \omega_n^2 + 2\lambda_1 \xi\omega_n$$

$$s^0 : c_2 b_0 \gamma_1 = \lambda_1 \omega_n^2$$

Then by using the fact that

$$c_2 \gamma_1 = \frac{\lambda_1 \omega_n^2}{b_0}$$

Therefore,

$$\Rightarrow I_0 = -c_2 b_1 + 2\xi\omega_n + \lambda_1 - \alpha_1$$

$$\Rightarrow c_2 = \frac{-2\xi\omega_n\alpha_1 - \lambda_1\alpha_1 + \alpha_1^2 + \omega_n^2 + 2\lambda_1\xi\omega_n - \left(\frac{b_1\lambda_1\omega_n^2}{b_0}\right)}{b_0 - \alpha_1 b_1}$$

$$\Rightarrow \gamma_1 = \frac{\lambda_1 \omega_n^2}{b_0 c_2}$$

Hence, by equating the controller's transfer function with its simplified form

$$C(s) = \frac{c_s s^2 + c_1 s + c_0}{s(s + I_0)} = \frac{c_2(s + \gamma_1)(s + \alpha_2)}{s(s + I_0)}$$

$$\Rightarrow c_1 = c_2(\gamma_1 + \alpha_2), \& \ c_0 = c_2 \alpha_2 \gamma_1$$

From the derivations above, it is can be seen that $I_0, c_2, \& \ \gamma_1$are depends on $\omega_n$, hence leading $c_1, \& \ c_0$to be dependent on $\omega_n$. Since the fact that

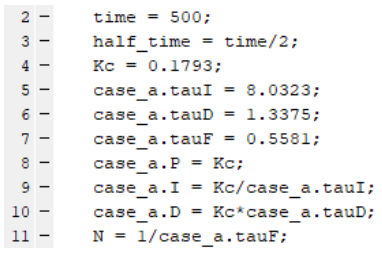
$$\tau_f = \frac{1}{I_0}$$

$$\tau_I = \frac{c_1}{c_0} - \tau_f$$

$$K_c = \tau_I \tau_f c_0$$
$$\tau_D = \frac{c_2 \tau_I \tau_f - K_c \tau_I \tau_f}{K_c \tau_I}$$
It proves that just adjusting the value of $\omega_n$ is sufficient enough to generate all parameters required to build the specifications/change behavior of the PID controller for this section's experiment.

## Case A:

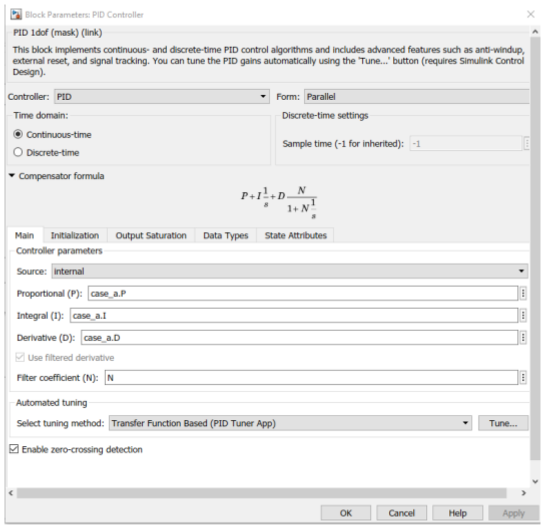
In this case, the team are given the closed-loop bandwidth and the damping coefficient;
$$\omega_n = 0.2\,rad/s, \xi = 0.707$$
Which with the help of pole-zero cancellation, the team get the controllers parameters as ;
$$K_c = 0.1793, \tau_I = 8.0323, \tau_D = 1.3375, \tau_f = 0.5581$$
Here for the simulation, the team has created a script to run the following parameters (Fig. 18).



```
time = 500;
half_time = time/2;
Kc = 0.1793;
case_a.tauI = 8.0323;
case_a.tauD = 1.3375;
case_a.tauF = 0.5581;
case_a.P = Kc;
case_a.I = Kc/case_a.tauI;
case_a.D = Kc*case_a.tauD;
N = 1/case_a.tauF;
```

**Figure 18.** Parameters for Case A.

As shown in Fig 17. the team has used a PID controller block for the simulation and the block is defined with the following parameters as shown in Fig 19.



Block Parameters: PID Controller

PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID    Form: Parallel

Time domain: [✓ Continuous-time] Discrete-time

Discrete-time settings: Sample time (-1 for inherited): -1

Compensator formula

$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}$$

Controller parameters

Source: internal

Proportional (P): case_a.P

Integral (I): case_a.I

Derivative (D): case_a.D

☑ Use filtered derivative

Filter coefficient (N): N

Automated tuning

Select tuning method: Transfer Function Based (PID Tuner App)

☑ Enable zero-crossing detection

**Figure 19.** PID configuration for Case A.

After applying the parameters to the function blocks and running the simulation, the team got the results from the scope as shown in Fig 20.

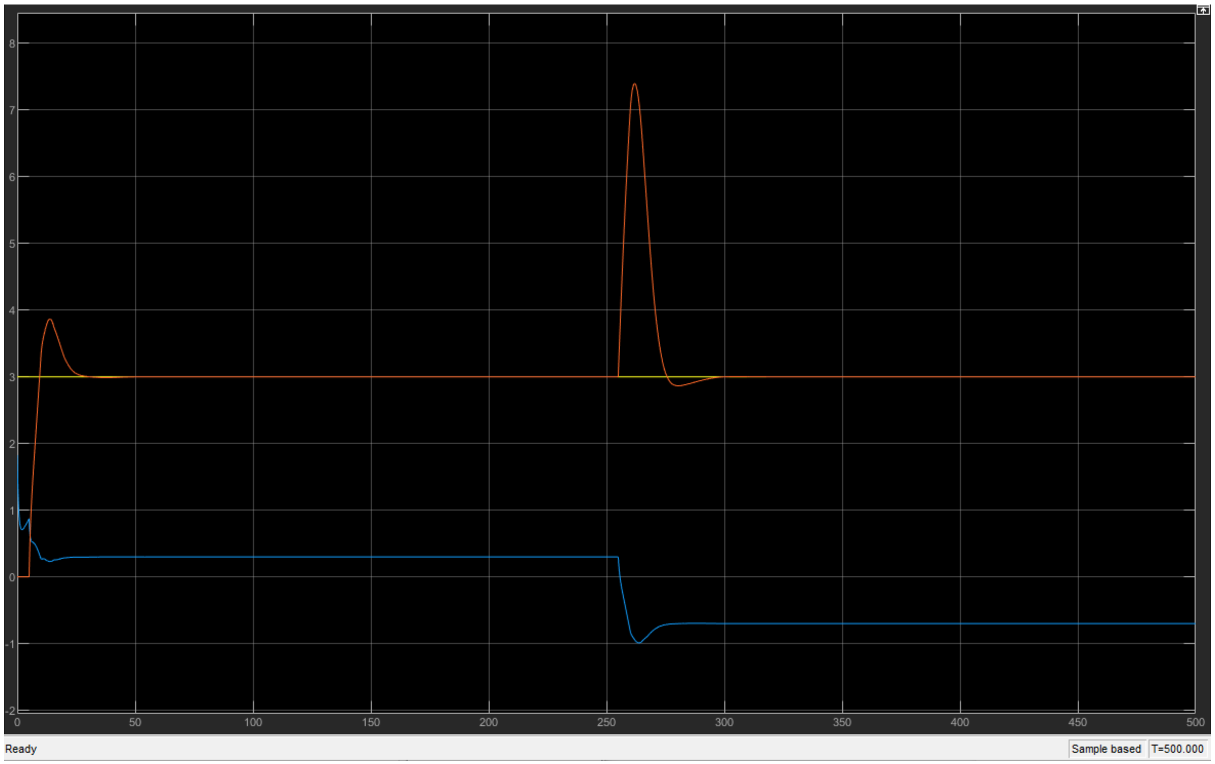


**Figure 20.** PID signals for Case A (— output signals, — control signals, — reference signals).
As seen in Fig 20. the team got to see the output signal in the red line.

## Case B:

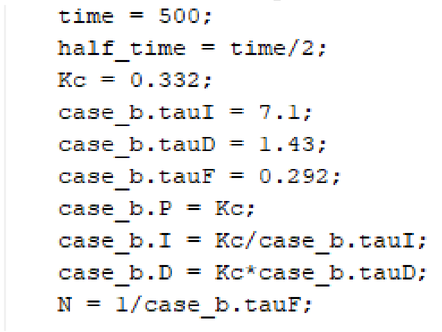
In this case, the team are given the closed-loop bandwidth and the damping coefficient of

$$\omega_n = 0.4\, rad/s, \xi = 0.707$$

With the help of pole-zero cancellation, the team gets the controller's parameters as shown below.

$$K_c = 0.332, \tau_I = 7.1, \tau_D = 1.43, \tau_f = 0.292$$

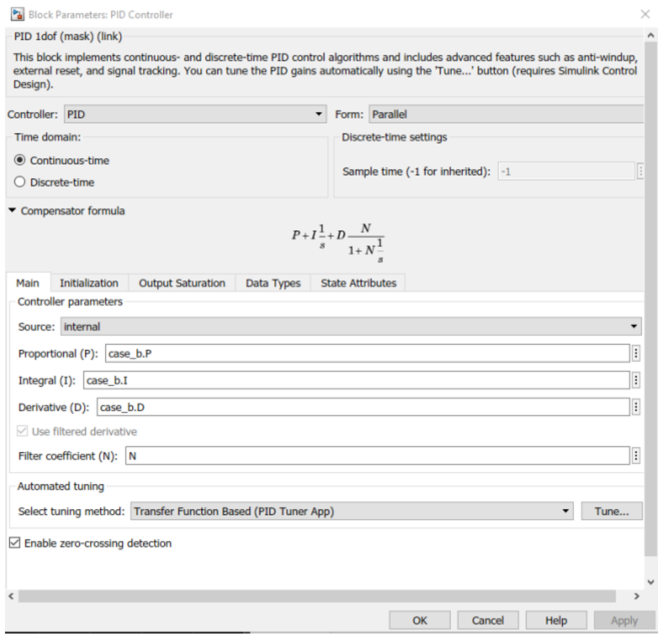
Here for the simulation, the team have created a script to run the following parameters (Fig. 21)



```
time = 500;
half_time = time/2;
Kc = 0.332;
case_b.tauI = 7.1;
case_b.tauD = 1.43;
case_b.tauF = 0.292;
case_b.P = Kc;
case_b.I = Kc/case_b.tauI;
case_b.D = Kc*case_b.tauD;
N = 1/case_b.tauF;
```

**Figure 21.** Parameters for Case B.

As shown in Fig 17. the team has used a PID controller block for the simulation and the block is defined with the following parameters as shown in Fig 22.



Block Parameters: PID Controller

PID 1dof (mask) (link)

This block implements continuous- and discrete-time PID control algorithms and includes advanced features such as anti-windup, external reset, and signal tracking. You can tune the PID gains automatically using the 'Tune...' button (requires Simulink Control Design).

Controller: PID
Form: Parallel

Time domain:
(●) Continuous-time
( ) Discrete-time

Discrete-time settings
Sample time (-1 for inherited): -1

Compensator formula

$$P+I\frac{1}{s}+D\frac{N}{1+N\frac{1}{s}}$$

Main | Initialization | Output Saturation | Data Types | State Attributes

Controller parameters
Source: internal
Proportional (P): case_b.P
Integral (I): case_b.I
Derivative (D): case_b.D
☑ Use filtered derivative
Filter coefficient (N): N

Automated tuning
Select tuning method: Transfer Function Based (PID Tuner App) Tune...

☑ Enable zero-crossing detection

OK Cancel Help Apply

**Figure 22.** PID configuration for Case B.

After applying the parameters to the function blocks and running the simulation, the team got the results from the scope as shown in Fig 23.

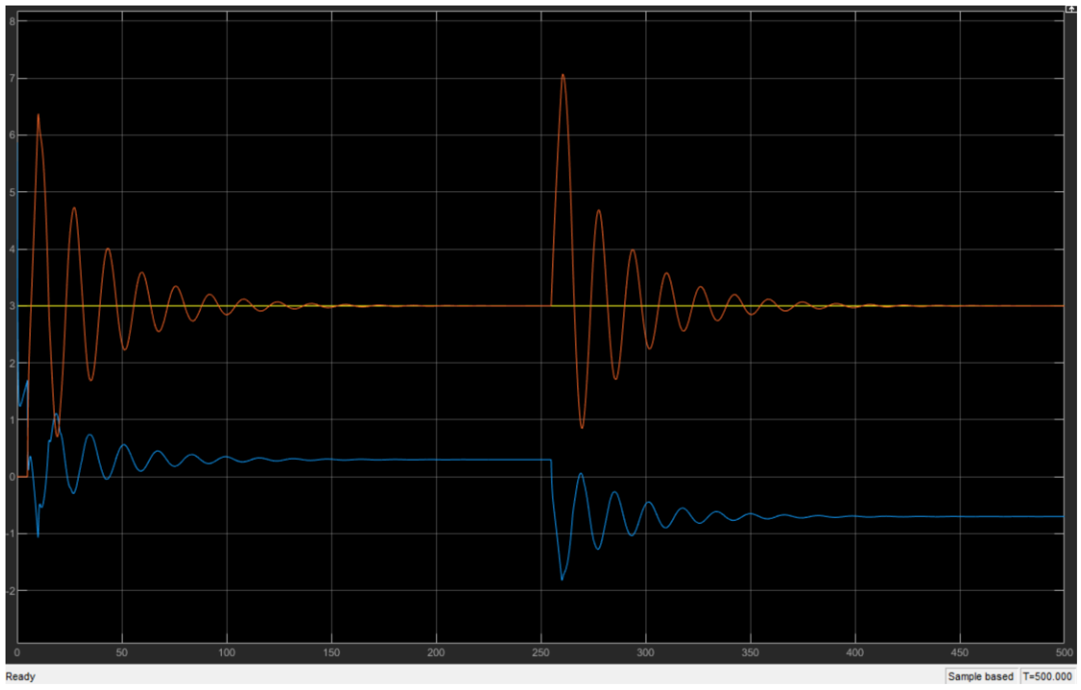


**Figure 23.** PID signals for Case B (— output signals, — control signals, — reference signals).

## Case C:

In this section let's consider the damping ratio of $\xi = 0.707$and the extra closed-loop poles at $\lambda_1 = 1$. According to by applying Pade's approximation to the given transfer function,

$$G(s) = \frac{10e^{-5s}}{10s + 1} \approx \frac{-s + 0.4}{(s + 0.1)(s + 0.4)}$$

Therefore, from the derived equation, the parameters required to calculate $\tau_f, \tau_I, K_c, \tau_D$are produced (See **Appendix A.2.1** for ModelPID class). Since,

$$\alpha_1 = 0.1 \,\&\, \alpha_2 = 0.4$$
$$b_1 = -1 \,\&\, b_2 = 0.4$$

Now, let's consider the value of natural frequency of 0.3 and 0.4 as shown below.

### Scenario 1: $\omega_n = 0.3\, rad/s$

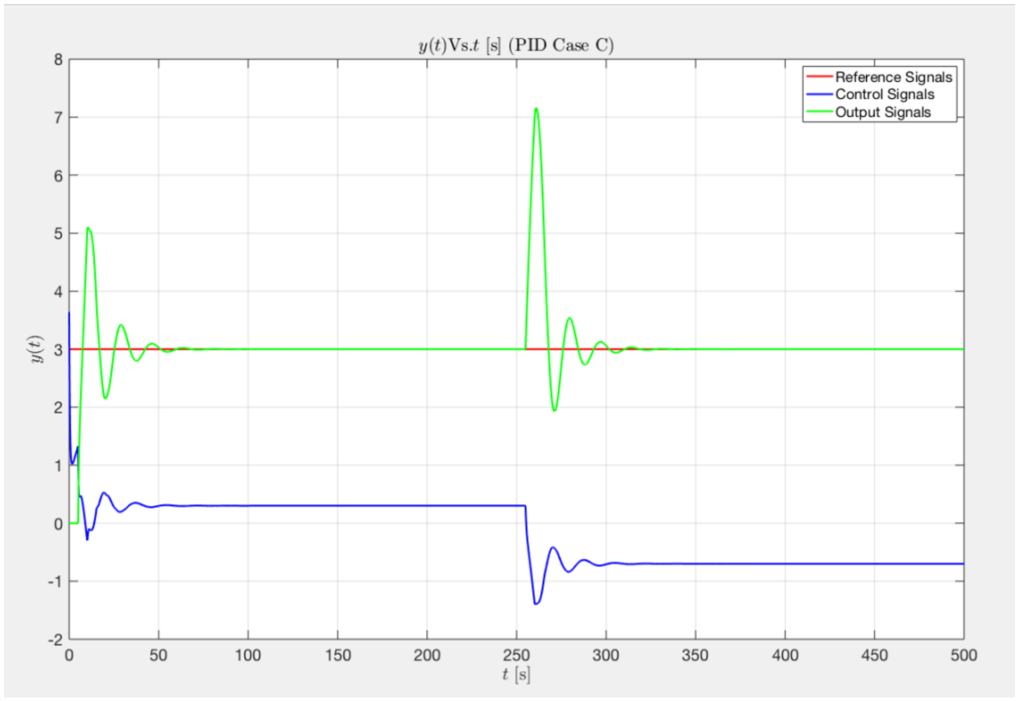


**Figure 24.** Signal plots of 0.3rad/s natural frequency.

Scenario 2: $\omega_n = 0.4\ rad/s$

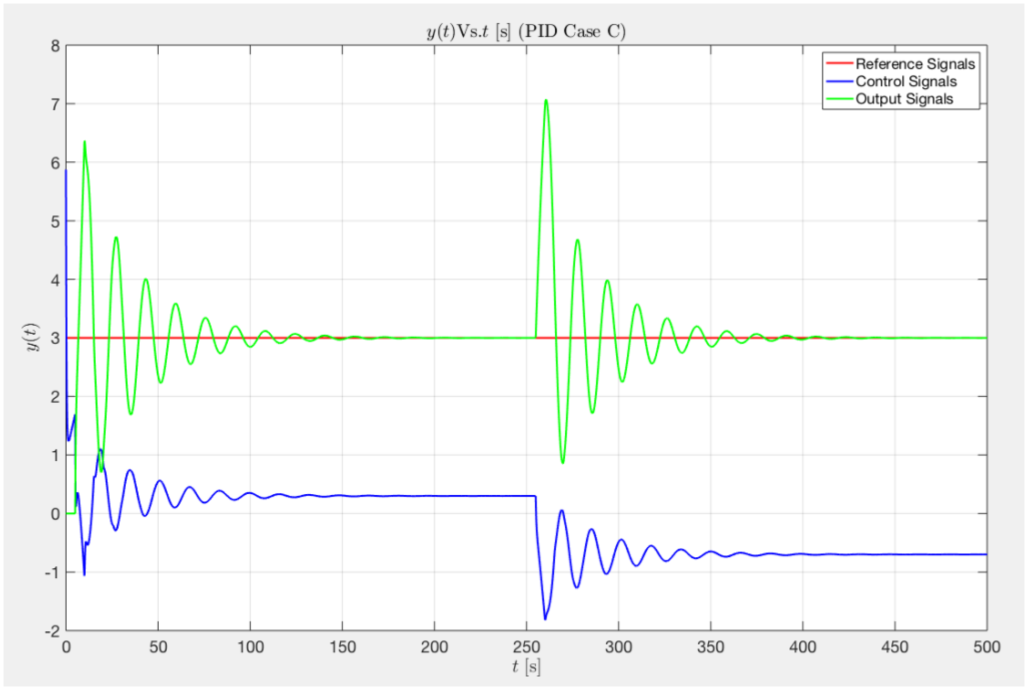
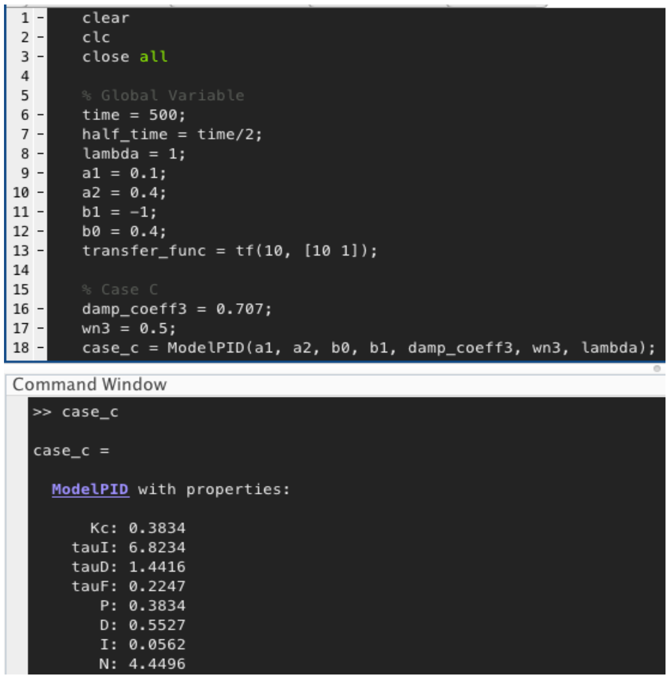


**Figure 25.** Signal plots of 0.4 rad/s natural frequency.

According to scenario 1 and scenario 2, it is observed that the amount of oscillation increased with the increase of natural frequency applied to the system. This phenomenon happens because the increase of natural frequency causes the closed-loop system to be far more sensitive to measurement noise and leads to faster response.

## Scenario 3: $\omega_n = 0.5\,rad/s$ (Sustained Oscillation)

Let's consider the value of natural frequency applied on the system as 0.5 rad/s. By utilizing the ModelPID MATLAB class that has been created for this section, it generates the required parameters to be embedded in the PID design as shown below (Reference **Appendix A.2.1** for ModelPID class that utilizes formulations/derivations above).



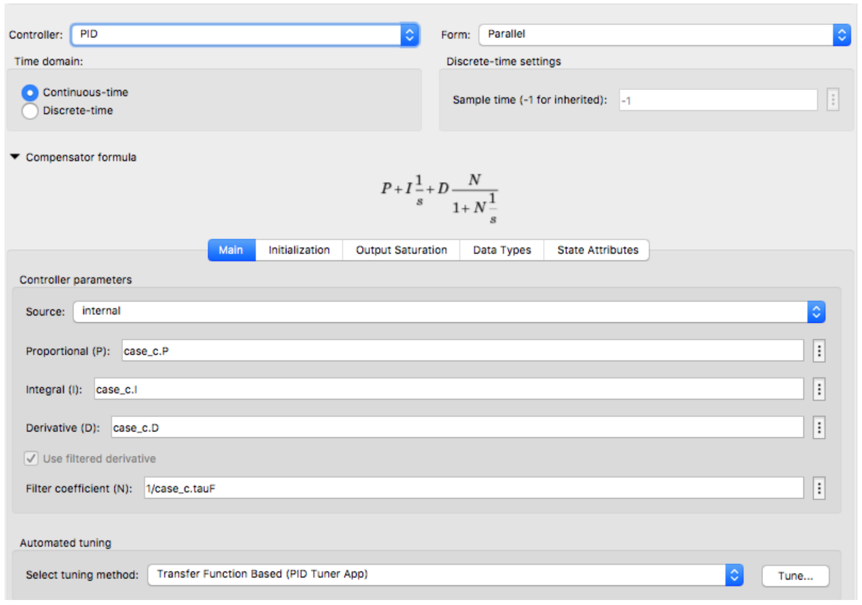
```
clear
clc
close all

% Global Variable
time = 500;
half_time = time/2;
lambda = 1;
a1 = 0.1;
a2 = 0.4;
b1 = -1;
b0 = 0.4;
transfer_func = tf(10, [10 1]);

% Case C
damp_coeff3 = 0.707;
wn3 = 0.5;
case_c = ModelPID(a1, a2, b0, b1, damp_coeff3, wn3, lambda);
```

Command Window

```
>> case_c

case_c =

  ModelPID with properties:

      Kc: 0.3834
    tauI: 6.8234
    tauD: 1.4416
    tauF: 0.2247
       P: 0.3834
       D: 0.5527
       I: 0.0562
       N: 4.4496
```

**Figure 26.** Parameters generated for sustained oscillation scenarios.

From here, utilize those parameters to the Simulink block diagram presented in **Figure 27.**



Controller: PID
Form: Parallel

Time domain:
Continuous-time [✓]
Discrete-time

Discrete-time settings
Sample time (-1 for inherited): -1

Compensator formula

$$P+I\frac{1}{s}+D\frac{N}{1+N\frac{1}{s}}$$

Controller parameters
Source: internal
Proportional (P): case_c.P
Integral (I): case_c.I
Derivative (D): case_c.D
✓ Use filtered derivative
Filter coefficient (N): 1/case_c.tauF

Automated tuning
Select tuning method: Transfer Function Based (PID Tuner App)

**Figure 27.** PID configuration for Case C.

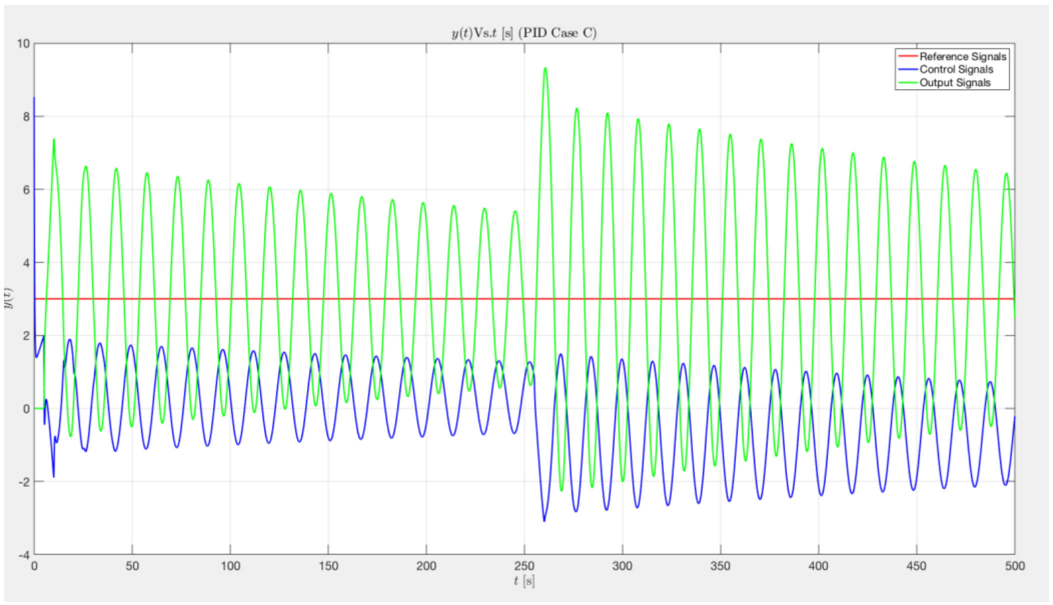
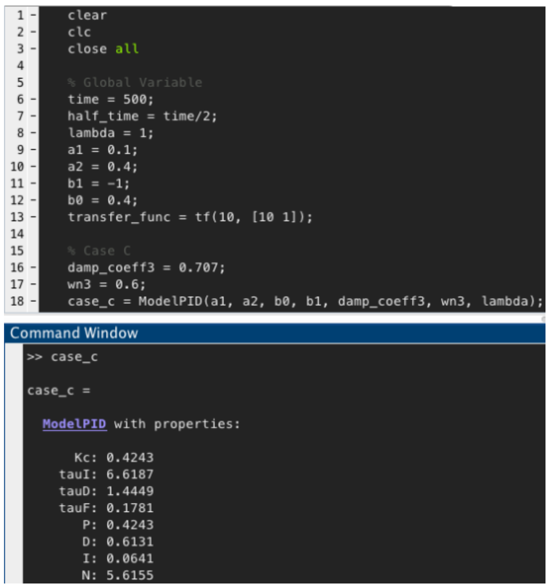


**Figure 28.** Sustained oscillation signal plots with 0.5rad/s natural frequency

Therefore, at the natural frequency of $\omega_n = 0.5\ rad/s$, the system produced a sustained oscillation output signal.

## Scenario 4: $\omega_n = 0.6\ rad/s$

In this scenario, the natural frequency is being increased beyond the sustained oscillation point, and it is expected that the signal will produce an unstable response. Now, let's consider the situation where the natural frequency of $\omega_n = 0.6\ rad/s$ being applied to the control system.



```
clear
clc
close all

% Global Variable
time = 500;
half_time = time/2;
lambda = 1;
a1 = 0.1;
a2 = 0.4;
b1 = -1;
b0 = 0.4;
transfer_func = tf(10, [10 1]);

% Case C
damp_coeff3 = 0.707;
wn3 = 0.6;
case_c = ModelPID(a1, a2, b0, b1, damp_coeff3, wn3, lambda);
```

Command Window

```
>> case_c

case_c =

  ModelPID with properties:

      Kc: 0.4243
    tauI: 6.6187
    tauD: 1.4449
    tauF: 0.1781
       P: 0.4243
       D: 0.6131
       I: 0.0641
       N: 5.6155
```

**Figure 29.** Parameters generated for an unstable system.

Hence, by inputting values of P, I, and D from case C, below is the response produced from the system.

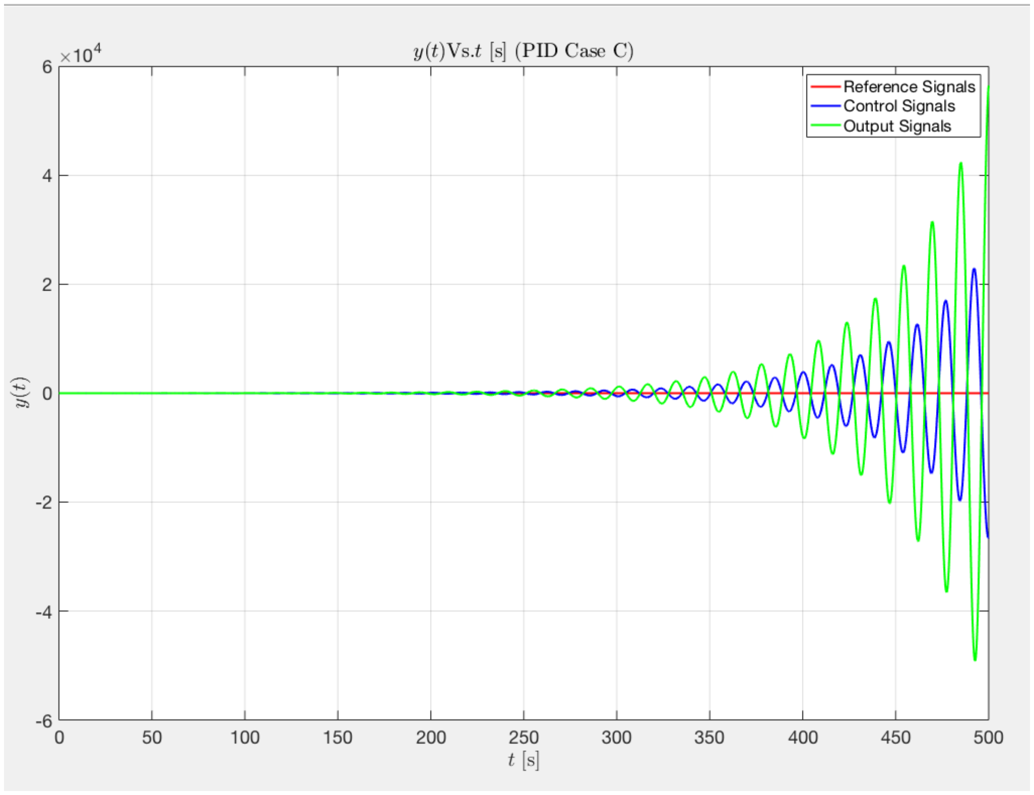


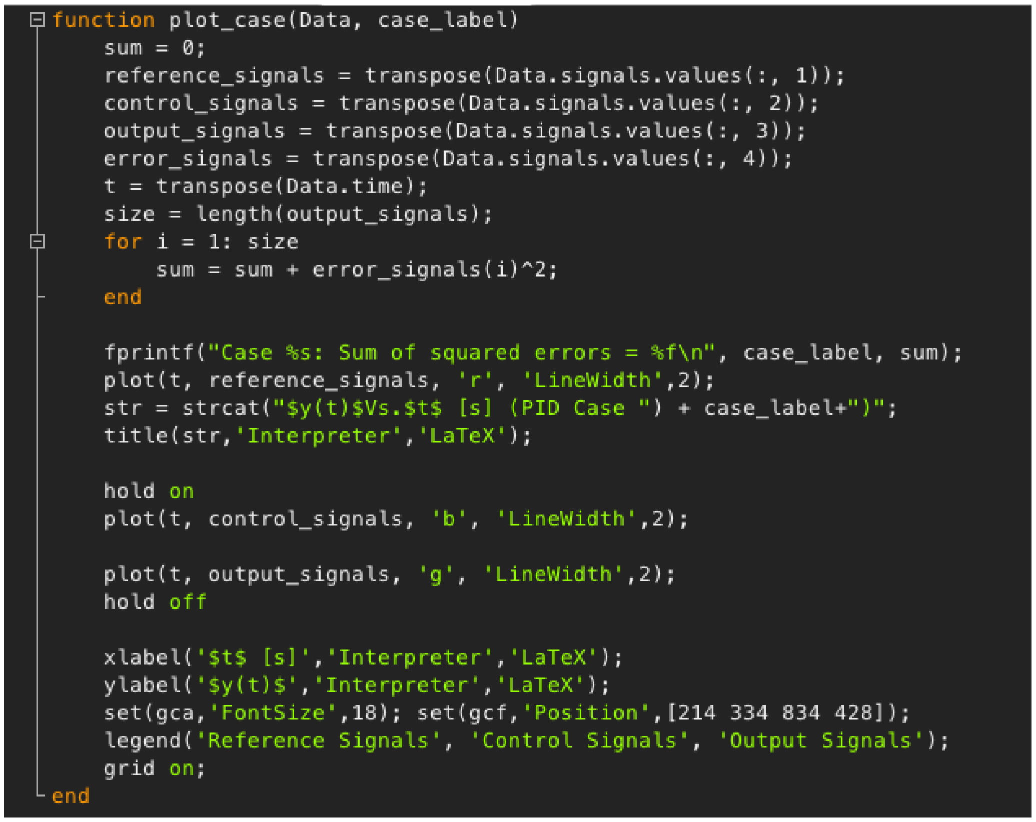
**Figure 30.** Unstable output response.

The response above is unstable as the oscillation will exponentially increase as $t \rightarrow \infty$. It is believed that the modeling of the time delay would affect the performance for the closed-loop system above since greater time delay leads the control system to be less sensitive, and have a slower response speed. In this scenario, it is possible that by the chosen natural frequency of 0.6 rad/s, the time delay might exceed the maximum allowable time delay generated from the chosen natural frequency.

# Appendix:

## Appendix A.1:

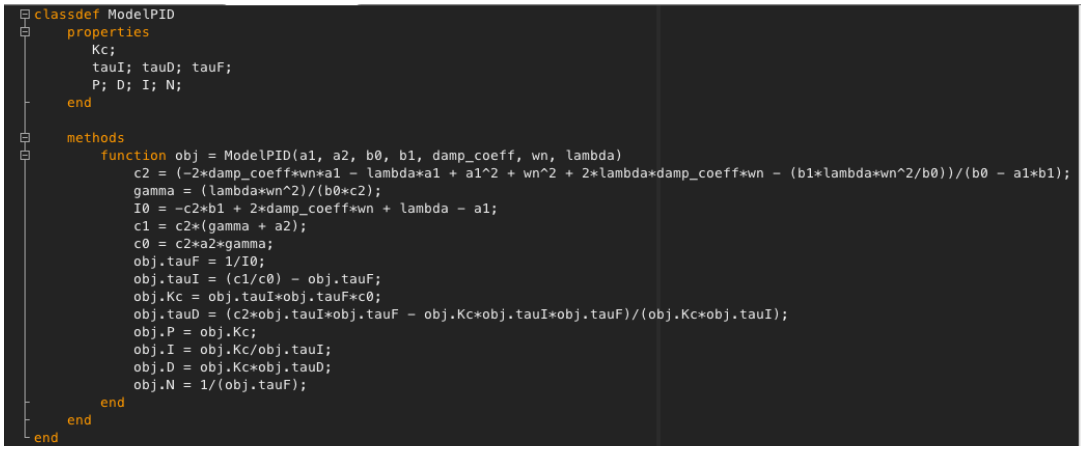
### Appendix A.1.1:



```
function plot_case(Data, case_label)
    sum = 0;
    reference_signals = transpose(Data.signals.values(:, 1));
    control_signals = transpose(Data.signals.values(:, 2));
    output_signals = transpose(Data.signals.values(:, 3));
    error_signals = transpose(Data.signals.values(:, 4));
    t = transpose(Data.time);
    size = length(output_signals);
    for i = 1: size
        sum = sum + error_signals(i)^2;
    end

    fprintf("Case %s: Sum of squared errors = %f\n", case_label, sum);
    plot(t, reference_signals, 'r', 'LineWidth',2);
    str = strcat("$y(t)$Vs.$t$ [s] (PID Case ") + case_label+")";
    title(str,'Interpreter','LaTeX');

    hold on
    plot(t, control_signals, 'b', 'LineWidth',2);

    plot(t, output_signals, 'g', 'LineWidth',2);
    hold off

    xlabel('$t$ [s]','Interpreter','LaTeX');
    ylabel('$y(t)$','Interpreter','LaTeX');
    set(gca,'FontSize',18); set(gcf,'Position',[214 334 834 428]);
    legend('Reference Signals', 'Control Signals', 'Output Signals');
    grid on;
end
```

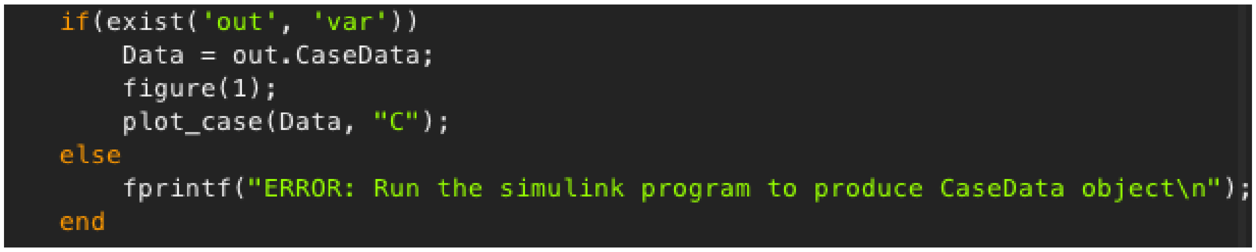
## Appendix A.2:

### Appendix A.2.1:



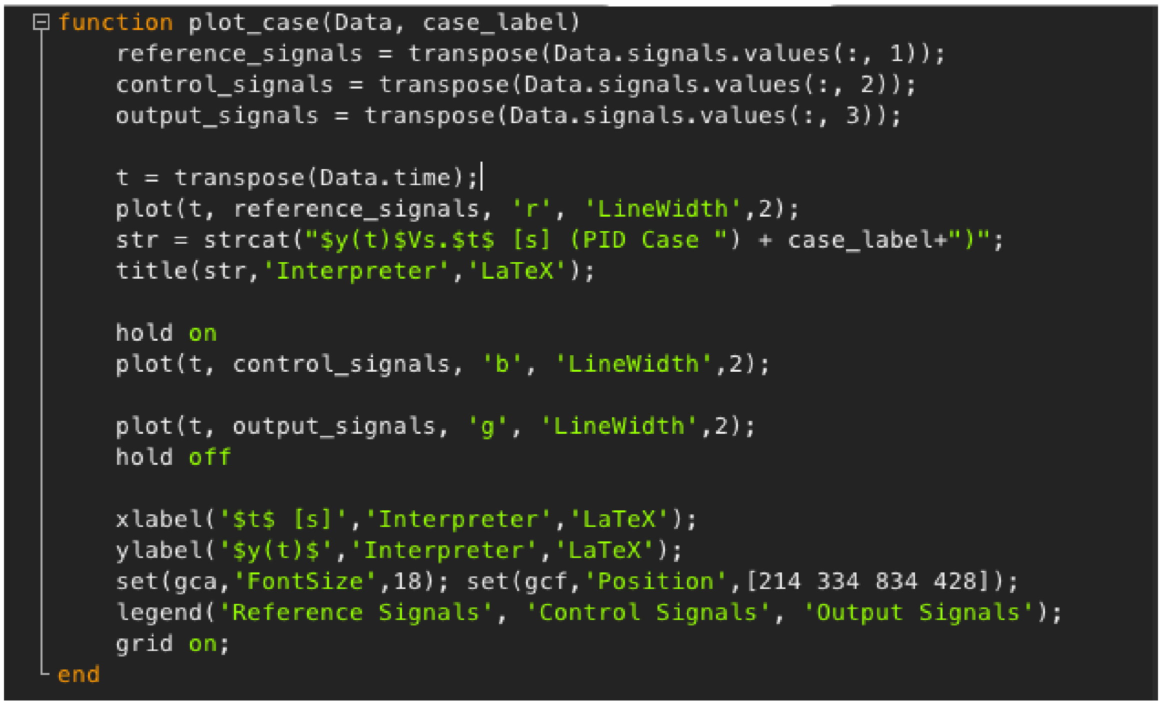
```
classdef ModelPID
    properties
        Kc;
        tauI; tauD; tauF;
        P; D; I; N;
    end

    methods
        function obj = ModelPID(a1, a2, b0, b1, damp_coeff, wn, lambda)
            c2 = (-2*damp_coeff*wn*a1 - lambda*a1 + a1^2 + wn^2 + 2*lambda*damp_coeff*wn - (b1*lambda*wn^2/b0))/(b0 - a1*b1);
            gamma = (lambda*wn^2)/(b0*c2);
            I0 = -c2*b1 + 2*damp_coeff*wn + lambda - a1;
            c1 = c2*(gamma + a2);
            c0 = c2*a2*gamma;
            obj.tauF = 1/I0;
            obj.tauI = (c1/c0) - obj.tauF;
            obj.Kc = obj.tauI*obj.tauF*c0;
            obj.tauD = (c2*obj.tauI*obj.tauF - obj.Kc*obj.tauI*obj.tauF)/(obj.Kc*obj.tauI);
            obj.P = obj.Kc;
            obj.I = obj.Kc/obj.tauI;
            obj.D = obj.Kc*obj.tauD;
            obj.N = 1/(obj.tauF);
        end
    end
end
```

Appendix A.2.2:



```
if(exist('out', 'var'))
    Data = out.CaseData;
    figure(1);
    plot_case(Data, "C");
else
    fprintf("ERROR: Run the simulink program to produce CaseData object\n");
end
```

Appendix A.2.3:



```
function plot_case(Data, case_label)
    reference_signals = transpose(Data.signals.values(:, 1));
    control_signals = transpose(Data.signals.values(:, 2));
    output_signals = transpose(Data.signals.values(:, 3));

    t = transpose(Data.time);
    plot(t, reference_signals, 'r', 'LineWidth',2);
    str = strcat("$y(t)$Vs.$t$ [s] (PID Case ") + case_label+")";
    title(str,'Interpreter','LaTeX');

    hold on
    plot(t, control_signals, 'b', 'LineWidth',2);

    plot(t, output_signals, 'g', 'LineWidth',2);
    hold off

    xlabel('$t$ [s]','Interpreter','LaTeX');
    ylabel('$y(t)$','Interpreter','LaTeX');
    set(gca,'FontSize',18); set(gcf,'Position',[214 334 834 428]);
    legend('Reference Signals', 'Control Signals', 'Output Signals');
    grid on;
end
```