BY GROUP J
Jul-Nov 2021

Process Control Lab Project: Group J

Group members:

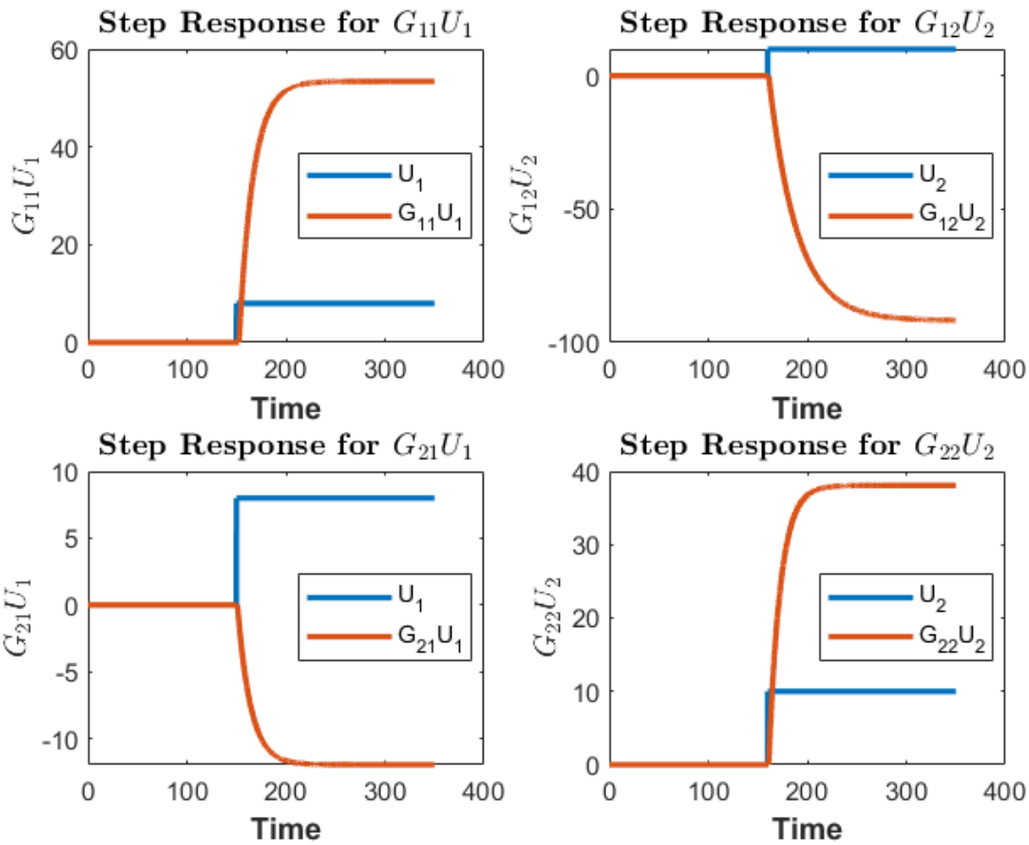
ROLL NUMBER	NAME
CH18B004	GAUD UNNAT
CH18B010	KINJARAPU SRIRAM
CH18B012	LINGAMOLLA VEDASRI SAI
CH18B020	S VISHAL
CH18B061	RISHIKESH S

Question-1: Distillation Column

Part a) Transfer Function Estimation

tfest function was used to estimate all the transfer functions.

```
G11 =

 From input "u1" to output "y1":
                  0.469
 exp(-2.5*s) * -----------
               s + 0.07025


G12 =
 From input "u2" to output "y1":
                 -0.3245
 exp(-1.1*s) * -----------
               s + 0.03536


G21 =

 From input "u1" to output "y2":
                 -0.1132
 exp(-1.1*s) * -----------
               s + 0.07582


G22 =

 From input "u2" to output "y2":
                  0.3288
 exp(-1.1*s) * -----------
               s + 0.08636
```



Part b) PI/PID Controller tuning

1: SISO with regards to top product composition

Tuning controllers using theoretical means:

$$G_{11}(s) = \frac{6.6757}{1.9360s + 1}e^{-2.5s}$$

=> $K = 6.675$
$\tau = 13.9360$
$\theta = 2.5$

PI Controller Tuning:

Theoretical

Using SIMC method:

$$K_c = \frac{\tau}{K(\tau_c + \theta)},\ \tau_i = \tau$$

Using Skogestad's guidelines for setting the control time constant (Skogestad, 2003):

$$\tau_c = \theta$$

=> $K_c = 0.4175$
$\tau_i = 13.9360$

Parallel form transfer function of the controller:

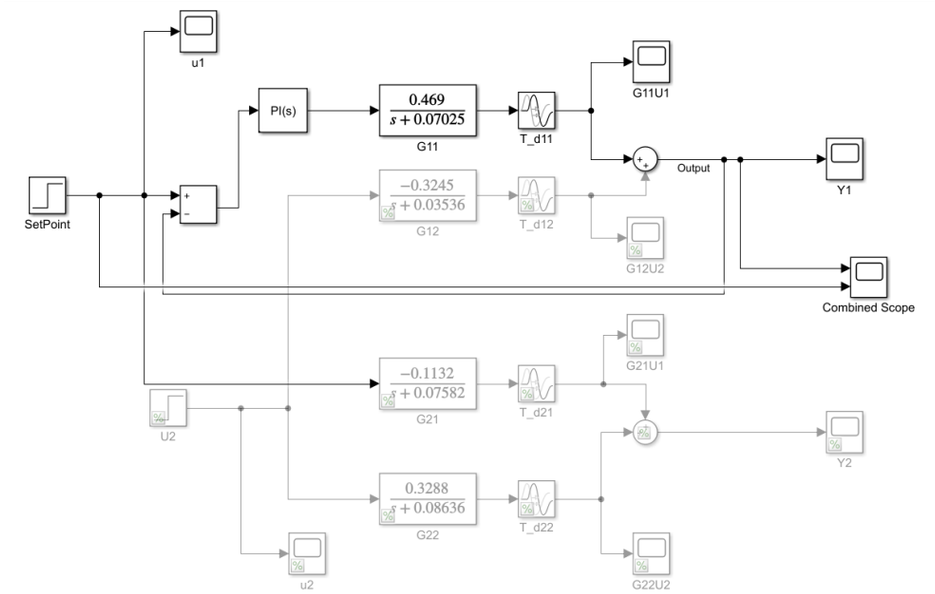
$$K_c(1+\frac{1}{\tau_I s}) \longleftrightarrow P+\frac{I}{s}$$

=>$P = 0.4175$

$$I = \frac{K_c}{\tau_i} = 0.03$$

So final controller: 0.4175 + 0.03/s

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: PI Form: Parallel

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

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

▼ Compensator formula

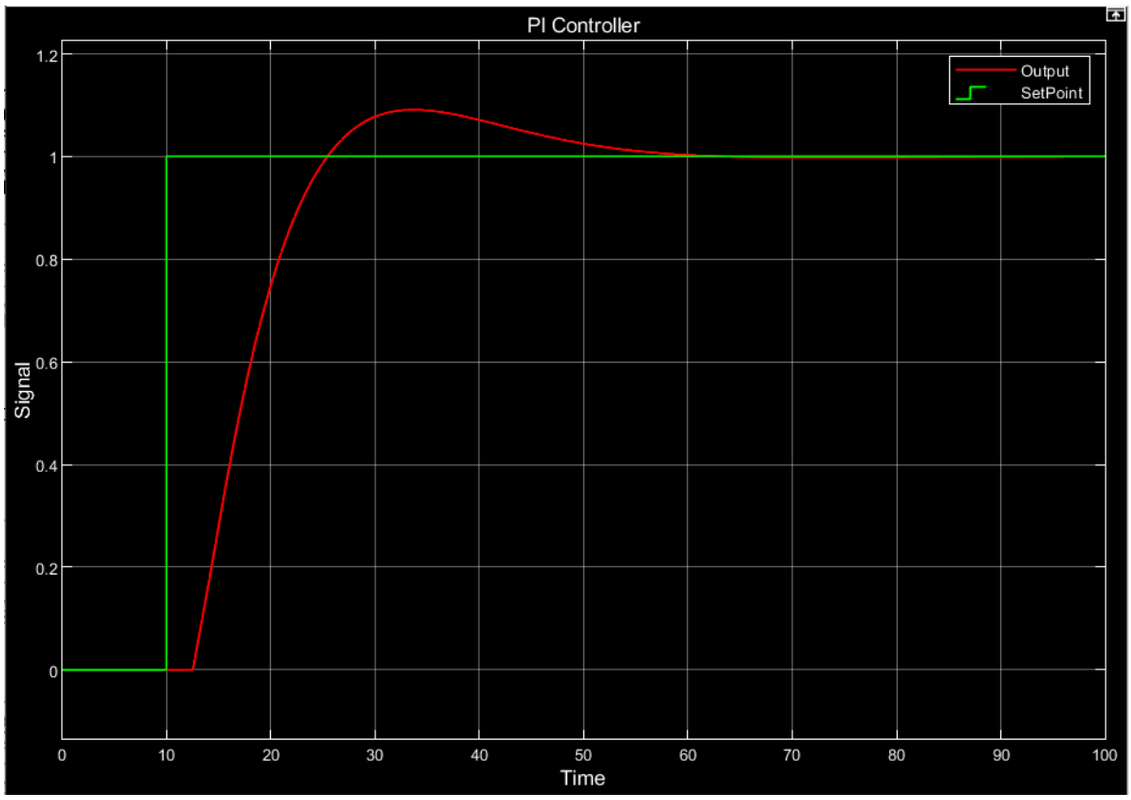
$$P+I\frac{1}{s}$$

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

Controller parameters
Source: internal
Proportional (P): 0.22952
Integral (I): 0.02531

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

☑ Enable zero-crossing detection



RED LINE: Y1 AND GREEN LINE: R1 (SETPOINT)

PID controller Tuning:

Theoretical:

Using SIMC method:

$$K_c = \frac{\tau + \theta/2}{K(\tau + \theta/2)}$$

Using Skogestad's guidelines for setting the control time constant (Skogestad, 2003):

$$\tau_c = \theta$$

=> $K_c = 0.1498$

$\tau_i = 15.1860,\ \tau_d = 1.1471,\ \tau_f = 2.5$

Parallel form transfer function of the controller:

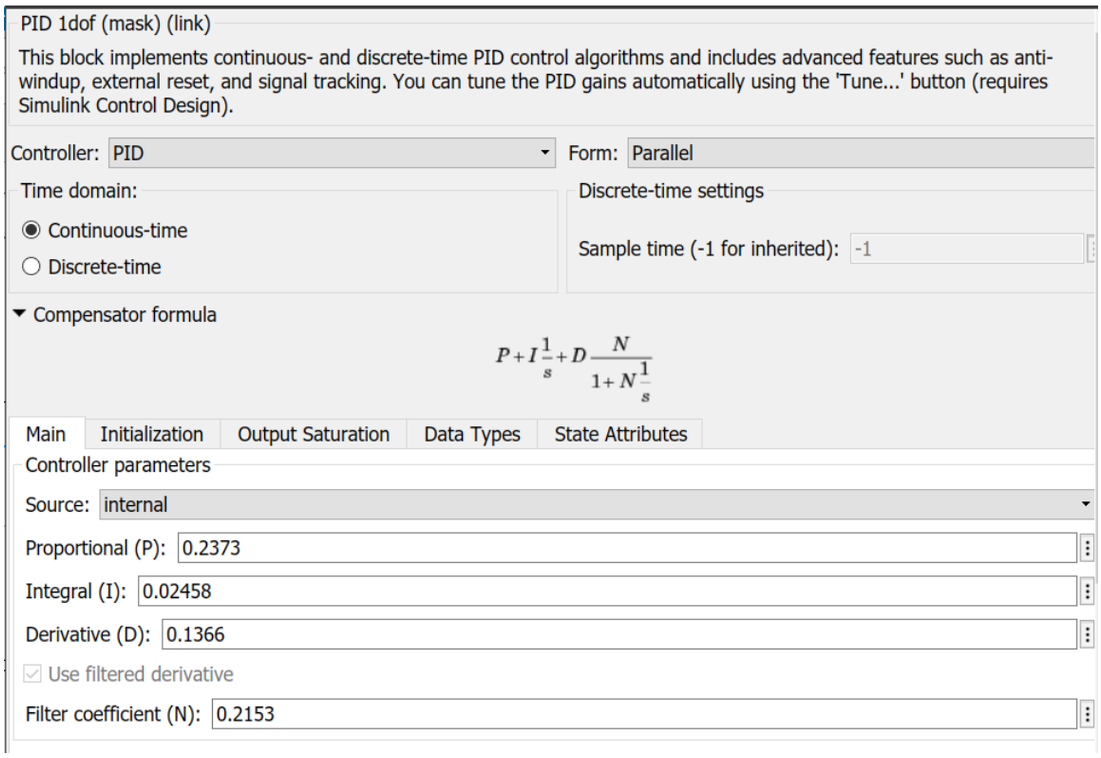
$$K_c(1 + \frac{1}{\tau_i s} + \tau_d s) \cdot f \longleftrightarrow (P + \frac{I}{s} + Ds) \cdot \frac{1}{1 + \tau_f s}$$

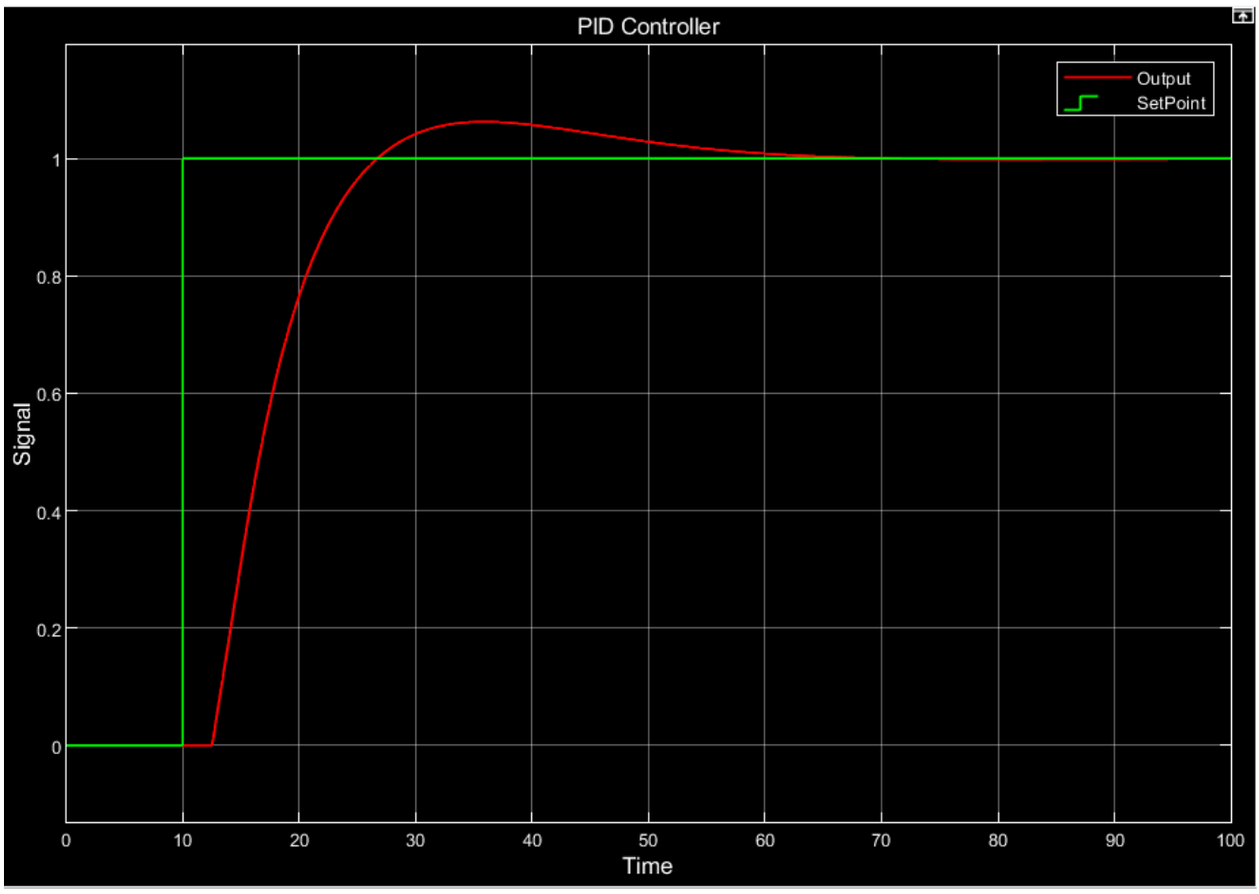
=> $P = 0.1498$

$I = \frac{K_c}{\tau_i} = 0.0099$

$D = K_c \tau_d = 0.1718$

$\tau_f = \tau c = 2.5$



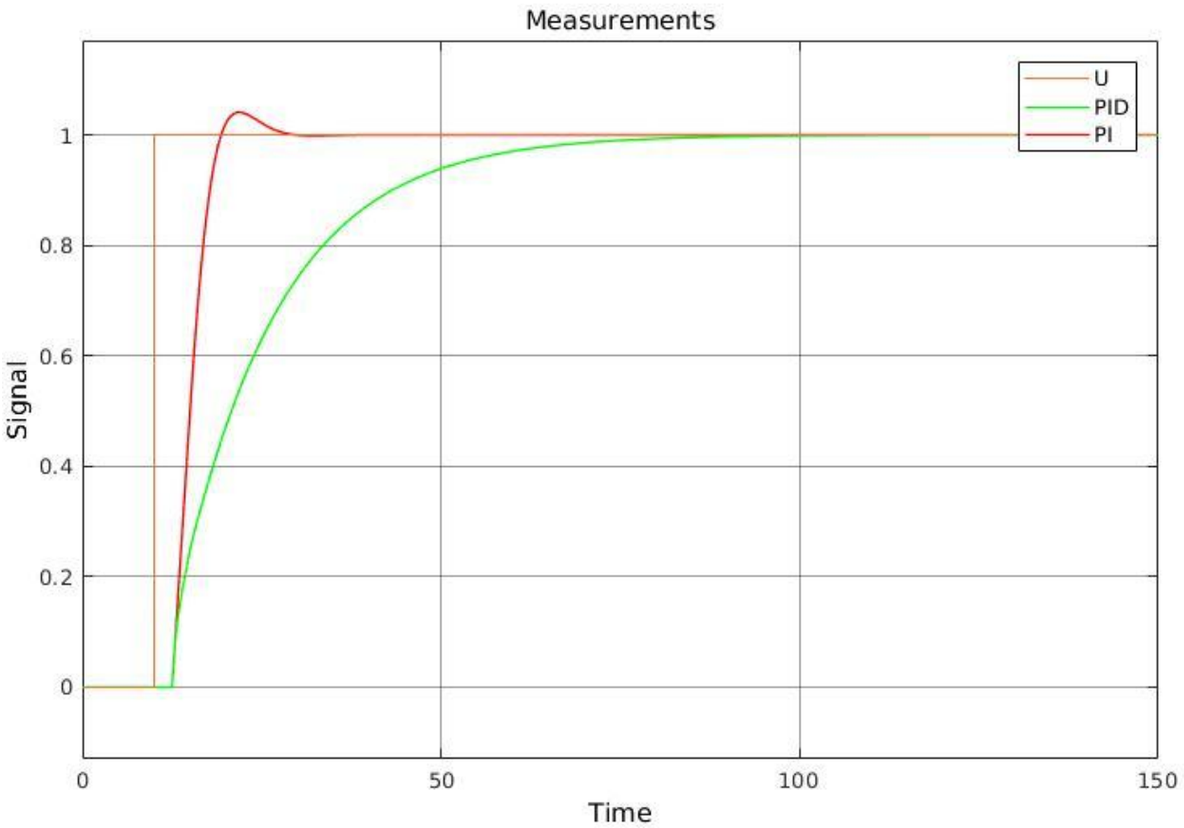


RED LINE: Y1 AND GREEN LINE: R1 (SETPOINT)

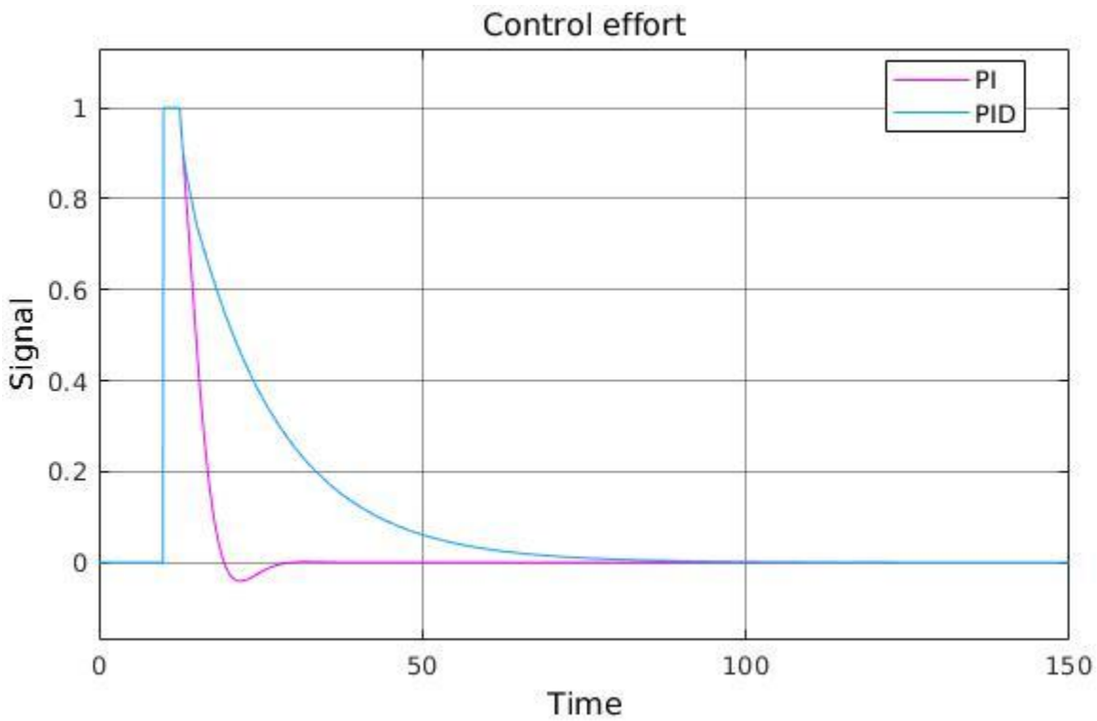
We cannot import these parameters directly into the PID controller block in SIMULINK, since the PID controller of SIMULINK uses a different controller structure with regard to the filter. In the case of IMC, the filter is applied on all three controller parts, namely Proportional (P), Integral (I) and Derivative (D), whereas in SIMULINK's PID block the filter component is applied only to the Derivative part. Therefore, a separate transfer function block was used for construction of the controller. The transfer function of the resultant PID controller is as follows:

$$G_c(s) = f(s)K_c(1+\frac{1}{\tau_i s}+\tau_d s) = \frac{Ds^2+Ps+I}{\tau_f s^2+s} = \frac{0.1718s^2+0.1498s+0.0099}{2.5s^2+s}$$

The following are the plots for the measurements and control effort obtained by using the two controllers:



The input signal is steps up to 1 at 10s. From the above plot, we can observe that there is a delay of 2.5s between the change in setpoint and the first change in the output.
The PI controller is able to stabilise the output signal in around 30s, whereas the PID controller takes around 80s. We can also observe that the PI controller slightly overshoots the required steady state value, while still settling relatively quickly. In the case of the PID controller, there is no offshoot observed, although it takes a longer time to reach the steady state value.



Here, we can observe that once the control effort kicks in (after a delay of 2.5s), the PI controller exerts a negative control effort for a short duration. The PID controller exerts a higher, longer sustaining effort to reach the setpoint than

the PI controller. From this, we can infer that the PI controller exerts less control effort when compared to the PID controller.

2: SISO with regards to bottom temperature

$$G_{22}(s) = \frac{3.808}{11.86s + 1}e^{-1.1s}$$

=> $K = 3.8084$
$\tau = 11.8590$
$\theta = 1.1$

PI Controller Tuning:

Using SIMC method:

$$K_c = \frac{\tau}{K(\tau_c + \theta)},\ \tau_i = \tau$$

Using Skogestad's guidelines for setting the control time constant (Skogestad, 2003):

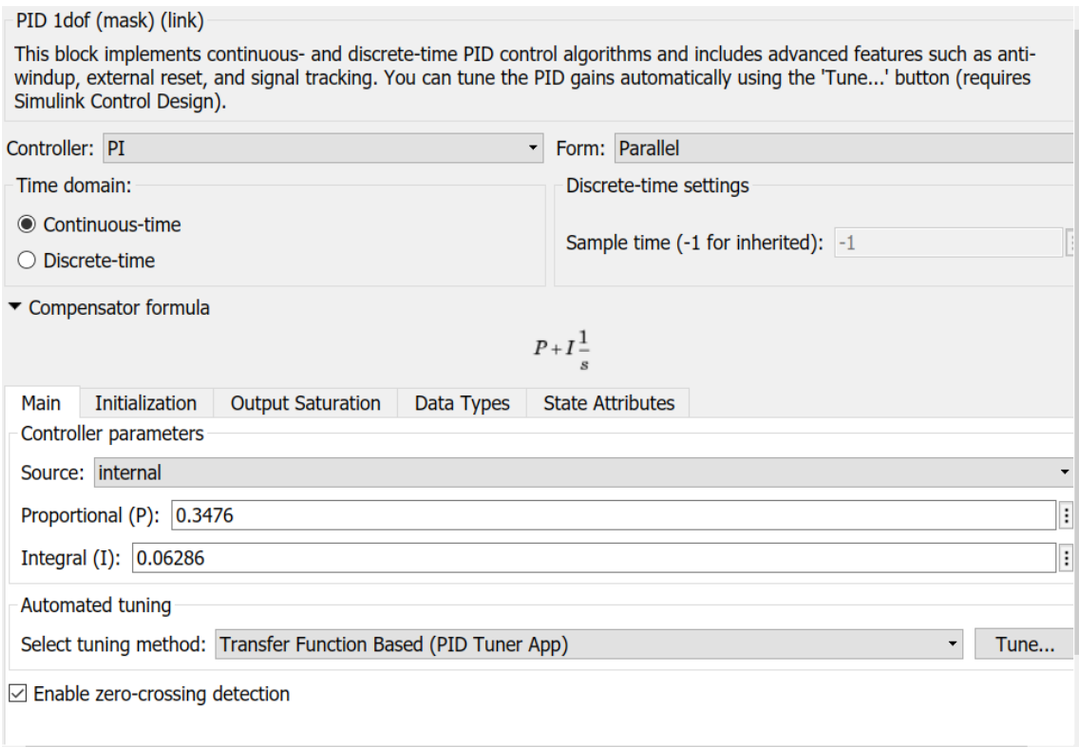
$$\tau_c = \theta$$

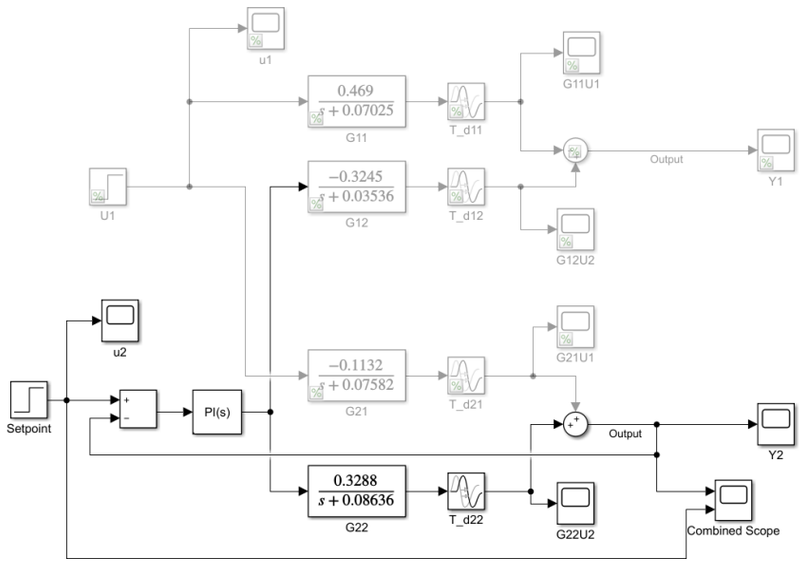
=> $K_c = 1.4154$
$\tau_i = 11.8590$

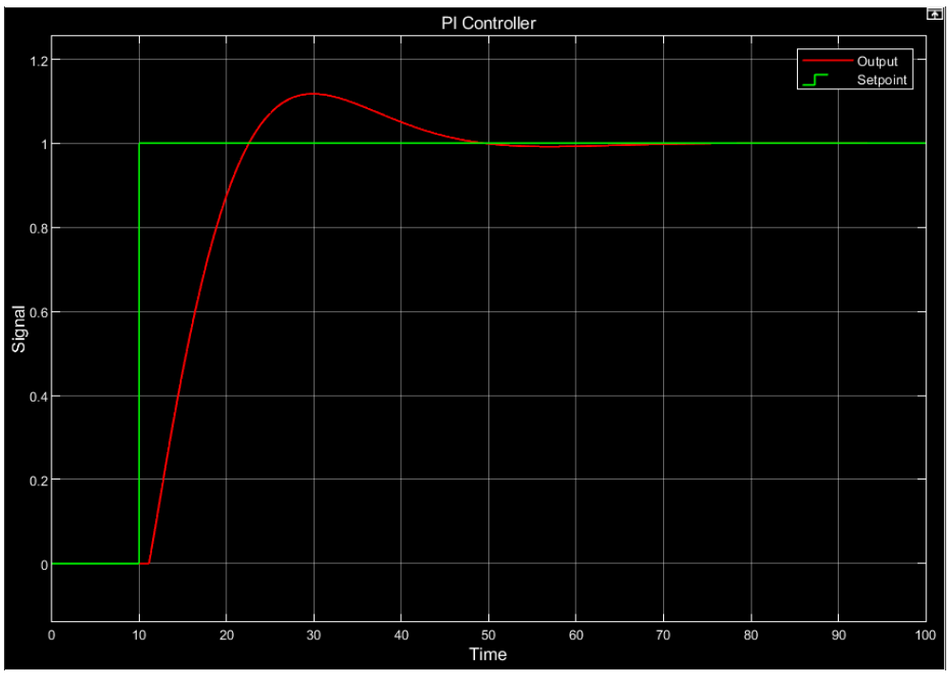
Parallel form transfer function of the controller:

$$K_c(1 + \frac{1}{\tau_I s}) \longleftrightarrow P + \frac{I}{s}$$

=> $P = 1.4154$
$I = \frac{K_c}{\tau_i} = 0.1194$







RED LINE: Y2 (BOTTOM COMPOSITION) GREEN: SETPOINT R2

PID controller Tuning:

Using SIMC method:

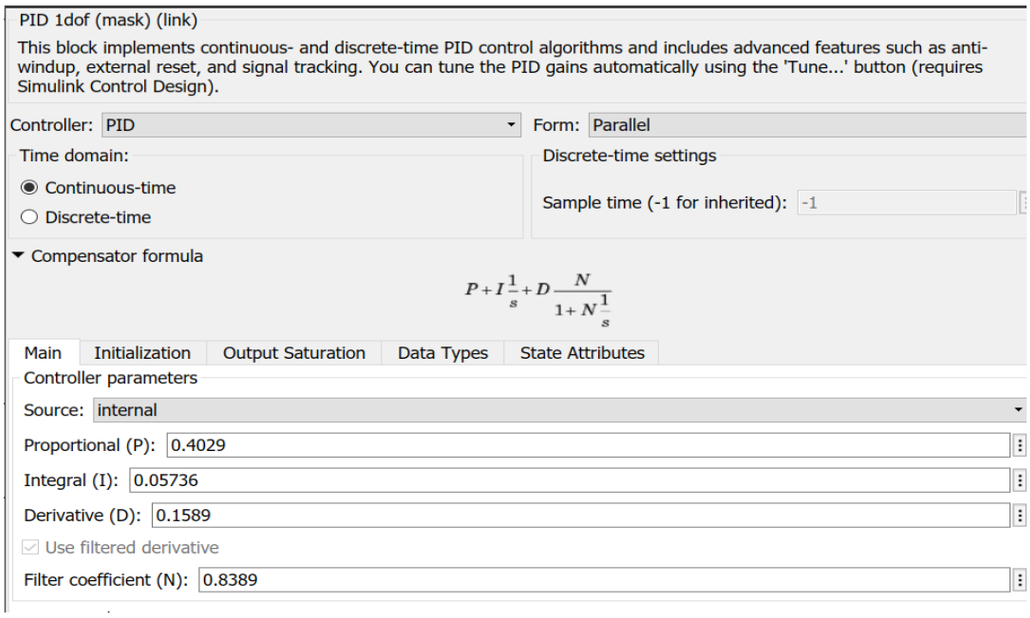
$$K_c = \frac{\tau + \theta/2}{K(\tau + \theta/2)}$$

Using Skogestad's guidelines for setting the control time constant (Skogestad, 2003):

$$\tau_c = \theta$$

=> $K_c = 0.7476$

$\tau_i = 12.409,\ \tau_d = 0.5256,\ \tau_f = 1.1$



Parallel form transfer function of the controler:

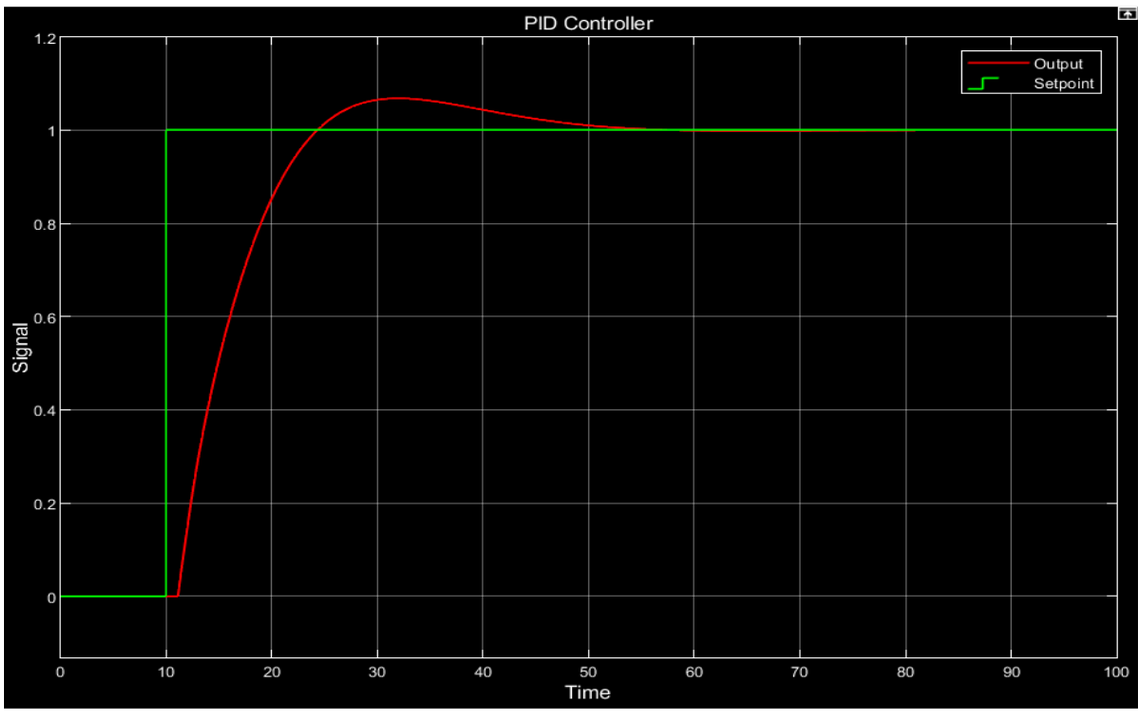
$$K_c(1+\frac{1}{\tau_i s}+\tau_d s)\cdot f \longleftrightarrow (P+\frac{I}{s}+Ds)\cdot\frac{1}{1+\tau_f s}$$

=> $P = 0.7476$

$I = \frac{K_c}{\tau_i} = 0.0602$

$D = K_c\tau_d = 0.3231$

$\tau_f = \tau c = 1.1$

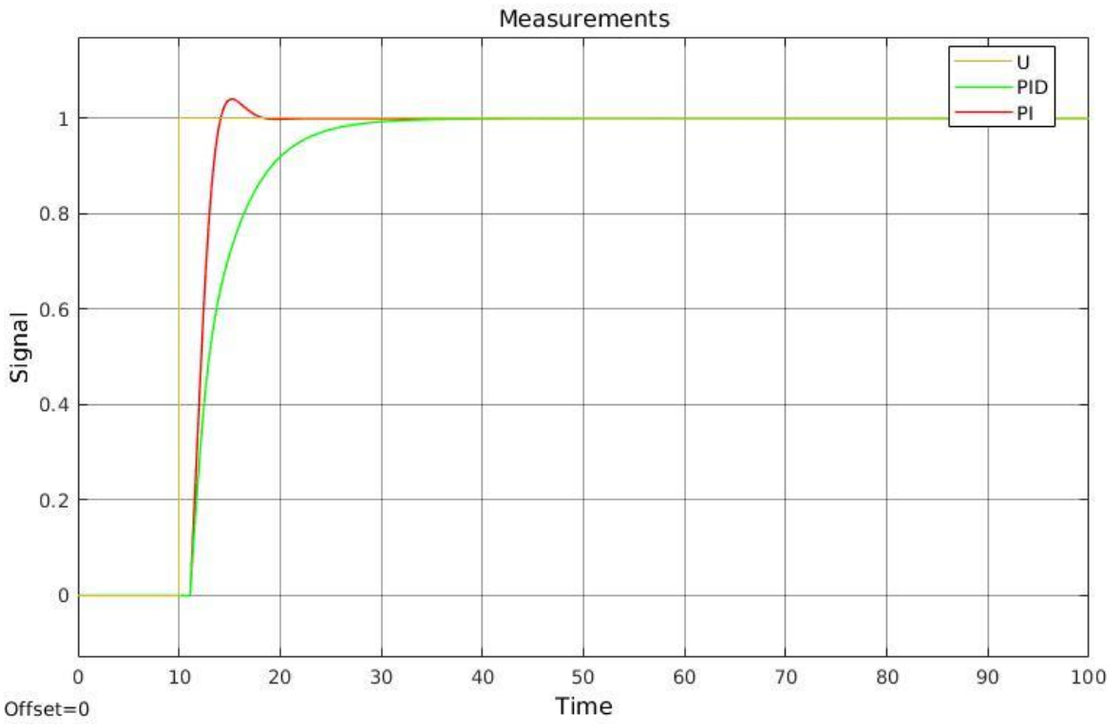


RED LINE: Y2 (BOTTOM COMPOSITION) GREEN: SETPOINT R2

The transfer function of the resultant PID controller is as follows:

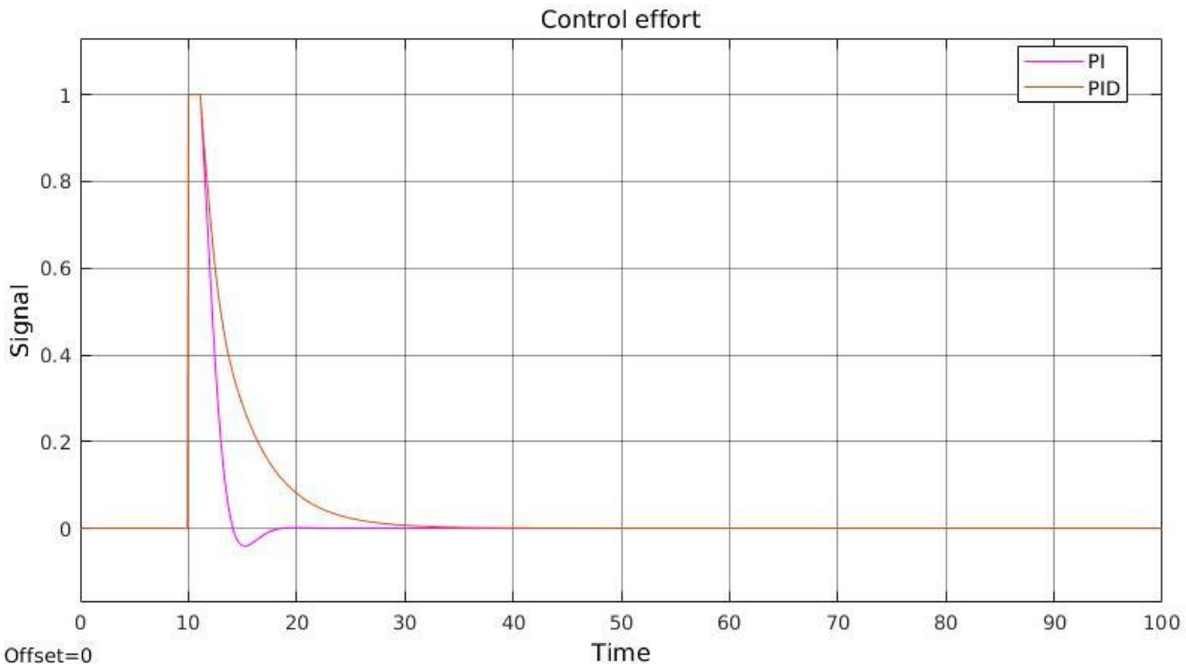
$$G_c(s) = f(s)K_c(1+\frac{1}{\tau_i s}+\tau_d s) = \frac{Ds^2+Ps+I}{\tau_f s^2+s} = \frac{0.3231s^2+0.7476s+0.0602}{1.1s^2+s}$$

The following are the plots for the measurements and control effort obtained by using the two controllers:



The input signal is steps up to 1 at 10s. From the above plot, we can observe that there is a delay of 1.1s between the change in setpoint and the first change in the output.

The PI controller is able to stabilise the output signal in around 18s, whereas the PID controller takes around 35s. We can also observe that the PI controller slightly overshoots the required steady state value, while still settling relatively quickly. In the case of the PID controller, there is no offshoot observed, although it takes a longer time to reach the steady state value.



Here, we can observe that once the control effort kicks in (after a delay of 1.1s), the PI controller exerts a negative control effort for a short duration. The PID controller exerts a higher, longer sustaining effort to reach the setpoint than the PI controller. From this, we can infer that the PI controller exerts less control effort when compared to the PID controller.

3: MIMO top product composition

PI Control:

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

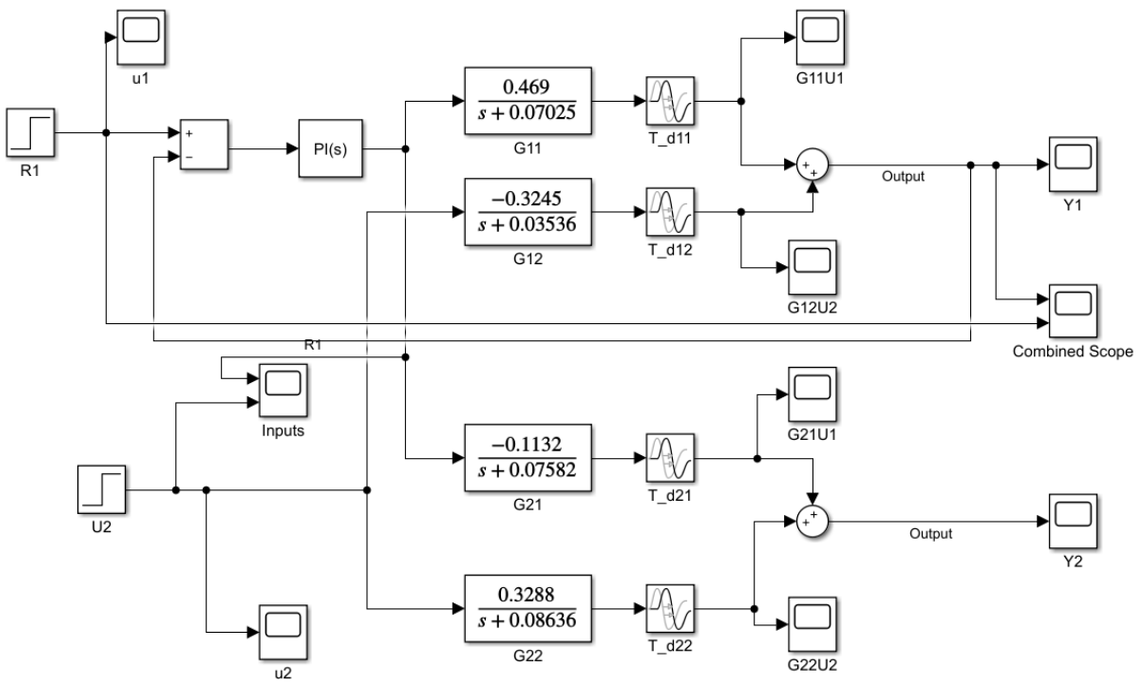
$$P + I\frac{1}{s}$$

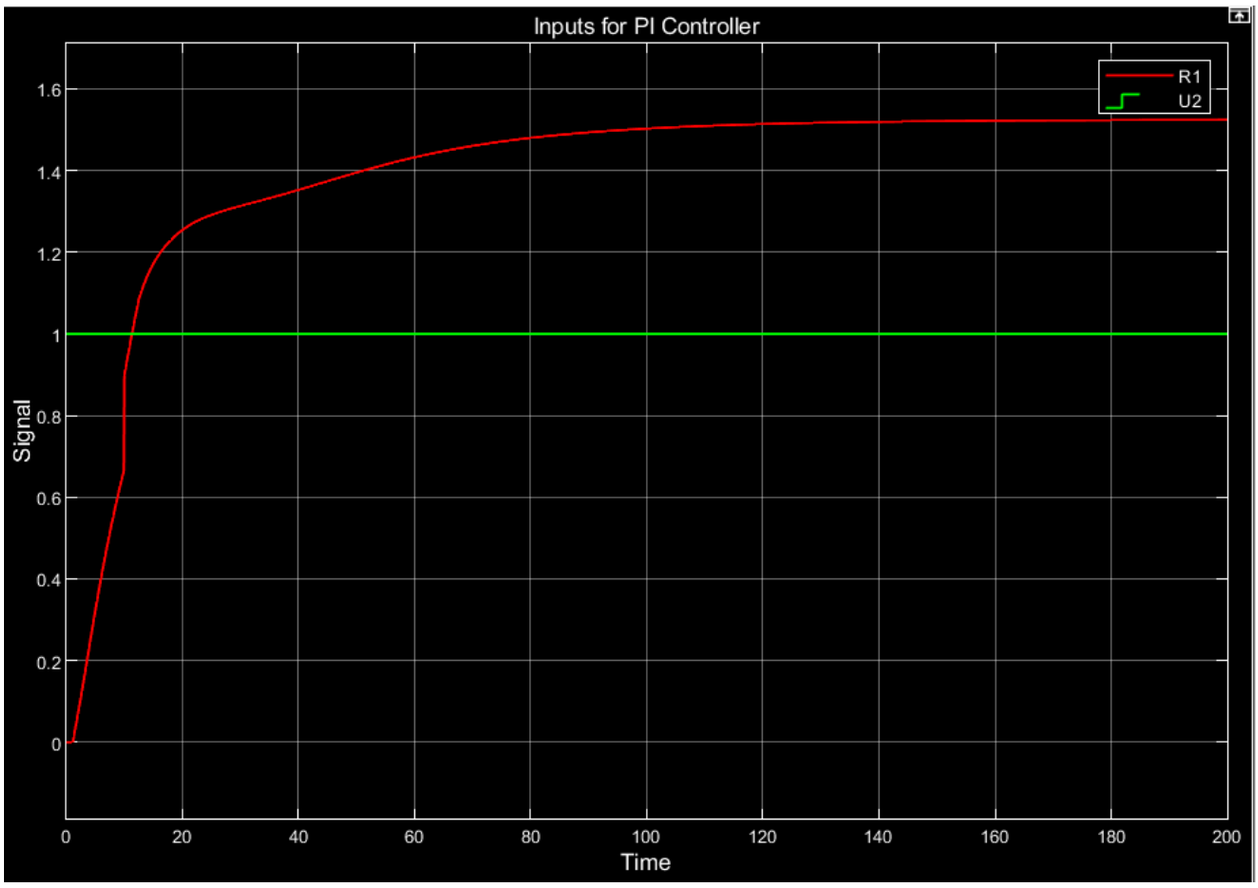
Main | Initialization | Output Saturation | Data Types | State Attributes

Controller parameters
Source: internal
Proportional (P): 0.2295
Integral (I): 0.02531

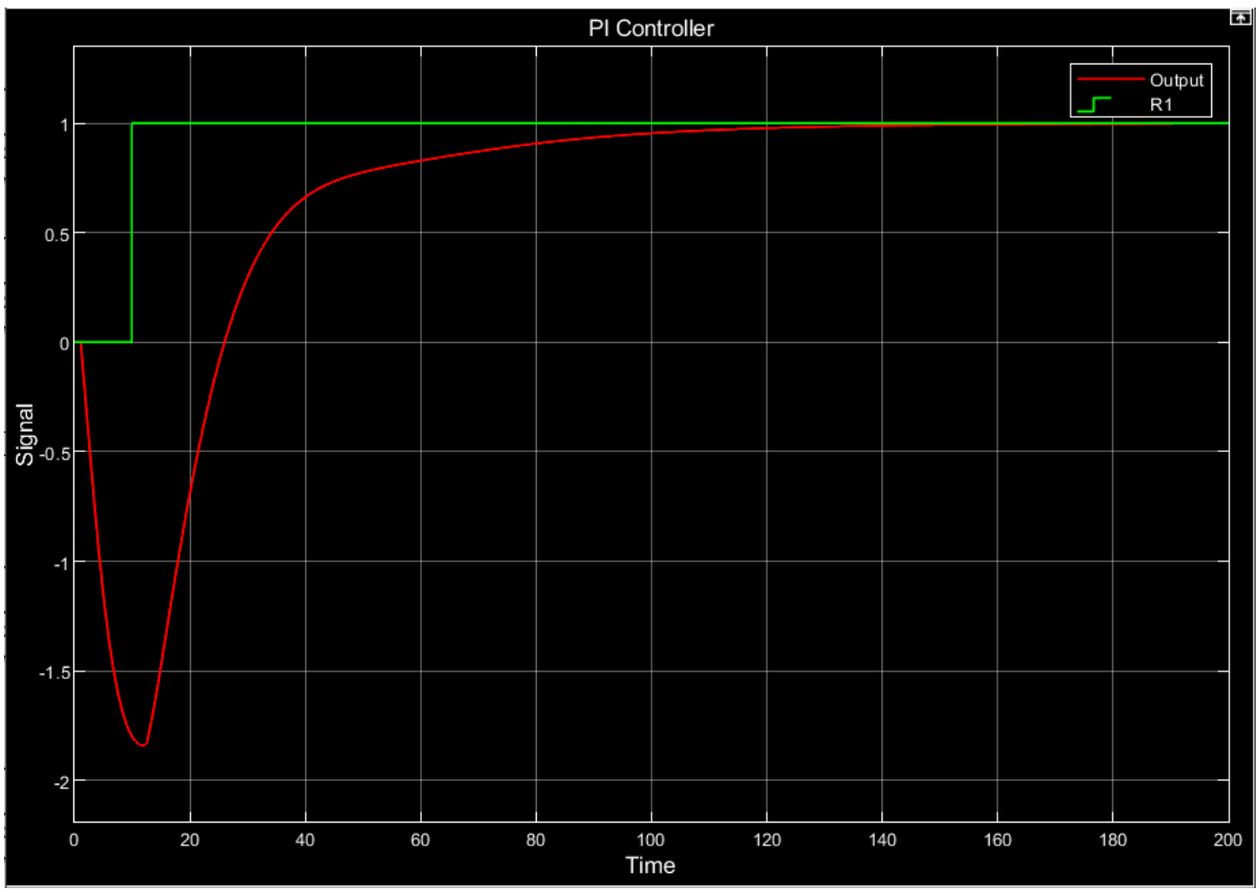
Automated tuning
Select tuning method: Transfer Function Based (PID Tuner App) Tune...

☑ Enable zero-crossing detection



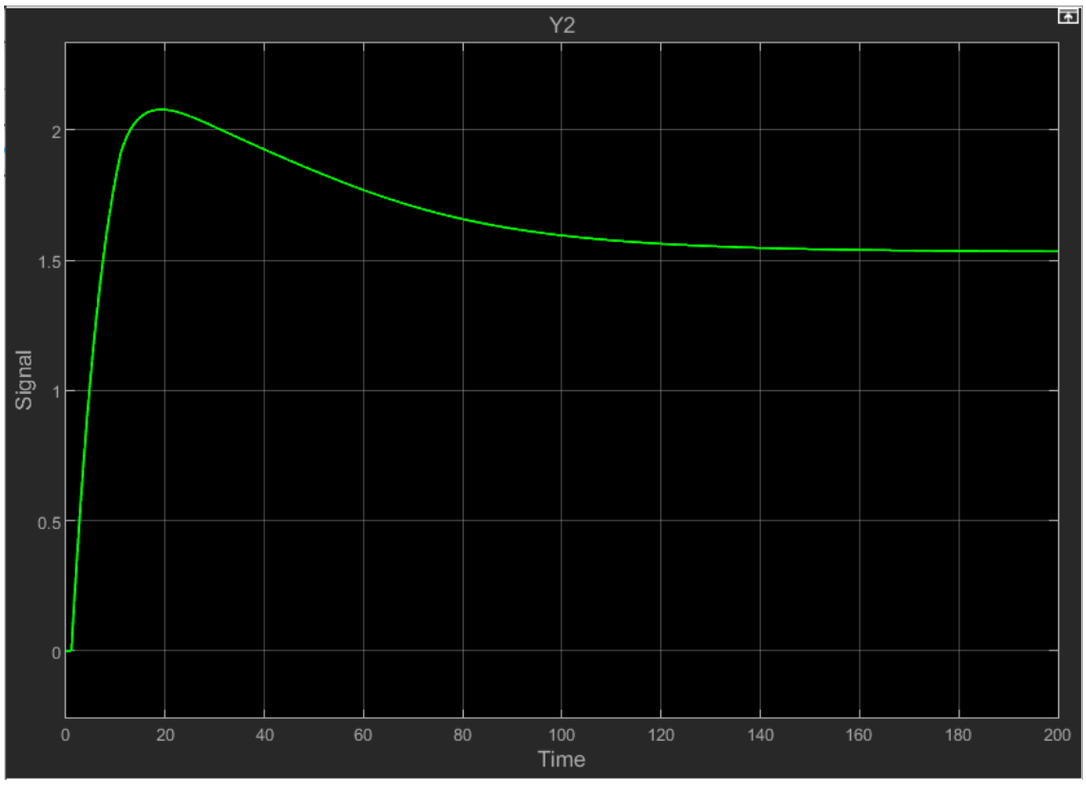


VARIATION OF INPUTS GIVEN BY PI CONTROLLER. RED LINE: INDICATES U1. GREEN: U2 (constant)



GREEN: SETPOINT; RED: TOP PRODUCT COMPOSITION

We see **composition** goes to negative value which is not physically possible. It could be because that the model would've been linearised around the steady state



Y2 (NOT CONTROLLED)

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

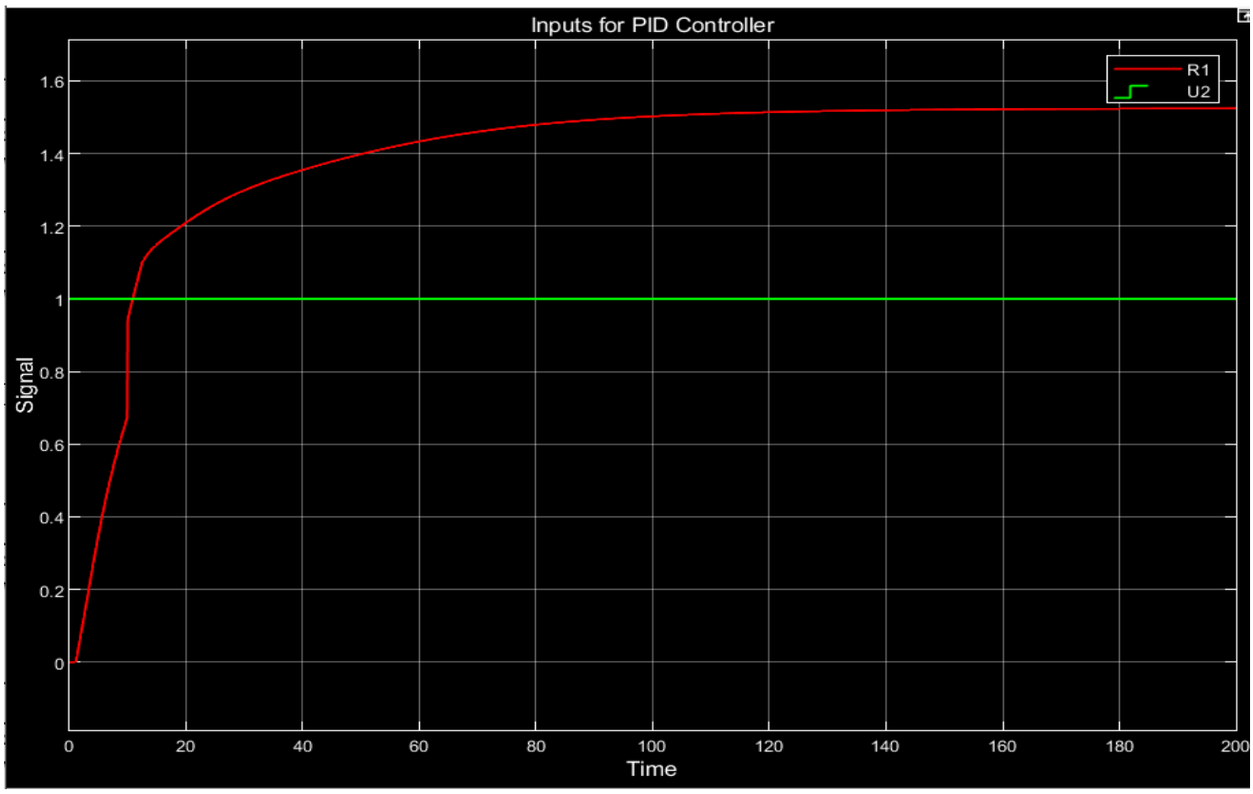
Proportional (P): 0.2373

Integral (I): 0.02458

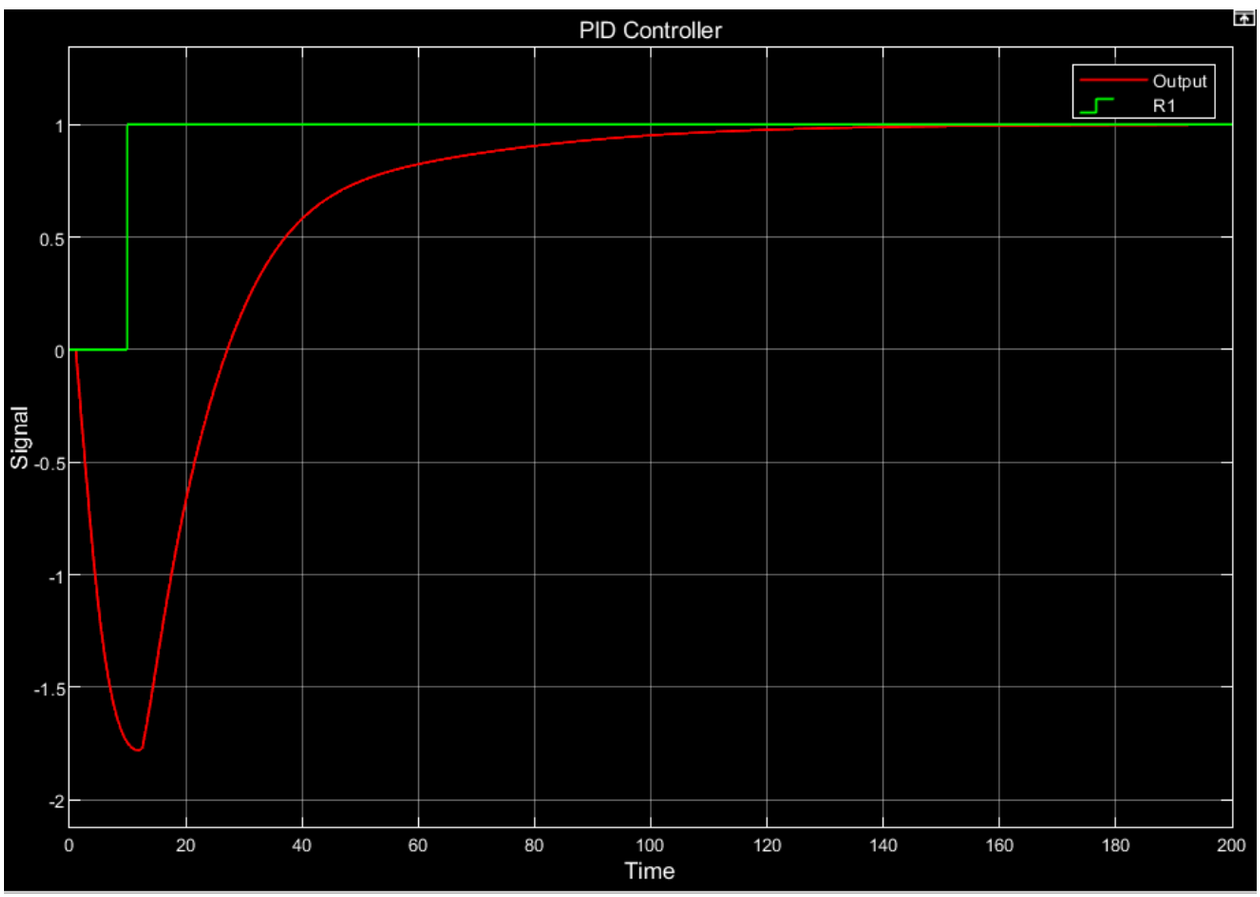
Derivative (D): 0.1366

☑ Use filtered derivative

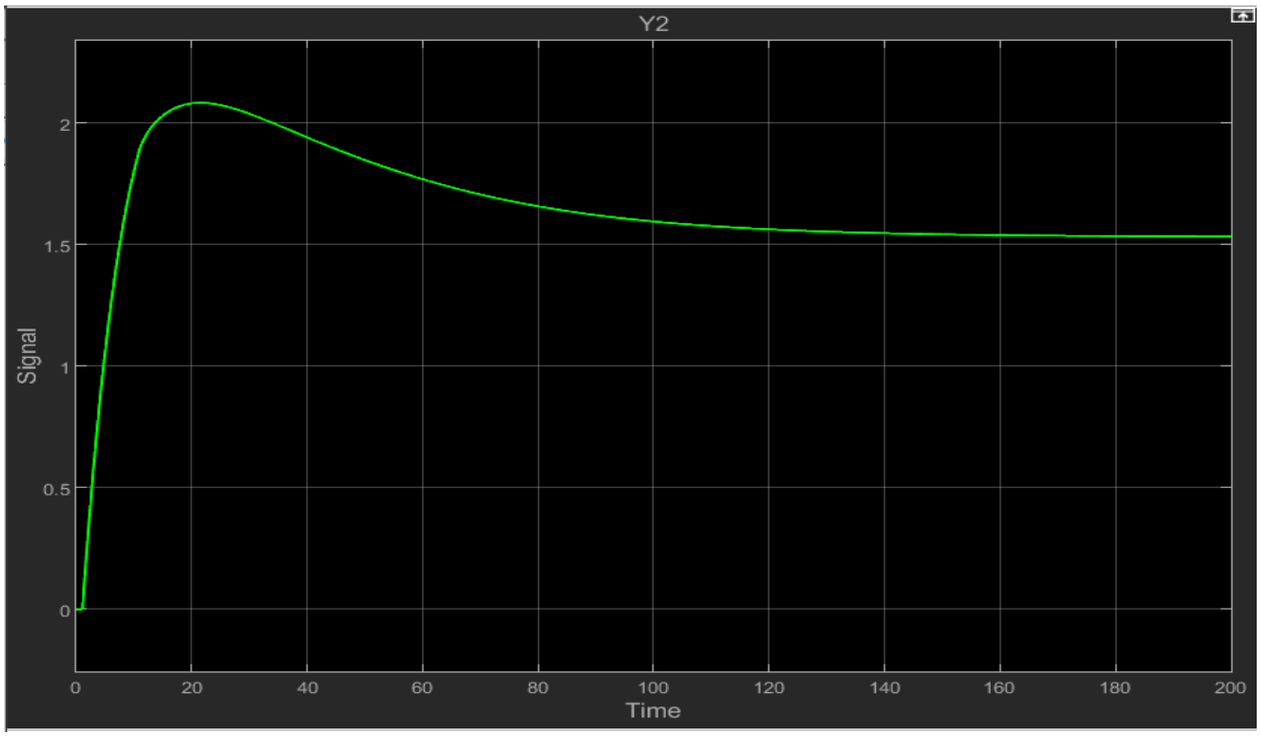
Filter coefficient (N): 0.2153



VARIATION OF INPUTS GIVEN BY PI CONTROLLER. RED LINE: INDICATES U1. GREEN: U2 (constant)



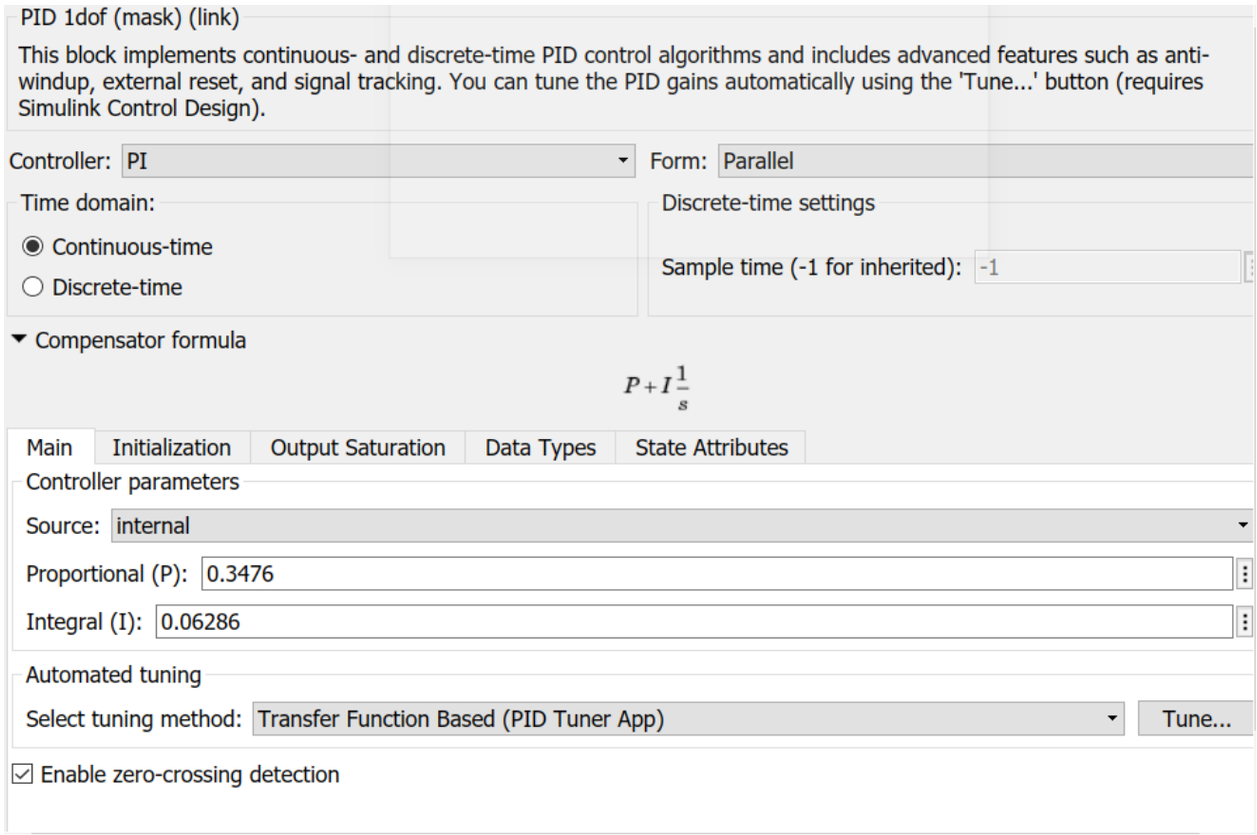
GREEN: SETPOINT; RED: TOP PRODUCT COMPOSITION

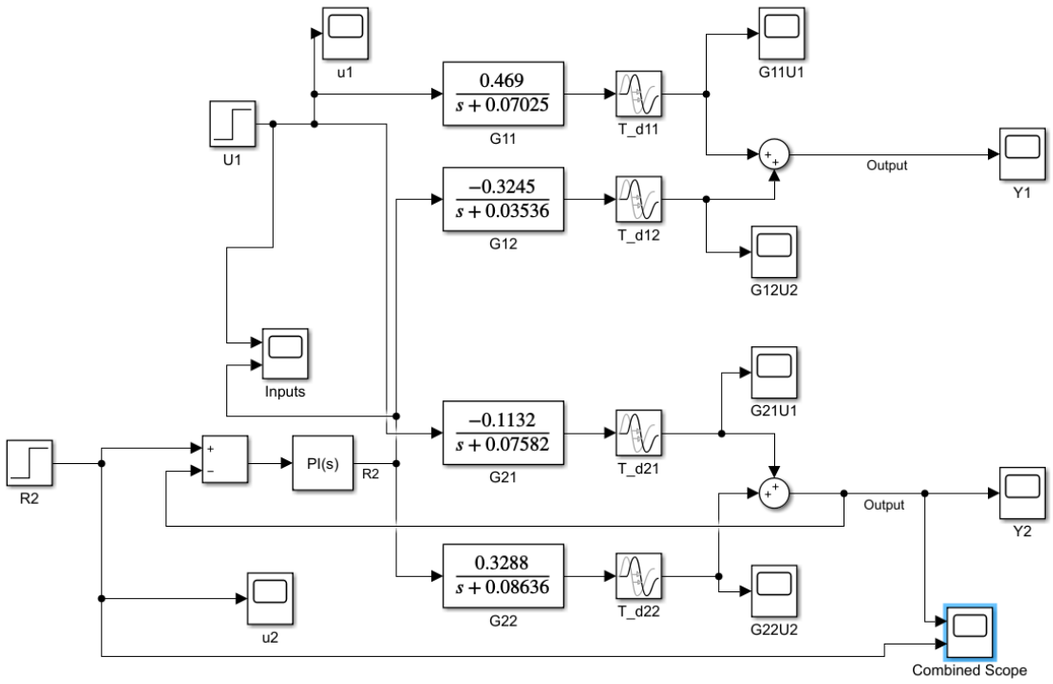


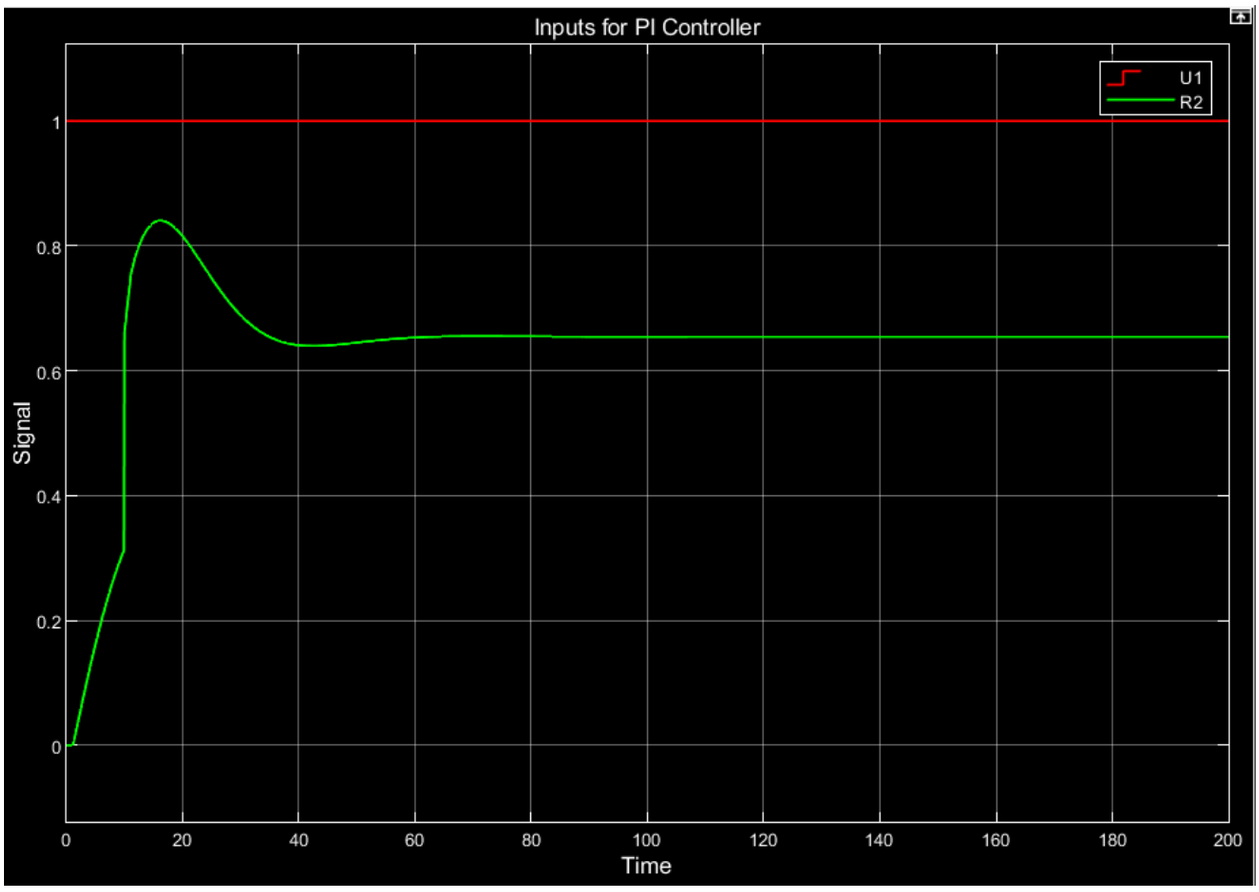
Y2 (NOT CONTROLLED)

4: MIMO bottom temperature

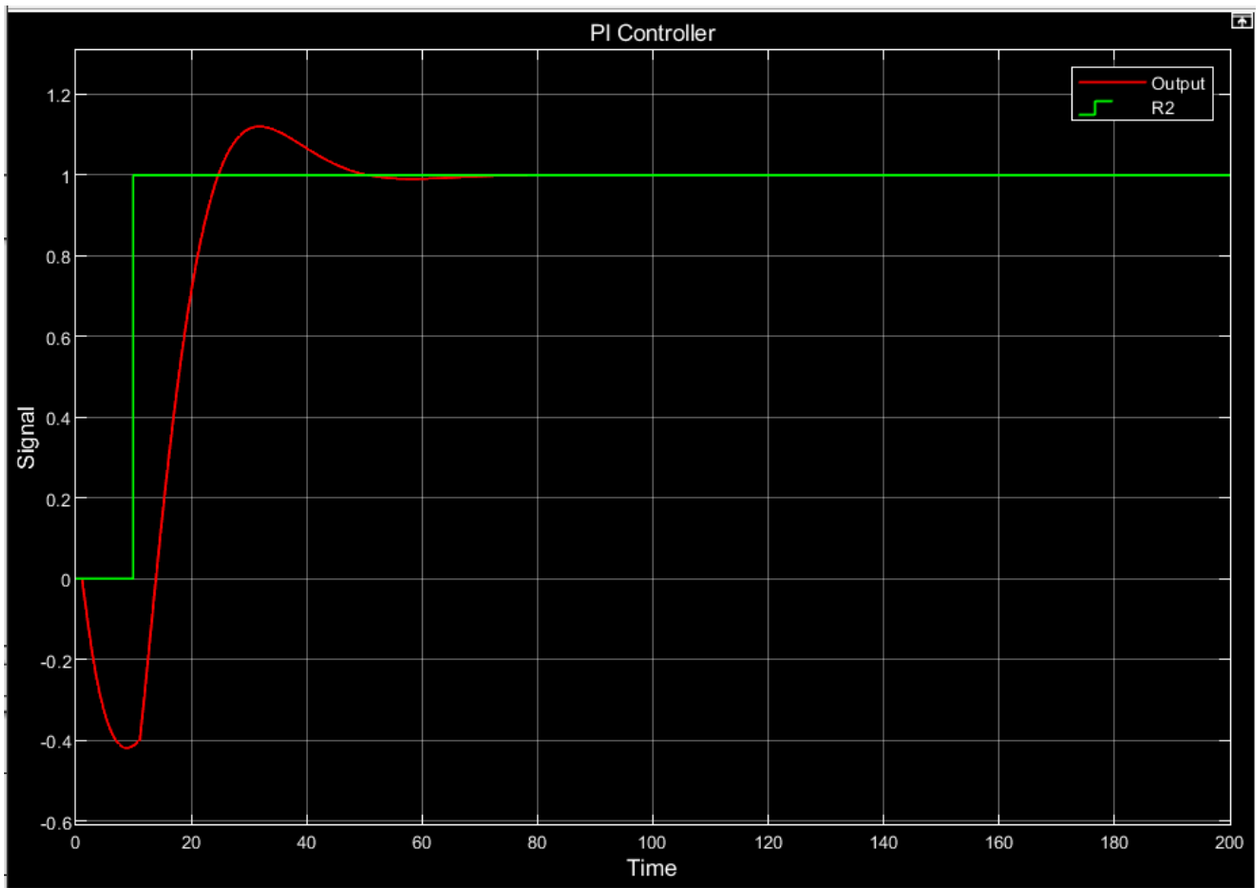
PI CONTROLLER



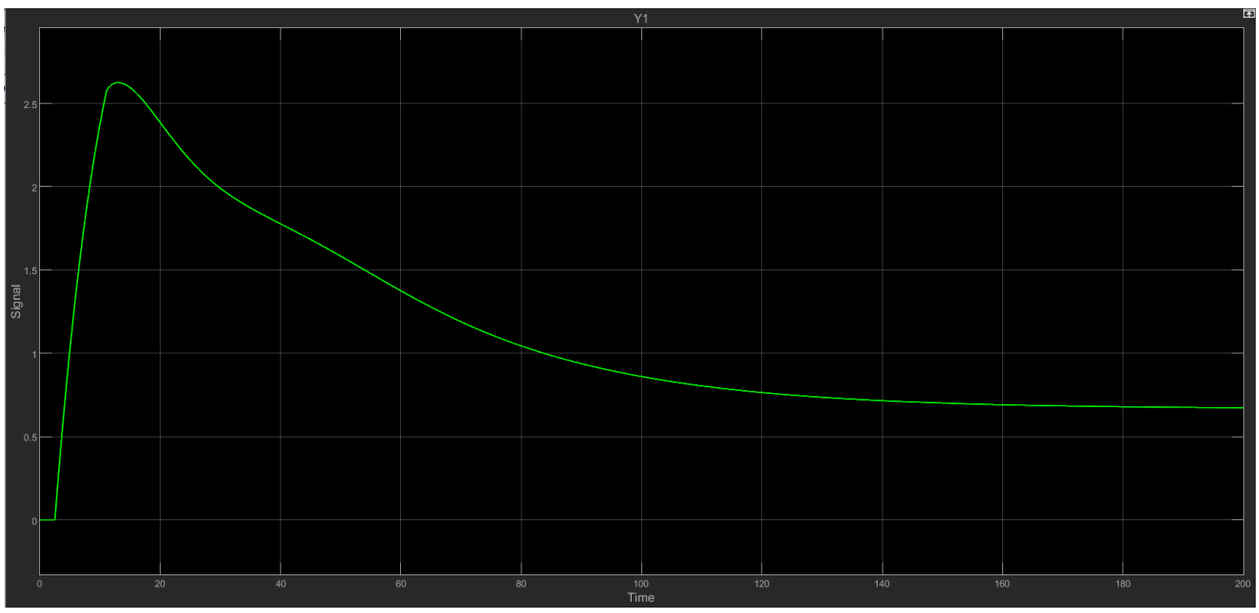




U2: GIVEN BY PI CONTROLLER



RED LINE: Y2; GREEN LINE:SETPOINT R2



GREEN LINE: OUTPUT Y1

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

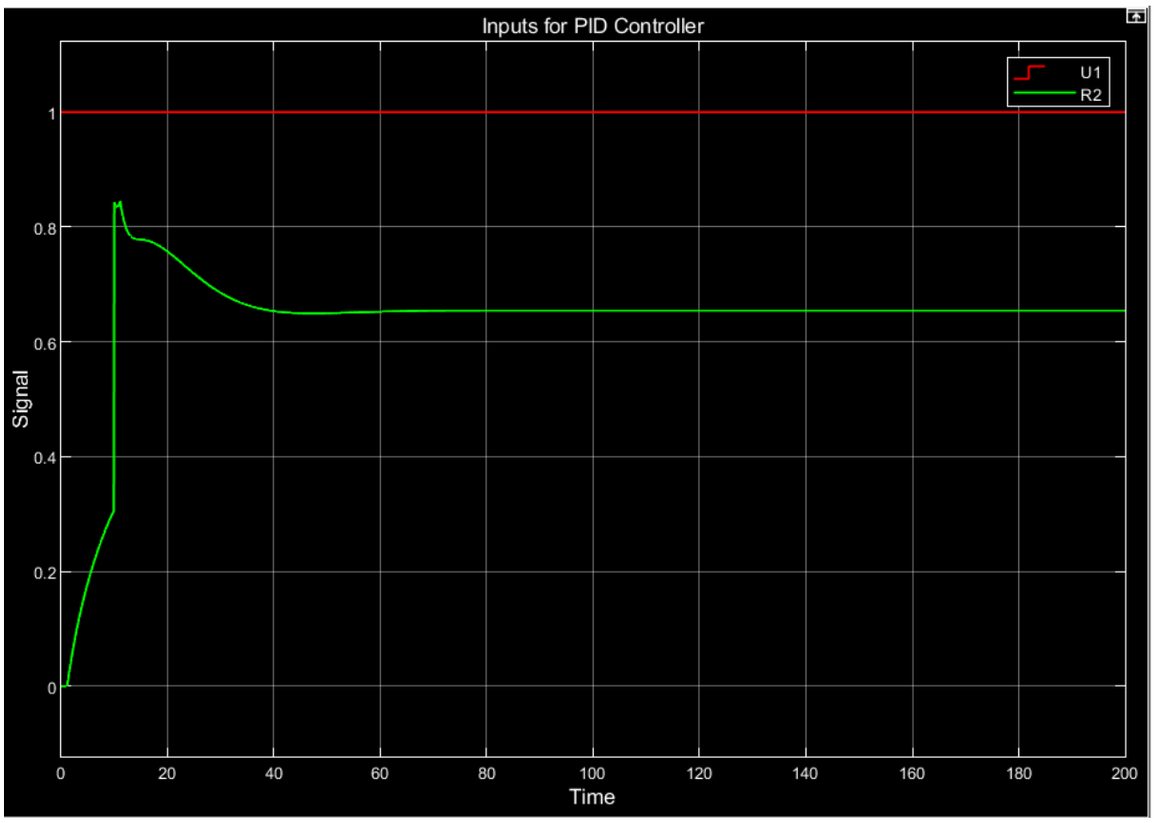
Discrete-time settings
Sample time (-1 for inherited): -1

▼ Compensator formula

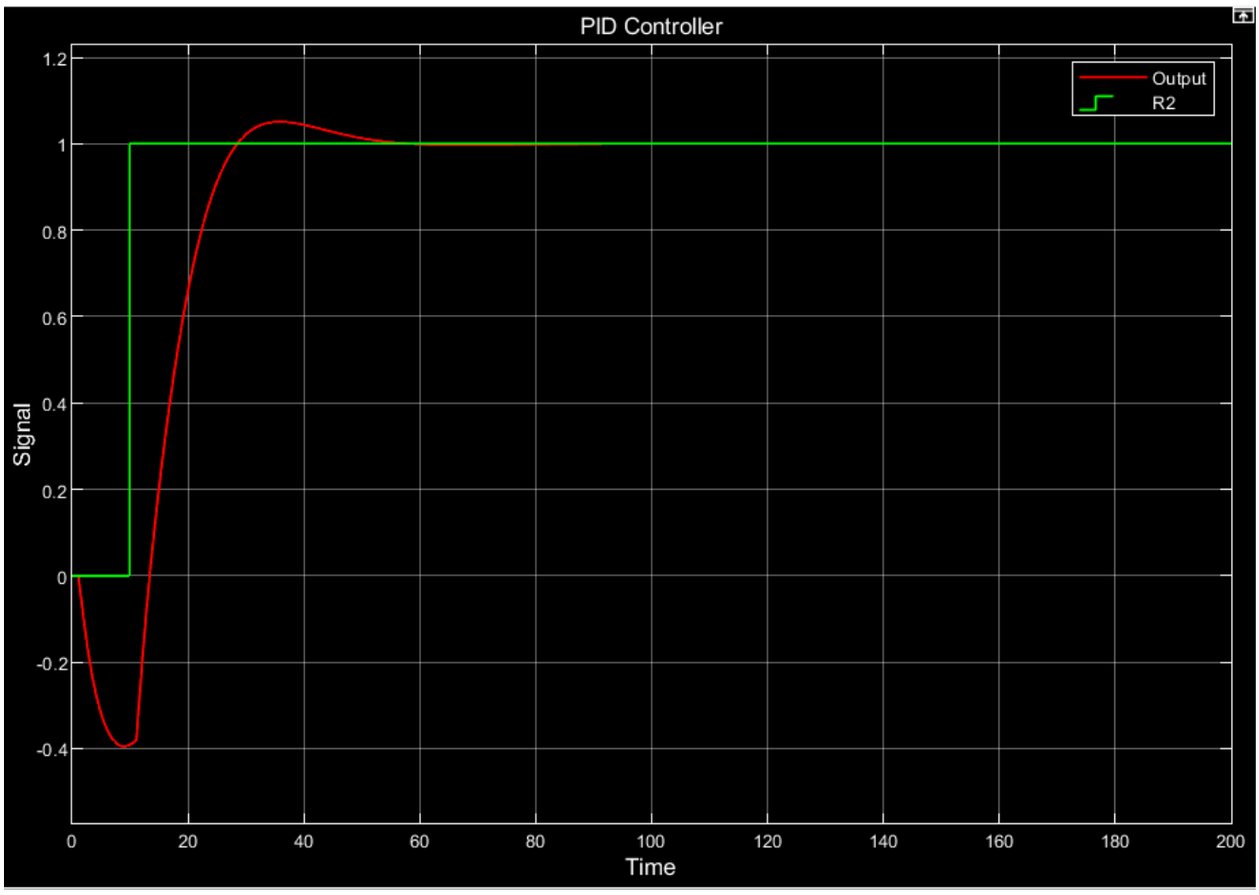
$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}$$

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

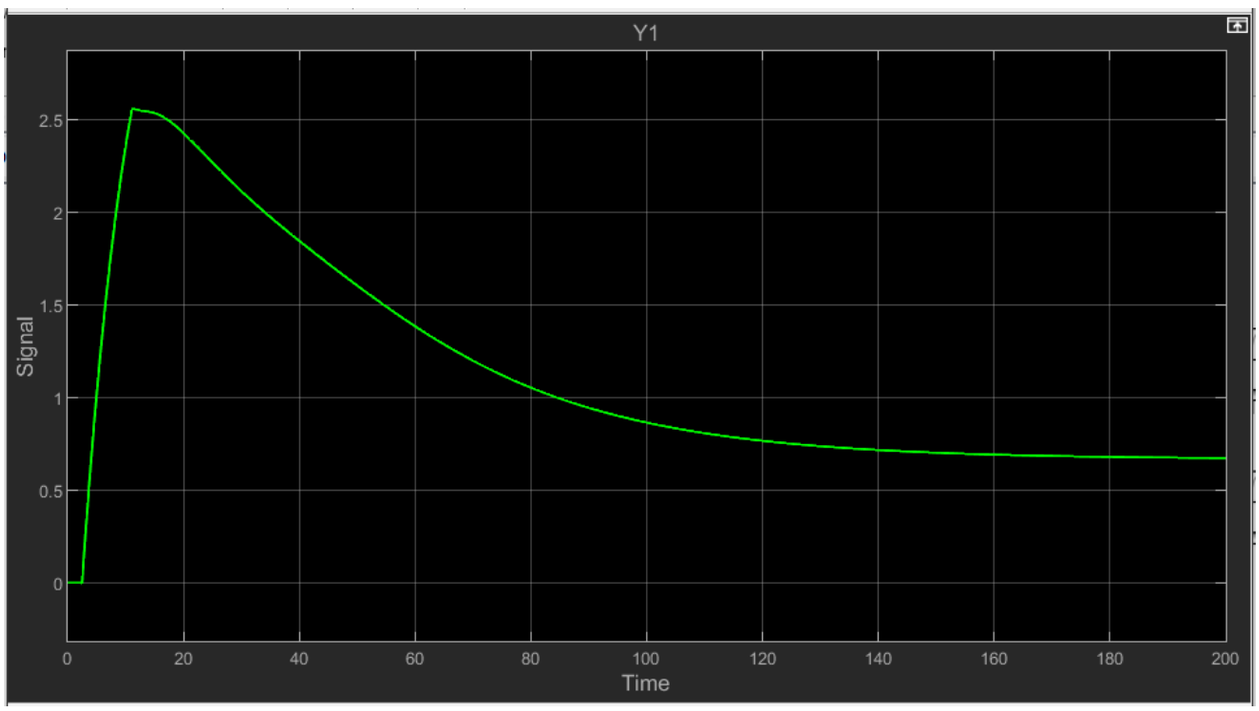
Controller parameters
Source: internal
Proportional (P): 0.4029
Integral (I): 0.05736
Derivative (D): 0.1589
☑ Use filtered derivative
Filter coefficient (N): 0.8389



U2: GIVEN BY PID CONTROLLER



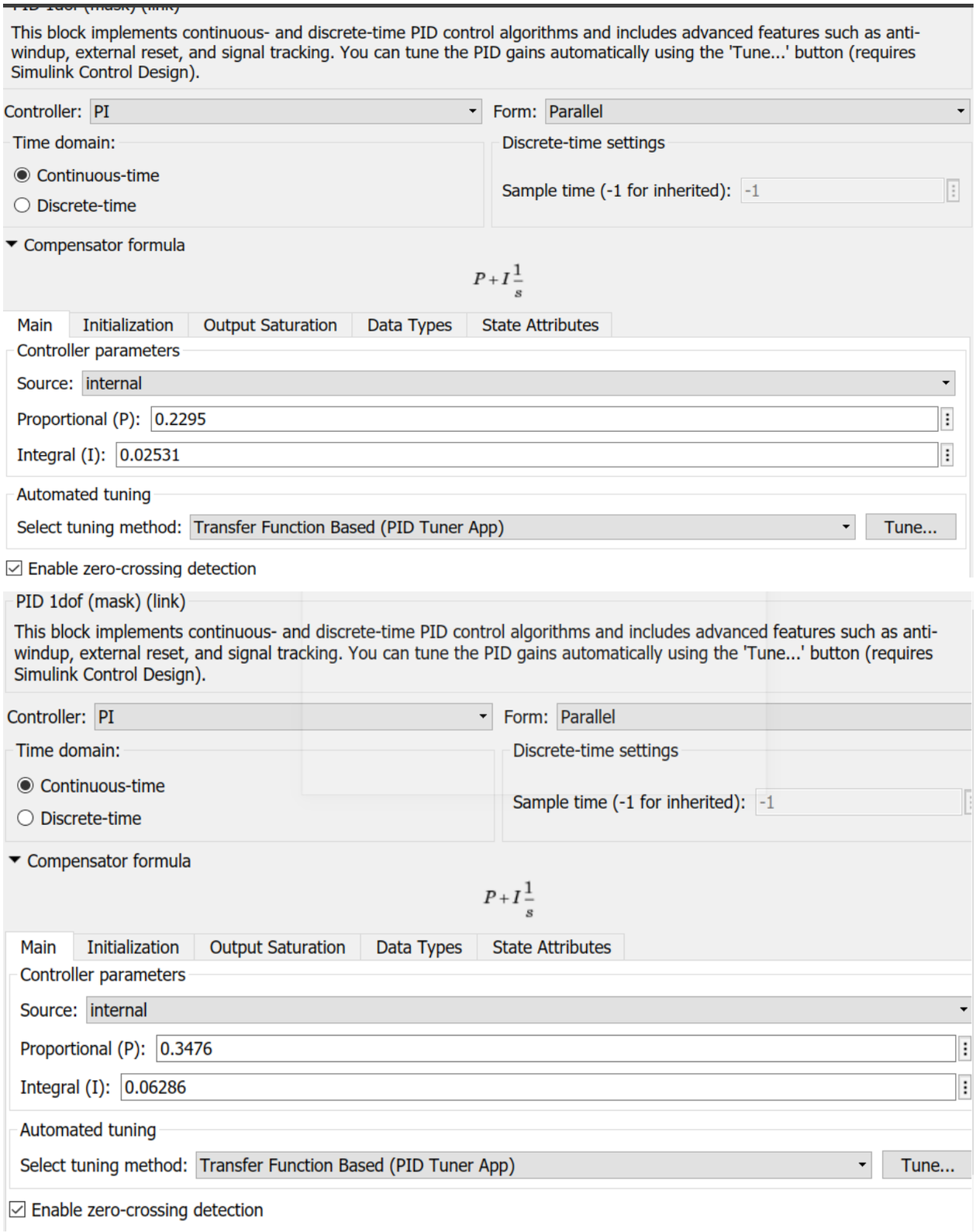
RED LINE: Y2; GREEN LINE:SETPOINT R2

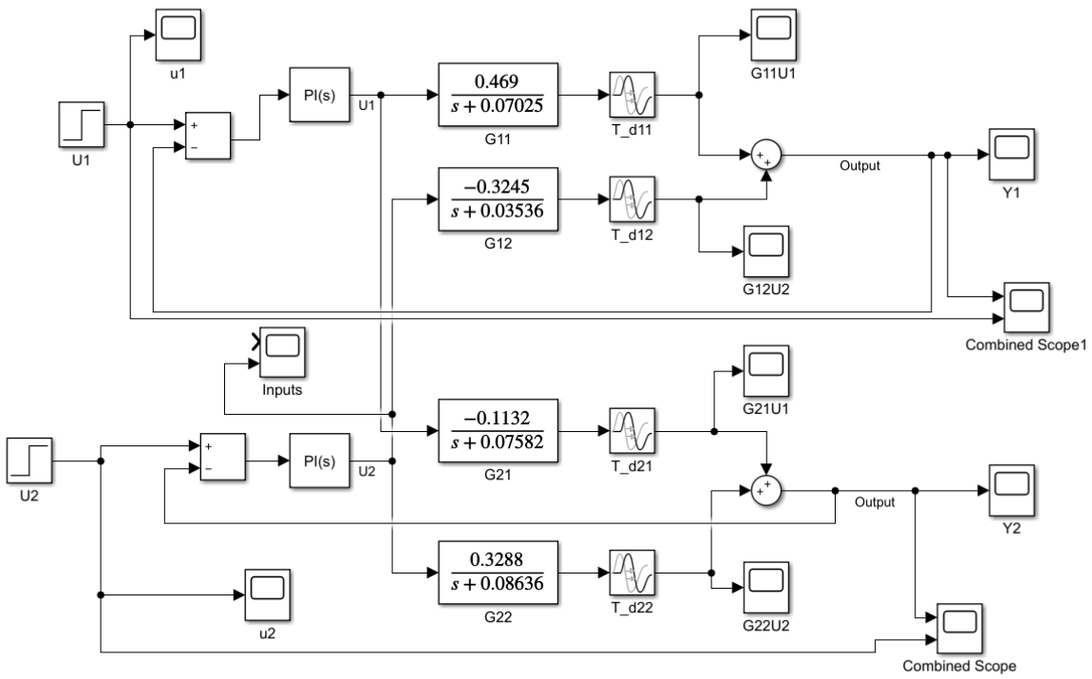


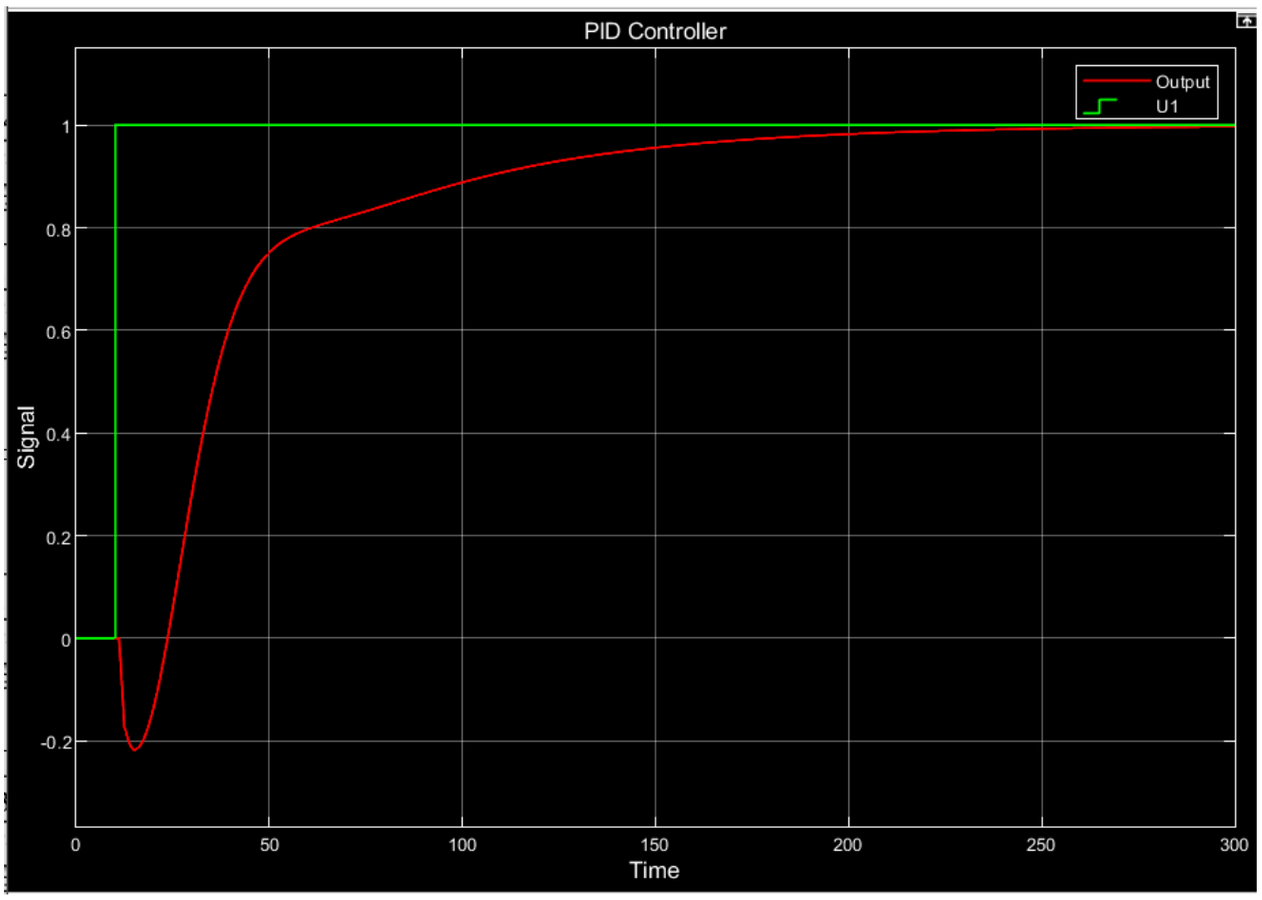
GREEN LINE: OUTPUT Y1

5: MIMO both loops active

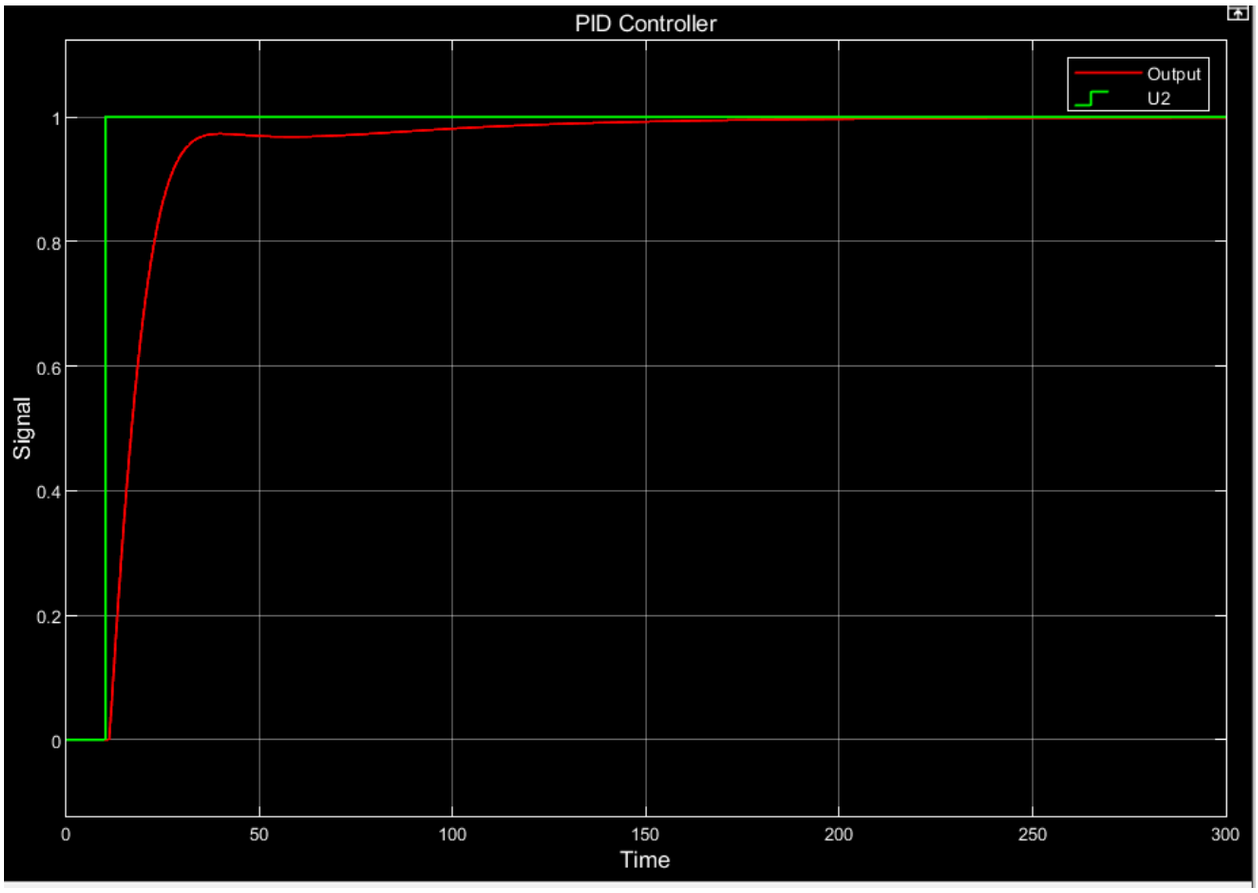
PI Controller:







GREEN: SETPOINT R1. RED: Y1 (TOP PRODUCT COMPOSITION)



GREEN: SETPOINT R2. RED LINE: Y2

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): 0.2373

Integral (I): 0.02458

Derivative (D): 0.1366

☑ Use filtered derivative

Filter coefficient (N): 0.2153

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

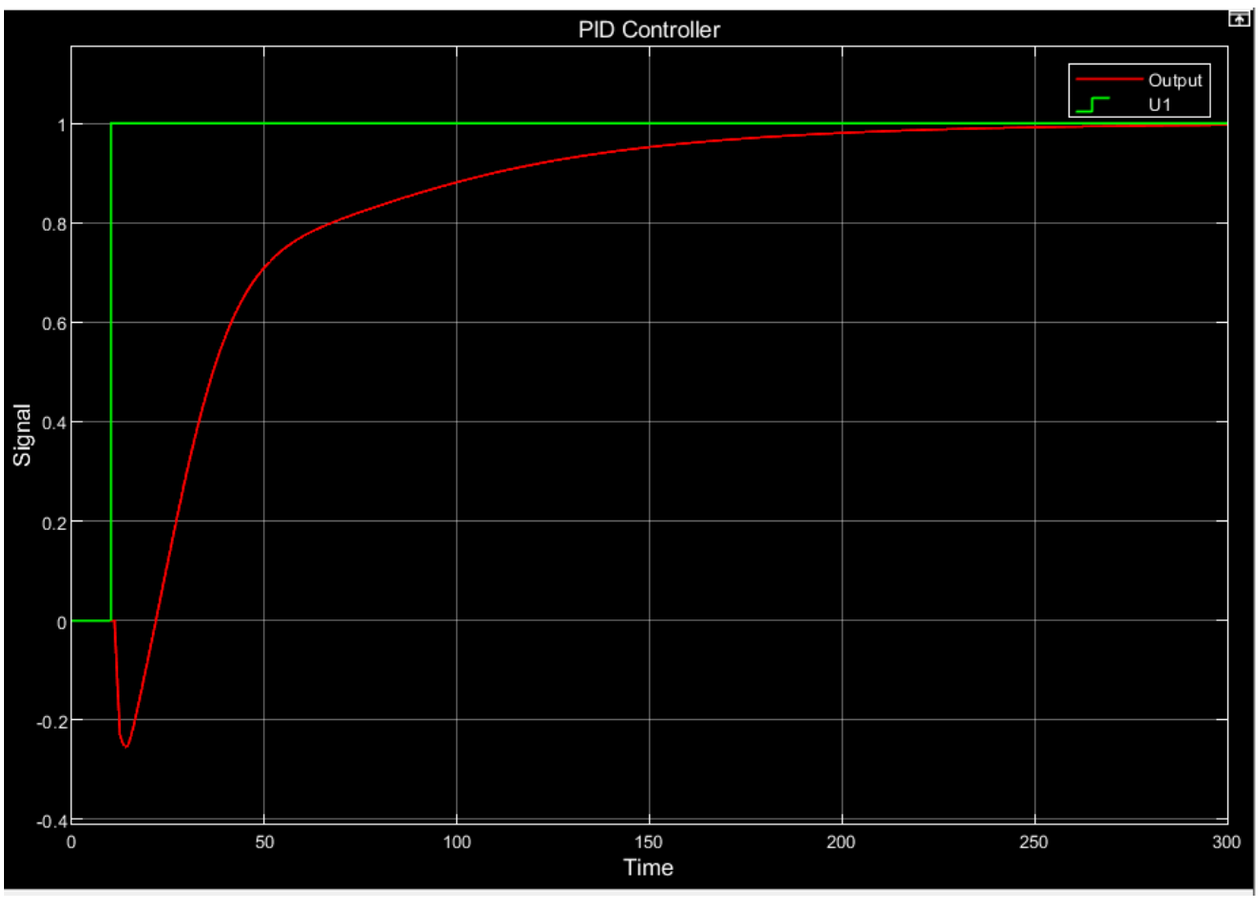
Discrete-time settings
Sample time (-1 for inherited): -1

Compensator formula

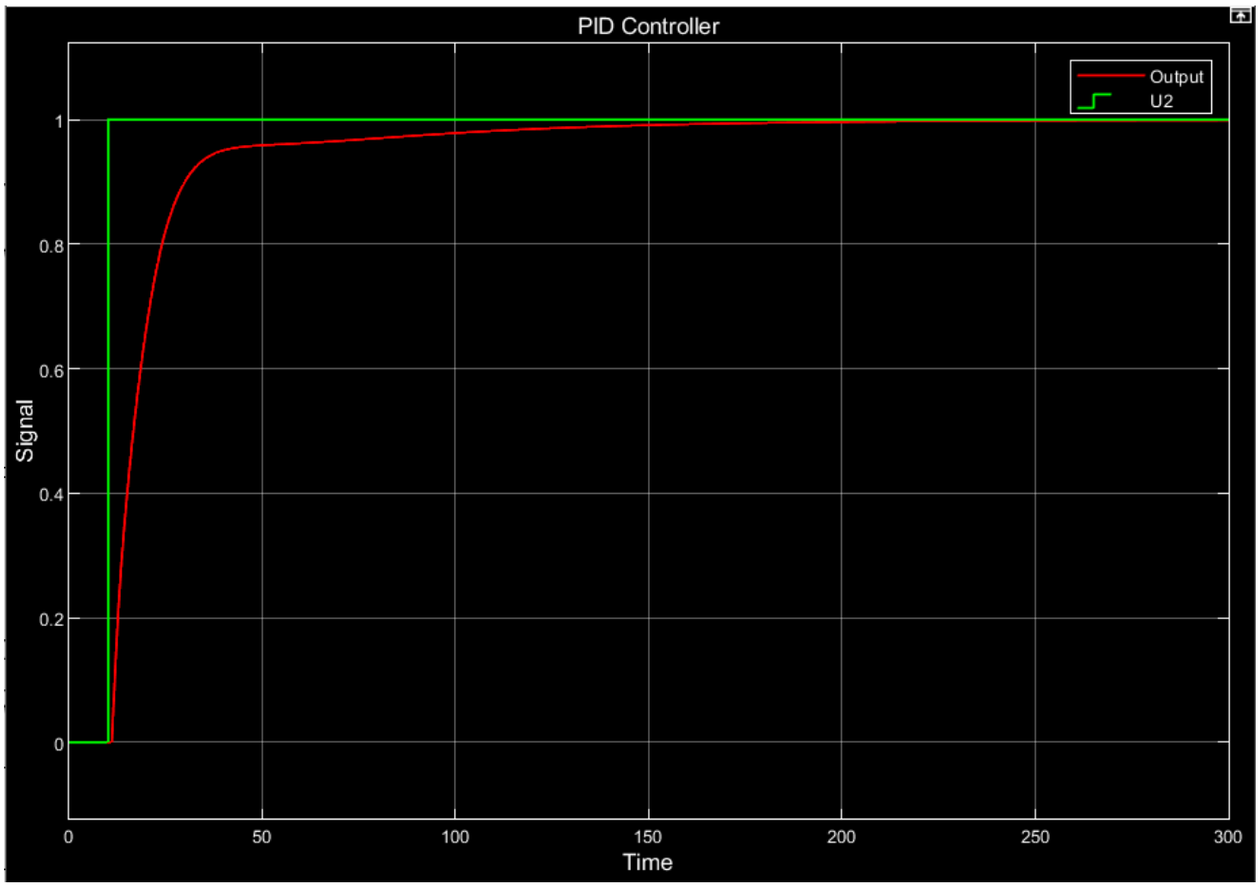
$$P + I\frac{1}{s} + D\frac{N}{1 + N\frac{1}{s}}$$

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

Controller parameters
Source: internal
Proportional (P): 0.4029
Integral (I): 0.05736
Derivative (D): 0.1589
[x] Use filtered derivative
Filter coefficient (N): 0.8389



GREEN: SETPOINT R1. RED LINE: Y1



GREEN: SETPOINT R2. RED LINE: Y2

Note: We used the controllers obtained from parts B3 and B4 for B5 because we weren't able to simultaneously tune both the controllers.

This method works because as far as PID controlling U1-Y1 loop is concerned, input U2 is like a disturbance. And since these are feedback controllers they are able to correct for them and reach the setpoint. We can employ the same line of reasoning for Y2. A disadvantage of this method is that the individual controllers don't recognise the presence of other controllers and at times their effects might be cancelling each other.

Strategies to circumvent this would either be trying to somehow decouple the loops by changing the manipulated variable so that it changes only one of the controlled variables. Or, one could go for controllers like MPC (model predictive control) which poses an optimization problem and solves for optimal U1 and U2 simultaneously.

General observation for part 1b): We see that PID controllers have slightly less overshoot/undershoot (more robust) and slightly faster settling times compared to their PI counterparts. In some cases it has more control effort too.

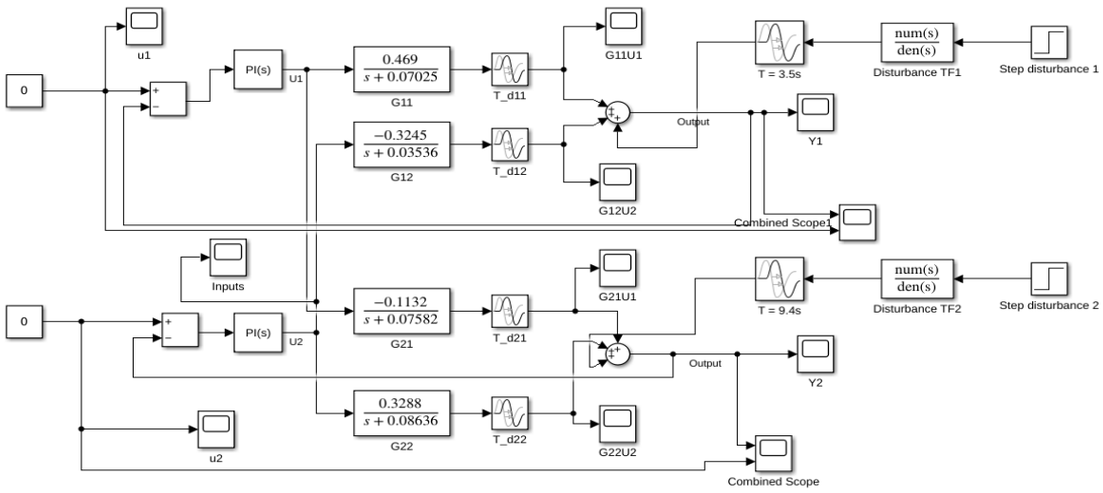
Part c) Effect of Disturbances

The tuned PI and PID controllers (from B5) will now be evaluated on the basis of their disturbance rejection capability. The disturbance signals that we'll use will be assumed to be output disturbance signals, and are therefore added to the process output of the top composition loop and the bottom temperature loop. For the purposes of controller design, it is assumed that the disturbance signal is not measured, and that the disturbance transfer functions are unknown.

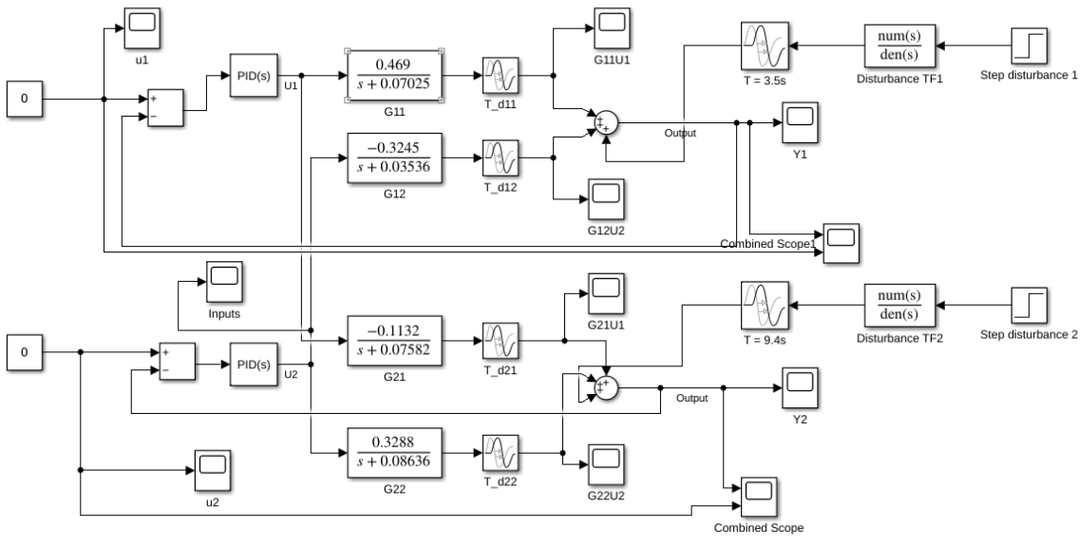
1: Step disturbance

The disturbance signals are again assumed to be output disturbance signals and therefore are added along with the process output of the top composition loop and the bottom temperature loop. The step time for the SIMULINK blocks has been set to 10s and the set magnitude is set to 5. The setpoint is assumed to be a constant value, and therefore no deviation from the setpoint is assumed throughout the process.

The SIMULINK model with PI controllers is as follows:

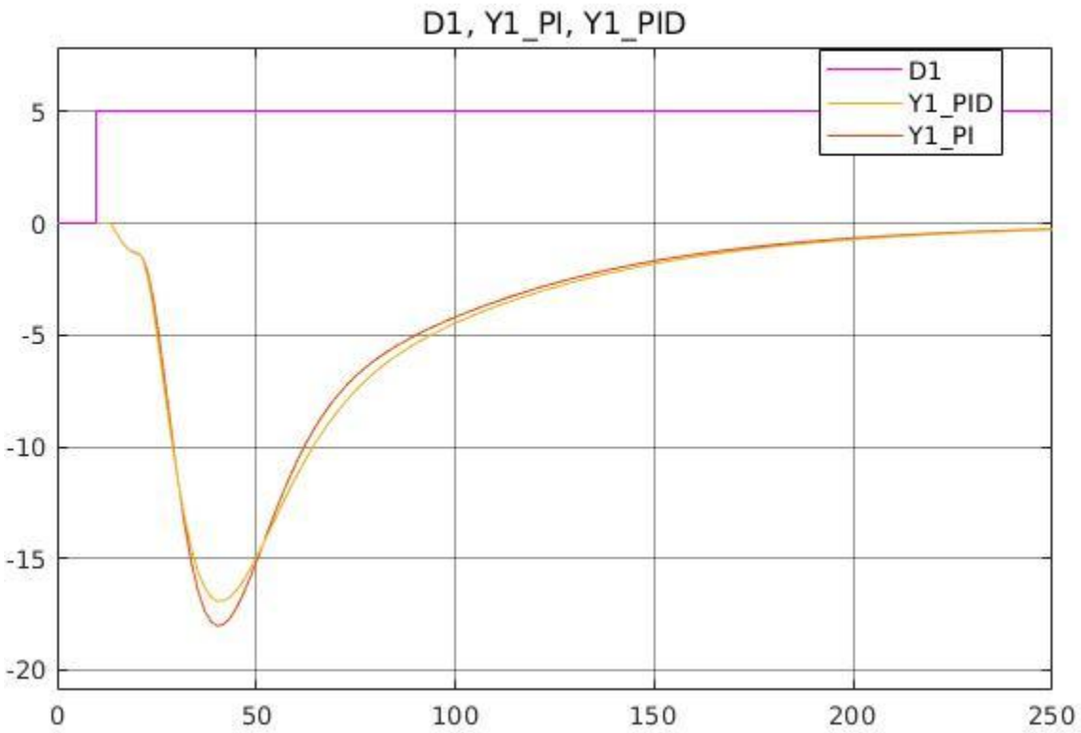


The SIMULINK model with PID controllers is as follows:

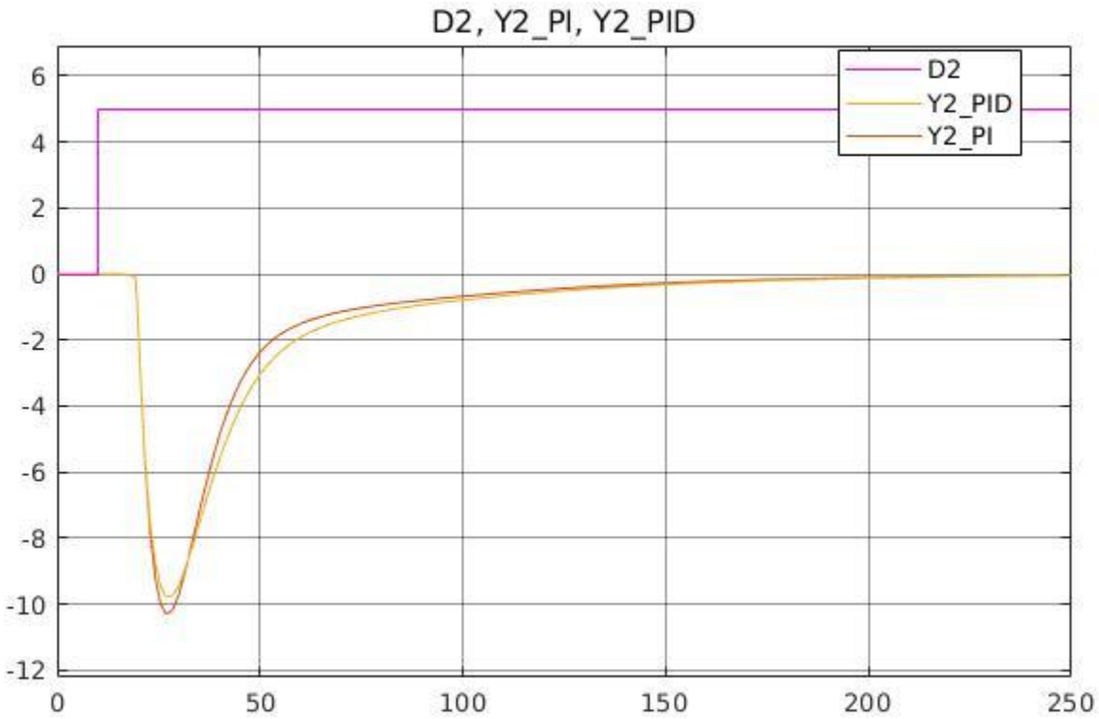


Results:

Simulation results for Y1:



Simulation results for Y2:



Y1:

The output signal begins to deviate 3.5s after the setpoint change has been introduced (t = 10 + 3.5 = 13.5s), when the step disturbance signal arrives. At 10.5s after the setpoint change (t = 20.5s), the coupling action caused due to action of the PI controller on the bottom temperature loop causes an additional disturbance on the output signal Y1, which causes further deviation from the setpoint. The delay in this coupling behaviour is due to the delay in the disturbance transfer function of D2 (9.4 seconds) combined with the delay in the transfer function G21 (1.1 seconds). With the addition of this secondary disturbance, the controller slowly reduces the magnitude of deviation and eventually is able to entirely reject the disturbance, at around t = 250 seconds.

The above is true for both the PI and PID controllers, although settling occurs faster with the PID controller.

Y2:

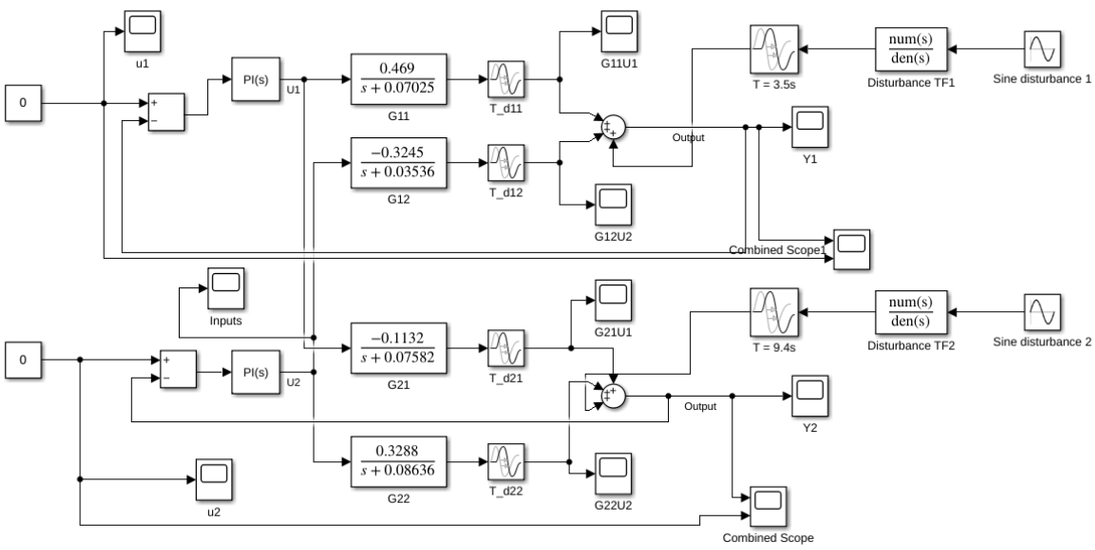
The output signal is initially affected by the disturbance caused due to the action of PI controller of the top product composition loop, through transfer function G21, at 4.6 seconds after t = 10s (16.4 seconds). This delay is due to the delay in disturbance transfer of D1 (3.5 seconds) and delay in transfer function G21 (1.1 seconds). This disturbance causes a slight deviation from set-point in the response signal. However, at 9.4 seconds after t=10 (19.4 seconds), the disturbance due to D2 causes a significantly larger deviation, as expected due to the large difference in gain magnitudes of D1 and D2. This deviation is steadily countered by action of controller and steady state disturbance rejection is achieved at around t = 200 seconds.

The above is true for both the PI and PID controllers, although settling occurs faster for the PI controller in this case.

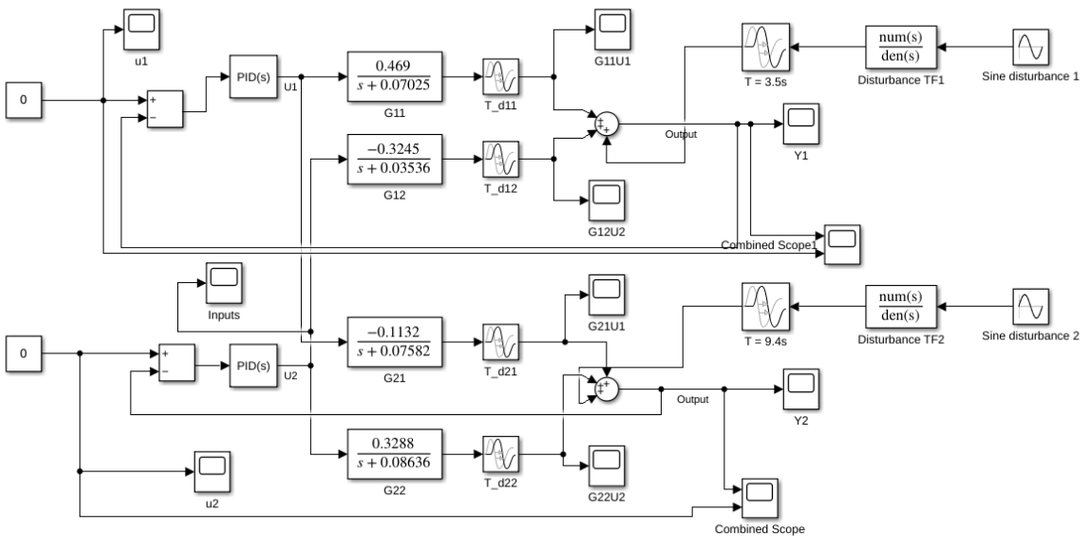
Therefore, we may infer that the disturbance rejection capability of the PID controllers used is better than that of the PI controllers used. However, as seen in Question 1B, purely for the purposes of setpoint tracking, the PI controller performs better and requires less control effort.
Therefore, the IMC-based PI and PID controllers are seen to have their respective comparative advantages over each other. Performance in set-point tracking may still be improved by modelling the controllers after taking into account the coupling interactions. Moreover, performance in disturbance rejection may also be improved under the assumption that the disturbance model is known beforehand, and that the disturbance is measured.

2: Sinusoidal disturbance

Now, we will apply a sinusoidal disturbance with frequency 0.01Hz (0.0628 rad/s) and amplitude 1. The SIMULINK sine wave blocks are set as prescribed. The setpoint is assumed to be a constant value, and therefore no deviation from the setpoint is assumed throughout the process. The assumption made here is that the disturbance rejection model is unknown for the purposes of controller design, and therefore it is not possible to design the IMC-based controllers to target disturbance rejection of the sinusoidal signals.
The SIMULINK model with PI controllers is as follows:



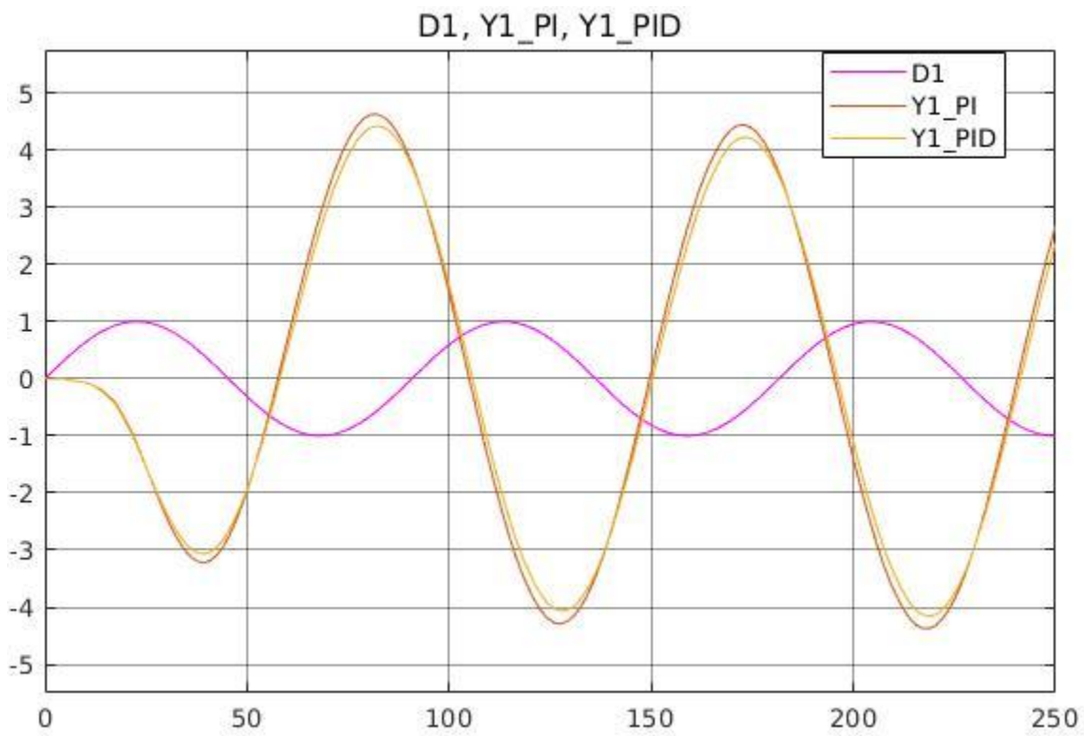
THe SIMULINK model with the PID controllers is as follows:



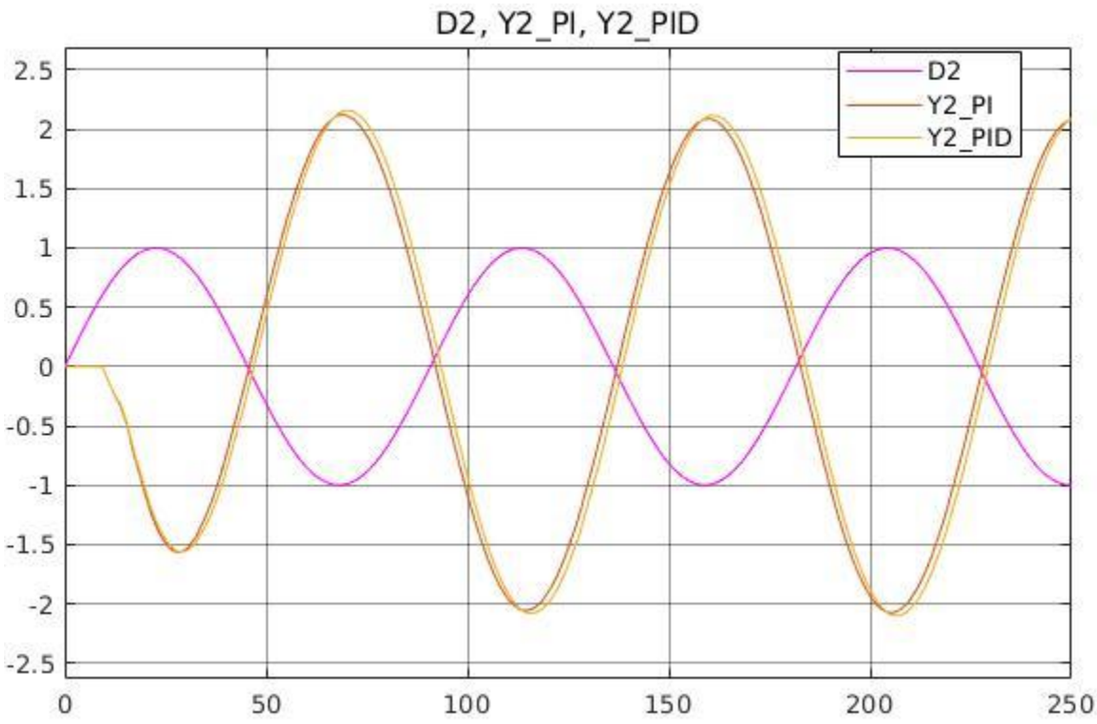
It is not expected that either the PI controller or the PID controller setup described above would be capable of complete disturbance rejection, in which case the steady-state output response would be 0 in the absence of a setpoint change. This is because, by the Internal Model Principle, in order for complete disturbance rejection, the generating polynomial of the transfer function must appear in the denominator of the controller transfer function. This is not the case for either the PI or the PID controllers used, for either disturbance signals. However, it is still desirable that the controller may mitigate the effects of the disturbance to an extent, i.e. the amplitude of oscillations of the output signal be lower than that of the disturbance.

Results:

Simulation results for Y1:



Simulation results for Y2:



The disturbance signals, shown by both the plots, are of amplitude 1 and frequency 0.01 Hz. The output signals in case of the PI controllers appear to share the same frequency as the input disturbance, whereas in the case of the PID controller, both Y1 and Y2 go out of control. Full rejection of disturbance is not observed in any of the cases.

Y1:

The disturbance transfer function for the top product composition loop is as follows:

$$G_{d1}(s) = \frac{-0.61}{8.64s + 1} e^{-3.5s}$$

Upon evaluation of the Frequency Response Function of this signal for an input signal of frequency 0.01 Hz, we note that steady state amplitude ratio (AR) is 0.5361 and the phase angle is 138.90°. These values are obtained using MATLAB's *bode* routine. It is desired that the controllers PI and PID controllers for the top product composition loop reject this disturbance.

We observe the output signal to oscillate with a phase of nearly 180° with respect to the input signal for both the controllers. The output signal amplitude ratio is around 4.5 for the PI controller, whereas it is 4.2 for the PID controller. In both cases therefore, the controller is not effective in rejecting the input disturbance signal, as we know that without the controllers present the output signal would have an amplitude ratio of 0.5361. This increase in amplitude is due to the coupling interaction between the top product composition loop and the bottom temperature loop. The action of the controller in the bottom temperature loop has the side-effect of causing a disturbance in the output of the top product composition loop. Therefore, the output Y1 is of a larger amplitude ratio than without the presence of a controller. A more aggressive controller that takes into account the coupling interaction with the other controller may be required in this case.

Y2:

The disturbance transfer function for the bottom temperature loop is as follows:

$$G_{d2}(s) = \frac{-6.2}{10.9s + 1} e^{-9.4s}$$

Upon evaluation of the Frequency Response Function of this signal for an input signal of frequency 0.01 Hz, we note that steady state amplitude ratio (AR) is 5.1153 and the phase angle is 111.75°. It is desired that the PI and PID controllers for the bottom temperature loop reject this disturbance.

We observe an oscillatory output in both cases. The phase between the disturbance and the output signal is roughly around 180° for both the controllers. For both the controllers, the amplitude ratio between the output and the disturbance is around 2.2. In both cases, the amplitude of disturbance is lower than the amplitude ratio expected in the absence of any control. However, the amplitude ratio is still higher than the amplitude of the disturbance signal. Therefore, disturbance rejection is not completely achieved with either controller setup.

It can be clearly seen that the PI controllers in both the cases are more effective at disturbance rejection than the PID controllers, for both Y1 and Y2. It is still unsatisfactory since the amplitude ratio of the output response is greater than the amplitude ratio of the disturbance.

In order for better performance in disturbance rejection, a more aggressive controller would be required, however, in order to increase the aggression of the controller using the IMC method, the controller time constant would have to be lowered below the minimum value ($\tau_C \geq \theta$). The utilised controller is such that $\tau_C = \theta$. Therefore, a different approach may be required to improve disturbance rejection.

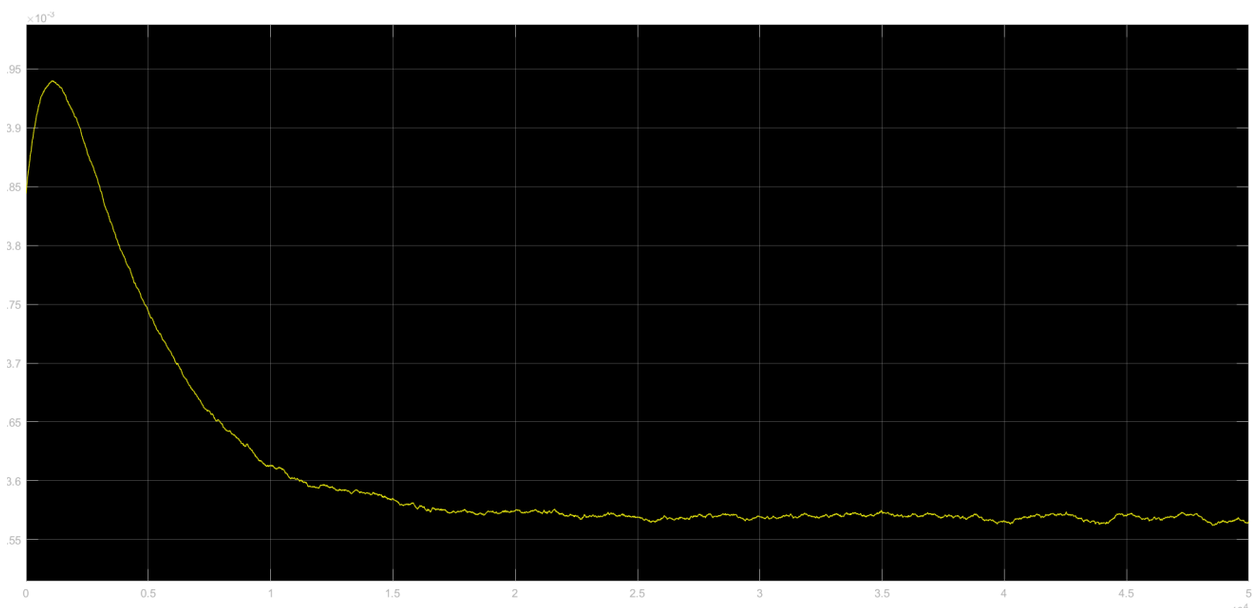
One such possible approach is to additionally implement a feedforward controller capable of rejecting the disturbance, if the disturbance model is known beforehand. However, the implemented feedback controller is still required to be present to ensure that set-point tracking is achieved.

Question 2: FCC System

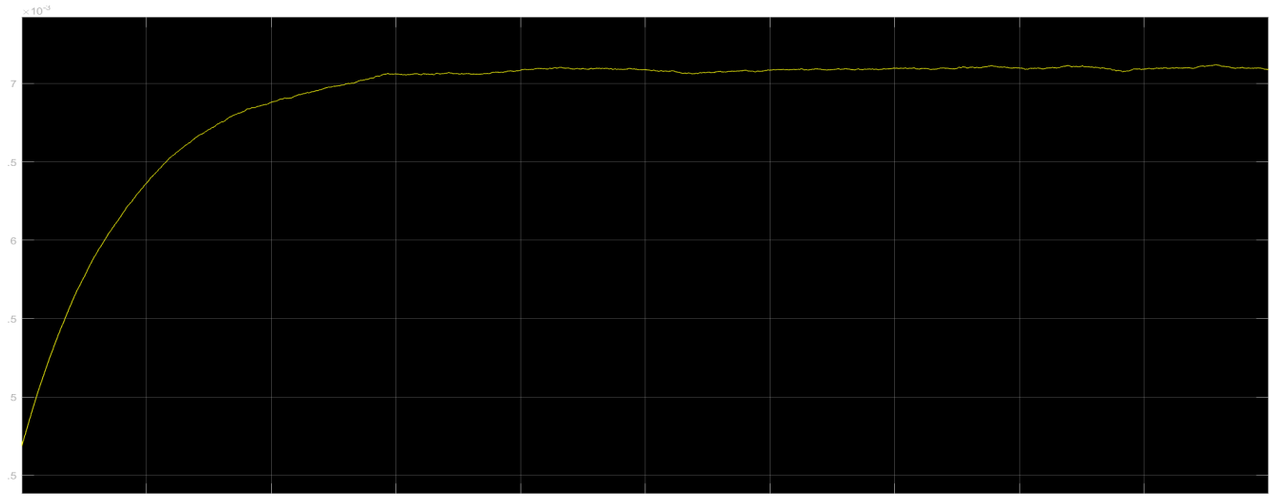
Part a) State-Space Model Estimation

The given SIMULINK model is run and the outputs are plot.

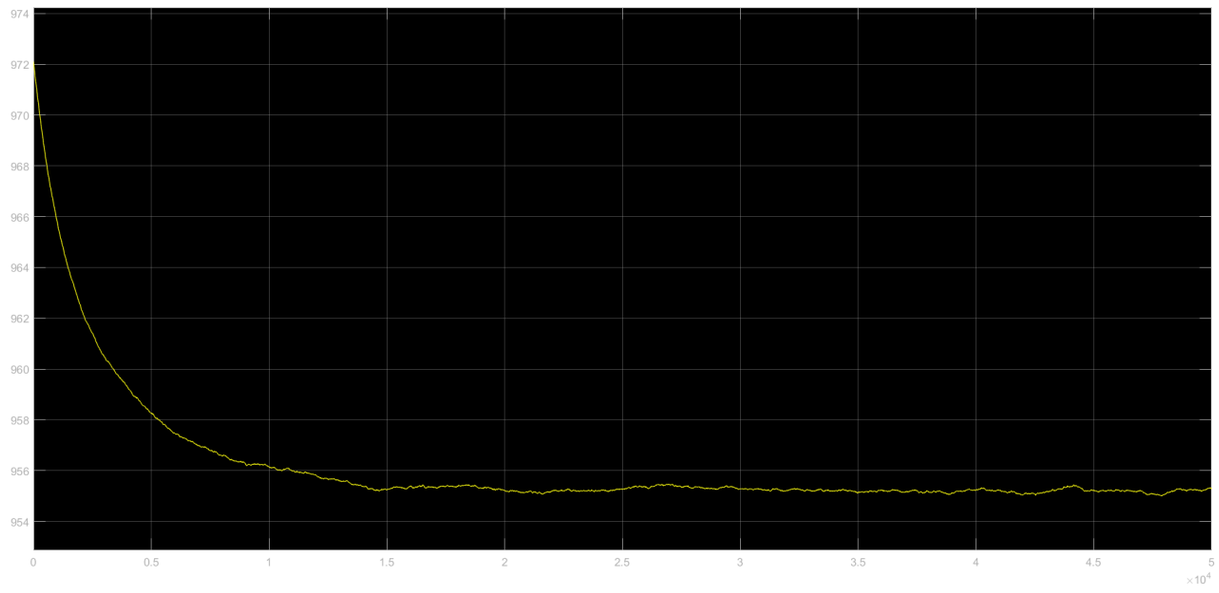
Actual Y1:



Actual Y2:



Actual Y3:



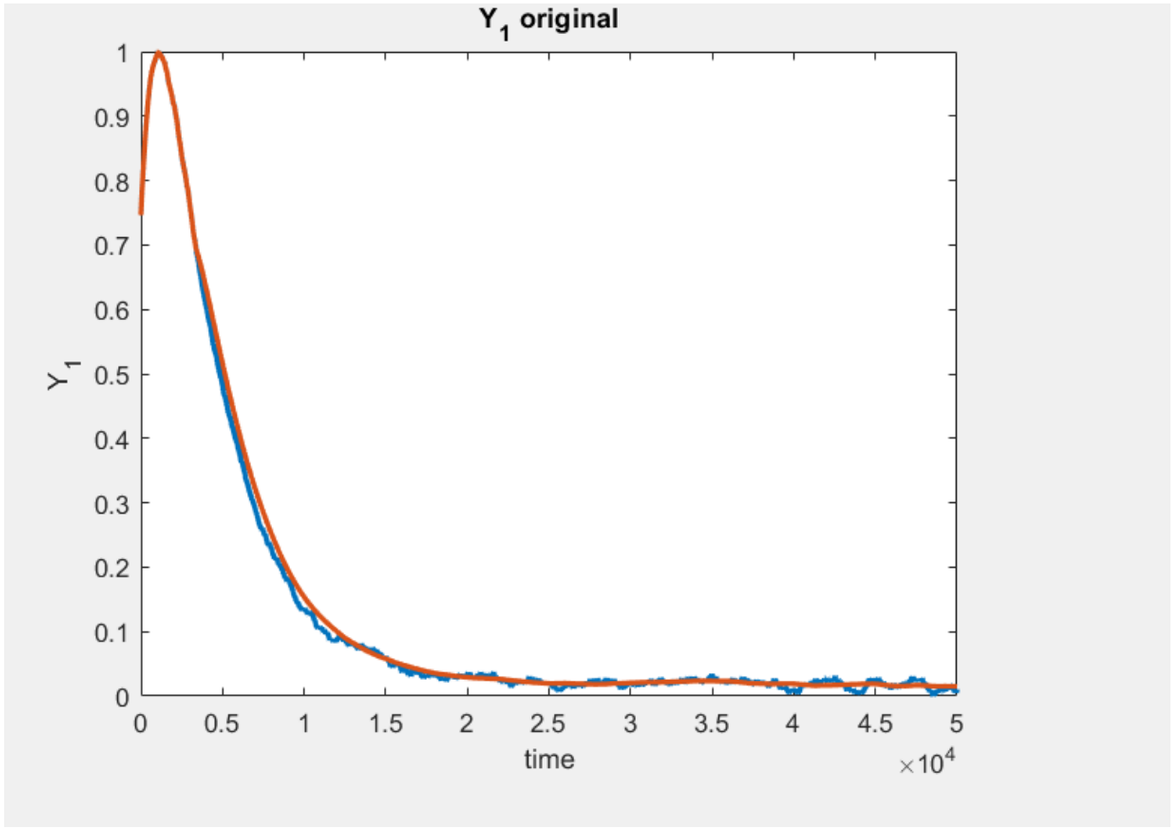
Two observations:

1. Scale of Y1, Y2, Y3 are not comparable.
2. All the 3 outputs have some noise in the end. (ie they seem to be moving up and down randomly near the steady state)

Solutions:

1. Rescale the outputs: I did min-max scaling to ensure all the output falls between 0 and 1. We can retrieve the original data by reversing this operation. (Will be useful when we finally want to implement our model for some purpose)
2. I tried using smoothdata to smooth Y1. It uses moving average to smooth the data. **Note that we want smoothing only in the end, where we are close to steady state.** So, in the initial parts smoothing might be unnecessary and might tone down the dynamics. Accordingly, I used smoothed values only in the end where it seems to settle at some point. Code for the same:

   ```
   dummy = smoothdata(Y1);

   dummy(1:13300)=Y1(1:13300);

   Y1 = dummy;
   ```



Orange curve is the smoothed version and blue curve is the original

This smoothing can be repeated for other variables but:

1. It proved to be cumbersome to decide the point from which I should perform the smoothing
2. It yielded no real improvement in the fit to estimate given by n4sid

The data was packed using iddata and n4sid was used to estimate the model

[sys,x0] = n4sid(data,Nx,'DisturbanceModel','none','Feedthrough',true(1,Nu),'Ts',0);

points to note:

1. **Disturbance model:** Set to None; Prevented estimation of term K given in the n4sid state-space model
2. **Feedthrough:** Allowed it to estimate the feedthrough term (D)

sys =

Continuous-time identified state-space model:

dx/dt = A x(t) + B u(t) + K e(t)

y(t) = C x(t) + D u(t) + e(t)

A =

	x1	x2	x3	x4	x5
x1	-1.224	-0.01398	-0.005502	-0.001946	0.05773
x2	1.066	0.00671	0.001509	0.003406	-0.0605
x3	0.02668	0.01404	0.007254	0.003808	-0.004923
x4	-0.2646	-0.3052	-0.1836	0.1033	-0.583
x5	-0.6563	-1.002	-0.8481	2.076	-8.322

B =

	u1	u2
x1	-0.01528	0.002231
x2	0.00924	-0.001713
x3	-0.0005101	2.933e-06
x4	-0.2381	0.01372
x5	-3.187	0.1818

C =

	x1	x2	x3	x4	x5
y1	-9.365	-11.48	60.63	4.383	-0.3264
y2	-64.6	-73.28	-51.48	-1.986	0.254
y3	-4.659	-5.884	43.82	2.95	-0.222

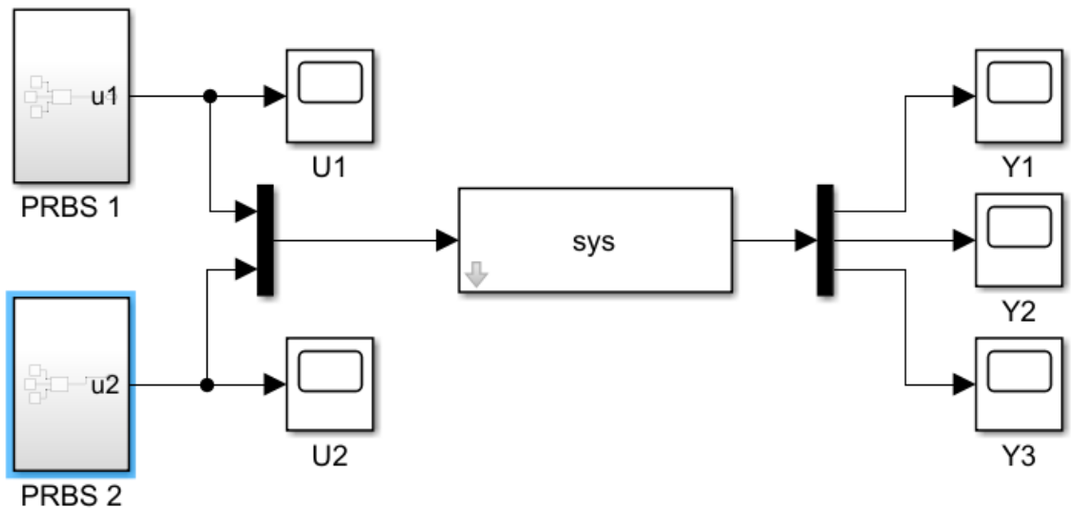
D =

	u1	u2
y1	-0.0004406	-3.563e-06
y2	-7.755e-06	3.342e-05
y3	1.967e-05	5.013e-05

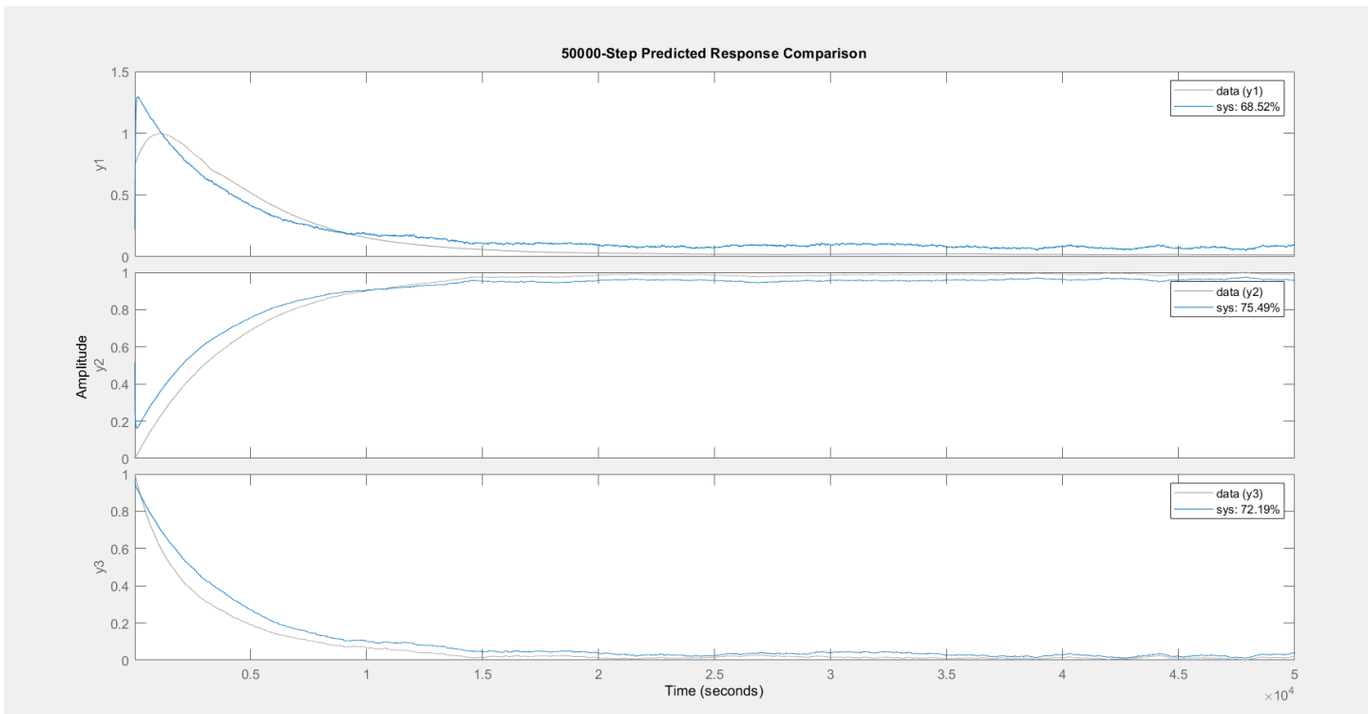
Fit to estimation data: [74.2;80.76;75.65]%

FPE: 5.879e-10, MSE: 0.006449

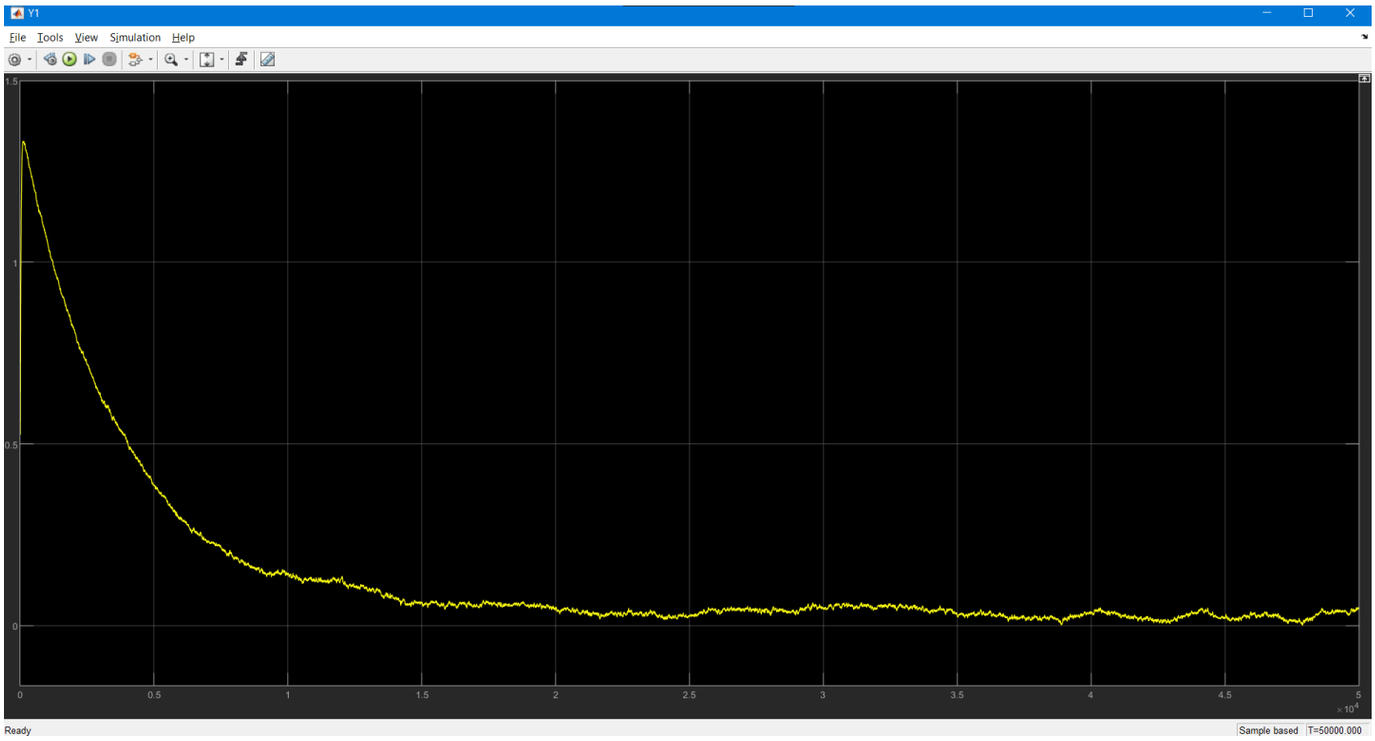
Since seed value has been given for the random number block in the prbs input, we use the same prbs input block to obtain output to verify our model.



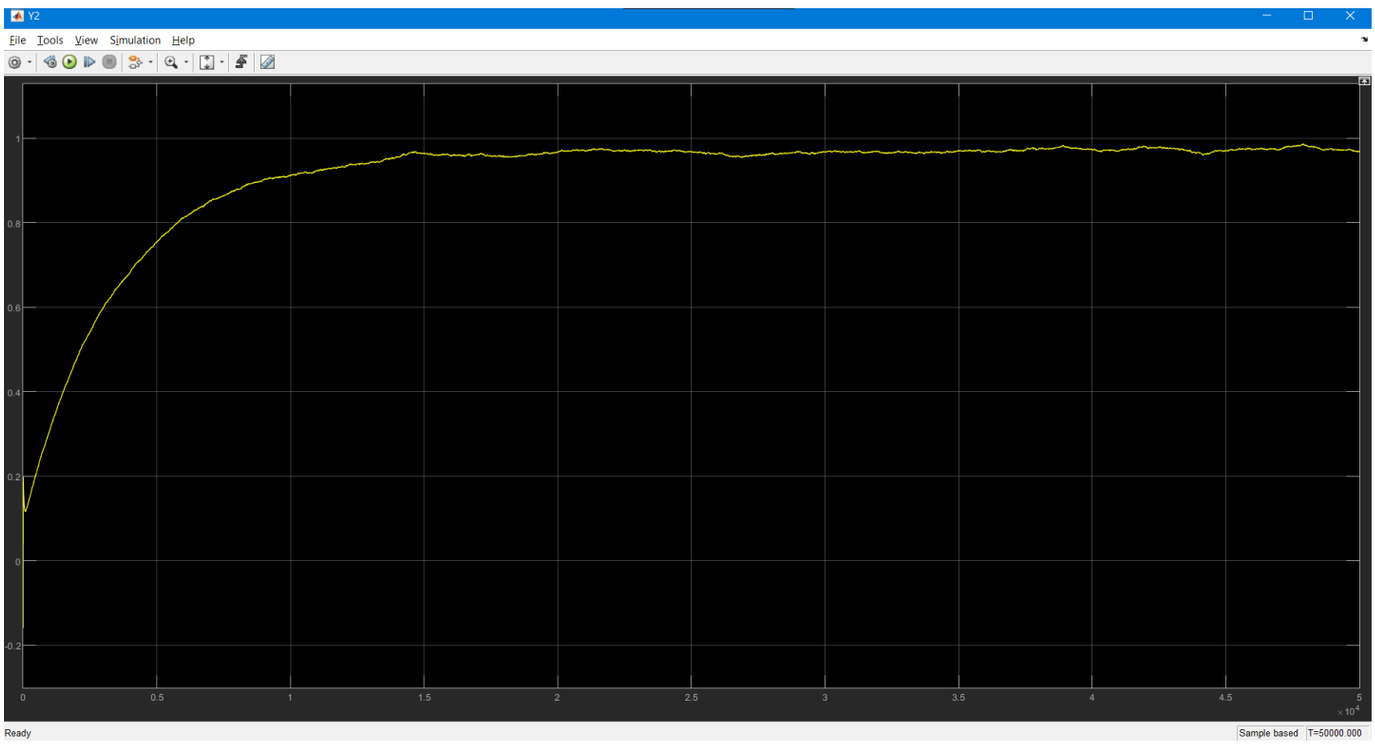
NOTE THAT THE SYSTEM STARTS AT NON-ZERO INITIAL STATE. The initial state obtained from n4sid was supplied here.



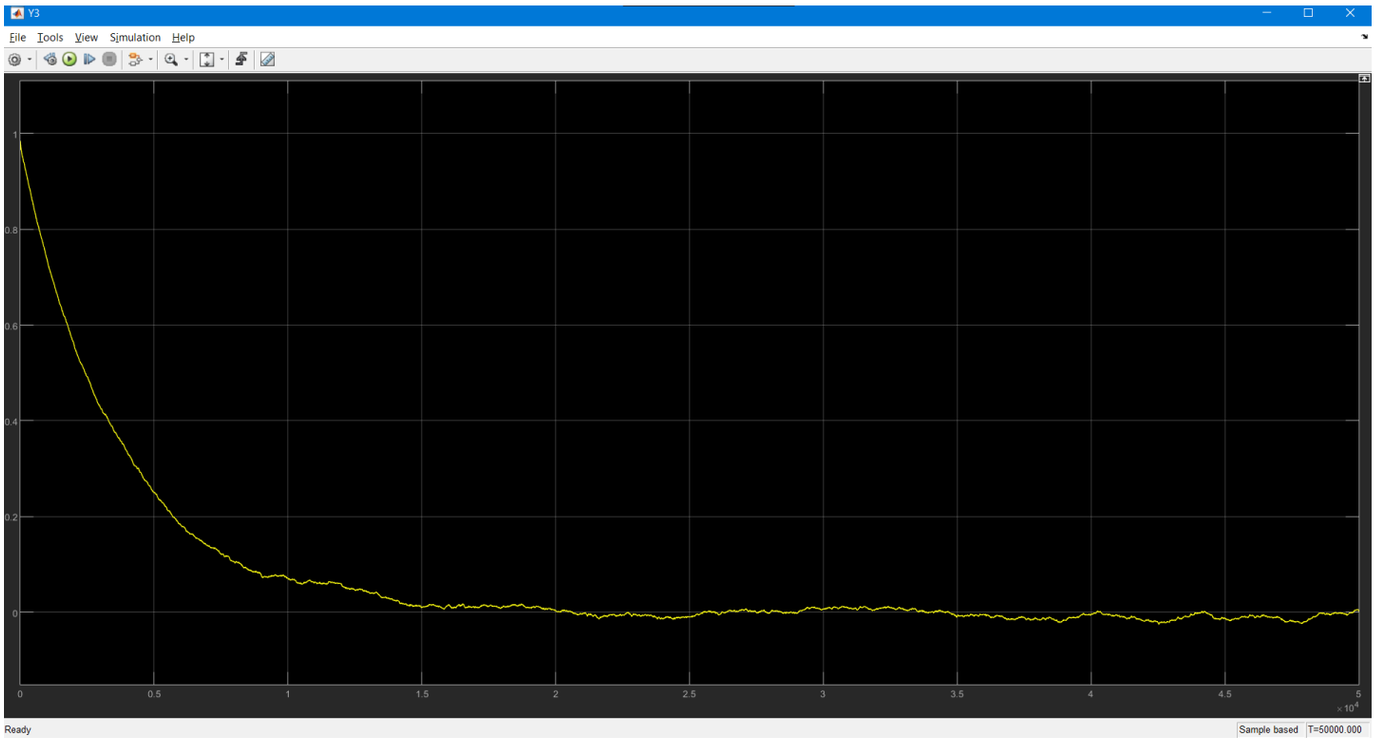
Output comparison using compare



Y1: from estimated model simulation in simulink



Y2: from estimated model simulation in SIMULINK



Y3: simulation of estimated model in SIMULINK

The model outputs match well with the actual output!!

Part b) Examine Open Loop Stability

poles (found using eigenvalues of A):

poles =

-8.1792

-1.2099

-0.0395

-0.0003

-0.0000

All real parts of the poles are negatively valued! This implies that the system is **open loop stable.**

It is further confirmed using isstable(sys) that the system is, indeed, stable.

Part c) Convert to tf, SIMULINK block, verification

Continuous time transfer functions are estimated by simply calling tf(sys) where sys is the estimated state space model.

tfs =

From input "u1" to output...

-0.0004406 s^5 - 0.001461 s^4 - 0.02655 s^3 - 0.031 s^2 - 2.729e-05 s + 7.674e-08

y1: ---

s^5 + 9.429 s^4 + 10.27 s^3 + 0.3935 s^2 + 0.0001121 s + 8.015e-11

-7.755e-06 s^5 - 0.0003997 s^4 - 0.007314 s^3 + 0.007126 s^2 + 0.0001251 s - 1.699e-08

y2: ---

s^5 + 9.429 s^4 + 10.27 s^3 + 0.3935 s^2 + 0.0001121 s + 8.015e-11

1.967e-05 s^5 - 0.0001493 s^4 - 0.0002206 s^3 - 0.0001176 s^2 - 1.803e-05 s + 5.41e-08

y3: ---

s^5 + 9.429 s^4 + 10.27 s^3 + 0.3935 s^2 + 0.0001121 s + 8.015e-11

From input "u2" to output...

-3.563e-06 s^5 - 0.0002739 s^4 + 0.0004014 s^3 + 0.0008627 s^2 - 4.356e-05 s - 8.062e-09

y1: ---

s^5 + 9.429 s^4 + 10.27 s^3 + 0.3935 s^2 + 0.0001121 s + 8.015e-11

3.342e-05 s^5 + 0.0004509 s^4 + 0.001786 s^3 + 0.0001997 s^2 + 1.63e-05 s + 1.799e-09

y2: ---

s^5 + 9.429 s^4 + 10.27 s^3 + 0.3935 s^2 + 0.0001121 s + 8.015e-11

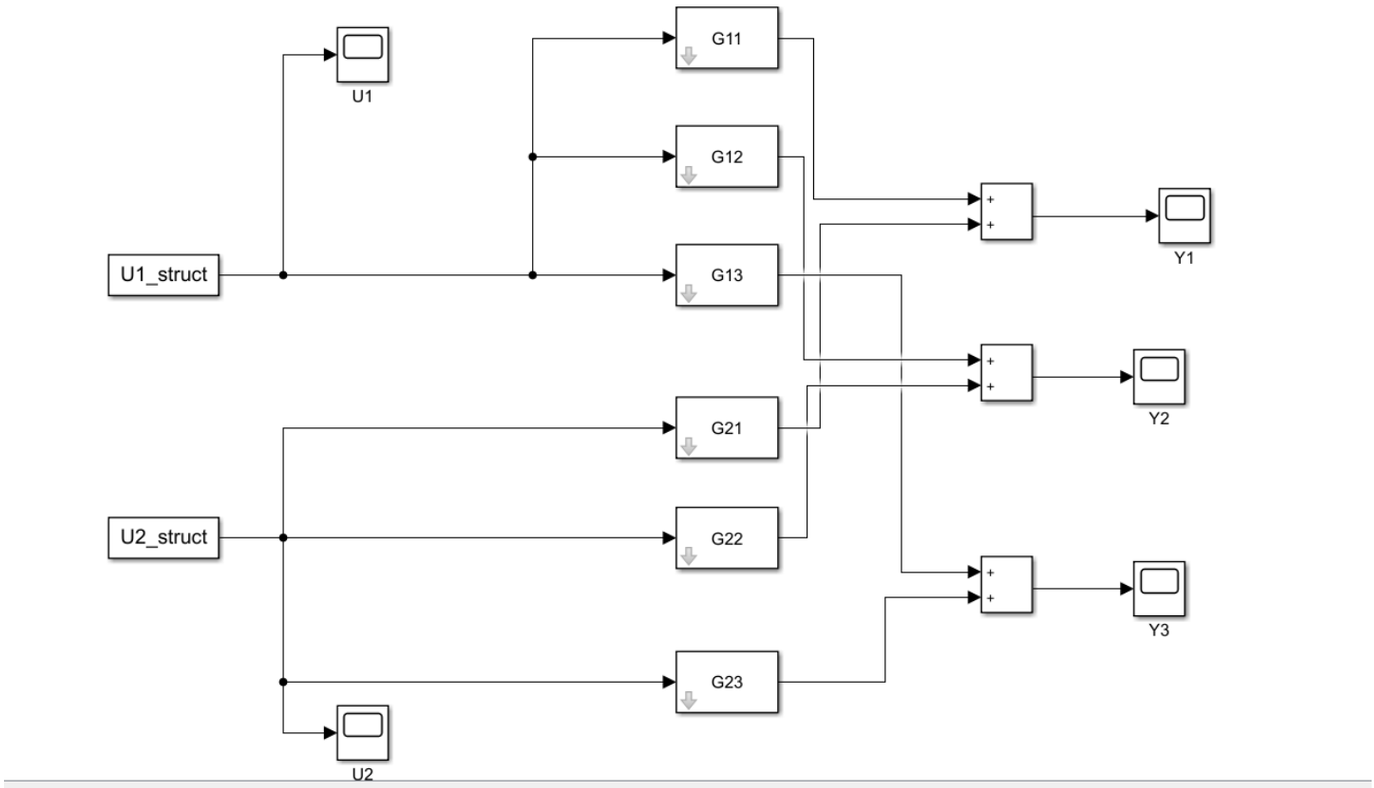
5.013e-05 s^5 + 0.0004081 s^4 - 0.0002488 s^3 - 0.0008277 s^2 - 3.139e-05 s - 5.684e-09

y3: ---

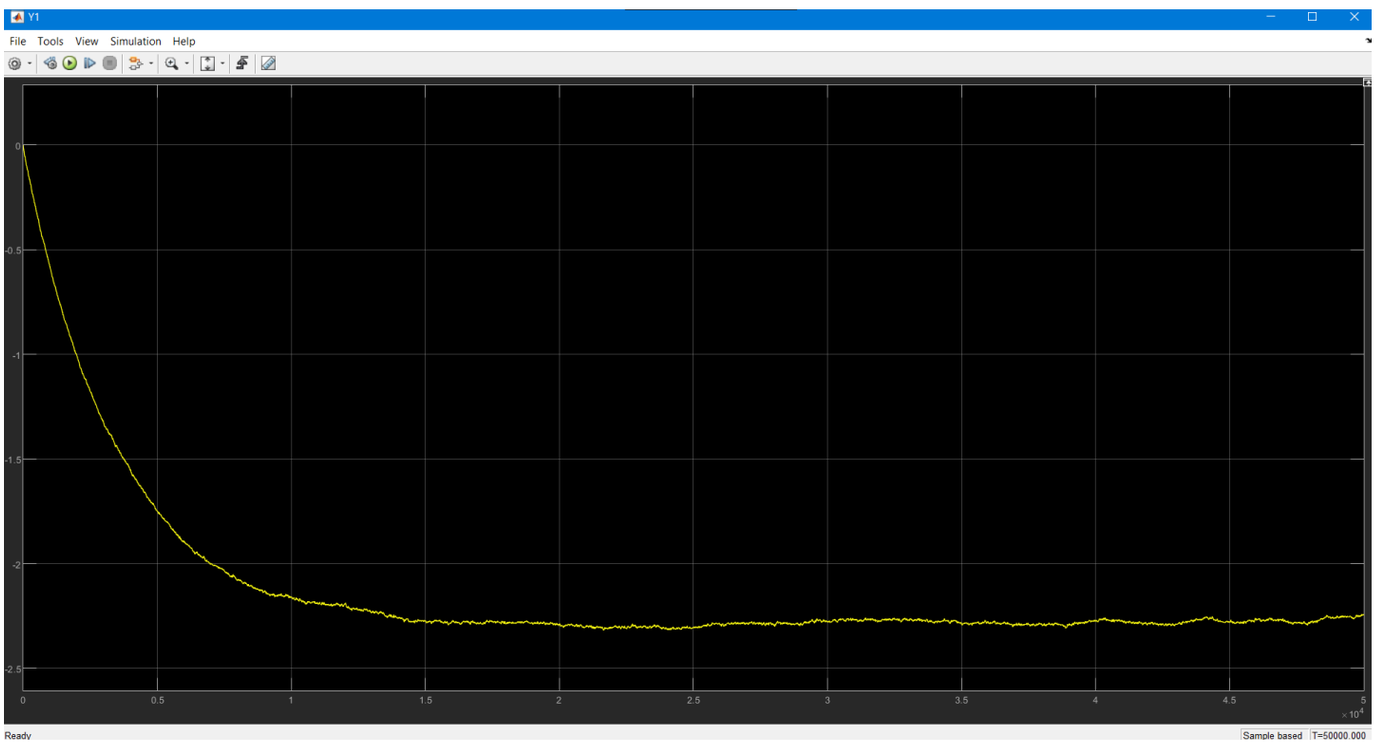
s^5 + 9.429 s^4 + 10.27 s^3 + 0.3935 s^2 + 0.0001121 s + 8.015e-11

Continuous-time transfer function.

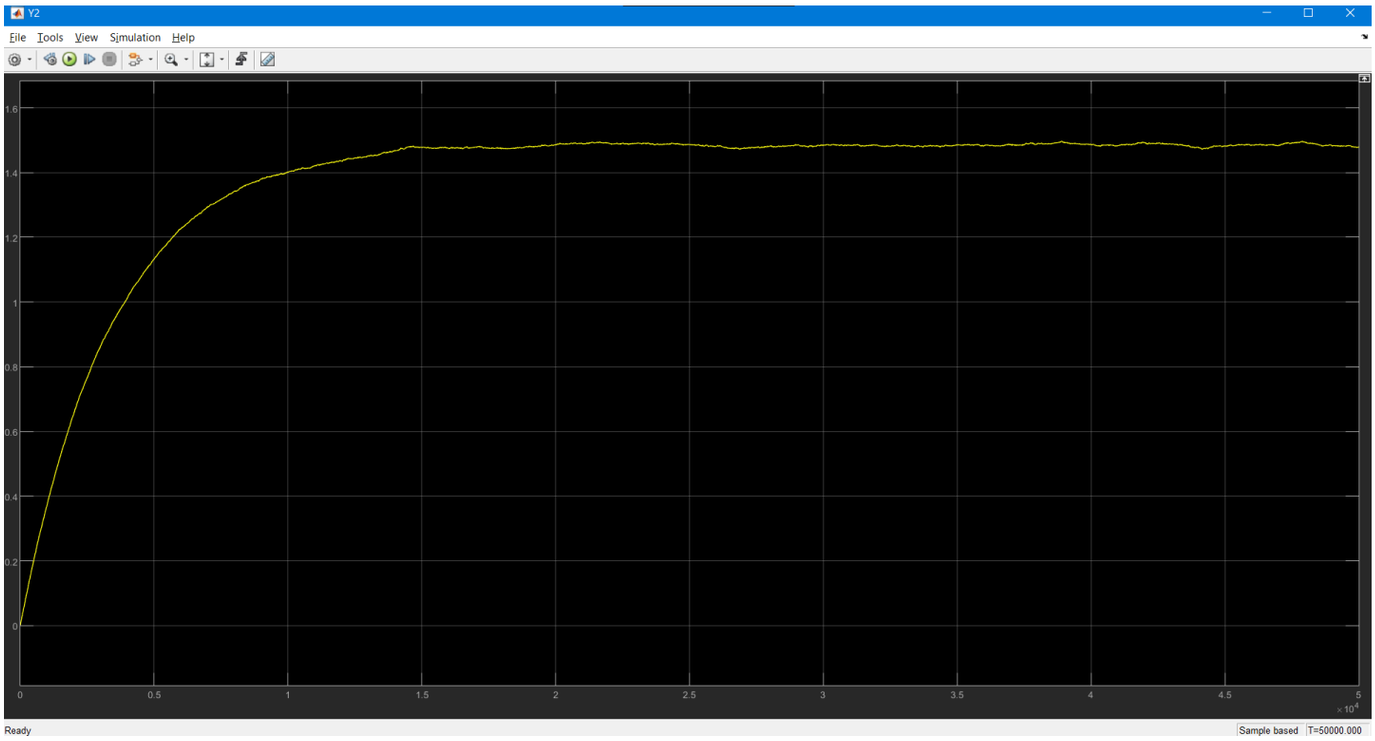
One can build a SIMULINK block diagram using the transfer functions as follows. **BUT BECAUSE OF NON ZERO INITIAL STATE, THE OUTPUTS WON'T MATCH WITH THOSE GIVEN BY THE INITIAL SYSTEM. IT IS AN ASSUMPTION IN TRANSFER FUNCTION MODELS THAT THE SYSTEM STARTS AT A ZERO iNITIAL STATE.**



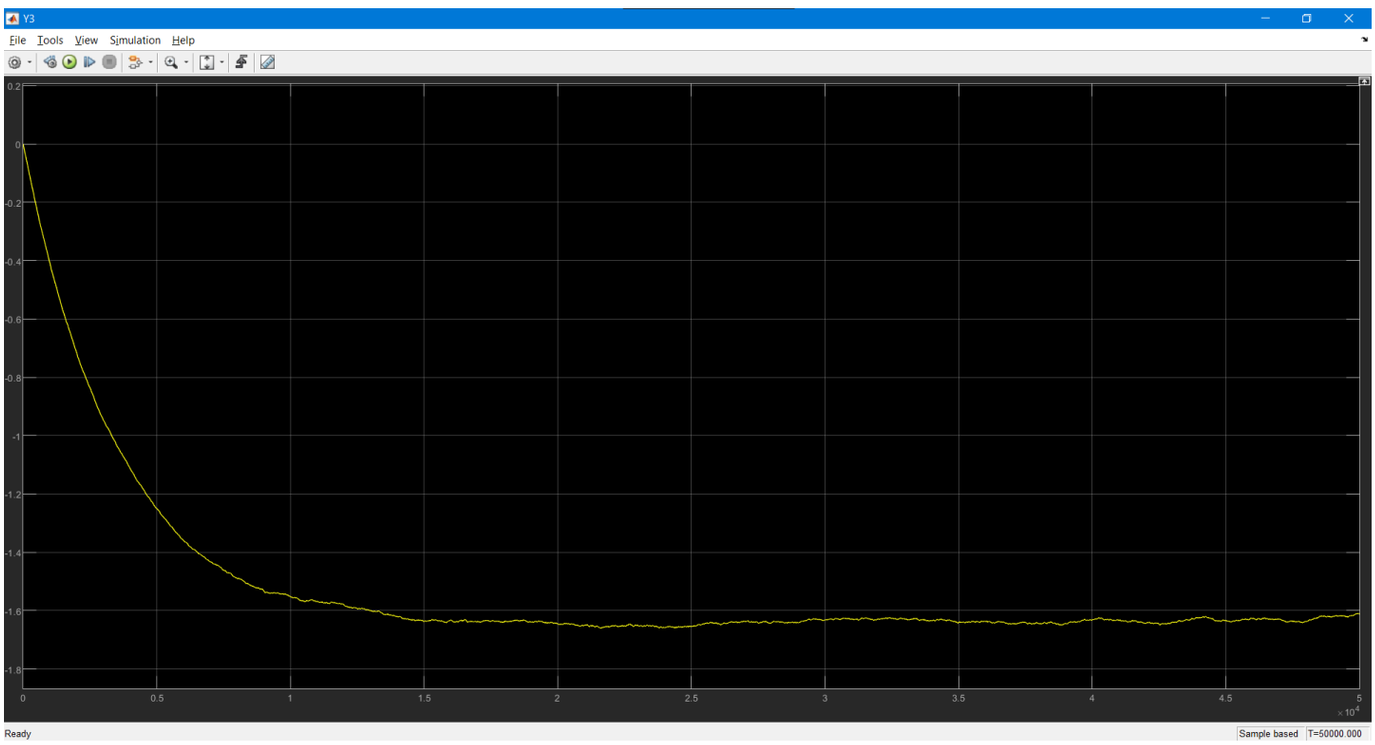
U1_ Struct and U2_Struct are just the prbs inputs saved in appropriate structure format



Y1: GOING TO NEGATIVE VALUES (-2)



Y2: CROSSES 1.5 (should lie between 0 and 1)



Y3: LIKE Y1, GOES AND SETTLES IN NEGATIVE VALUES

As expected they did not match with the original outputs.

To circumvent this, we can recast the given equations in terms of the deviation variables.

$d(x-x0)/dt = A(x-x0) + B(u-u0) + Ax0 + Bu0$

$y = C(x-x0) + D(u-u0) + Cx0 + Du0 = C(x-x0) + D(u-u0) + y0$

But $u0=0$

=>

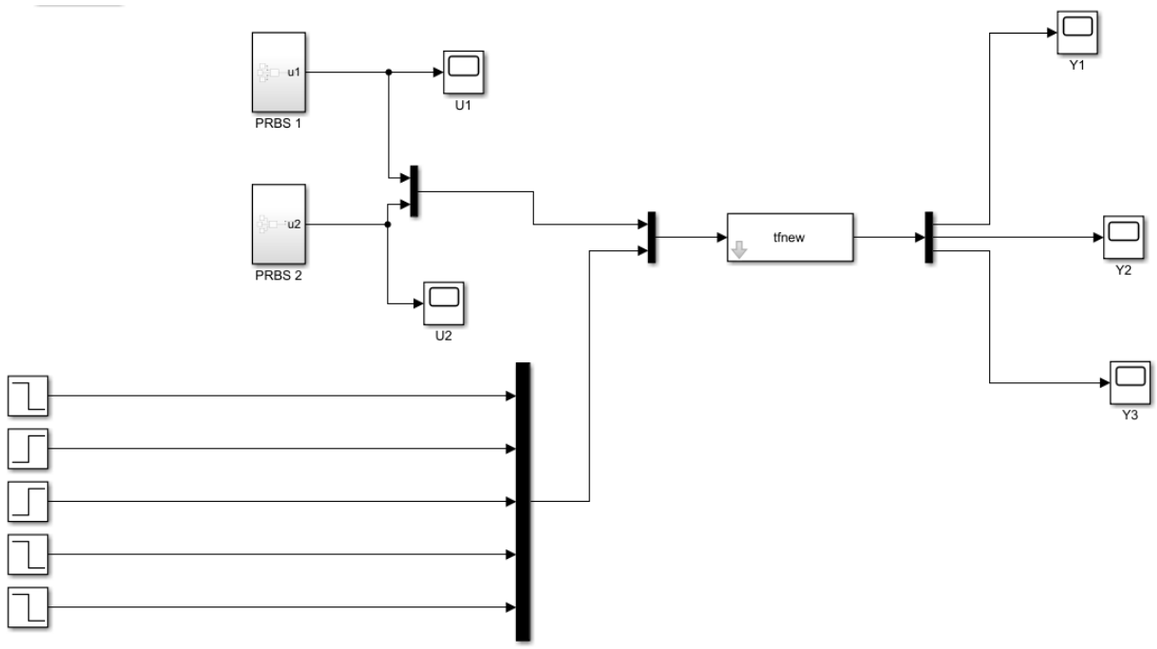
dxhat/dt = Axhat + Buhat + **Ax0**

yhat = Cxhat + Duhat

The Ax0 term can be treated as a step disturbance, with the step value given at t=0 and its value being equal to that of x0.

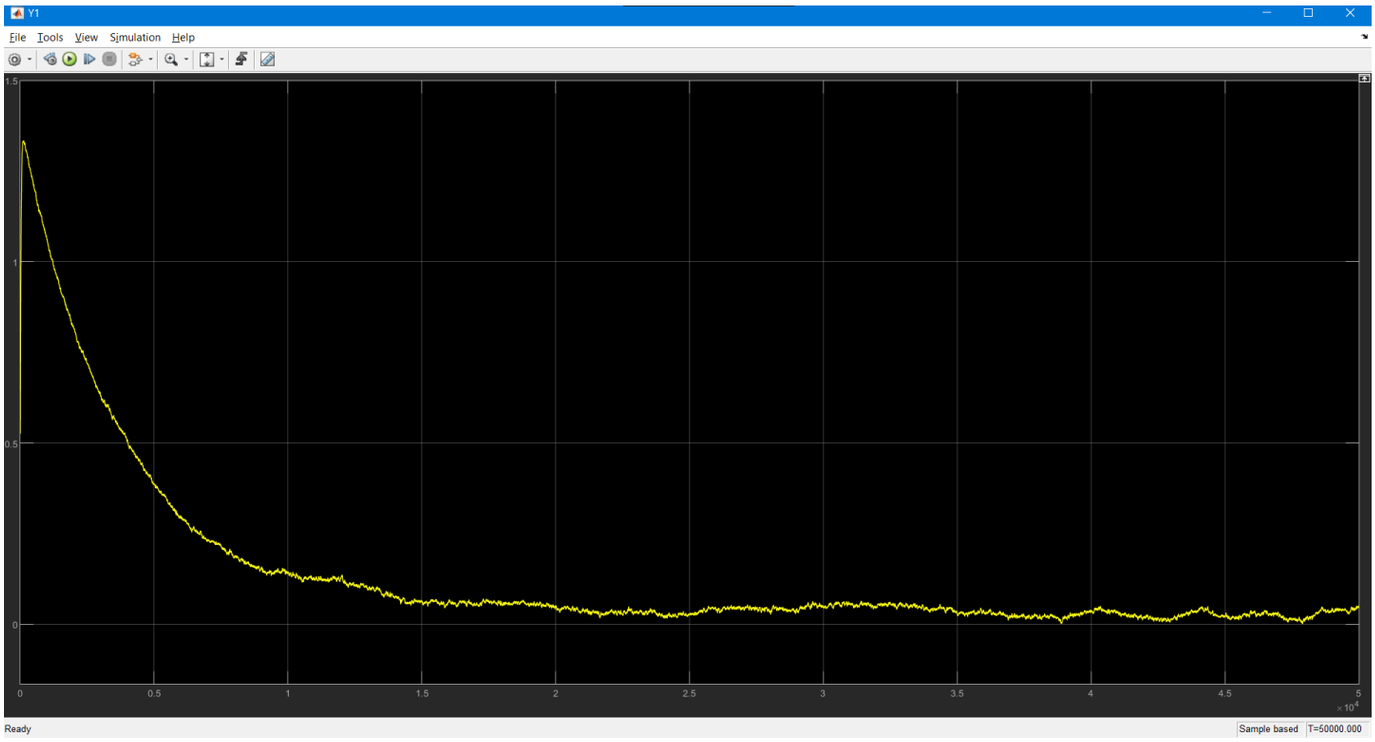
```
stability = isstable(sys)
%% Intermission: Need to account for initial state
Bnew = zeros(Nx,Nu+Nx);
Bnew(:,1:2) = sys.B;
Bnew(:,3:end) = sys.A;
Dnew = zeros(Ny,Nx+Nu);
Dnew(:,1:2) = sys.D;
Dnew(:,3:end) = sys.C;
ss_again = ss(sys.A,Bnew,sys.C,Dnew);
tfnew = tf(ss_again);
```

The disturbance terms are accommodated in B and D, and a new transfer function model is built. This will have 5 additional inputs (representing the 5 states). So total number of inputs = 5+ 2 =7. And you will have 7*3 = 21 transfer functions in total.

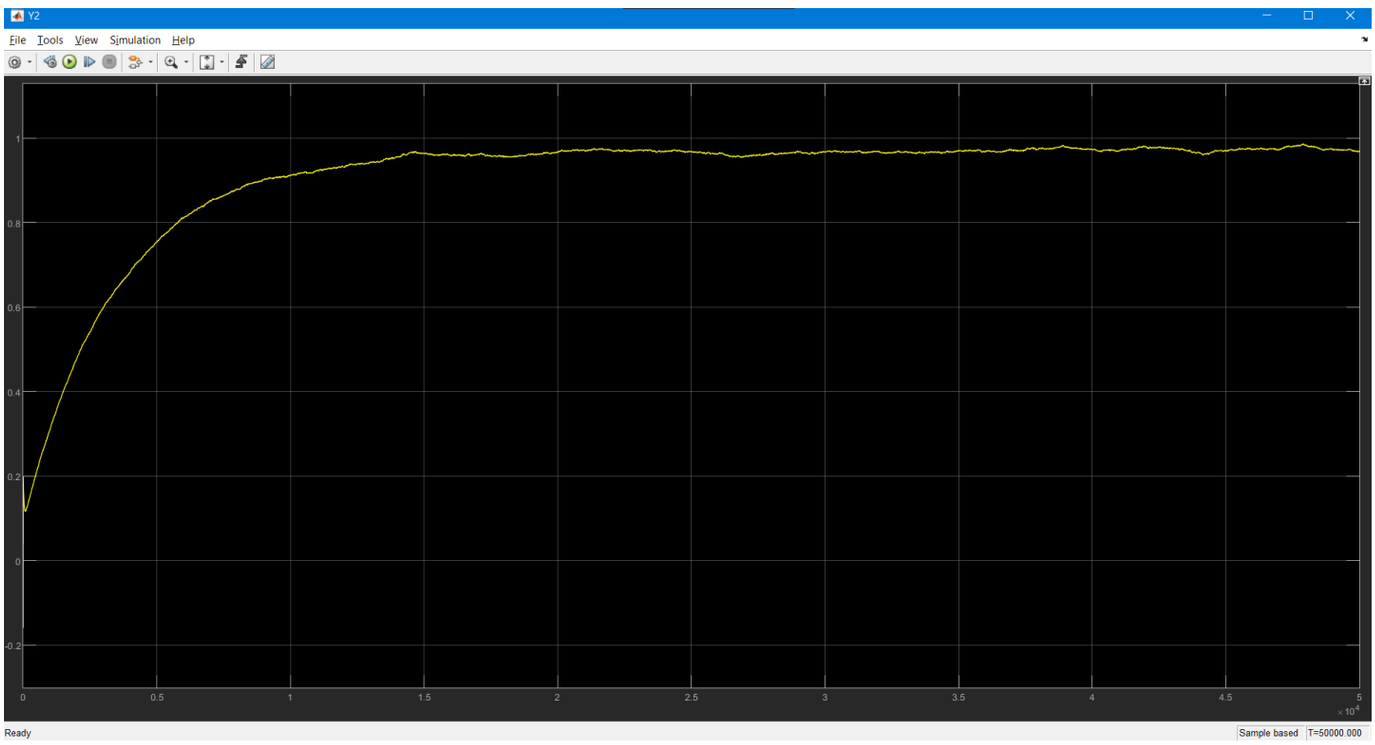


SIMULINK representation. The step inputs represent the initial states

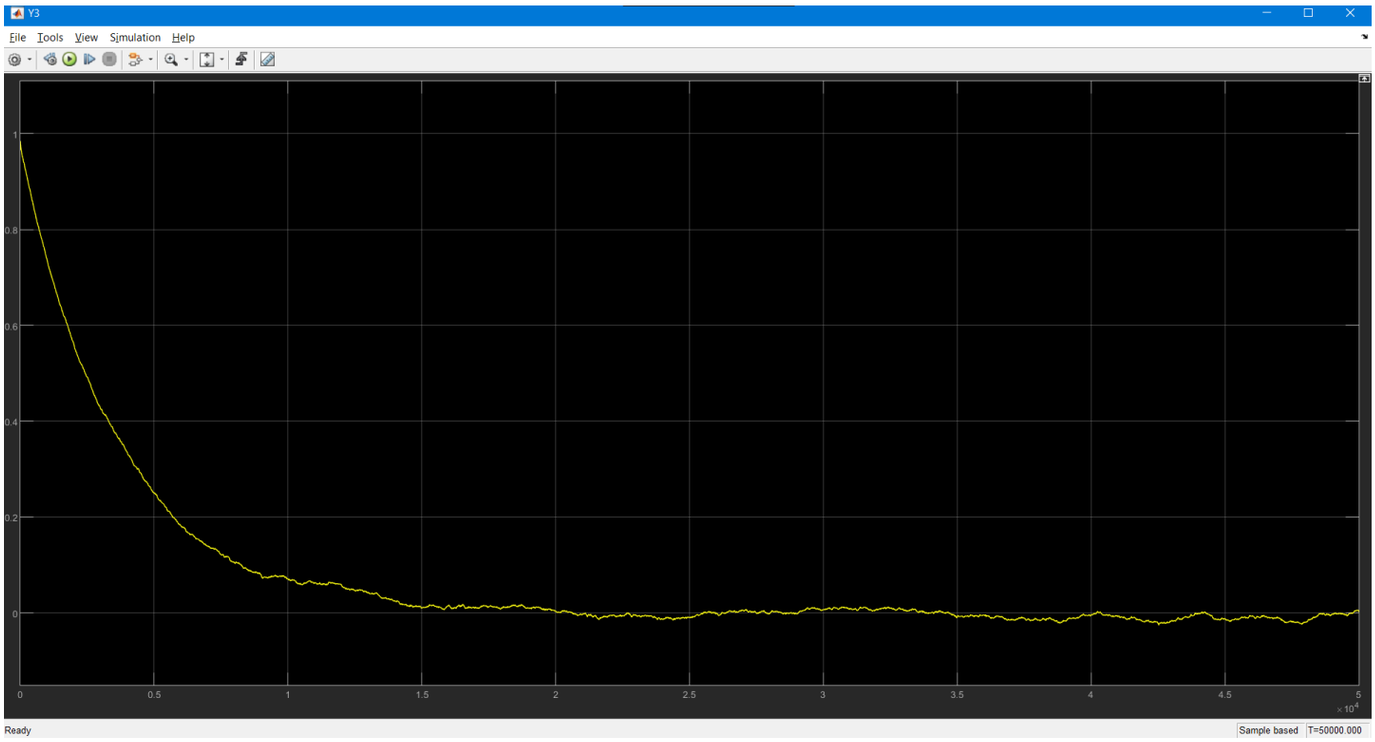
This model gives an output that resembles the output of the original system.



Y1: notice it has the initial rise as in the original output



Y2

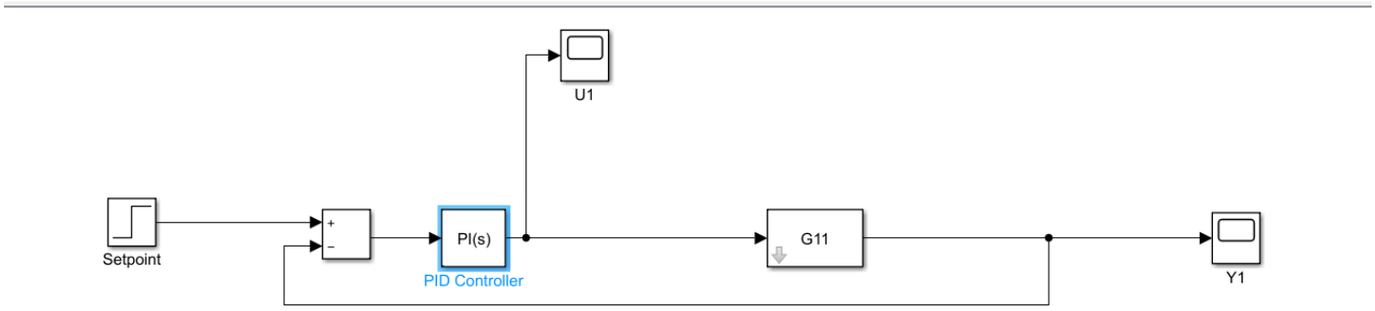


Y3

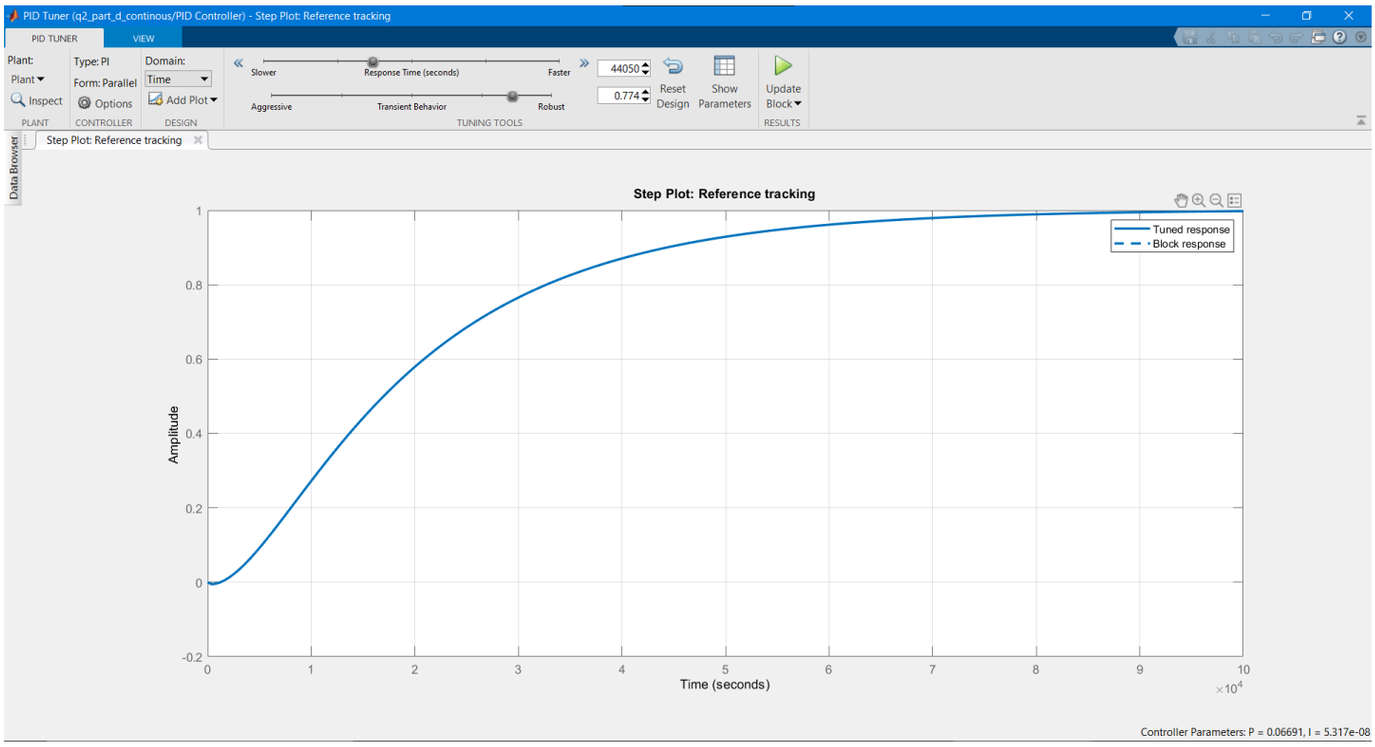
Much better! All are close to the original range of 0-1 and resemble the simulated output from the given model.

Part d) Controller tuning and Closed Loop stability

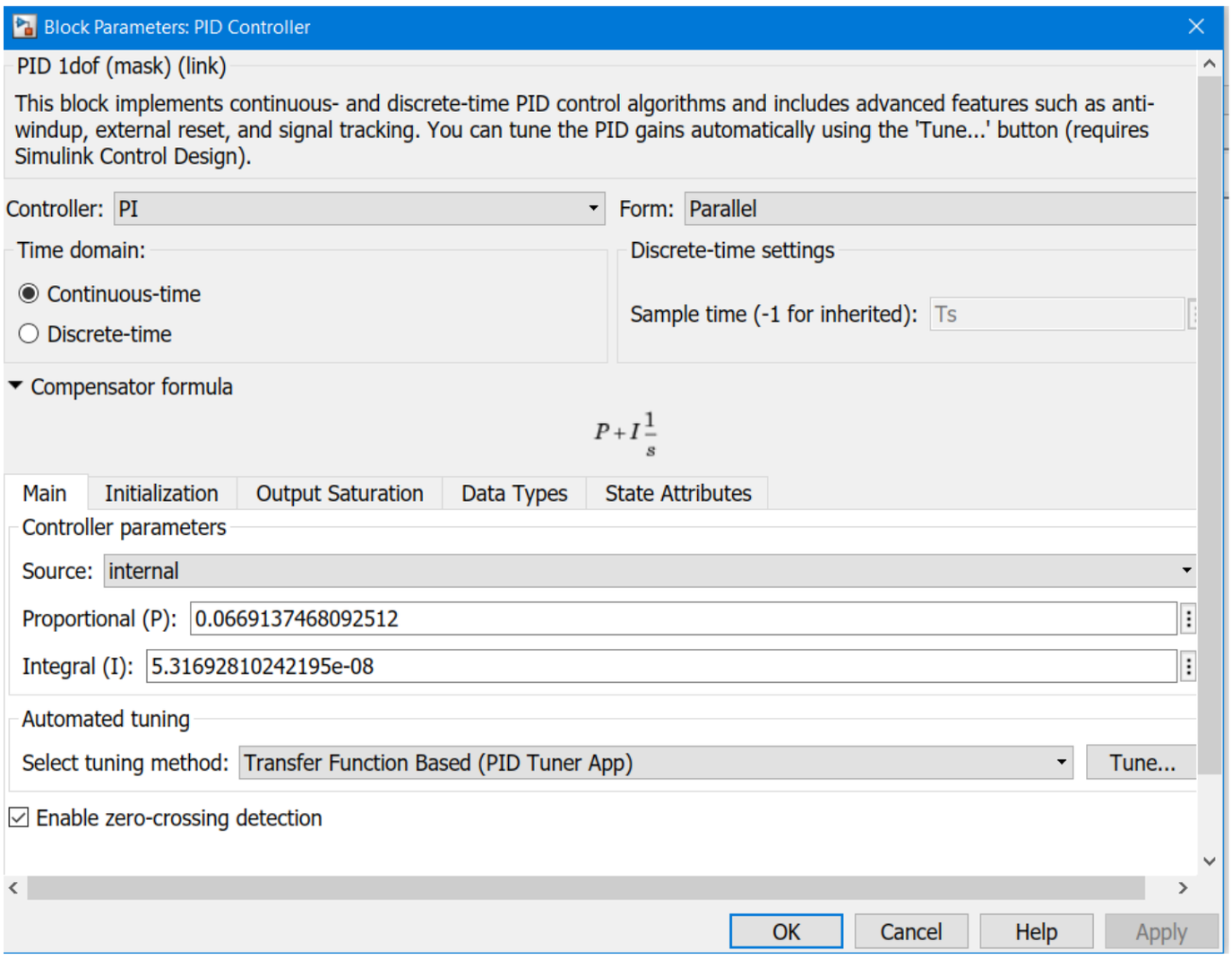
We simply use G11 alone so that we can operate it as SISO system with U1 as input and Y1 as output



PI tuning



Tuned using auto-tuner



<u>Criteria:</u>

1. Response time (slow vs fast):

   We can choose between a fast response and a slow response. A fast response might result in overshoots and oscillations, which I have tried to avoid here. I have opted for a **slower** response time.

2. Transient behaviour (Aggressive vs Robust):

   Aggressive behaviour means that the initial inputs would be very high so that we rapidly reach the steady state. Here, I have opted for a more **robust response**. This generally requires less control efforts and the output won't oscillatory behaviour.

Routh Hurwitz criterion

Denominator polynomial computed from MATLAB:

s^6 + 9.429 s^5 + 10.27 s^4 + 0.3915 s^3 + 0.0001103 s^2 + 5.213e-09 s + 4.08e-15

Code:

```
%% Initialisation
Kc = 0.0669137468092512;
KI = 5.31692810242195e-08;
Gm = G11;
Gc = tf([Kc KI],[1 0]);
s = tf('s');
%% RH criterion
Dr = 1 + G11*Gc;
[num,~] = tfdata(Dr,'v');
tf(num,1)
```

Routh Hourwitz table computed from code given in MATLAB file exchange.

0.999971	10.26783	0.00011	4.08E-15
9.428765	0.391456	5.21E-09	0
10.22632	0.00011	4.08E-15	0
0.391354	5.21E-09	0	0
0.00011	4.08E-15	0	0
5.20E-09	0	0	0
4.08E-15	0	0	0

We see no sign changes!

=> No RHP poles

So, the system is stable.

Root Locus analysis.

For root locus analysis we cast the denominator of the form 1 + beta*L where beta is the parameter of interest. In this case, we are focusing on **KI**

Gc: Gcontroller and Gm: Gmodel

Denominator; 1 + GcGm = 0

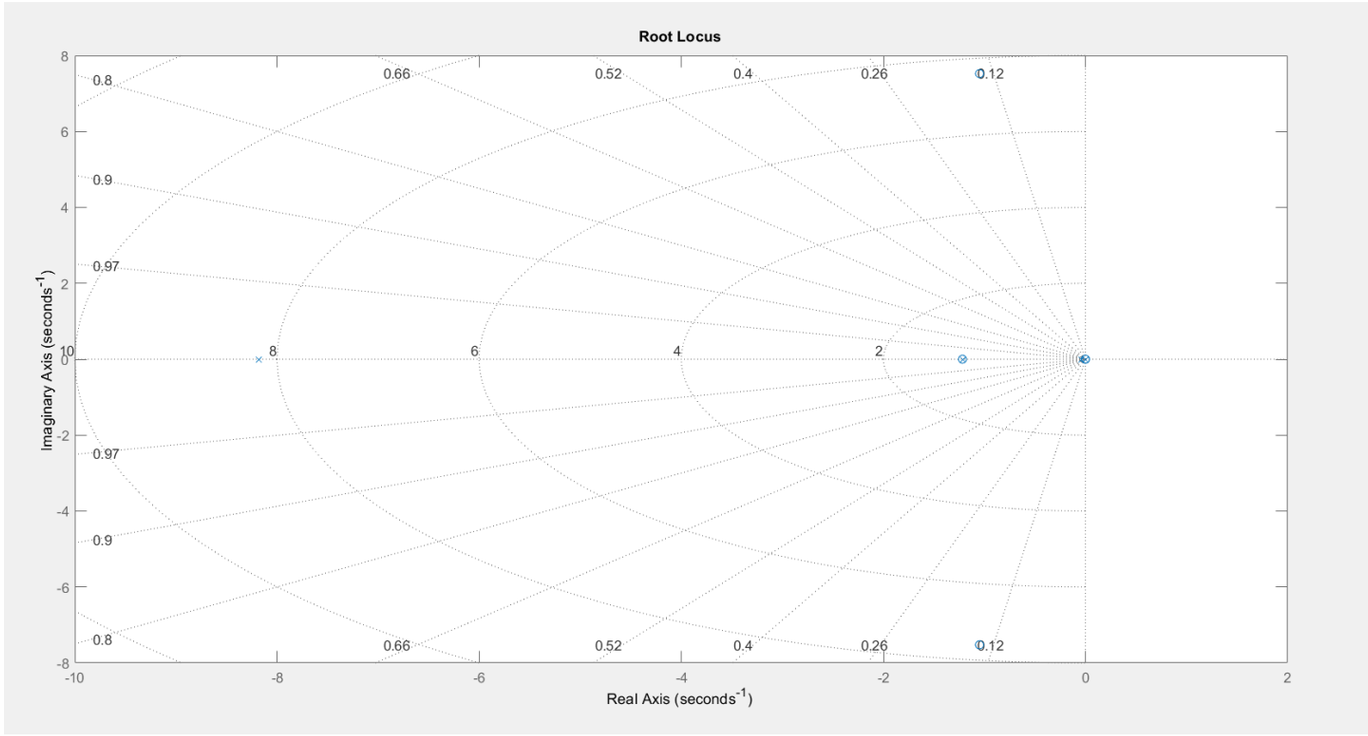
=> 1 + (Kp + KI/s)Gm = 0

=> 1 + KI*(Gm/(s*(1+Kc*Gm)) = 0

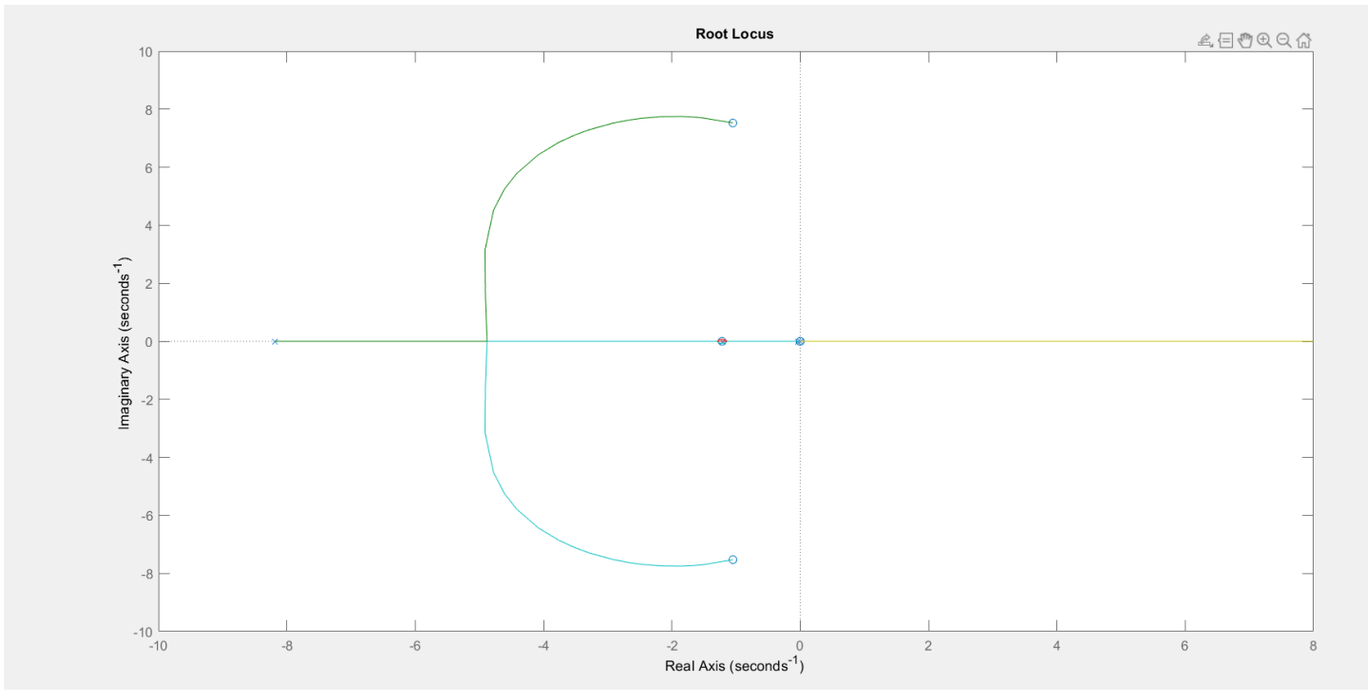
So L = Gm/(s*(1+Kc*Gm)

Used minreal to cancel out common factors (pole-zero cancellation)

```
%% Root locus
L = minreal(Gm/(s*(1+Kc*Gm)));
rlocusplot(L,KI);
grid on;
figure;
rlocusplot(L)
```

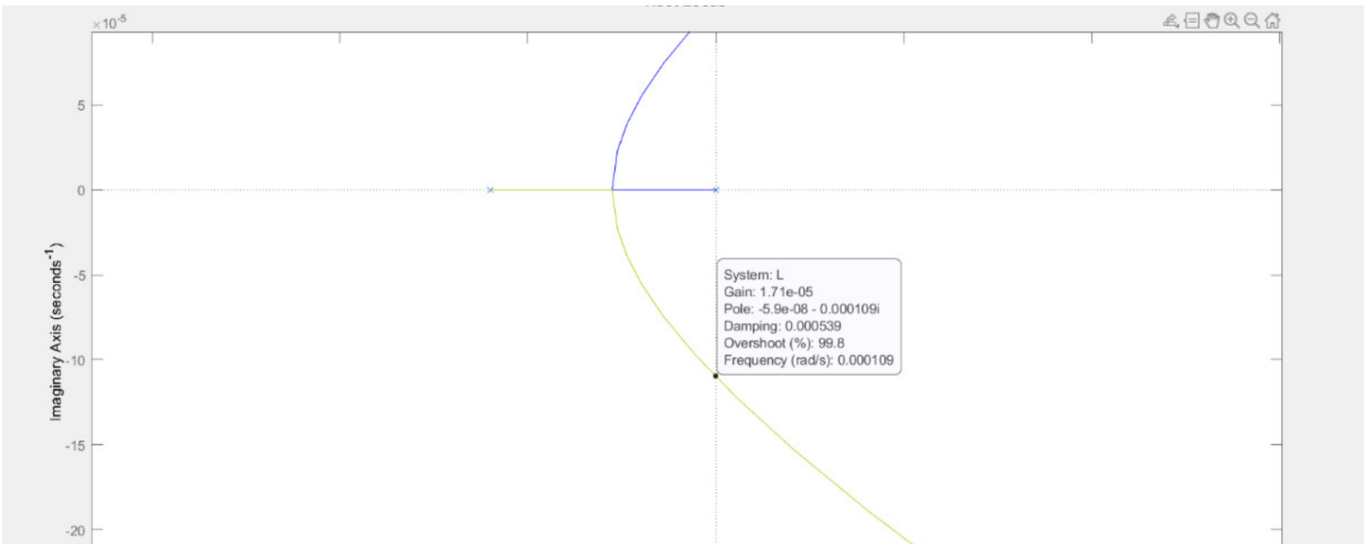


Poles (x) and Zeros (o) for given KI. We find that all Poles lie in LHP with some of them at origin. No RHP poles! So the system is stable



Rootlocus plot

Zooming the root locus plot multiple times near the origin:



We find that **ultimate gain** (gain for which the Root Locus cuts the y axis) is **1.71e-5**. Our gain is 5.3169e-08.

KI < KI,ultimate

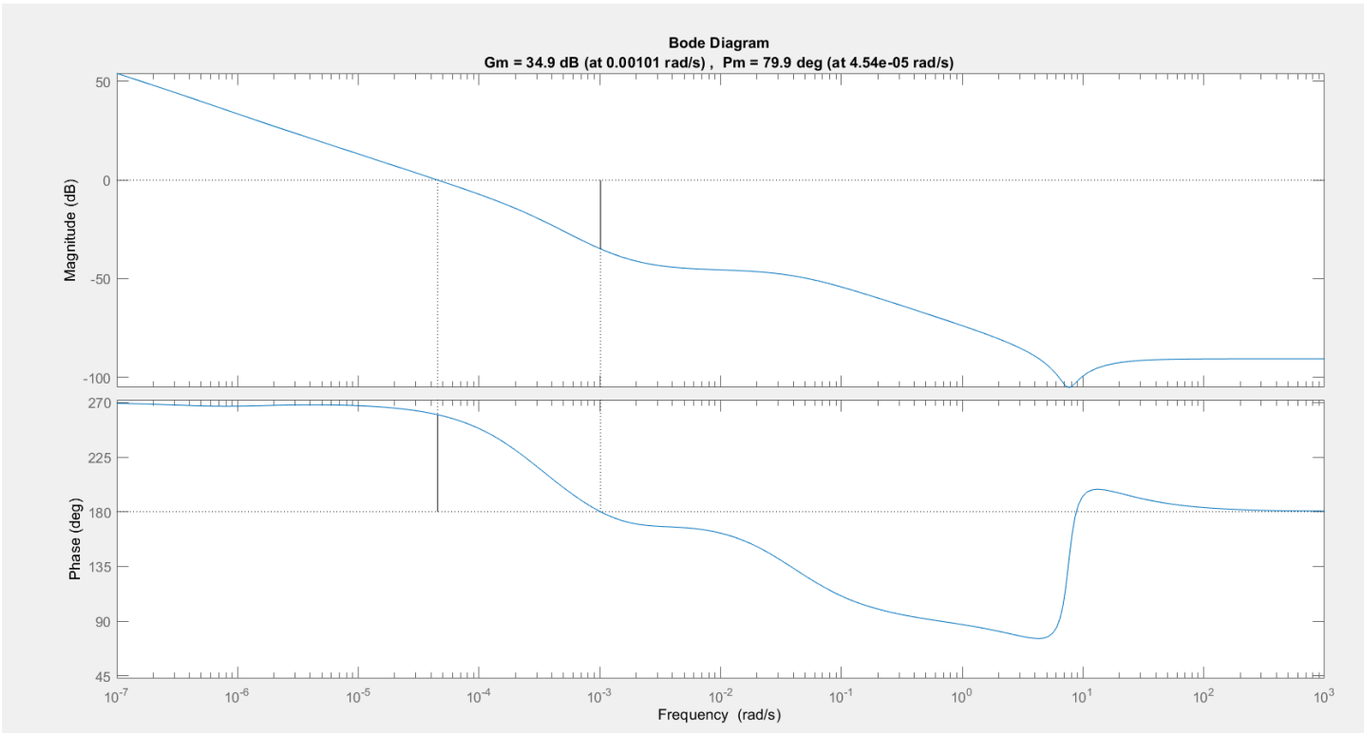
Therefore, the system is **stable**!

Bode Stability Criterion and the Nyquist Stability Criterion.

Bode:

We find the frequency response of Lg = Gcontroller*Gmodel. Where the controller transfer function is Kp + KI/s

```
%rltool(L);
%% Bode plot
% CL_sys = minreal((Gm*Gc)/(1+Gm*Gc));
Lg = Gc*Gm;
figure;
margin(Lg);
```



Gain margin:

Find where the phase crosses 180 degrees. Find the gain at that point. Value of gain needed to make the gain to 0 dB gives us the gain margin. ie (0 - Gain at that point)

Phase margin:

Find where the gain is exactly zero. At that point, find the phase phi. Then Phase Margin = phi - 180.

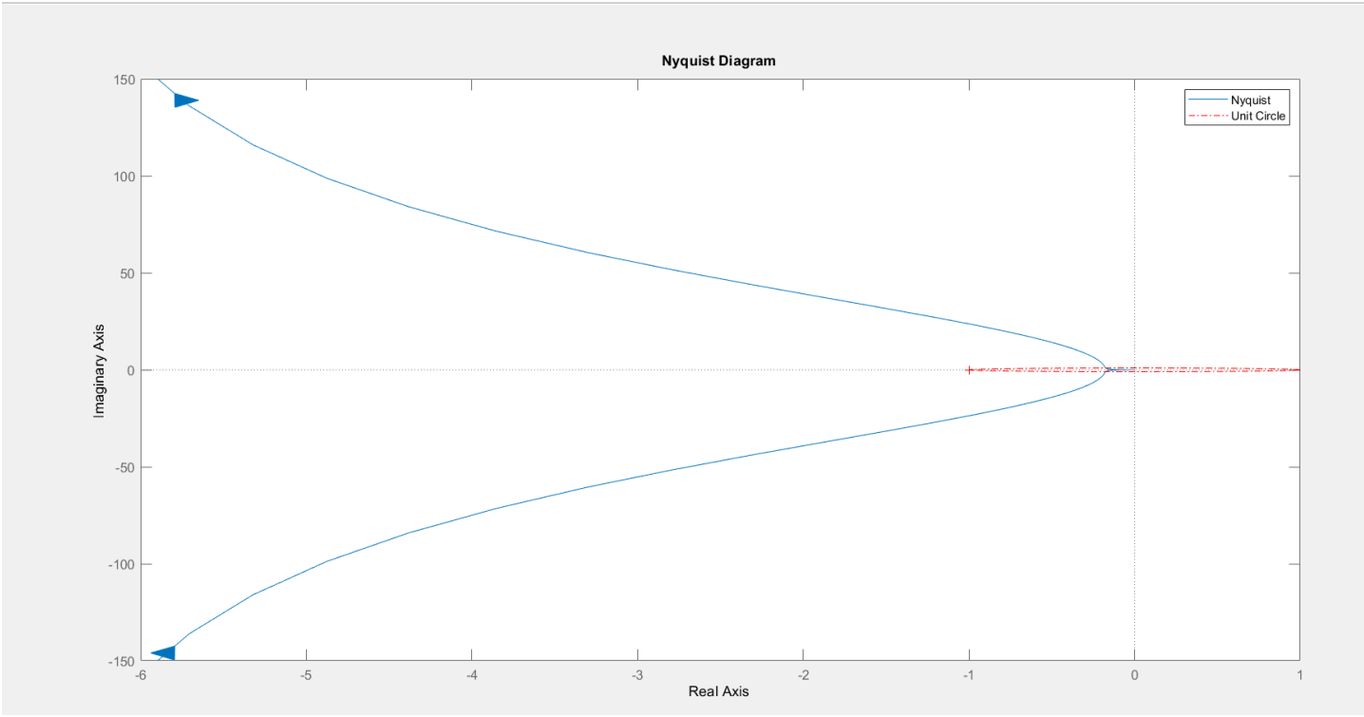
Negative gain or phase margin denotes that the system is unstable! That is there is no more margin in terms of gain or phase and we have to change the value of KI such that our margin increases.

We have positive gain and phase margins so our system is **stable.**

Nyquist plot:

The Lg used in Bode is used here too.

```
%% Nyquist Plot
figure;
nyquist(Lg);
hold on;
n = 500;
theta = linspace(0,2*pi,n);
x = cos(theta); y = sin(theta);
plot(x,y,'r-.');
hold off;
legend('Nyquist','Unit Circle');
```



We see that the number of encirclements around (-1,0) is 0. (**N = 0**)

Poles of Lg

pole(Lg)

ans =

0

-8.1792

-1.2099

-0.0395

-0.0003

-0.0000

We see that the number of RHP poles is 0. (**P = 0**)

Therefore, number of RHP zeros of 1 + Lg Z = N + P = 0

Zeros of 1 + Lg denote the poles of the closed loop transfer function.

=> number of RHP poles of the closed loop system = 0

=> The system is **stable**

Part e) Significance of the number of states

More the number of states, more complex dynamics the model can capture. By increasing the number of states, we increase the order of the system, that is, the order of the governing differential equation of the system.

Denominator of transfer function is of the form C*(s*I - A), if number of states increases (nx), number of terms in the denominator increases. (terms from 1+s+s^2+...+s^nx) . In time domain this translated to derivatives till nxth order

Solving higher order differential equations will help us fit more complicated output curves. This can also be looked at as an increase in the number of coefficients that are going to be estimated. In fact, I got a better fit when I increased the number of states from 5 to 8.

When we decrease the number of states to a number like 2, the model becomes more simplistic. It won't be able to capture all the dynamics of the system. So the fit will be poorer. This is especially so, since the outputs have a lot of variations for a long time (looks noisy in the end).

To summarise, we face an underfit-overfit trade off which we face while trying to identify a system.

A model with a higher number of states will necessarily improve the mean squared error and give better fit to given data. But the given data can contain **measurement errors** or the system can have **noise** in the input or while giving feedback. We dont want our model to reproduce the measurement errors or the noise. So, if the output of the fit model is compared to that of the true plant when given a different series of inputs, it is likely that the model will fail to represent it.

A model with a lower number of states might not fit the given data as well as the one with a higher number of states. But it might be more robust in case of presence of noise or measurement errors. Of course, the drawback is some of the complicated parts of the dynamics will be approximated poorly and won't be captured.

So the number of states should be decided carefully after considering the engineering/scientific aspects of the system

Part f) Effect of Time Delay

Time delays happen due to two reasons:

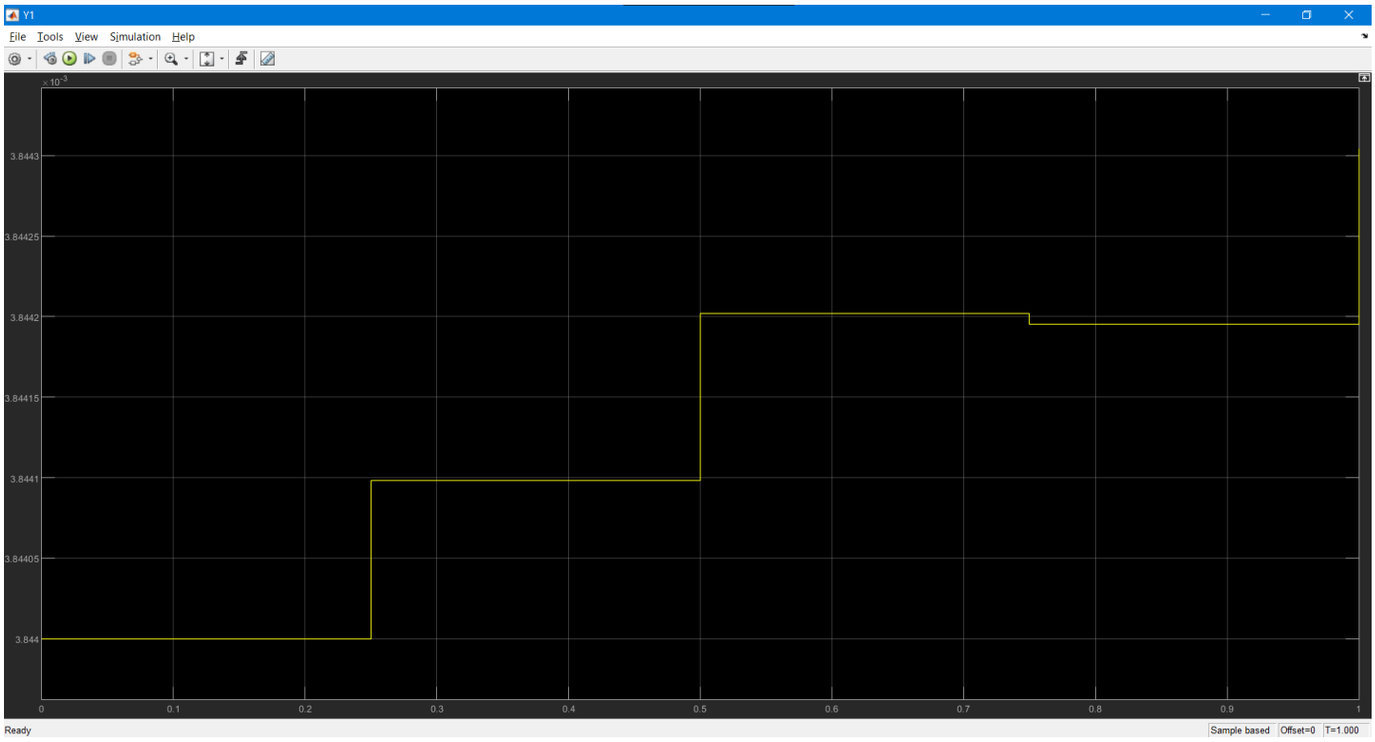
1. **Measurement delay:** Reading of a sensor measuring the output may not be sent to the controller instantaneously. This could possibly be because of the location sensor being far away from the location of the controller.
2. **Process delay:** Intrinsically the process could have a delay of its own. That is, a change in input does not reflect an immediate change in the output. For **example**, change in mass flow rate of air to regenerator (u1) may not affect the catalyst temperature (y3) instantaneously, it might take time for the increase in burning to happen and then affect the temperature. However, it is important to note that, in case of a delay time of delta, there should not even be an **infinitesimal** change in the output till delta. So, one has to be careful while modelling process delays.

Another example could be a person bathing in shower. In a shower there is a certain amount of hot water flowing and a certain amount of cold water flowing and both are mixed together. When a person wants the incoming water to be more cold, the person will increase the cold water flow. There would be some **delay due to physical transport of water** from the tap to the shower head.

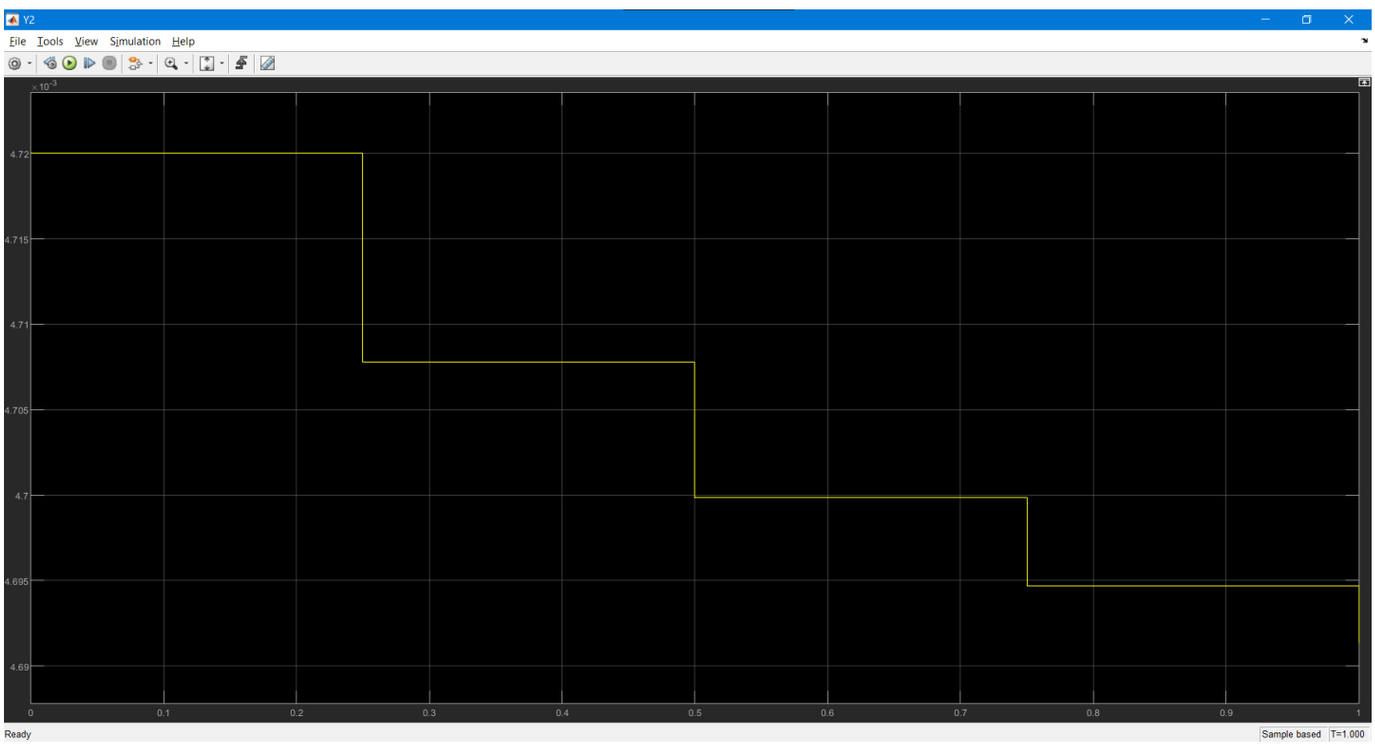
U: Flow rate

Y: Exit Temperature

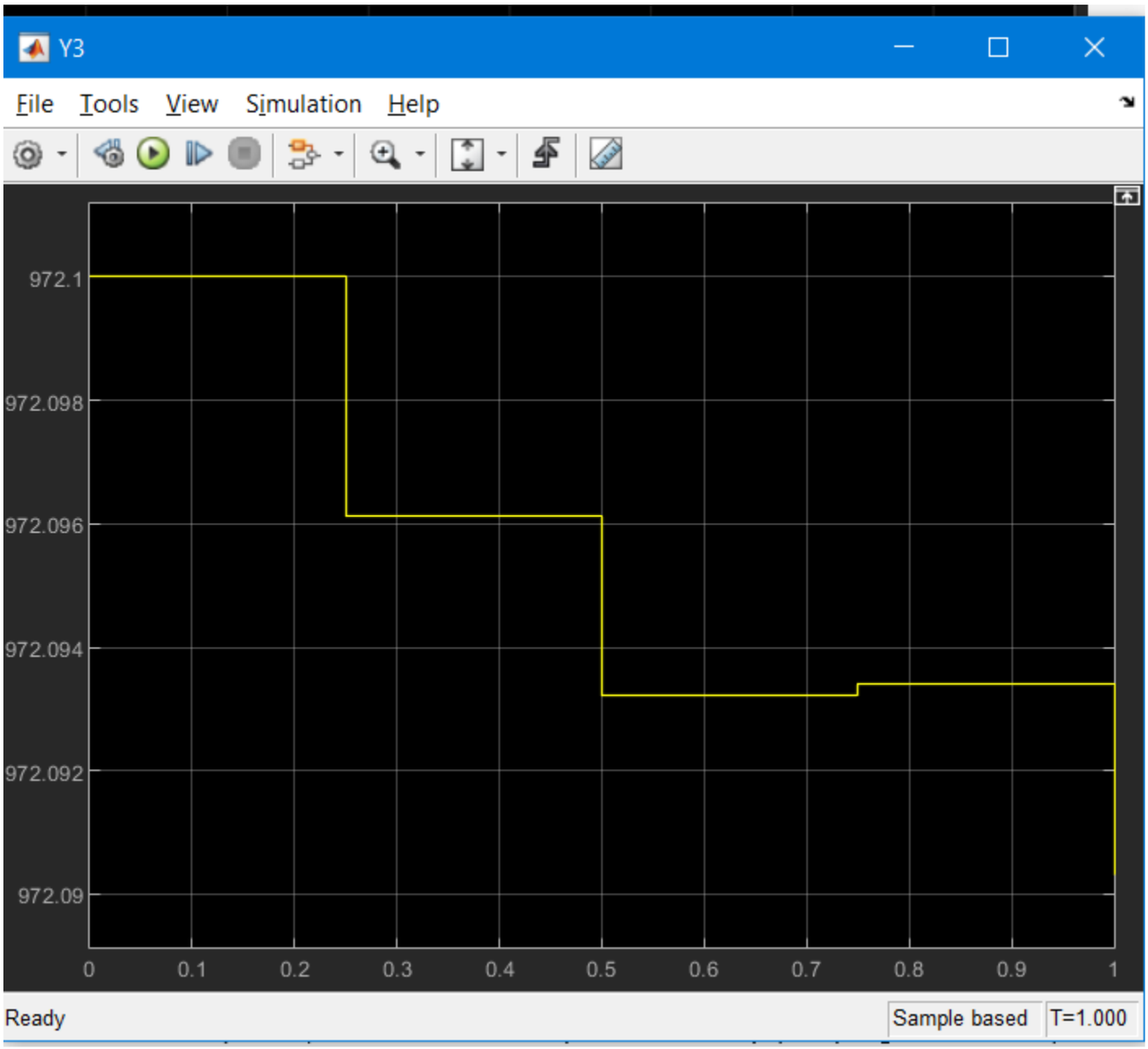
Delay reason: due to transport of water from tap to shower head



Y1



Y2



Y3

When we simulate for a very short time of 1s and plot Y1, we find that the output has no delay (sampling time is 0.25s). Even if there was a delay it must be less than that of Ts = 0.25s because we see a output at that point. So essentially, such a small delay is negligible compared to out total time period of simulation.

Conclusion: The FCC Model will not see a significant improvement if a time delay parameter was included.

One more way delay could be used is to approximate the **sluggishness** due to higher order dynamics, such as the model we are dealing with. It is likely that the true system is of higher order than 5, so incorporating time delay might help us approximate the dynamics of the higher order. This property is used in **Skogestad's half-rule.** However, as the graphs depict no delay, that might be a futile attempt.