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

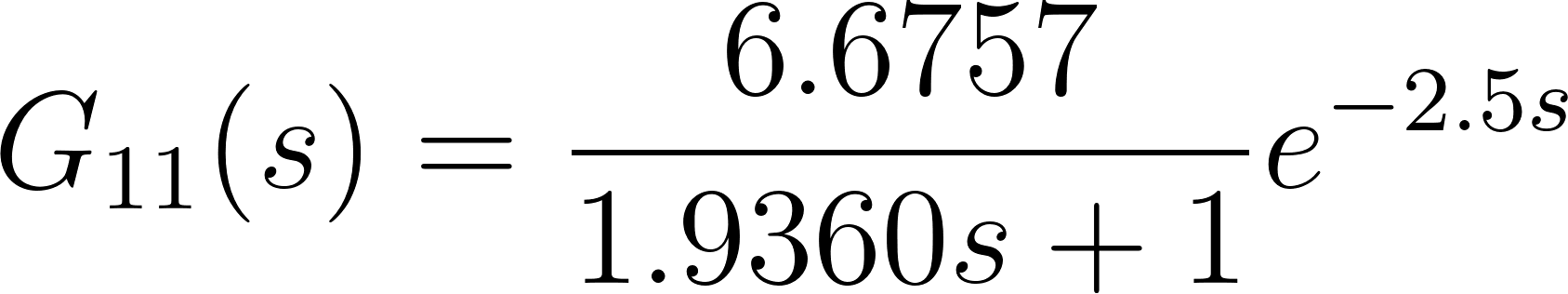
exp(-1.1\*s) \* -----------

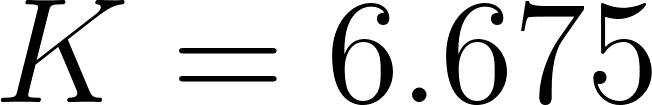
s + 0.08636

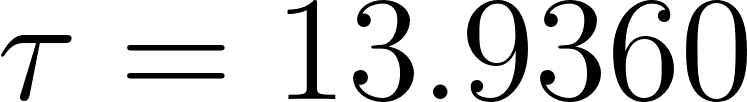
## **Part b) PI/PID Controller tuning**

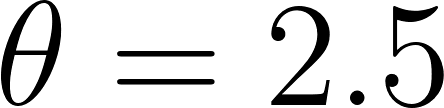
### 1: SISO with regards to top product composition

Tuning controllers using theoretical means:

[](https://www.codecogs.com/eqnedit.php?latex=G_%7B11%7D(s)%20%3D%20%5Cfrac%7B6.6757%7D%7B1.9360s%20%2B%201%7De%5E%7B-2.5s%7D#0)

=>[](https://www.codecogs.com/eqnedit.php?latex=K%20%3D%206.675#0)

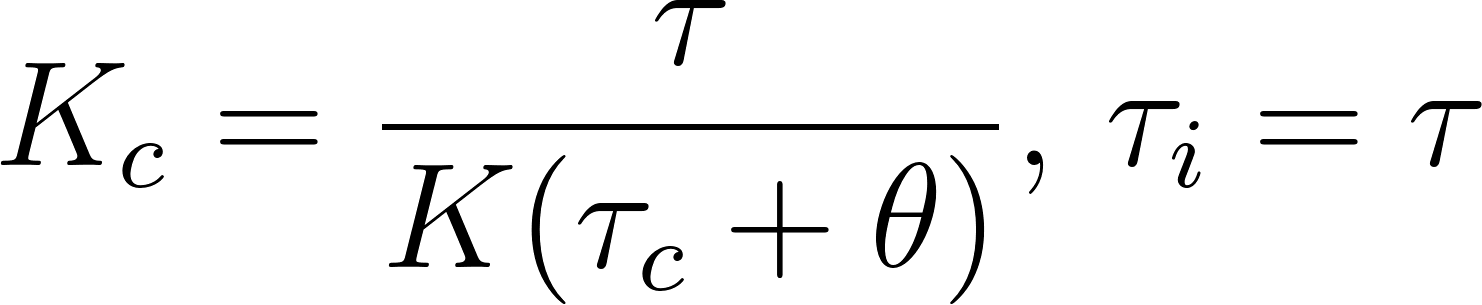
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau%20%3D%2013.9360#0)

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctheta%20%3D%202.5#0)

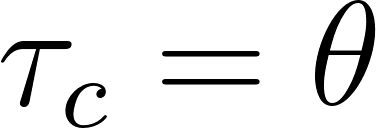
PI Controller Tuning:

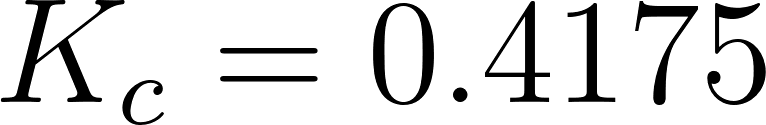
**Theoretical**

Using SIMC method:

[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%20%5Cfrac%7B%5Ctau%7D%7BK(%5Ctau_%7Bc%7D%20%2B%20%5Ctheta)%7D%2C%20%5C%2C%5Ctau_%7Bi%7D%20%3D%20%5Ctau#0)

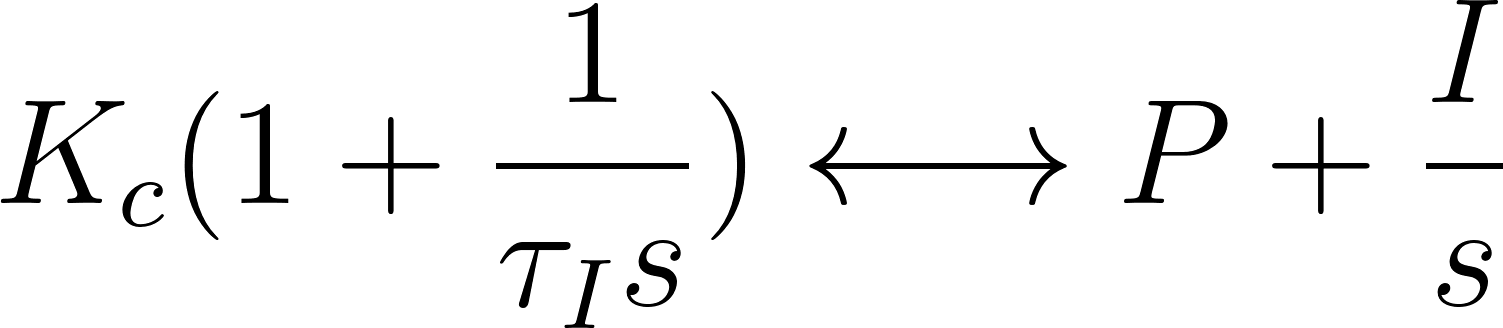
Using Skogestad’s guidelines for setting the control time constant (Skogestad, 2003):

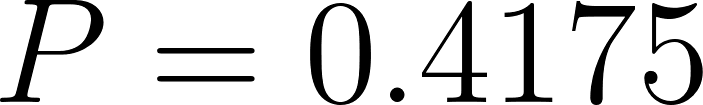
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bc%7D%20%3D%20%5Ctheta#0)

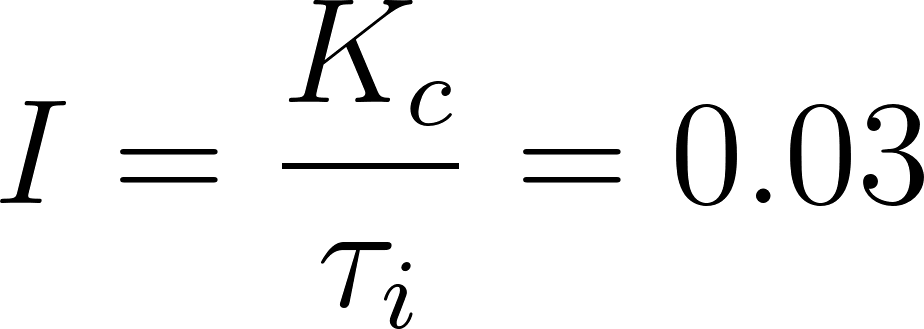
=>[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%200.4175#0)

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bi%7D%20%3D%2013.9360#0)

Parallel form transfer function of the controller:

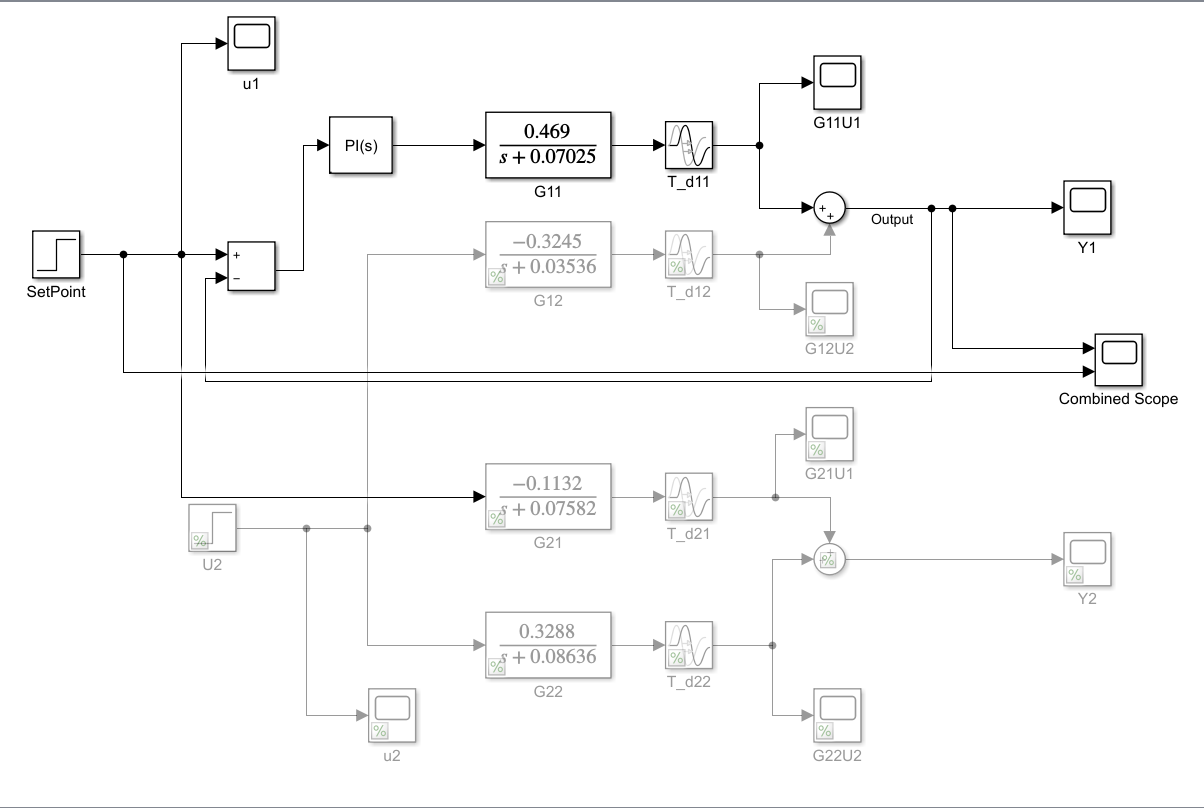
[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D(1%20%2B%20%5Cfrac%7B1%7D%7B%5Ctau_%7BI%7Ds%7D)%20%5Clongleftrightarrow%20P%20%2B%20%5Cfrac%7BI%7D%7Bs%7D#0)

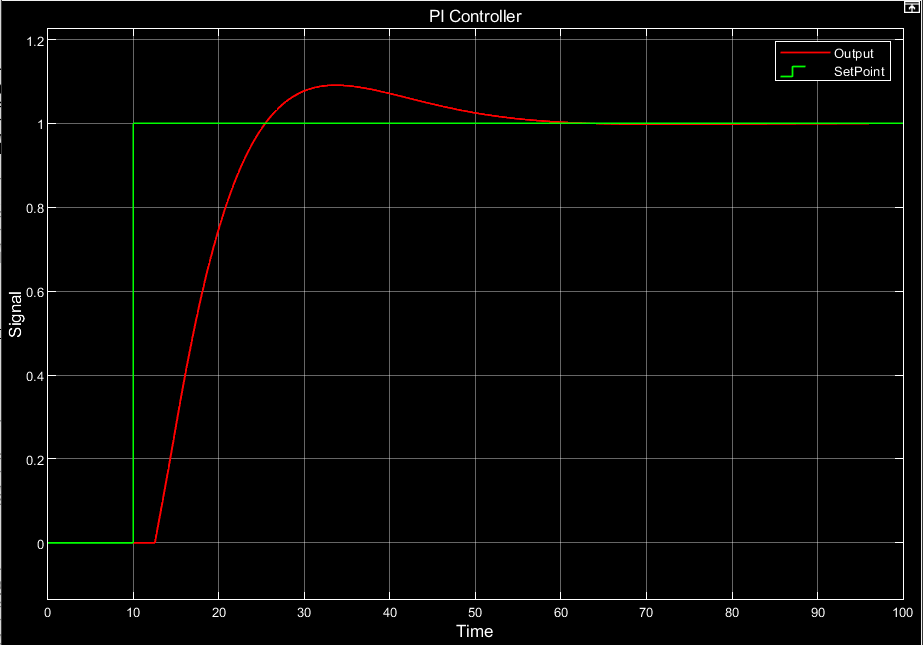
=>[](https://www.codecogs.com/eqnedit.php?latex=P%20%3D%200.4175#0)

[](https://www.codecogs.com/eqnedit.php?latex=I%20%3D%20%5Cfrac%7BK_%7Bc%7D%7D%7B%5Ctau_%7Bi%7D%7D%20%3D%200.03#0)

So final controller: 0.4175 + 0.03/s

**Simulink**

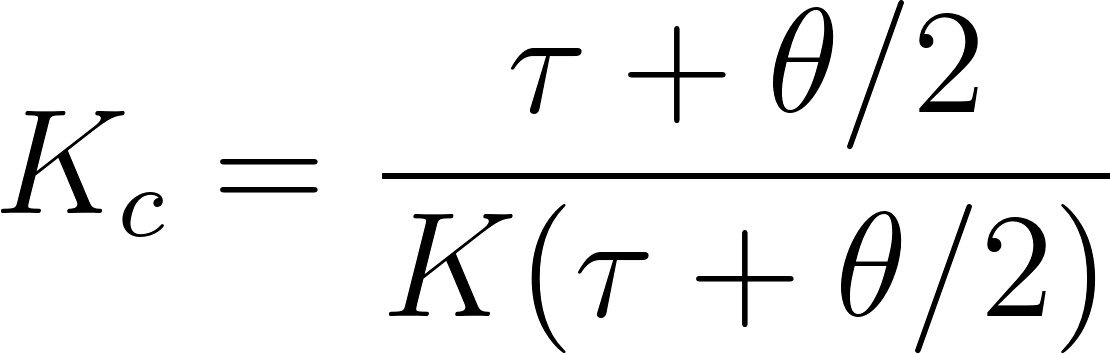




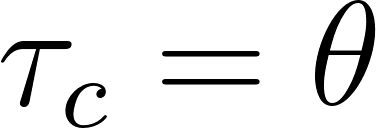
PID controller Tuning:

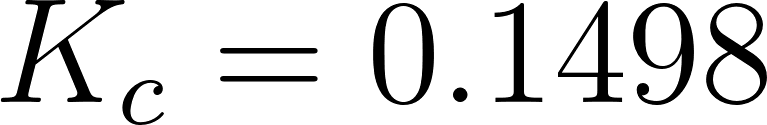
**Theoretical:**

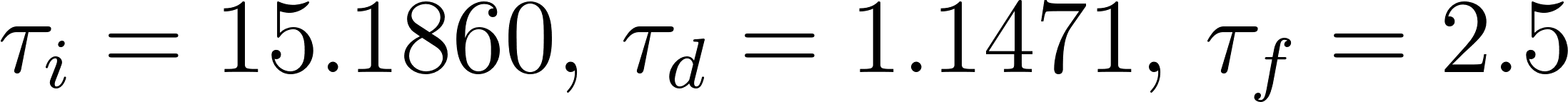
Using SIMC method:

[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%20%5Cfrac%7B%5Ctau%2B%5Ctheta%2F2%7D%7BK(%5Ctau%2B%5Ctheta%2F2)%7D#0)

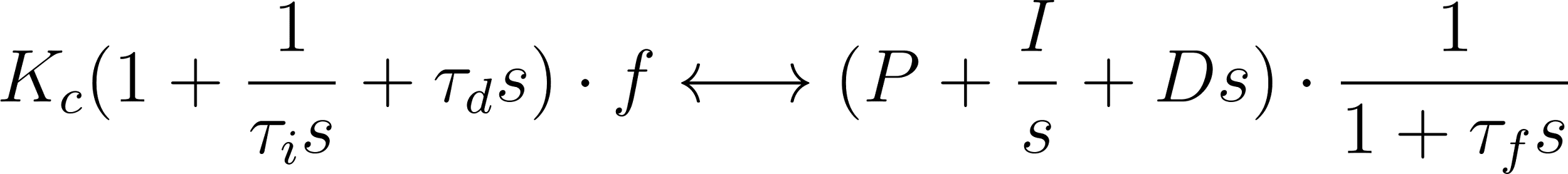
Using Skogestad’s guidelines for setting the control time constant (Skogestad, 2003):

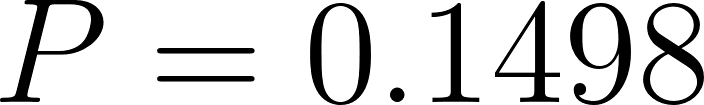
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bc%7D%20%3D%20%5Ctheta#0)

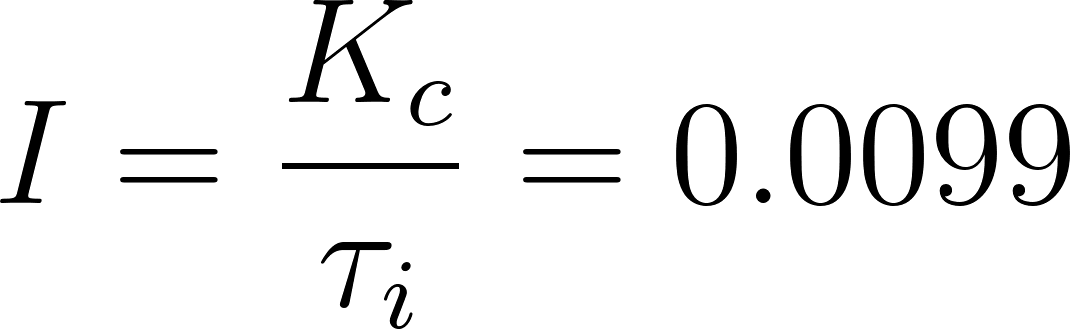
=>[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%200.1498#0)

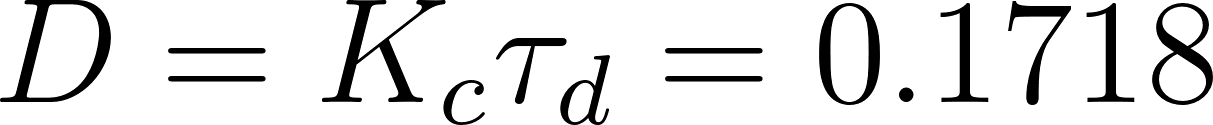
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bi%7D%20%3D%2015.1860%2C%20%5C%2C%20%5Ctau_%7Bd%7D%20%3D%201.1471%2C%20%5C%2C%20%5Ctau_%7Bf%7D%20%3D%202.5#0)

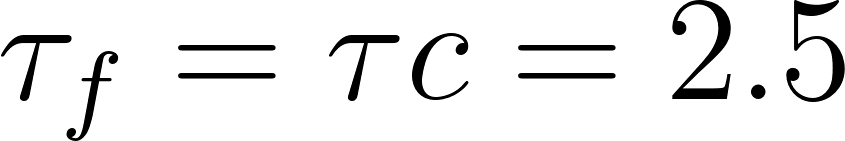
Parallel form transfer function of the controller:

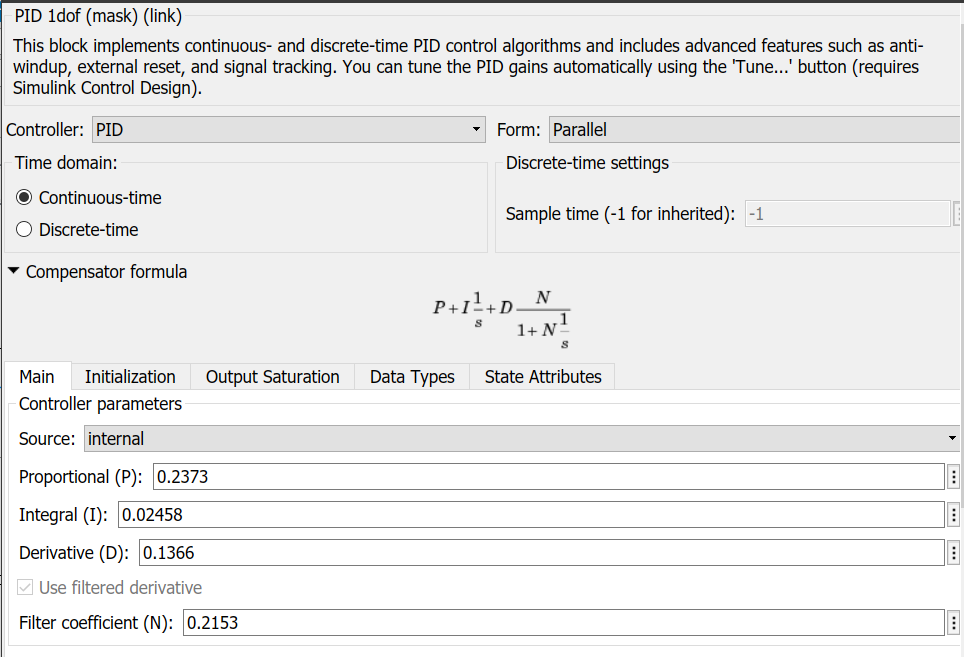
[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D(1%20%2B%20%5Cfrac%7B1%7D%7B%5Ctau_%7Bi%7Ds%7D%20%2B%20%5Ctau_%7Bd%7Ds)%20%5Ccdot%20f%20%5Clongleftrightarrow%20(P%20%2B%20%5Cfrac%7BI%7D%7Bs%7D%20%2B%20Ds)%20%5Ccdot%20%5Cfrac%7B1%7D%7B1%2B%5Ctau_%7Bf%7Ds%7D#0)

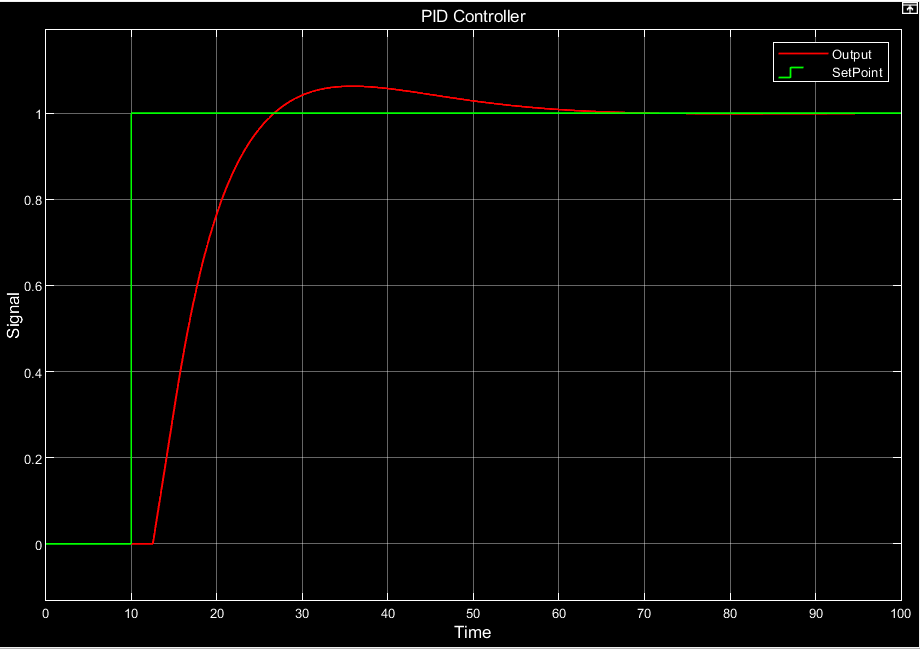
=> [](https://www.codecogs.com/eqnedit.php?latex=P%20%3D%200.1498#0)

[](https://www.codecogs.com/eqnedit.php?latex=I%20%3D%20%5Cfrac%7BK_%7Bc%7D%7D%7B%5Ctau_%7Bi%7D%7D%20%3D%200.0099#0)

[](https://www.codecogs.com/eqnedit.php?latex=D%20%3D%20K_%7Bc%7D%5Ctau_%7Bd%7D%20%3D%200.1718#0)

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bf%7D%20%3D%20%5Ctau%7Bc%7D%20%3D%202.5#0)





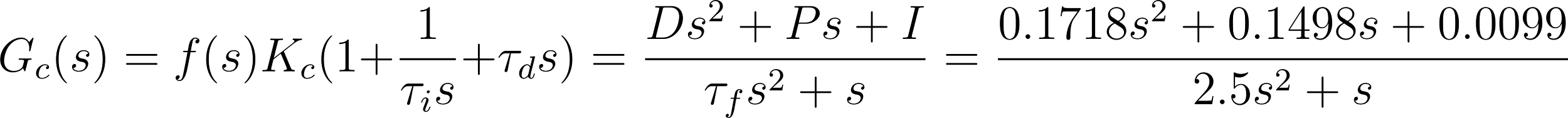
We cannot import these parameters directly into the PID controller block in SIMULINK, since the

PID controller of SIMULINK uses a different controller structure with regard to the filter. In the

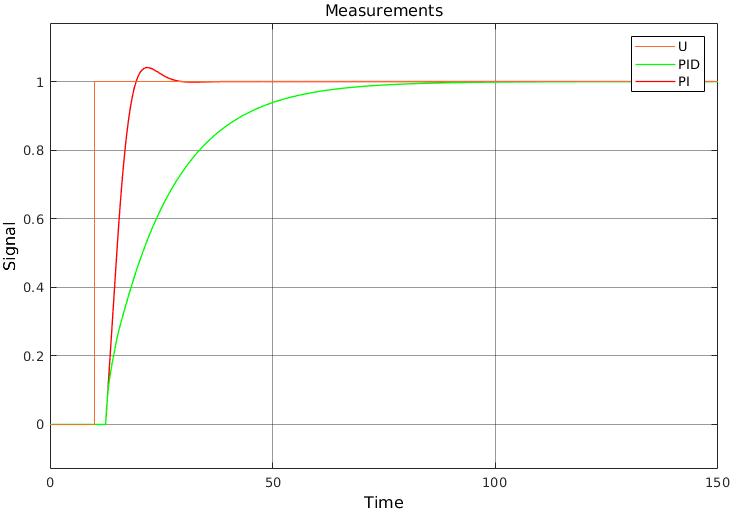
case of IMC, the filter is applied on all three controller parts, namely Proportional (P), Integral (I) and Derivative (D), whereas in SIMULINK’s PID block the filter component is applied only to the

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:

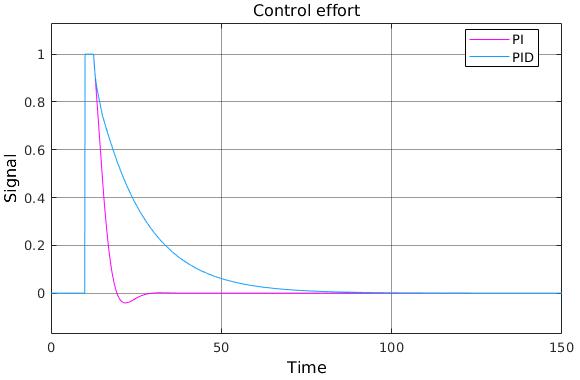
[](https://www.codecogs.com/eqnedit.php?latex=G_%7Bc%7D(s)%20%3D%20f(s)K_%7Bc%7D(1%20%2B%20%5Cfrac%7B1%7D%7B%5Ctau_%7Bi%7Ds%7D%20%2B%20%5Ctau_%7Bd%7Ds)%20%3D%20%5Cfrac%7BDs%5E2%2BPs%2BI%7D%7B%5Ctau_%7Bf%7Ds%5E2%20%2B%20s%7D%20%3D%20%5Cfrac%7B0.1718s%5E2%2B0.1498s%2B0.0099%7D%7B2.5s%5E2%2Bs%7D#0)

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



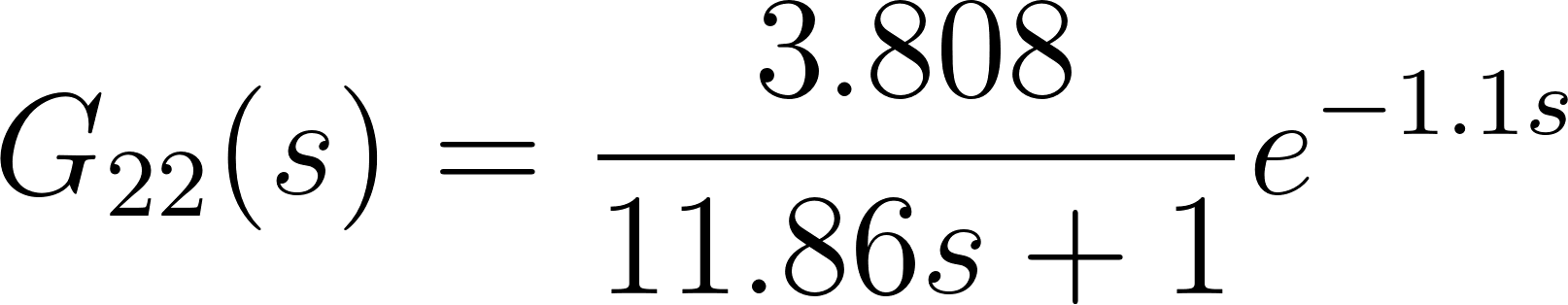
The input signal is steps up to 1 at 10s. From the above plot, we can observe that there is a delay of 2.5s between the change in setpoint and the first change in the output.

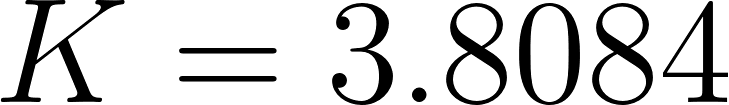
The PI controller is able to stabilise the output signal in around 30s, whereas the PID controller takes around 80s. We can also observe that the PI controller slightly overshoots the required steady state value, while still settling relatively quickly. In the case of the PID controller, there is no offshoot observed, although it takes a longer time to reach the steady state value.

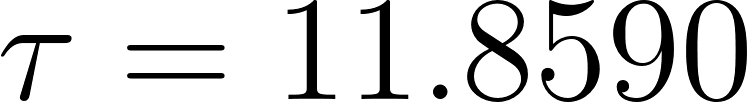


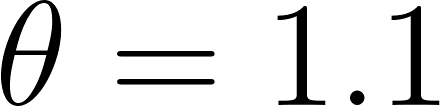
Here, we can observe that once the control effort kicks in (after a delay of 2.5s), the PI controller exerts a negative control effort for a short duration. The PID controller exerts a higher, longer sustaining effort to reach the setpoint than the PI controller. From this, we can infer that the PI controller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

### **2: SISO with regards to bottom temperature**

[](https://www.codecogs.com/eqnedit.php?latex=G_%7B22%7D(s)%20%3D%20%5Cfrac%7B3.808%7D%7B11.86s%20%2B%201%7De%5E%7B-1.1s%7D#0)

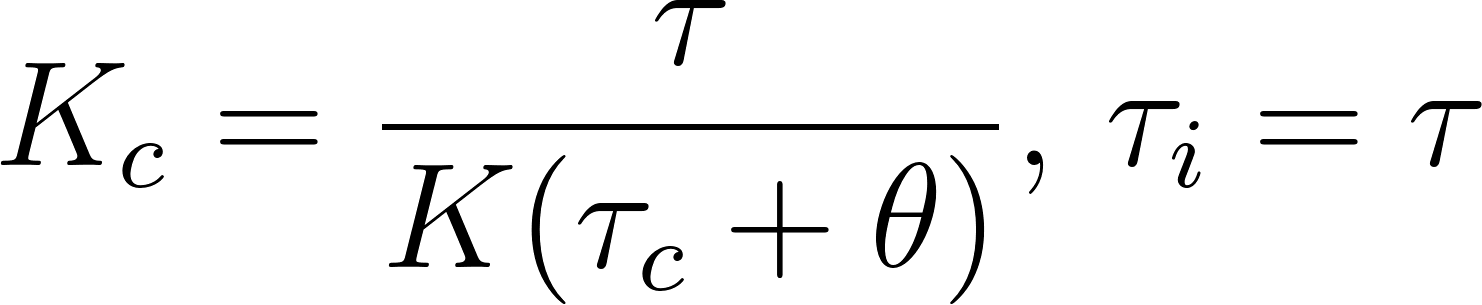
=>[](https://www.codecogs.com/eqnedit.php?latex=K%20%3D%203.8084#0)

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau%20%3D%2011.8590#0)

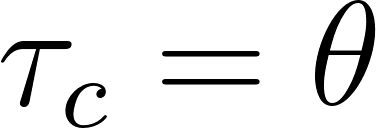
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctheta%20%3D%201.1#0)

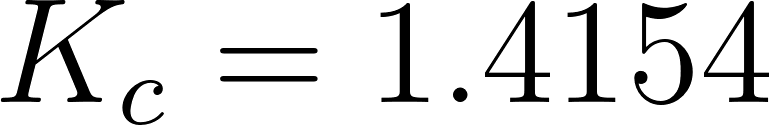
PI Controller Tuning:

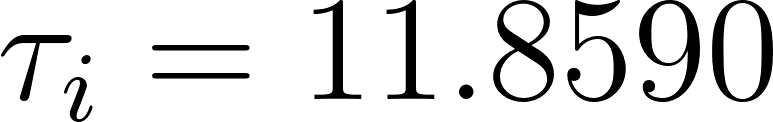
Using SIMC method:

[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%20%5Cfrac%7B%5Ctau%7D%7BK(%5Ctau_%7Bc%7D%20%2B%20%5Ctheta)%7D%2C%20%5C%2C%5Ctau_%7Bi%7D%20%3D%20%5Ctau#0)

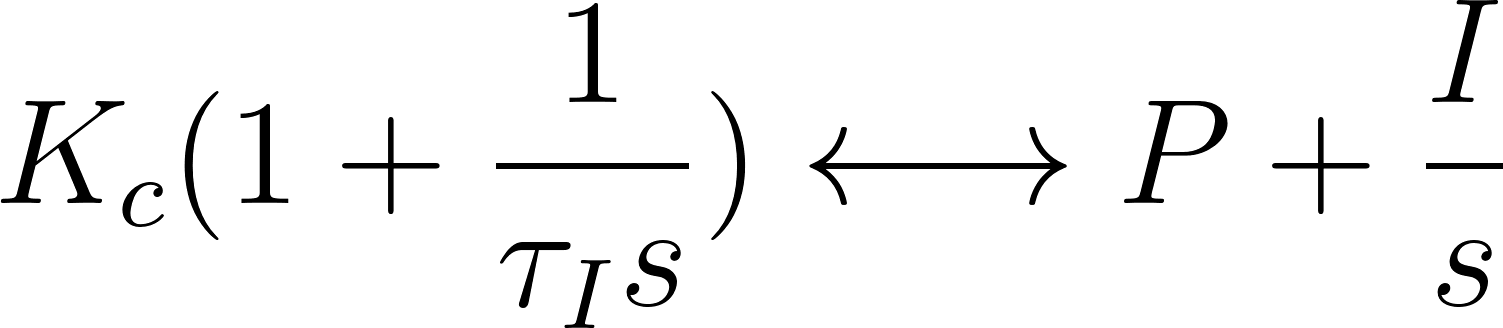
Using Skogestad’s guidelines for setting the control time constant (Skogestad, 2003):

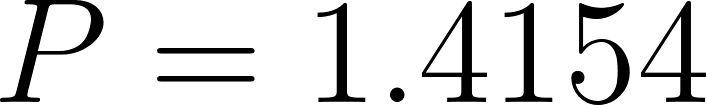
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bc%7D%20%3D%20%5Ctheta#0)

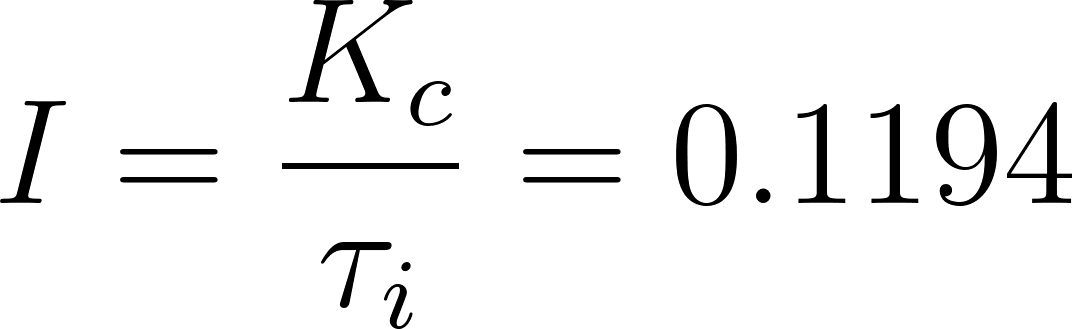
=>[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%201.4154#0)

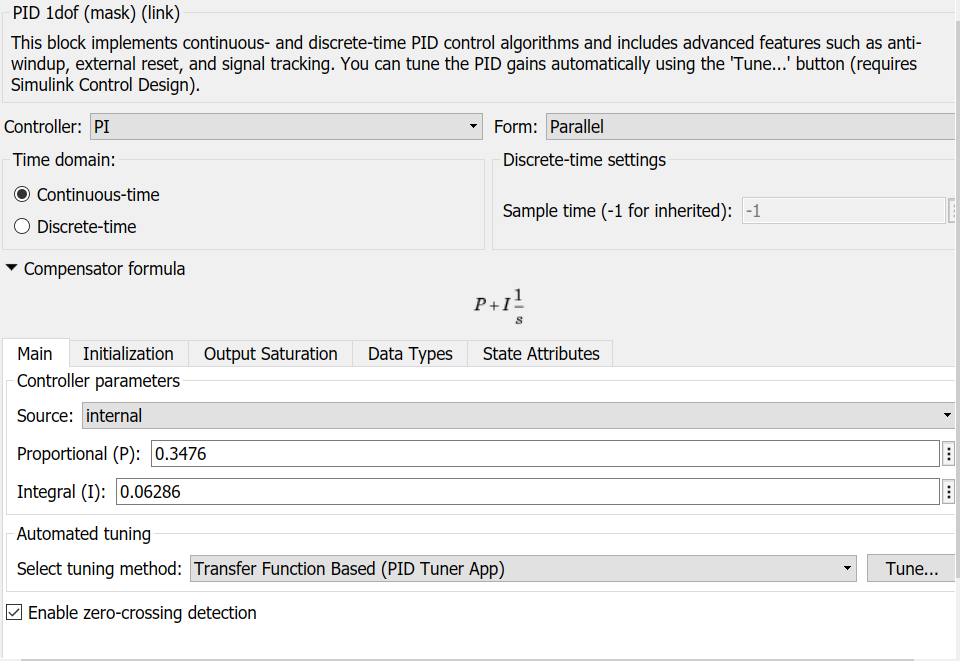
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bi%7D%20%3D%2011.8590#0)

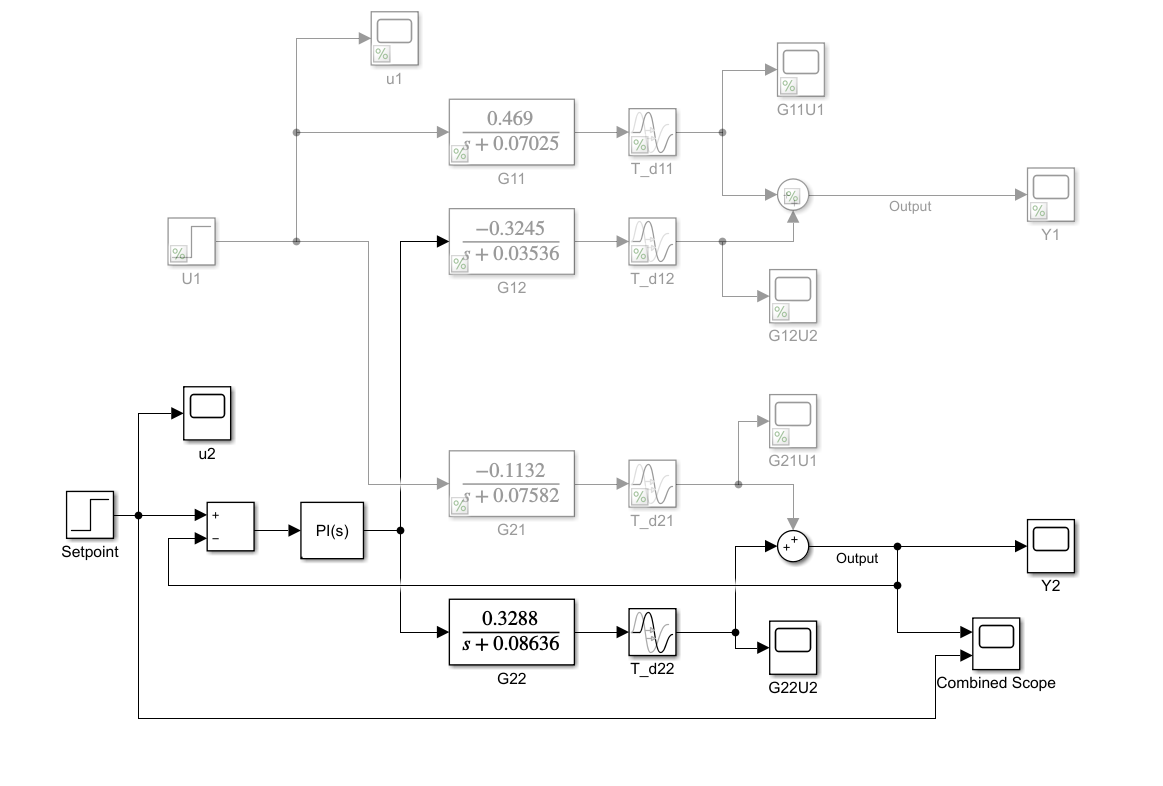
Parallel form transfer function of the controller:

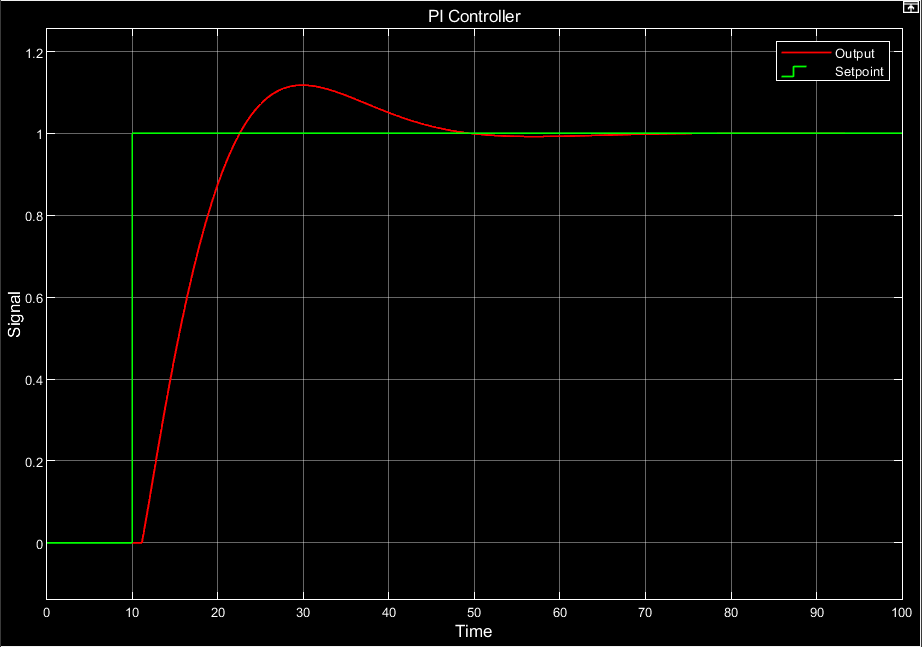
[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D(1%20%2B%20%5Cfrac%7B1%7D%7B%5Ctau_%7BI%7Ds%7D)%20%5Clongleftrightarrow%20P%20%2B%20%5Cfrac%7BI%7D%7Bs%7D#0)

=>[](https://www.codecogs.com/eqnedit.php?latex=P%20%3D%201.4154#0)

[](https://www.codecogs.com/eqnedit.php?latex=I%20%3D%20%5Cfrac%7BK_%7Bc%7D%7D%7B%5Ctau_%7Bi%7D%7D%20%3D%200.1194#0)

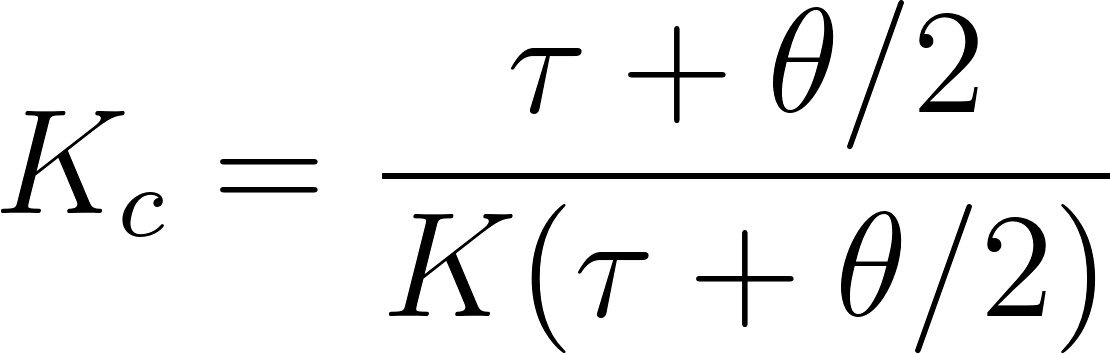




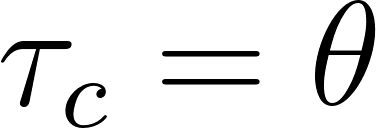


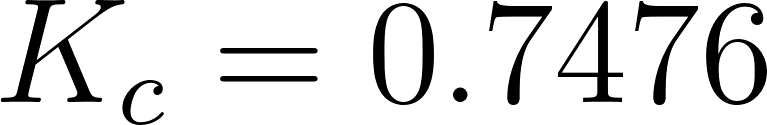
PID controller Tuning:

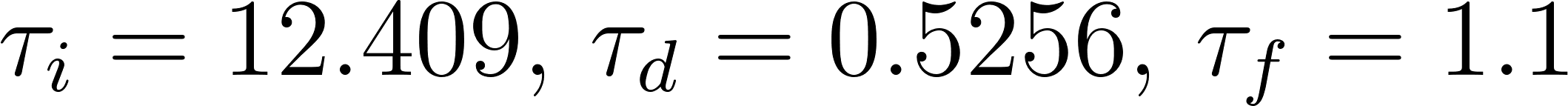
Using SIMC method:

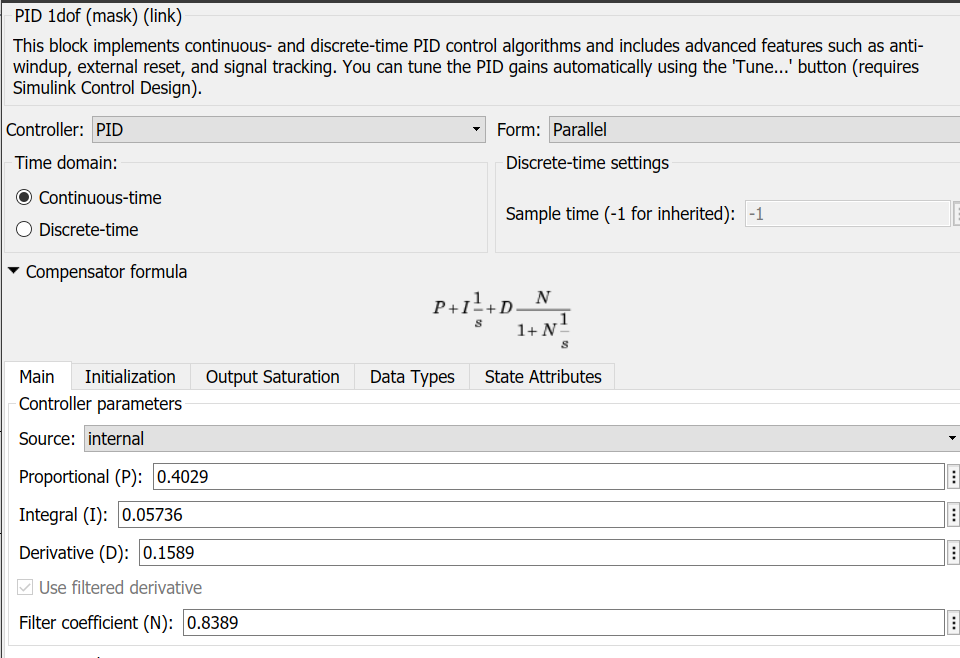
[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%20%5Cfrac%7B%5Ctau%2B%5Ctheta%2F2%7D%7BK(%5Ctau%2B%5Ctheta%2F2)%7D#0)

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

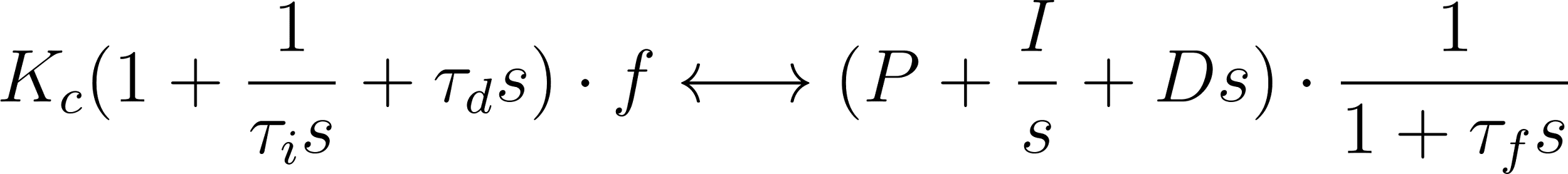
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bc%7D%20%3D%20%5Ctheta#0)

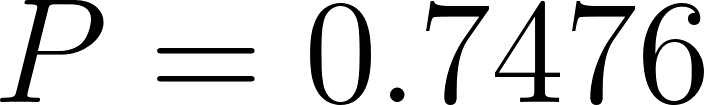
=>[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D%20%3D%200.7476#0)

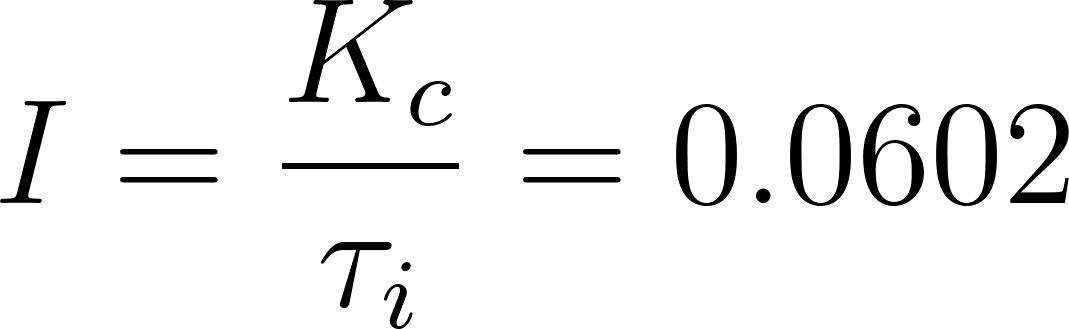
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bi%7D%20%3D%2012.409%2C%20%5C%2C%20%5Ctau_%7Bd%7D%20%3D%200.5256%2C%20%5C%2C%20%5Ctau_%7Bf%7D%20%3D%201.1#0)

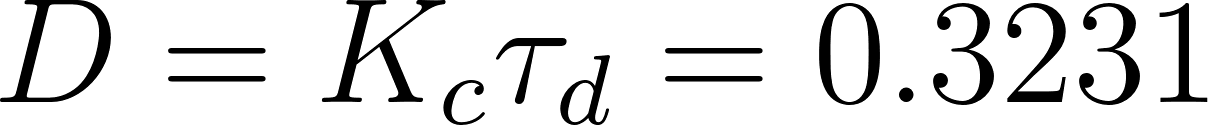
l

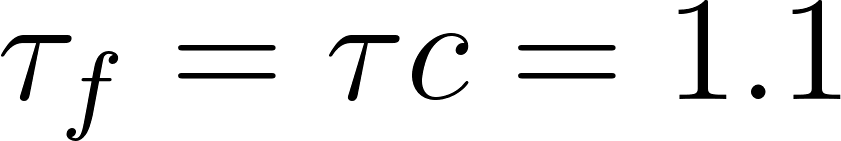
Parallel form transfer function of the controler:

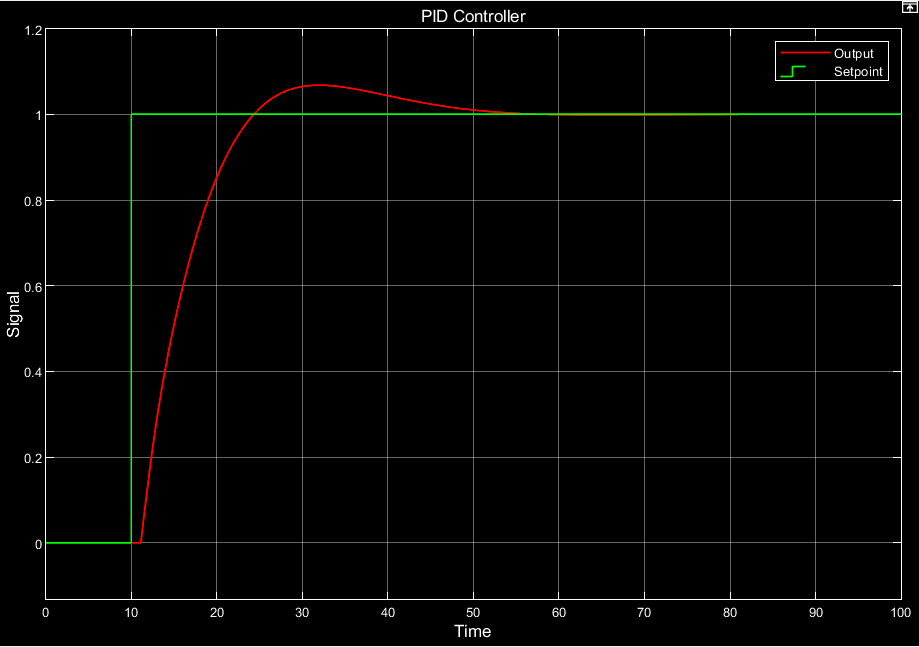
[](https://www.codecogs.com/eqnedit.php?latex=K_%7Bc%7D(1%20%2B%20%5Cfrac%7B1%7D%7B%5Ctau_%7Bi%7Ds%7D%20%2B%20%5Ctau_%7Bd%7Ds)%20%5Ccdot%20f%20%5Clongleftrightarrow%20(P%20%2B%20%5Cfrac%7BI%7D%7Bs%7D%20%2B%20Ds)%20%5Ccdot%20%5Cfrac%7B1%7D%7B1%2B%5Ctau_%7Bf%7Ds%7D#0)

=> [](https://www.codecogs.com/eqnedit.php?latex=P%20%3D%200.7476#0)

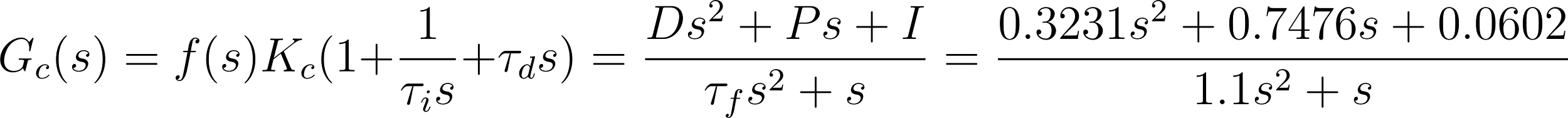
[](https://www.codecogs.com/eqnedit.php?latex=I%20%3D%20%5Cfrac%7BK_%7Bc%7D%7D%7B%5Ctau_%7Bi%7D%7D%20%3D%200.0602#0)

[](https://www.codecogs.com/eqnedit.php?latex=D%20%3D%20K_%7Bc%7D%5Ctau_%7Bd%7D%20%3D%200.3231#0)

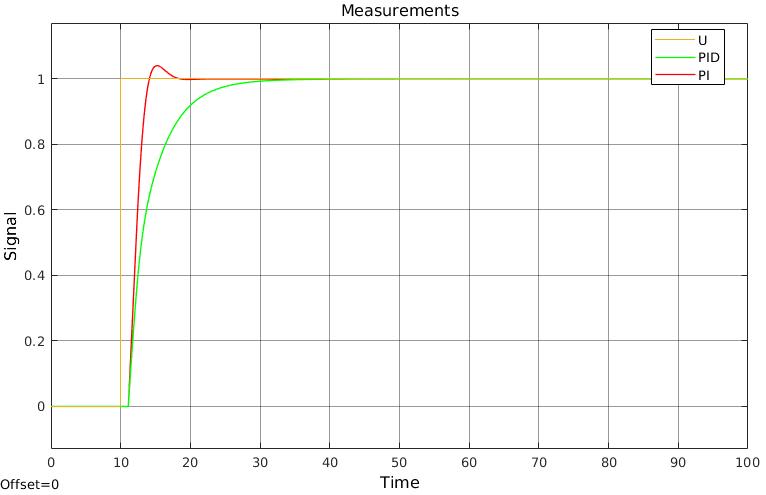
[](https://www.codecogs.com/eqnedit.php?latex=%5Ctau_%7Bf%7D%20%3D%20%5Ctau%7Bc%7D%20%3D%201.1#0)



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

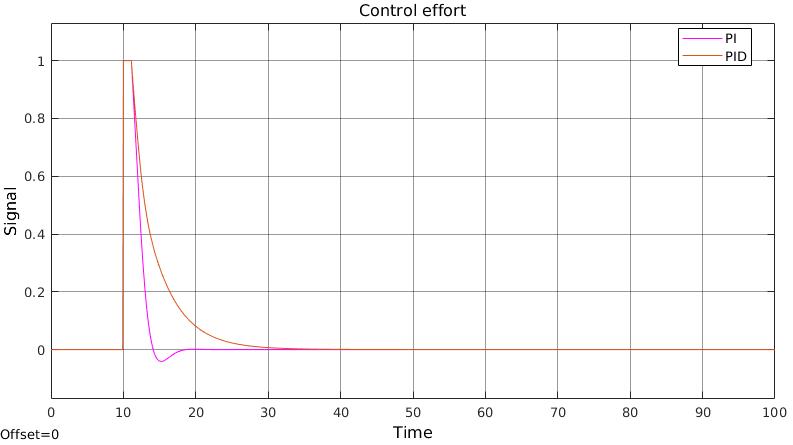
[](https://www.codecogs.com/eqnedit.php?latex=G_%7Bc%7D(s)%20%3D%20f(s)K_%7Bc%7D(1%20%2B%20%5Cfrac%7B1%7D%7B%5Ctau_%7Bi%7Ds%7D%20%2B%20%5Ctau_%7Bd%7Ds)%20%3D%20%5Cfrac%7BDs%5E2%2BPs%2BI%7D%7B%5Ctau_%7Bf%7Ds%5E2%20%2B%20s%7D%20%3D%20%5Cfrac%7B0.3231s%5E2%2B0.7476s%2B0.0602%7D%7B1.1s%5E2%2Bs%7D#0)

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.

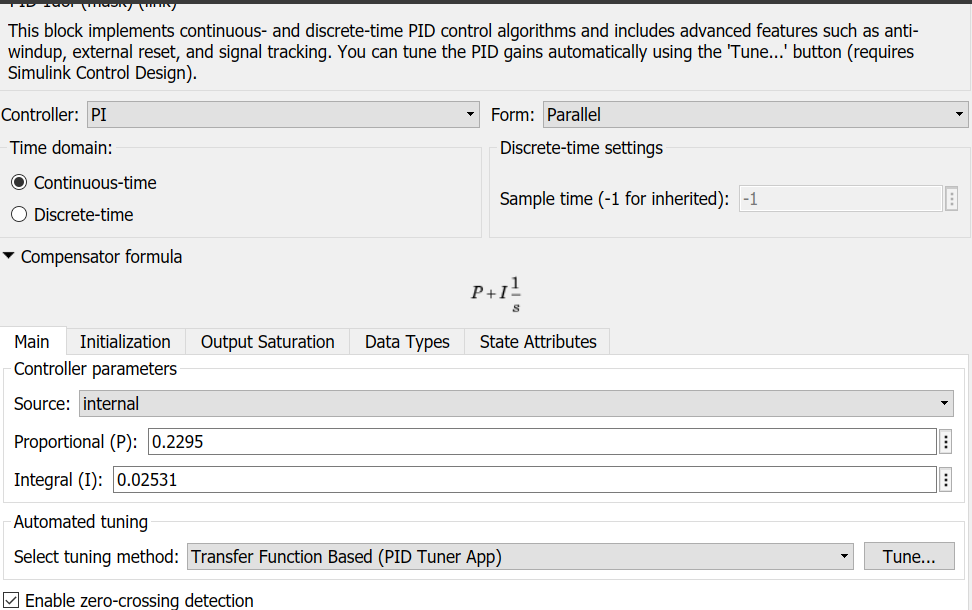
The PI controller is able to stabilise the output signal in around 18s, whereas the PID controller takes around 35s. We can also observe that the PI controller slightly overshoots the required steady state value, while still settling relatively quickly. In the case of the PID controller, there is no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

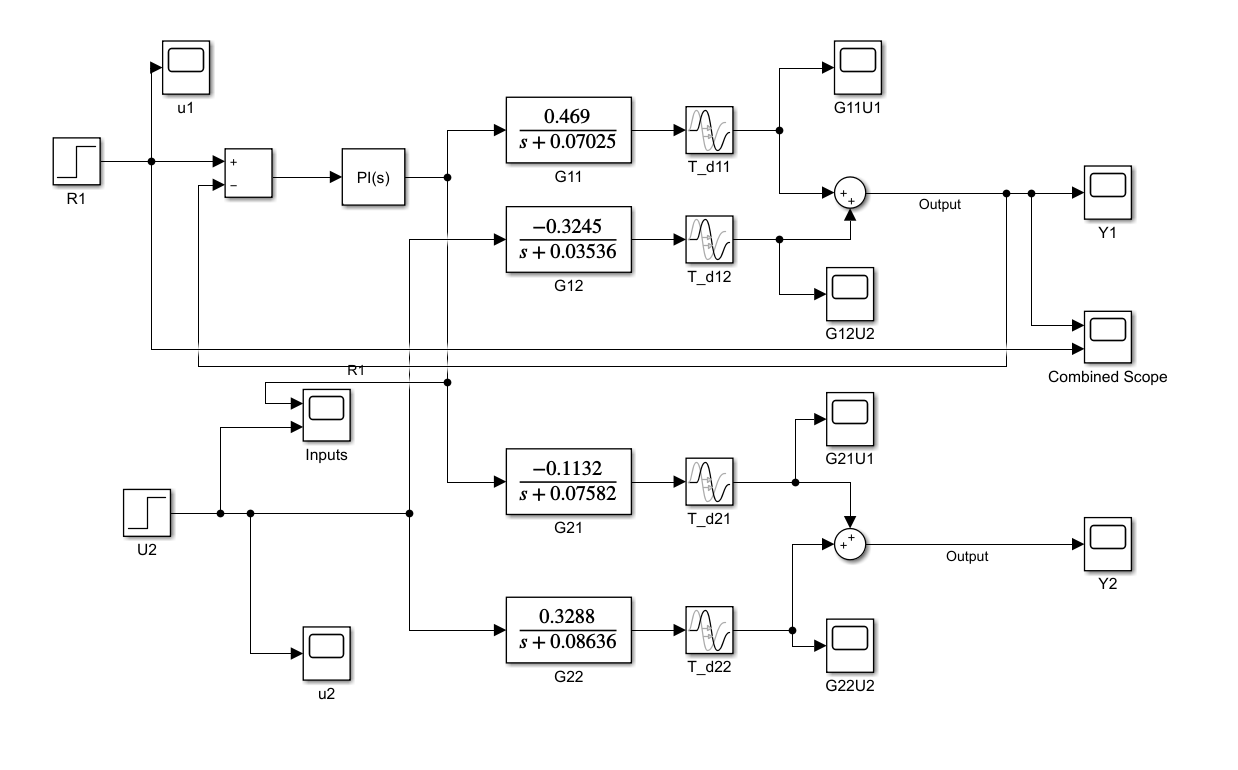


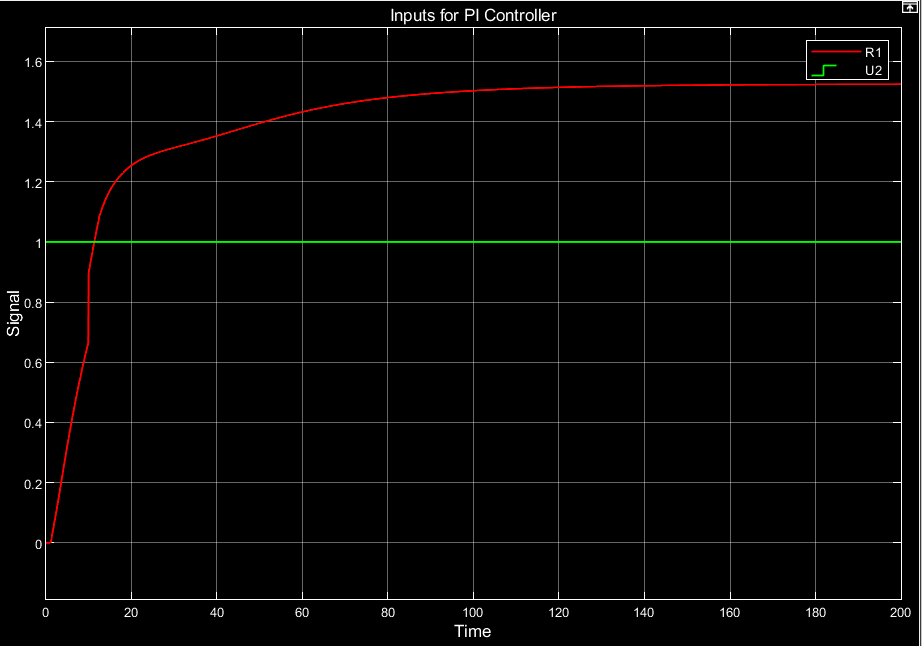
Here, we can observe that once the control effort kicks in (after a delay of 1.1s), the PI controller exerts a negative control effort for a short duration. The PID controller exerts a higher, longer sustaining effort to reach the setpoint than the PI controller. From this, we can infer that the PI controller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

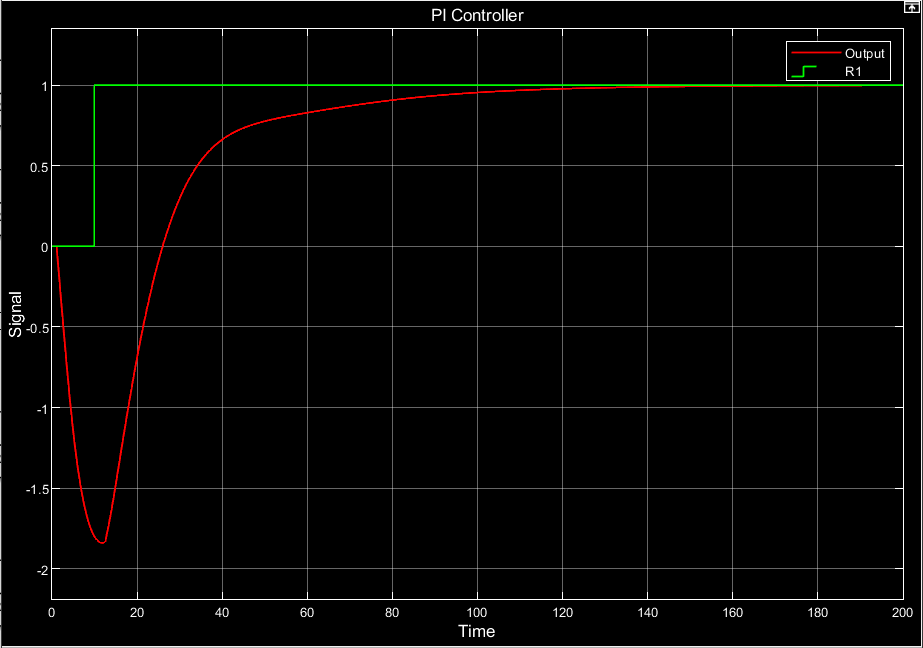
### **3: MIMO top product composition**

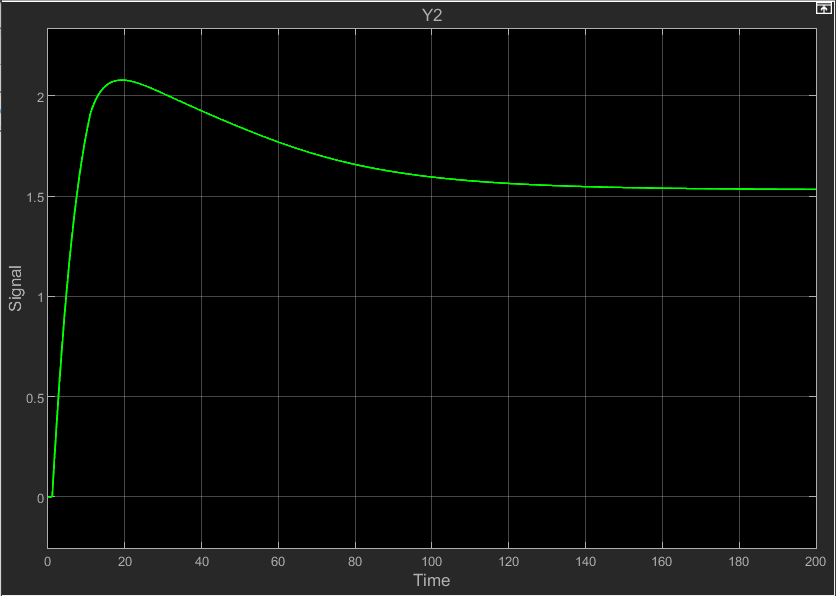
PI Control:



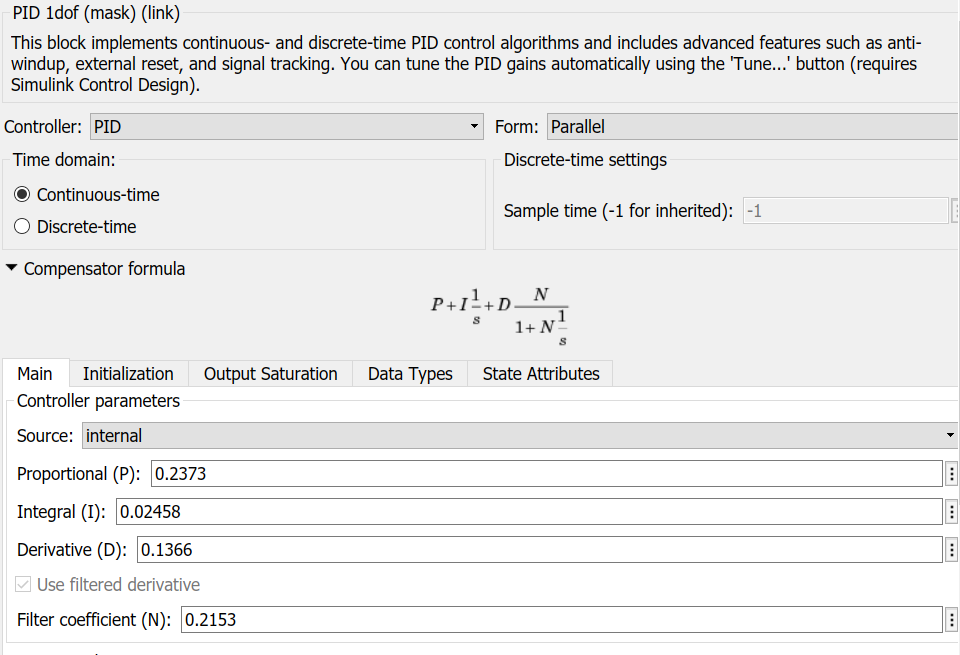


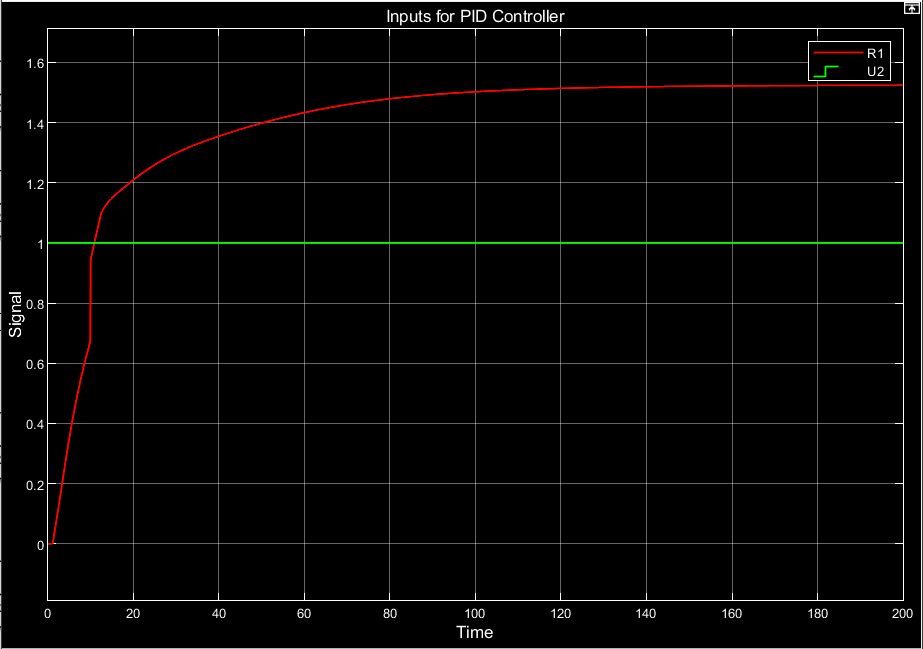


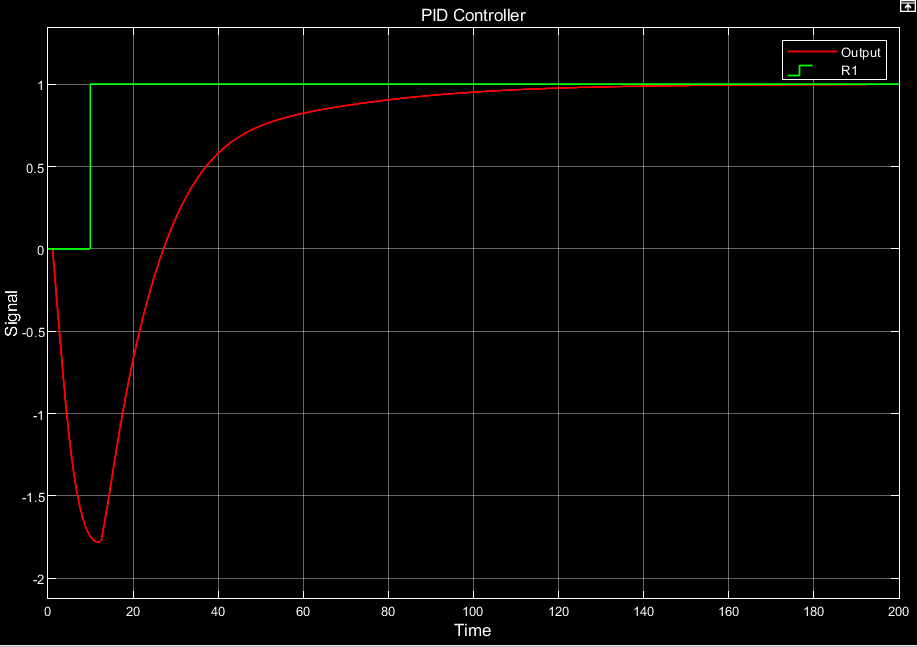


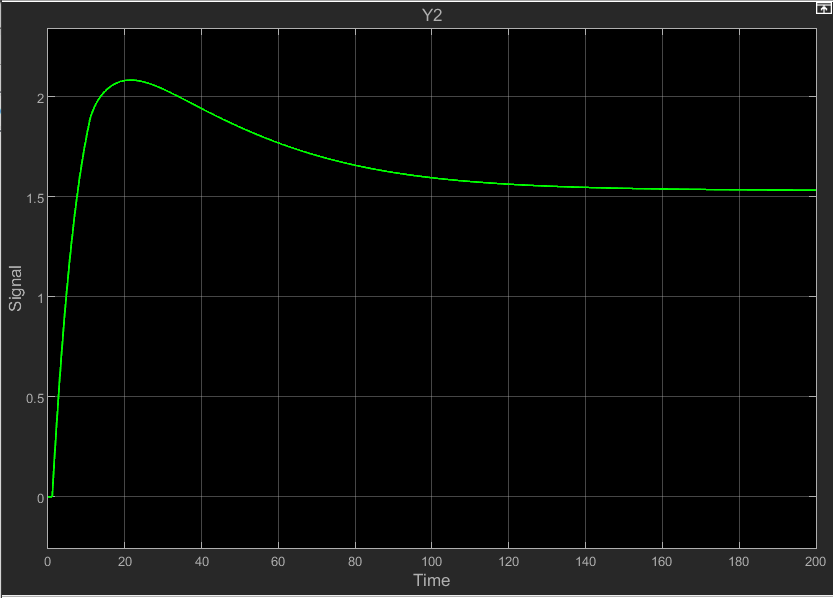


PID Controller:

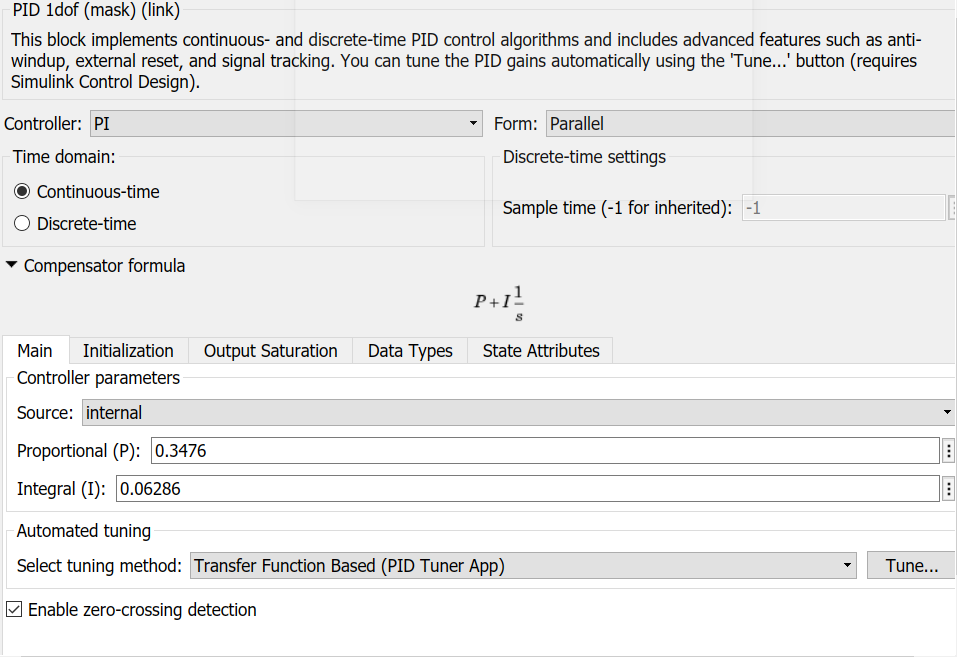


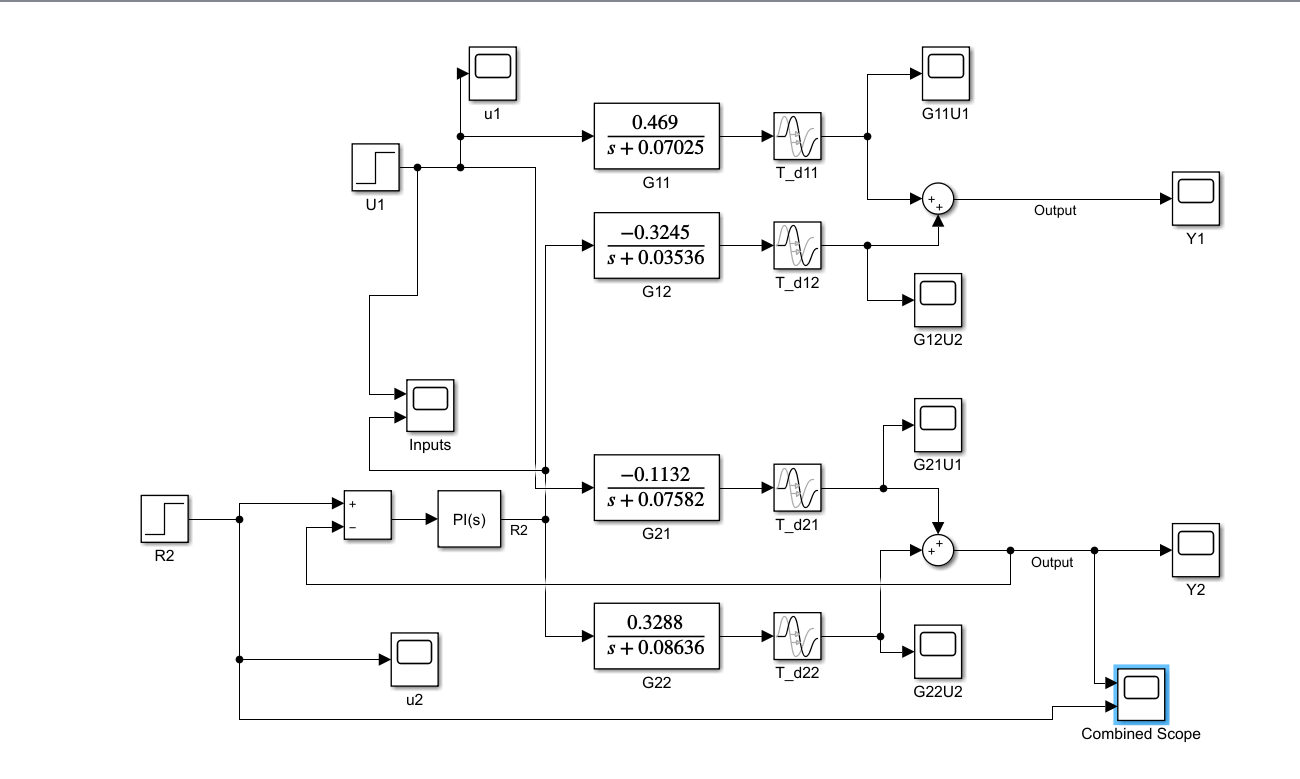


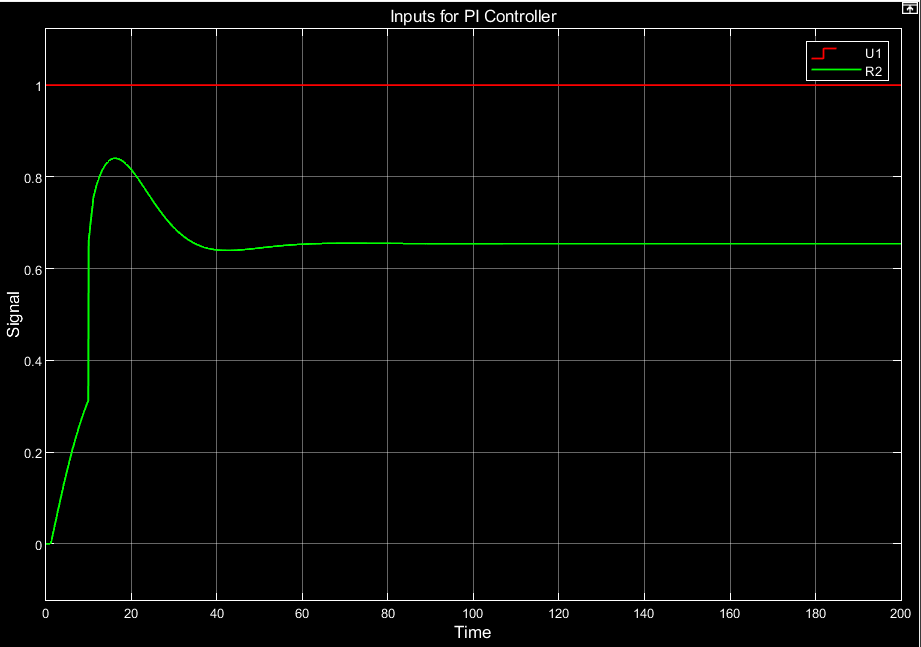


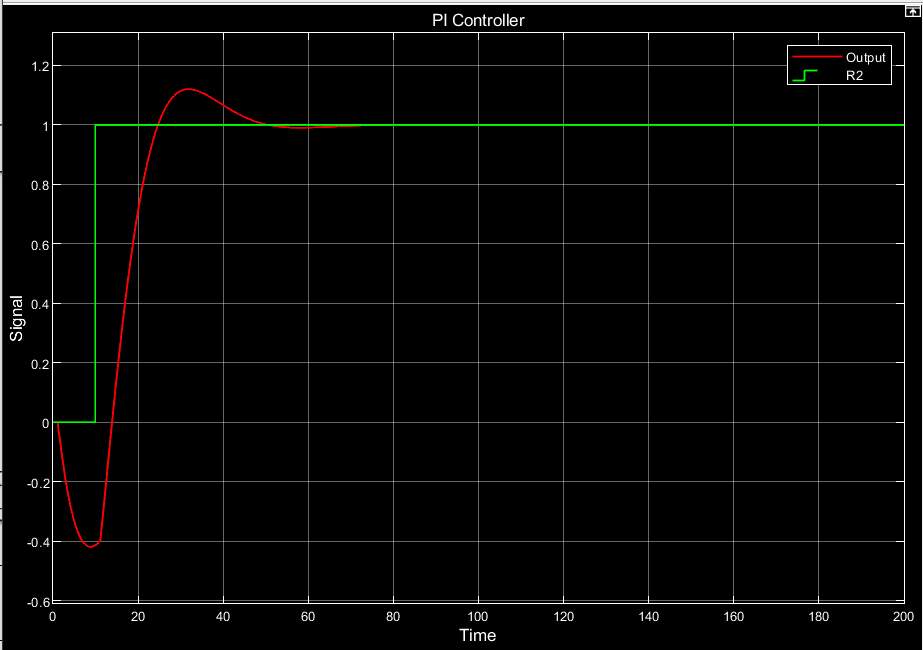


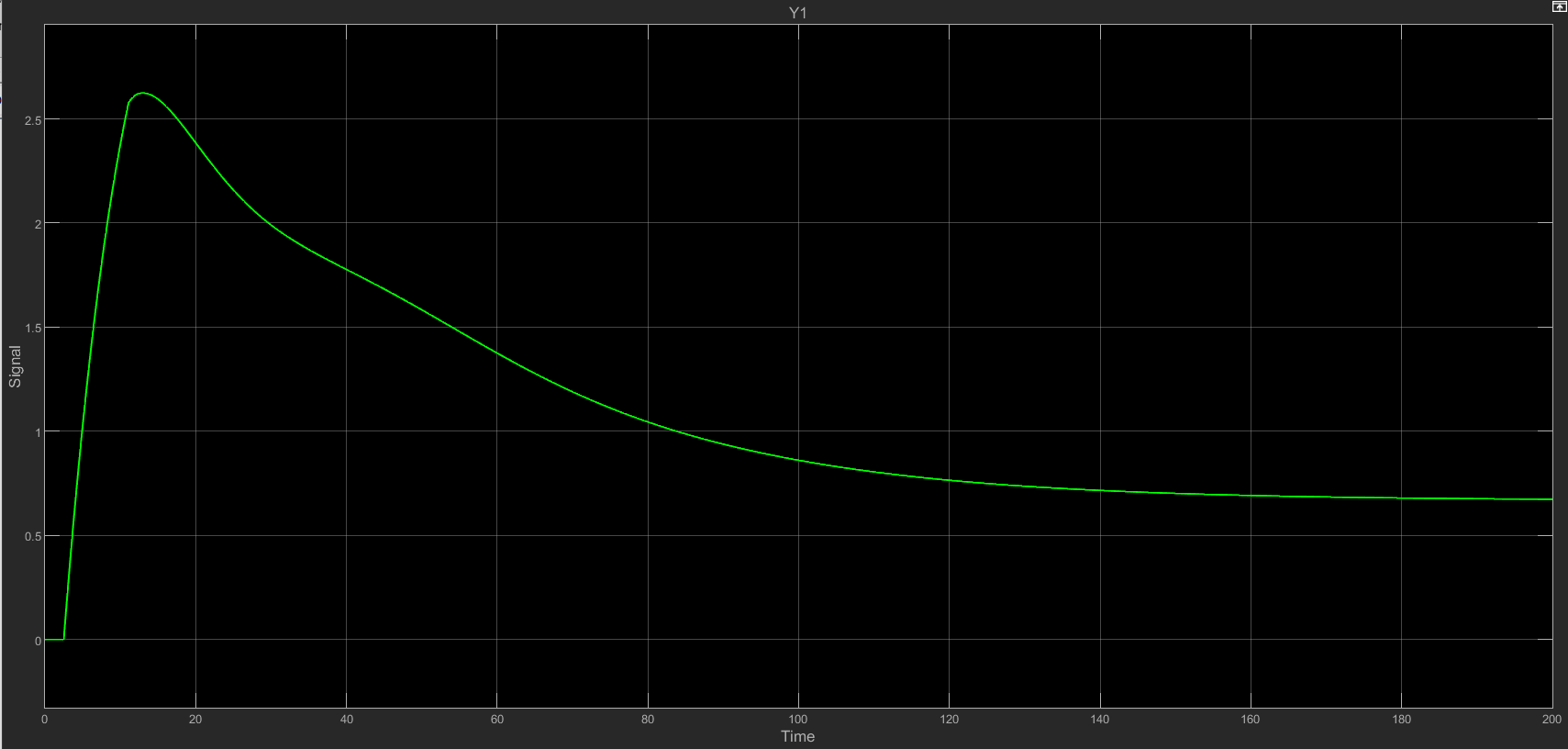
### **4: MIMO bottom temperature**

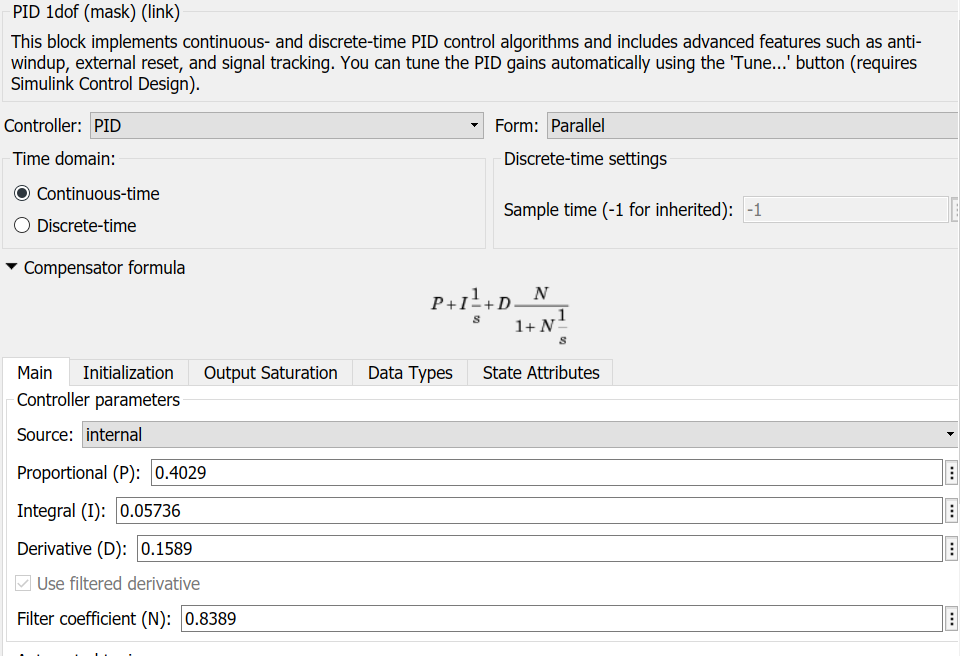


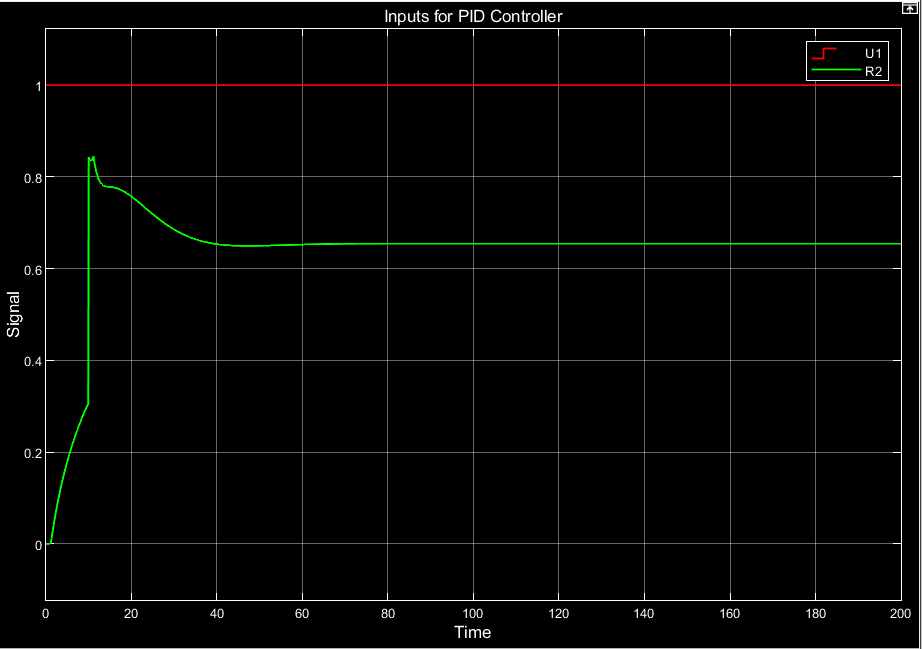




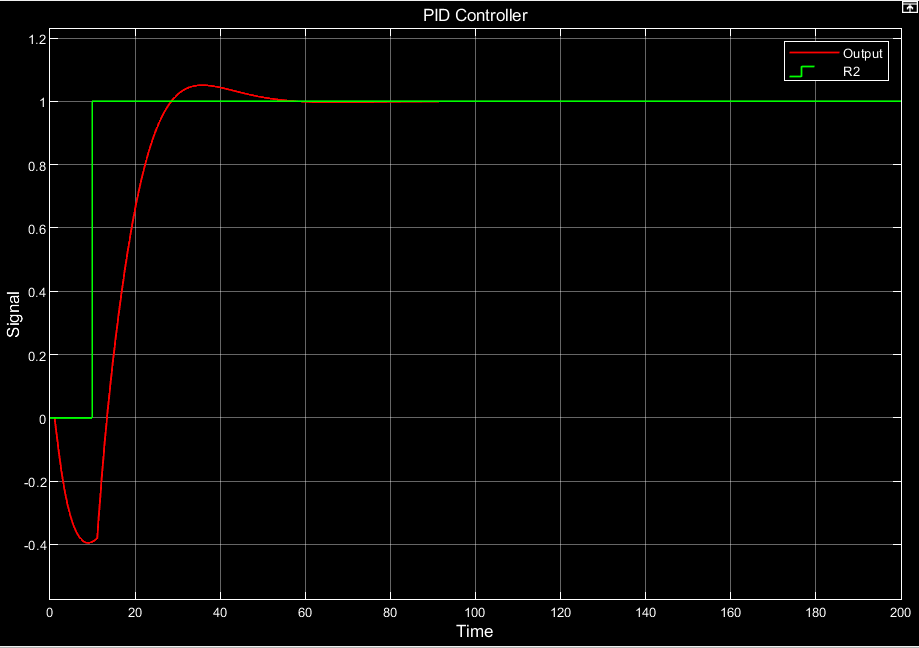


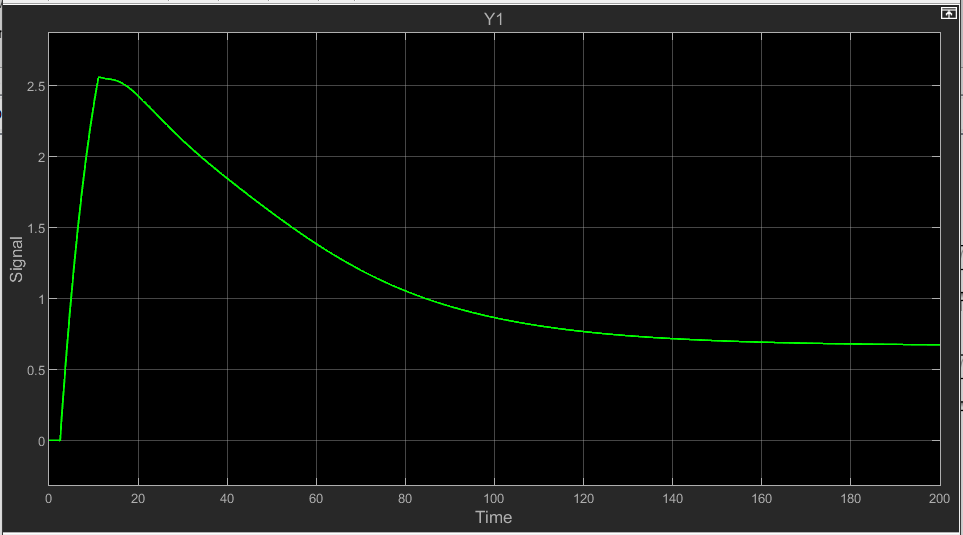




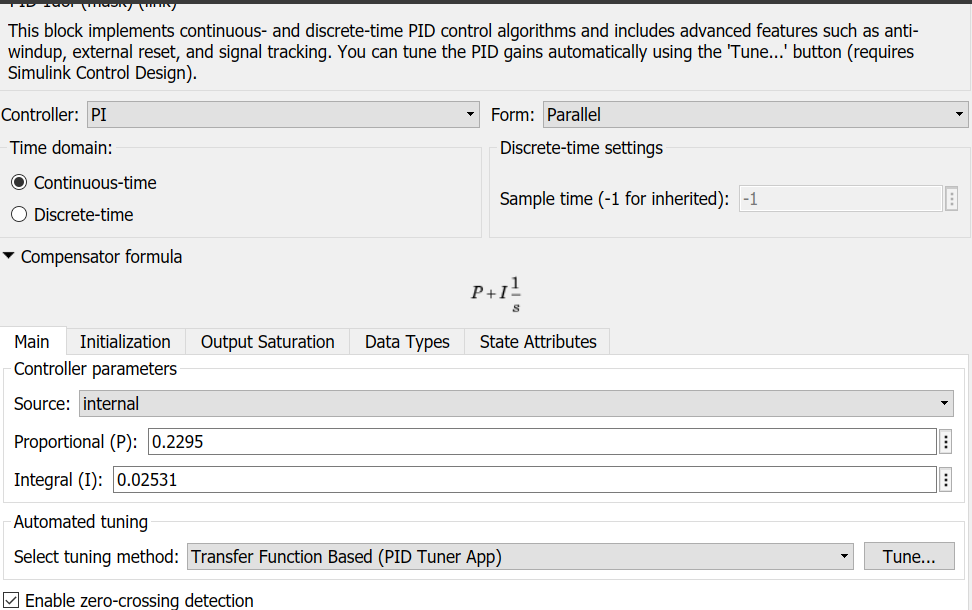


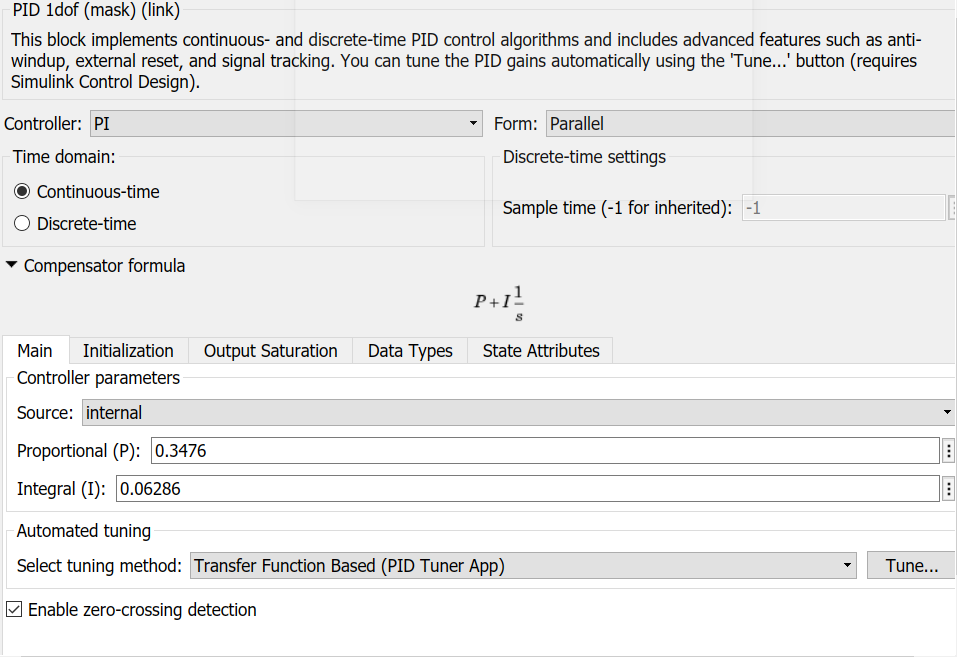
PID Controller

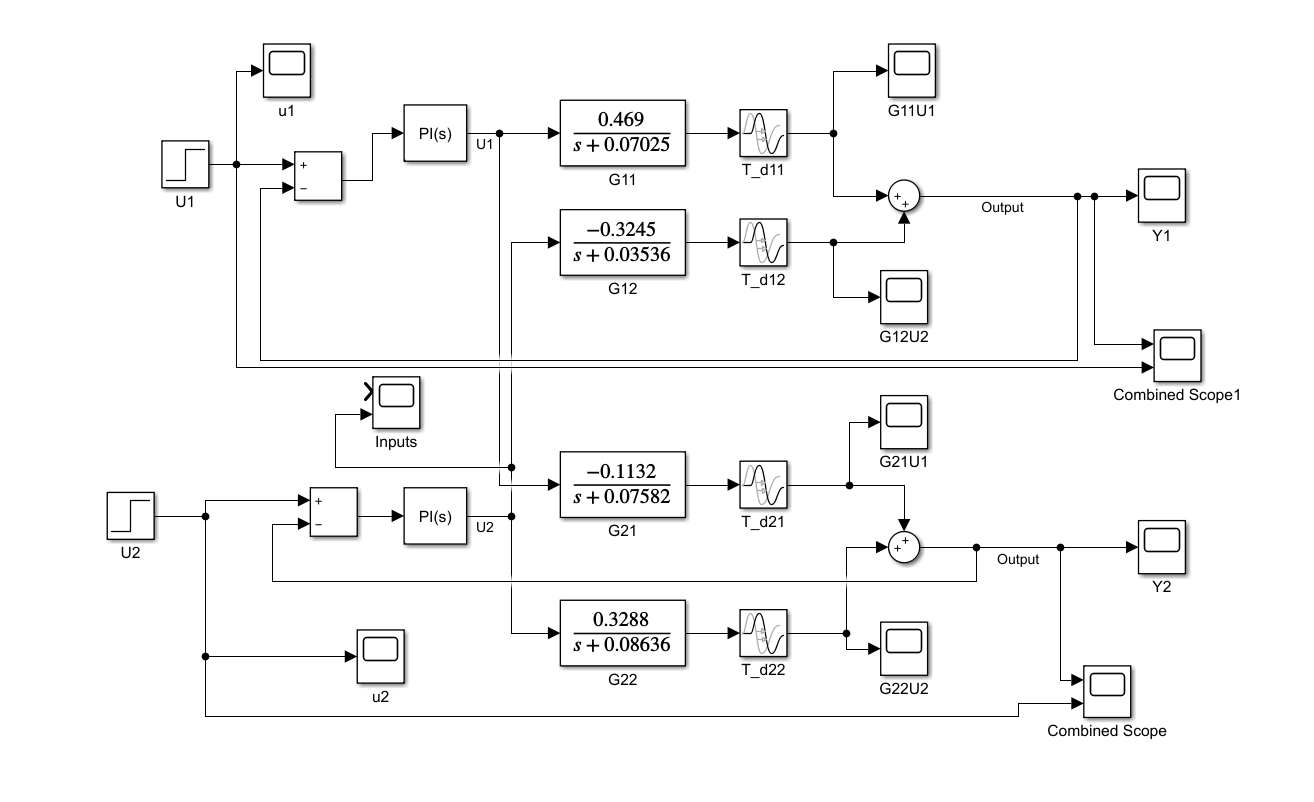


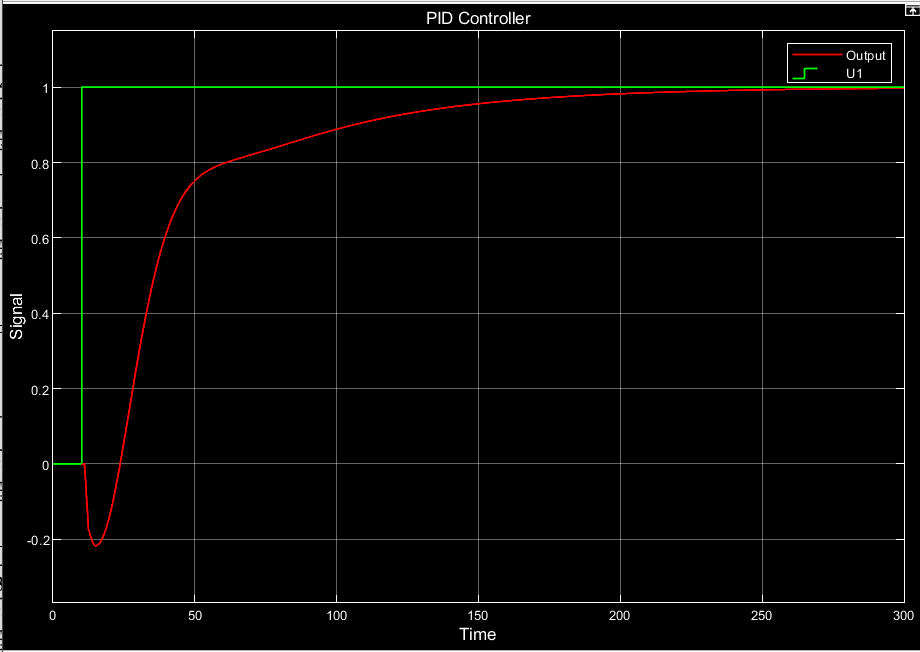


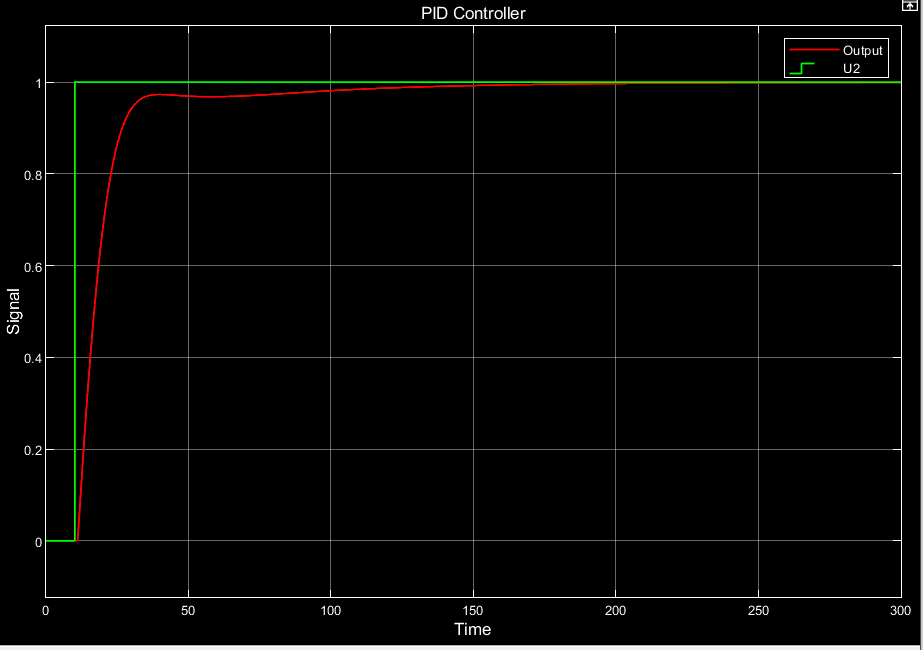
### **5: MIMO both loops active**

PI Controller:

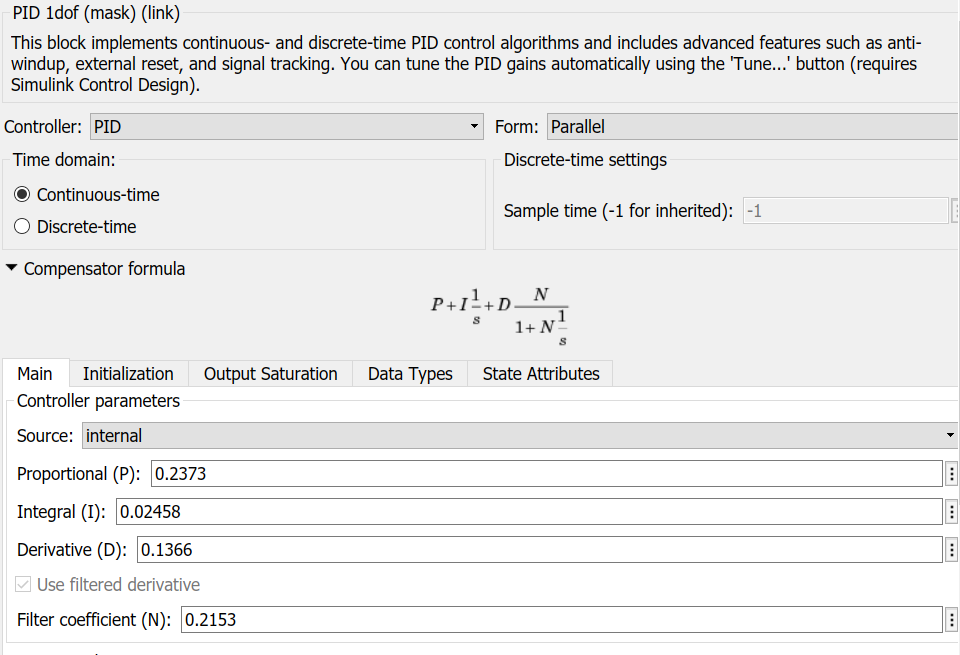


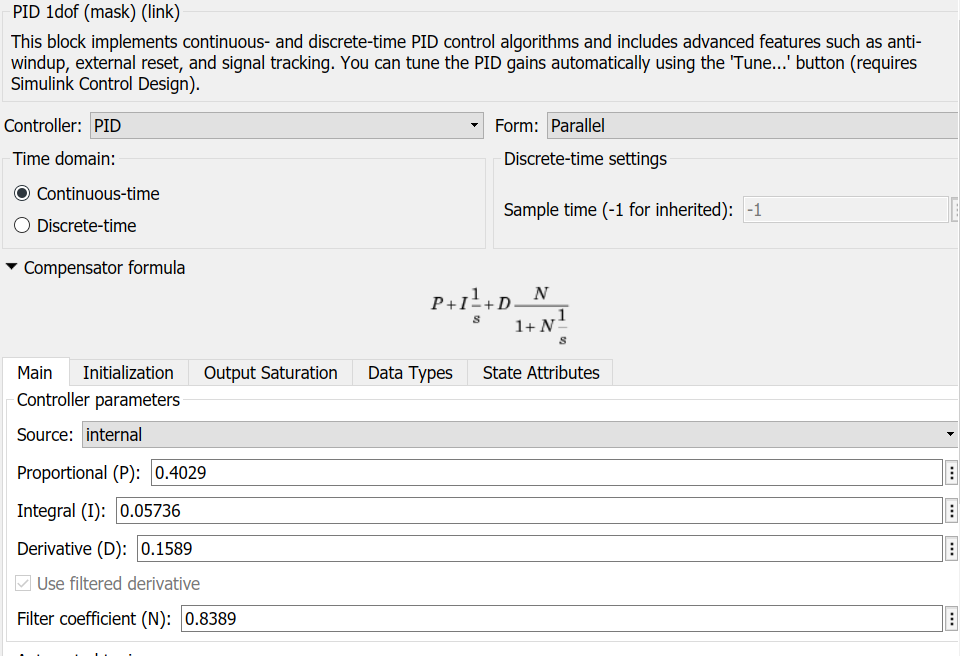


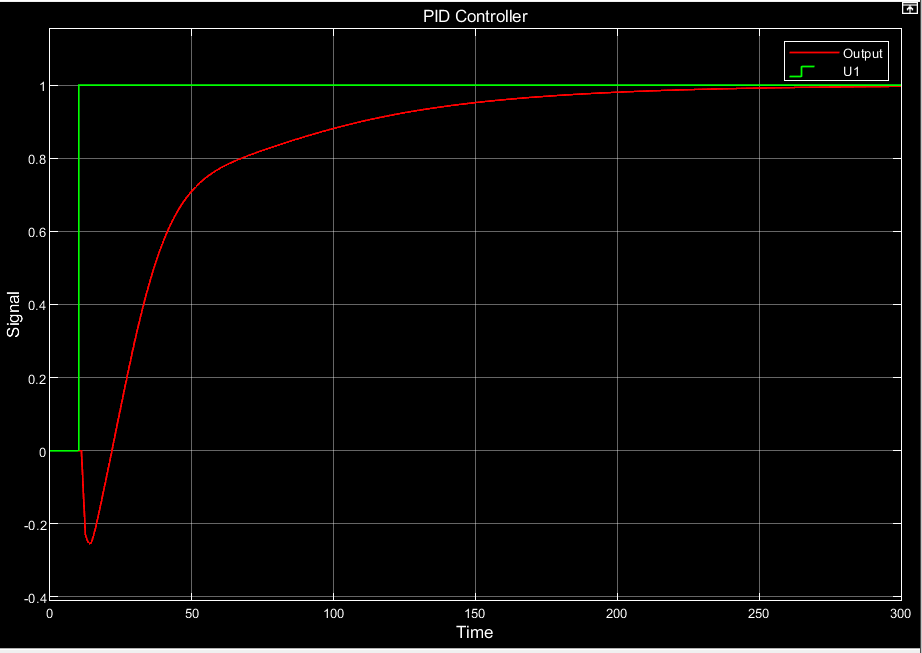


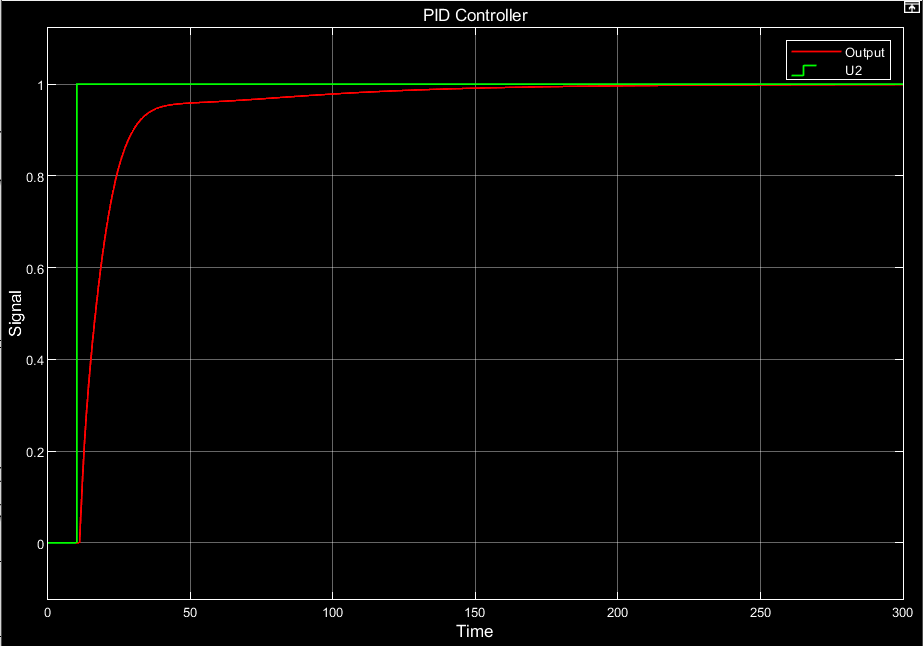


PID Controller:









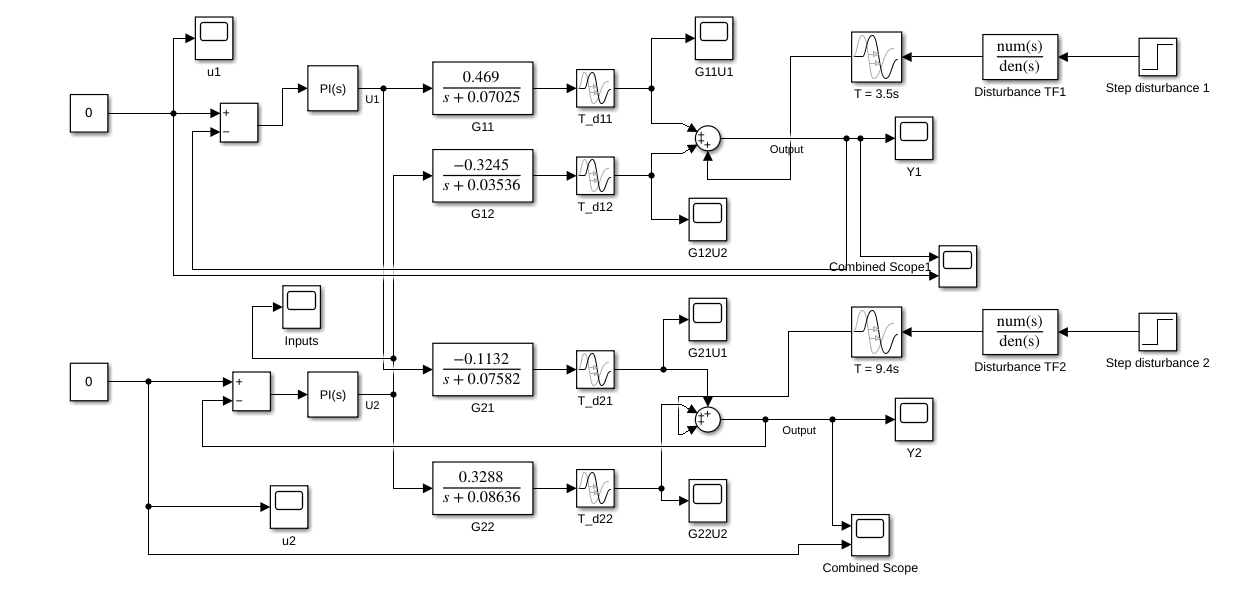
## **Part c) Effect of Disturbances**

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.

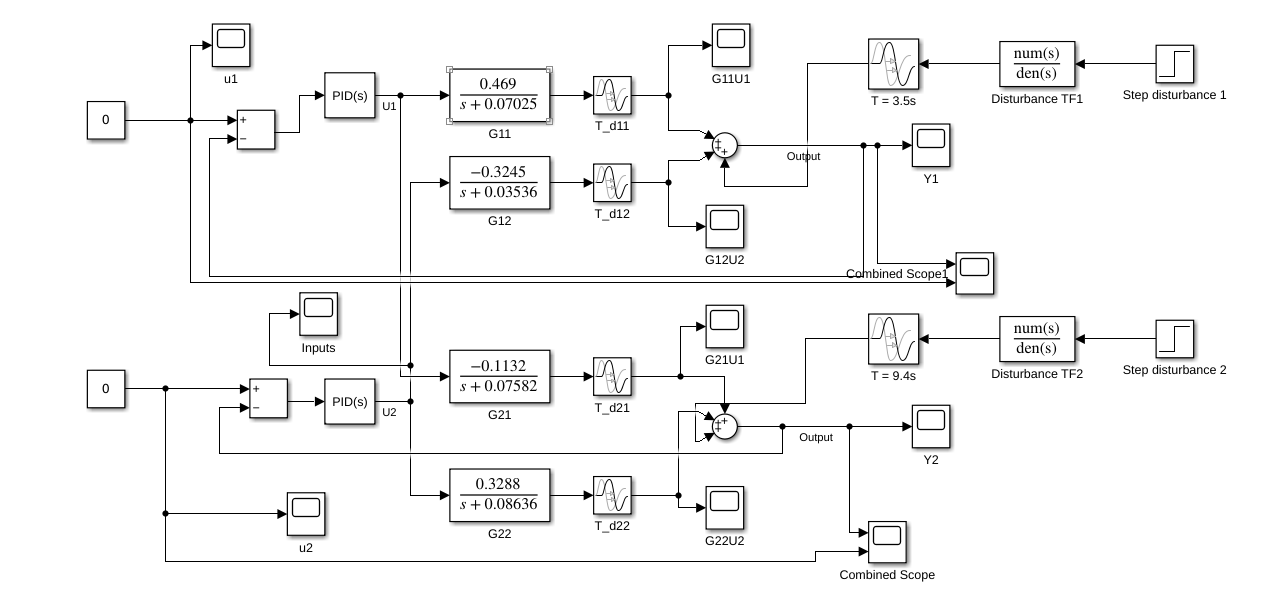
### **1: Step disturbance**

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:

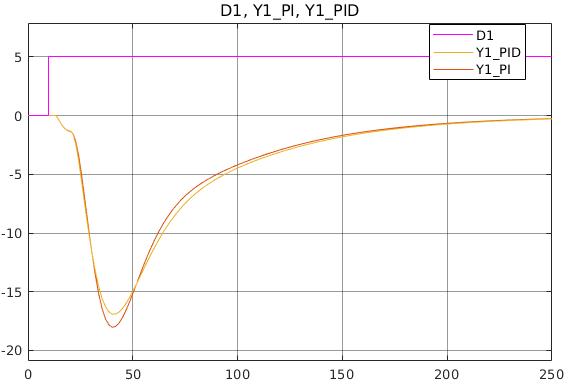


The SIMULINK model with PID controllers is as follows:

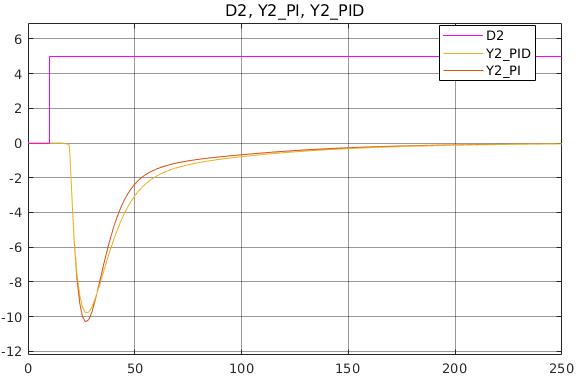


Results:

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

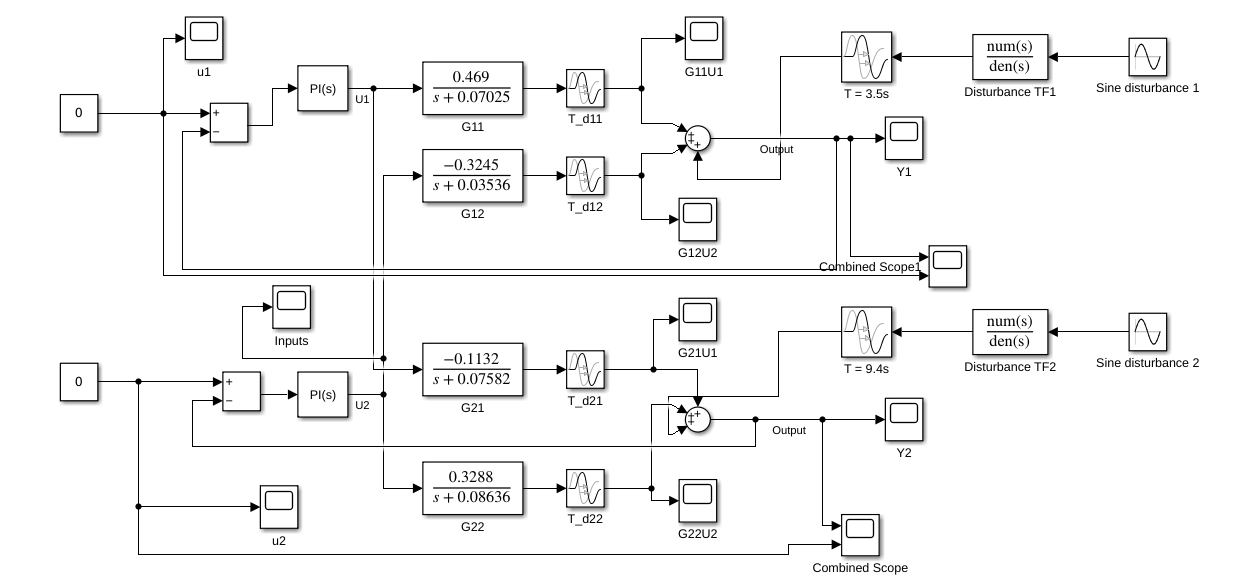
advantages over each other. Performance in set-point tracking may still be improved by modelling the controllers after taking into account the coupling interactions. Moreover,

performance in disturbance rejection may also be improved under the assumption that the disturbance model is known beforehand, and that the disturbance is measured.

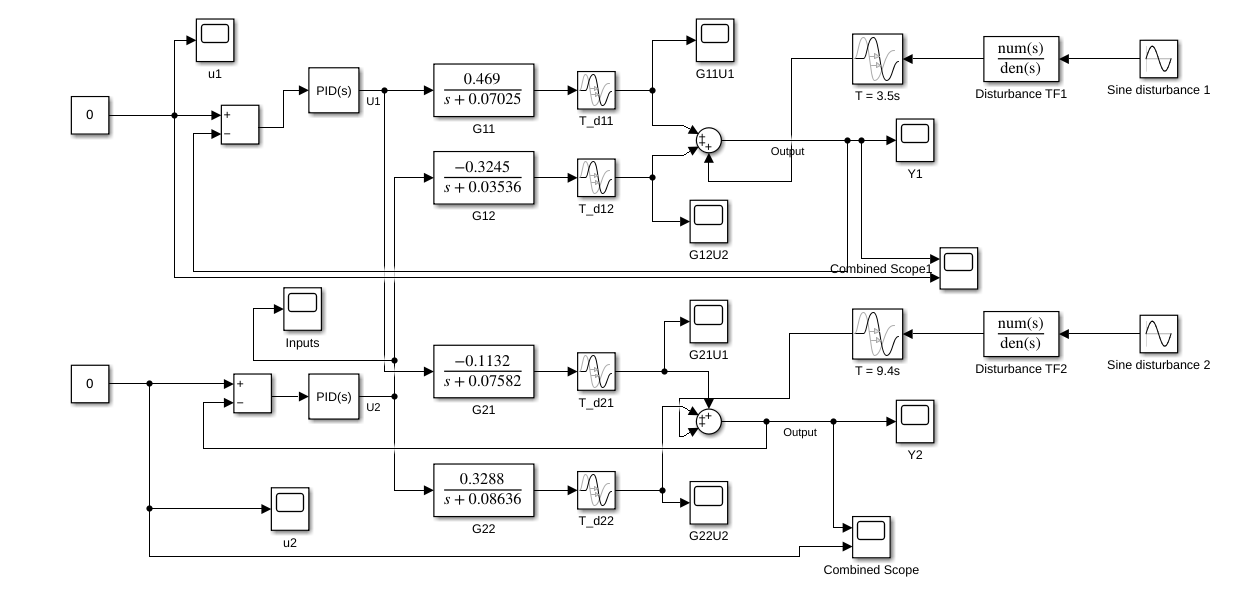
### **2: Sinusoidal disturbance**

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:



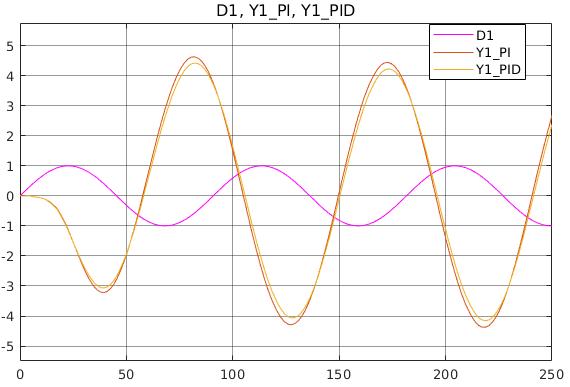
It is not expected that either the PI controller or the PID controller setup described above would

be capable of complete disturbance rejection, in which case the steady-state output response

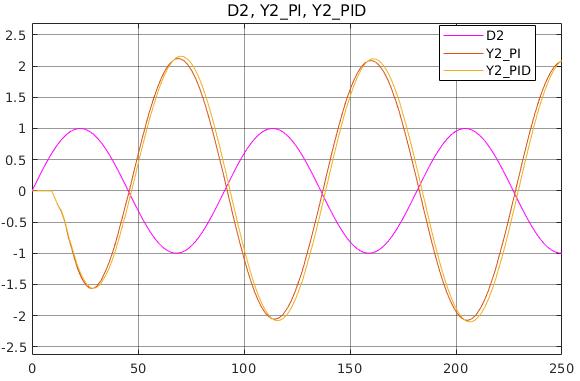
would be 0 in the absence of a setpoint change. This is because, by the Internal Model Principle, in order for complete disturbance rejection, the generating polynomial of the transfer function must appear in the denominator of the controller transfer function. This is not the case for either the PI 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:



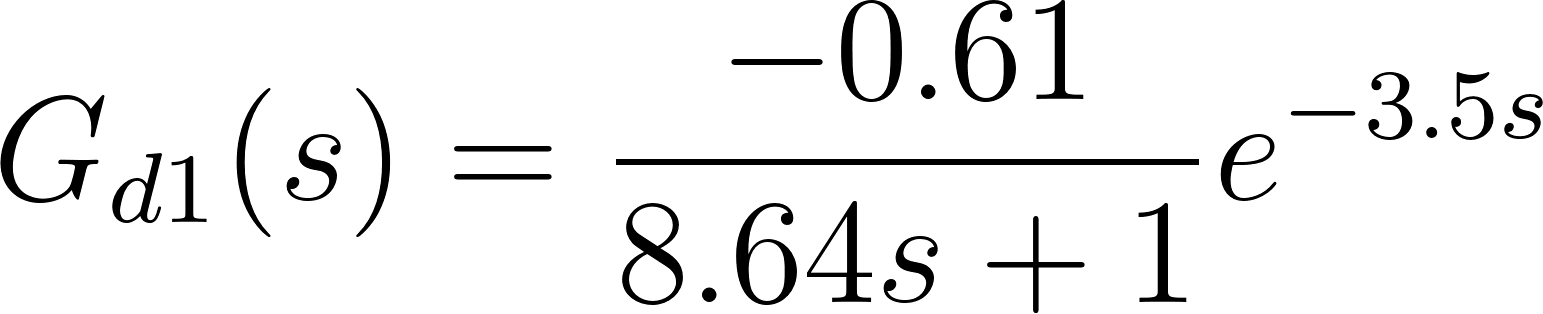
Simulation results for Y2:



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:

[](https://www.codecogs.com/eqnedit.php?latex=G_%7Bd1%7D(s)%20%3D%20%5Cfrac%7B-0.61%7D%7B8.64s%2B1%7De%5E%7B-3.5s%7D#0)

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

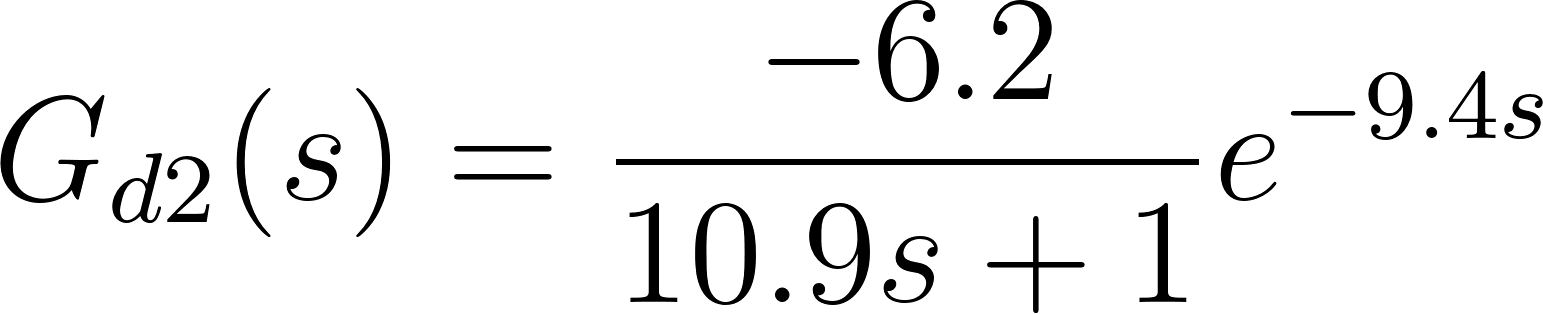
Y1 is of a larger amplitude ratio than without the presence of a controller. A more aggressive

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:

[](https://www.codecogs.com/eqnedit.php?latex=G_%7Bd2%7D(s)%3D%5Cfrac%7B-6.2%7D%7B10.9s%2B1%7De%5E%7B-9.4s%7D#0)

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 ( τC ≥ θ ). The utilised controller is such that τC = θ . Therefore, a different approach may be required to improve disturbance rejection.

One such possible approach is to additionally implement a feedforward controller capable of

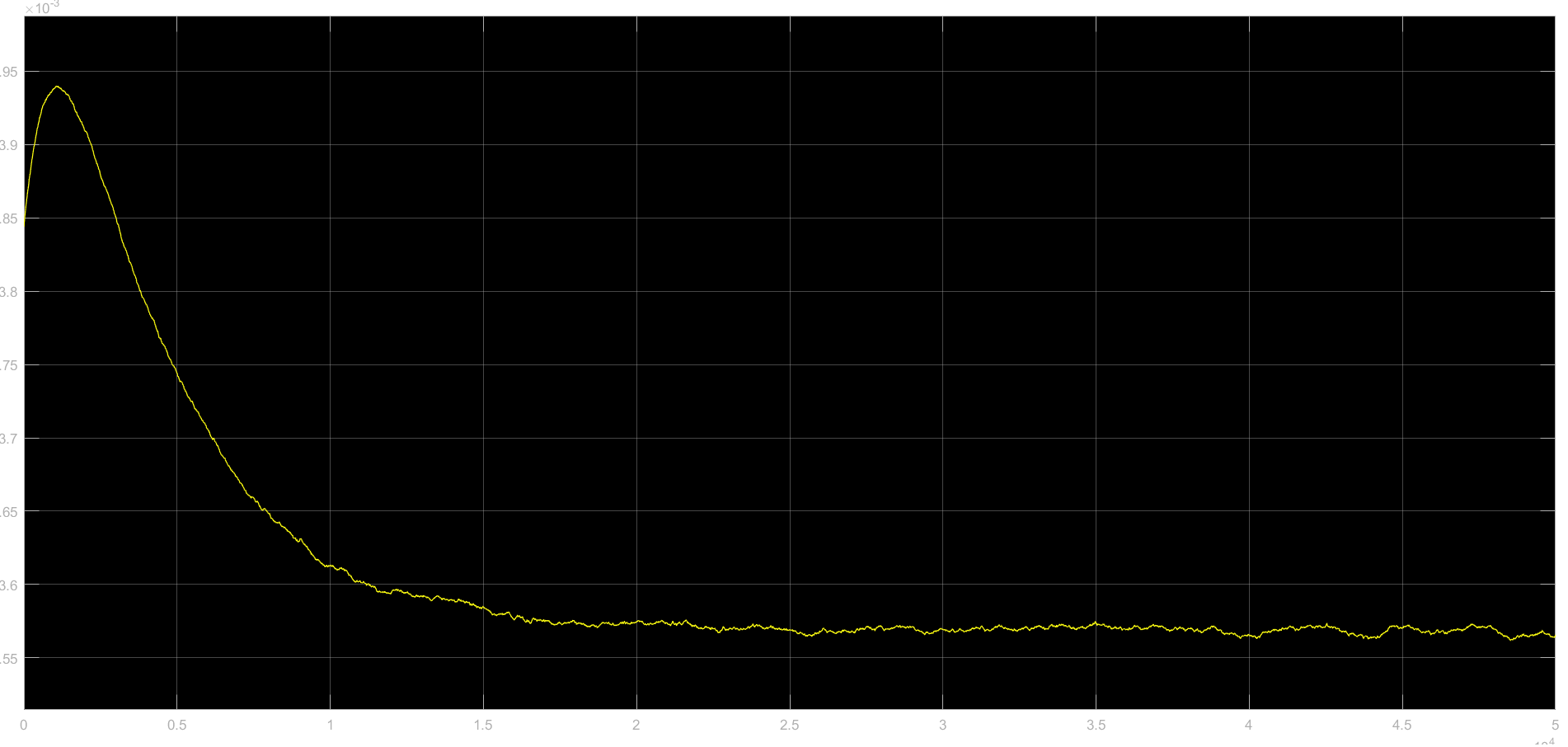
rejecting the disturbance, if the disturbance model is known beforehand. However, the

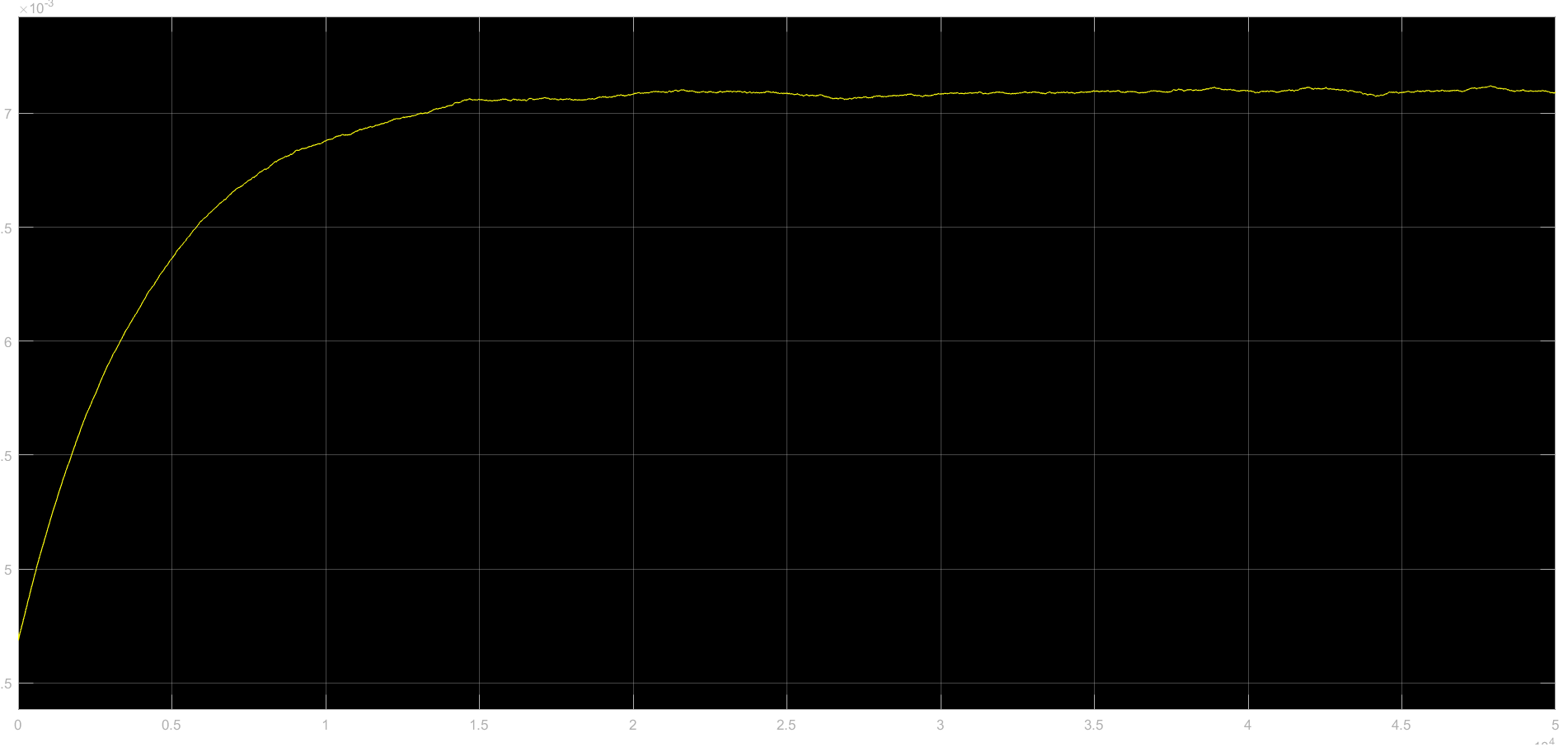
implemented feedback controller is still required to be present to ensure that set-point tracking is achieved.

# **Question 2: FCC System**

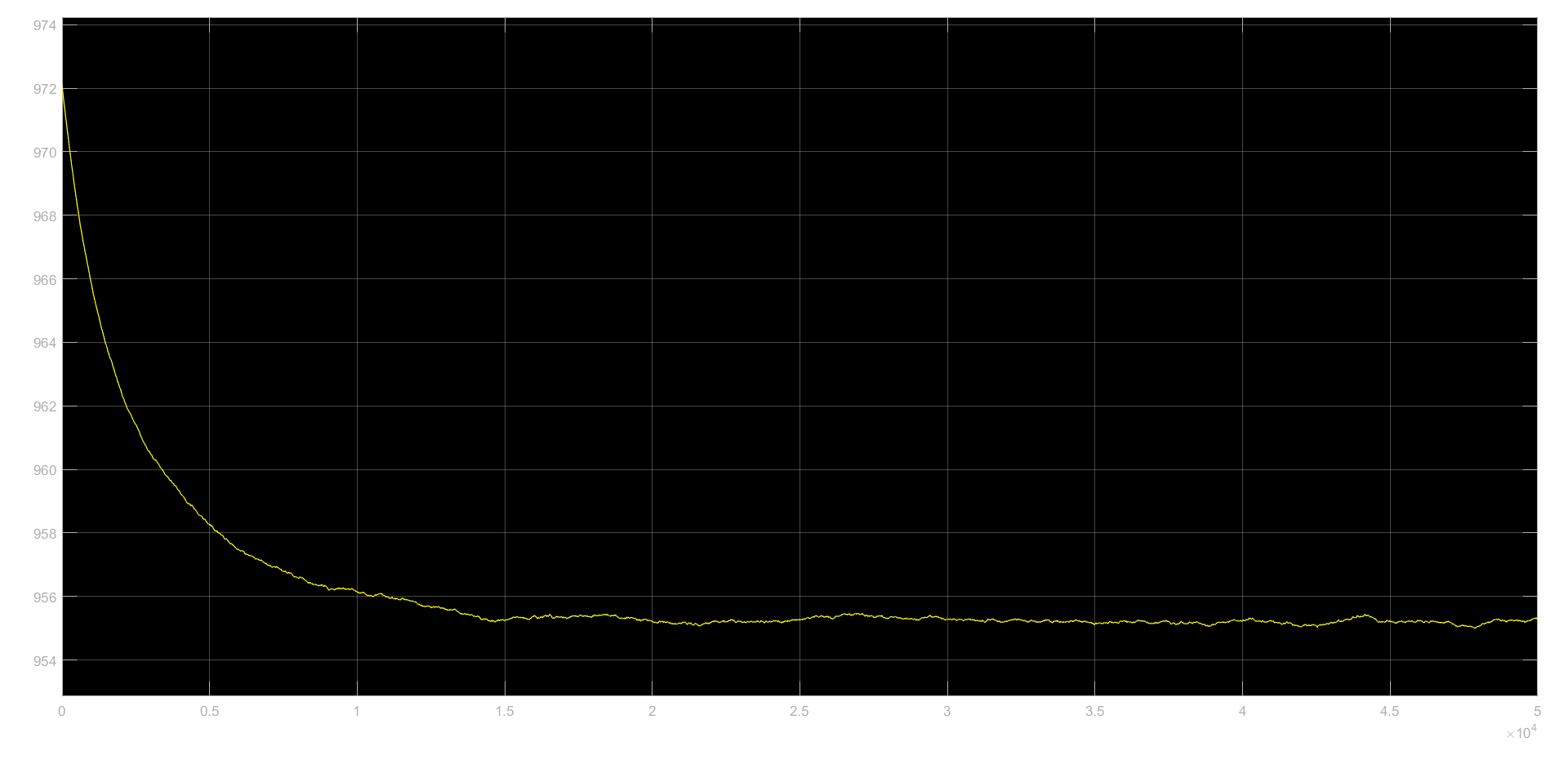
## **Part a) State-Space Model Estimation**

Actual Y1:

Actual Y2:



Actual Y3:



Discrete-time identified state-space model:

x(t+Ts) = 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 0.7561 -0.02677 0.05401 -0.2974 -0.8088

x2 -0.2134 0.9758 0.04676 -0.2486 -0.7092

x3 0.002332 0.003604 1.001 -0.05336 0.03338

x4 -0.004833 0.02551 0.01833 0.5774 0.06777

x5 -0.005665 0.006667 0.01871 0.2931 -0.3222

B =

u1 u2

x1 28.17 -2.122

x2 24.69 -1.863

x3 -1.009 0.08644

x4 -1.74 0.2347

x5 39.46 -2.726

C =

x1 x2 x3 x4 x5

y1 -0.256 0.2927 0.03082 0.004464 0.0005535

y2 -0.4895 0.5578 -0.09152 0.02745 -0.000622

y3 -6.778e+04 7.746e+04 3647 1741 99.7

D =

u1 u2

y1 0.0001019 1.036e-05

y2 -0.0003632 -3.706e-05

y3 0.005963 0.000483

K =

y1 y2 y3

x1 0 0 0

x2 0 0 0

x3 0 0 0

x4 0 0 0

x5 0 0 0

Sample time: 0.25 seconds

Parameterization:

FREE form (all coefficients in A, B, C free).

Feedthrough: yes

Disturbance component: none

Number of free coefficients: 56

Use "idssdata", "getpvec", "getcov" for parameters and their uncertainties.

Status:

Estimated using N4SID on time domain data "data".

Fit to estimation data: [-522.9;-287.4;78.76]%

FPE: 3.225e-15, MSE: 0.2745

## **Part b) Examine Open Loop Stability**

poles (found using eigenvalues of A) =

poles =

-16.5374 + 0.0000i

-1.5747 + 0.3913i

-1.5747 - 0.3913i

-0.0000 + 0.0000i

-0.0002 + 0.0000i

stability =

logical

1

% We realise that all poles are within the unit disk and are real part < 0

isstable(sys) confirms in MATLAB. System is stable

## **Part c) Convert to tf, SIMULINK block, verification**

From input "u1" to output...

0.001215 s^5 - 0.004067 s^4 - 0.01493 s^3 - 0.01261 s^2 - 2.613e-05 s + 6.109e-10

y1: ---------------------------------------------------------------------------------

s^5 + 19.69 s^4 + 54.72 s^3 + 43.55 s^2 + 0.008086 s + 3.458e-09

-0.003676 s^5 + 0.00962 s^4 + 0.05423 s^3 + 0.01402 s^2 + 0.000131 s + 6.025e-10

y2: --------------------------------------------------------------------------------

s^5 + 19.69 s^4 + 54.72 s^3 + 43.55 s^2 + 0.008086 s + 3.458e-09

0.07048 s^5 - 0.2011 s^4 - 1.168 s^3 - 0.2985 s^2 - 1.082 s + 0.000145

y3: ----------------------------------------------------------------------

s^5 + 19.69 s^4 + 54.72 s^3 + 43.55 s^2 + 0.008086 s + 3.458e-09

From input "u2" to output...

-7.081e-05 s^5 + 0.0005978 s^4 + 0.002403 s^3 + 0.002115 s^2 + 1.595e-06 s - 6.162e-11

y1: --------------------------------------------------------------------------------------

s^5 + 19.69 s^4 + 54.72 s^3 + 43.55 s^2 + 0.008086 s + 3.458e-09

0.0002044 s^5 - 0.001786 s^4 - 0.008173 s^3 - 0.008443 s^2 - 8.976e-06 s - 5.856e-11

y2: ------------------------------------------------------------------------------------

s^5 + 19.69 s^4 + 54.72 s^3 + 43.55 s^2 + 0.008086 s + 3.458e-09

-0.004429 s^5 + 0.0272 s^4 + 0.1733 s^3 + 0.2256 s^2 + 0.02719 s - 1.455e-05

y3: ----------------------------------------------------------------------------

s^5 + 19.69 s^4 + 54.72 s^3 + 43.55 s^2 + 0.008086 s + 3.458e-09

Continuous-time transfer function.

<insert images>

## **Part d) Controller tuning and Closed Loop stability**

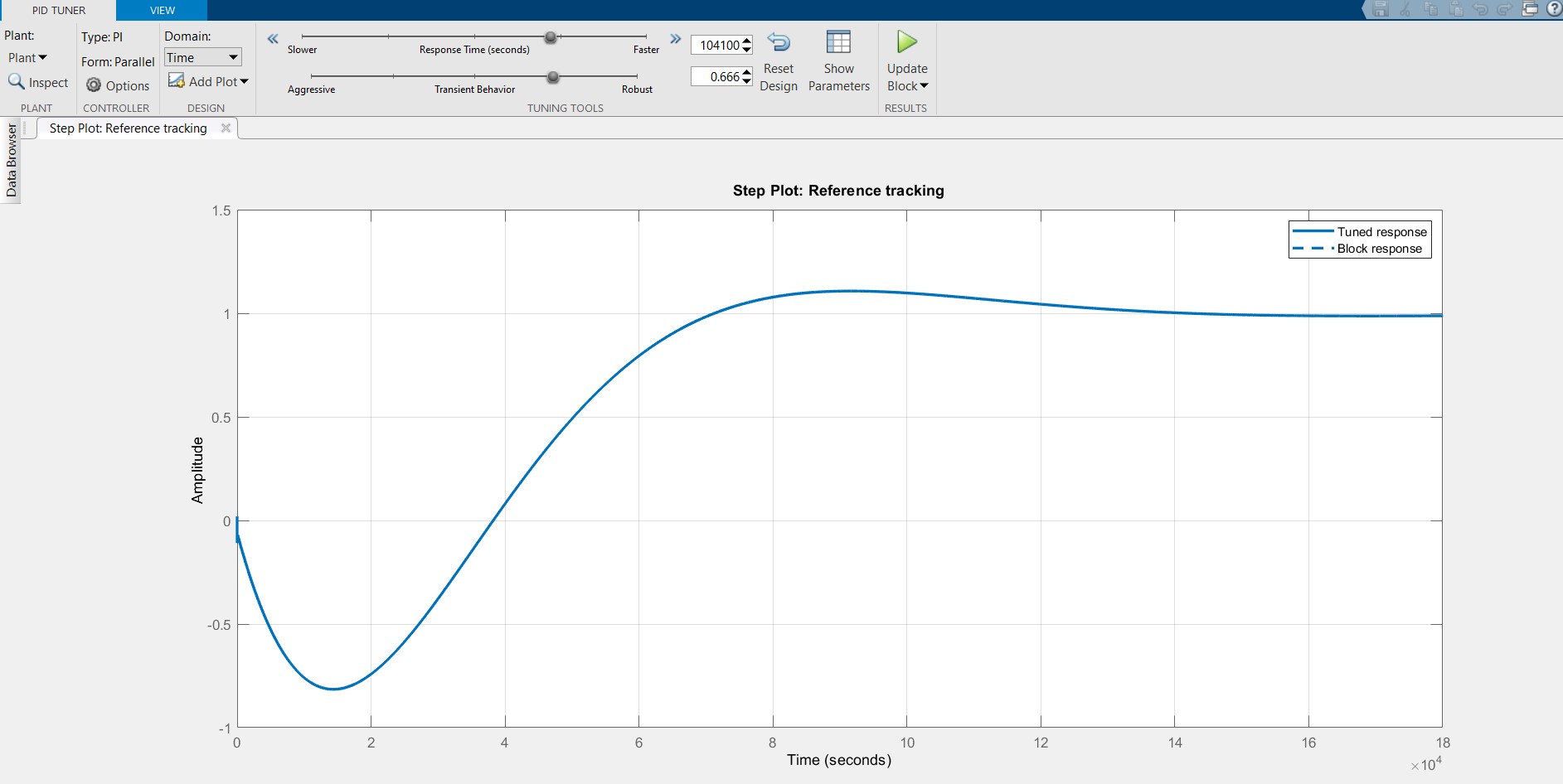
### PI tuning

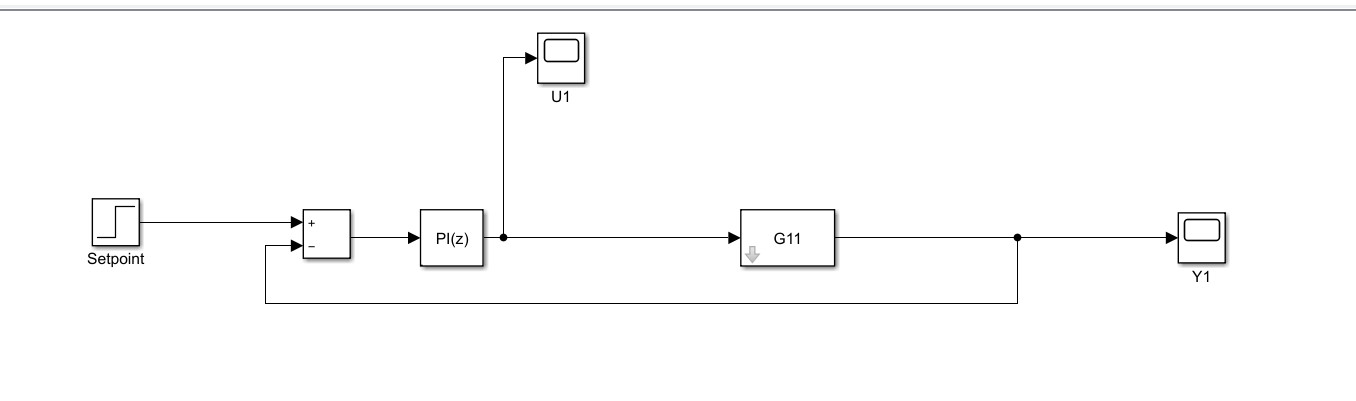
Tuned using Auto-Tuner.

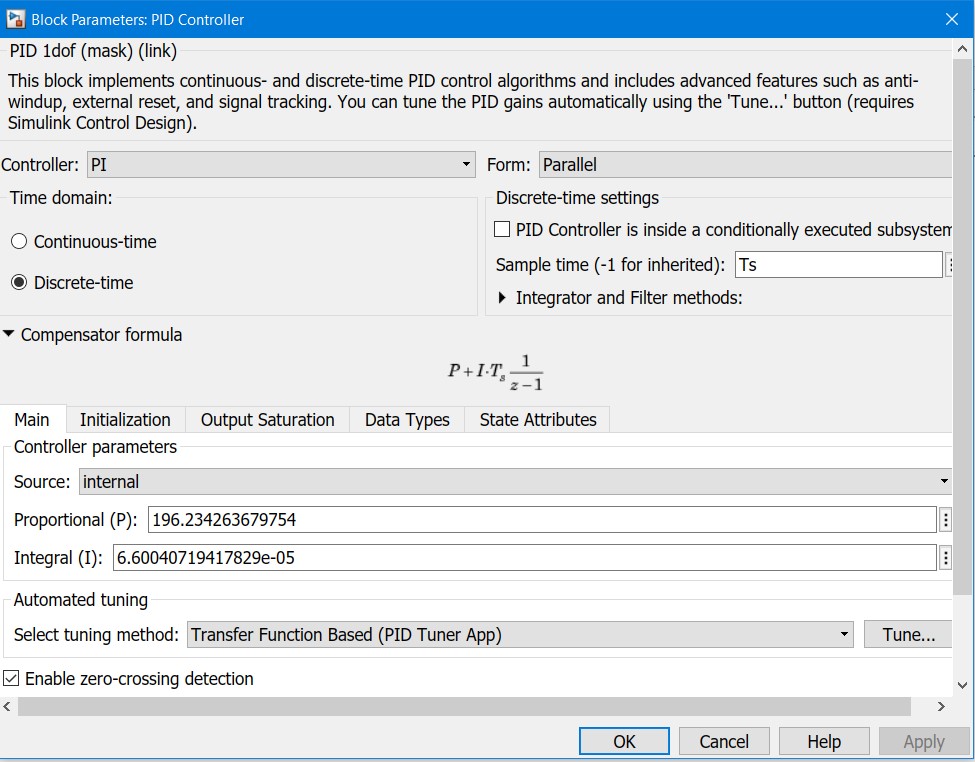
Criteria: Response time (slow vs fast) and Transient behaviour (Aggressive vs Robust)

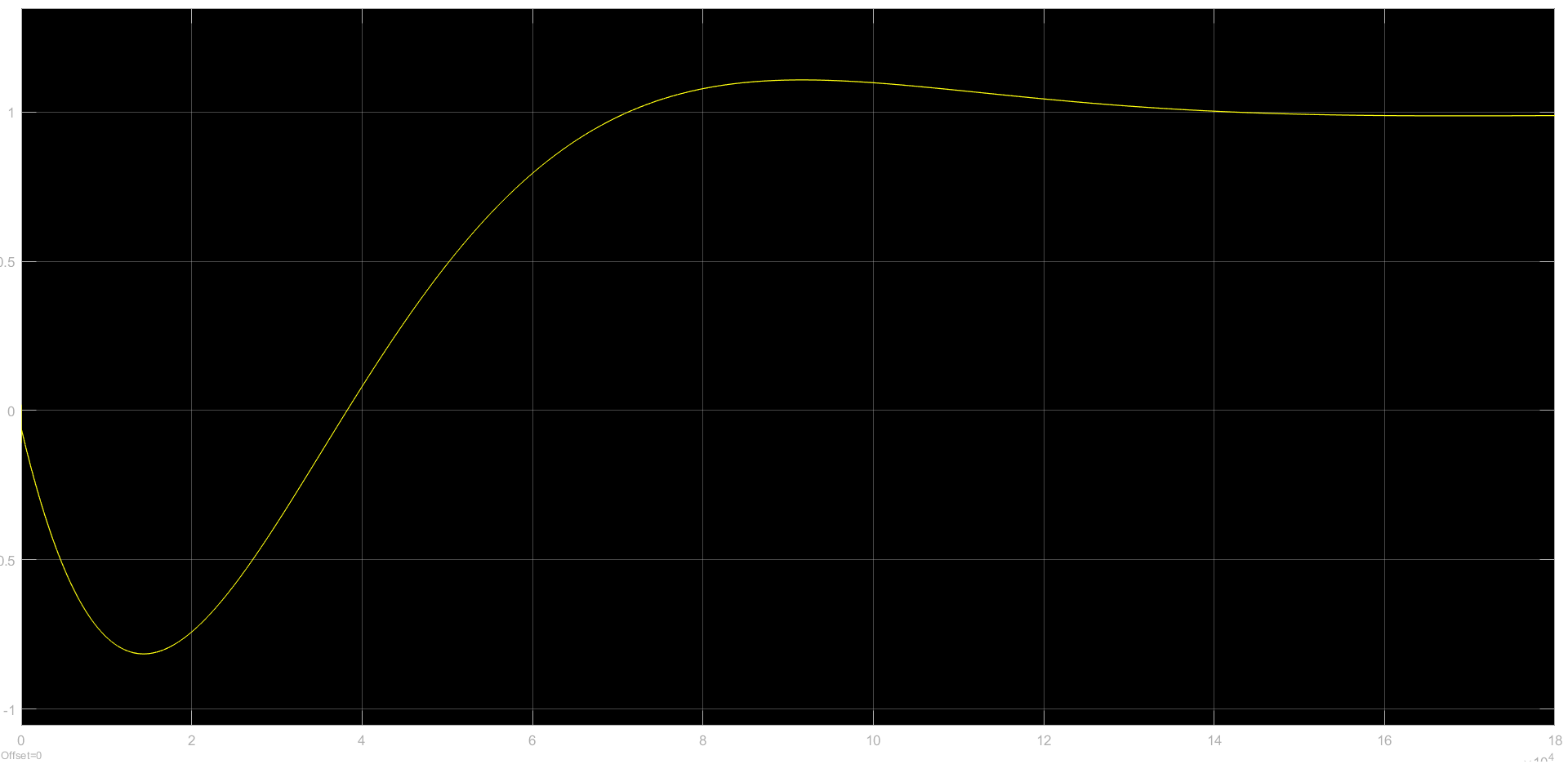
Opted for a faster response time and robust transient behaviour

Jump at t=0 because G has number of zeros = number of poles.









### **Routh Hurwitz criterion**

1.218 s^6 + 18.96 s^5 + 52.04 s^4 + 41.29 s^3 + 0.003406 s^2 + 1.115e-07 s + 3.185e-14

Can you do the method for this^?

### Root Locus analysis.

### Bode Stability Criterion and the Nyquist Stability Criterion.

## **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.

ie Denominator of transfer function form is 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 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 lot of variations for a long time (looks noisy in the end).

## **Part f) Effect of Time Delay**