ASSIGNMENT

Transfer function for industrial system

$$G(s) = \frac{(s+1)}{s(s^2+4s+5)} = \frac{(s+1)}{s^3+4s^2+5s}$$

CASE I; Controller 1:

- Overshoot less than 5%
- Rise time less than 1 seconds
- Settling time of 2% error less than 4 seconds

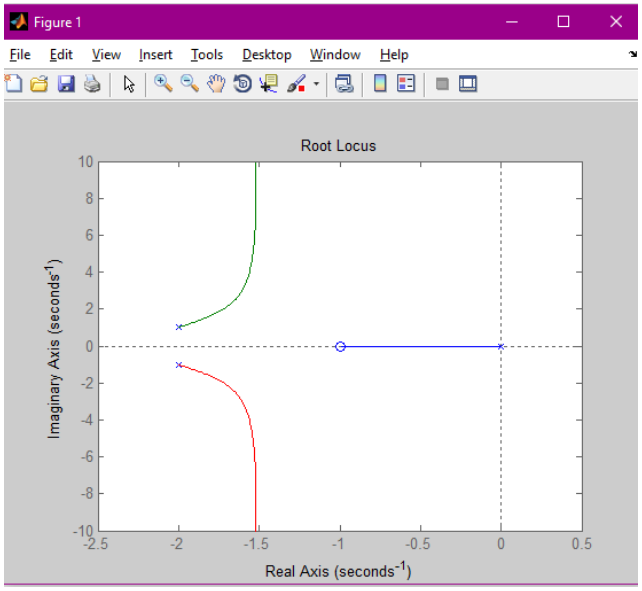
Using root locus criteria, we create from command window the transfer function for G

```
s = tf('s');
```

```
G = (s+1)/(s^3+4*s^2+5*s)
```

And see its root locus chart and poles typing the commands:

```
rlocus(G)
```




```
poles=pole(G)
```

```
ans =
```

```
0.0000 + 0.0000i
```

```
-2.0000 + 1.0000i
```

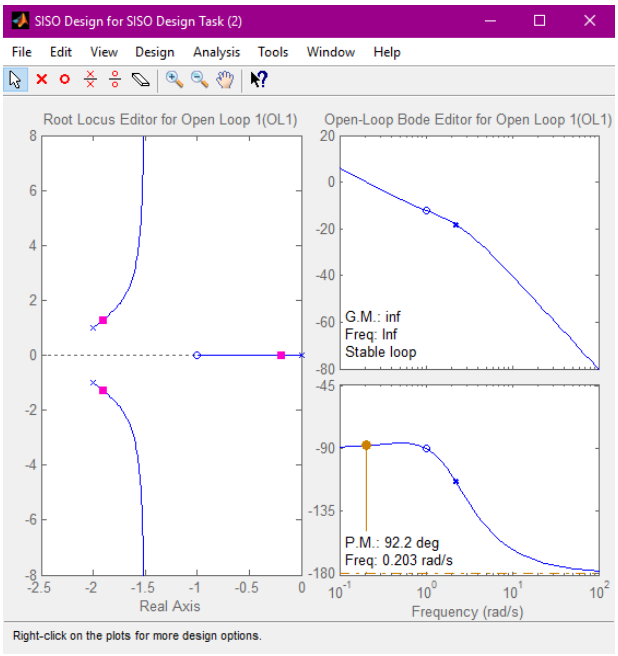
```
-2.0000 - 1.0000i
```

Here we can see that all poles are located to left semi plane, that makes the system marginally stable. The main idea of root locus design is to predict the closed-loop response from the root locus plot which depicts possible closed-loop pole locations and is drawn from the open-loop transfer function. Then by adding zeros and/or poles via the controller, the root locus can be modified in order to achieve a desired closed-loop response.

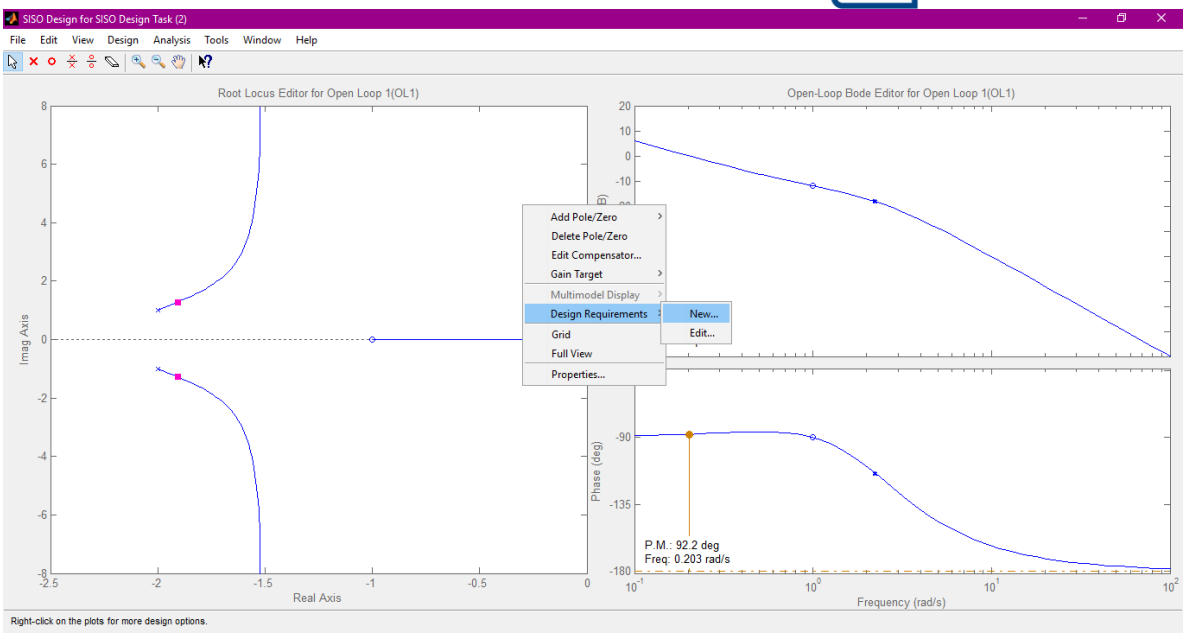
In general, the real part of a pole indicates how quickly the transient portion of the corresponding mode decays to zero (assuming negative real part). Therefore, if you have a transfer function which has one (or more) poles much farther to the left in the complex plane (more negative) than the other poles, their effect on the dynamic response will be hidden by the slower, more dominant poles.

Now we can start other matlab tool called sisotool typing the command

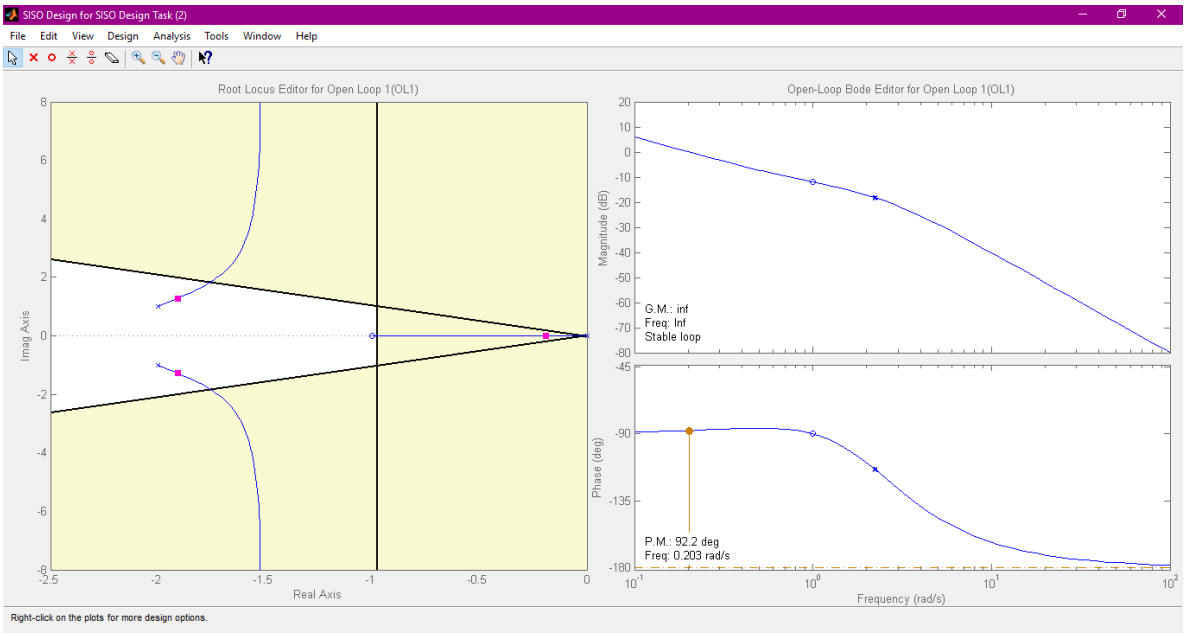
Sisotool(G)



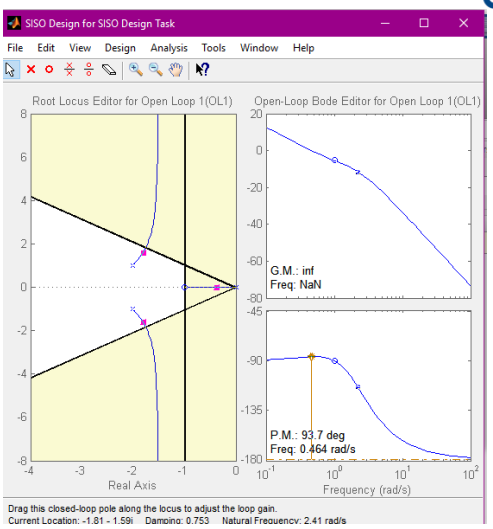
Here we can indicate some of design criteria as follows.



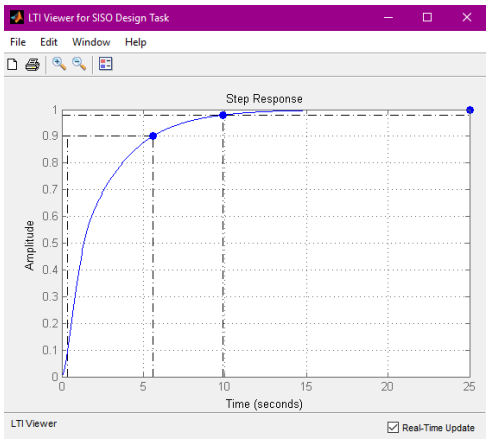
After indicate overshoot less than 5% and stting time less than 4s, we'll see this



Took close loop poles (pink) and move it slowly to black lines, this give this situation



The step response would be



This means, that we don't have overshoot, but while other close loop pole is behind vertical line we can't achieve the requirement for settling time less than 4 s.

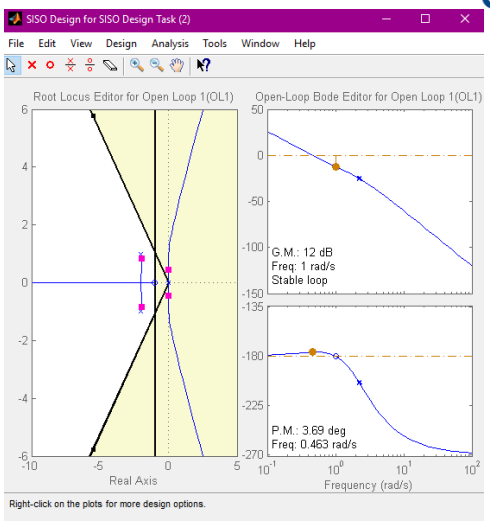
INTEGRAL CONTROL

so first we have to test with integral control.

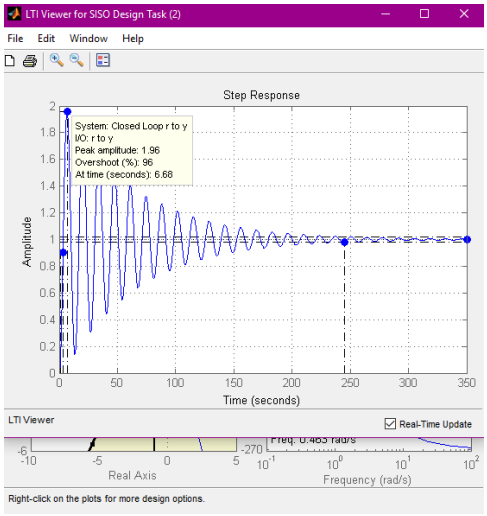
We need an integrator in the controller (not just in the system) to remove the steady-state error due to a constant disturbance.

C = 1/s;

>> sisotool(C*G)



As far as can we saw, the system never will be stable with only integral control, let´s see its transient response.



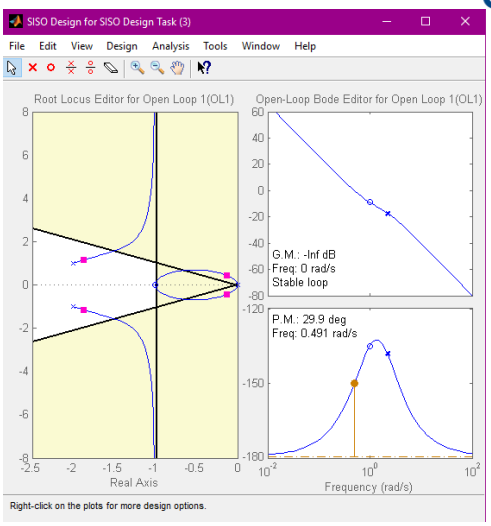
Here is possible observe that the system never will meet the requirements with only integral control, the transient response would be undamped. Therefore we need other control type.

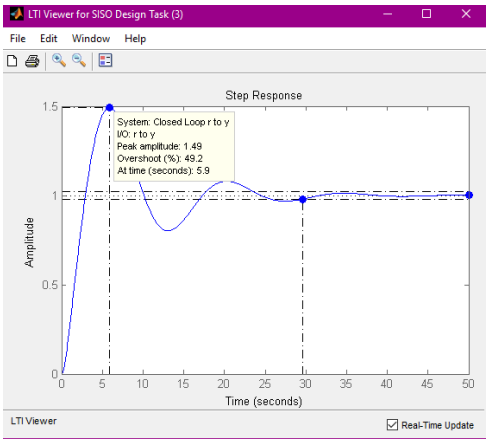
PI CONTROL

Now, let's modify the integral controller to a PI controller. Using PI instead of I control adds a zero to the open-loop system. We'll place this zero at *s* = -1. The zero must lie between the open-loop poles of the system in this case so that the closed-loop system will be stable.

C = (s + 1) / s;

>> sisotool(C*G)





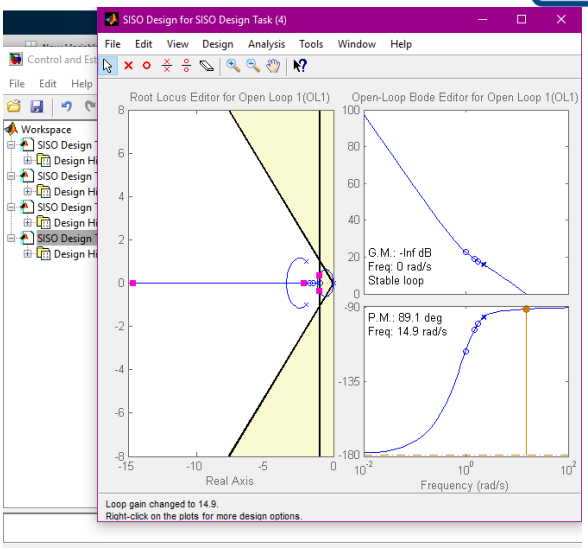
We have managed to stabilize the system and achieve zero steady-state error to a constant disturbance, but the system is still not fast enough and its overshoot is high.

PID CONTROL

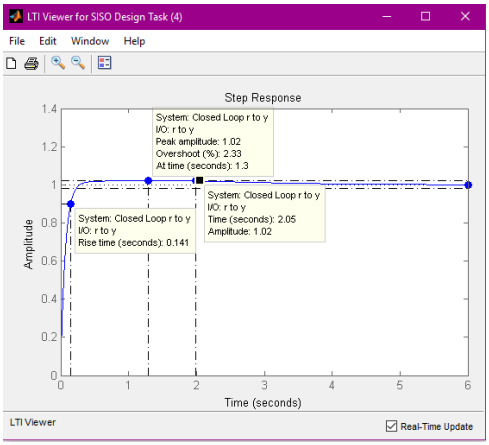
In order to pull the root locus further to the left, to make it faster, we need to place a second open-loop zero, resulting in a PID controller. After some experimentation, we place the two PID zeros at *s* = -1.5 and *s* = -1.75. Change the lines defining the controller in your m-file to the following. Re-run your m-file and you will generate a plot like the one shown below.

C = (s + 1.5)*(s + 1.75) / s;

>> sisotool(C*G)



Here we take poles close loop (pink) and move them very close to vertical line to make the response a little bit faster. The transient response is:



And with this criteria we achieve our designs requirements. The final controller is:

C = 14.9*(s + 1.5)*(s + 1.75) / s;

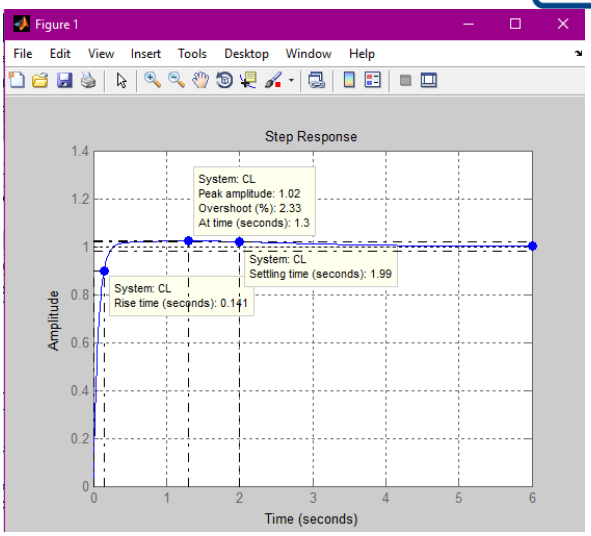
CL=feedback(C*G,1)

CL =

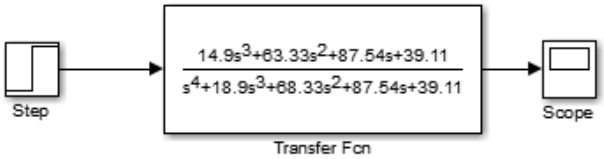
$$\frac{14.9\,s^3 + 63.32\,s^2 + 87.54\,s + 39.11}{s^4 + 18.9\,s^3 + 68.32\,s^2 + 87.54\,s + 39.11}$$

Continuous-time transfer function.

>> step(CL)



To implement this control in simulink, is possible use a transfer function block and set the data for transfer function close loop, as follows

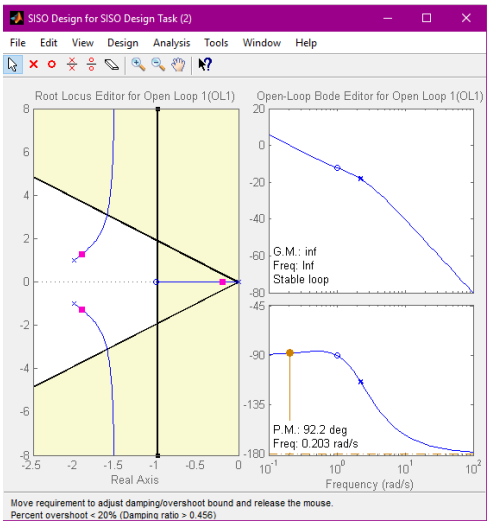


And the scope plot is:

CASE II; Controller 2:

- Overshoot less than 20%
- Rise time less than 0.5 seconds
- Steady-state error less than 5%.

We going to follow the same steps used in case i.

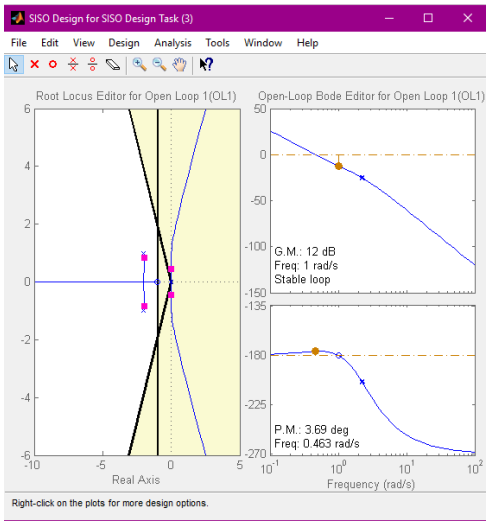


This would be the root locus for G conditioning the shades for 20% overshot and 4s of settling time.

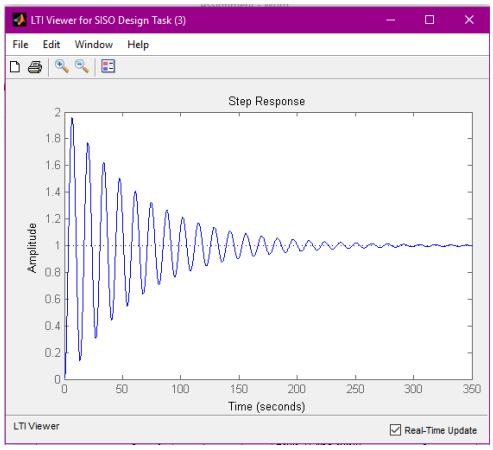
INTEGRAL CONTROL

I = 1/s;

>> sisotool(I*G)



As we could saw in case I, a pure integral control is not recommended because the system is not stable. Let´s confirm this with transient response

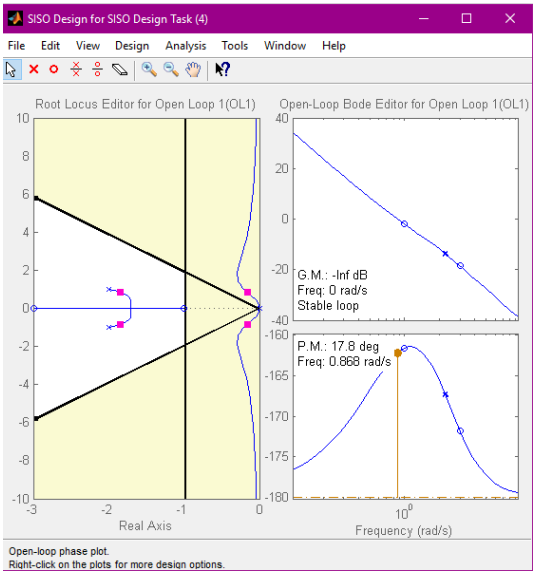


As we can see, the system in unstable, the response is undamped.

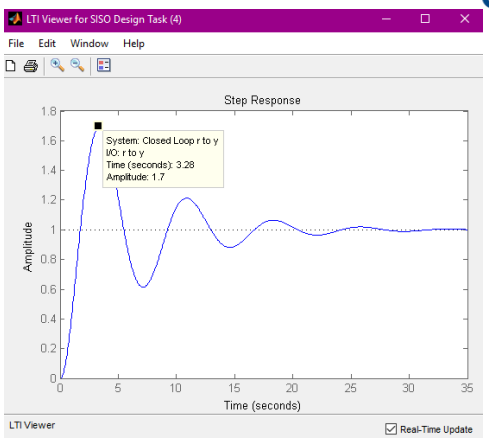
PI CONTROL

C = (s + 3) / s;

>> sisotool(C*G)



The response is better than pure integral control, but the overshot is high and the response is slow, lets confirm this with transient chart.

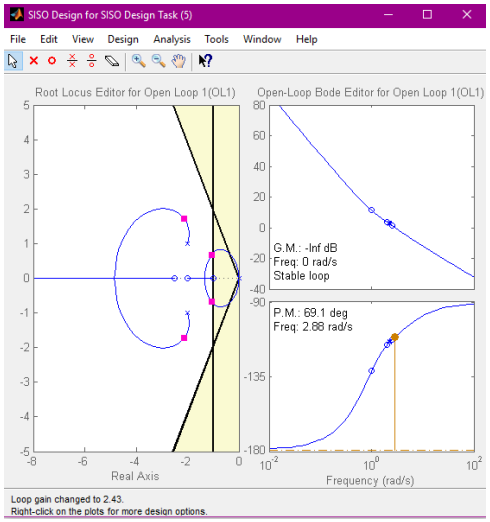


PID CONTROL

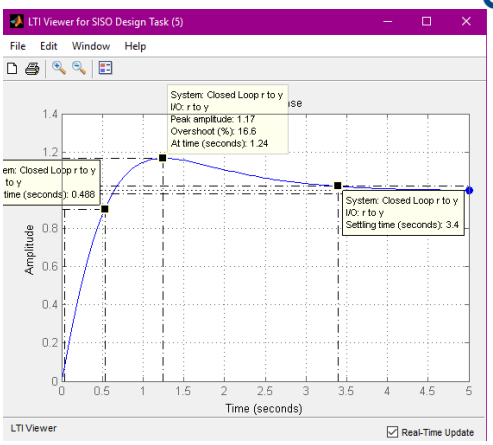
we select poles for -2 and -2.5 considering that need a faster response, for that our poles has to be located to the left of vertical line.

PID = (s + 2)*(s + 2.5) / s;

>> sisotool(C*G)



Selecting with mouse the poles of right we moved them to limit, over vertical line to achieve a faster response.



As we can see in transient response, our controller met all requirements.

The controller is:

$$PID = 2.4337* (s + 2)*(s + 2.5) / s$$

And close loop transfer function is

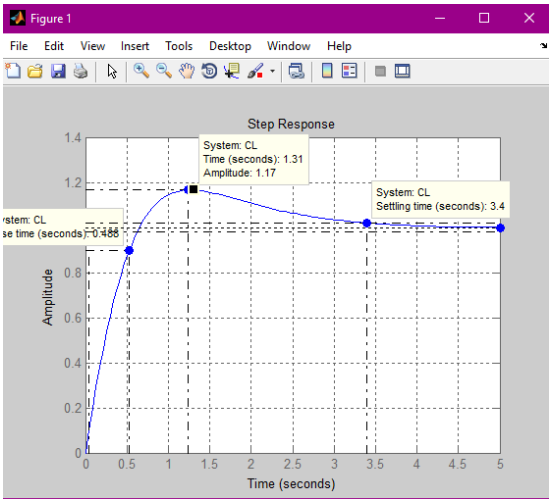
CL=feedback(PID*G,1)

CL =

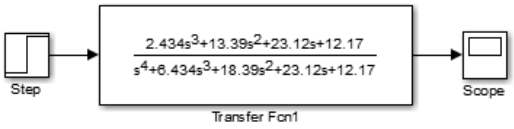
$$\frac{2.434\ s^3 + 13.39\ s^2 + 23.12\ s + 12.17}{s^4 + 6.434\ s^3 + 18.39\ s^2 + 23.12\ s + 12.17}$$

Continuous-time transfer function.

Step(CL)



Now to implement controllers 1 and 2 in Simulink we use transfer function block and set the arrangement as follows:



This is the close loop transfer function.. the simplified version of typical control block. The response obtained is: