Real-Time Control Design Examples by using Scicos and FLEX

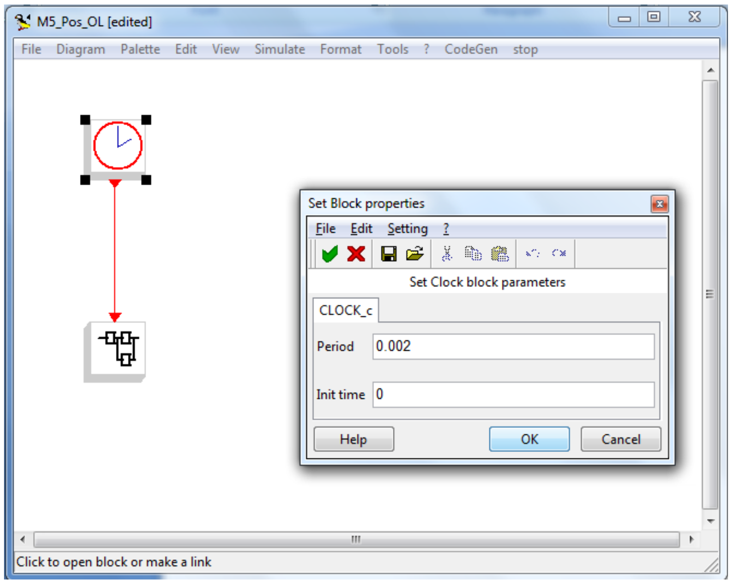
Position and Speed Control

Tan Chin Luh

Contents

Experiment 1: Open Loop Motor Position

Objective: Understanding how sensors and actuators work.

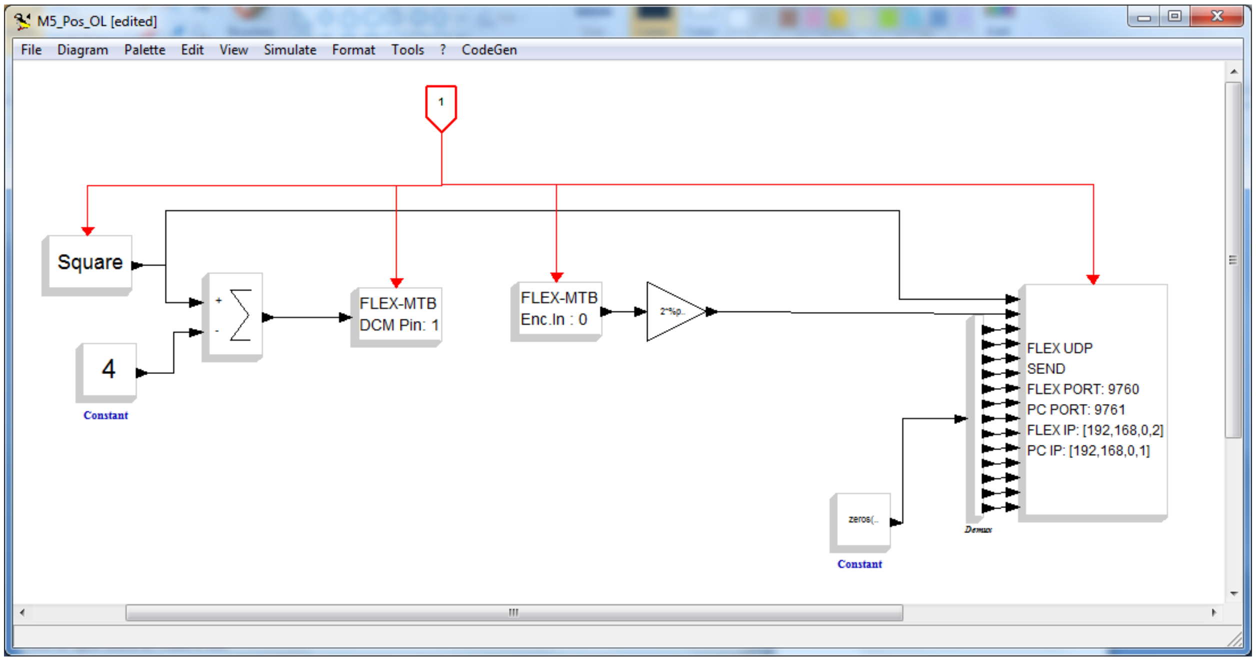

M5_Pos_OL [edited]
File Diagram Palette Edit View Simulate Format Tools ? CodeGen stop
Set Block properties
File Edit Setting ?
Help
OK
Cancel
Click to open block or make a link

Set Clock block parameters

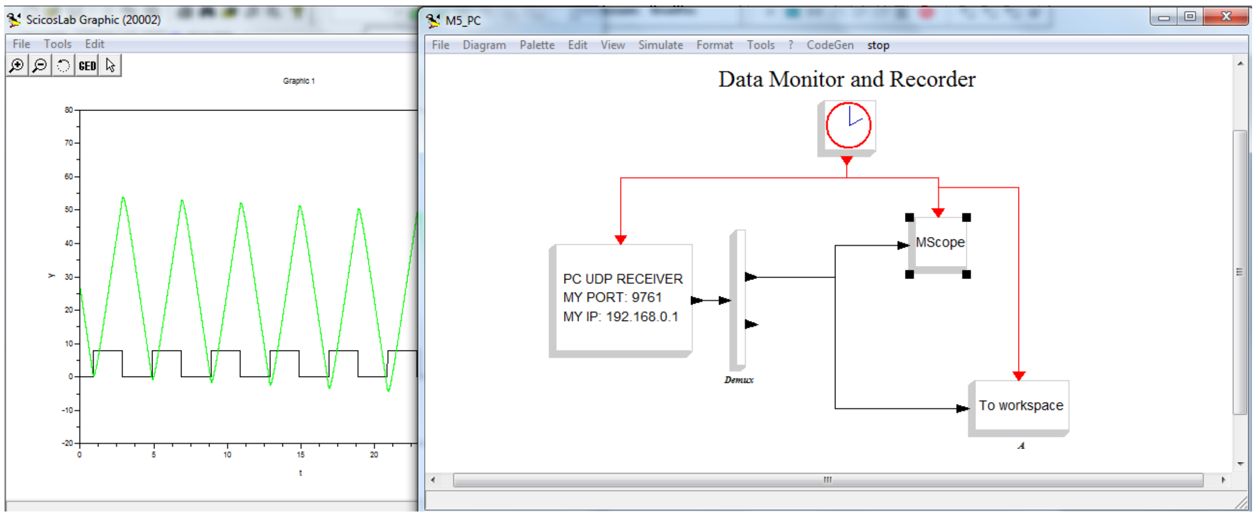
CLOCK_c

Period 0.002

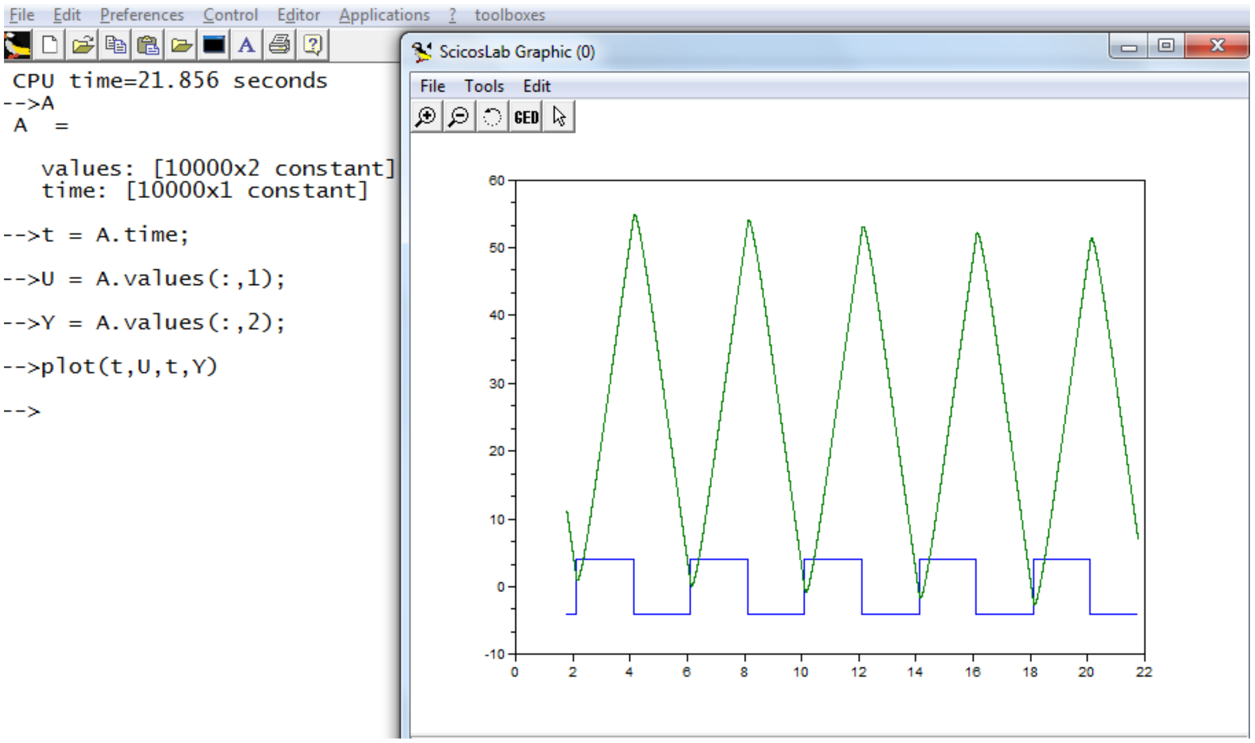
Init time 0



1. The model M5_Pos_OL.cos is compiled and run in the FLEX Demo2 Pack.
2. A square ware with amplitude -4 to 4 and period of 4 second is sent to drive the motor.
3. The position of the motor is sensed by the encoder and feed it back to the PIC.
4. The data for both input and output are sent back to computer through the TCPIP.


ScicosLab Graphic (20002)
File Tools Edit
Graphic 1
M5_PC
File Diagram Palette Edit View Simulate Format Tools ? CodeGen stop
Data Monitor and Recorder
PC UDP RECEIVER
MY PORT: 9761
MY IP: 192.168.0.1
Demux
MScope
To workspace
A

5. Model M5_PC is used to get the data back from the PIC, to scope as well as to Scicoslab workspace for system Identification.
6. The data is then extracted to variable t, U, and Y for system identification.

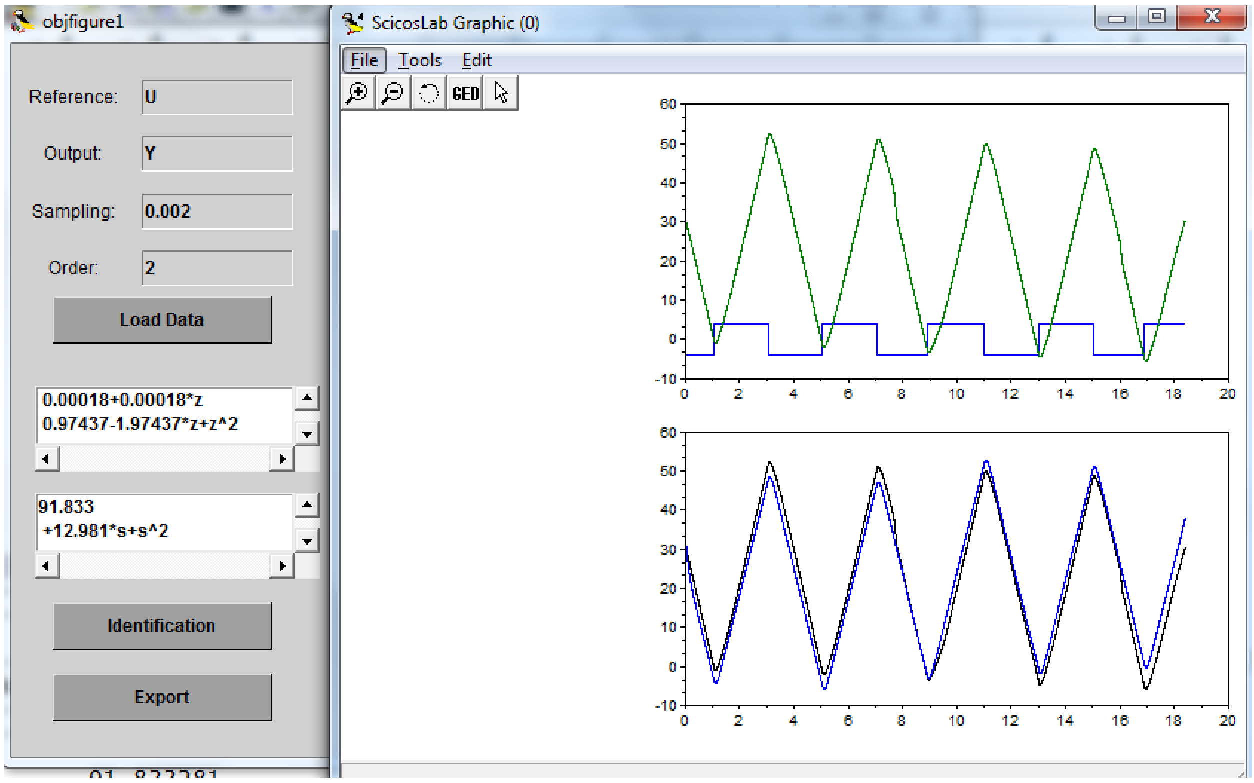

File Edit Preferences Control Editor Applications ? toolboxes
ScicosLab Graphic (0)
File Tools Edit

```
 CPU time=21.856 seconds
-->A
 A  =

    values:  [10000x2 constant]
    time:  [10000x1 constant]

-->t = A.time;

-->U = A.values(:,1);

-->Y = A.values(:,2);

-->plot(t,U,t,Y)

-->
```

Experiment 2: System Identification (Data Driven Modeling) for DC Motor Position

Objective: Quick introduction to “black box” modeling in comparing with mathematic modeling.

1. With the Open Loop position data, launch the “sciIdent” for simple system identification GUI.
2. Make sure the fields Reference, Output, and Sampling match the variables you created.
3. Click Load Data button to verify that the data is imported correctly.
4. Click Identification to get the Model of the system via Identification method.
5. Click Export button to get the output echo on the scicoslab windows.



Reference: U

Output: Y

Sampling: 0.002

Order: 2

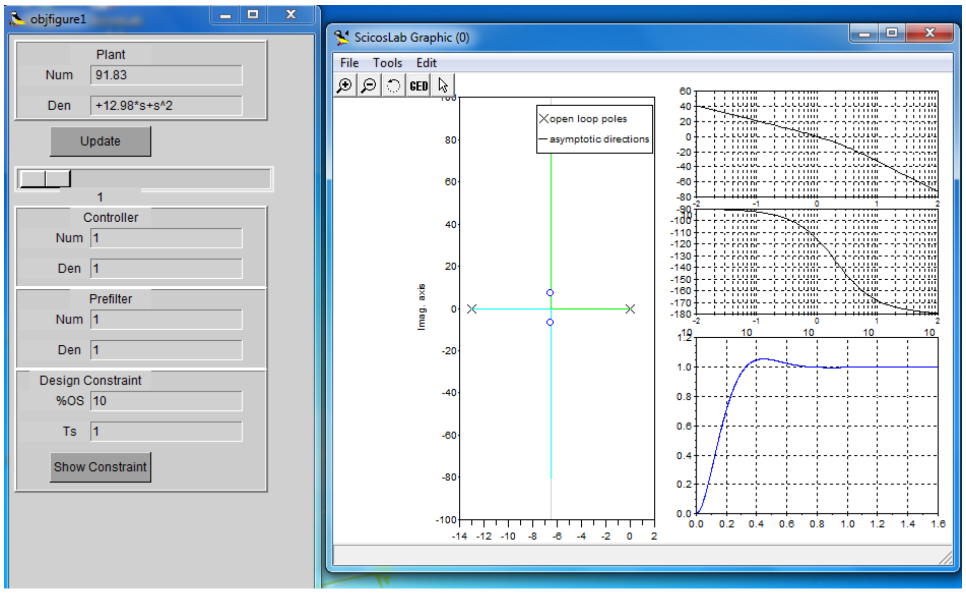
```
0.00018+0.00018*z
0.97437-1.97437*z+z^2
```

```
91.833
+12.981*s+s^2
```

Experiment 3: Controller Design for Position Control

Objective: Controller Design Using RLDesign

1. By using the transfer function obtained from previous exercise, load the rldesign gui and call it using command rldesign(Gs), in which the Gs is the rational transfer function.
2. The following GUI will appear.



Plant
Num 91.83
Den +12.98*s+s^2

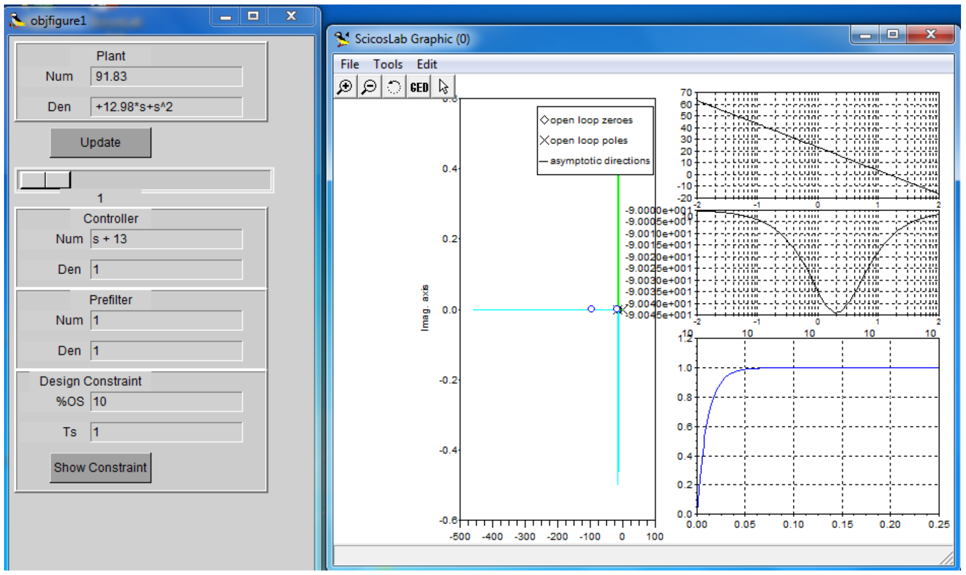
1

Controller
Num 1
Den 1

Prefilter
Num 1
Den 1

Design Constraint
%OS 10
Ts 1

3. Try to add a zero at -13 location.



Plant
Num 91.83
Den +12.98*s+s^2

1

Controller
Num s + 13
Den 1

Prefilter
Num 1
Den 1

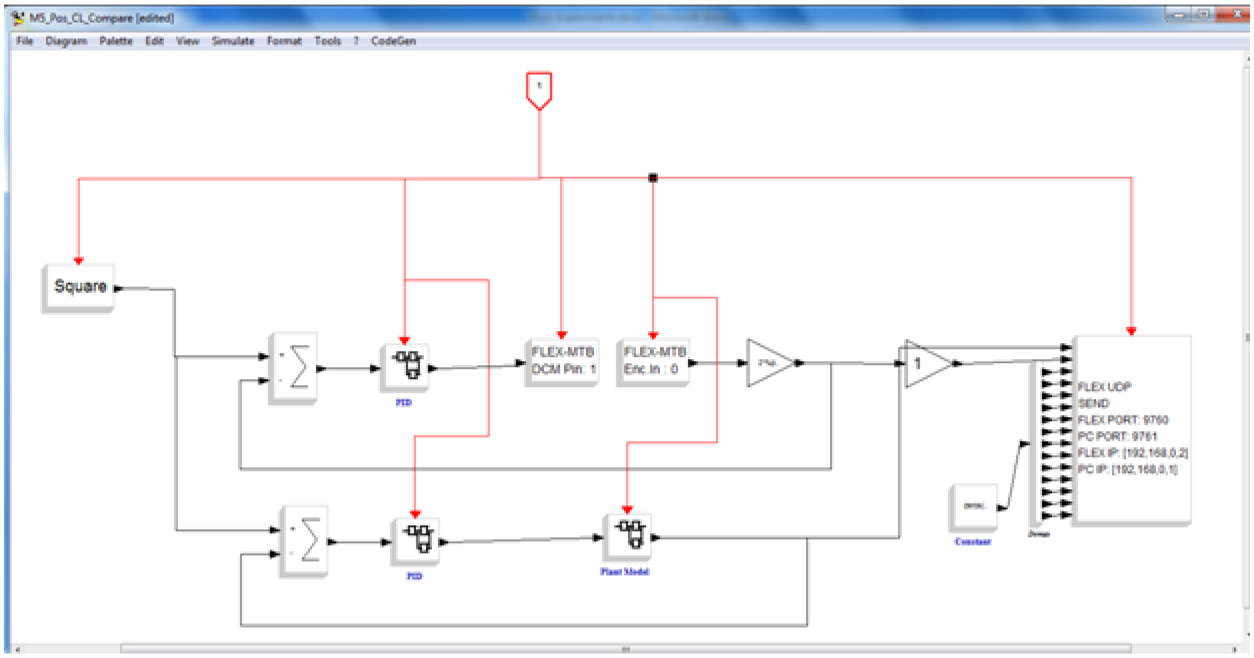
Design Constraint
%OS 10
Ts 1

4. The respond of the simulation seems to be faster and without overshoot.
5. We are going to try to implement the controller in the hardware and verify it.

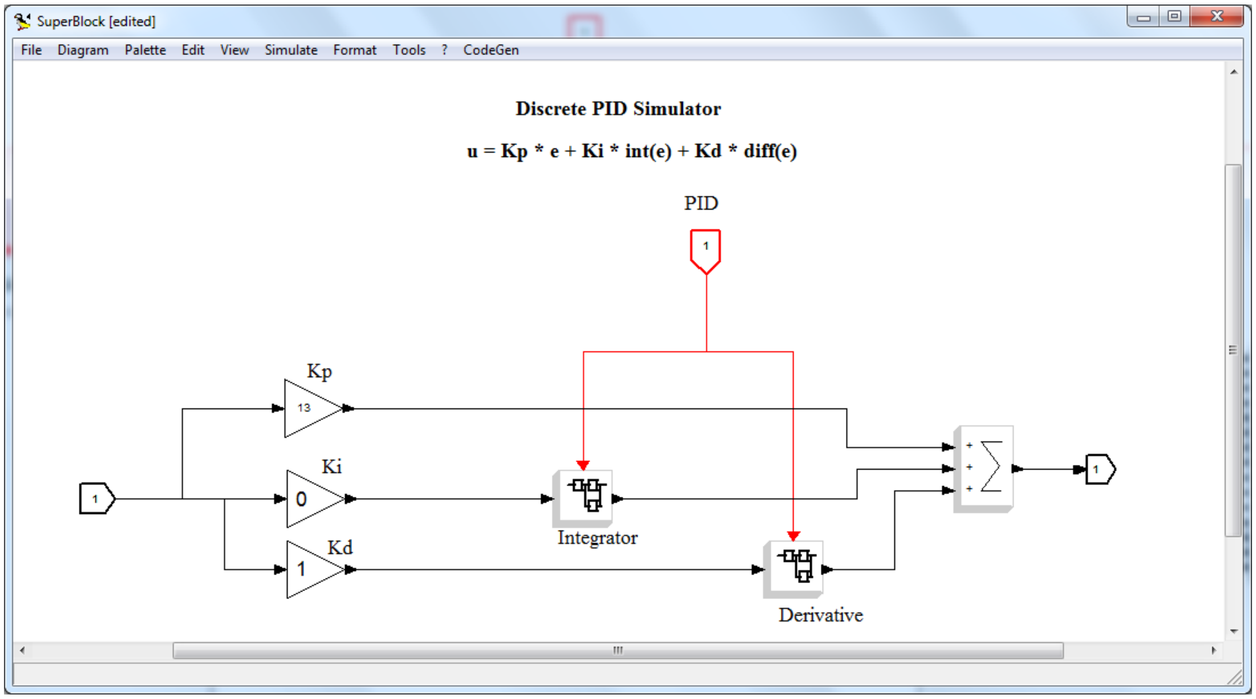
Experiment 4: Controller Implementation and Verification

Objective: To implement the controller in the hardware to verify with the simulation result

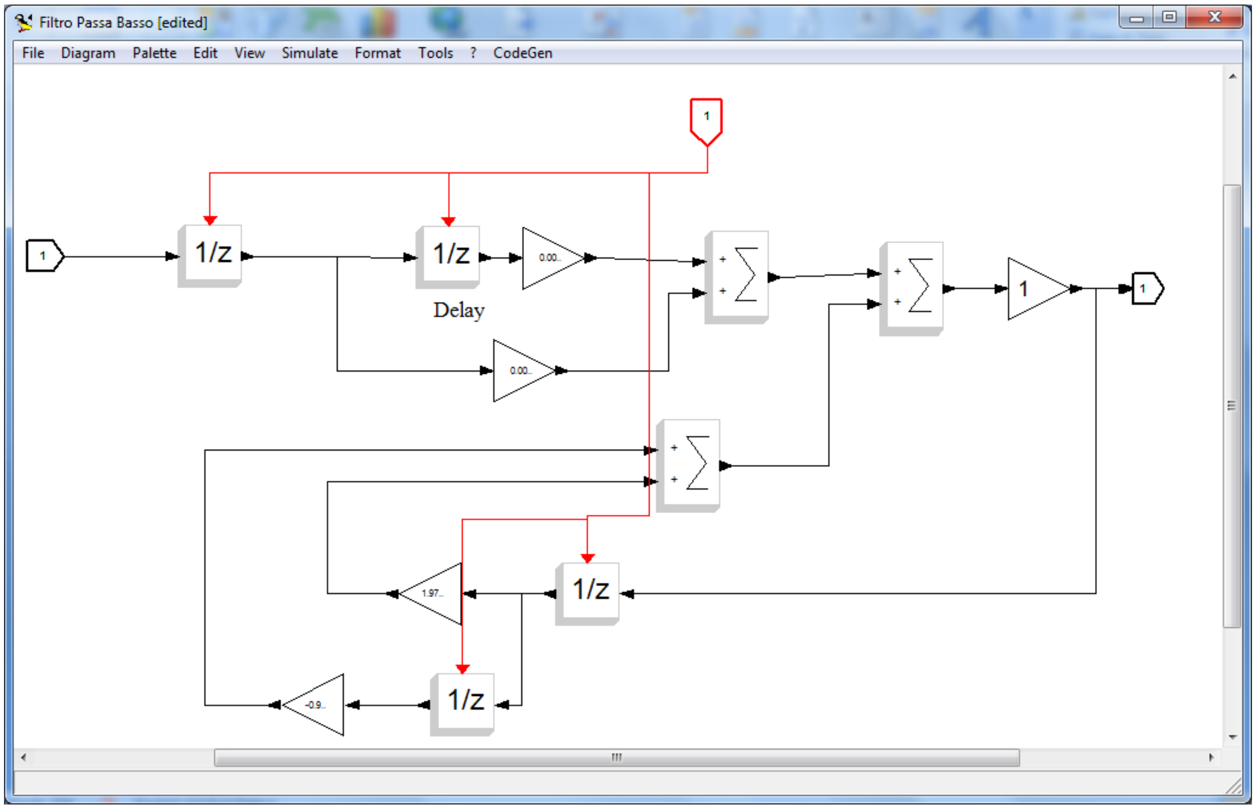
1. From the controller transfer function, we could derive that the Kp = 13 and Kd = 1.
2. Implement in the Scicos.


M5_Pos_CL_Compare [edited]
File Diagram Palette Edit View Simulate Format Tools ? CodeGen
Square
PID
FLEX-MTB DCM Pin: 1
FLEX-MTB Enc In : 0
1
FLEX UDP SEND FLEX PORT: 9760 PC PORT: 9761 FLEX IP: [192,168,0,2] PC IP: [192,168,0,1]
Constant
PID
Plant Model

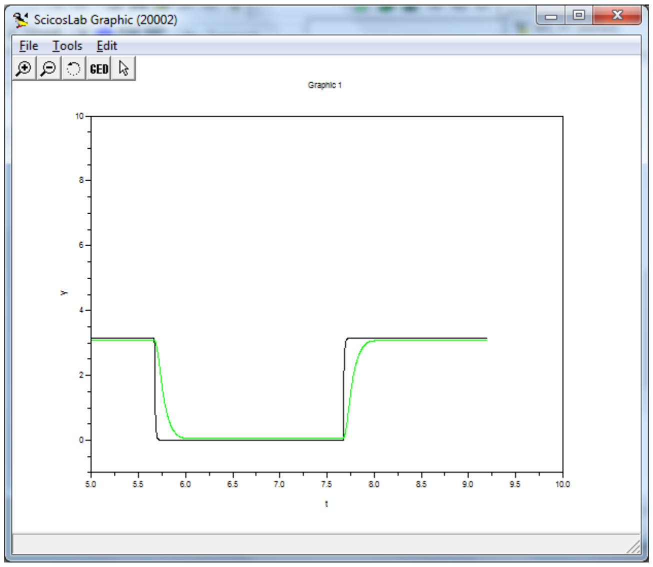
3. The PID for both real-time path and simulation path are identical.


SuperBlock [edited]
File Diagram Palette Edit View Simulate Format Tools ? CodeGen
Discrete PID Simulator
u = Kp * e + Ki * int(e) + Kd * diff(e)
PID
1
Kp
13
Ki
0
Kd
1
Integrator
Derivative
1
1

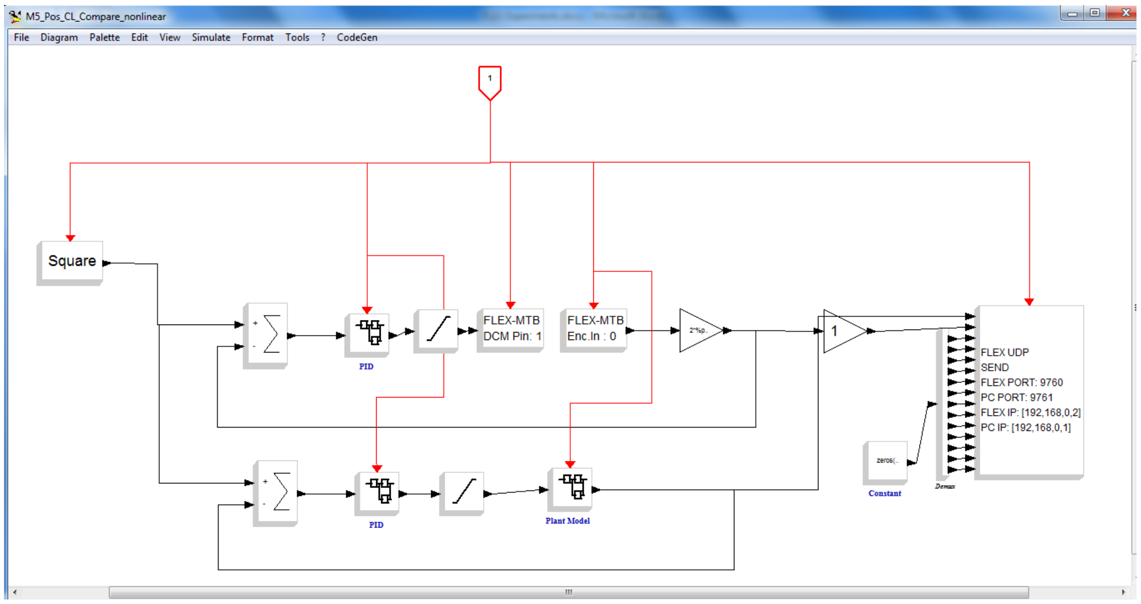
4. The model is constructed with the delay blocks, as shown below. It is equivalent to the z-domain transfer function from the previous experiment, which is:
(0.00018 + 0.00018*z)/ (0.97437 - 1.97437*z + z^2), which could be also obtained by using dscr function to the s domain transfer function.


Filtro Passa Basso [edited]
File Diagram Palette Edit View Simulate Format Tools ? CodeGen
1
1
1/z
1/z
0.00..
Delay
0.00..
1
1
1.97..
1/z
-0.9..
1/z

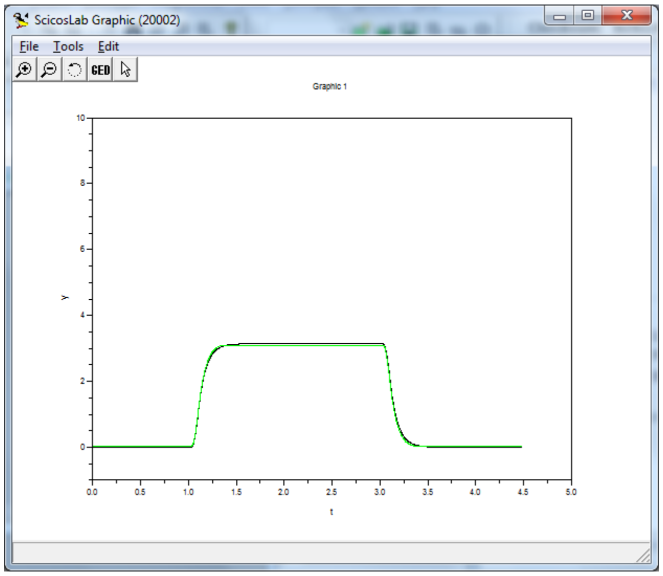
5. Download the model into the FLEX and run the program. The motor now should move from 0 to 180 degree alternatively.
6. Using the M5_PC.cos, compare the result of real system and simulation.



7. The black color line is the simulation result while the green is the real response. The transient response is quite different. This is due to the hardware limitation on the DC motor module, which is program to drive at maximum +- 6V.
8. To modify our model to a non linear model which includes the saturation of the controller output, we add the saturation block after the controller.



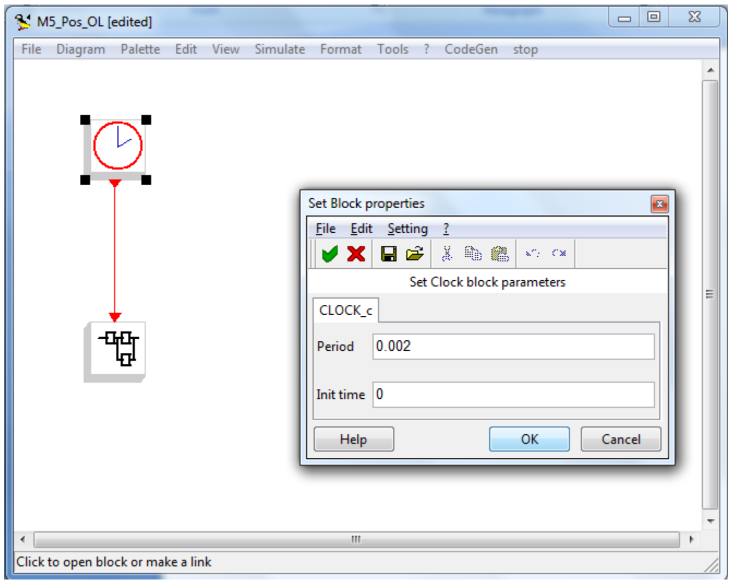
9. Repeat step 5-6 to compare the result again.



10. Now you could see the result match better, which concludes:
    a. Scicos is good to model non-linearity in real system
    b. The controller that we design works!

Experiment 5: Open Loop Motor Speed

Objective: Understanding how to derive the motor speed from encoder.

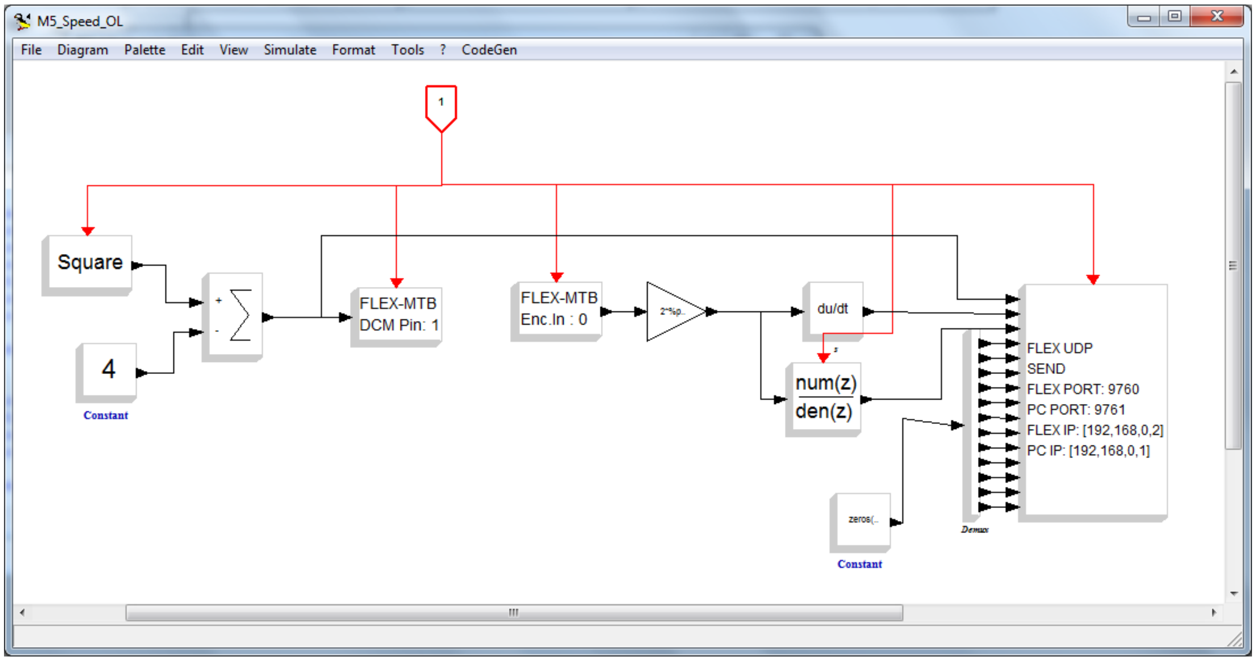

M5_Pos_OL [edited]
File Diagram Palette Edit View Simulate Format Tools ? CodeGen stop
Set Block properties
File Edit Setting ?
Help
OK
Cancel
Click to open block or make a link

Set Clock block parameters

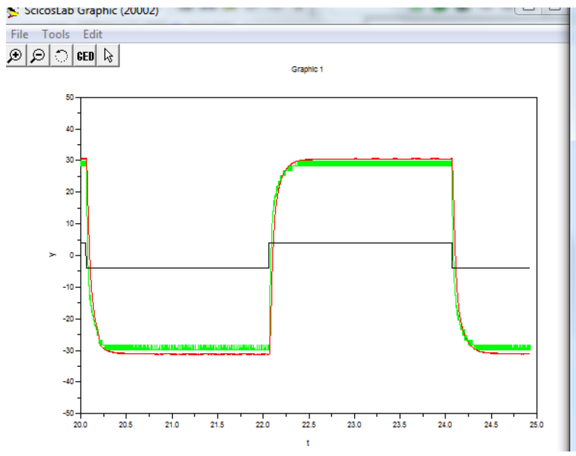
CLOCK_c

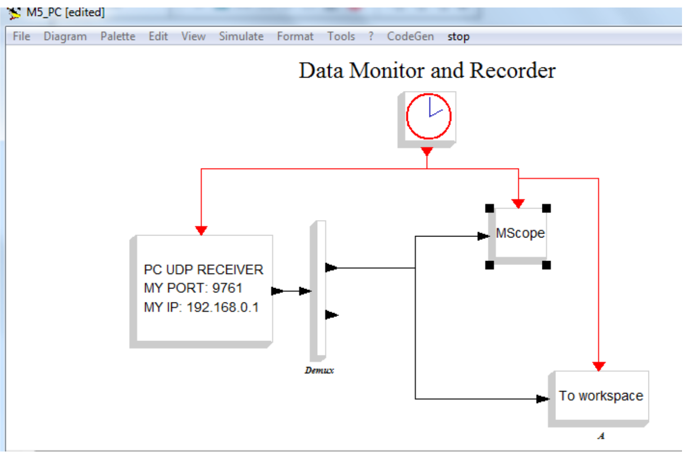
Period 0.002

Init time 0



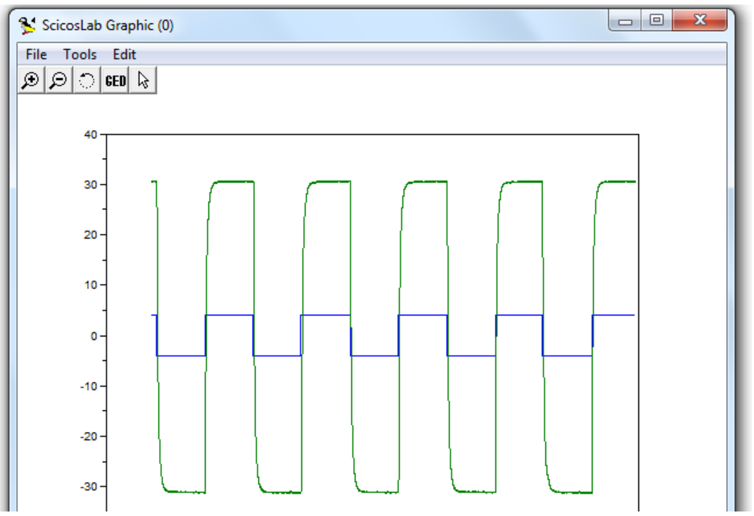
1. The model M5_Speed_OL.cos is compiled and run in the FLEX Demo2 Pack.
2. A square ware with amplitude -4 to 4 and period of 4 second is sent to drive the motor.
3. Do note that the speed is derived using 2 methods, derivative block and also a discrete transfer function block which use to implement the derivative with filter.
4. We try to compare the derived signals using both methods.
5. The data for both input and output are sent back to computer through the TCPIP.




M5_PC [edited]
File Diagram Palette Edit View Simulate Format Tools ? CodeGen stop
Data Monitor and Recorder
PC UDP RECEIVER
MY PORT: 9761
MY IP: 192.168.0.1
Demux
MScope
To workspace
A

6. Do note that the speed derived with derivative block is very noisy, due to the block is compare 2 samples of signals, and divide by the small sampling time which is 0.002 in this case. So all the noises would be amplified as well.
7. The second block which implement derivative with filter would pass the derivative signal into a low pass filter, and yield a smoother signal.
8. Model M5_PC is used to get the data back from the PIC, to scope as well as to Scicoslab workspace for system Identification.
9. The data is then extracted to variable t, U, and Y for system identification.

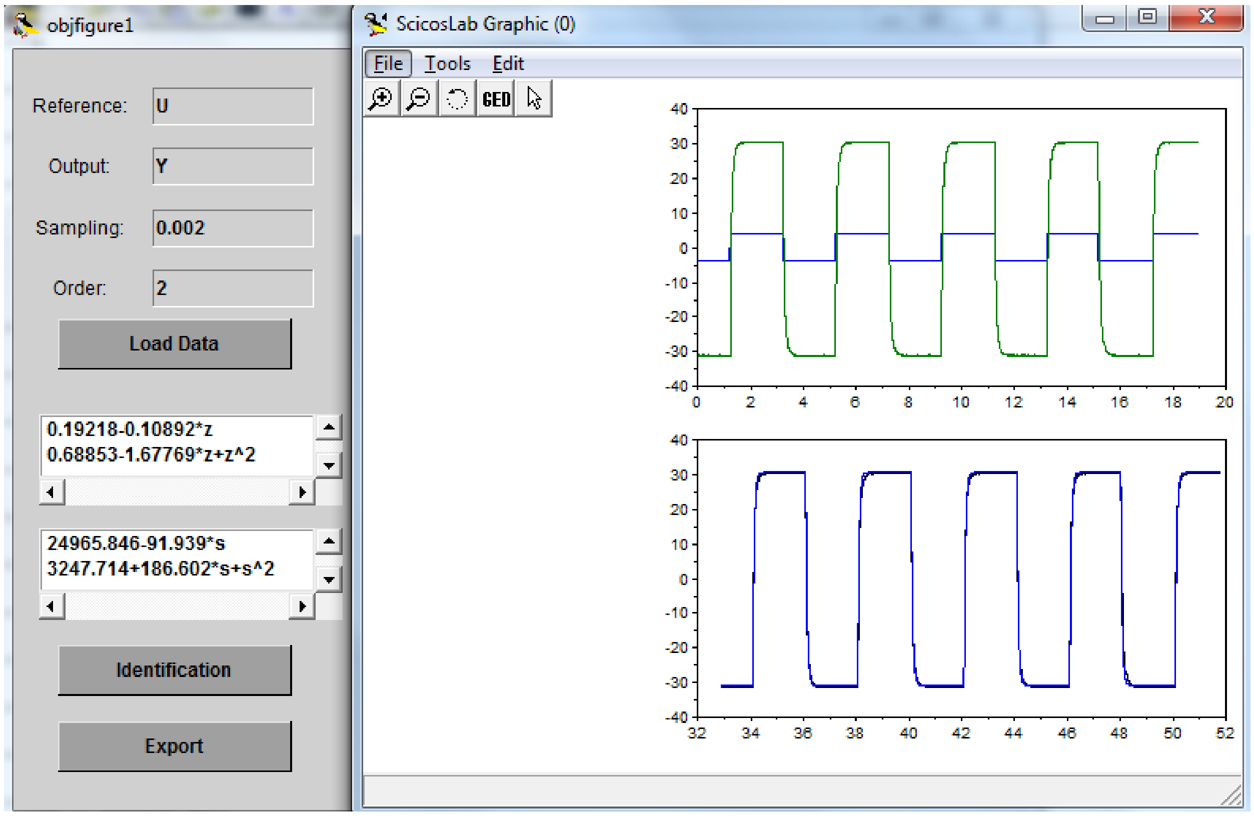
```
-->A
 A  =

   values: [10000x3 constant]
   time: [10000x1 constant]

-->

-->plot(A)
WARNING:Error inside input argument : no data

-->plot(A.values)

-->t = A.time;

-->U = A.values(:,1);

-->Y = A.values(:,3);

-->plot(t,U,t,Y);

-->
```



Experiment 6: System Identification (Data Driven Modeling) for DC Motor Speed

Objective: Quick introduction to "black box" modeling in comparing with mathematic modeling.

1. With the Open Loop speed data, launch the "sciIdent" for simple system identification GUI.
2. Make sure the fields Reference, Output, and Sampling match the variables you created.
3. Click Load Data button to verify that the data is imported correctly.
4. Click Identification to get the Model of the system via Identification method.
5. If the result is not satisfactory, try following steps:
   a. Crop the initial data if the data captured are started with a spike. (changes from low value to high value or vice versus after few samples)
   b. Try to smooth the data further.
   c. If none of those works, try to capture the data again.
6. Click Export button to get the output echo on the scicoslab windows.



Reference: U

Output: Y

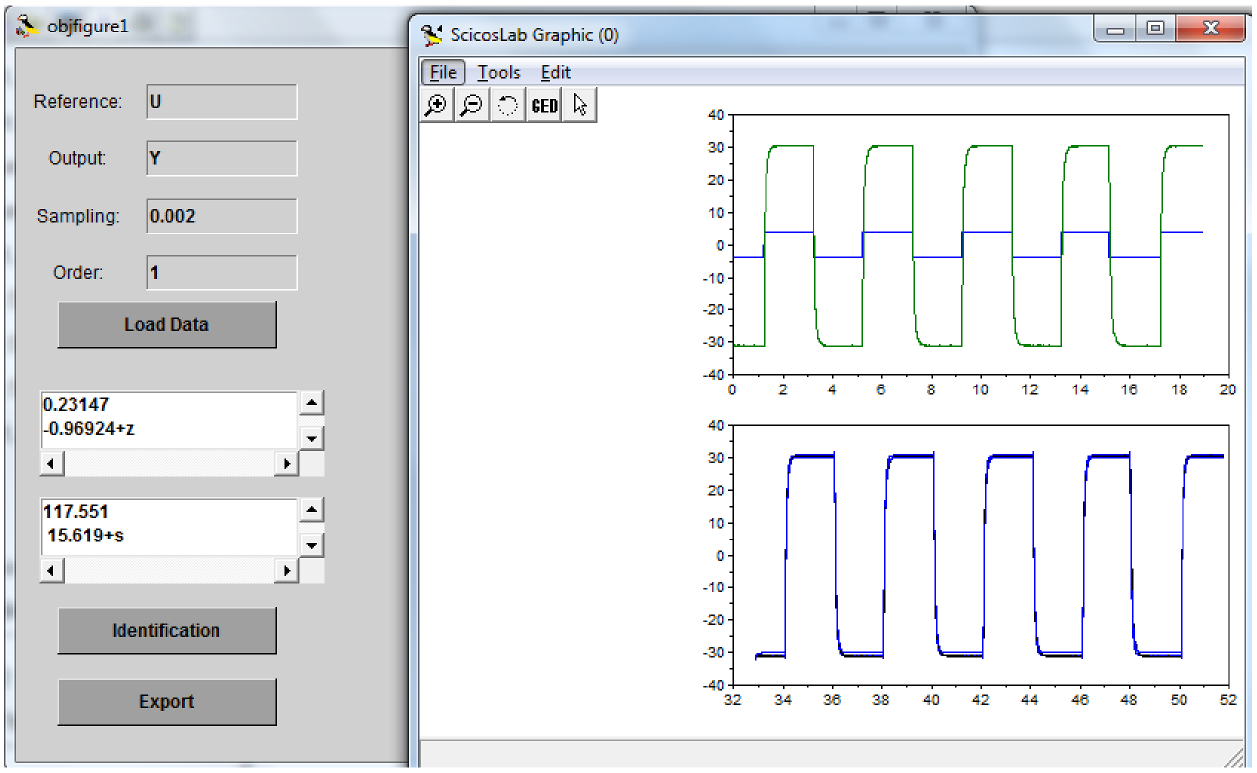
Sampling: 0.002

Order: 2

```
0.19218-0.10892*z
0.68853-1.67769*z+z^2
```

```
24965.846-91.939*s
3247.714+186.602*s+s^2
```

7. Try to set the order to 1, and perform the identification again.



Reference: U

Output: Y

Sampling: 0.002

Order: 1

```
0.23147
-0.96924+z
```

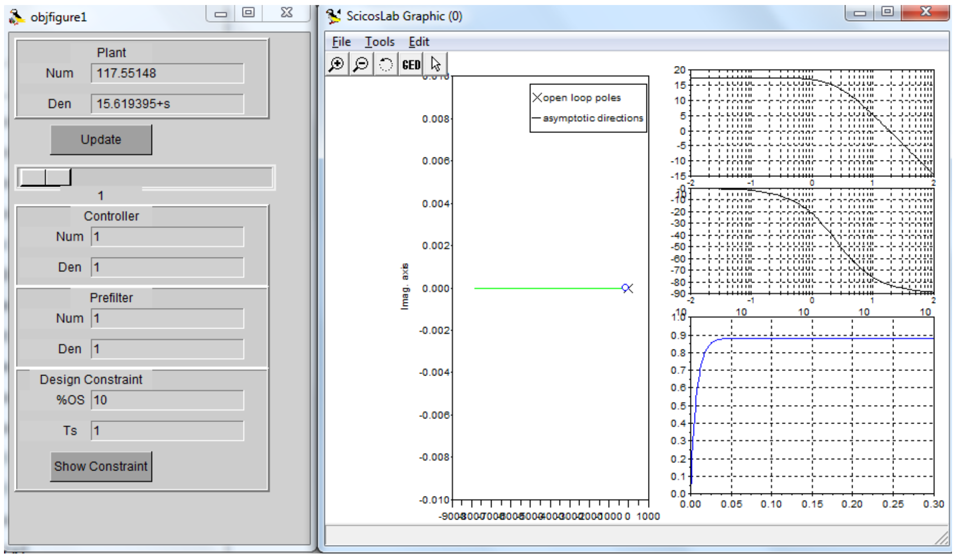
```
117.551
 15.619+s
```

8. You'll notice that the first order is still descript the system well.
9. Compare the result for the position and the speed, you'll notice that the gain and the pole of the system are similar, or in other words, the position is the integration of the speed. (with extra pole at 0)

Experiment 7: Controller Design for Speed Control

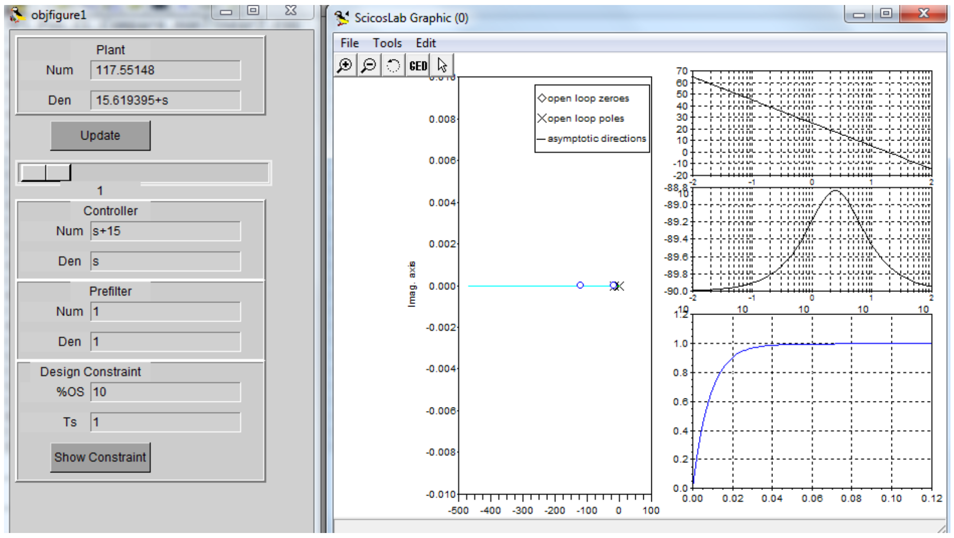
Objective: Controller Design Using RLDesign

1. By using the transfer function obtained from previous exercise, load the rldesign gui and call it using command rldesign(Gs), in which the Gs is the rational transfer function.
2. The following GUI will appear.



Plant
Num 117.55148
Den 15.619395+s
Update
1
Controller
Num 1
Den 1
Prefilter
Num 1
Den 1
Design Constraint
%OS 10
Ts 1
Show Constraint

3. As this is a type 0 system, the system would have the steady state error. To eliminate this, add a integrator to the system.



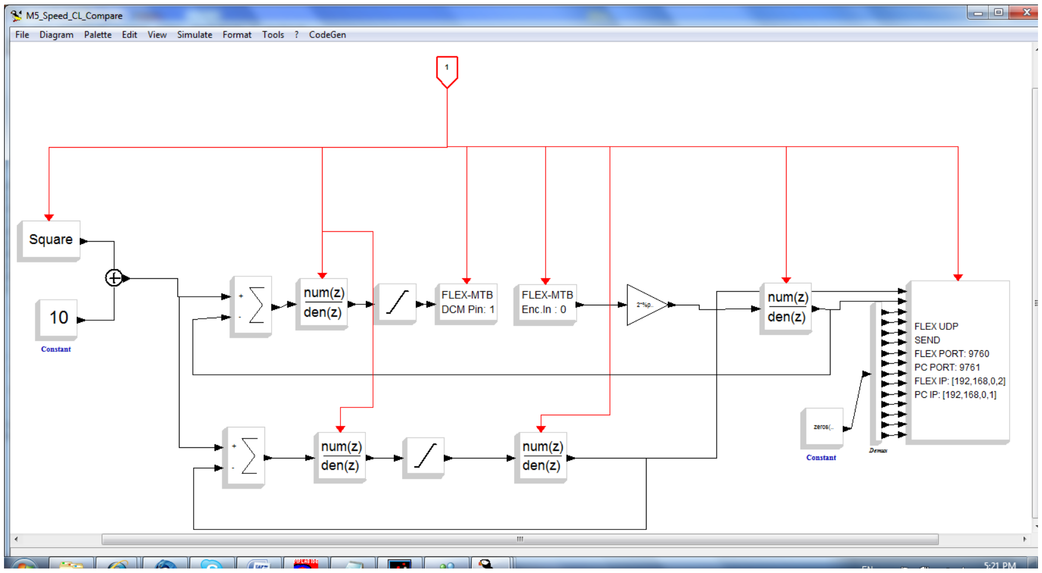
Plant
Num 117.55148
Den 15.619395+s
Update
1
Controller
Num s+15
Den s
Prefilter
Num 1
Den 1
Design Constraint
%OS 10
Ts 1
Show Constraint

4. Add a zero at location -15.
5. We now are having a system with no steady state error and fast response time.
6. We are going to try to implement the controller in the hardware and verify it.

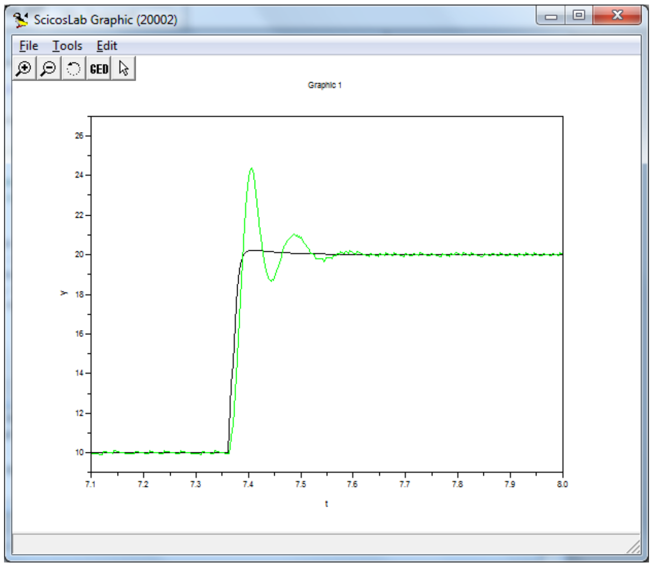
Experiment 8: Controller Implementation and Verification

Objective: To implement the controller in the hardware to verify with the simulation result

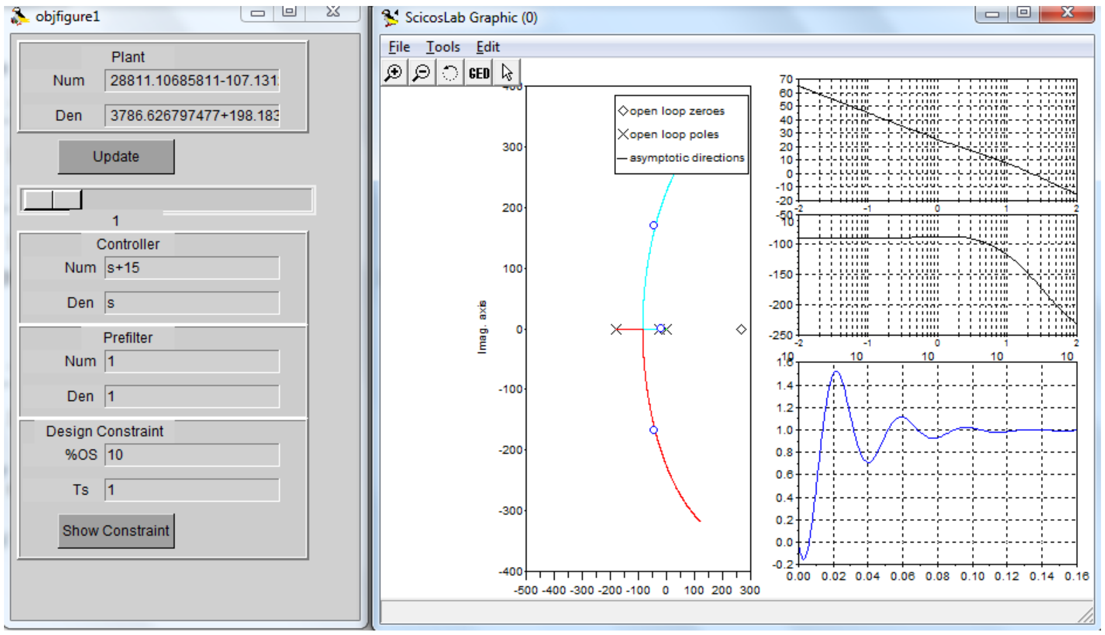
1. In this example, we are going to use the discrete transfer function block to realize the PI controller.The PI controller is (s+15)/s in s-domain.Converting to z-domain, we get (z-0.97)/(z-1) .
2. Implement in the Scicos.



3. The PI for both real-time path and simulation path are identical.
4. The model is constructed with discrete transfer function block, which is: 0.2314687/ (-0.9692441 + z), which could be also obtained by using dscr function to the s domain transfer function.
5. Download the model into the FLEX and run the program. The motor now should move at 10 rad/s and 20 rad/s alternatively.
6. Using the M5_PC.cos, compare the result of real system and simulation.

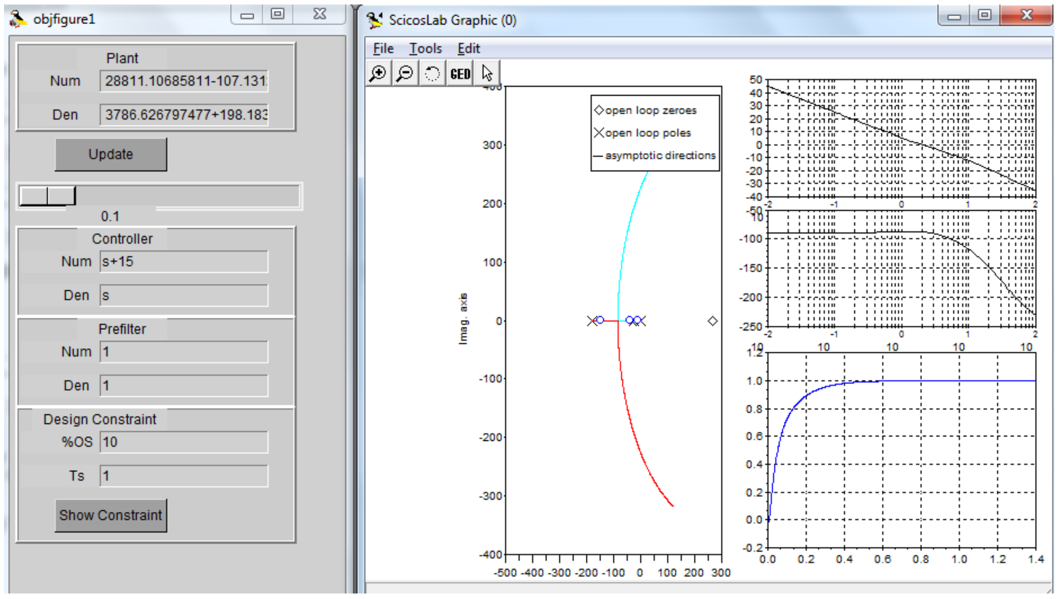


7. The black color line is the simulation result while the green is the real response. The real system seems to have overshoot while the simulation system response is flat at steady state.
8. Now we will use the second order model from the system identification in rldesign and enter the same controller, which is (s+15)/s.
9. Notice that the step response looks like closer to the real response, which is, with overshoot before converge.


objfigure1

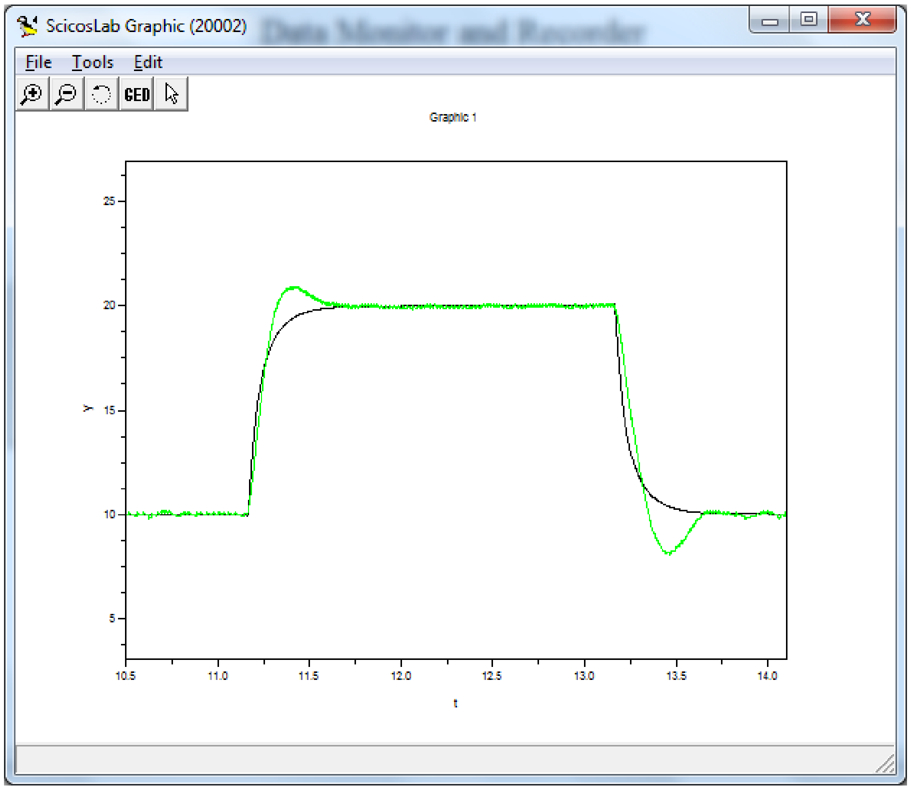
Plant
Num 28811.10685811-107.131
Den 3786.626797477+198.183
Update
1
Controller
Num s+15
Den s
Prefilter
Num 1
Den 1
Design Constraint
%OS 10
Ts 1
Show Constraint

10. Now adjust the close loop gain to 0.1, and the result shown as following figure:

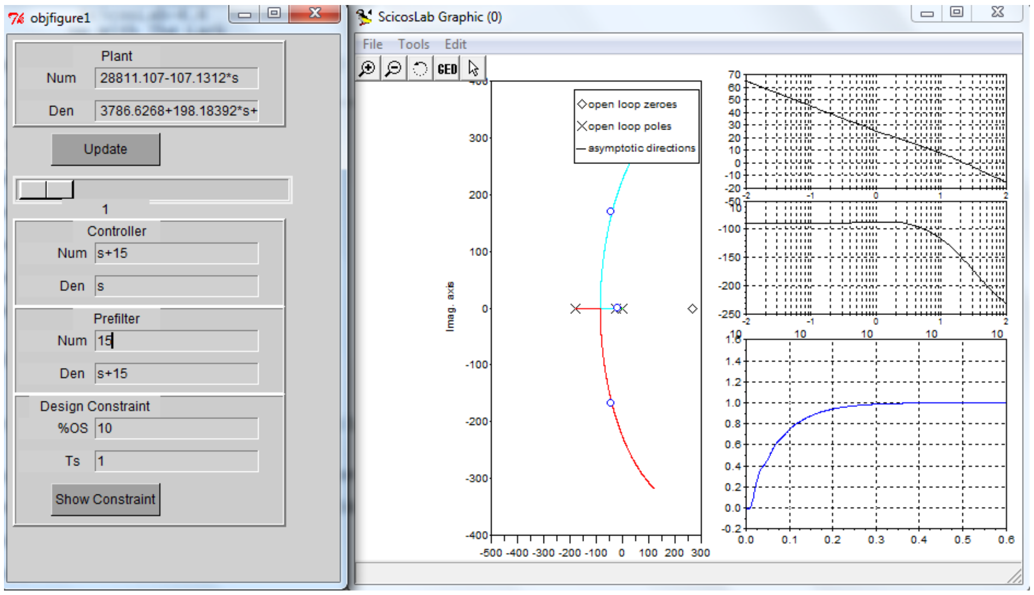

objfigure1

Plant
Num 28811.10685811-107.131
Den 3786.626797477+198.183
Update
0.1
Controller
Num s+15
Den s
Prefilter
Num 1
Den 1
Design Constraint
%OS 10
Ts 1
Show Constraint

11. We are getting closer response; however, the overshoot is still there. We could continue redesign our controller or fine tuning our model, or we could try something else.



12. Set the close loop gain back to 1, and add a pre-filter which has a pole to match the controller zero, and also the dcgain of 1, we could reduce the overshoot.


objfigure1

Plant
Num 28811.107-107.1312*s
Den 3786.6268+198.18392*s+

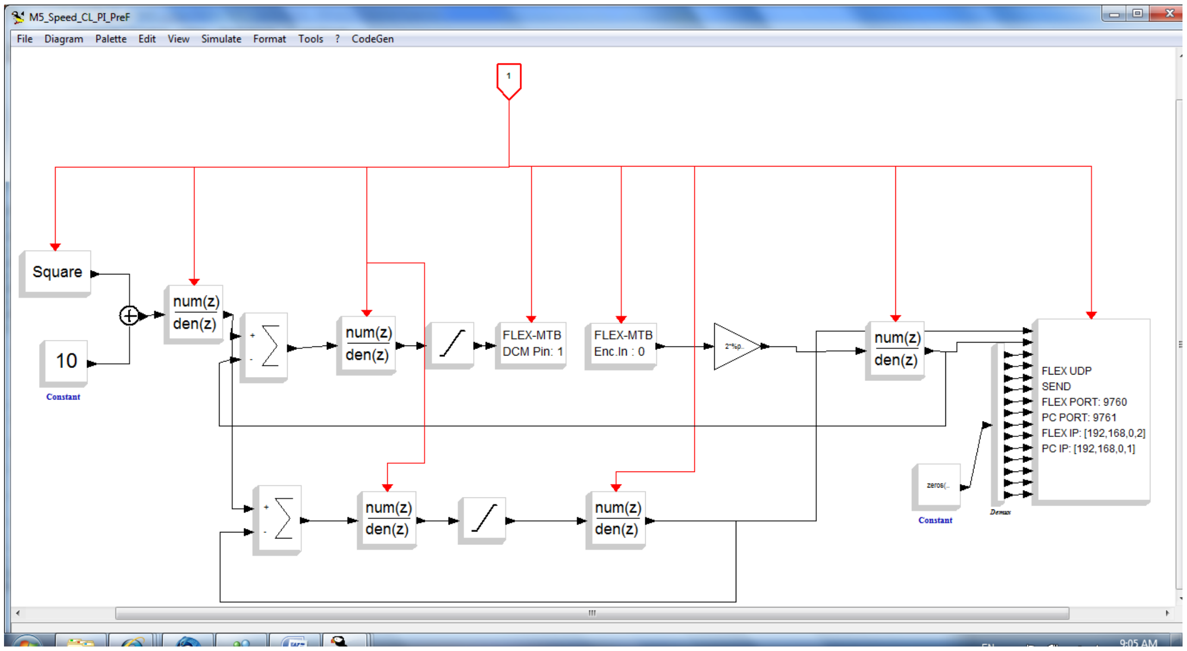
1

Controller
Num s+15
Den s

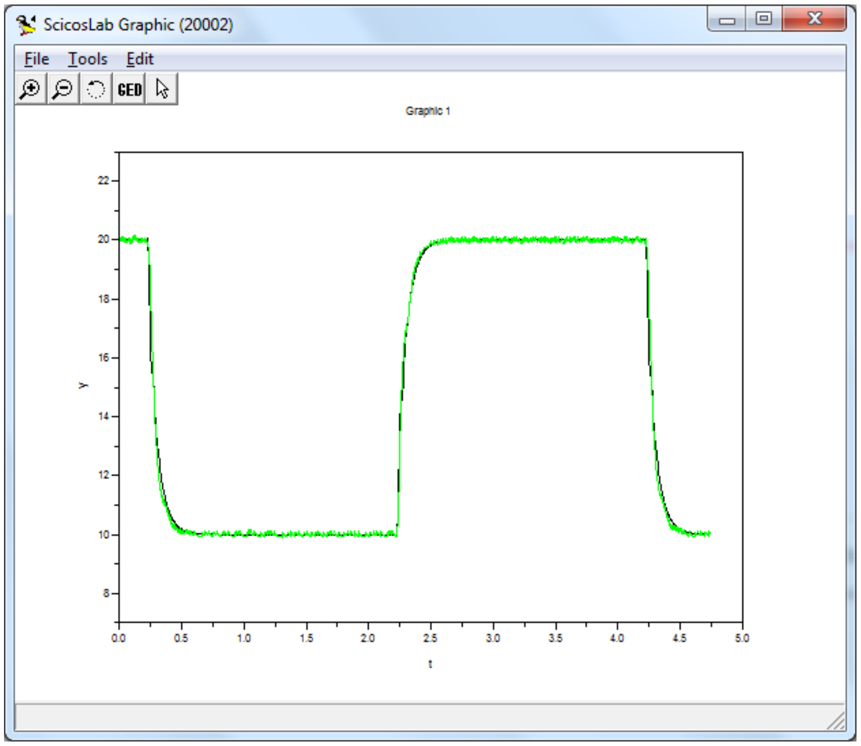
Prefilter
Num 15
Den s+15

Design Constraint
%OS 10
Ts 1

13. Run the new controller in the FLEX, as shown in figure:



14. Now you could see the response is much better, without overshoot and steady state error.


ScicosLab Graphic (20002)
File Tools Edit
GED
Graphic 1
y
t