Hybrid simulation of dynamic systems using Python

Prof. Roberto Bucher - SUPSI-DTI
January 17, 2014

1 Install the files

The following files must be installed in order to run this example:

**Python**

- Numpy from http://sourceforge.net/projects/numpy/files/
- Scipy from http://sourceforge.net/projects/scipy/files/
- Matplotlib from http://sourceforge.net/projects/matplotlib/
- Slycot from https://github.com/avventi/Slycot
- control-0.6c from http://sourceforge.net/projects/python-control/files/

2 The plant

We start from a system with the following transfer function:

\[ G(s) = \frac{2}{s \cdot (s + 5)} \]

3 Developping the controller

For the given plant we develop under Python the following systems:

- A discrete state feedback controller with integral part
- A discrete full observer

The following python code performs the previous tasks:

```python
from yottalab import *
from control import *
from RCPblk import *
import numpy as np
import scipy as sp

# Transfer function
```
g=tf([2],[1,5,0])

# state space form
a=[[0,1],[0,-5]]
b=[[0],[2]]
c=[[1,0]];
d=[0];

# Continuous state space form
sysc=ss(a,b,c,d)

# Discrete representation
Ts=0.01  # Sampling time
sysd=bb_c2d(sysc,Ts,'zoh')  # bb_c2d from yachtalab.py
      # instead of c2d from control

# Control system design
# State feedback with integral part

wn=10
xi=sqrt(2)/2

cl_p1=[1.2*xi*wn,wn**2]
c1_p2=[1,wn]
c1_poly=sp.polymul(cl_p1,cl_p2)
c1_poles=sp.roots(cl_poly);  # Desired continuous poles
c1_polesd=sp.exp(cl_poles*Ts)  # Desired discrete poles

sz1=sp.shape(sysd.A);
sz2=sp.shape(sysd.B);

# Add discrete integrator for steady state zero error
Phif=np.vstack((sysd.A,-sysd.C*Ts))
Phif=np.hstack((Phif,[[0],[0],[1]]))
Gf=np.vstack((sysd.B,zeros((1,1))))

# Perform pole placement
kint=place(Phif,Gf,cl_polesd)

# Full order observer
p_oc=10*(cl_poles[1:3])  # continuous observer poles
p_od=sp.exp(p_oc*Ts);  # discrete observer poles

L = place(sysd.A.T, sysd.C.T,p_od)
L = mat(L).T

Ao = mat(sysd.A) - L*mat(sysd.C)
Bo=mat._hstack((sysd.B-L*sysd.D,L))
n=shape(Ao)
m=shape(Bo)
Co=eye(n[0],n[1])
Do=zeros((n[0],m[1]))
obs=StateSpace(Ao,Bo,Co,Do,sysd.dt)

4 Simulation

In order to simulate the hybrid system (controller and observer are discrete time systems, plant is a continuous time system) we have to create the block diagram of figure 1.
First of all we have to define the nodes (1 to 7) of the block diagram as in figure 2

Now we can create the description for the simulation:

```plaintext
# Block Diagram for simulation
intg = tf(Ts, [1, -1], Ts)
Kfbk = [-kint[0,2], -kint[0,0], -kint[0,1]]

# Defining the blocks
Ref = squareBlk (1, 1, 6, 3, 0, 0)
sum1 = sumBlk ([1,5], 2, [1,-1])
Intg = dssBlk (2, 3, intg, 0)
sum2 = sumBlk ([3,6,7], 4, Kfbk)
plant = cssBlk (4, 5, g, 0)
obsv = dssBlk ([4,5], [6,7], obs, 0)
Out = printBlk ([1,5])
fname = 'test'
genCode(fname, 7, Ts, [Ref, sum1, Intg, sum2, plant, obsv, Out])
genMake(fname)
```

The command “genCode” find first the right execution sequence by searching the blocks that have direct connection between input and output. In a second step the routine creates the source code for the simulation or the RT executable.

The utility “pytest” with the following code

```bash
./$1 > x.x
```
gnuplot -p -e "set grid; plot for [col=2:$2] 'x.x' using 1:col with lines"
rm -f x.x

allows to connect the output of the simulation to gnuplot, to see the results of the simulation.
First of all we have to compile the generated code on the shell using

make

and then we can launch the simulation with

pythest 'test -f 30' 3

where

- “test” is the generated executable
- “-f 30” means 30 sec of simulation
- “3” is the number of the signals passed to gnuplot (first column contains the simulation time)

Figure 3 show the output of the simulation

Figure 3: Simulation output

5 A minimal Block diagram editor

At present we don’t have a GUI like Simulink to implement block diagrams for the simulation.
The previous code has been written by hand, using a quite complex syntax, different for each block.
The help of each block is useful to see which parameters should be passed to create the element of class “RCPblk”.
In order to simplify the generation of this code, I’ve developed a tool called pyEditor, which allows to generate the blocks sequence in a simpler way. From the menu "Blocks" we can choose a Block and a dialog editor is open (5). We have only to set a name for the block and the right parameters (??). By giving "OK" a line is appended to text window (7). We can set all the blocks in the same way (8). After editing all the needed blocks, we can copy all the lines in the clipboard with the "COPY TO CLIPBOARD" and then we can paste them in our python script.
Figure 5: Parameter editor

Figure 6: Parameter editor
Figure 7: Parameter editor

Figure 8: Parameter editor