Rapid Controller Prototyping at the SUPSI Laboratory

Roberto Bucher
University of Applied Sciences of Southern Switzerland
Galleria 2, Lugano-Manno 6928, C H
roberto.bucher@supsi.ch

Abstract

The Scuola Universitaria Professionale della Svizzera Italiana (SUPSI) is a university of applied sciences which offers students the possibility to get a specialization in mechatronic in the third study year. One of the most important parts of this study is represented by the mechatronic laboratory, where students can put all the tasks connected to the different lectures into practice. Rapid Control Prototyping methods (RCP) play a key rule to achieve these objectives.

Since 2003, the SUPSI has based his control laboratory on Linux RTAI. The RTAI-Lab project allows to directly integrate the code generated from the commercial suite Matlab/Simulink/RTW or from the open source suite Scilab/Scicos/RTAICodeGen into Linux RTAI.

Different blocks and functions which facilitate the plant identification and the control design have been created for both environments.

1 Introduction

The Scuola Universitaria Professionale della Svizzera Italiana (SUPSI) is a university of applied sciences which offers students the possibility to get a specialization in mechatronic in the third study year. One of the most important parts of this study is represented by the mechatronic laboratory, where students can put all the tasks connected to the courses of *Modelling and Identification*, *Control Design*, *Mechatronic systems* and *Sensors and actuators* into practice. Rapid Control Prototyping methods (RCP) are used to achieve these objectives.

Section 2 gives an overview about the mechatronic laboratory at SUPSI. Section 3 presents a state feedback control of a simple servo DC motor. The identification of a voice coil motor for nanopositioning is described in section 4. The classical inverted pendulum is described in section 5, where the PC is connected to the plant using a CAN bus. Some conclusions about the experiences collected in the past 4 years are reported in section 6.

2 The SUPSI laboratory

2.1 Equipment

Each labor place is composed of a PC (Pentium 4, 2.8 GHz) with Linux RTAI as real-time operating system, a CAN bus interface (Peaks PCAN or a self built EPP CAN Dongle) and an embedded system (Compact PCI) as supplementary RT target for distributed control. Some PCs are equipped with AD/DA cards (Computer Measurements PCI-1200 DAQ cards).

2.2 Linux RTAI

The RTAI extension was created as an environment for implementing low cost data acquisition and digital controller systems ([1], [2]) and is intended as an open source replacement of other hard real-time OSs such as QNX or VxWorks. This extension adds hard real-time capabilities to Linux, allowing sample frequencies up to several thousands of cycles per second with a jitter of just a few microseconds. Real-time processes can run both in the kernel and in the user area. The low latencies guaranteed by Linux RTAI allows us to implement controllers for systems with a bandwith up to a few hundred hertz.

2.3 RTAI-Lab

RTAI-Lab is an open source project which integrates the code generated by a CACSD (Computer Aided Control System Design) environment into a Linux RTAI hard real-time task. At present, three CACSD suites are supported: the commercial

MATLAB/Simulink/RealTime-Workshop (RTW) suite, the commercial EicasLab suite and the open source SCILAB/Scicos suite. The system has been widely described in [3], [4], [5], [6] and [7]. The document [8] describes in detail how to install the full RTAI-Lab toolchain. RTAI-Lab provides a GUI application for remote monitoring and controlling of hard real-time generated tasks.

3 Control of a DC servo motor

3.1 Plant description

The plant is represented by a servo DC motor from Siemens AG (Fig. 1)

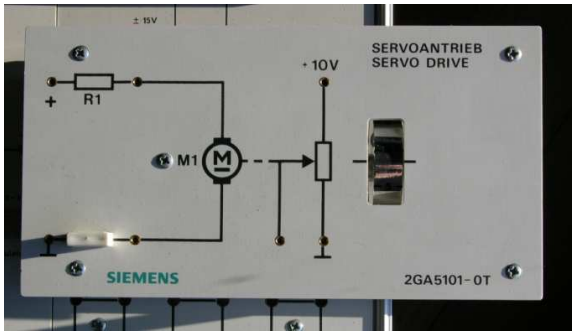

FIGURE 1: *Servo DC motor*

This plant is part of an Universal Experimenter Didactic System, which allows the investigation of PID controllers. In our case, the controller is implemented as a digital controller in the Linux RTAI OS and only the current amplifier and the plant are considered. The interface between the PC and the plant is implemeted by a AD/DA card of Computer Measurements (PCI-1200) using the COMEDI interface([9]) which sends the control signal to the motor and reads the position using a potentiometer.

3.2 Identification

A simplified model of the servo DC motor is represented by

$$G(s) = \frac{K}{s^2 + \alpha \cdot s} \qquad (1)$$

The Scicos block diagram used for the identification is represented in Fig. 2.

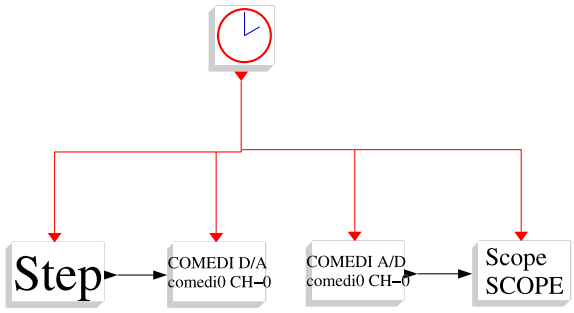

FIGURE 2: *Block diagram for the identification*

The motor can get control signals between $-10V$ and $10V$; a step signal of $8V$ is sent to the real plant and the response is collected using the *xrtailab* application. A simple script (see 3.6) performs the identification using the *leastq* command of the Scilab environment. The measured data and the step response of the identified plant are shown in Fig. 3.

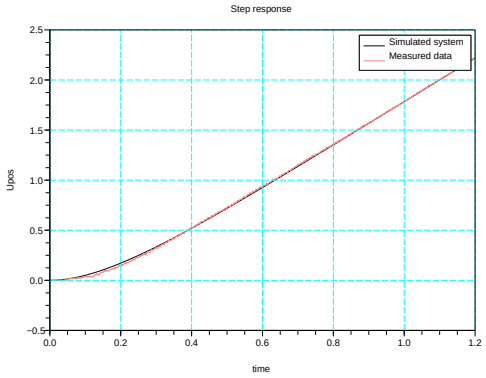

FIGURE 3: *Measured and simulated step response*

3.3 Controller design

The state feedback controller has been implemented using the pole placement method. The 3 poles of the controlled system are placed near to the position of the plant pole. The script which calculates the controller is the same used for the identification (see subsection 3.6). A discrete reduced order observer is used to generate the unknown state of the plant (see A.1).

3.4 Simulation

The same Scicos block diagram has been used for simulation and realtime control; the only difference between the two block diagrams is represented by the superblock *plant* (see Fig. 4 and Fig. 5) and by the *scope* block.

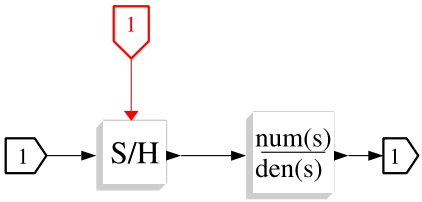

FIGURE 4: *Scicos blocks for simulation*

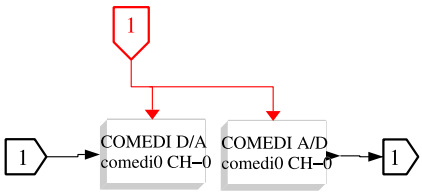

FIGURE 5: *Scicos blocks for real-time code generation*

3.5 Real-time controller

In the case of the real-time control, the mathematical representation of the plant is substituted by two COMEDI blocks, in order to send and collect data from the AD/DA card (see Fig. 5).

The controller has been realized under Scicos (see Fig. 7). It contains a discrete reduced order observer and an integral part with anti-windup to eliminate the steady state error.

Fig. 6 shows the plot of the simulation vs. the plot of the real plant.

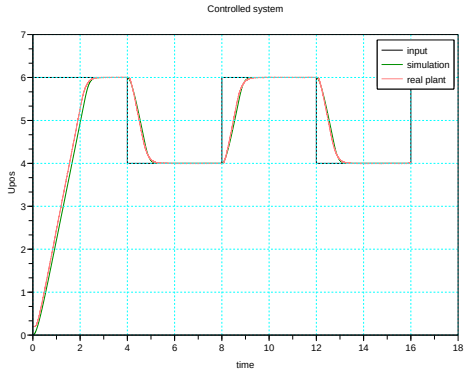

FIGURE 6: *Controlled system - Response of the simulation and of the real plant*

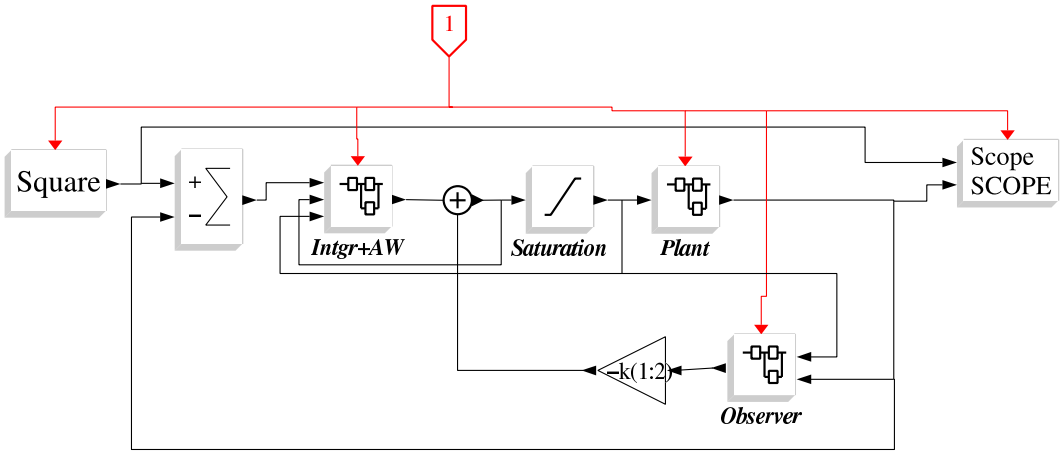

FIGURE 7: *Block diagram for the controller (simulation*

3.6 Scilab script

```
// == Servo DC: Identification and Control design ==
x=read('SCOPE',-1,2);
t=x(:,1);
y=-x(:,2);
y=y-y(1);
t=t(1:4001);
y=y(1:4001);
xbasc()
plot(t,y)

ts=1e-3;

function z=fun(p,t,y)
z=y+p(1)/p(2)^2-p(1)/p(2)*t-p(1)/p(2)^2*exp(-p(2)*t);
endfunction

Uo=8;
p0=[1,4];
p02=[1,4,1];

[ff,p]=leastsq(list(fun,t,y/8),p0);

g=syslin('c',p(1)/(s*(s+p(2))));
xbasc()
bb_step(Uo*g,4)
plot(t,y)

num=g.num;
den=g.den;

g_ss=tf2ss(g);
p_g=roots(den);
```

```
g_dss=dscr(g_ss,ts);
p_max=max(abs(p_g));

p_c=[-p_max,
     -p_max*cos(pi/6)+%i*p_max*sin(pi/6),
      -p_max*cos(pi/6)-%i*p_max*sin(pi/6)];
k_poles=1.2;
p_c=k_poles*p_c;

p_d=exp(p_c*ts);
[Phi,G,C,D]=abcd(g_dss);

[m1,n1]=size(Phi);
[m2,n2]=size(G);
Phi_f=[Phi,zeros(m1,1);-C(1,:)*ts,1];
G_f=[G;zeros(1,n2)];

k=ppol(Phi_f,G_f,p_d);
k_sat=k(3)/30;

p_oc=[-5*p_max];
p_od=exp(p_oc*ts);

T=[0,1];
[Ao,Bo,Co,Do]=redobs(Phi,G,C,D,T,p_od);
```

4 Identification of a voice coil motor

4.1 Plant description

One of the most difficult tasks in control system design is represented by the identification of the plant model. This section describes how Scilab/Scicos/RTAI has been used to find the non-parametric and the parametric identification of a voice coil motor. SUSPI has great experience in controlling high precision systems ([10]). The plant to be identified is a voice coil motor (Fig. 8) used for high precision positioning. It can perform deplacements up to $1mm$ with a precision of about $50nm$.

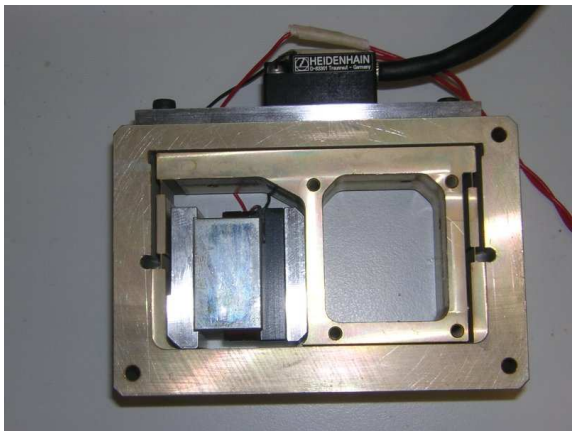

FIGURE 8: *Voice coil motor*

The controller is implemented in a Compact PCI system, where a 10MB Linux-RTAI with full network support is stored into a Flash Card. The two cards to interface the real plant (a DA analog card and a sincos encoder card) have been designed and built at SUPSI. The hard realtime drivers have been also created.

The controller is designed and simulated on a second PC. The generated code is then downloaded and executed into the Compact PCI. The *xrtailab* monitor application can be started on each PC in the LAN to monitor the hard real-time task.

4.2 Identification

The scicos block diagram for the identification is shown in Fig. 9.

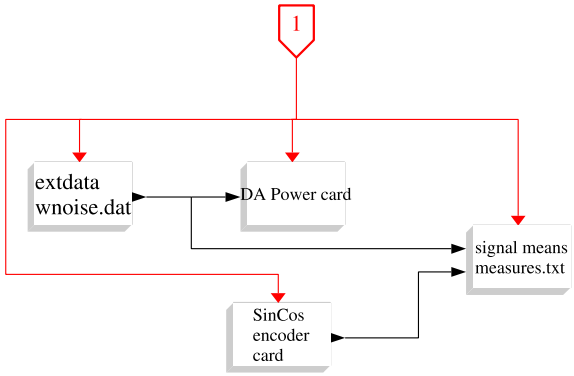

FIGURE 9: *Block diagram for the identification*

The plant is excited by a white noise signal of $10s$ which is sent repeatedly to the plant. This signal and the sensor values are collected by the block *signal means*. This block collects the input data and creates a cumulative sum of packet of $10s$. When the generated hard realtime task is stopped, this block creates a mean of the collected signals in order to reduce the effect of sensor noise, and store it in the file *measures.txt* . The script in subsection 4.3 performs the identification of the plant. The core of the script is the function *xspectrum* (see A.4) which performs the non-parametric identification of the system (represented in Fig. 10); the parametric identification of the plant is calculated using the scilab function *frep2tf* (the 2. order term in the numerator has been eliminated). The non parametric model and the gainplot of the parametric model are represented in Fig. 10.

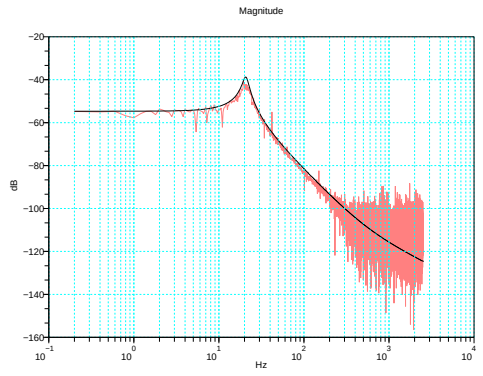

FIGURE 10: *Plot of the measured and identified system*

4.3 Identification script

```
data=read('measdata.dat',-1,3);
x=data(:,2);
y=data(:,3)/12;
ts=1/5000;

P=xspectrum(x,y);
frq=0:1/ts/2/length(P):(1/ts/2-1/ts/2/length(P));
xbasc()
gainplot(frq(2:$),P(2:$)')

N=2835;
h=frep2tf(frq(2:N),P(2:N),2);
h=ss2tf(tf2ss(h));
num=h.num
den=h.den
coeff_n=coeff(num)
coeff_n=coeff_n(1:2)
num=poly(coeff_n,'s','coeff')
sys=syslin('c',num/den)
repf=repfreq(sys,frq(2:$));
gainplot(frq(2:$),repf)
```

5 Control of the inverted pendulum

5.1 Plant description

The inverted pendulum is a classic application in a student laboratory. Our system is connected to the controller (a normal PC with Linux RTAI) through a CAN Bus. The CAN interfaces under Scicos have been realized in a three layer hierarchy: a middle layer has been introduced as a *generic CAN interface* to connect the different CAN devices to the different CAN cards (as shown in Fig. 11). In the SUPSI laboratory we have different CAN cards: for most of them we rewrote the driver in order to use them in a hard real-time environment. In particular, we have real-time drivers for CAN cards of Peak System ([11]) and for a self-built parallel port dongle.

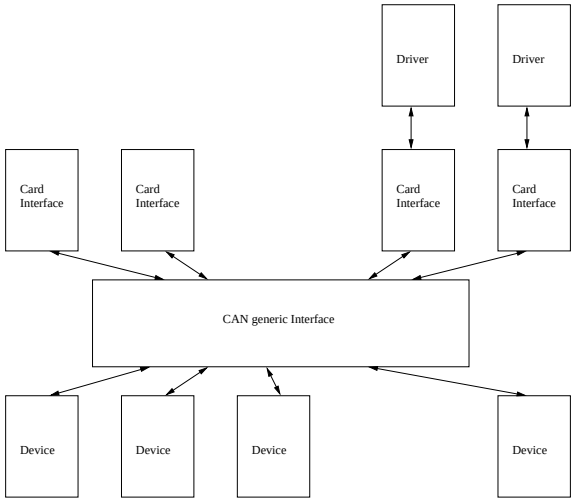

FIGURE 11: *Three layer CAN implementation*

The actuator is a DC motor from Maxon (RE 40) which is controlled by current through an Epos driver (EPOS P 24/5) ([12]). The same driver is used to count the impulses of the incremental encoder of the motor to get its position. The angle of the pole is measured using a second incremental encoder, which sends via RF the impulses to a receiver. The power supply to this encoder and to the transmitter has been realized by current induction.

5.2 Identification

The linear state-space model of the inverted pendulum is

$$\begin{bmatrix} \dot{\varphi} \\ \dot{\omega} \\ \dot{x} \\ \dot{v} \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ \frac{m \cdot r \cdot (M+m) \cdot g}{\Theta \cdot (M+m) - m^2 \cdot r^2} & 0 & 0 & \frac{-m \cdot r \cdot d}{\Theta \cdot (M+m) - m^2 \cdot r^2} \\ 0 & 0 & 0 & 1 \\ \frac{m^2 \cdot r^2 \cdot g}{\Theta \cdot (M+m) - m^2 \cdot r^2} & 0 & 0 & \frac{-\Theta \cdot d}{\Theta \cdot (M+m) - m^2 \cdot r^2} \end{bmatrix} \cdot \begin{bmatrix} \varphi \\ \omega \\ x \\ v \end{bmatrix} \quad (2)$$

$$+ \begin{bmatrix} 0 \\ \frac{m \cdot r}{\Theta \cdot (M+m) - m^2 \cdot r^2} \\ 0 \\ \frac{\Theta}{\Theta \cdot (M+m) - m^2 \cdot r^2} \end{bmatrix} \cdot F \quad (3)$$

where M is the mass of the car, m is the mass of the pole, r is the distance to the center of mass of the pole, Θ is the inertia of the pole and d is the friction constant of the car.

The force F is obtained by the torque given by the motor and it is proportional to the motor current:

$$F = K_t \cdot \frac{I_{mot}}{r_p} \quad (4)$$

where r_p represents the radius of the motor gear.

The distance r to the pole center of mass is calculated from the geometry of the pole (different volumes V_a and V_m of the pole parts)

$$r = \frac{V_a \cdot \frac{r_a}{2} + V_m \cdot (r_a + \frac{r_m}{2})}{V_a + V_m} \quad (5)$$

The value of Θ can be identified by measuring the swing frequency ($\rightarrow$ the oscillation time T_{osc}) of the pole as

$$\Theta = m \cdot g \cdot r \cdot \frac{T_{osc}^2}{(4 * \pi^2)} \qquad (6)$$

5.3 Simulation

By the simulation the inverted pendulum has been implemented using a *C-Block2* block in Scicos. The C-code of this block contains the non linear description of the inverse pendulum ([13], pages 245-247).

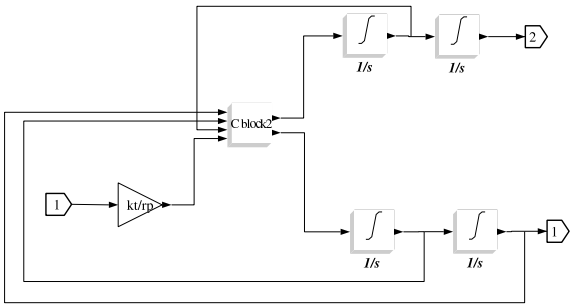

FIGURE 12: *Inverted pendulum - Block used for simulation*

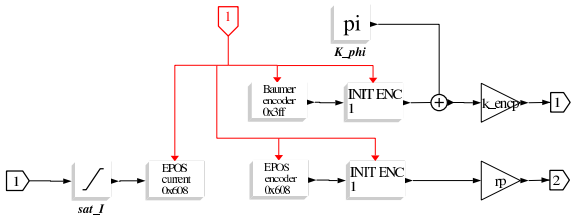

FIGURE 13: *Inverted pendulum - Block used for realtime controller*

Fig. 12 shows the superblock of the pendulum used for the simulation. The subsection 5.5 shows the code of the *C-Block2* block.

5.4 Real-time controller

Fig. 14 shows the scicos block diagram of the implemented controller.

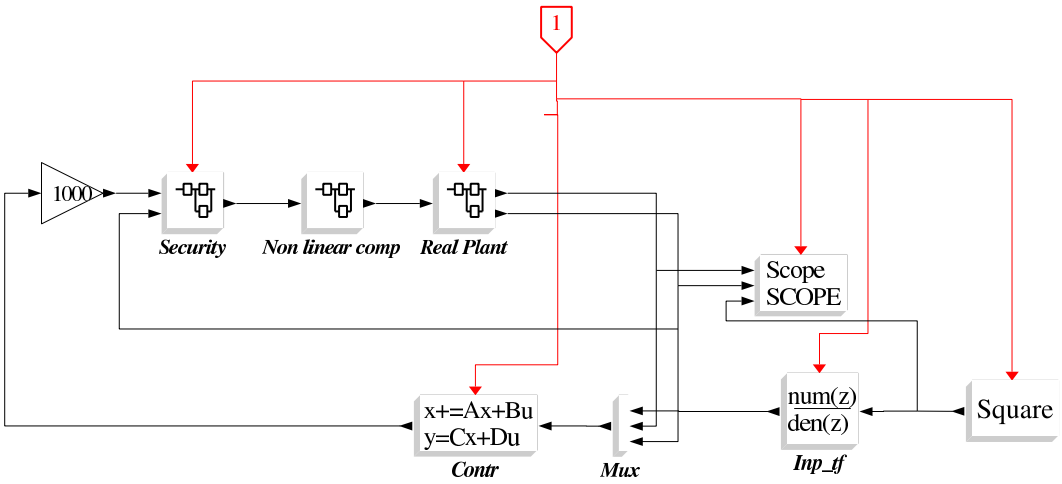

FIGURE 14: *Controller of the inverted pendulum*

All the scilab commands needed to implement the controller are shown in the subsection 5.6. The state feedback controller has been implementing using a LQR approach. The function *redobs.sci* creates the 4 matrices for the reduced order observer. As a last step, the reduced order observer, the integral part and the state feedback gains are concentrated in a single state space system, in order to avoid any possible algebraic loop in the scicos block diagram. The block of the pendulum for the real controller is represented in Fig. 13. The *Init encoder* block fixes the position of the encoder after 1s as *0* position.

A special block is represented by the *Security* block (see Fig. 15).

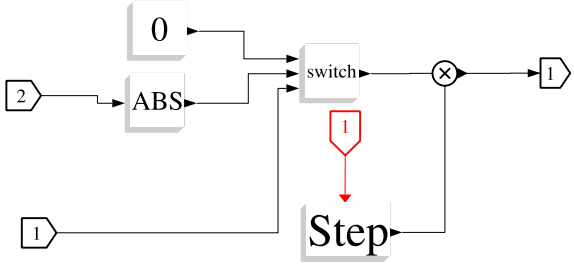

FIGURE 15: *Inverted pendulum - Security block*

The "switch" block stops the controller if the car position exits from a given interval. The *step* block is used to start the controller after an initialization time.

5.5 C-code of the non linear model

```
/* Inverted pendulum - Nonlinear model */
#include <math.h>
#include <stdlib.h>
#include <scicos/scicos_block.h>
void inv_pend(scicos_block *block,int flag)
{
  double M,m,g,Theta,r,d,th,thd,xd,u,delta,thdd,zdd;

  M=block->rpar[0];
  m=block->rpar[1];
  r=block->rpar[2];
  Theta=block->rpar[3];
  d=1.2571;
  g=9.81;
  th=block->inptr[0][0];
  thd=block->inptr[1][0];
  xd=block->inptr[2][0];
  u=block->inptr[3][0];

  delta=Theta*(M+m)-m*m*r*r*pow(cos(th),2);
  thdd=m*r*((M+m)*g*sin(th)-m*r*thd*thd*sin(th)*cos(th)+
       (u-d*xd)*cos(th))/delta;
  zdd=(m*m*r*r*g*sin(th)*cos(th)-Theta*m*r*thd*thd*sin(th)+
       Theta*(u-d*xd))/delta;
  block->outptr[0][0]=zdd;
  block->outptr[1][0]=thdd;
}
```

5.6 Control design of the inverted pendulum

```
// Inverted pendulum - State feedback control design
M=0.62336
m=0.08377+0.10834;
g=9.81;
Tosc=29/21;

r=0.026+.297;
Theta=m*g*r*Tosc^2/(4*pi^2);
d=1.2165
kt=60.3e-3;
rp=0.6/20.91044;
k_encp = -1;

Ts=5e-3;

Imin=0;  // For nonlinear compensation

A=[0, 1, 0, 0;
 m*r*(M+m)*g/(Theta*(M+m)-m*m*r*r),0,0,-m*r*d/
       (Theta*(M+m)-m*m*r*r);
 0, 0, 0, 1;
 m*m*r*r*g/(Theta*(M+m)-m*m*r*r),0,0,-Theta*d/
       (Theta*(M+m)-m*m*r*r)];
B=[0;
 m*r*kt/(rp*(Theta*(M+m)-m*m*r*r));
 0;
 kt*Theta/(rp*(Theta*(M+m)-m*m*r*r))]';

C=[1 0 0 0; 0 0 1 0];
D=[0;0];

sys=syslin('c',A,B,C,D);

sysd=dscr(sys,Ts);
[Ad,Bd,Cd,Dd]=abcd(sysd);

// LQR discrete controller
// weights
Q=diag([250,1,200,1,250]);
R=[1];

[m1,n1]=size(Ad);
[m2,n2]=size(Bd);
[m3,n3]=size(Cd);

// Add suppl. state (integral) to SS form
Ad_f=[Ad,zeros(m1,1);-Cd(2,:)*Ts,1];
Bd_f=[Bd;zeros(1,n2)];
Cd_f=[Cd,zeros(m3,1);zeros(1,n3),1];
Dd_f=[Dd;zeros(1,n2)];

k_lqr=bb_dlqr(Ad_f,Bd_f,Q,R);

preg=max(abs(spec(A)));

// Reduced order observer

poli_oss=exp([-preg,-preg]*10*Ts);
T=[0,0,0,1;0,1,0,0];
[Ao,Bo,Co,Do]=redobs(Ad,Bd,Cd,Dd,T,poli_oss);

// Compact form
Contr=comp_form_i(Ao,Bo,Co,Do,Ts,k_lqr,[0,1]);

// Filetr for input square signal
g=syslin('c',10/(s+10));
gz=ss2tf(dscr(tf2ss(g),Ts));
Ainp=0.3
```

6 Conclusion

This paper presented three practical examples explaining some basic activities connected with control system design. The Scilab/Scicos/RTAI suite has demonstrated to have reached enough maturity and to be an interesting alternative to other expensive commercial suites. This tool can be used for data acquisition, analysis, identification, control design and simulation. Just with a few steps, the same scicos block diagram used for simulation can be tranformed into a real-time controller running on a standard PC.

References

[1] D. Beal, E. Bianchi, L. Dozio, S. Hughes, P. Mantegazza, and S. Papacharalambous, "RTAI: real time applications interface," *Linux Journal*, April 2000.

[2] E. Bianchi and L. Dozio, "Some experience in fast hard real-time control in user space with RTAI-LXRT," in *Real Time Linux Workshop*, Orlando, 2000.

[3] R. Bucher and L. Dozio, "CACSD with Linux RTAI and RTAI-Lab," in *Real Time Linux Workshop*, Valencia, 2003.

[4] R. Bucher, "Interfacing Linux RTAI with Scilab/Scicos," in *Real Time Linux Workshop*, Singapore, 2004.

[5] R. Bucher, L. Dozio, and P. Mantegazza, "Rapid Control Prototyping with Scilab/Scicos

and Linux RTAI," in *International Scilab Conference*, Paris, 2004.

[6] R. Bucher and S. Balemi, "Scilab/Scicos and Linux RTAI - A unified approach," in *IEEE conference on Control Applications*, Toronto, 2005.

[7] R. Bucher, "Targeting the Scicos Code Generator - The Linux RTAI Example," in *SCILAB Research, Development and Applications*, Wuhan, China, 2005.

[8] R. Bucher, S. Mannori, and T. Netter. (2006) RTAI-Lab tutorial: Scilab, Comedi and real-time control. [Online]. Available: https://www.rtai.org/RTAILAB/RTAI-Lab-tutorial.pdf

[9] COMEDI. Linux Control and Measurement Device Interface. [Online]. Available: http://www.comedi.org

[10] S. Balemi, J. Moerschell, J.-M. Breguet, D. Braendlin, S. Bottinelli, and I. Beltrami, "Surface Inspection System for Industrial Applications," in *Conf. on Robotics and Mechatronics*, Aachen, Germany, Sept. 2004, pp. 1597–1602.

[11] Peaks System. PCAN Linux website. [Online]. Available: http://www.peak-system.com/linux/index.htm

[12] Maxon motor. Maxon motor website. [Online]. Available: http://www.maxonmotor.com/

[13] S. Campbell, J.-P. Chancelier, and R. Nikoukhah, *Modeling and Simulation in Scilab/Scicos*. Springer, 2006.

A Some useful functions

A.1 Function *redobs.sci*

```
function [A_redobs,B_redobs,C_redobs,D_redobs]=redobs(A,B,C,D,T,poles)
P=[C;T]
invP=inv([C;T])

AA=P*A*invP

ny=size(C,1)
nx=size(A,1)
nu=size(B,2)

A11=AA(1:ny,1:ny)
A12=AA(1:ny,ny+1:nx)
A21=AA(ny+1:nx,1:ny)
A22=AA(ny+1:nx,ny+1:nx)

L1=ppol(A22',A12',poles)';

nn=nx-ny;

A_redobs=[-L1 eye(nn,nn)]*P*A*invP*[zeros(ny,nn); eye(nn,nn)];
B_redobs=[-L1 eye(nn,nn)]*[P*B P*A*invP*[eye(ny,ny);L1]]*[eye(nu,nu) zeros(nu,ny); -D, eye(ny,ny)];
C_redobs=invP*[zeros(ny,nx-ny);eye(nn,nn)];
D_redobs=invP*[zeros(ny,nu) eye(ny,ny);zeros(nx-ny,nu) L1]*[eye(nu,nu) zeros(nu,ny); -D, eye(ny,ny)];
```

A.2 Function *compform.sci*

```
function [Contr]=comp_form(A,B,C,D,Ts,K)
  // Create the compact form of the Observer ABCD and the
  // gain K,
  //
  // A,B,C,D: Observer matrices
  // Ts: sampling time
  // K: stte feedback gains

  ss_sys=syslin('d',A,B,C,D);
  ss_sys(7)=Ts;
  g_sys=ss2tf(ss_sys);

  gu=g_sys('num')(:,1)./g_sys('den')(:,1);
  gy=g_sys('num')(:,2:$)./g_sys('den')(:,2:$);

  Greg=[1/(1+K*gu),-K*gy/(1+K*gu)];
  Contr=tf2ss(Greg);
```

A.3 Function *compform_i.sci*

```
function [Contr]=comp_form_i(A,B,C,D,Ts,K,Cy)
// Create the compact form of the Observer ABCD and the
// gain K, using an integrator at the input
// to eliminate the steady state error
//
// A,B,C,D: Observer matrices
// Ts: sampling time
// K: stte feedback gains
// Cy: matrix to extract the output for the steady
// state feedback

  [larg,rarg]=argn(0);

  if rarg ~= 7 then
    Cy = [1]; // only 1 output
  end

  ss_sys=syslin('d',A,B,C,D);
  ss_sys(7)=Ts;
  g_sys=ss2tf(ss_sys);

  g_int=syslin('d',Ts/(%z-1));
  g_int(7)=Ts;

  gu=g_sys('num')(:,1)./g_sys('den')(:,1);
  gy=g_sys('num')(:,2:$)./g_sys('den')(:,2:$);

  nn=size(K,2);

  Ke = K(1,nn);
  K  = K(1,1:nn-1);

  Greg=[-Ke*g_int/(1+K*gu),(Ke*Cy*g_int-K*gy)/(1+K*gu)];
  Contr=tf2ss(Greg);
```

A.4 Function *xspectrum.sci*

```
function Txy=xspectrum(x,y)

  nfft=int(size(x,'*')/2);
  wind=window('hn',nfft)';
  n=size(x,'*');
  nwind=size(wind,'*');
  index=1:nwind;
  k=fix(n/nwind);

  Pxx=zeros(nfft,1);
  Pxy2=Pxx;Pxy=Pxx;

  for i=1:k
    xw=wind.*detrend(x(index));
    yw=wind.*detrend(y(index));
    index=index+nwind;
    Xx=fft(xw(1:nfft));
    Yy=fft(yw(1:nfft));
    Xx2=abs(Xx).^2;
    Xy=Yy .* conj(Xx);
    Pxx=Pxx+Xx2;
    Pxy=Pxy+Xy;
  end

  if modulo(nfft,2)==1 then
    selct=(nfft+1)/2;
  else
    selct=nfft/2+1;
  end

  Pxx=Pxx(1:selct);
  Pxy=Pxy(1:selct);

  Txy = Pxy./Pxx;
```