Interfacing Linux RTAI with Scilab/Scicos

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

Abstract

Scilab/Scicos is an open source project developed at the INRIA (Insitut National de Recherce on Informatique et on Automatique) in France which can be used to successfully replace other commercial and expensive software in solving mathematical and control problems. This software integrates a code generator for Scicos which has been modified and completed to automatically generate a code for the RTAI-Lab environment. The result is a full open source rapid prototyping system based on Linux RTAI, RTAI-Lab and Scilab/Scicos which has been successfully used to quickly implement real-time controllers and data acquisition systems.

Users can work within a unique environment during all phases of control system design (analysis, controller design, simulation, implementation). After these first steps, a Scicos scheme can be designed in order to implement the derived control algorithm. This scheme can be completed with blocks from a new specific Scicos library which provides all the modules needed to interface the Scicos model to data acquisition cards, input signals and RTAI-Lab. Some I/O blocks allow distributed control in a real-time network or in a local machine and can be used to mix Scilab/Scicos with Matlab/Simulink/RTW generated codes.

Some applications are presented to demonstrate the capabilities and the performances of the environment.

1 Introduction

Rapid Control Prototyping techniques subsume normally the need of expensive commercial software like Matlab/Simulink ([5]) or National Instrument MatrixX ([4]). Using Scilab/Scicos and the RTAI-Lab environment ([6]) a low cost full open source system for the Linux RTAI OS can be implemented. Almost all the features of the commercial system Matlab/Simulink/RTW have been implemented except that the code generator doesn't support continous transfer functions and continous state space blocks. Actually Scilab doesn't contain an integration algorithm which can be linked to the generated C-code. A new scicos library contains the blocks needed to interface the generated code with RTAI-Lab and a set of input signals (sinus, square, step, etc.). Using the COMEDI drivers ([7]) the generated real-time code can interact with real plants. This solution has been used to implement different control systems at the SUPSI control laboratory where it successfully substitutes a previous system based on Matlab/Simulink/RTW.

At the Realtime Linux Workshop in Valencia a first implementation of this environment was presented in [6]. Since this first prototype a lot of improvements have been done:

- Sensors and actuators are now integrated in the Scicos scheme, and therefore, the user doesn't call or use an external program to configure them anymore.
- Each block can be identified by a name which can be used for parameter tuning by xrtailab.
- The RTAI add-ons for Scilab/Scicos can be now integrated in the Scilab/Scicos environment without touching or patching any original file. Installation of the system becomes easier.

All the necessary files for this new release are integrated in the last RTAI-3.1 version.

In the next sections the integration of Scilab under Linux RTAI is presented. Some examples demonstrate the capabilities of the tool.

2 Scilab

Scilab ([1]) is a scientific software package for numerical computations providing a large set of functions for engineering and scientific applications. It has been developed since 1990 by researchers from INRIA (Insitut National de Recherce on Informatique et on Automatique, [2]) and ENPC (École Nationale des ponts et chausée, [3]) and it can be freely downloaded from the Internet. Scilab is currently used in educational and industrial environments around the world.

June 2004 INRIA releases Scilab 3.0 which contains a lot of improvements compared to the previous versions.

Scilab includes different toolboxes and the possibility to add programs written in various languages(C, Fortran, ...). It contains high level data structures including arrays, lists, polynomials, rational functions and linear systems. The syntax is very similar to Matlab and the porting of applications written for this environment is very easy.

Scilab integrates different toolboxes specific for control tasks:

- General System and Control Toolbox
- Robust control toolbox
- Arma modelisation and simulation toolbox
- Identification toolbox

Scilab contains a tool called "Scicos" which allows the implementations of block diagrams in a graphical mode. Simulation of the designed scheme can be performed directly from the Scicos window. Since version 2.6 Scicos has been completed with a code generator which translate the graphical scheme into C-code for further use. This C-code generator ("CodeGeneration_sci") produces two kinds of objects:

- A dynamic library used in the scicos scheme to substitute blocks with a C-code.
- A set of C files that can be used to produce a stand-alone executable.

Scilab doesn't deliver a "main" file needed to produce a stand-alone executable. Input and output signal have to be integrated by hand in the code because the standard I/O scicos blocks can't be handled by the code generator.

3 Interfacing Scilab with Linux RTAI

3.1 The RTAI solution

RTAI-3.1 contains all the files needed to integrate the generated code in the RTAI environment. The RTAI add-ons are composed by:

- A modified Scicos code generator ("RTAICodeGen_.sci").
- A "main" file ("rtmain.c").
- A Scicos library with the RTAI specific blocks ("RTAI-Lib.cosf").
- A RTAI library with the code needed by the I/O blocks ("libsciblk.a").
- Utilities to integrate new I/O devices in the environment ("gen_dev").

3.2 Installing the software

The installation of the RTAI add-ons for Scilab is very simple. After installing Scilab the user has to perform two steps:

1. Add the RTAI specific files to Scilab.
2. Add some info to the user scilab start-up file.

The first step adds a new directory "RTAI" to the Scilab macros. This directory contains the scicos block library "RTAI-Lib.cosf", the RTAI specific code generator "RTAICodeGen_.sci" and the scilab files related to the new RTAI-Lab blocks.

The second step modifies the user start-up file ".scilab" in order to add the RTAI-Lib library to the scicos palettes and a new scicos menu for the RTAI specific code generation.

No original file from Scilab/Scicos is modified by these steps!

A "Makefile" is provided in order to perform these two steps. User has to:

1. Modify the macro "SCILAB_DIR" in the "Makefile" to fit the Scilab installation.
2. Run "make install" as superuser to install all the files.
3. Run "make user" as normal user to modify his own startup script.

3.3 Implementation

3.3.1 The code generator

The new code generator specific for Linux RTAI contains only the stand-alone generator with a lot of modifications, in particular:

- Includes the name of the blocks in the generated files.
- Improves the model_io.c in order to facilitate the possible integration of sensors and actuators by hand.

The code generator produces the following files:

<**model**>**_standalone.c** contains the C code of each block.

<**model**>**_io.c** : if the super-block has input or output ports, this file allows to integrate by hand the code to handle them.

<**model**>**c.c** contains the code from C-Blocks (blocks implemented directly in C language) and from the RTAI specific blocks (sensors and actuators from the scicos RTAI Library).

<**MODE**>**_Makefile** is the makefile needed to compile and create the stand-alone executable.

3.3.2 The main file

The main file performs two basic tasks:

- Handles the real-time task (initialization, ISR, termination).
- Communicates with the monitor application (e.g. "xrtailab").

The real-time task is handled by the following code:

```
iopl(3);
rt_task_use_fpu(rt_BaseRateTask, 1);
NAME(MODEL,_init_blk)();
grow_and_lock_stack(stackinc);
if (UseHRT) {
  rt_make_hard_real_time();
}
rt_send(rt_MainTask, 0);
rt_task_suspend(rt_BaseRateTask);
t0 = rt_get_cpu_time_ns();
rt_task_make_periodic(rt_BaseRateTask, rt_get_time() +
                      rt_BaseRateTick, rt_BaseRateTick);
while (!endBaseRate) {
  WaitTimingEvent(TimingEventArg);
  if (endBaseRate) break;
  TIME = (rt_get_cpu_time_ns() - t0)*1.0E-9;
  set_nevprt(nevprt);
  NAME(MODEL,main1)(NAME(block_,MODEL),z, &TIME);
  NAME(MODEL,main2)(NAME(block_,MODEL),z, &TIME);
}
if (UseHRT) {
  rt_make_soft_real_time();
}
NAME(MODEL,_end)(NAME(block_,MODEL),z, &TIME);
```

The default "WaitTiminigEvent" function is defined as "rt_task_wait_period". The user can overwrite this function using the "-e" by starting the real-time task; this method can be used to implement asynchronous systems.

3.3.3 The block library

Under Scicos, a new "palette" has been created specifically for the RTAI environment. In particular this palette contains:

- Input signals (sinus, square, step) to substitute the scicos input signals that can't be used for code generation.
- RTAI-Lab specific modules (scope, led, meter, fifo).
- Blocks needed to interface with COMEDI devices.
- Other specific I/O devices not yet covered by the COMEDI project.

Figure 1 shows the Scicos RTAI-Lib library with the present implemented blocks.

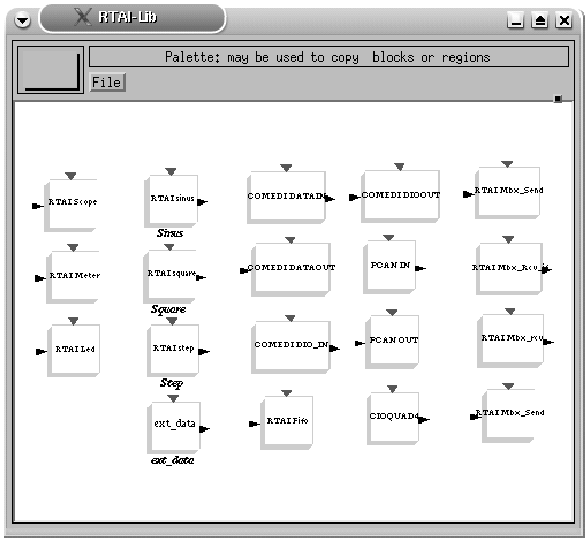

FIGURE 1: *Scicos RTAI Library*

All the blocks have been implemented starting from the original "C-Block2" from Scicos. Except for the input signals, part of the C-code is implemented in the specific "<block>.sci" function and part is implemented in the "libsciblk.a" library which is integrated in the RTAI project.

4 A simple example

4.1 Scheme

In the following a simple example will be analyzed. A sinus signal is sent to a DA-Card, read from an AD-Card and sent to the RTAI Scope. The I/O devices are implemented using the COMEDI driver.

4.2 Implementation

4.2.1 Designing the scheme

First of all the user has to design the block diagram using Scicos. In this example all the block can be extracted from the RTAI-Lib palette.

Figure 2 represents the Scicos scheme of this example.

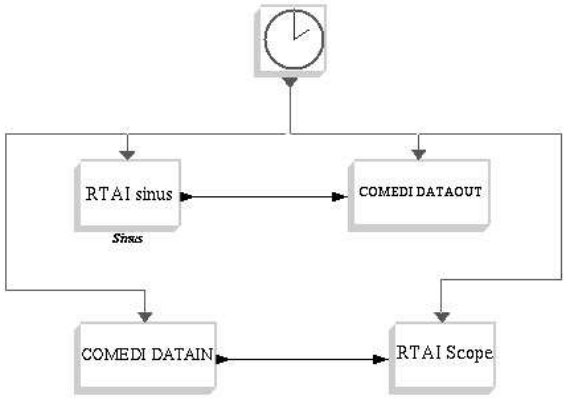

FIGURE 2: *Scicos scheme*

At present each input respectively each output block should have a different port number:

- RTAI sinus: 1
- COMEDI DATA IN: 2
- COMEDI DATA OUT: 1
- RTAI Scope: 2

Scilab/Scicos generates a code for a "super block" and not for a full scheme. To create a super block the user should choose the Scicos menu "Diagram → Region to Super Block" and select all the blocks except the clock. He obtains the system shown in figure 3.

FIGURE 3: *Super Block*

The default name of the "superblock" is "Untitled". Before starting with the code generation and compilation, the "superblock" can be renamed:

1. Open the Super Block by clicking on it.
2. Open the Scicos menu "Diagram → Rename" and give a new Name (default is "Untitled").
3. Close the Super Block.

Now the user can set the sampling time simply by opening the "Clock" block and giving the desired value under "Period".

The code generation and compilation will now be performed with the new Scicos Menu "RTAI → RTAI CodeGen". A dialog box allows to change the default compilation parameters.

5 Didactic example

5.1 Plant

The inverted pendulum of Figure 4 has been controlled by software generated with this environment.

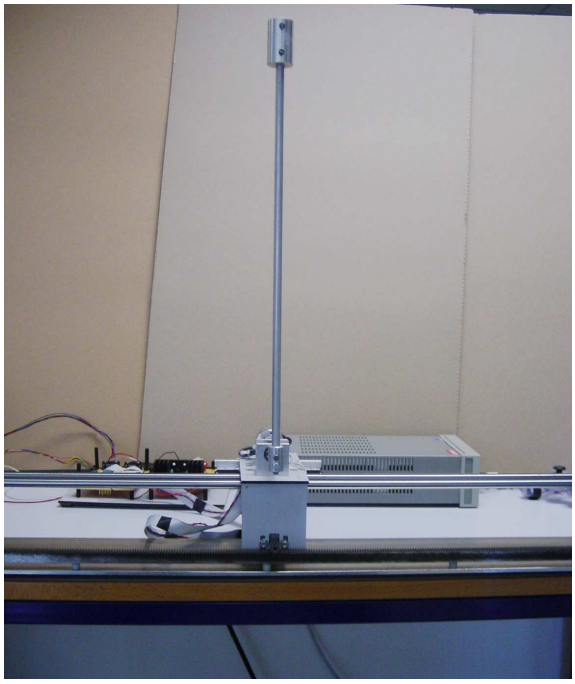

FIGURE 4: *The Plant*

5.2 The real plant

Figure 5 shows the real inverted pendulum with the I/O modules.

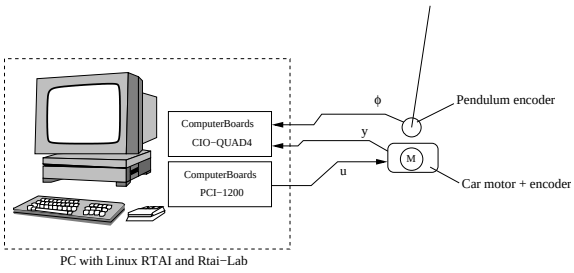

FIGURE 5: *I/O scheme of the inverted pendulum*

The PC generates the control signal and gives it to the plant through a ComputerBoards DA Card (PCI-1200). The pole and cart positions are read using an encoder card (ComputerBoards CIO-QUAD4).

5.3 Control of the inverted pendulum

A state-space feedback controller with reduced order observer has been implemented. The Scicos model contains the controller-observer pair implemented with 2 discrete transfer functions.

Figure 6 shows the controlled system. The state feedback gains (K_{lqr}) can be calculated using a LQR approach. The missing states can be obtained using a reduced order observer.

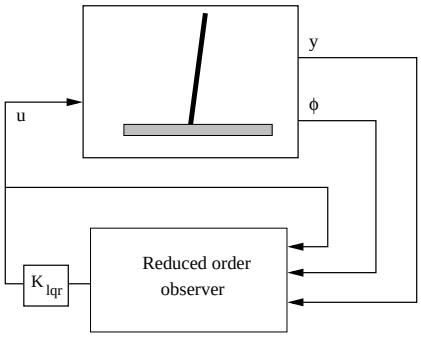

FIGURE 6: *Control scheme of the inverted pendulum*

The state feedback can be written as

$$u = -K_{lqr} \cdot \mathbf{x} \tag{1}$$

The reduced order observer can be calculated using the provided function "redobs" (see appendix A). The state space representation of the observer can be transformed into two transfer functions "G_u" (transfer function between the input u and the states $\mathbf{x}$) and "$\mathbf{G_y}$" (transfer function between the output signals $[y(t); \varphi(t)]$ and the states $\mathbf{x}$).

Figure 7 shows the signal flow diagram of the controller.

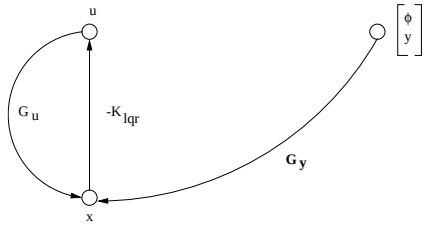

FIGURE 7: *Signal flow diagram of the controller*

The transfer function of the reduced order observer is given by

$$x = G_u \cdot u + \mathbf{G_y} \cdot \begin{bmatrix} \varphi \\ x \end{bmatrix} \tag{2}$$

Inserting 2 into 1 gives

$$u = \frac{-K_{lqr} \cdot \mathbf{G_y}}{1 + K \cdot G_u} \cdot \begin{bmatrix} \varphi \\ x \end{bmatrix} \tag{3}$$

The script in appendix A shows how easy the full controller can be programmed under Scilab.

5.4 Implementation

The controller has been implemented under Scicos as shown in figure 8

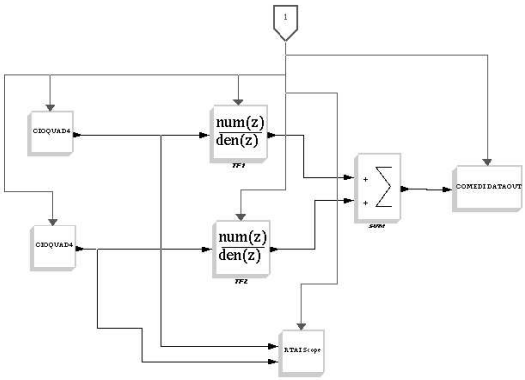

FIGURE 8: *Super block of the controller*

More details about this project can be found at the author's home page ([9]).

5.5 Distributed control

The same mechanism used to communicate between the real-time task and the RTAI-Lab GUI can be used to establish a communication between different hard real-time tasks. This makes it possible to create a distributed control system within a single PC or in a LAN, using the net_rpc layer and the rtnet hard real-time network driver ([8], see Figure 9).

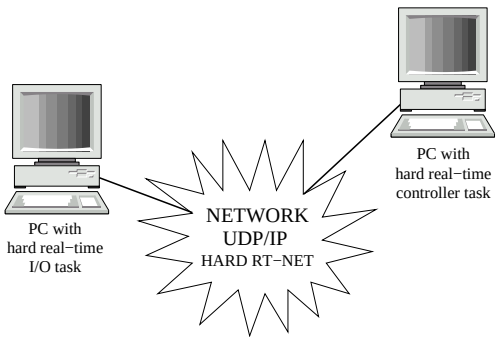

FIGURE 9: *Distributed control for the pendulum task*

Figure 10 shows the two super-blocks for the distributed control.

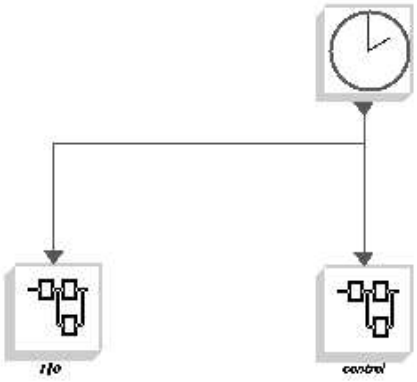

FIGURE 10: *Main scicos scheme*

The contents of the two super-blocks are represented in the figures 11 and 12. The I/O task reads and sends the sensor values using the RTAI mailbox "SENS". The control task reads these values from the mailbox "SENS", calculates and sends the control value using the mailbox "CTR". The I/O task gets the value from the mailbox "CTR" and writes it to the COMEDI device.

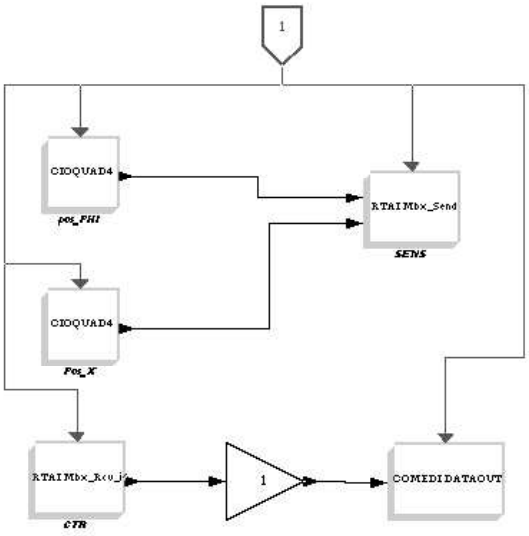

FIGURE 11: *Contents of the superblock "I/O"*

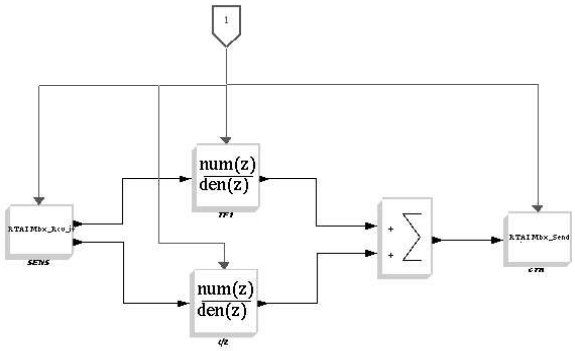

FIGURE 12: *Contents of the superblock "control"*

This distributed control system has been implemented and tested at the SUPSI. The I/O task and the control task were on two different PCs in the SUPSI LAN.

6 Conclusions

A low cost, high quality and full open source rapid control prototyping system have been presented. Different applications at the SUPSI laboratory have demonstrated the potentialities of this tool. In the last year it has successfully substituted a previous commercial system based on Matlab/Simulink/RTW as educational environment, but industrial applications are already planned.

References

[1] www.scilab.org

[2] www.inria.fr

[3] www.enpc.fr

[4] www.ni.com

[5] www.mathworks.com

[6] R. Bucher, L. Dozio, 2003, *CACSD under RTAI Linux with RTAI-Lab*, Realtime Linux Workshop, Valencia, Spain

[7] www.comedi.org

[8] www.rts.uni-hannover.de/rtnet

[9] www.dti.supsi.ch/~bucher/scilab.html

A Scilab script for the inverted pendulum controller

```
// Inverted pendulum!

Ts=1e-3;   // Sampling time

// System variables

M=1.316;            // cart mass
g=9.81;
m=0.10632+0.08555;  // Pole mass
r=331.33e-3;        // Pole length
J=m*r^2;            // Inertial
alfa=3.656;         // alfa=kt*kb/(M*ra*r^2)+d/M
kangle=1;           // angle constant
kx=123.6374/0.75;   // position constant [rad/m]
u0=1.5;
k=122.317/kx;       // k=kt/(ra*rr*M)

// Linearized system

den=J*(M+m)-m^2*r^2;

a=[0,1,0,0;
  (m*r*(m+M)*g)/den,0,0,-m*r/den*alfa*M;
   0,0,0,1;
   (m^2*r^2*g)/den,0,0,-J/den*alfa*M];
b=[0;
   m*r/den*k*M;
   0;
   J/den*k*M];
c=[kangle,0,0,0;
   0,0,kx,0];
d=[0;
   0];

sys=syslin('c',a,b,c,d);

sysd=dscr(sys,Ts);
[ad,bd,cd,dd]=abcd(sysd);

// State feedback controller [LQR]

// weights

Q=diag([1000,1,10000,1]);        // 4 by 4
R=[1];                            // 1 by 1

k_lqr=bb_dlqr(ad,bd,Q,R);

E=spec(ad);

plqr=abs(real(log(E)/Ts))';
pmax=max(plqr);

k_lqr=-k_lqr;

preg=spec(a);

// Reduced order observer

poli_oss=exp([real(preg(3)),real(preg(4))]*10*Ts);
T=[0,0,0,1;0,1,0,0];
[ao,bo,co,do]=redobs(ad,bd,cd,dd,T,poli_oss);

Greg=comp_form(ao,bo,co,do,Ts,k_lqr);

[g1n,g1d]=tfdata(Greg(:,2));
[g2n,g2d]=tfdata(Greg(:,3));

//*********************************************************

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)];
endfunction

//***********************************************************

function [Gu,Gy]=get_gu_gy(G)
Gu=G('num')(:,1)./G('den')(:,1);
Gy=G('num')(:,2:$)./G('den')(:,2:$);
endfunction

//***********************************************************

function [Greg]=comp_form(A,B,C,D,Ts,K)

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

[gu,gy]=get_gu_gy(g_sys);

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

//***********************************************************

function [num,den]=tfdata(G)
num=G('num');
den=G('den');
endfunction
```