CACSD under RTAI Linux with RTAI-LAB

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

Lorenzo Dozio
Dipartimento di Ingegneria Aerospaziale del Politecnico di Milano
Via La Masa 34, Milano 20156, Italy
dozio@aero.polimi.it

Abstract

This paper describes a tool named RTAI-Lab, which provides a common structured framework to design, build, run and monitor any suite of RTAI-based single/multitasking controllers and real time simulators, either specifically coded in a high level procedural language, typically C/C++, or automatically generated by proprietary MATLAB / Simulink/Real-Time-Workshop, and/or fully free open source SCILAB/Scicos/CodeGeneration.

1 Introduction

Computer Aided Control System Design (CACSD) subsumes a broad variety of computational tools and environments for control system design, real time simulation, with and without hardware in the loop, making the best use of high desktop computer power, graphical capabilities and ease of interaction with low hardware cost. Integrated CACSD software environments allow an iteractive control system design process to be automated with respect to multi-objective performances evaluation and multi-parameter synthesis tuning. Visual decision support provides the engineer with the clues for interactively directing an automated search process to achieve a well balanced design under many conflicting objectives and constraints. Local/remote on line data down/upload makes it possible a seamless interaction with the control system, in order to supervise its operation and to adapt it to changing operational needs.

RTAI-Lab is an open source project having the goal to provide a common structured framework for the integration of RTAI into CACSD environments. The present implementation includes support for the commercial MATLAB/Simulink/Real-Time-Workshop (RTW) suite and the open source SCILAB/Scicos suite. The RTAI-Lab internal architecture should allow easy porting to other CACSD software. Moreover it can be used to run and monitor any suite of RTAI-based single/multitasking controllers not derived by any CACDS tool, but specifically coded in a high level procedural language, typically C/C++.

Basically RTAI-Lab relies completely upon the CACSD software for the control system code design and generation, with only the addition of some specific blocks and building options. The generated code is then embedded in a RTAI framework to be executed in soft/hard real time and monitored by a generic graphical user interface, with no more need of the related CACSD environment. The interface to the real time application involves changing tunable parameters on the fly, and scoping and logging signals and generic multidimensional data. The building/running and interfacing frameworks represent the two main aspects of RTAI-Lab, which will be separately outlined in the next section.

It is believed that automatic control system generation from visual block schemes cannot meet the performances achievable by direct coding. In fact the intrinsic generalization of such procedure restricts the level of the final optimization of the whole assembled code, even if a single built-in piece of code implementing a specific operation could be highly efficient. This could be a problem for real time systems. Nonetheless the high computational power afforded by today general-purpose CPUs makes it possible to avoid caring some efficiency loss in favour of a faster and shorter development cycle allowed by the tighter design, simulation and control system implementation that comes with such integrated CACSD environments.

2 RTAI-Lab architecture

The basic concept of RTAI-Lab is to allow two separate systems, the host and the target, to communicate. In a remote implementation, the host is the machine where the RTAI-Lab graphical user interface (GUI) is executing in soft real time, the target is the machine where the generated hard real time code runs. In a local setup, the two processes run on the same hardware communicating by means of local messages. The remote interfacing is implemented by exploiting the features of the small and effective RTAI real time middleware layer, called net_rpc. The host sends/receives messages using net_rpc requesting the target to accept parameters changes and to send signal data for graphical displaying and/or file logging.

Such a scheme has been derived by the native client/server architecture of some CACSD software such as the MATLAB external mode. In this environment MATLAB/Simulink is the client and the target is the server. External mode works by establishing a communication channel between the Simulink process and the Real-Time-Workshop generated code. This channel is implemented by a low-level independent transport layer that handles physical transmission of messages. In RTAI-Lab this layer is fully integrated into the real time code via net_rpc without the need to adapt and link it with an ad hoc transport layer implementation. Thus it is possible to run on the same machine many RTAI-Lab GUI sessions, in order to monitor and interface many sets of targets simultaneously. Moreover the adopted scheme allows to extend the present support by providing the capability to communicate with a network of PCs running a distributed control system. The interoperability of local/remote/distributed usage is provided by setting some identifiers, as well as the remote node address. The identifiers refer to the real time objects of the executables which are devoted to the host interface, and thus permit to distinguish and separate corresponding environments of RTAI-Lab sessions simultaneously executing on the same host.

From the code building/running side, the setting up of the control system within the CACSD supported environments is performed in three main steps. First, the designer creates a Simulink/Scicos block visual diagram that represents the system implementing the chosen control strategy. Such a diagram is composed by blocks coming from the CACSD software built-in library, along with some specific RTAI and DAQ blocks providing the interface to the RTAI-Lab monitoring and I/O support, respectively. In this way the integration of sensors and actuators in the control scheme is fully accomplished. The second step involves the C code generation. The process is completely straightforward and the user has to choose only the right template makefile for the target language compiler to have the control programming automatically generated by a mouse click. At this point the generated code should be compiled by using the mating generated makefile and linked with the RTW/Scicos specific interface code to the RTAI-Lab GUI framework, called rtmain, to obtain the real time target executable. The code building is such that the final standalone executive can be run along with a set of options that allow choosing between soft/hard real time execution modes, periodic/oneshot timing, internal/external timer source, infinite/finite final time. The standalone code can be put on any machine running the same version of RTAI used to build it without the need of the related CACSD software.

The module rtmain creates a suitable frame onto which the generated control system runs. This architecture involves a task, called rt_Main(), that initializes all the real time stuff and manages the start/stop/final-time events of the real time executive, a task, called rt_HostInterface(), that creates a double way communication channel from/to the RTAI-Lab interfacing counterpart, and as many hard timed tasks as the number of different sampling rates of the corresponding CACSD model containing the computation of the control system operations along with the I/O analog-to-digital and digital-to-analog ones. Taking the MATLAB/Simulink/RTW development environment as an example, there are two execution modes, called singletasking and multitasking. In the first mode, the model operations are included in one hard timed task called rt_BaseRate(), which runs at the selected base rate of the model. In the multitasking mode, multiple tasks are created, one for each sample rate in the model. The task rt_BaseRate() executes the components of the Simulink model code run at the highest sample rate and thus it is the highest priority RTAI task. The module rtmain also spawns a separate thread rt_SubRate() for each additional sample rate in the system, scheduled in a rate monotonic order.

3 RTAI-Lab GUI

The RTAI-Lab graphical user interface (GUI) has been fully redesigned for the latest version of RTAI (rtai-24.1.12) using the last version of the EFLTK library. This graphical library is an FLTKv2 fork with some extended capabilities of the available FLTK widgets and the addition of some powerful widgets especially developed for the building of a simple, fast and minimal memory usage desktop environment called Equinox.

The RTAI-Lab GUI (Figure 1) is based on a Multiple Document Interface widget, that allows to have a main application window with a menu bar, a tool bar, and a status bar. All the managers and instruments windows are created inside the central widget. The menu and tool bars contain the user commands to connect/disconnect to/from the real time target, to start/stop the real time application, and to open/close the manager/instrument windows. The connect command opens a dialog widget to select the identifiers of the remote real time code. The parameters to insert includes the remote node address (0 if the application is local), and the strings associated to the real time objects of the available instruments. These values should be the same of those used in the running target options. The default names correspond to a target executed without specifying those options.

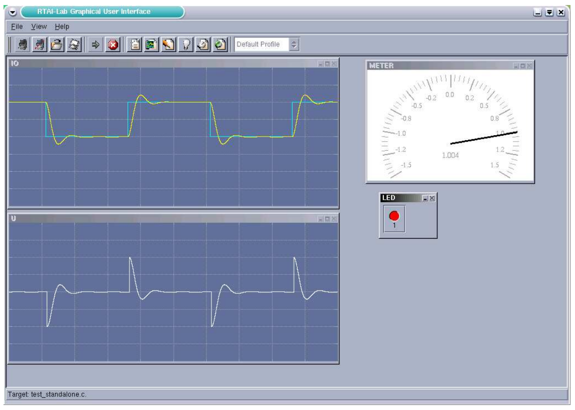


FIGURE 1: *RTAI-Lab graphical user interface*

The present version makes available to the user three types of measurement instruments, digital scopes, leds and meters, with their corresponding manager windows, one manager to view and change the target tunable parameters on the fly, and one window to manage the logging of generic multidimensional data.

Each scope canvas (Figure 2) can show multiple traces of different colors, offsets and amplitude scales. The manager of the digital scopes (Figure 3) shows on the left side the list of available scopes with the name given by the user in the CACSD design diagram, and allows to view/hide each trace independently, to view/hide the grid, to change the background and grid colors, to pause the scope visualization, to select the second per division value, and to manage the logging of the signal data.

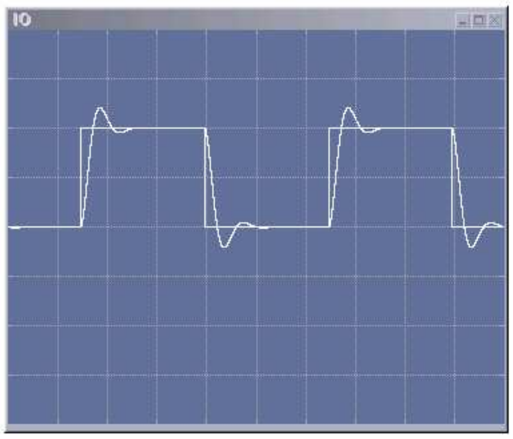
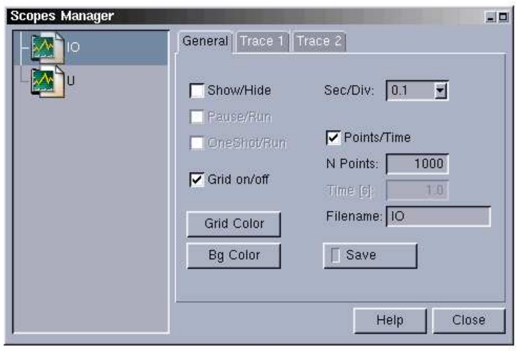


FIGURE 2: *RtaiLab - Scope*



Scopes Manager

IO
U

General | Trace 1 | Trace 2

☐ Show/Hide
☐ Pause/Run
☐ OneShot/Run
☑ Grid on/off
Grid Color
Bg Color

Sec/Div: 0.1
☑ Points/Time
N Points: 1000
Time [s]: 1.0
Filename: IO
Save

Help
Close

FIGURE 3: *RtaiLab - Scope Manager*

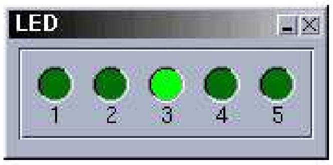


FIGURE 4: *RtaiLab - Led*

The led instrument is a simple panel of coloured binary leds (Figure 4). The related manager windows (Figure 5) allows to show/hide the selected led panel and to change the corresponding active/passive color.

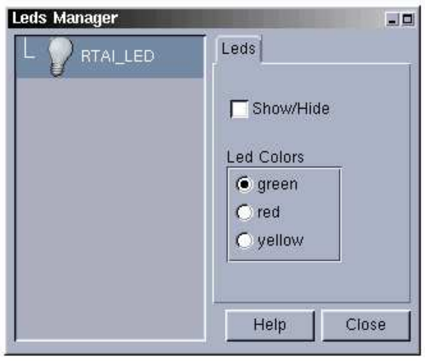


FIGURE 5: *RtaiLab - Led Manager*

The meter instruments (Figure 6) are digital representations of analog meters, along with the visualization of the current value . The meters manager (Figure 7) allows to view/hide the meter selected in the left side list, to set the minimum and maximum values of the graduate scale, and to change the color of the arrow, the grid and the background.

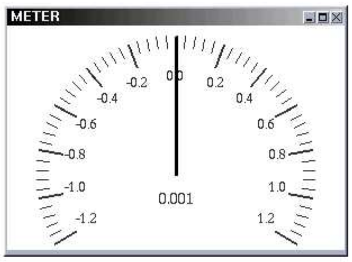


FIGURE 6: *RtaiLab - Meter*

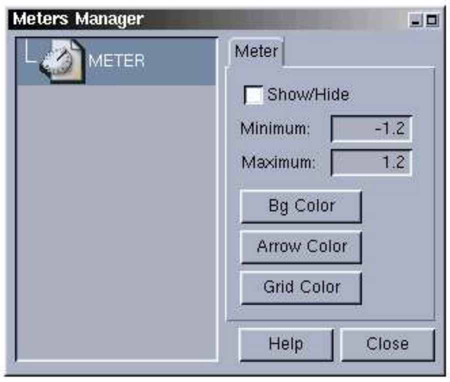


FIGURE 7: *RtaiLab - Meter Manager*

The target tunable parameters can be viewed and modified by means of a specific manager window (Figure 8). It lists the scalar, vector and matrix blocks of the CACSD model; for each block it shows the name and the current value of the related parameters. The single parameter can be changed by simply typing the new value inside the corresponding input widget. By default, the new value will be downloaded to the real time target when the enter key is pressed. If the batch download option is selected, many values can be modified without affecting directly the related target ones. The changes will be effective only when the download button is clicked.

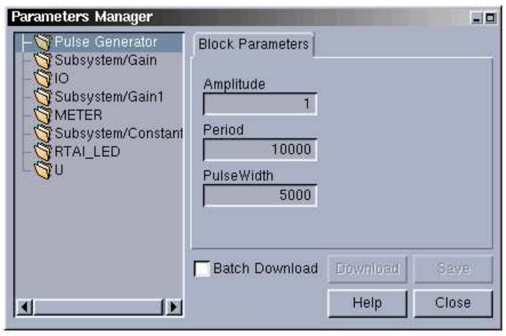


FIGURE 8: *RtaiLab - Parameter Manager*

The generic data logging manager window have been included to allow to log multidimensional data. The log options are quite similar to those included in the logging part of the scopes manager, with the capability to select the number of points to save and the name of the log file.

4 Supported CACSD environments

4.1 Matlab/Simulink/RTW

The Matlab/Simulink/RTW suite [1] is a commercial product widespread in universities and industries. Near to an accurate graphical interface it gives the possibility to create C code, using the Realtime-Workshop Toolbox. The generated code can be easily adapted and downloaded to different targets (see [3]. [4] e [5]). The first integration of a CACSD system in the Linux RTAI environment was implemented using Matlab/Simulinik/RTW in kernel space ([6]), followed by a similar implementation in user space under LXRT ([7]). The present implementation works with newlxrt and integrates the possibility to use the COMEDI drivers [8].

One of the advantages of this suite is the integration of sensors and actuators in the control scheme. Sensors and actuators are directly realized as C-MEX

S-Functions and integrated into the generated code. No more steps are needed before using the resulting executable.

On the other hand it should be mentioned that licences for Matlab are very expensive, in particular the licence for RTW. This licence is required if the user wants to change parameters and monitor signals using the RTW external mode.

The core of the implementation under Matlab is represented by the 2 target files "rtai.tlc" and "rtai.tmf" and by the different C-MEX S-Functions that implement the data transfer to the RTAI-Lab environment and to the drivers of the DAQ cards (COMEDI or user specific). Different input signals can be simply implemented using the normal Simulink source blocks. Figure 9 shows the current Simulink RTAI Library.

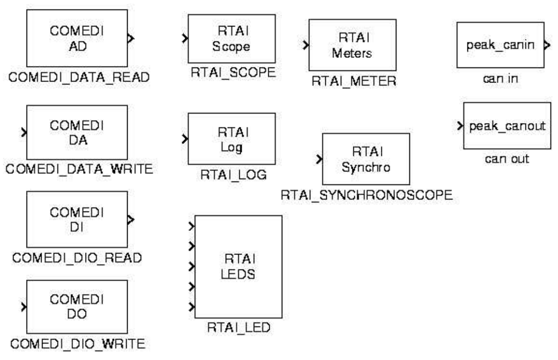


FIGURE 9: *Simulink RTAI Library*

4.2 Scilab/Scicos/CodeGen

The Scilab/Scicos suite [2] is a full open source project wich integrates similar functionalities like Matlab. The graphical environment is less sophisticated than that of Simulink, but it is complete and functional. The code generator is now integrated in the Scicos macros, and it has been modified to allow the generation of RTAI-Linux specific code. Basically, the code generator macro "CodeGeneration" transforms a part of the Scicos scheme included within a Super block into a dynamic function used to speed up the simulation under Scilab/Scicos. A modification of this macro allows the preparation of the files needed to create the RTAI executable.

Under Scicos, the integration of the I/O blocks can't be made directly in the scheme. The user has to integrate I/O blocks in a second step, before the compilation.

The I/O blocks have been developed separately and are available in a library (ulibsci.a), including blocks to integrate COMEDI drivers into the Scicos executable. A common interface for all the blocks facilitates the program and integration of new blocks in this library. A set of utilities to automate these operations is provided.

5 A simple example

5.1 Scheme

In the following, a simple example will be analized in both environment, Matlab/Simulink and Scilab/Scicos. The system is represented by a transfer function

$$Gs(s) = \frac{20}{s^2 + 4s}$$

with unity feedback,

In both cases the system has been implemented as discrete-time transfer function

$$Gz(z) = 10^{-6}\frac{9.987z + 9.973}{z^2 - 1.996z + 0.996}$$

with a sampling time of $1ms$. Different signals are sent to scopes, meters, and leds.

In both cases the model is saved with the name "test".

5.2 Implementation under the Matlab environment

Figure 10 shows the Simulink scheme of this system.

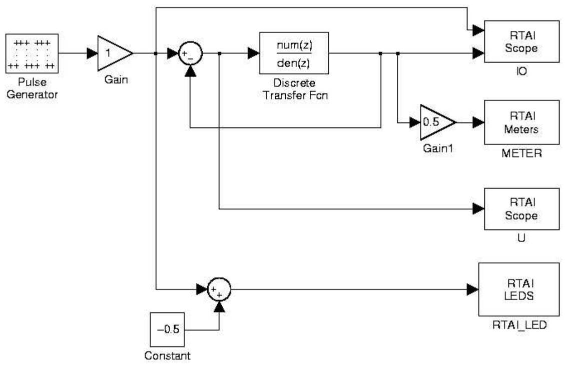


FIGURE 10: *Simulink scheme*

The I/O blocks are integrated as C-MEX S-Functions in this scheme. The code generation runs along the following steps:

- choose "rtai" as "System target file" under "Real-Time Workshop - Options"
- start "Real-Time Workshop - Build Model"

An executable with the same name as the model ("test") will be generated.

5.3 Implementation under Scilab

Under Scilab/Scicos the code generation is not as simple as under Matlab/Simulink. Figure 11 represents the Scicos scheme of the example.

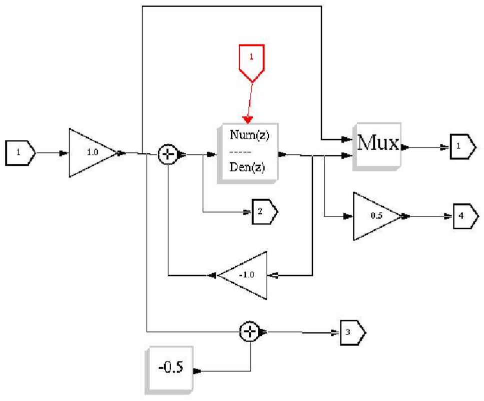


FIGURE 11: *Scicos scheme*

In order to generate the code this scheme must be transformed into a "Super Block" which can be used to generate the code. The result is a compiled function into the scheme (Figure 12).

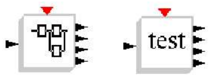


FIGURE 12: *Super block before and after the compilation*

The next steps are to be executed outside of the Scilab environment. First, the user must declare the input and the 4 outputs of the system using a configuration file. The file "config" needed by the example is:

```
rtai_scope out 1 2 IO 0 0 0 0 0
square inp 1 0 0 1.0 10.0 5.0 0.0 0.0
rtai_scope out 2 1 U 0 0 0 0 0
rtai_led   out 3 1 LED 0 0 0 0 0
rtai_meter out 4 1 METER 0 0 0 0 0
end
```

For each block following items have been defined:

- the type (example: rtai_scope)
- input (inp) or output (out) port
- port number
- identifier or channel number
- a name
- 5 parameters (block dependent)

Some parameters are not used in some blocks. Example:

```
square inp 1 0 0 1.0 10.0 5.0 0.0 0.0
   |    |  | | |  |    |    |   |   |_delay
   |    |  | | |  |    |    |   |_bias
   |    |  | | |  |    |    |_pulse width
   |    |  | | |  |    |_period
   |    |  | | |  |_amplitude
   |    |  | | |_name (unused)
   |    |  | |_id/ch (unused)
   |    |  |_port on scicos model
   |    |_sensor
   |_type is square generator
```

Using the utility "gen_io" the file "test_void_io.c" generated by scicos will be modified into the file "test_io.c", which includes now the calls to the I/O functions.

The last step is given by the compilation of the "test_standalone" executable file, which is performed with the following command:

```
make -f test_Makefile test_standalone
```

6 Laboratory applications

6.1 Plant

The following example shows how RTAI-Lab can be used to design and implement a controller for the 4th order plant shown in Figure 13.

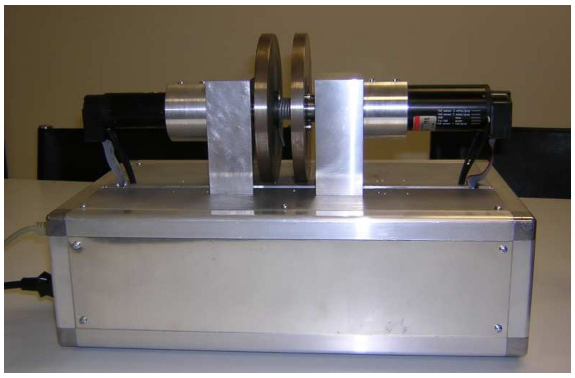


FIGURE 13: *Plant*

Two rotating motor-disk systems are connected by a spring. The position of the left disk must be controlled by appliying the control signal to the motor on the right. Both motor positions are collected. The left motor can be used to introduce a disturbance signal to the left mass. A CAN bus dongle plugged in the parallel port (Figure 14) connects the controller (PC with Linux RTAI) to the motor drivers. The controller works with a sampling time of 10ms. The system was developed as student project at the University of Applied Sciences of Southern Switzerland (SUPSI).

FIGURE 14: *Peaks can dongle*

The identification task takes advantage of the RTAI-Lab capability to collect signals in real time. The identification task has been designed under Simulink (Figure15): the two current inputs I_1 and I_2 and the two outputs φ_1 and φ_2 are collected.

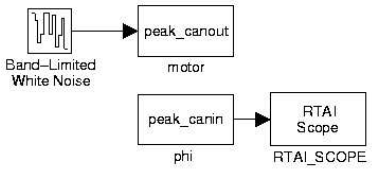


FIGURE 15: *scheme for the identification*

A parametric identification routine delivers the state space model

$$\begin{bmatrix} \dot{\varphi}_1 \\ \dot{\omega}_1 \\ \dot{\varphi}_2 \\ \dot{\omega}_2 \end{bmatrix} = A \begin{bmatrix} \varphi_1 \\ \omega_1 \\ \varphi_2 \\ \omega_2 \end{bmatrix} + B \begin{bmatrix} I_1 \\ I_2 \end{bmatrix}$$

where φ and ω represent *angle* and *velocity* of the two masses, I_1 and I_2 represent the input current to the motors.

Starting from this model a state feedback controller with integral compensation of the static error has been implemented in both Matlab and Scilab environments. The feedback values have been calculated using a LQR approach. A reduced order observer has been implemented to obtain the missing states. Under Matlab/Simulink the controller has been implemented using a state space realization as shown in the Figure 16.

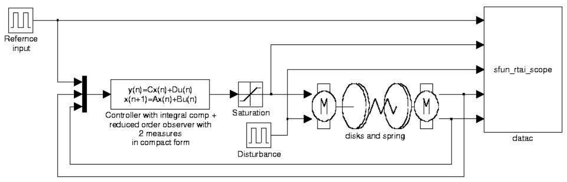


FIGURE 16: *Simulink scheme*

A mathematical model of the disks and spring plant (Figure 17) used for simulating the closed loop systems has to be substituted with a block containing the I/O interfaces (Figure 18) before the compilation of the RTAI real-time controller.

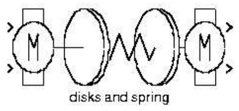


FIGURE 17: *Model of disks and spring*

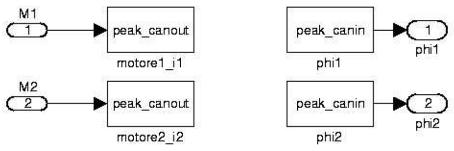


FIGURE 18: *I/O interfaces of the system*

Under Scilab/Scicos the compilation of "Super block" containing state space models is not yet possible. In order to solve this problem the controller has been implemented using discrete-time transfer functions. Figure 19 shows the controller implemented under Scicos.

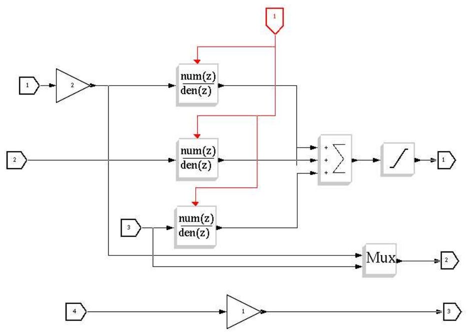


FIGURE 19: *Scicos controller*

The I/O blocks have been implemented using the following configuration file:

```
square     inp 1 0     0     1    20    10 0 0
pcan       inp 2 1537  0     8000 4000  0 0 0
pcan       inp 3 1538  0     1000 500   0 0 0
square     inp 4 0     0     3000 10    3 0 23
pcan       out 1 1537  0     8000 4000  0 0 0
rtai_scope out 2 2     Scope 0    0     0 0 0
pcan       out 3 1538  0     1000 500   0 0 0
end
```

Figure 20 shows the running system.

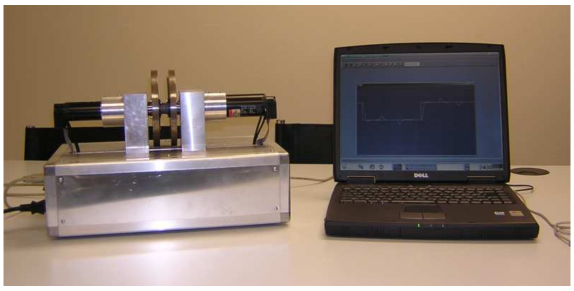


FIGURE 20: *Running system*

7 Conclusions

RTAI-Lab represents a powerful tool to implement control applications for the Linux RTAI environment. This application has been extensively used by the SUPSI and by the DIAPM together with Matlab and Simulink, in order to control electromechanical systems with different control complexity, from PID controller to state feedback controller and fuzzy systems.

In order to obtain a full open system, the implementation with Scilab/Scicos allows to obtain a full open source system for Rapid Controller Prototyping. Even if there is a great margin of improvement, this system has shown that it can be successfully used for complex controllers too.

In future, only the development under (new)lxrt in user space will be officially supported. At the author's homepage ([6]) the needed files for the Matlab/Simulink implementation in kernel space are still provided for the last RTAI release 2.24.12.

References

[1] www.mathworks.com

[2] www.scilab.org

[3] www.lme.die.supsi.ch/08S1-DAVID.jpg

[4] www.lme.die.supsi.ch/08R1-HOVER.jpg

[5] www.lme.die.supsi.ch/08S1-TICTR.jpg

[6] www.die.supsi.ch/˜bucher

[7] G. Quaranta, P. Mantegazza, 2001, *Using MATLAB-Simulink RTW to Build Real Time Control Applications in User Space with RTAI-LXRT*, Realtime Linux Workshop, Milano, Italy

[8] www.comedi.org

[9] Roberto Bucher, 2003, *Rapid Controller Prototyping with Matlab/Simulink and Linux*, ACE 2003, Oulu, Finland

[10] L. Dozio, P. Mantegazza, 2003, *Linux Real Time Application Interface (RTAI) in low cost high performance motion control*, Motion Control 2003, ANIPLA, Milano, Italy

[11] L. Dozio, P. Mantegazza, 2003, *Real Time Distributed Control Systems Using RTAI*, Sixth IEEE International Symposium on Object-Oriented Real-Time Distributed Computing, Hokkaido, Japan