Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

CAN-bus based rapid control prototyping system for education laboratories

Roberto Bucher * **Silvano Balemi** *

* *University of Applied Sciences of Southern Switzerland (SUPSI), Department of innovative technologies, CH-6928 Lugano-Manno,* {*roberto.bucher,silvano.balemi*}*@supsi.ch*

Abstract: The central element of the educational activities in mechatronics at SUPSI is the mechatronics laboratory, where students integrate the topics introduced by various courses.
In order to increase the value of the laboratory to students and to reduce the cost of its creation and operation, a modular approach has been implemented, which relies on sensors and actuators communicating through the CAN bus as a standard communication protocol.
A library of interfaces for many CAN capable devices has been implemented both for Matlab/Simulink and for Scilab/Scicos. Rapid control prototyping techniques can be used to generate code for existing applications or to build new applications based on available devices. Additional devices can also be easily added thanks to a 4-tier design of the interfaces.

Keywords: Rapid Controller Prototyping, Computer Aided Control System Design, Education, CAN bus.

1. INTRODUCTION

The senior bachelor students in mechanical engineering, electrical engineering and computer science at the University of Applied Sciences of Southern Switzerland (SUPSI) can opt for a minor in mechatronics.

As the bachelor curricula at SUPSI put particular emphasis on practical aspects of engineering, laboratories play a key role: they allow students to integrate concepts and theoretical knowledge acquired during various courses. Therefore, when the minor in mechatronics had to be prepared, the laboratory of mechatronics was the element needing more attention.

The following objectives for the laboratory were stated: the value of the laboratory to students had to be maximized and the cost of its creation and operation minimized while avoiding typical problems of mechatronics laboratories encountered at many universities. In fact, mechatronics laboratories often rely on hardware and proprietary tools from different vendors of didactic systems or they consist of hardware and software solutions built in house. In the former case, the laboratory is inflexible and its creation is only possible with important investments; in the latter case its creation requires considerable work effort. In both cases, the maintenance of the laboratory is very expensive.

Thus, a concept for the mechatronics laboratory at SUPSI was defined with the aim to

- offer a standard teaching environment based on tools that students will find again on the job
- rely on standard components from industry in order to avoid specific developments and to reduce the dependence from single source vendors
- simplify the maintenance and reduce its cost
- minimize the time and effort to expand the infrastructure
- make the didactic activities more flexible and valuable to students

The concept includes following key elements (see Figure 1)

- a general purpose computer used both as development platform and as control hardware
- common CACSD environments
- plants built on commercial CAN capable actuators and sensors (preferably using the CANopen protocol)
- physical CAN bus interfaces between the computer and the CAN capable devices

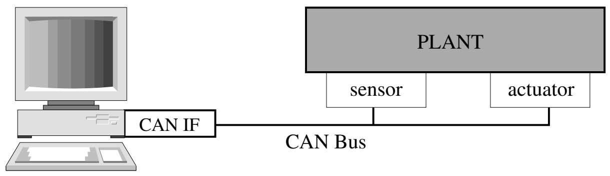

Fig. 1. PC with CAN interface and and plant with sensors and actuators communicating through the CAN bus

In the following sections, these elements as well as details of the hardware and software implementation are explained and different examples are presented.

2. THE CAN BUS

The CAN bus is a serial communication bus originally developed for automotive application but now widely used also in industry.

Many vendors offers sensors and actuators with CAN bus interfaces, in particular in the motion control area. Protocols are standardized (see for instance the CANopen or DeviceNet protocols). Own developments of CAN bus devices can rely on numerous development tools (e.g. monitoring programs) and on the presence of many microprocessors and DSP with CAN bus interfaces.

The CAN bus was selected because it offers great flexibility, limited cost, and a large choice of manufacturers. It represents an excellent choice for an educational laboratory and in particular for our particular project because different providers deliver their CAN bus interfaces for PCs with open source drivers.

3. THE CONTROL SYSTEM

3.1 Controller Hardware

The laboratory is composed of multiple places consisting of standard PC (Pentium IV, 2.8 GHz) with a physical CAN bus interface. Different physical CAN bus interfaces can be used: the standard one is the PCAN dongle from Peak Systems GmbH, connected to the parallel port of the PC. Alternatives are a self-built dongle also connected to the parallel port or miscellaneous PCI-CAN and USB-CAN interfaces.

3.2 The operating system: Linux RTAI

The PCs are running under Linux RTAI, a real-time version of the known Linux operating system (OS). 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 for other hard real-time OS's 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 either in the kernel or in the user area. The low latencies guaranteed by Linux RTAI make it possible to implement controllers for systems with closed-loop bandwidths up to a few hundred Hertz.

One of the authors is member of the development team of Linux RTAI.

3.3 The CACSD environments

Identification, control design and code generation exploit common Computer Aided Control System Design (CACSD) environments, in particular the commercial Matlab/Simulink/RTW and the open source Scilab/Scicos/RTAICodegen suites.

The main advantage of Matlab/Simulink ([Mathworks]) is that it is used at most universities and numerous industrial companies. On the other side, Scilab/Scicos ([Scilab]) is increasingly used at many universities and research groups, in particular in developing countries but also in Italy, France and in Germany. Scilab/Scicos can be freely downloaded from the web: it is a very powerful tool and a valid alternative to Matlab/Simulink in various application fields.

3.4 RTAI-Lab

RTAI-Lab is an open source project integrating the code generated by a CACSD environment into a Linux RTAI hard real-time task. At present, four CACSD suites are supported: the commercial MATLAB/Simulink/RealTime-Workshop (RTW) suite, the commercial EicasLab suite, the commercial MatrixX suite and the open source SCILAB/Scicos suite. The system has been widely described in [6], [3], [8], [5] and [7]. The document [9] describes in detail how to install the full RTAI-Lab tool chain.

RTAI-Lab provides also a GUI application for remote monitoring and control of hard real-time generated tasks.

Thanks to RTAI-Lab, the students can generate control code either from MATLAB/Simulink or from SCILAB/Scicos, start the real-time control task, monitor the controlled plant and store measurement data on the same PC used for identification and control design [4].

4. THE SOFTWARE CONCEPT

The definition of the concept for the mechatronics laboratory also implies the highest flexibilty in the use of components from various sources. In particular, it should be possible to connect CAN devices (e.g. sensors and actuators) to the PC using different physical CAN bus interfaces. Also, the generation of control code should be possible either from Matlab/Simulink or Scilab/Scicos.

The introduction of specific software interfaces for all the various combinations of CACSD environment, physical interfaces and CAN devices had to be avoided because the effort required for the development and for the following maintenance and future adaptations would be unmanageable. The solution to this problem was found with a 4-tier implementation of the software interfaces satisfying the following objectives:

- The introduction of a new CAN bus capable device should be possible with the implementation of a unique C-code usable for all combinations of physical CAN bus interfaces and CACSD tools.
- The introduction of a new physical CAN bus interface should require the development of a unique C-code available to all current and future software interfaces.

The solution is presented in figure 2 and explained more in detail in the following.

4.1 The CACSD specific layer

This layer contains the code needed to interface the CAN devices to the CACSD environment. This code is implemented as a masked C-MEX S-Function under Matlab/Simulink. The implementation under Scilab/Scicos requires an interface function, programmed as a ".sci" function, and an implementation file, programmed in C.

This layer implements the following functions:

- Drawing of the block in the CACSD graphical environment.
- Definition of the block parameters through a dialog box.
- Call of the specific procedures for initialization, input/output and termination of the next layer.

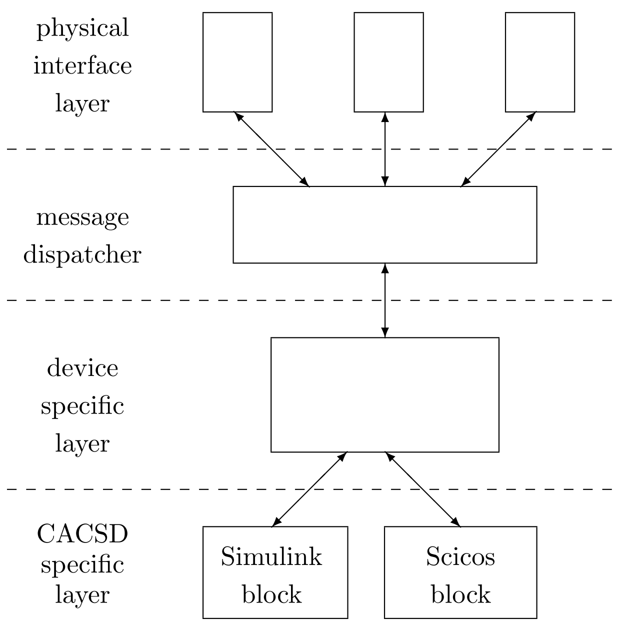

Fig. 2. The 4-tier architecture

Through the dialog box the CAN bus parameters can be defined, namely the physical interface, the CAN address of the device considered and the Baud rate.

4.2 The device specific layer

This layer implements the mechanism to communicate with a specific CAN device, like a motor, an encoder, etc. In this layer, the various CAN frames for the specific device are assembled based on the CAN address, on the block parameters and on the variables read from the CACSD block input. For example this means sending a current to a motor at a known CAN bus address.

Then the frames are sent to and received from the CAN bus with standard commands defined in the message dispatcher.

4.3 The message dispatcher

The commands made available to the device specific layer by the message dispatcher are:

- `SUPSICAN_init`: an initialization function, which initializes the CAN hardware and sends the frames necessary to set up the CAN device.
- `SUPSICAN_write`: a function for writing to the CAN device.
- `SUPSICAN_read`: a function for reading from the CAN device.
- `SUPSICAN_end`: a termination function.

The dispatcher then exchanges messages with the physical CAN interface layer thanks to the "CAN_dev" structure below.

```
struct CANdev{ int fd;
               unsigned short port;
               int (*write)(struct CANdev  dev,
                            DWORD ID, BYTE DATA[],
                            int len);
               int (*read)(struct CANdev dev,
                           DWORD ID, BYTE DATA[],
                           int len);
               int (*close)(struct CANdev dev);
};
```

This structure defines the appropriate read and write functions for the selected physical interface.

Finally, the dispatcher hands over the data from the physical layer interface to the device specific layer and vice versa.

4.4 The physical interface layer

This layer contains the drivers for the different physical interfaces used to connect the PC running the real-time application with the CAN bus.

The drivers can be implemented using different approaches:

(1) a module which calls a Linux driver.
(2) a module which calls a RTDM driver.
(3) a module which implements all I/O commands (`inb`, `inw`, `outb`, `outw` etc.) to directly access the CAN registers.

The first method is immediate because it exploits standard Linux drivers provided with the CAN interface, but cannot be used if hard real-time determinism is required. Call of standard Linux drivers is handled using system calls, which is performed under Linux RTAI in soft real-time: this causes a repeated switching from hard real-time to soft real-time.

RTDM (Real Time Driver Model) drivers can be used under different real-time Linux variants. Hardware can be reached without passing through system calls. Some physical CAN interfaces on the market are already delivered with a RTDM driver.

The last method is quite simple but it requires detailed knowledge about the implemented hardware.

4.5 Advantages of the proposed solution

The proposed solution has the following advantages:

- If a new CAN device must be added to the CACSD environment only the CACSD interface (CACSD specific layer) and a file for the device specific layer must be written. Modifications of the other layers are not necessary.
- If a new physical CAN interface must be added, the corresponding driver on the corresponding layer must be implemented. Then, only minimal modifications are required in the next layer (message dispatcher)
  - · The code needed to fill the CAN_dev struct with the init, I/O and termination functions must be added.
  - · The CACSD interface code must be modified by adding the new interface to the pop-up menu of the masked S-function under Simulink. No modifications are required under Scicos.

5. THE IMPLEMENTED INTERFACES

Interfaces for several devices under both Matlab/Simulink and Scilab/Scicos have been implemented. All interfaces can be used in combination either with the PCAN dongle from Peak Systems GmbH or with the self-built dongle both connected to the parallel port.

Figure 3 shows the library with the CAN device blocks created for the Matlab/Simulink environment. The same library implemented for Scilab/Scicos is shown in figure 4

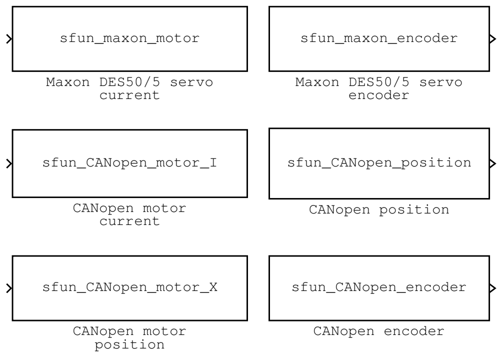

Fig. 3. Simulink library to access the CAN devices

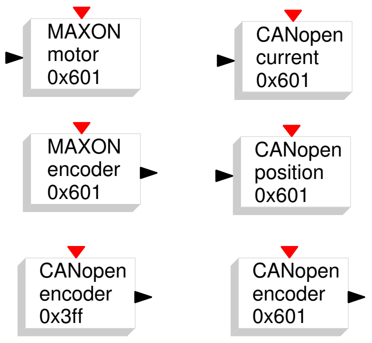

Fig. 4. Scicos library to access the CAN devices

6. LABORATORY PLANTS

Miscellaneous plants are available at the SUPSI mechatronics laboratory. They are based on various CAN bus capable commercial devices like encoders, motor drivers, PLCs, joysticks but also on custom-built devices where necessary.

Some examples of plants are the classical inverted pendulum, a crane, disks and spring systems, a mass and spring system, a SCARA robot and a force-feedback manipulator.

6.1 The disks-and-spring process

This is the one of the first experiments proposed to the students. Two motorized disks are connected together by a flexible joint (Figure 5).

The position of the right disk must be controlled by actuating the motor on the left. Sensors collect the positions of both motors. The right motor can be used to introduce a disturbance signal to the right disk.

First, students must perform a non-parametric and a parametric identification of the plant, and then implement

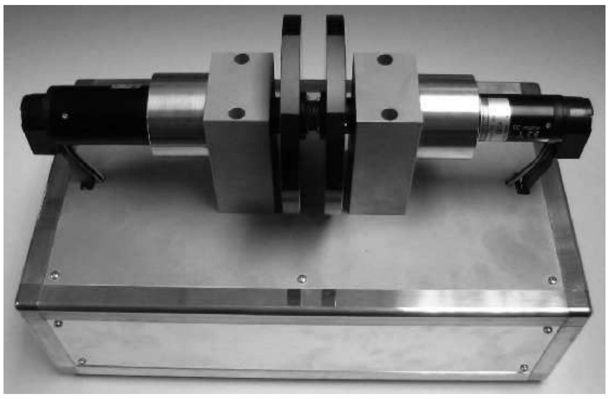

Fig. 5. The disks-and-spring plant

a state-feedback controller. Figure 6 presents the block used to interface the hardware with the code generated by Matlab/Simulink.

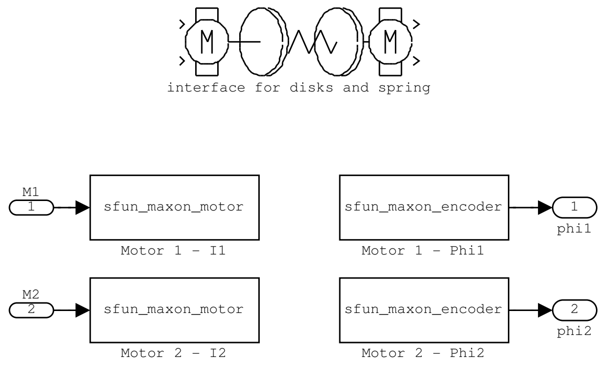

Fig. 6. Simulink block (top) and its content (bottom)

Figure 7 shows Bode diagrams from the non-parametric and from the parametric identification of the disks-and-spring plant.

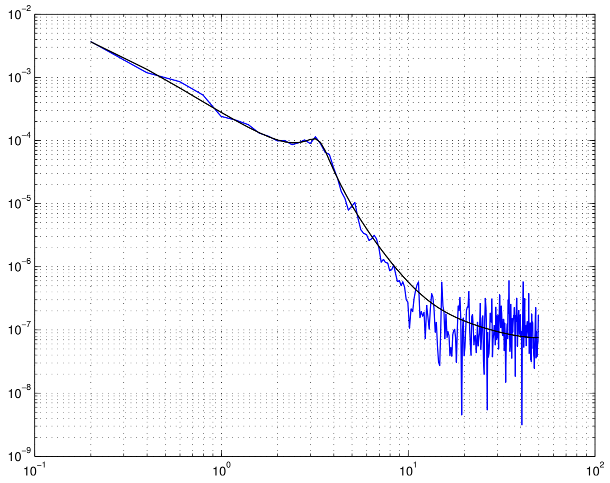

Fig. 7. Non-parametric and parametric identification of the disks-and-spring plant

The controller has been designed for the following 4 cases:

- State-feedback controller using pole placement with full-state and with reduced-order observer.
- State-feedback controller with a LQR design, with a full-state and with reduced-order observer.

Figure 8 shows the controller implemented in the Matlab/Simulink environment.

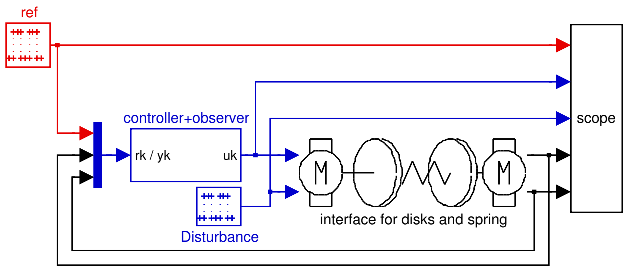

Fig. 8. Simulink file for the controlled plant

6.2 *The inverted pendulum*

The classical inverted pendulum at SUPSI also exploits the communication through the CAN bus (see Figure 9).

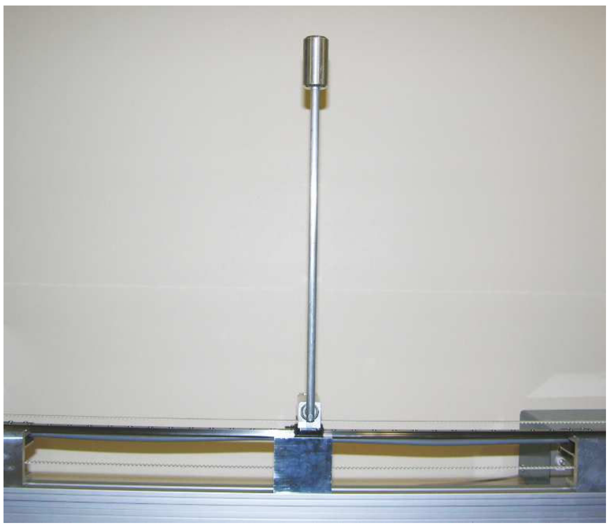

Fig. 9. The inverted pendulum

The actuator is given by a DC motor from Maxon (RE 40 with Epos driver 70/10). The same motor driver delivers the motor angle (i.e. the chart position) obtained from the incremental encoder of the motor. The angle of the pole is measured using a wireless incremental encoder, which sends the impulses to a wireless-CAN bridge. Figure 10 shows the block used to interface the hardware with the code generated by Scilab/Scicos.

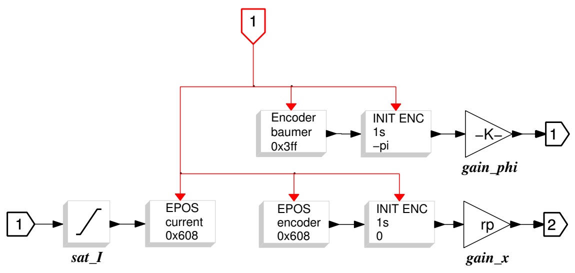

Fig. 10. Inverted pendulum - Block used for real-time controller

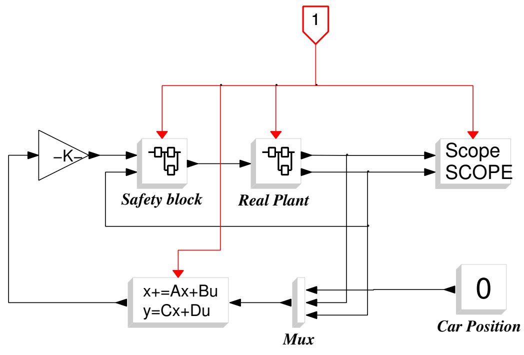

Fig. 11. Inverted pendulum - Controller

The LQ controller is implemented in the Linux RTAI PC using the Scicos environment (see Figure 11).

6.3 *The Delta Haptic device*

A haptic delta manipulator has been built for the SUPSI laboratory. Figure 12 shows this device. Each of the

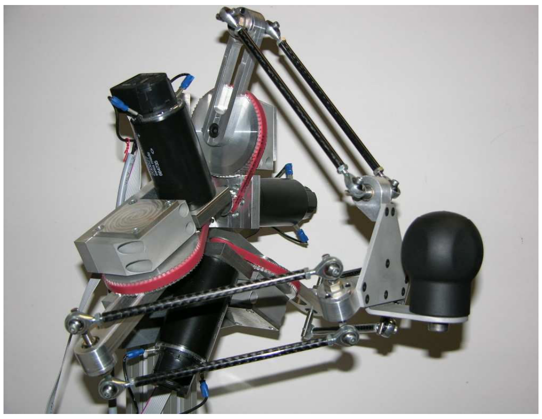

Fig. 12. The delta haptic device

three motors was connected to a CANopen capable driver (EPOS 24/5 from Maxon) for which the drivers were already available. Two Simulink files have been created. The first file generates the control code for the delta haptic device (Figure 13), the second one the visualization program which exploits the virtual reality toolbox of Matlab/Simulink.

The communication between the hard-real-time control process and the visualization program is guaranteed by a FIFO block (see Figure 14).

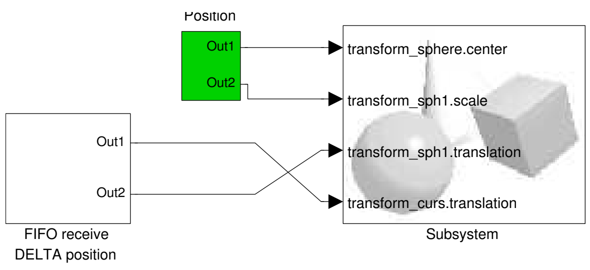

Fig. 14. Simulink diagram for virtual reality representation

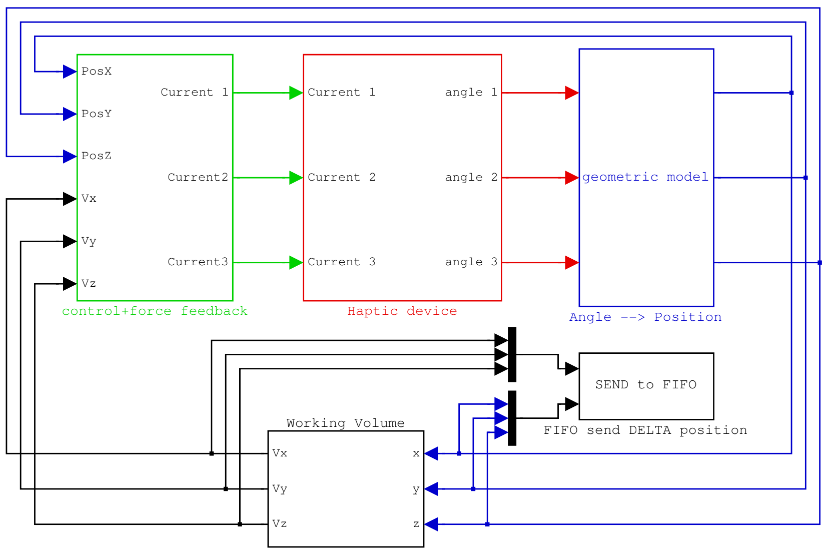

Fig. 13. Controller of the delta haptic device

7. BENEFIT FOR EDUCATION

The benefits for the students are in particular related with the possibility to conceive and realize new plants by using already available hardware. It suffices to assemble known CAN bus capable devices with the help with the necessary mechanical parts.

Moroever, new CAN devices can be integrated in the given architecture with minimal effort. The development of code for a new device can be immediately used in combination with all possible CACSD environments and physical CAN interfaces.

Students can learn how to conceive, build, characterize and control mechatronic systems using standard tools. There is no need for the students to learn didactic environments which they will never see again in their life.

8. INDUSTRIAL APPLICATIONS

The usefulness of the concept above is not restricted to didactical activities alone. In an industry related project the same concept was applied to the development of a test and characterization system based on a CompactPCI platform running the Linux RTAI operating system.

The use of CAN bus devices from the mechatronics laboratory only required the development of the specific C-code for the new physical CAN bus interface (a CompactPCI board from HICOCAN).

The complete system with the CAN devices and with other CompactPCI boards interfacing the plant was up and running within a couple of days.

9. CONCLUSIONS

The paper presented a concept for the infrastructure of mechatronics laboratories thought to increase the value for students while minimizing the investment and operation costs. It is based on a rapid control prototyping system using the CAN bus to connect various devices composing the plant to be controlled.

The concept has shown to be valid, with the laboratory being well appreciated by the students. The low maintenance cost and the high flexibility have been proven both during educational activities as well as in the context of research activities.

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. Interfacing Linux RTAI with Scilab/Scicos. In *Real Time Linux Workshop*, Singapore, 2004.

[4] R. Bucher and S. Balemi. Rapid controller prototyping with Matlab/Simulink and Linux. *Control Engineering Practice*, 12(2):185–192, 2006.

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

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

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

[8] Roberto Bucher, Lorenzo Dozio, and Paolo Mantegazza. Rapid Control Prototyping with Scilab/Scicos and Linux RTAI. In *International Scilab Conference*, Paris, 2004.

[9] Roberto Bucher, Simone Mannori, and Thomas Netter. RTAI-Lab tutorial: Scilab, Comedi and real-time control, 2006. URL `www.rtai.org/RTAILAB/RTAI-Lab-tutorial.pdf`.

[Mathworks] Mathworks. The MathWorks. URL `http://www.mathworks.com`.

[Scilab] Scilab. A Free Scientific Software Package. URL `http://www.scilab.org`.