WEB-BASED VIRTUAL LAB AND REMOTE EXPERIMENTATION USING EASY JAVA SIMULATIONS

R. Pastor, J. Sánchez, S. Dormido*

** Dpto de Informática y Automática, UNED, c\ Juan del Rosal 16, 28040 Madrid, Spain. Phone:+34913987151, Fax:+34913986697, E-mail:rpastor@dia.uned.es; jsanchez@dia.uned.es; sdormido@dia.uned.es*

Abstract: This paper presents an ongoing schema to develop virtual models of physical setup equipment and their integration into the corresponding remote laboratory. For this task, we use Easy Java Simulations (*Ejs*), a tool that allows us to make a complete simulation model based on the physical laws of the setup. *Ejs* is also able to produce virtual views that are necessary for model user interaction. These virtual views can be used with the simulation or real plant data. Accordingly, a communication model between *Ejs* and real equipment has to be defined in order to work with the same view in virtual mode (simulation) or remote mode (real setup). An example of a servomotor setup will be presented showing the virtual views and the way of working in both modes. *Copyright © 2005 IFAC*

Keywords: Remote and virtual laboratories, control education, web-based simulations, teleoperation, Java simulations.

1. INTRODUCTION

Progress in Internet technology has prompted many educational institutions to develop web-based laboratories using either models or physical resources, i.e., didactic setups. With the help of an online Internet laboratory for control experiments, educators can be encouraged to design control engineering courses that combine theoretical issues with practical activities. Via the Internet, web-based laboratories can offer more flexibility in preparing assignments for students who require experimentation with virtual or real phenomena. In addition, an Internet laboratory allows better use of equipment since students can access labs from anywhere and any time with just an Internet connection. This sharing of resources not only brings down the experiment cost per student, but equipment is also available to more students.

However, the development of useful web-based control labs does not just consist of programming a set of HTML pages and Java applets so that users can introduce many parameters to change the state of a model or physical system, observe the results, and so on. Students also need to immediately know the consequences of the decisions taken during the design process and the relationships among the different experiment state variables. Thus, students need to appreciate the gradient of change in the performance criteria given for the elements that they are manipulating. Obviously, to reach this goal it is necessary to take into account the approach known as "interactive design" (Dormido, 2004).

Interactive design provides a flexible and user-friendly method to define the experiments performed on the model or plant. During the interactive run, the user can change the value of the model inputs, parameters and state variables, perceiving instantly how these changes affect the system dynamics. In such a high interactive system, several graphics windows, reflecting the value of every active element and the constraints among them, are constantly updated. As a consequence, interactive design develops and enhances the understanding of the system behavior. This capability is especially useful when the model is being used for educational purposes. Examples of this new educational philosophy for teaching automatic control are

described in (Piguet and Gillet, 1999; Schmid, 1999; Sánchez *et al.*, 2002; Dormido and Esquembre, 2003; Pastor *et al.*, 2003; Dormido *et al.*, 2004).

Unfortunately, the construction of these interactive applications has two main drawbacks: programming skills and developing time. Indeed, a control engineering teacher knows how to use different modeling and simulation environments and how to create environments and interfaces to conduct experiments over a physical resource in the traditional fashion. Yet most of these tools are typically not oriented to the development of web-based labs using an interactive approach.

Whether a virtual or real system is selected, extensive use of interactivity with these tools would require non-trivial effort. Therefore, it seems clear that the academic control community demands a tool to develop web-based interactive labs with low programming effort, interactivity as the main feature, and minimum economic cost.

The paper is organized as follows. First, a software program called *Ejs* is introduced as an appropriate tool for the creation of interactive control applications. Next, we describe the use of *Ejs* to develop an example of interactive application in the field of control engineering: a virtual and remote lab to conduct experiments with a DC servomotor.

2. *EJS* FUNDAMENTALS

Easy Java Simulations, *Ejs* for short, is an open-source tool developed in Java, originally designed to help non-programmers to create scientific simulations (Esquembre, 2004). The user needs to provide (at a reasonably high level) the analytical model, design the graphical view, and decide about the interactivity offered. *Ejs* will then automatically generate the Java source, compile it into Java classes, pack the classes in a Jar file, and produce several HTML pages with the author-provided narrative and the ready-to-run applet for the simulation.

The tool architecture derives from the model-view-controller (MVC) paradigm. The philosophy of this paradigm is that interactive simulations must consist of three different parts:

- the representation of the phenomena in terms of variables and relationships among these variables (the model),
- the graphical representation of the states of the phenomena (the view), and
- the specifications of the user's actions to perform on the simulation (the controller).

These three parts are deeply interconnected. The model obviously affects the view, since a change in the state of the model must be made graphically evident to the user. The control affects the model because control actions can (and usually do) modify the value of the model variables. Finally, the view affects the model and the control, because the graphical interface can contain components that allow the user to modify variables or perform the predefined actions. In fact, going a step further in the process of simplifying the construction of a simulation, *Ejs* suppresses the control part, merging it half into the view, half into the model.

The model conveys the scientific part of the simulation and is thus the responsibility and the main interest of the target user. Control teachers usually describe their algorithms in terms of mathematical equations and translate these into algorithms of a computer language, or use high-level tools such as Matlab/Simulink. The main target when creating a simulation model is to concentrate on the analytical description of the phenomena, the content, and the accuracy of the simulation. However, the creation of the necessary graphical user interface (i.e., the view) is the part of the simulation that demands more knowledge of advanced programming techniques. To make the situation even worse, the addition of interactivity to this interface involves mastering sophisticated software techniques (such as event-handling or multitasking). All this requires a huge investment in time and effort.

Ejs helps its users with both tasks. While retaining the flexibility of a general programming language so that it is possible to specify almost any type of algorithm, the tool provides extensive scaffolding to define the model. This is pedagogically important, since the process of learning good control fundamentals consists, to a great extent, of knowing the basic principles for building models.

Nevertheless, the use of the MVC paradigm allows us to transform a virtual lab, i.e., a simulation, into a remote lab in just a few steps. This strict separation between view+control and model lets us replace the simulation of the phenomena with the real version without modifying most of the interactive application. In a virtual lab, model and view+control exchange information by local state variables, so the transformation in a remote lab just consists of reading the state variables remotely in order to access the real system. Since *Ejs* lets developers insert pure Java code into the view+control, the communication with the didactic setup can be done by TCP/UDP sockets or RMI.

3. DEVELOPING VIRTUAL LABS WITH *EJS*

Accordingly, *Ejs* simulations are created by specifying a model for the simulated system and by building a view that continuously visualizes the state of this model and readily responds to user interactions. In order to describe a simulation, an author needs to be able:

1. To define the model in three steps: (a) identify the set of variables that describe the system, (b) initialize these variables, and (c) describe the mathematical equations.
2. To construct a view that offers a schematic or realistic visualization of the phenomena.
3. To establish the appropriate model user interaction by establishing links between the graphical interface of the simulation and the response of the model to user-driven changes to it.
4. Optionally, to include textual information about the simulation (narrative).

Ejs offers two ways of specifying the dynamics of the model. The first one is a built-in editor and solver for systems of ordinary differential equations (ODEs). Users write the equations very much like they would on a blackboard, and the system automatically generates the code that numerically solves the system using one of several of the provided standard algorithms (Figure 1). The second facility is a connection to Matlab/Simulink that lets users design and solve their models with the help of these tools (Figure 2). In this last situation, the model is fully defined by a Simulink block diagram and a text file is necessary for *Ejs* to know the names of the Matlab/Simulink variables. *Ejs* can thus carry out the bidirectional linkage between view (Java-coded interactive interface) and the model (Simulink diagram) once the final application has been released (Sánchez *et al.*, 2004).

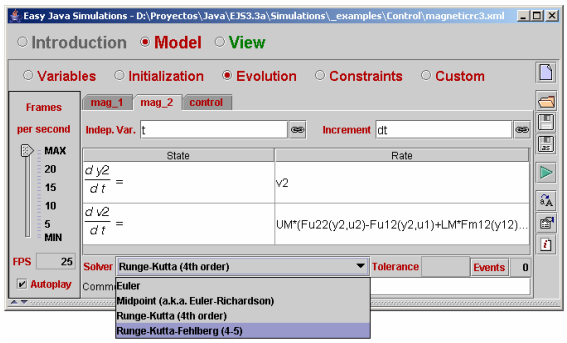

Fig. 1. Edition panel to write the ODEs and select the solver.

This feature is obviously useful for Matlab users who can then access and run their Matlab code from an *Ejs* application. It is however also useful for those who have already developed a model using Simulink block diagrams. *Ejs* can then be used as a tool to create the dynamic interactive user interface that is needed to turn this simulation into an effective virtual educational laboratory.

To create the view, *Ejs* provides a set of advanced graphical elements (Figure 3) that build on top of both standard Java Swing components (containers, buttons, text fields, sliders, combo boxes,…) and on specific scientific two- and three-dimensional visualization classes from the Open Source Physics project (Christian, 2003) (particles, vectors, images, vector and scalar fields,…). These elements are used in a simple drag-and-drop way to build the interface. The user needs to design the view so that it will offer a visualization of the phenomenon appropriate to the desired pedagogical goals. In particular, the view should encourage students to explore the phenomenon under different engineering perspectives in order to gain better insights of the system.

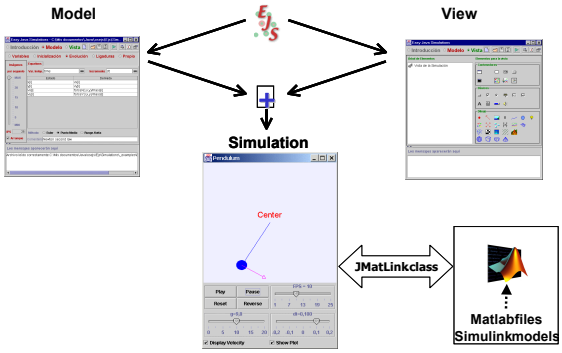

Fig. 2. Building a simulation with *Ejs* and Simulink.

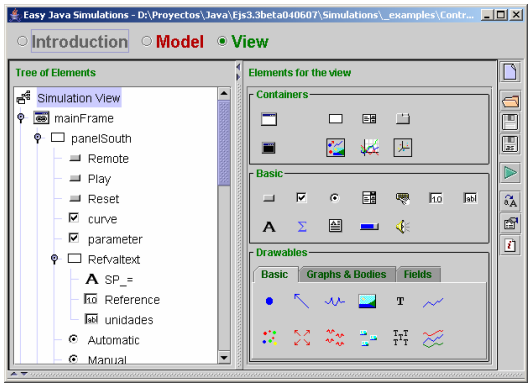

Fig. 3. Library of graphical elements of *Ejs*.

To complete the view, the different view elements have to be instructed to display on the screen according to the model variable values. Every graphical element of *Ejs* has internal values, called properties, which can be customized to make the element look and behave in a certain way (change its displayed value, position, size, etc.) The user can then connect the properties of the graphical elements of the view to the value of the different model variables. Some of these properties can also be instructed to trigger user-defined actions (typically routines defined in the model) when the user interactively changes them. This procedure, which we call *linking* the model and the view, is what turns the simulation into a real dynamic, interactive application. This mechanism constitutes a simple, though very effective way of building advanced interactive user interfaces.

The reason for this is that linking is a two-way connection. When one or more variables of the model change, this change is passed on to the view so that it immediately displays the new state of the model. Also, because the elements provided have built-in

interactive capabilities, any student interaction with the interface immediately affects the model variables that are connected to it. For example, let us imagine an *Ejs* simulation of the control of a magnetic levitator. The magnets are represented in the view by means of rectangles with two colors to represent the two polarities. If the dimension of a magnet is modified by dragging and stretching the corners of the rectangle with the mouse, the variable representing the magnet mass in the analytical model will reflect this change (thus affecting the system dynamics).

With all this high-level information, which only the user can provide, Ejs takes care of all the low-level procedures needed to create the final simulation. It will generate the Java code that handles all the internal tasks, compile it into Java classes, pack the classes in a compressed file, and end up with a ready-to-use simulation. Simulations created with just Ejs can be used as stand-alone applications under different operating systems (for instance, a .BAT file is provided when running under Windows), or be distributed via the Internet and run as applets within HTML pages (which are also generated by Ejs) by using any Java-enabled web browser. The tool also includes a simple HTML editor to help the teacher enhance the generated web pages with pedagogical information and/or instructions for the simulation.

4. REMOTE EXPERIMENTATION INTEGRATION

Once the *Ejs* virtual laboratory is available, our aim is to use it to control the equipment located in the real laboratory, i.e., a didactic setup. Thus a client/server framework (Figure 4) is necessary in order to get/set the data to/from the didactic setup. On the client side, the same view of the *Ejs* virtual lab is used to provide the user with the interaction needed to teleoperate the system and recover the stream of real data. On the server side a Java class, named `Ejs_Live_Data_Server`, has been designed to implement the communication model between *Ejs* and the remote laboratory. It is the developer's responsibility to integrate this class into his server side development in order to exchange the necessary data with the *Ejs* client side, i.e., the interface.

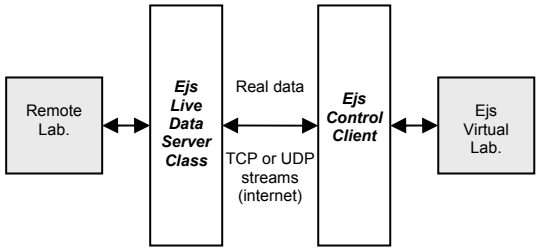

Fig. 4. Communication model between the remote laboratory and the *Ejs* virtual lab.

If this architecture is chosen, we need to make one change in the *Ejs* virtual laboratory by adding a global variable named `remote_mode` of type boolean. Moreover, a button with the name `Remote` must be included in the main window of the view generated by *Ejs*. The button will be used for switching to remote or simulation mode by changing the value of the `remote_mode` variable and hence pause/resume the *Ejs* simulation.
Once this change is introduced, we need to indicate which *Ejs* parameters are required to use the *Ejs* Control Client on the client side. This is done by creating a simple text file that informs the *Ejs* Control Client about the configuration of the communication model and the variables that will be accessible in the remote laboratory and *Ejs*. Figure 4 shows an excerpt of a configuration file designed for this example.

```
[Ejs]
model=motor12
location=D:\Ejs\simulations\motor12.jar
[live data server]
ip=plantas1.dia.uned.es
protocol=TCP
port=10000
sampleTime=100
[variables to remote]
var=automaticControl,boolean,1
var=posControl,boolean,1
var=Kp,double,1
[variables from remote]
var=u,double,20
var=th,double,20
var=Reference,double,20
[gui]
windows=mainFrame,dialog
component=Automatic,mainFrame,radiobutton,automatic
Control
component=Manual,mainFrame,radiobutton,automaticCon
trol
component=Reference,mainFrame,textfield,Refdeg
component=Position,mainFrame,radiobutton,poscontrol
component=uvalor,mainFrame,textfield,u
```

Fig. 5. Text file to configure the connection between the remote setup and the *Ejs*.

As Figure 5 illustrates, the configuration file has several sections:

- The `[Ejs]` section defines the *Ejs* model parameters that the *Ejs* Control Client needs to use the *Ejs* virtual lab.
- The `[live data server]` shows the connection parameters for the communication model that uses TCP or UDP streams to exchange data between *Ejs* and the remote lab.
- The `[variables to remote]` and `[variables from remote]` are the specifications of the data packets to be sent and received to/from the remote setup. Each line of these two sections indicates the name of the variable, which must be previously defined in the *Ejs* model, the type, and the data length.
- The `[gui]` section is useful to give information about the *Ejs* visual components that interact with the user, respond to the events, and send the corresponding variables to the remote laboratory.

The windows of the *Ejs* view, all the windows' components, the type of visual component and the *Ejs* model variables must be specified.

The last step is the connection of the previously developed virtual lab to a remote servomotor. To achieve this, the `Ejs_Live_Data_Server` class has to be integrated into the server side of our laboratory development. The next section presents an example of this task.

5. A CASE STUDY: THE SERVO MOTOR

The remote laboratory developed in this case study corresponds to the well-known model of an electrical drive. In this electromechanical process the main objectives consist of controlling either the angular position or velocity of a load.

5.1 Ejs virtual laboratory.

Up to now, the virtual view developed by *Ejs* has given the user basic interactivity with a simulated motor. The main window (Figure 6) provides a virtual representation of the motor setup, i.e., displaying the plant state while the experiment is being conducted. Furthermore, the view lets user tune PID parameters, switch to manual/automatic control mode and select the control of position or speed. The views in Figures 8 and 9 show the time evolution of speed, position, and control variables in many signal scopes.

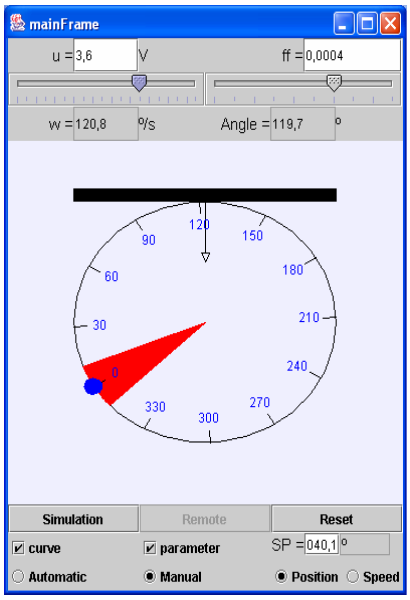

Fig. 6. Virtual representation of the motor.

5.2 Remote laboratory setup.

For the real laboratory (Figure 7), the real-time services were developed using the RT Series System of Labview (National Instruments, 2003). This software allows us to design robust real-time applications embedded in several real-time targets. In this instance, we have used a NI PCI-7030/6040E Real-Time Multifunction I/O Board, which has a 486/133 MHz processor dedicated to real-time operations. The development cycle is quite simple: A vi file with the real-time operations (DAQ operations and math operations to compute the control actions based on PID algorithm) is implemented. This code (vi file) is downloaded into the Real-Time target and is executed. After, a Labview application (another vi file) is developed to communicate with the Real-Time target and to provide TCP/IP communication streams.

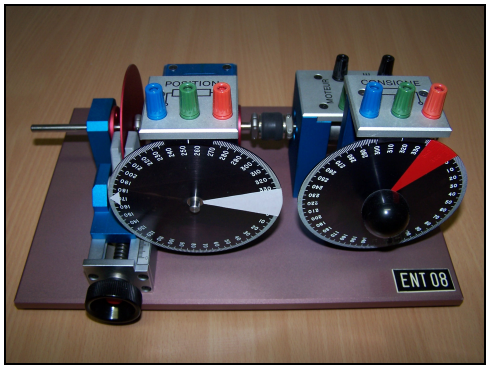

Fig. 7. Electrical drive equipment.

5.3 Integrating the `Ejs_Live_Data_Server` class.

By means of the TCP/IP communication streams defined in the Labview application it is possible to specify a communication protocol between Labview and a Java application. This software makes use of the `Ejs_Live_Data_Server` class to push the motor's real data towards the *Ejs* Control Client across the Internet. This application also receives data from the *Ejs* Control Client, like the PID parameters or set points, and resends them to the Labview application, i.e., to the motor. Of course, this application runs on the same computer as the Labview software in order to avoid further delay in the data transmission. Thus, the delay value is in the order of milliseconds, which is negligible compared with the *Ejs* communication delay.

5.4 Remote and virtual experiments.

Several didactic experiments can be performed using this remote/virtual setup. For example, in Figures 8 and 9 two very common experiments are shown: control of speed and control of position. While experiments are running, the real state of the motor is shown in the *Ejs* view if the remote mode is on. In the other case, the virtual view developed by *Ejs* means that it is possible to practice with the virtual motor, preparing the user to work with the real equipment. Whether working in real or simulation mode, by using the `Simulation` button (Figure 6) it is possible to change the PID values interactively or to switch to simulation mode.

Generally, a user cannot distinguish whether the real plant or the simulation model is being used. For this

reason, the combination of virtual and remote laboratories in just one virtual view must be completed with new visual elements, like video and audio channels, in order not to confuse users.

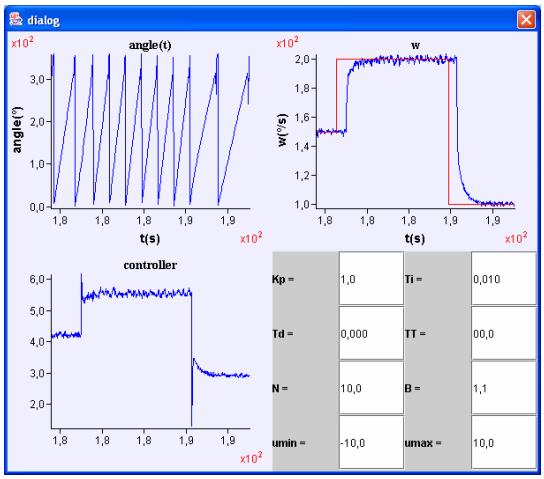

Fig. 8. Signals from the speed control experiment.

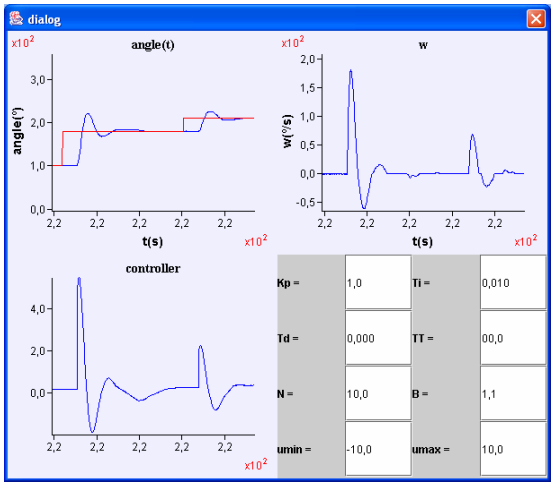

Fig. 9. Signals from the position control experiment.

6. CONCLUSIONS

The way in which the hands-on teaching and learning of control engineering has been traditionally done in higher education is experiencing a rapid transition. This is due to the combination of two key elements in the development of new control laboratories, the Internet and interactivity, producing the so-called web-based laboratories. The purpose of these new labs is three-fold: 1) to access labs anywhere and any time, 2) to share the resources and experiments among different groups, and 3) to open new visualization possibilities that improve learning and facilitate a more active role of students in the learning process.

To facilitate the development of these innovative labs, it is clear that convenient tools must be provided to the control education community. Tools in which the use of interactivity and Internet technologies are managed for the developer, i.e., the instructor, in a natural way. In this paper, we have briefly described a tool, *Easy Java Simulations (Ejs)*, which can be considered as a starting reference to create staggering interactive computer learning experiences. *Ejs* is a software tool that helps create dynamic and interactive scientific simulations in Java language very easily. Thus, according to the model-view-control paradigm, *Ejs* facilitates the creation of the view and control, and the model can be either a simulation (virtual lab) or a didactic setup accessible across the Internet (remote lab).

To highlight some of the many features of *Ejs*, an example of how to build an interactive virtual and remote lab of a DC servomotor has been included in the paper. This remote and virtual lab and other case studies are available at *http://virtual-lab.dia.uned.es*. *Ejs* can be freely downloaded from the website: *http://fem.um.es/Ejs/*.

ACKNOWLEDGEMENTS

This work has been supported by the Spanish CICYT under grant DPI2004-01804.

REFERENCES

Christian, W. (2003). The Open Source Physics Project. http://www.opensourcephysics.org.

Dormido, S. (2004). Control Learning: Present and Future. *Annual Reviews in Control*, **28**, 1, 115-136.

Dormido, S. and F. Esquembre (2003). The Quadruple-Tank Process: An Interactive Tool for Control Education. *Proceedings of the European Control Conference*, Cambridge, UK.

Dormido, S., C. Martín, R. Pastor, J. Sánchez and F. Esquembre (2004). Magnetic Levitation System: A Virtual Lab in Easy Java Simulation. *American Control Conference ACC'04*, Boston (USA), July.

Esquembre, E. (2004). Easy Java Simulations: A software tool to create scientific simulations in Java. *Comp. Phys. Comm*. **156**, 2, 199-204.

National Instruments (2003). Labview Real-Time Module information on http://www.ni.com.

Pastor, R., J. Sánchez, and S. Dormido (2003). A XML-based framework for the development of web-based laboratories focused on control systems education. *International Journal of Engineering Education*, **19**, 3, 445-454.

Piguet, Y. and D. Gillet (1999). Java-based remote experimentation for control algorithms prototyping. *Proceedings of the American Control Conference*, San Diego, CA (EE.UU.), 1465-1469.

Sanchez, J., S. Dormido, F. Esquembre and R. Pastor (2004). Interactive learning of control concepts using Easy Java Simulations. *2nd IFAC Workshop on Internet Based Control Education* (IBCE'04), Grenoble (France).

Sánchez, J., F. Morilla, S. Dormido, J. Aranda and P. Ruipérez (2002). Virtual control lab using Java and Matlab: A qualitative approach. *IEEE Control Systems Magazine*, **22**, 2, 8-20.

Schmid, C. (1999). A Remote laboratory using virtual reality on the Web. *Simulation*, **71**, 1, 13-21.