

Easy Java Simulations: an Open-Source Tool to Develop Interactive Virtual Laboratories Using MATLAB/Simulink*

J. SÁNCHEZ, F. ESQUEMBRE, C. MARTÍN, S. DORMIDO, S. DORMIDO-CANTO, R. D. CANTO, R. PASTOR and A. URQUÍA

Department of Computer Sciences and Automatic Control, UNED, Cl. Juan del Rosal 16, 28040 Madrid, Spain. Email: jsanchez@dia.uned.es, fem@um.es

This paper describes the make-up of interactive computer simulations that implement virtual laboratories in the field of Control Engineering education. We introduce Easy Java Simulations (EJS), a Java-based tool designed to help control educators with a low profile in programming to create interactive scientific simulations. This tool can be used on its own, generating stand-alone Java applications or applets, or in conjunction with MATLAB/Simulink, using them as the internal engine that describes and solves the model. This tool allows users to develop complete, interactive simulations in three steps: writing the mathematical model (optionally using MATLAB/Simulink), building the graphical user interface (GUI) using off-the-shelf graphical elements, and linking the GUI elements to the variables of the model. In this way, models of control engineering can be specified either using the programming support provided by EJS or by using MATLAB/Simulink, which can then be fully controlled from EJS. In this paper we describe this particular feature in detail, and provide some examples that show the advantages that this tool offers to the world-wide engineering education community.

INTRODUCTION

RECENT DEVELOPMENTS in computer hardware and software now make it possible to provide students with interactive programs that can be considered as midway between regular labs and lectures and that allow us to display multiple-view representations of a given dynamic system, and some of its attributes, on the computer screen [1]. Visualization thus appears naturally in the origins of automatic control, in the discovery of new relationships among mathematical objects, and in the transmission and communication of knowledge. These tools, called *interactive virtual labs*, can be used to explain basic concepts, to provide new perspectives on and insights into a problem, and to illustrate analysis and design topics [2].

However, as a natural consequence of the enhanced performance and graphical capabilities of modern computers, the search for interactivity must also take us a step forward. Nowadays, the use of the word “interactivity” in the design of automatic control systems means, to many people, a process of repeating several times a loop whereby a user introduces some parameters in the software, runs the algorithms of design, and analyzes and compares the results graphically in static plots. If the results are not as expected, the user must enter new parameters and repeat the execution-analysis process until (s)he finds a configuration that is

satisfactory. Proceeding in this way, however, it is difficult for the user to recognize the relationship between the gradient of change in the results (usually, the system response in temporal and frequency domains) and the values of the input parameters. This perception would be considerably improved if the user could work with simultaneous graphical representations of the input parameters and the model’s dynamic properties, and if there were an instantaneous translation of any modification of an input parameter to the system’s response. This situation could then be considered true interactivity, since a change in the simulation’s parameters has an immediate effect on the different views of the system, according to the mathematical relationships that govern the dynamics of the system. Figs. 1 and 2 present two examples of introducing interactivity into virtual labs. Fig. 1 shows a virtual lab for testing different control strategies for a Furuta pendulum. The next figure, Fig. 2, corresponds to an interesting application for the study of the theoretical control background. In such applications, the signals inside a plot can be dragged by the mouse (thus behaving as input parameters) and the new values have an immediate consequence for the other graphical representations of the system.

Let us now apply our discussion on interactivity to the real educational world, where the *de facto* standard software for instructional purposes is MATLAB/Simulink. If we try to apply our concept of interactivity to teaching with

* Accepted 2 July 2005.

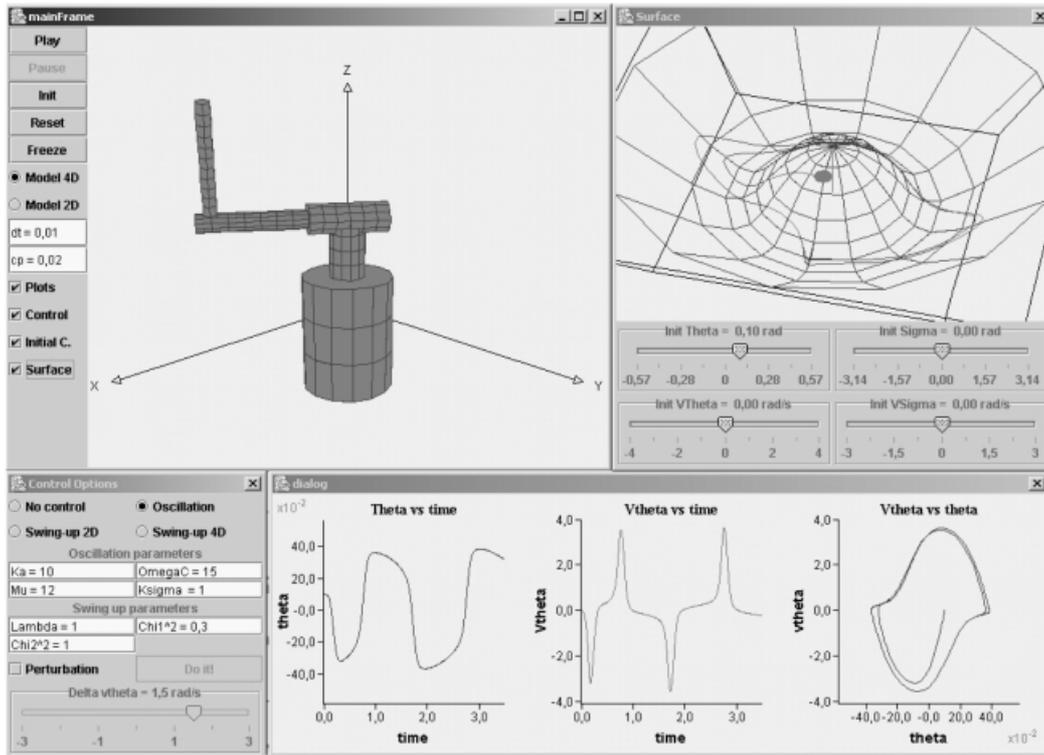


Fig. 1. Virtual lab for studying control problems related to the Furuta pendulum.

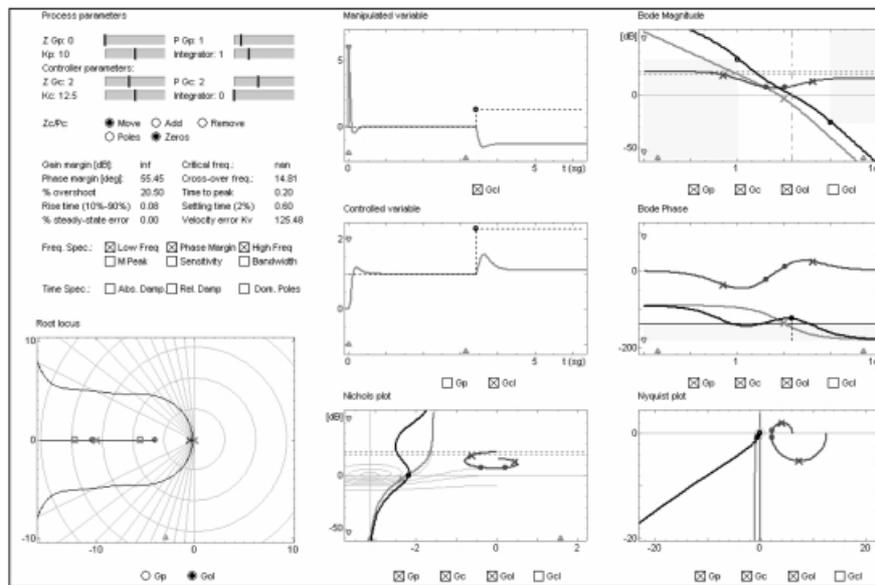


Fig. 2. Interactive toolbox for analyzing and designing control linear systems.

MATLAB/Simulink, we will encounter serious difficulties, because this software was not designed to develop graphical applications using this new approach. The creation of rich graphical interfaces using only MATLAB/Simulink is not an easy task, and high programming skills are required to develop useful applications with even a basic level of interactivity. When an instructor uses MATLAB/Simulink, s/he is an expert in the subject that is to be explained and in the use of

the mathematical facilities of the software to program algorithms. S/he is also capable of presenting the results by means of different types of plots and numerical values. However, control engineering instructors are not true programmers and the addition of an extra layer of interactivity to their MATLAB/Simulink applications would be very time-consuming. This statement also applies, even more strongly, to the development of virtual and remote labs.

Two software products now exist that can help solve this problem: SysQuake [3] and Easy Java Simulations [4]. The first is a commercial tool oriented to the development of interactive applications using a language fully compatible with MATLAB. Consequently, MATLAB legacy code (i.e. M-files) can be reused and benefit from SysQuake's high graphical capabilities to produce valuable, interactive instructional control applications. Additional information on this software can be found at <http://www.calerga.com>, and examples of advanced control applications developed with it are described in [5–9].

Easy Java Simulations (EJS) is an open source (free of charge) software tool that helps create dynamic, interactive scientific simulations in the Java language. The tool is targeted for both science students and teachers with basic programming skills and is therefore very much suited to the situation one finds in most university classrooms. Within 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 that readily responds to user interaction. Models can be described in EJS applications either using its self-contained facilities (ODE solvers and Java code editors) or Simulink block diagrams. If using the latter, EJS can wrap the Simulink model with a Java-made layer of interactivity. The final product is a Java application that efficiently interchanges data (input parameters and states variables) with the Simulink model through MATLAB's workspace (Fig. 3).

The tool can be used as a timely first step for control educators as a way of adopting an interactive-based approach in order to be at the cutting edge of innovative initiatives in education, such as the different world-wide open-source initiatives

that physics instructors are currently encountering [10]. Because of its interest, this paper describes EJS in more detail and presents two case studies of the development of control virtual labs.

The paper is organized as follows. First, Easy Java Simulations are briefly described, with particular emphasis on the development of control virtual labs that use MATLAB/Simulink as the simulation and mathematical engine. The next section shows an example of this collaboration between tools to develop a virtual lab on a magnetic levitator. Then a more advanced virtual lab is described, the quadruple-tank process, which also uses Simulink to develop the model. The paper ends with some conclusions on the combined use of EJS and MATLAB/Simulink as a way to foster the use of interactivity in control education, and, more precisely, the construction of instructional control virtual labs.

A brief survey of Easy Java Simulations

A quick description of Easy Java Simulations places it in the category of “code generators”. This means that users need to provide only the most relevant core of the simulation's algorithm and the tool automatically generates all the Java code needed to create a complete interactive simulation, including a wide range of sophisticated software techniques (such as handling computer graphic routines, communication protocols, multithreading and others). What makes EJS very special within this category of software is that it has been conceived *by* teachers and *for* teachers, and for students who are more interested in understanding the simulated phenomena than in the underlying computer-specific aspects.

The tool uses an original simplification of the successful Model-View-Control software paradigm

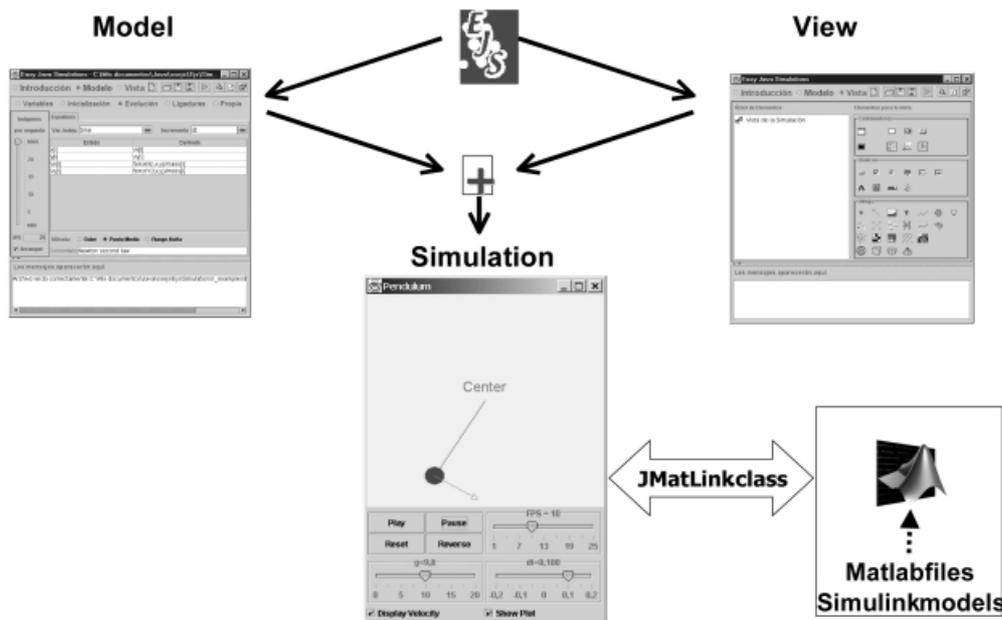


Fig. 3. Building a simulation using EJS and MATLAB/Simulink.

(MVC), and structures simulations in two main parts: model and view. The MVC paradigm states that a simulation is composed of three parts.

1. The model, which describes the phenomenon under study in terms of:
 - variables, which hold the different possible states of the phenomenon, and
 - relationships among these variables (corresponding to the laws that govern the phenomenon), expressed by computer algorithms.
2. The control, which defines certain actions that a user can perform on the simulation.
3. The view, which shows a graphical representation (either realistic or schematic) of the different states that the phenomenon can have.

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 variables of the model. 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. As mentioned above, EJS further simplifies the construction of a simulation by merging the control, half into the view, half into the model.

The model is what conveys the scientific part of the simulation and is thus the responsibility and the main interest of the user. Control teachers are used to describing their algorithms in terms of mathematical equations and to expressing these using a given computer language, or by means of high-level tools such as MATLAB/Simulink. The main target when creating the model of a simulation 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 (that is, the view)

is the part of the simulation that demands more knowledge of advanced programming techniques. And, to make the situation even worse, the addition of interactivity to this interface involves mastering sophisticated software techniques which demand a big investment of time and effort.

EJS helps its users through both tasks. The tool provides extensive scaffolding to define the model, while still retaining the flexibility of a general programming language, so that it is possible to specify almost any type of algorithm. This is pedagogically important, since the process of learning good control fundamentals consists, to a great extent, in understanding the basic principles required to build models.

In order to define the model in EJS, the user follows a prescribed sequence:

- declare the variables that describe the system;
- initialize these variables to a correct initial state;
- describe how the value of these variables changes over time; and
- establish how they affect each other when the user interacts with the system and modifies one or more of their values.

The last two steps consist in translating the mathematical equations or physical laws that rule the phenomenon under study into computer algorithms. Sometimes, though, the actual implementation of these algorithms can be a difficult task. For this reason, EJS provides two extra facilities. The first is a built-in editor and solver for systems of ordinary differential equations (ODE). The user writes the equations in a way that is similar to how s/he would write on a blackboard, and the system automatically generates the code that numerically solves the equations using one of several of the standard algorithms provided (Fig. 4). The editor also includes support to handle simple state events of the ODE. The second facility is a connection to MATLAB/Simulink that lets users design and solve their models with the help of these tools. In

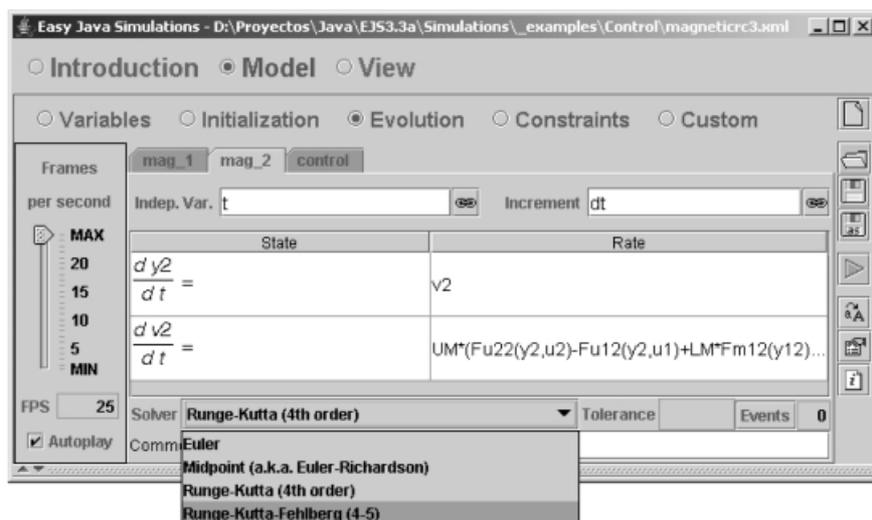


Fig. 4. Edition panel to write the ODE and select the solver.

this second case, the model is fully defined by a Simulink block diagram, and all that is needed is an M-file that informs EJS about the names of the MATLAB/Simulink variables so that it can create, once the final application is generated, a bi-directional link between the Java-coded interactive interface (the view) and the Simulink diagram for the model. Two examples on the use of this option are further described in the following sections.

In order to help create the view, EJS provides a set of advanced graphical elements (Fig. 5) that build on top of both standard Java *Swing* components (containers, buttons, text fields, sliders, combo boxes, etc.) and on specific scientific two- and three-dimensional visualization classes from the Open Source Physics project [10] (particles, vectors, images, vector and scalar fields, etc.). These elements can be used in a simple drag-and-drop way to build the interface. The user just 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 from different engineering perspectives in order to gain a better insight into the system.

To complete the view, its different elements have to be instructed to display on the screen according to the values of the variables of the model. Every graphic element of EJS has certain internal values, called “properties”, that can be customized to make the element look and behave in a particular way (change its displayed value, position, size, etc.). The user can very easily connect the properties of the graphical elements of the view to the value of the different variables of the model. Some of these properties can also be instructed to trigger user-defined actions (typical routines defined in the model) when the user interactively changes them. This procedure, which is called “linking” the

model and the view, is what turns the simulation into a dynamic, interactive application. This mechanism configures a simple, though very effective, way of designing and building advanced interactive user interfaces (Fig. 6).

The reason for this is that linking is a two-way connection. When the variables of the model change, the view is informed, so that it immediately displays the new state of the model. In return, since the view elements have built-in interactive capabilities, any interaction between the student and the interface immediately affects the model variables that have a connection to it. For example, let us imagine an EJS simulation of the control level of a tank. The tank and the water inside it are represented in the view by means of two coloured rectangles. If the dimensions of the tank are modified by dragging with the mouse the corners of the outer rectangle, the variables representing the tank width and height in the analytical model will reflect this change (thus affecting the dynamics of the system). Similarly, if the water level changes as a consequence of the model’s evolution, the visible height of the rectangle will change to display the current water level.

With all the 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 using EJS can either be used as stand-alone applications under different operating systems (for instance, a .BAT file is provided when running under Windows) or can be distributed via the Internet and run as applets within HTML pages (also generated by EJS) by using any Java-enabled web browser. The tool also includes a simple

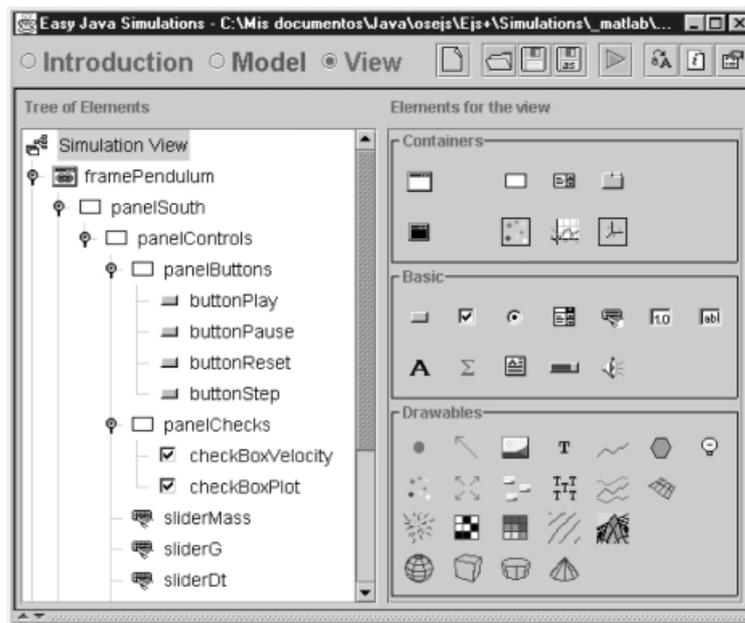


Fig. 5. Library of the graphical elements of EJS.

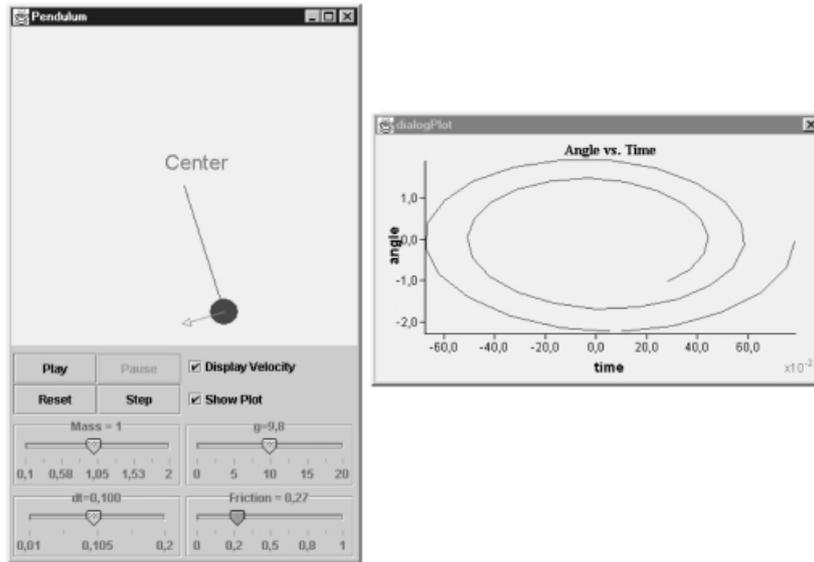


Fig. 6. View of an interactive application developed using EJS.

HTML editor to help the teacher enhance the generated web pages with pedagogical information and/or instructions for the simulation. At present, the use of MATLAB/Simulink is not supported when running simulations as applets, although we are working on a future model that will allow EJS to generate distributed applications with Java interfaces and Simulink models running in different computers using the Internet to interchange information.

A virtual lab for the magnetic levitator system

The system modeled in this section is the ECP magnetic levitation system (model 730) [11]. Fig. 7 shows the free-body diagram of two suspended magnets. Each magnet is acted on by forces from either driver coil, from the other magnet, from gravity and from friction (modeled as viscous). The differential equations for both magnets are:

$$m_1\ddot{y}_1 + c_1\dot{y}_1 + F_{m12} = F_{u11} - F_{u21} - mg \quad (1)$$

$$m_2\ddot{y}_2 + c_2\dot{y}_2 + F_{m12} = F_{u22} - F_{u12} - mg \quad (2)$$

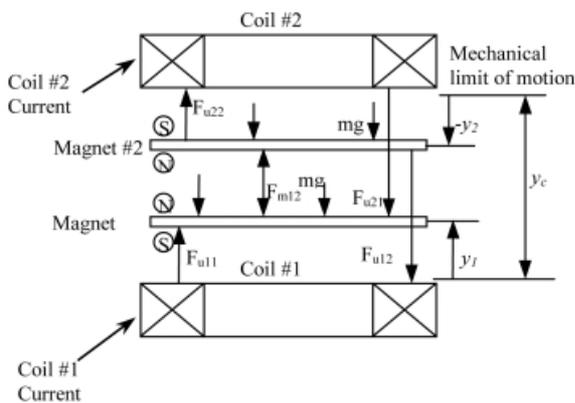


Fig. 7. Free-body diagram of the magnetic levitator.

The magnetic force terms are modeled by [11]:

$$F_{u11} = \frac{i_1}{a(y_1 + b)^N} \quad (3)$$

$$F_{u12} = \frac{i_1}{a(y_c + y_2 + b)^N} \quad (4)$$

$$F_{u21} = \frac{i_2}{a(y_c - y_1 + b)^N} \quad (5)$$

$$F_{u22} = \frac{i_2}{a(-y_2 + b)^N} \quad (6)$$

$$F_{m12} = \frac{c}{(y_{12} + d)^N} \quad (7)$$

$$y_{12} = y_c + y_2 - y_1 \quad (8)$$

where a, b, c, d and N are constants which may be determined by numerical modeling of the magnetic configuration. In this case, $N = 4$ and a, b, c and d have been determined by empirical methods in order to adjust the simulation and experimental results. The identification of the parameters of the model was done by measuring the specific input/output characteristics of the magnet/coil actuators and the magnet/magnet interactions as they vary with relative position. The strong magnetic field nonlinearity, however, is inherent to this class of magnetic systems. For most control design purposes, the upper and lower actuators' characteristics may be assumed to be identical.

The interaction of the two magnets has a nonlinear force characteristic. When two magnets are used in the configuration, they are usually placed with like poles facing each other so that they repel. When the lower magnet is stationary, the upper magnet comes to rest in equilibrium between the upward repulsive magnet force and downward gravitational force. The magnetic force

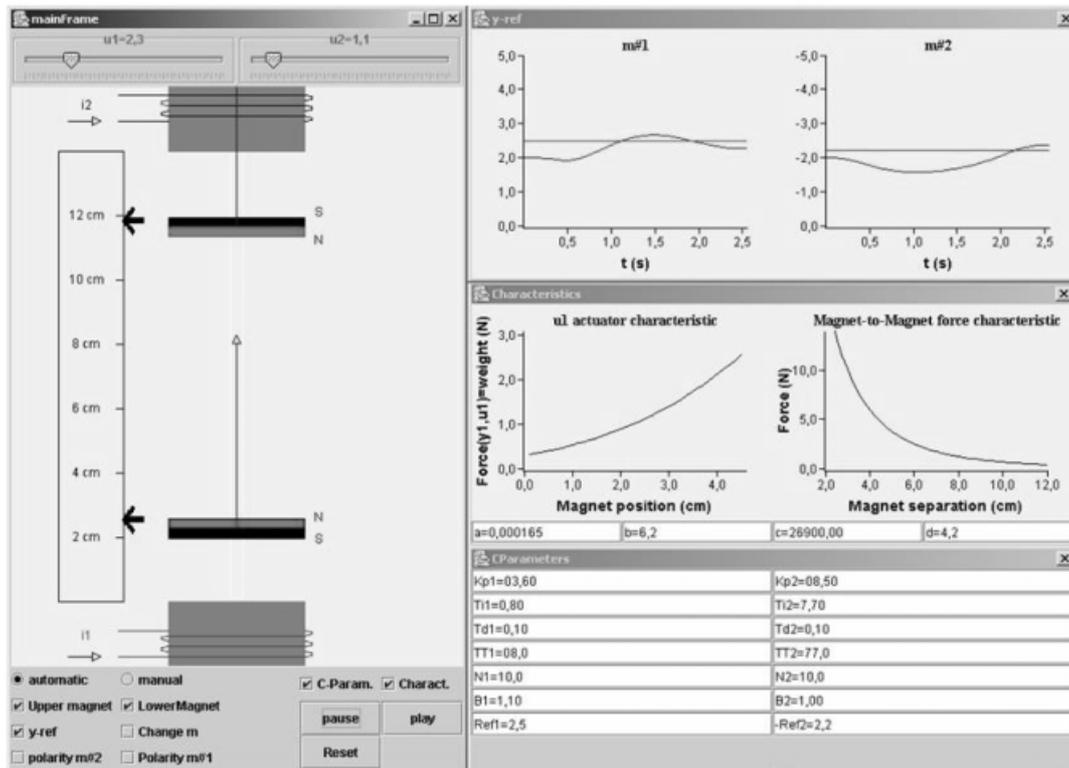


Fig. 8. Magnetic levitation virtual lab.

characteristic may be measured by adding and subtracting weight from the upper magnet and measuring the equilibrium height. Using this procedure, we obtained the following values for the parameters: $a = 1.65$ N, $b = 6.2$ cm, $c = 2.69$ Ncm³ y $d = 4.2$ cm.

Implementation of the magnetic levitator with EJS and MATLAB/Simulink

When developing this type of simulation, one of the most important things that the teacher needs to bear in mind is utilising the correct configuration of windows and operating controls of the simulation in order to facilitate the student's understanding of the virtual lab. In our case, this view (using EJS terminology) is divided into several elements, as displayed in Fig. 8. Because the simulation is to be used for didactical purposes, it was very important for us to provide a visualization that displays the physical system as realistically as possible, but that also provides support for graphic tools commonly used in control (plots, diagrams, input devices, controllers, etc.). A second requirement was to provide a set of interactive elements that could be manipulated by the students to make changes in a dynamic way during an experiment. The view is therefore structured in one main window and three secondary dialog windows. The main window is located at the left-hand side of Fig. 8. The big central part of the main window schematically displays the magnetic levitator and allows the user to change interactively the set points by dragging the arrows up and down. The

lower part of the main window is filled with buttons and check boxes that allow the user to choose the operating conditions of the system. With these elements the user can select to:

- Play, pause and reset the simulation.
- Choose the control system mode: manual or automatic.
- Change the mass of the magnets' (change m).
- Change the magnets' polarity (polarity m#1 and polarity m#2).
- Select the magnets that will operate the levitator (upper magnet and/or lower magnet).

The last two options can be used to configure the virtual lab as a variety of single-input single-output (SISO) and multi-input multi-output (MIMO) systems. By using a repulsive force from the lower coil to levitate a single magnet, an open-loop stable SISO system is created. Attractive levitation via the upper coil configures an open-loop unstable system. Two magnets may be raised by a single coil to produce a SIMO plant. If two coils are used, a MIMO plant is produced. These may be locally stable or unstable, depending on the selection of the magnets' polarities and the nominal magnet positions. The plant has inherently strong nonlinearities, due to the natural properties of magnetic fields. Thus, this virtual lab provides a dynamically rich test bed for implementation of introductory and advanced control methods.

However, the most relevant feature of this didactical application is that MATLAB/Simulink

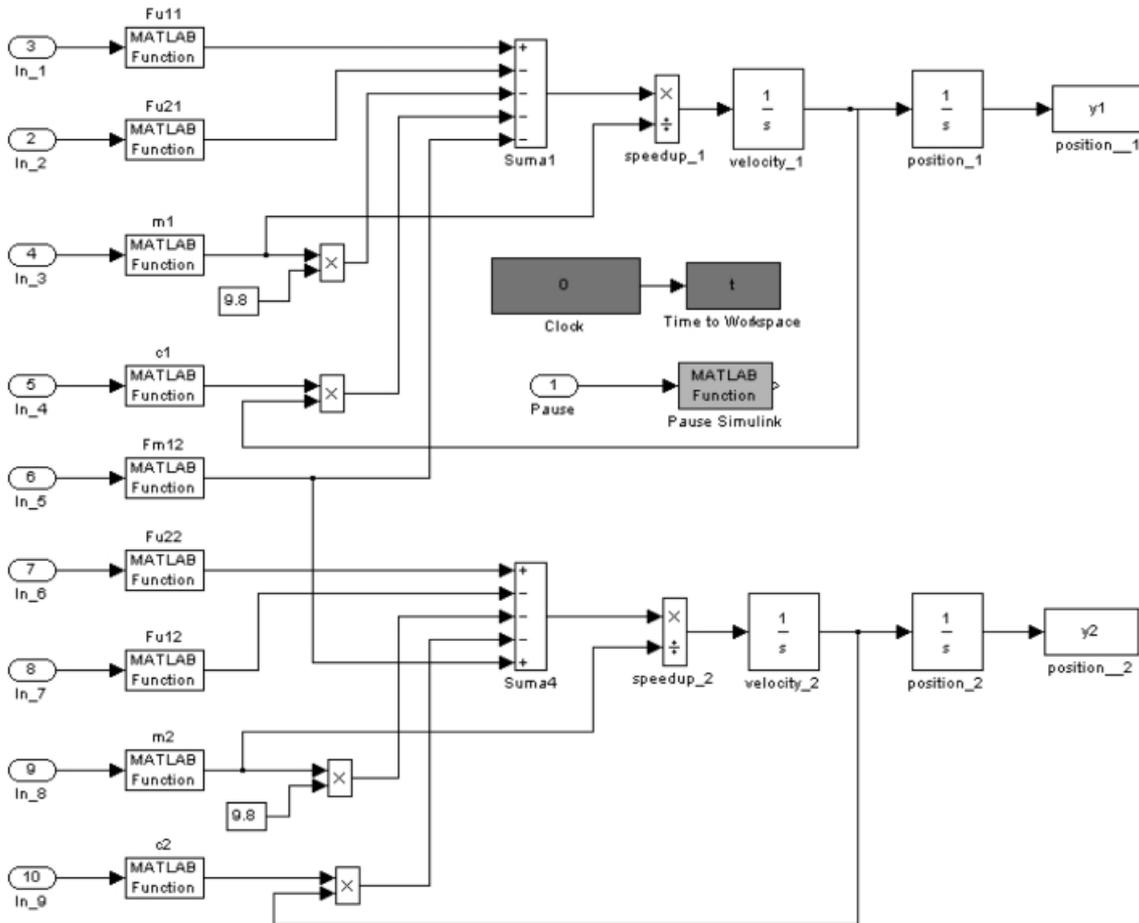


Fig. 9. Modified Simulink block diagrams.

is running in the background. Fig. 9 shows the Simulink block diagram of the magnetic levitator described in equations 1 to 8 with the modifications needed for it to be used by EJS.

There are three main steps to using a standard Simulink block diagram within EJS. The first consists in making some changes to the original Simulink model. These changes are necessary because Simulink models are, obviously, created to be run within Simulink. But, since the communication between Simulink and EJS goes through MATLAB's workspace, we first need to change the Simulink model so that it sends/receives the value of some of its variables and parameters to/from MATLAB's workspace. Fig. 9 shows how some system parameters of the block diagram have been adapted to be read from MATLAB's workspace: magnets' masses, polarities, and parameters of the magnetic configuration. The block diagram has also been slightly modified so that it writes the simulation state in MATLAB's workspace (time and positions of the two magnets) after every integration step.

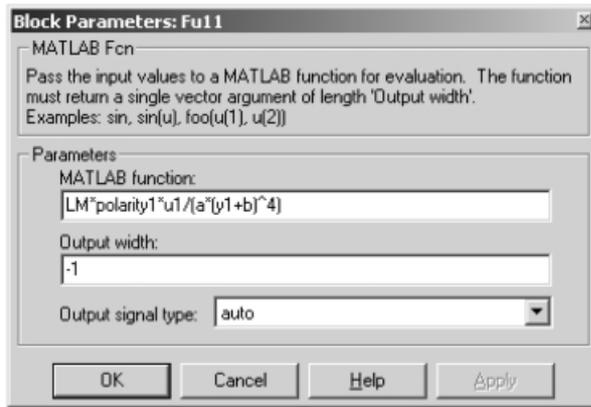
The reading of the values from MATLAB's workspace variables is done by including "MATLAB function" blocks in the diagram to evaluate variables and/or expressions. Fig. 10a shows a Simulink block in which an expression

using MATLAB variables is evaluated to generate one of the magnetic forces. To write the model state back into MATLAB's workspace after each integration step, it is necessary to include a "To workspace" block in the diagram for each of the variables in the state vector of the model. As an example, Fig. 10b shows the block used to send the position of magnet 1 (variable y_1) back to the workspace.

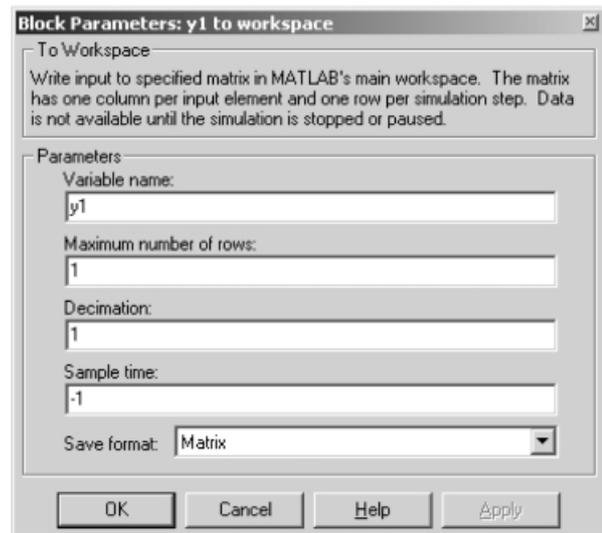
A second difference to a standard block diagram comes from the fact that Simulink models are usually played and paused by the user through Simulink's user interface. Hence, the model must be changed so that it stops after every integration step. For this, it is necessary to include in the model diagram one "MATLAB function" block with the MATLAB command "set_param (gcs, 'SimulationCommand', 'Pause')". In this way, EJS can control exactly when the model needs to play.

The second step needed to connect the Simulink model with EJS is the creation of a simple M-file. This file provides EJS with basic information about the model and about the variables and parameters that can be accessed through MATLAB's workspace. Fig. 11 shows an extract from the file that we created for this example.

The syntax for this text file is rather simple.



(a)



(b)

Fig. 10. Source and sink blocks to communicate MATLAB's workspace and Simulink.

What enables EJS to read and access these variables is the special comment at the end of each line. This comment always starts with the keyword “EJS” immediately after the comment character “%”. The keyword must be followed by one of the following words: *Model*, *Variable*, or *Parameter*.

- *Model* is used to tell EJS the name of the Simulink file that describes the model.
- *Variable* provides EJS with a description of a variable of the Simulink model. A variable can be declared to be accessible for reading, writing or both. If no modifier is indicated, the variable can be freely accessed and EJS will synchronize its value with the associated EJS variable before and after every integration step of Simulink, as, for example, happens with the positions of the magnets (variables y_1 and y_2). If declared as *InputOnly*, however, EJS will understand that the variable is not to be changed by the Simulink model and will always force its value to whatever it is within EJS. In our example, the masses (variables m_1 and m_2) and the control actions (u_1 and u_2) are input-only variables that will be set by the user through the simulation interface. Finally, if declared as *OutputOnly*, EJS will read the variable after every integration step of

the Simulink model is run, but it will not try to change its value in the Simulink model. For instance, the time variable of the model is read only since it is the result of Simulink execution after each integration step.

- *Parameter* declares variables that are defined inside Simulink blocks. Ejs makes no distinction between variables and parameters for computational use. However, for technical reasons, the comment for a parameter must include information about the Simulink block or menu in which the parameter is defined and the function of this parameter within the block. In our example, the integration step dt is declared to be the general (that is, not defined within any block) parameter $maxstep$, since it is located in a dialog panel of the menu *Parameters* located in the Simulink menu *Simulation*.

The third and final step in this process consists of connecting the variables of the EJS model with MATLAB's workspace variables (which are in turn associated with variables of the Simulink model). Fig. 12 shows the variables in EJS for our example and how they are connected to MATLAB's variables. Notice that we have chosen similar (if not identical) names for both sets of variables.

```

model='levitator.mdl'; %Simulink file           %EJS Model
y1;                %Position of magnet 1      %EJS Variable
y2;                %Position of magnet 2      %EJS Variable
t;                 %Current time              %EJS Variable OutputOnly
dt;                %Integration step          %EJS Parameter= maxstep InputOnly
u1;                %Control action 1          %EJS Variable InputOnly
u2;                %Control action 2          %EJS Variable InputOnly
a;                 %Magnetic configuration %EJS Variable InputOnly
b;                 %Magnetic configuration %EJS Variable InputOnly
c;                 %Magnetic configuration %EJS Variable InputOnly
d;                 %Magnetic configuration %EJS Variable InputOnly
m1;                %Mass of magnet 1           %EJS Variable InputOnly
m2;                %Mass of magnet 2           %EJS Variable InputOnly

```

Fig. 11. Extract from the file *levitator.m*, used to connect Simulink and EJS.

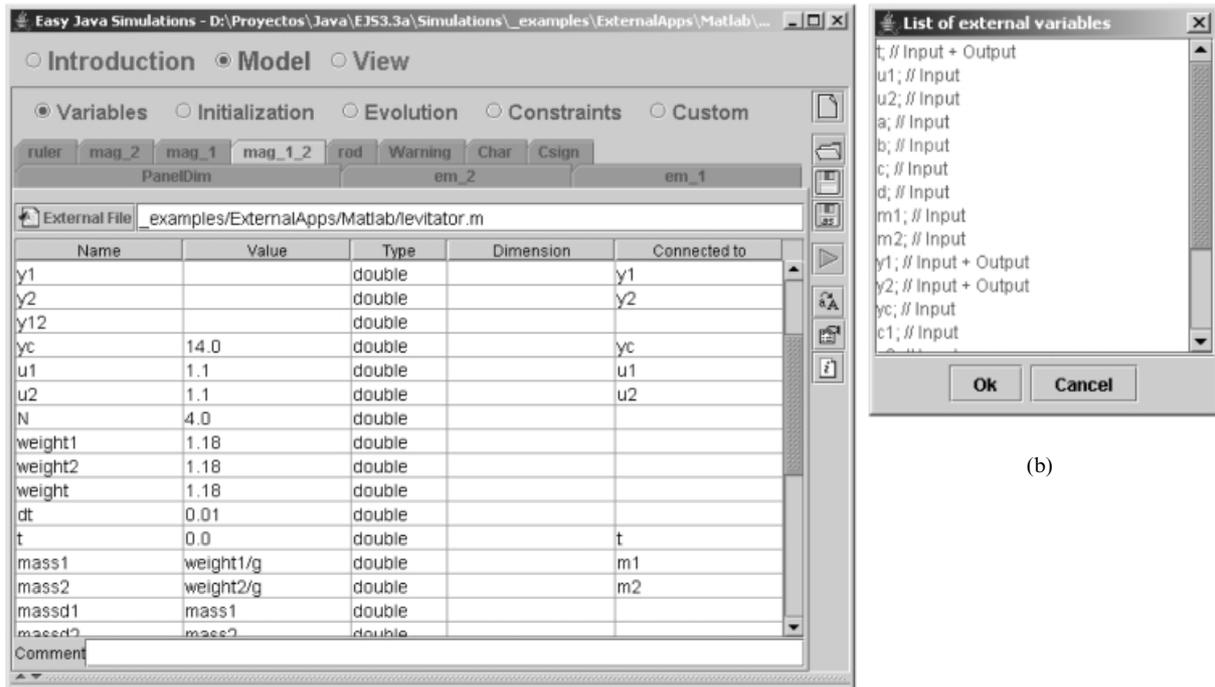


Fig. 12. EJS page of variables showing the connection to MATLAB's variables.

To make the connection between EJS and MATLAB/Simulink it is necessary to open a page of external variables in the EJS application (Fig. 12a). In this page it is necessary to introduce into the *External File* text field the path to the information M-file previously created for the levitator. Now, whenever you create an EJS variable and right-click on it, you will be given a window with a list of the MATLAB variables to which you can connect your EJS variable (Fig. 12b). By clicking on any of these variables, the connection between the EJS variable and the MATLAB variable will be automatically established. Establishing a connection means exactly that: a) the value of the EJS variable will be pushed to the variable of the model before running Simulink, and b) the EJS

variable will be given back whatever value the Simulink variable has after the model has run one integration step.

Our Simulink model is now ready to be used. EJS will run the Simulink model whenever we insert the Java sentence

```
external.step(1);
```

in a page of the evolution section (the main loop) of EJS (Fig. 13). A call to this method has the following effects:

1. The value of every EJS variable that is connected to a MATLAB variable (except those declared as *OutputOnly*) is pushed to the Simulink model.

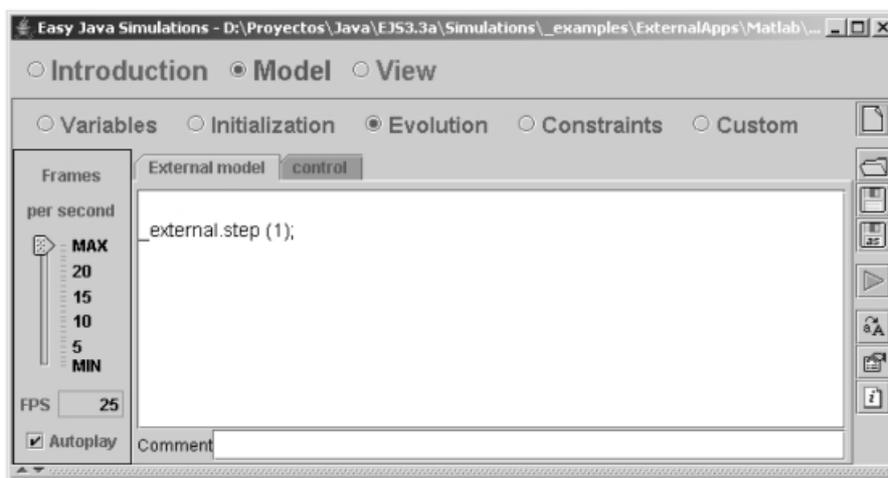


Fig. 13. Evolution page of the EJS to control an external model running in MATLAB/Simulink.

2. The Simulink model is run exactly one integration step.
3. The value of all MATLAB variables that are connected to EJS variables (except those declared as `InputOnly`) are retrieved from the Simulink model.

We can now operate EJS in the standard way in order to build our desired view and to link it to the variables of the model in a form that suits our pedagogical needs. Fig. 14a shows the tree representation of the view elements that we chose for our view (see also Fig. 8). As an example, Fig. 14b displays the table of properties for the graphical element `mass2`, which allows users to resize magnet 2 and change its mass. Fig. 14c shows how a Java method called `ChangeMass2` is linked to the “On Drag” action of this graphical object. In this way, when the user drags the object the magnet will be resized and the Java method will be invoked to compute the new value of the magnet mass according to its current dimensions.

Finally, it is worth noting that EJS is capable of running more than one Simulink model from within one single simulation. This is simply done by creating two or more separate pages of variables, each with a different M-file, and connecting EJS variables to the corresponding MATLAB variables. When the system runs, EJS will take care of opening as many MATLAB sessions as are needed and of managing all connections.

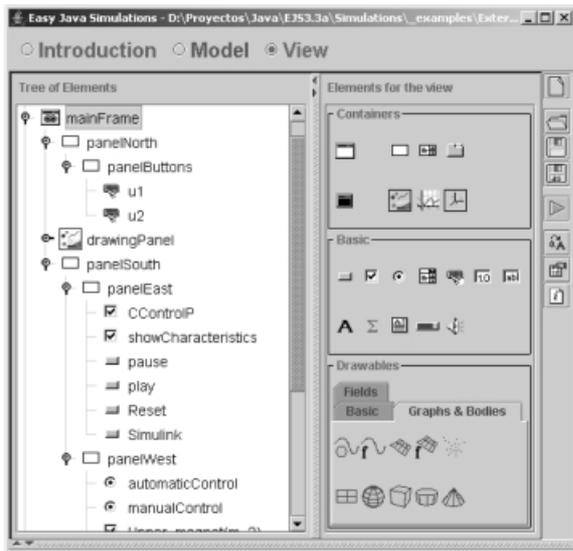
An advanced virtual lab: the quadruple-tank process

Our second example of an EJS/MATLAB application is the quadruple-tank process. Originally introduced by Johansson [12], it has received great attention because it presents interesting properties in both control education and research. The quadruple-tank exhibits complex dynamics in an elegant way. Such dynamic characteristics

include interactions and a transmission zero location that can be tuned in operation. With adequate tuning, this system presents non-minimum-phase behavior that arises due to the multivariable nature of the problem. For this reason, the quadruple-tank has been used to show the results of different control strategies and as an educational tool in teaching advanced multivariable control techniques. Although the setup is simple, the process can still illustrate interesting multivariable phenomena. The process flowsheet is displayed in Fig. 15. The target is to control the levels h_1 and h_2 in the lower two tanks with two pumps. The process inputs are the voltages, v_1 and v_2 , applied to the pumps.

The differential equations representing the mass balances in this quadruple-tank process are:

$$\begin{aligned} \frac{dh_1}{dt} &= -\frac{a_1}{s_1(h_1)}\sqrt{2g|h_1|} \\ &\quad + \frac{a_3}{s_1(h_1)}\sqrt{2g|h_3|} + \frac{\gamma_1 k_1 v_1}{s_1(h_1)} \\ \frac{dh_2}{dt} &= -\frac{a_2}{s_2(h_2)}\sqrt{2g|h_2|} \\ &\quad + \frac{a_4}{s_2(h_2)}\sqrt{2g|h_4|} + \frac{\gamma_2 k_2 v_2}{s_2(h_2)} \\ \frac{dh_3}{dt} &= -\frac{a_3}{s_3(h_3)}\sqrt{2g|h_3|} \\ &\quad + \frac{(1-\gamma_2)k_2 v_2}{s_3(h_3)} - \frac{k_{d1}d_1}{s_3(h_3)} \\ \frac{dh_4}{dt} &= -\frac{a_4}{s_4(h_4)}\sqrt{2g|h_4|} \\ &\quad + \frac{(1-\gamma_1)k_1 v_1}{s_4(h_4)} - \frac{k_{d2}d_2}{s_4(h_4)} \end{aligned} \quad (9)$$



(b)

```
public void ChangeMass2() {
    xpm2=xpm1;
    w2=-ypm2+ym2;
    . . . . .
    mass2=w2/wd*massd2;
    weight2=mass2*g;
    . . . . .
}
```

Fig. 14. Elements for the view of the magnetic levitator.

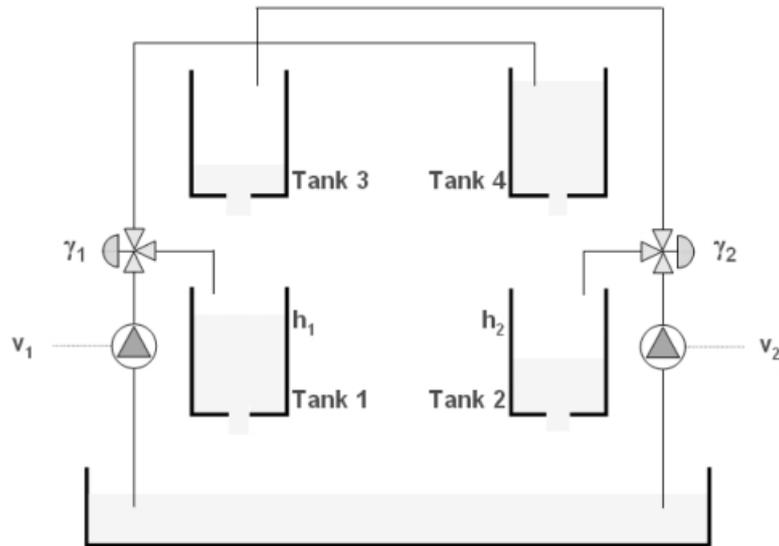


Fig. 15. Schematic view of the quadruple-tank process.

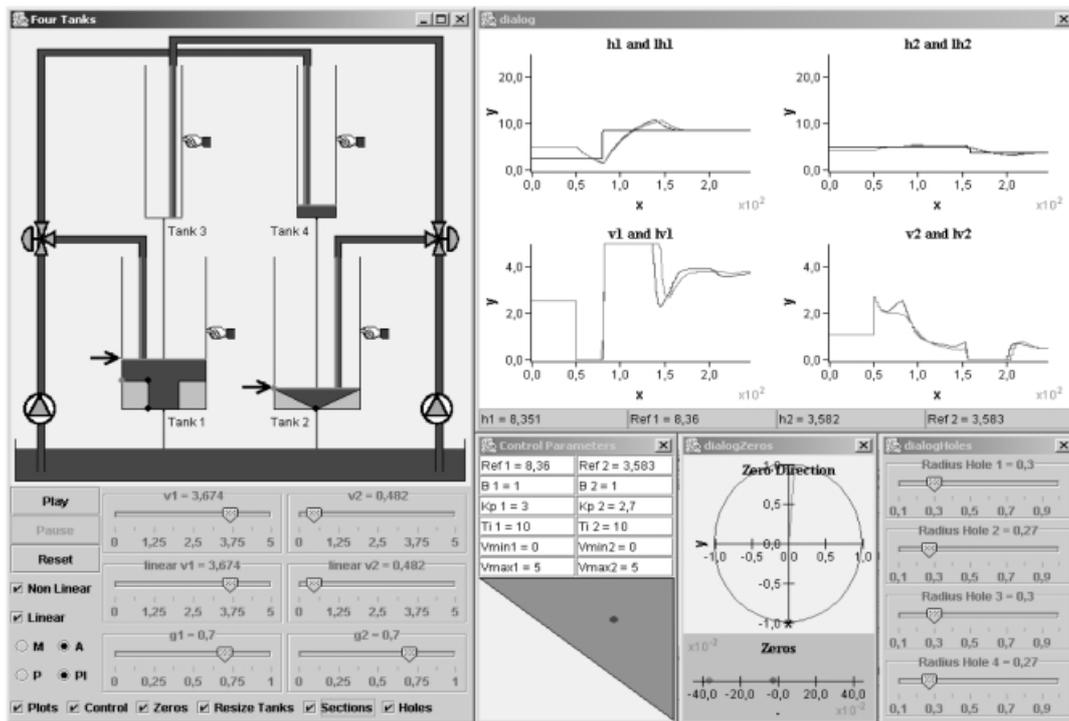


Fig. 16. View of the quadruple-tank virtual lab.

where h_i is the liquid level in tank i ; a_i is the outlet cross-sectional area of tank i ; $s_i(h_i)$ is the cross-sectional area of tank i ; v_j is the speed setting of pump j , with the corresponding gain k_j ; γ_j is the portion of the flow that goes into the upper tank from pump j ; and d_1 and d_2 are flow disturbances from tank 3 and tank 4 respectively, with corresponding gains k_{d1} and k_{d2} . The process manipulated inputs are v_1 and v_2 (speed settings to the pumps) and the measured outputs are y_1 and y_2 (voltages from level measurement devices). The measured level signals are assumed to be propor-

tional to the true level; i.e., $y_1 = k_{m1}h_1$ and $y_2 = k_{m2}h_2$. The level sensors are calibrated so that $k_{m1} = k_{m2} = 1$.

This process exhibits interacting multivariable dynamics, because each of the pumps affects both of the outputs. The linearized model of the quadruple-tank process has a multivariable zero, which can be located in either the left or the right half-plane by simply adjusting the throttle valves γ_1 and γ_2 . Johansson showed that the inverse response (non-minimum phase) occurs when $0 < \gamma_1 + \gamma_2 < 1$ and minimum phase for $1 < \gamma_1 + \gamma_2 = 2$. The valve

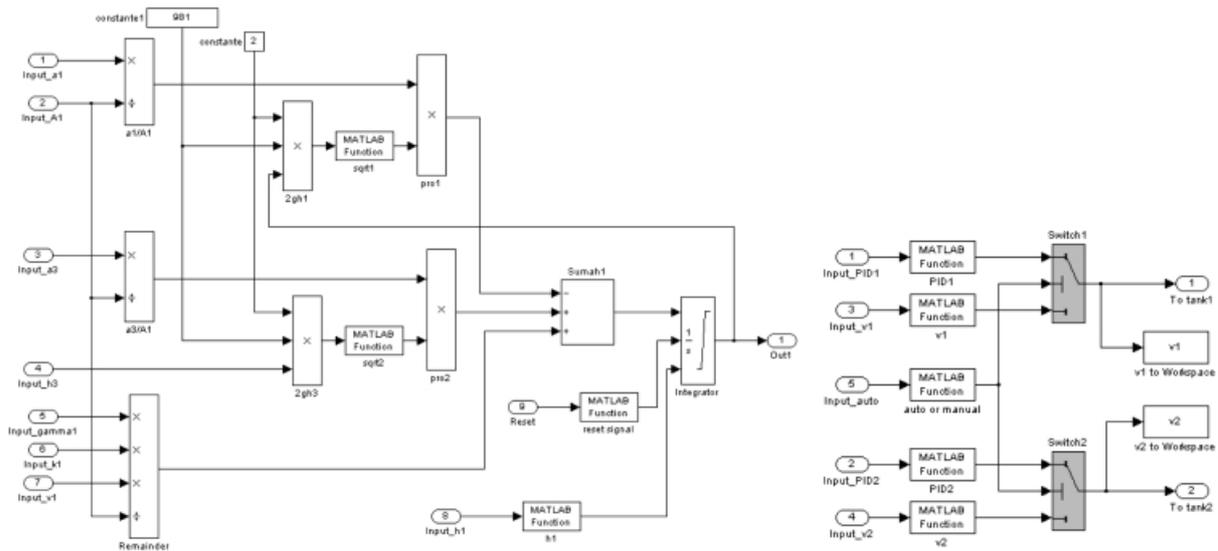


Fig. 17. Simulink subsystems of tank 1 and the PID controllers.

```

model='fourtanks.mdl'; %Ejs Model
time; %Ejs Variable Outputonly
h1; %Ejs Variable
h2; %Ejs Variable
.....
diam1; %Ejs Variable Inputonly
diam2; %Ejs Variable Inputonly
gamma1; %Ejs Variable Inputonly
gamma2; %Ejs Variable Inputonly
v1; %Ejs Variable
v2; %Ejs Variable
controller1; %Ejs Variable
controller2; %Ejs Variable
auto; %Ejs Variable Inputonly

```

Fig. 18. Extract of the M-file *fourtanks.m* to connect Simulink and EJS.

settings will give the overall system entirely different behavior from a multivariable control viewpoint. Unmeasured disturbances can be applied by pumping water out of the top tanks and into the lower reservoir. This exposes students to disturbances rejection as well as reference tracking.

Implementation of the quadruple-tank with EJS and MATLAB/Simulink

The main window of the EJS application for this example can be seen at the left-hand side of Fig. 16. Schematically displayed in its upper part are the four tanks and the set of dripping holes, pipes, pumps and valves that configure the system. We tried to make this first image as self-explanatory as possible. The black arrows indicate the set points of the lower tanks, those under control. These set points can be changed interactively by dragging the arrows up and down. The lower part of the main window is occupied by primary controls that let the user specify how the system should run. With the left-hand column of buttons, the user can perform the following actions:

- Play, pause and reset the simulation.
- Select to run the nonlinear or linear models (or both) for the system.
- Choose whether to control the system manually (M) or automatically (A).
- Select, in the case of automatic control, whether to use a P or PI controller (this can easily be extended to other types of controllers).

A detailed explanation about the possibilities of this control application can be found at [13]. However, in that work, the application was fully developed using EJS. In this paper, the four-tank application presented simulates the nonlinear part of the model using a Simulink block diagram, and the linear model and the view have been programmed under EJS.

The process required to create such a hybrid application is similar to the one used for the magnetic levitator example. For the first step of this process, we first had to implement the nonlinear differential equations corresponding to the tanks (Equations 9) in a Simulink file (Fig. 17a

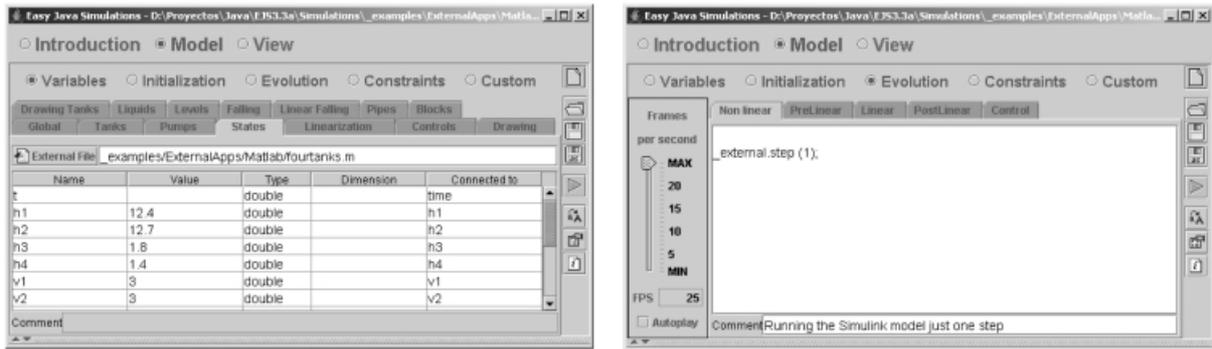


Fig. 19 (a). Variable page illustrating the link between some EJS and MATLAB variables. (b) Evolution page showing the EJS instruction to order Simulink runs one step of simulation.

shows the Simulink subsystem of tank 1). We also created an M-function for the PID controller used to control the tank levels and included this M-function in two Simulink blocks connected to the nonlinear models of tanks 1 and 2. Fig. 17b shows the diagram of the Simulink subsystem used to implement the manual and automatic control system; the PID M-function is included in the PID1 and PID2 blocks, which are used to compute the control signals v_1 and v_2 (the voltages of the two pumps); two switch blocks, named Switch1 and Switch2, allow us to select automatic or manual control, depending on the value of a MATLAB variable, `auto`, that can be interactively changed using EJS view. In this way, when the user selects automatic control in the graphical view, the values of the control signals v_1 and v_2 are computed and the two PID blocks are selected to drive the Simulink subsystems for tanks 1 and 2 respectively; however, when the control is manual, the values of the variables v_1 and v_2 are set directly by the user and sent to the MATLAB workspace, from which the Simulink blocks, v_1 and v_2 , read and pass them to the subsystems for tanks 1 and 2.

We also needed to adapt the block diagram for interaction with EJS. The mechanism for this is similar to that used for the magnetic levitator example and requires the introduction of enough source and sink blocks as needed to exchange information between Simulink and MATLAB's workspace.

The second step of the process was to create the M-file to inform EJS about accessible MATLAB variables. Fig. 18 shows an extract of the text file used for this example.

The third and final step was to open this file from EJS and to connect MATLAB variables with EJS variables. Fig. 19a shows how EJS variables are connected to the MATLAB variables defined in this M-file.

Once all this was done, we could introduce a simple sentence in the form `"_external.step(1);"` in one of EJS's evolution pages to instruct Simulink to run the block diagram once. This way, at every evolution step of EJS, the two

models (the linear and the nonlinear) advance in a synchronized way, generating the state of the system. Note that both models share the same user interface: the nonlinear model developed in Simulink, and the linear model written using the features of EJS to define and solve differential equations. The use of different models within the same application is completely transparent to the user. EJS generates the necessary Java code to exchange the information between MATLAB and EJS variables.

CONCLUSION

Using computer simulations in an instructional context usually implies using computers to build models of real-world phenomena in order to help students grasp the fundamentals of the behavior of complex systems. Interacting with an instructional simulation can enable learners to gain better engineering understanding of a real system, process or phenomenon through exploring, testing hypotheses, and discovering explanations for the mechanisms and processes. Furthermore, students have an excellent opportunity to "experiment" with their own ideas in terms of engineering design by simple interaction with the tool. Not only can interactive virtual labs be effective in presenting engineering concepts in the classroom, they also can be beneficial in extending students' experience in analysis and design assignments. This invitation to creativity can be most useful when it comes to specialized control engineering student projects.

In this context, many control educators would be more willing to use interactive simulations in their lectures and labs if they had a set of tools that would facilitate their development. An obvious way to achieve this is to select one of the well-known software packages in the control engineering field, such as MATLAB/Simulink, and to provide a tool that would wrap models with the necessary layers of interactivity without demanding a deep knowledge of computer programming. From the point of view of MATLAB's users, the benefits of this approximation is twofold: it lets

them reuse all the MATLAB legacy code and also takes advantage of their experience in the use of this de facto standard.

This paper shows two examples of the type of control applications that have been created using Easy Java Simulations, a software tool that helps create, in a very easy way, dynamic and interactive scientific simulations in the Java language. In the view of the two final developments, it is clear that EJS takes a step forward in the use of interactivity and MATLAB in an instructional environment. A more detailed explanation of the development of EJS applications using MATLAB/Simulink can be

found in the EJS documentation. The software package and the examples shown in this paper are available for free at <http://fem.um.es/EJS>.

Finally, we would like to emphasize that under no circumstances should instructors forget that their ultimate goal is to arouse students' curiosity to discover, question, and wonder. Interactivity is one of the inherent ways for human beings to perform this.

Acknowledgements—This work has been supported by the Spanish CICYT under grant DPI2001-1012 and DPI2004-01804.

REFERENCES

1. S. Dormido, The role of interactivity in control learning (Plenary Lecture), IFAC Symposium on Advances Control Education ACE'03, Oulu, Finland, June 2003.
2. S. Dormido, Control learning: Present and future, *Annual Reviews in Control*, **28** (2004), pp. 115–136.
3. Y. Piguet, *SysQuake: User Manual*, Calerga (1999).
4. E. Esquembre, Easy Java Simulations: A software tool to create scientific simulations in Java, *Comp. Phys. Comm.* (2002). See also <http://fem.um.es/Ejs>.
5. S. Dormido, J. Aranda, J. M. Díaz and S. Dormido-Canto, Interactive educational environment for design by QFT methodology, *Proceedings of 5th International Symposium on Quantitative Feedback Theory and Robust Frequency Domain Methods*, pp. 223–230, Pamplona (August 2001).
6. S. Dormido, F. Gordillo, S. Dormido-Canto and J. Aracil, An interactive tool for introductory nonlinear control systems education, IFAC World Congress b '02, Barcelona, July 2002.
7. P. Albertos, J. Salt, S. Dormido and A. Cuenca, An interactive simulation tool for the study of multirate sampled data systems, IFAC Symposium on Advances Control Education ACE '03, Oulu, Finland, June 2003.
8. S. Dormido, M. Berenguel, S. Dormido-Canto and F. Rodríguez, Interactive learning of introductory constrained generalized predictive control, IFAC Symposium on Advances Control Education ACE '03, Oulu, Finland, June 2003.
9. N. Tan, D. Atherton and S. Dormido, Systems with variable parameters: Classical control extensions for undergraduates, IFAC Symposium on Advances Control Education ACE '03, Oulu, Finland, June 2003.
10. W. Christian, The open source physics project (<http://www.opensourcephysics.org>).
11. ECP Educational Control Products, *Magnetic Levitation System*, manual for model 730 (1999).
12. K. H. Johansson, The quadruple-tank process: A multivariable laboratory process with an adjustable zero, *IEEE Trans. on Control Systems Technology*, **8**(3) (2000), pp. 456–465.
13. S. Dormido and F. Esquembre, *The Quadruple-Tank Process: An Interactive Tool for Control Education*, Proceedings of the European Control Conference, Cambridge, UK (2003).

José Sánchez received his Computer Sciences degree from Madrid Polytechnic University in 1994 and his Ph.D. from UNED in 2001. Since 1993, he has been working at UNED's Department of Computer Sciences and Automatic Control as an Associate Professor.

Francisco Esquembre was born in Alicante (1963) and received a Ph.D. in Mathematics in June 1991 from the University of Murcia, Spain, where he has worked since 1986, holding a permanent job as an Associate Professor from 1994.

Sebastián Dormido received his Physics degree from Madrid Complutense University (1968) and his Ph.D. from Country Vasque University (1971). In 1981, he was appointed Professor of Control Engineering at UNED. He has supervised 25 Ph.D. theses and co-authored more than 150 conference papers and 100 journal papers. Since 2002 he has been President of the Spanish Association of Automatic Control, CEA-IFAC. His scientific activity includes various subjects in the control engineering field: computer control of industrial processes, model-based predictive control, robust control, modeling and simulation of hybrid systems and control education with special emphasis on remotes and virtual labs.

Carla Martín received her M.Sc. in Electrical Engineering in 2001 from University Complutense of Madrid. She is currently completing her doctorate studies at UNED, Spain.

S. Dormido-Canto received his M.Sc. in Electronic Engineering in 1994 from the ICAI and his Ph.D. in 2001 from UNED. Since 1994, he has been working at UNED's Department of Computer Sciences and Automatic Control as an Assistant Professor.

R. D. Dormido received her M.Sc. in Physics in 1994 from University Complutense of Madrid and her Ph.D. in 2001 from UNED. Since 1994, she has been working at UNED's Department of Computer Sciences and Automatic Control as an Assistant Professor.

R. Pastor received his M.Sc. in Physics in 1994 from University Complutense of Madrid. He is currently completing his doctorate studies at UNED, Spain.

A. Urquía received his M.Sc. in Physics in 1992 from University Complutense of Madrid and his Ph.D. in 2000 from UNED. Currently he is working as an associate professor at UNED, Spain.