The Learning of Control Concepts Using Interactive Tools

JOSÉ SÁNCHEZ,[1] SEBASTIÁN DORMIDO,[1] FRANCISCO ESQUEMBRE[2]

[1]*Department of Computer Sciences and Automatic Control, Spanish National University for Distance Education, Madrid, Spain*

[2]*Department of Mathematics, University of Murcia, Murcia, Spain*

Received 24 January 2003; accepted 20 July 2004

ABSTRACT: This paper discusses the creation of interactive computer simulations that implement virtual laboratories in the field of Control Engineering education. Easy Java Simulations (Ejs, http://fem.um.es/Ejs/), a Java-based tool that helps create interactive dynamic simulations, is introduced. 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. We describe in this paper this particular feature in detail, and provide some examples that show the advantages that this tool offers to the world-wide engineering education community. Ejs is a freeware, open source, tool aimed for educators with low profile in programming but deep knowledge on certain technical and scientific fields. The tool allows to develop complete, interactive simulations in three steps: describing the mathematical model (optionally using Matlab/Simulink), building the user interface using off-the-shelf graphical elements, and connecting certain properties of these elements to the variables of the model. This last step is the key point to transform a classical simulation into a full interactive application. ©2005 Wiley Periodicals, Inc. Comput Appl Eng Educ 13: 84–98, 2005; Published online in Wiley InterScience (www.interscience.wiley.com); DOI 10.1002/cae.20033

Keywords: control education; interactivity; web-based simulation; virtual laboratory; dynamic picture; interactive module

INTRODUCTION

It is easy to realize, while flying through any book on control engineering, that most of the ideas, concepts, and methods in this topic, are so rich in visual content that they can be represented and manipulated intuitively and geometrically (root locus, Bode diagrams, Nyquist, and Nichols plot, to name a few). Actually, many of the basic ideas in automatic control often arise from very specific and visual situations. 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 the knowledge. One of the most

Correspondence to J. Sánchez (jsanchez@dia.uned.es).

Contract grant sponsor: Spanish CICYT; contract grant numbers: DPI2001-1012, DPI2004-01804.

Contract grant sponsor: Fundación Séneca, The Center for Coordination of Research of the Region of Murcia, Spain.

important tasks for teachers in control engineering is to transfer to students not only the formal and logic structure of the discipline but also the strategic and intuitive aspects of the subject [1].

The use of interactive multi-windowed computer-aids in instruction (*interactive modules*) is particularly significant because it provides practical insight into control system fundamentals. In essence, an interactive module in control education is a collection of graphical windows whose components are active, dynamic, clickable; and that is intended to explain just a few concepts [2]. Students can change any active element in the graphical interface and an immediate recalculation and dynamic presentation automatically starts. This way, students perceive instantly how their modifications affect the system behavior.

Interactive modules make possible an early introduction of concepts such as linearization, effects of non-linearities, integrator windup, effects of time delay, and others. At the same time, they allow for demonstration of design iterations interactively. For example, a pole on a lead–lag network may be dragged on a root locus plot using the mouse and the effect on the transient response displayed instantly. Hence, design iterations using classical methods (root locus, Bode) and modern methods (pole-placement, LQG, H_∞) may be carried out and demonstrated in an instructive fashion. The use of interactive and instructional graphic tools would reinforce active participation of students. For educators, interactive modules can provide a very useful way to testing control principle ideas and how difficult it is to get a particular control concept across to students.

Furthermore, the student has an excellent opportunity to "experiment" with his/her own ideas in terms of engineering design by simple interaction with the tool. Interactive modules cannot only be effective in presenting engineering concepts in the classroom but also beneficial in extending students experience in analysis and design assignments. This invitation of creativity can be most useful when it comes to specialized control engineering student projects.

Many good lessons can be extracted from similar experiences in other disciplines. An outstanding example comes from the field of Physics, namely *Physlets* [3,4]. Physlets (Physics applets) are simple (just dealing with one facet of a phenomenon), small (easy downloadable across the network), web-oriented (developed in Java), scriptable animations. They are also free for non-commercial use. Thanks to all these attributes, this collection of animations allows an easy integration of the computer into the classroom. Even better, the tool helps build a world-wide community of physics instructors generating free educational resources.

Obviously, it would be desirable to make a translation of the Physlets strategy to the control engineering educational community, but this community presents some peculiarities. As a consequence of the high impact of the control engineering in the industry, there exist many high-priced, commercial tools that are used to design and simulate control systems. In particular, the Matlab/Simulink environment has been considered a de facto standard for this community. During the last years, several free and worthy educational applications have been developed using this tool or, directly, generic programming languages (Java, VisualBasic, C++), with more or less use of interactivity and graphics [5–8]. The unique statement in which all control engineering instructors seem to agree is that the programming of instructional software using commercial tools is a very time-consuming activity, and high programming skills are required to develop helpful applications. This statement can be extended, even with more strength, to the development of virtual and remote labs.

In this paper, we introduce Easy Java Simulations (Ejs), a tool that can help address this problem. Originated also for the teaching of Physics, the tool is being adapted to serve the needs of control engineering education. Thus, the tool can be considered as a timely first step for the community in control education in the way of adopting a Physlets-like approach in order to be on the brink of the new pedagogical tendencies as Physics instructors are trying to do.

The article organizes as follows. We first discuss the use of a new approach in the design of control systems for instruction, that is, using interactivity as a tool to involve students in their learning process. Next, we provide a general introduction to Ejs. Finally, we discuss the features of Ejs that can be of special interest for control engineering educators. In particular, we describe in full detail how Ejs works in connection with Matlab and Simulink, illustrating the procedure with the help of two concrete examples.

The Interactive Approach in Control Engineering

According to Skogestad and Postlethwaite [9], a control designer (teacher, engineer, or student) must fulfill the following procedure in order to design a suitable control system for a plant:

1. Study the plant to be controlled and obtain initial information about the control objectives.
2. Model the plant and simplify the model, if necessary.

3. Analyze the resulting model; determine its properties.
4. Decide which variables are to be controlled (controlled outputs).
5. Select the control configuration.
6. Decide on the type of controller to be used.
7. Decide on performance specifications, based on the overall control objectives.
8. Design a controller.
9. Analyze and if the specifications are not satisfied, modify it or the type of controller.
10. Simulate the resulting controlled system, either on a computer or pilot plant
11. Repeat from step 2, if necessary.
12. Choose hardware and software and implement the controller.
13. Test and validate the control system, and tune the controller on line, if necessary.

It can be observed that if the design of the control system leads to unsatisfactory results of the simulations (step 10), we would be forced to go back to step 2 and proceed to repeat steps 2 to 10, analyzing them one-by-one. The approach of Skogestad and Postlethwaite to design a control system using traditional tools can be considered a ''non-interactive approach.'' This is due to the fact that it is not possible to know the consequences of the decisions taken during the design process until we are very near the end (step 11), just before the final hardware implementations. This situation produces that the same steps must be repeated over and over in order to correctly tune the design parameters, making this a very time-consuming activity. This is a direct consequence of the non-existence of a real-time link between the design and the analysis phases, preventing the designer from appreciating the gradient of change in the performance criteria given for the elements he/she is manipulating.

If there were such a real-time connection or link between the decisions taken during the design phase and the results obtained in the analysis phase, both phases could be merged in one. In this new approach, decisions taken in some steps would show on-the-fly the differences between the simulation results and the original control specifications fixed in step 1. It would not be necessary to iterate so many times across every step trying to tune the parameters.

But this proposed link to merge analysis and design can be brought to existence if we give interactivity the central role that it deserves in the development of new software tools for control education. Nowadays, the high performance of current computers, the advanced object-based programming tools and methodologies, and the staggering computer graphic capabilities allow us to design control applications in which any modification of the system parameters has a real-time translation in the graphical visualization of the system response (Fig. 1). This new approach is known as *interactive design* [10].

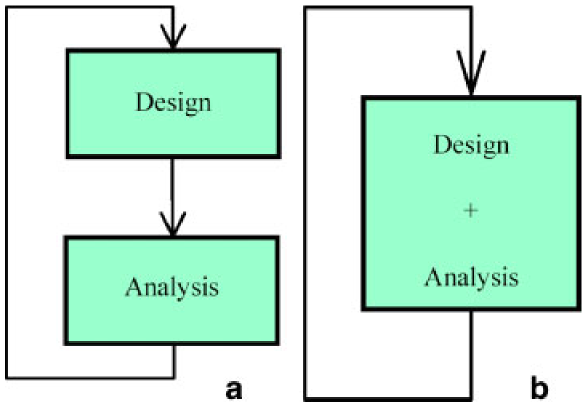


Figure 1 Outlines of the non-interactive (a) and interactive (b) approaches. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

In such a high-interactive system, several graphic windows are updated immediately, reflecting the value of every active element, and the constraints among them. This lets us establish a reactive behavior among all active components of the system, hiding the underlying mathematical basis. Every change in the values of system parameters has a dynamic repercussion in the system response. Since a good design usually involves multiple control objectives using different representations (time domain or frequency domain), it is possible to display several graphic windows that can be updated simultaneously during the manipulation of the active elements.

Furthermore, interactive design with instantaneous performance display let us go beyond. In many cases, it is not only possible to calculate the position of a graphic element (be it a curve, a pole, or a template) from the model, controller or specifications, but also calculate a new controller from the position of the element. For instance, a closed loop pole can be computed by calculating the roots of the characteristic polynomial, itself based on the model and controller; and the controller parameters can be synthesized from the set of closed loop poles if some conditions on the degrees are fulfilled [11].

Consequently, the control designer could appreciate the impact of his/her actions in the system's behavior in a glimpse and from different perspectives. The designer can identify the bottlenecks of the designs in a very easy way and can attempt to work them out. But, what is probably even more important, he/she cannot only identify the effect of the manipulation of a design parameter displayed, but its direction and amplitude also become apparent. The gradient of the change flashes and opens the mind of the designer.

Ictools and CCSdemo [12,13], developed at the Department of Automatic Control at Lund Institute of Technology, and SysQuake at the Institut d'Automátique of the Federal Polytechnic School of Lausanne [14,15] are good examples of this new educational philosophy for teaching automatic control. Very significant didactic interactive modules developed using SysQuake can be found in References [2,16–18]. Figure 2 shows an example of how this new way of interactive control education provides practical insight into control systems fundamentals, in this case, to explain the "aliasing" phenomenon. This is an interactive module in the sense mentioned earlier, and when the student manipulates some active element in the Figure, the new result is automatically produced.

All these previous examples are mainly intended for the explanations of mathematical concepts in control theory. They help students figure out the relationships and constraints among different parameters of a system which is, in many cases, rather difficult to visualize just observing the equations in the book. Another type of interactivity-based control applications have raised during the last years, allowing a more qualitative approach to the control of different plants (Fig. 3 shows an example of this). These applications focus on the user interaction with realistic models of different systems with either many control strategies or controllers. They let users know how the systems will react in certain situations, thus becoming very useful qualitative teaching/training systems for learning how to deal with certain cases. These environments help instructors bring the theoretical concepts (basically quantitative) to the practical world (more qualitative and intuitive), bridging the gap between both worlds. Examples of such interactive modules are described in References [6,19–22].

However, the construction of these virtual labs has to cope with the same drawbacks that the theory-focused interactive modules: programming skills and developing time, being the last a direct consequence of the first one. Usually, the control engineering teacher is a scholar in the control field, not in programming languages, methods, and techniques. He/she knows how to use different modeling and simulation environments, but these tools are not oriented

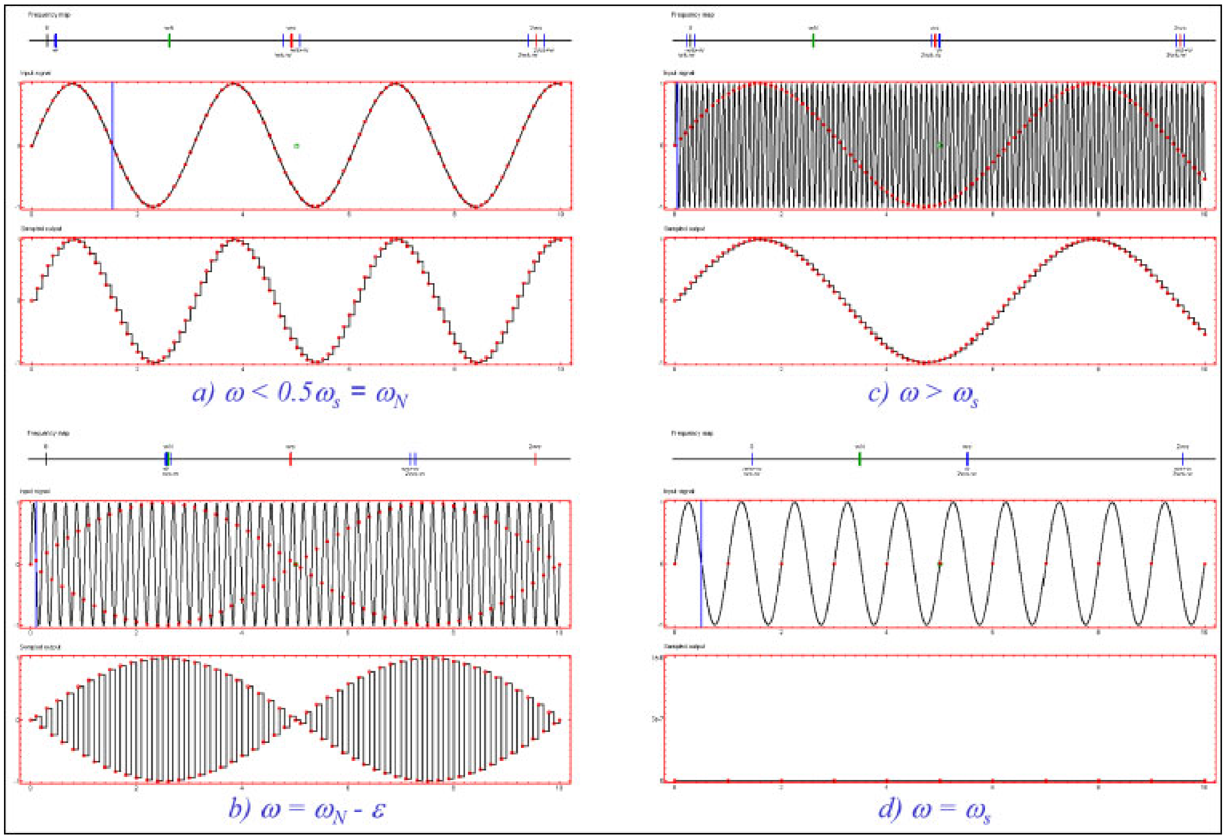


Figure 2 Interactive module to explain the "aliasing" [2]. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

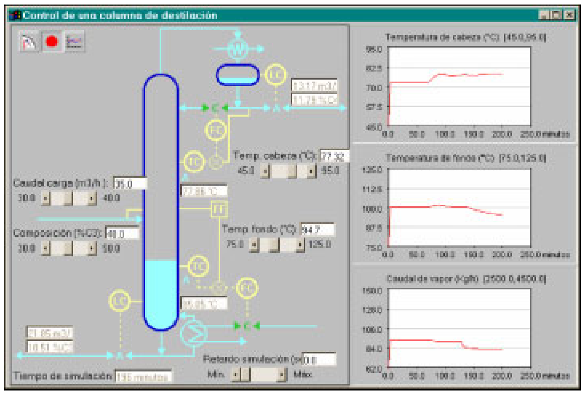

Control de una columna de destilación
Temperatura de cabeza (°C) [45.0,95.0]
Temperatura de fondo (°C) [75.0,125.0]
Caudal de vapor (Kg/h) [2500.0,4500.0]
Temp. cabeza (°C)
Temp. fondo (°C)
Caudal carga (m3/h)
Composición (%C3)
Tiempo de simulación
Periodo simulación (s)
Min.
Máx.
minutos

Figure 3 Advanced control of a distillation column using an interactive approach [21]. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

to the development of dynamic pictures from the perspective of the interactive design. To make an extensive use of interactivity with these tools requires a non-trivial effort or the programming task becomes very tricky and extensive. Therefore, it seems clear that the academic control community demand a tool to develop interactive modules: instructional simulations with low programming cost, interactivity as the main feature, and as low as possible an economic cost.

Why Is not Ejs Just Another Tool to Develop Interactive Modules?

Ejs is a software tool that helps create dynamic, interactive scientific simulations in Java language [23]. Ejs falls in the category of *code generators*. This means that users need to provide the most relevant core of the simulation's algorithm and the tool automatically generates a whole lot of extra code needed to create the complete interactive simulation. What makes Ejs special in this category is that it has been conceived *by* teachers and *for* teachers (and students) which are more interested in the understanding of the simulated phenomena that in the computer specific aspects.

The tool uses an original simplification of the successful model-view-control software paradigm, and structures simulations in two main parts: model and view. The model is what conveys the scientific part of the simulation and is thus the responsibility and the main interest of the target user. Most teachers are usually able (or can easily learn) to describe physical phenomena in terms of mathematical equations and to translate these into algorithms of a computer language, or by means of high-level tools such as Matlab/Simulink. The main objective when creating the model of a simulation is to concentrate in 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. This activity can lead to long searches in reference manuals looking for the right library and graphical routines. And, 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 big investment of time and effort (Fig. 4).

Ejs helps its users through both tasks. While retaining the flexibility of a general programming language, so that it is possible to specify almost any type of algorithms, the tool provides extensive scaffolding for the model. Thus, the user follows a guided sequence of steps in which he/she defines the model by

(a) declaring the variables that describe the system,
(b) initializing these variables to a correct initial state, and
(c) writing the algorithms that state how the value of these variables change in time or under user interaction.

This last step consists in translating the mathematical equations or physical laws that govern 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 conveniences. The first one is a built-in editor and solver for systems of ordinary differential equations (ODE). The user writes his/her equations in a way much similar to what he/she would write on a blackboard, and the system automatically

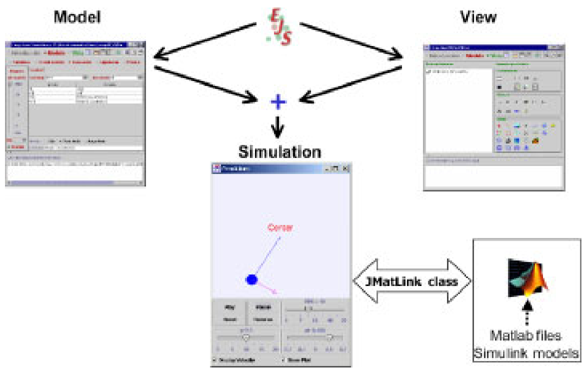


Figure 4 Building a simulation with Easy Java Simulations (Ejs). [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

generates the code that numerically solves the system using one of several of the provided standard algorithms. An illustrative interactive module on the quadruple-tanks process that uses this feature can be found in Reference [22]. The second facility is a connection to Matlab/Simulink that lets users specify and solve their models with the help of these tools. This option is described in detail in the next section.

To create the view, Ejs provides a set of advanced graphical elements (see Fig. 5), which 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 [24] (particles, vectors, images, vector and scalar fields . . .). These element 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 perspectives in order to gain a better understanding of 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 . . .). The user can then connect the properties of the graphical elements of the view to the value of the different model's variables. Some of these properties can also be instructed to

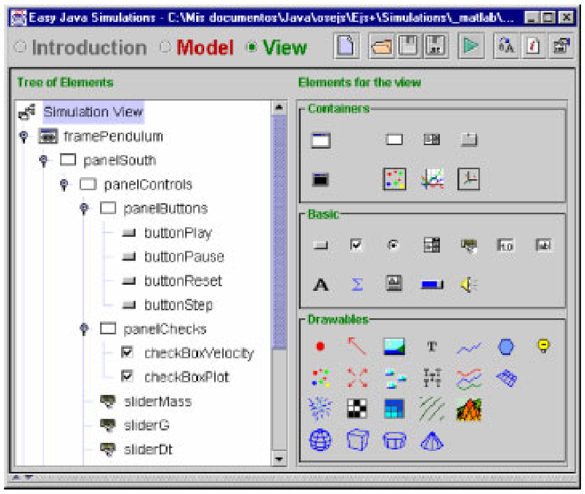


Figure 5 Library of graphical elements of Ejs. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

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 turn the simulation into a real dynamic, interactive application. This mechanism conforms a simple, though very effective, way of designing and 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 interaction of the student with 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 colored 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 dynamic of the system). Similarly, if the water level changes as consequence of the model's evolution, the visible height of the rectangle will change to show the current water level.

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 using 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 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.

Using Matlab/Simulink With Ejs: A First Example

We now turn to the description of one of the features of Ejs that can be of most interest to control engineering educators. As we said above, users can choose to specify and numerically solve their models using their own algorithms, with the help of Ejs built-in ODE editor and solvers, or (and this is the option we discuss now in detail) using Matlab/Simulink, if they have it installed in their computers. (Actually, Ejs allows the user to use the three of these options at the same time, taking care of all the internal calls in order to run and synchronize the simulation properly.) This means that a simulation created with Ejs can

(a) call any Matlab function (either built-in or defined in an M-file) at any point in the model and
(b) run external Simulink models.

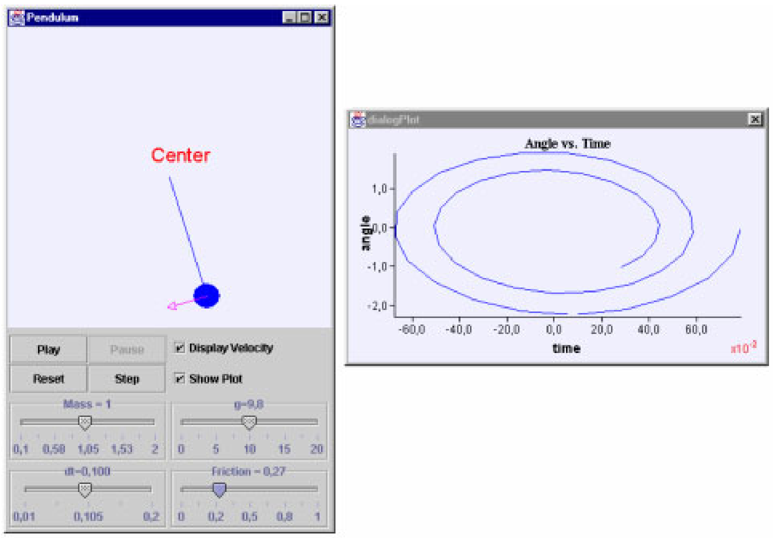
Developers can then insert Matlab code in an Ejs application to make calculations in the Matlab workspace. But also, the model of a system can be completely or partially developed using just Simulink block diagrams. In this last situation, Ejs can just be the tool that will help create the view of the simulation, i.e., the visualization and user interface for the Simulink model. This process has been made as simple as possible. Once a Simulink simulation has been selected in Ejs as part of the model, all the Simulink variables can be accessed from Ejs and be associated to regular variables. So, developers can link them to the graphical elements like regular Ejs variables. When running the simulation, Ejs will automatically take care of running the Simulink model and of the two-way exchange of information between the model and the view (on the side of Ejs) and the Matlab workspace.

Let us illustrate this procedure by means of an example taken from the list of demo applications included in Simulink standard distribution. It consists of an interactive simulation of a two-dimensional pendulum formed by a bob of fixed mass that is suspended from a rod (of negligible mass) under the action of gravity. When the bob is displaced from the stable equilibrium state (the lower vertical position with zero initial velocity), gravitational forces tend to recover the original position. In the absence of friction, the movement observed is called simple harmonic motion. The differential equation that model the simple pendulum is:

$$x'' = \frac{F - l \times c \times x' - m \times g \times \sin(x)}{m \times l} \tag{1}$$

Figure 6 shows the view of the application developed with Ejs using Matlab/Simulink as the simulation engine. This application allows users a total interactivity with the simulated pendulum. It can be easily appreciated that there are different elements that help change many of the parameters of the system (the mass of the bob, gravity, friction coefficient, integration step). In addition, the graphical representation of the pendulum is highly interactive as well. It is possible to modify the rod length, the initial angular position, and velocity just by clicking with the mouse the bob and the velocity vector and dragging them. In this way, every system parameter is under control. Users can use the mouse, as our hand does picking up a true pendulum's bob and dropping it, and create many different experiments changing gravity and friction and observing the results in the display and on the attached plot.

However, the most relevant features of this example is that Matlab/Simulink is running at the background. Figure 7a shows the original Simulink block diagram implementing the differential equation



Mass = 1
g=9,8
dt=0,100
Friction = 0,27

Figure 6 View of the interactive module developed by Ejs. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

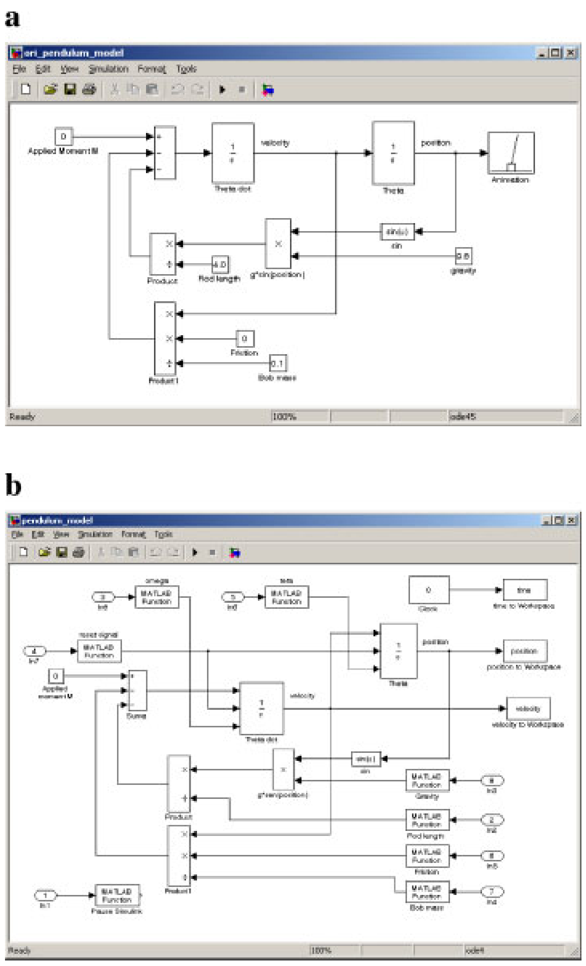


Figure 7 Original and modified Simulink block diagrams. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

above. Three steps are required in order to use an existing Simulink model within Ejs.

The first step consist in doing some changes to the original Simulink model. These changes are necessary because Simulink models are (obviously) created to be run within Simulink. But, since 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. The second change 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 each integration step. This way, Ejs can control exactly when the model needs to play. The final change (though not compulsory) is to remove from the Simulink block diagram any visualization of the state, since Ejs will most likely take care of all our visualization needs. Figure 7b shows the resulting block diagram, after we modified it to run in connection to Ejs.

It can be seen in Figure 7 that system parameters have been adapted to be read from Matlab's workspace: rod length, bob mass, gravity, friction, initial angular position, and velocity, integration step. Also, the block diagram has been slightly modified so that it will write the simulation state in the Matlab's workspace (time, position, and velocity) after each iteration.

The second step that we need to connect the Simulink model to Ejs is the creation of a simple Matlab M-file. This M-file provides Ejs with basic information about the model, and the variables and parameters that can be accessed across the Matlab's workspace. Figure 8 shows the M-file that we created for this example.

The syntax for this text file is rather easy. The value of the variables has to be defined and initialized in Matlab language, that is, as if this file were to be used as a typical Matlab M-file. (Actually, the file *will be* executed before playing the Simulink model.) However, what enables Ejs to read and access these variables is the special comment at the end of each line. This comment starts with the keyword "Ejs" immediately after the Matlab begin-of-comment char-

```
teta = pi/4;                  % Initial condition for ang. pos.     %Ejs Variable InputOnly
position = teta;              % Angular position                    %Ejs Variable OutputOnly
omega = 0.0;                  % Initial condition for velocity      %Ejs Variable InputOnly
velocity = omega;           % Angular velocity                      %Ejs Variable OutputOnly
mass   = 0.1;                 % Mass of the bob                     %Ejs Variable InputOnly
length1 = 4.0;                % Length of the rod                   %Ejs Variable InputOnly
gravity = 9.8;                % Gravity force                       %Ejs Variable InputOnly
friction = 0.0;               % Friction coefficient                %Ejs Variable InputOnly
time= 0;                      % Current time                        %Ejs Variable OutputOnly
ic = 0;                       % Reset for initial conditions        %Ejs Variable InputOnly
dt= 0.01;                     % Integration step                    %Ejs Parameter=maxstep InputOnly
model='pendulum_model.mdl';   % Simulink file                       %Ejs Model
```

Figure 8 M-file *pendulum.m* to connect Simulink and Ejs.

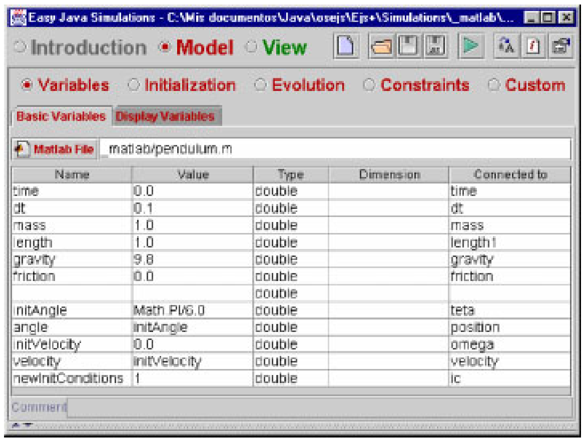
acter "%." The keyword must be followed by one of the following words: Model, Variable, or Parameter.

- Model is used to tell Ejs the name of the Simulink file that describes the model.
- Variable provides Ejs with the 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 take care of synchronizing its value with the associated Ejs variable before and after the Simulink model runs. 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, gravity and the initial conditions are input-only variables that will be fixed by the user though the simulation interface. Finally, if declared as OutputOnly, Ejs will read the variable after the Simulink model is run, but will not try to change its value in the Simulink model. For instance, the three state variables of the model are read-only since they are 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 uses. However, for technical reasons, the comment for a parameter must include information about the block or menu in which the parameter is defined and used and the function of this parameter within the block. In our example, the integration step d*t* 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 Simulink menu bar.

The third and final step consists in connecting the variables in Ejs model with the Matlab's workspace variables (that are in turn associated to variables of the Simulink model). Figure 9 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.

Our Simulink model is now ready to be used. Ejs will run the Simulink model whenever we insert the Java sentence matlab.play ("_matlab/pendulum.m"); in any suitable place, most likely in a page of the evolution (Ejs' main loop) section of Ejs. A call to this method has the following effects:

1. The value of all Ejs variables, which are connected to a Matlab variable (except those declared as OutputOnly) is pushed to the Simulink model.



_matlab/pendulum.m

Name	Value	Type	Dimension	Connected to
time	0.0	double		time
dt	0.1	double		dt
mass	1.0	double		mass
length	1.0	double		length1
gravity	9.8	double		gravity
friction	0.0	double		friction
		double		
initAngle	Math.PI/6.0	double		teta
angle	initAngle	double		position
initVelocity	0.0	double		omega
velocity	initVelocity	double		velocity
newInitConditions	1	double		ic

Figure 9 Ejs table of variables showing the connection to Matlab's variables. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

2. The Simulink model is run exactly once.
3. The value of all Matlab variables which 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 link to it the variables of the model in a form that suits our pedagogical needs. Figure 10a shows the tree-like of view elements that we chose for our view. Figure 10b displays the table of properties for the graphical element that represents the pendulum's bob.

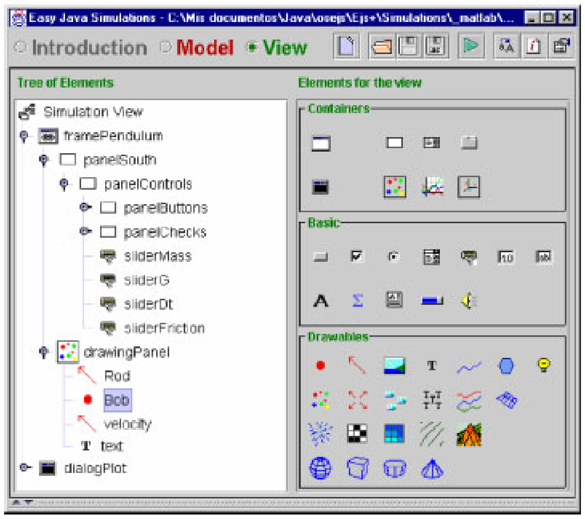
Finally, it is worth noting that Ejs is capable of running more than one Simulink model from within one single simulation. This is simply done by creating two or more separate pages of variables, each with a suitable M-file, and connecting the Ejs variables to the corresponding Matlab variables. When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

A more detailed explanation of the development of Ejs applications using Matlab/Simulink can be found in Ejs documentation. This example of the simple pendulum and more are included in Ejs standard distribution available at http://fem.um.es/Ejs/.

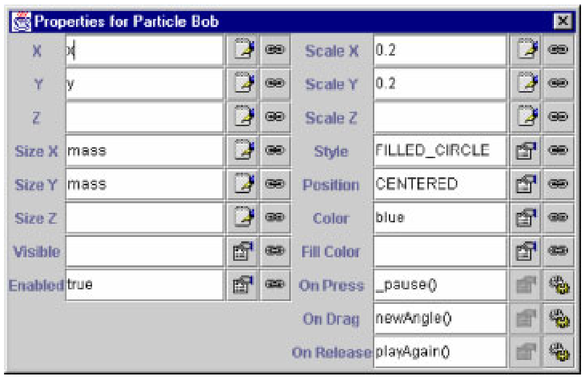
A Second Example: A MIMO System

In this second example, we use Ejs to build a simulation that explains the use of SISO techniques in MIMO control. The example covers the use of completely decentralized control as a mechanism for

a



b


Properties for Particle Bob

X	x	Scale X	0.2
Y	y	Scale Y	0.2
Z		Scale Z	
Size X	mass	Style	FILLED_CIRCLE
Size Y	mass	Position	CENTERED
Size Z		Color	blue
Visible		Fill Color	
Enabled	true	On Press	_pause()
		On Drag	newAngle()
		On Release	playAgain()

Figure 10 Elements for the view of the pendulum. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

directly exploiting SISO methods in a MIMO setting. This approach considers that interactions can be treated as a form of disturbances. For example, if the first output, $y_1(t)$ of a $m \times m$ MIMO plant is described as:

$$Y_1(s) = G_{11}(s)U_1(s) + \sum_{i=2}^{m} G_{1i}(s)U_i(s) \qquad (2)$$

then the contribution of each input, other than u_1, could be considered as an output disturbance $G_{1i}(s)U(s)$ for the first SISO loop. Whereas this approach is incorrect from a formal point of view, under certain conditions it can produce acceptable results. In the example, we use a conventional multi-loop control scheme consisting of a 2×2 process and two feedback controllers [25]. Four transfer functions of the two-input, two-output, plant are necessary to completely characterize the process dynamics:

$$G(s) = \begin{bmatrix} G_{11}(s) & G_{12}(s) \\ G_{21}(s) & G_{22}(s) \end{bmatrix} \qquad (3)$$

$$G_{11}(s) = \frac{2}{s^2+3s+2} \quad G_{12}(s) = \frac{k_{12}}{s+1} \qquad (4)$$

$$G_{21}(s) = \frac{k_{21}}{s^2+2s+1} \quad G_{22}(s) = \frac{6}{s^2+5s+6} \qquad (5)$$

where the coefficients k_{12} and k_{21} depend on the operating point, and can determine the stability of every individual control loop when the other loop is opened. The transfer functions of the two feedback controllers $C_1(s)$ and $C_2(s)$ are given by:

$$C_1(s) = \frac{4.5(s^2+3s+2)}{s(s+4)} \quad C_2(s) = \frac{1.5(s^2+5s+5)}{s(s+4)} \qquad (6)$$

Another aspect to be taken into account in a decentralized approach is the important practical problem of how the controlled variables Y_i and the manipulated variables U_i should be paired in the multiloop control scheme. A correct pairing is crucial since an incorrect configuration can result in poor control system performance and reduced stability margins. In the 2×2 control system example, there are two possible control configurations. In the first one, Y_1 is controlled by adjusting U_1, while U_2 is used to control Y_2. This configuration is referred as the 1-1/2-2 control scheme. The alternative strategy is to pair Y_1 with U_2 and Y_2 with U_1, configuration termed as 1-2/2-1.

The method to be used to suggest pairings is the relative gain array (RGA) developed by Bristol [26]. This approach requires only steady-state information and provides two important items of information: a measure of process interactions, and a recommendation concerning the most effective pairing of controlled and manipulated variables. A detailed explanation on the calculation of the RGA can be found in Reference [27]. In our example, and considering the system presented in Equation 2, the RGA obtained is:

$$\Lambda = \begin{bmatrix} \lambda & 1-\lambda \\ 1-\lambda & \lambda \end{bmatrix} = \begin{bmatrix} \dfrac{1}{1-k_{12}k_{21}} & \dfrac{-k_{12}k_{21}}{1-k_{12}k_{21}} \\ \dfrac{-k_{12}k_{21}}{1-k_{12}k_{21}} & \dfrac{1}{1-k_{12}k_{21}} \end{bmatrix} \qquad (7)$$

In particular, one usually aims to choose pairings such that the diagonal entries of Λ are large. One also tries to avoid pairings that result in negative diagonal entries in Λ. In our MIMO plant, for $1 > k_{12} > 0$, $1 > k_{21} > 0$, the RGA suggests that the 1-1/2-2 control scheme must be used.

To gain further insight into the stability problem, it is necessary to calculate the characteristic equation of both control schemes: 1-1/2-2 and 1-2/2-1. The transfer function of closed-loop system can be organized in matrix form as:

$$\begin{bmatrix} Y_1(s) \\ Y_2(s) \end{bmatrix} = (\mathrm{I} + \mathrm{G}(s)\mathrm{C}(s))^{-1}\mathrm{G}(s)\mathrm{C}(s)\begin{bmatrix} R_1(s) \\ R_2(s) \end{bmatrix}$$
$$\mathrm{C}(s) = \begin{bmatrix} C_1(s) & 0 \\ 0 & C_2(s) \end{bmatrix} \tag{8}$$

where $G(s)$ changes according to the pairing. Thus, for the 1-1/2-2 configuration $G(s)$ is the same as in Equation 3, but in the 1-2/2-1 control scheme $G(s)$ is:

$$\mathrm{G}(s) = \begin{bmatrix} G_{12}(s) & G_{11}(s) \\ G_{22}(s) & G_{21}(s) \end{bmatrix} \tag{9}$$

Hence, the characteristic polynomial d(s) of both configurations are:

$$\begin{aligned} \mathrm{d}_{1-1/2-2}(s) = {} & s^6+10s^5+51s^4+(148 - 6.75k_{12}k_{21})s^3 \\ & + (259 - 47.25k_{12}k_{21})s^2 \\ & + (234 - 108k_{12}k_{21})s + 81(1 - k_{12}k_{21}) \end{aligned} \tag{10}$$

$$\begin{aligned} \mathrm{d}_{1-2/2-1}(s) = {} & s^6 + (4.5\mathrm{k}_{12}+10)\mathrm{s}^5 \\ & + (36k_{12}+1.5k_{21}+ 8.25)s^4 \\ & + (6.75k_{12}(k_{21}+3.5) \\ & + 0.5(6.75k_{21}+20))s^3 \\ & + (2.25k_{12}(5.25 \times k_{21}+11) \\ & + 9.75k_{21} - 16.25)s^2 \\ & + (27k_{12}k_{21}+9k_{12}+9k_{21} - 40.5)s \\ & + 81k_{12}k_{21} - 81 \end{aligned} \tag{11}$$

where k_{12} and k_{21} determine the stability of the closed-loop system. In this way, it is possible to establish stability regions for specified values of k_{12} and k_{21}. Clearly, the development of an interactive application using this MIMO plant (or others) with two SISO controllers will allow to change the operating points on-the-fly and measure the process interactions between loops and how sensible is to modify the pairing.

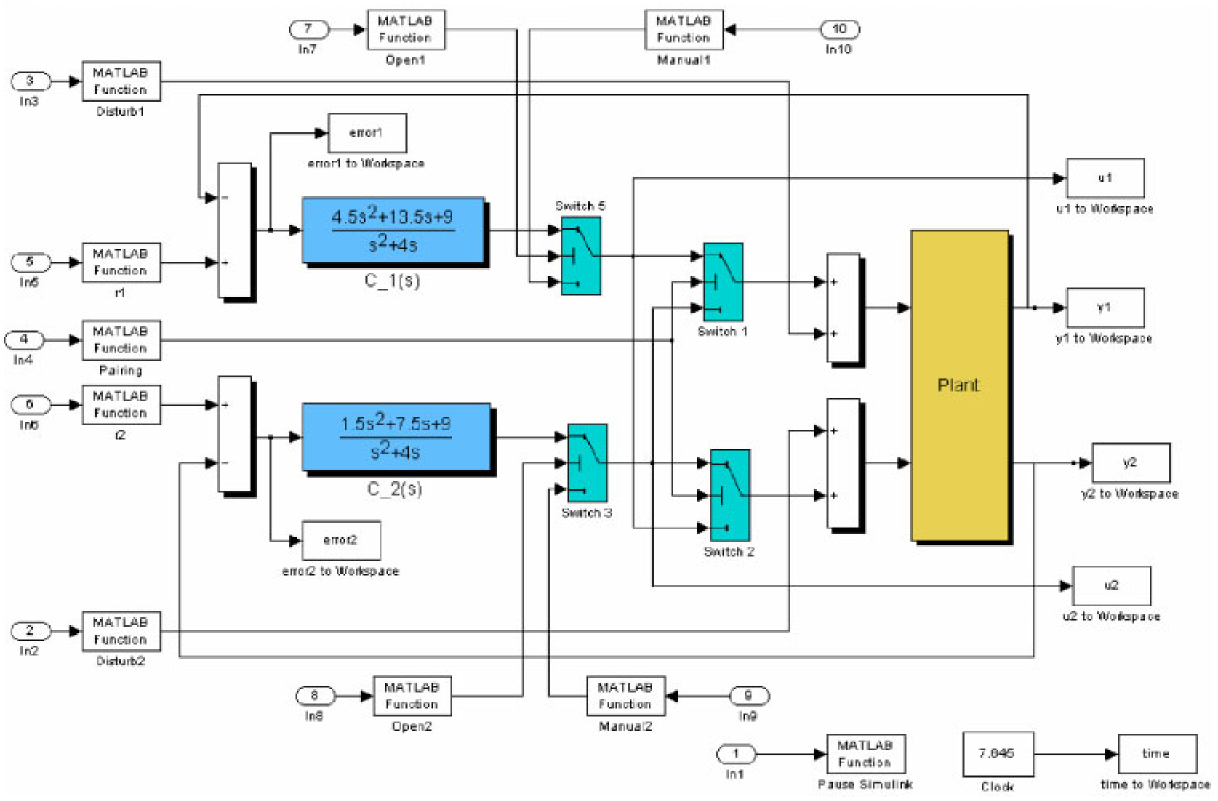
After this brief on the theoretical background of the application, the first step to start the design of our Ejs application using Simulink is to create the block diagrams for the system and the plant. These are shown in Figure 11.

It can be appreciated in Figure 11a that the block diagram is already adapted to be controlled from Ejs. The variables to be read from the workspace are the references (variables r_1 and r_2) and the disturbances (variables *disturb1* and *disturb2*). The information sent back to the workspace after each integration step are the plant outputs (variables y_1 and y_2), the two control actions (variables u_1 and u_2), the two error signals (variables *error1* and *error2*), and the time. Notice that the block diagram is adapted to pair inputs and outputs by means of four switch blocks controlled by a variable named *pairing*. Therefore, modifying the value of this variable from Ejs, and hence in the workspace, the structure of the system is changed, and so, the pairing of inputs and outputs is made. The block diagram has also been modified to open or close the loops from the workspace by changing the value of the *open1* and *open2* variables.

It is important to notice in Figure 11b that the coefficients k_{12} and k_{21} are the numerators in the G_{12} and G_{21} blocks, respectively. Figure 12 shows the Matlab file needed to tell Ejs how to interact with the Simulink model through Matlab's workspace. Please notice how the comments on k_{12} and k_{21} tell Ejs their precise locations and functions in the diagram.

The view of the final Ejs application is shown in Figure 13. It is composed of several windows to get a complete control of the decentralized control system. The main window (upper-left corner) allows users to play the simulation, to change the value of some variables, to open/close information panels, to choose the type of control in each loop (automatic or manual), and to pair inputs/outputs according to the information of the RGA panel (lower-right corner). The four signal scopes of the plot panel (lower-left corner) let users appreciate the effects of interactions in the two control loops with the SISO design. The relevant detail of the control parameter panel is the possibility of changing interactively the values of k_{12} and k_{21} just by dragging the red dot in the diagram with the mouse. Obviously, any change either the k's or the pairing will produce changes in the stability of the system, and that will be shown in the characteristic polynomial roots panel of the closed-loop system. Thus, all these panels visualize on-the-fly how sensible it is to pair the inputs with the outputs in function of the k's values and under which conditions the closed-loop system is stable.

In addition, the application allows to study some robustness issues in decentralized control using a diagonal nominal model. In this case, Matlab is instructed from the application to draw the picture of the a



b

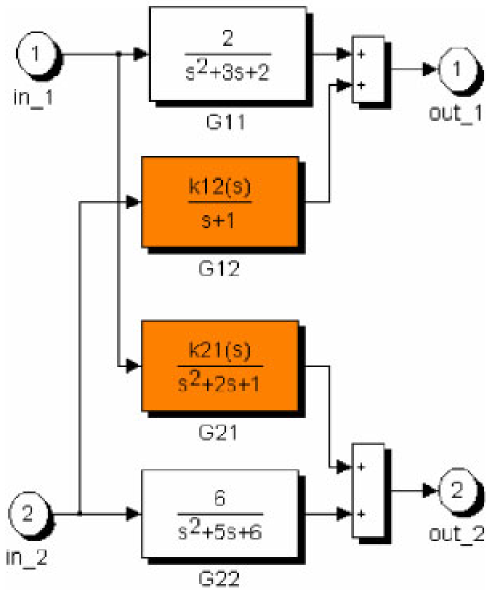


Figure 11 Simulink model of the decentralized control architecture (a) and MIMO plant (b). [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

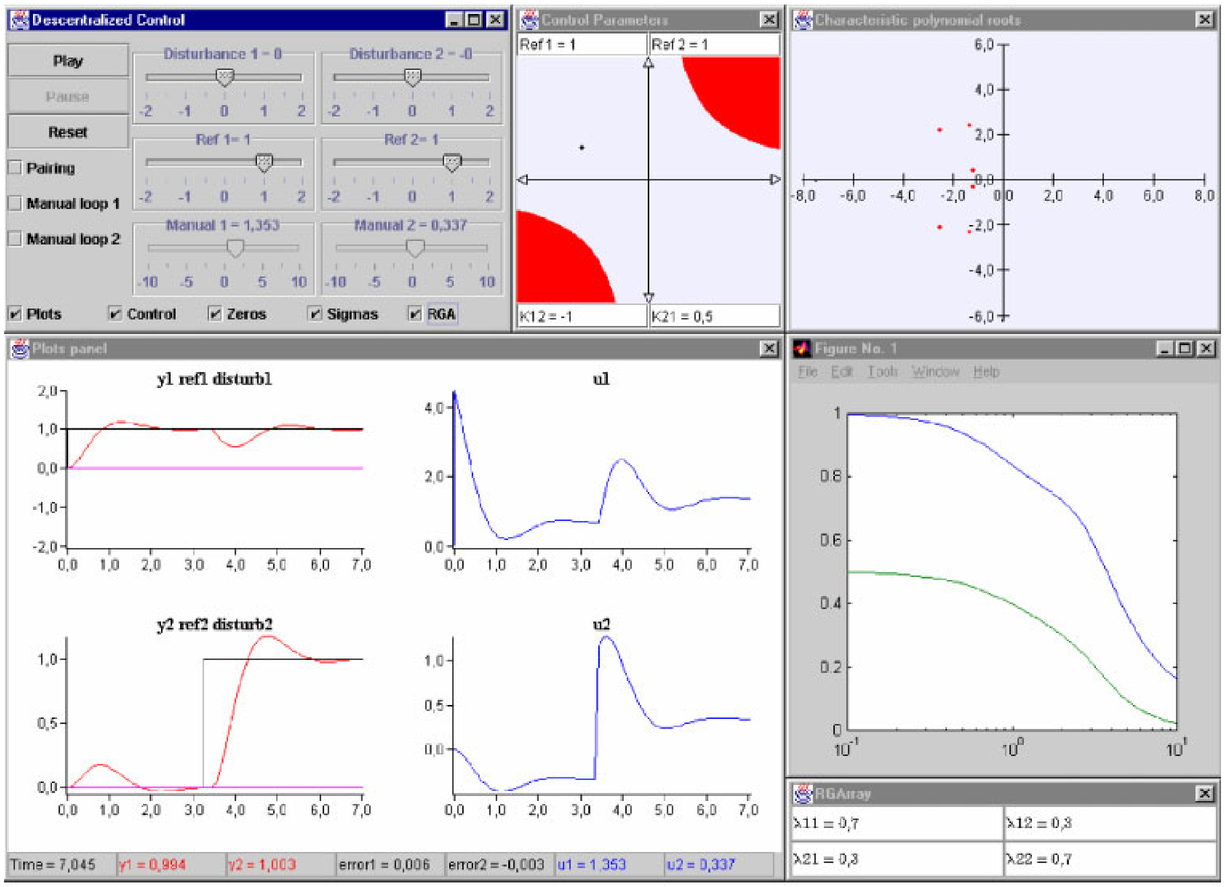
```
model = 'mimo_model.mdl';   %Ejs Model
u1 = 0;                     %Ejs Variable OutputOnly
u2 = 0;                     %Ejs Variable OutputOnly
y1 = 0;                     %Ejs Variable OutputOnly
y2 = 0;                     %Ejs Variable OutputOnly
error1 = 0;                 %Ejs Variable OutputOnly
error2 = 0;                 %Ejs Variable OutputOnly
time= 0;                    %Ejs Variable OutputOnly
ref1= 0.0;                  %Ejs Variable InputOnly
ref2= 0.0;                  %Ejs Variable InputOnly
pairing= 1.0;               %Ejs Variable InputOnly
open1 = 1.0;                %Ejs Variable InputOnly
open2 = 1.0;                %Ejs Variable InputOnly
manual1 = 0.0;              %Ejs Variable InputOnly
manual2 = 0.0;              %Ejs Variable InputOnly
disturb1 = 0.0;             %Ejs Variable InputOnly
disturb2 = 0.0;             %Ejs Variable InputOnly
sigmas_view = 0.0;          %Ejs Variable InputOnly
k12 = 0.1;                  %Ejs Parameter=Plant/G12:Numerator InputOnly
k21 = 0.1;                  %Ejs Parameter=Plant/G21:Numerator InputOnly
```

Figure 12 M-file *mimo.m* to connect Simulink and Ejs.

singular values of the system (middle-right corner in Fig. 13). This is possible because the view has been designed so that it will call a user-defined method every time the *k*'s values are changed interactively. This user-defined method includes a Java statement of the form _matlab.eval(mFile); that evaluates an M-file that contains all the Matlab sentences required to plot the singular value response of a continuous-time system. In this simple way, all of Matlab's powerful graphic capabilities can also be used interactively from within simulations created with Ejs.

CONCLUSIONS

Examining the state of the art in interactivity-based didactic applications for control engineering it is clear that they are a step behind other scientific fields, as for example, Physics. The control engineering field is, however, a discipline very well suited for the use of interactivity as a way to reinforce the study of many theoretical concepts. But, until now, the biggest part of computer-based education on control has been conducted using basic theoretical tutorials, most of them featured for the static of their contents.



Disturbance 1 = 0 | Disturbance 2 = -0 | Ref 1= 1 | Ref 2= 1 | Manual 1 = 1,353 | Manual 2 = 0,337

☐ Pairing ☐ Manual loop 1 ☐ Manual loop 2

☑ Plots ☑ Control ☑ Zeros ☑ Sigmas ☑ RGA

Ref 1 = 1 | Ref 2 = 1 | K12 = -1 | K21 = 0,5

Time = 7,045 | y1 = 0,994 | y2 = 1,003 | error1 = 0,006 | error2 = -0,003 | u1 = 1,353 | u2 = 0,337

λ11 = 0,7	λ12 = 0,3
λ21 = 0,3	λ22 = 0,7

Figure 13 Full view of the Ejs application. [Color figure can be viewed in the online issue, which is available at www.interscience.wiley.com.]

In control engineering, the relationships among the variables of a system are accurately and clearly defined by mathematical language, but its visualization is complicated for students. This is not a simple mental exercise. It is true that there are many mathematical packages to program these relationships and thus to observe the results of changing some parameters of the system, but these tools are mainly oriented towards a non-interactive approach composed of two stages: design and analysis. That is, students have to afford many iterations with the tool to appreciate the level and sense of the change in some parameters. Moreover, such a static approach does not always facilitates the global view of the relationships among the variables of the system. The amount of data that a static approach can produce and the number of iterations needed to obtain them make it difficult to students to build a mental scheme of the interactions among the components of the system. In these cases, a qualitative and conceptual view of the system can be a better approach to lead students to a deeper understanding of the system.

However, the control engineering teacher is an expert in his/her subject but not in programming techniques. The development of interactive didactic applications is not so easy with the typical tools that a control engineering teacher uses to explain concepts, as for example, Matlab/Simulink. At the same time, there are many commercial tools designed to create animations and multimedia but, in general, scientific simulations cannot be created with them because they make a poor or inexistent use of the mathematical language to define the system to model.

Ejs is a tool that can be the first in the colonization of this no-man's land. It should be considered as a first step in the development of further and more advanced tools intended for control engineering educators with some knowledge of programming, not for advanced programmers with little knowledge of control and modeling. Ejs has been conceived for people who are more interested in the content of the simulation, the simulated phenomenon itself, and much less in the technical aspects needed to build the simulation. Hence, Ejs provides a conceptual structure and simplified tools that allow concentrate most of the time in the description of the model of the system to simulate. To help assist educators in the field of control-engineering, Ejs has been extended so that it can be used in conjunction with Matlab/Simulink, a de facto standard in control engineering. Inspired in the model-view-control software paradigm, Ejs facilitates the creation of the user interface, and Matlab/Simulink let teachers develop the model using the language they are familiar with.

After examining the state of the art in other disciplines, it is clear that an exciting task must be accomplished to improve the computer learning experience in the control engineering curriculum. New interactive modules for concepts in control systems have to be developed, and new tools must arise. From the experience obtained with the creation of the first such ones, new ideas to design the second generation must flourish.

REFERENCES

[1] B. S. Heck (Ed.), Special report: Future directions in control education, IEEE Control Systems Mag 19 (1999), 35−58.

[2] S. Dormido, The role of interactivity in control learning [Plenary lecture], Proceedings of the 6th IFAC Symposium on Advances in Control Education, Oulu, Finland, 2003, pp 11−22.

[3] W. Christian and M. Belloni, Physlets: Teaching physics with interactive curricular material, Prentice Hall, Upper Saddle River, NJ, 2001. See also http://webphysics.davison.edu/applets/applets.html.

[4] W. Christian and M. Belloni, Physlet® physics: Interactive illustrations, explorations, and problems for introductory physics, Upper Saddle River, NJ, 2004.

[5] R. C. Garcia and B. S. Heck, Enhancing classical control education via interactive GUI design, IEEE Control Systems Mag 19 (1999), 77−82.

[6] D. Cooper and D. Fina, Training simulators enhance process control education, Proceedings of the American Control Conference, San Diego, CA, 1999, pp 997−1001.

[7] I. Benyo, G. Lipovszki, and J. Kovács, Advanced control: Simulation tools in labview environment, Proceedings of the 6th IFAC Symposium on Advances in Control Education, Oulu, Finland, 2003, pp 275−279.

[8] J. P. Keller, Interactive control system design, Proceedings of the 6th IFAC Symposium on Advances in Control Education, Oulu, Finland, 2003, pp 365−370.

[9] S. Skogestad and I. Postlethwaite, Multivariable feedback control: Analysis and design, Wiley, New York, 1996.

[10] S. Dormido, Control learning: Present and future, Annu Rev Control 28 (2004), 115−136.

[11] S. G. Crutchfield and W. J. Rugh, Interactive learning for signal, systems, and control, IEEE Control Systems Mag 18 (1998), 88−91.

[12] M. Johansson, M. Gäfvert, and K. J. Åström, Interactive tools for education in automatic control, IEEE Control Systems Mag 18 (1998), 33−40.

[13] B. Wittenmark, H. Häglund, and M. Johansson, Dynamic pictures and interactive learning, IEEE Control Systems Mag 18 (1998), 26−32.

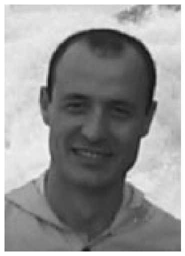
[14] Y. Piguet, SysQuake: User Manual, Calerga, 1999.

[15] Y. Piguet, U. Holmberg, and R. Longchamp, Instantaneous performance visualization for graphical control design methods, Proceedings of the 14th IFAC World Congress, Beijing, China, 1999.
[16] S. Dormido, M. Berenguel, S. Dormido-Canto, and P. Rodríguez, Interactive learning of constrained generalized predictive control, Proceeding of the 6th IFAC Symposium on Advances in Control Education, Oulu, Finland, 2003, pp 201–206.
[17] N. Tan, D. P. Atherton, and S. Dormido, Systems with variable parameters: Classical control extensions for undergraduates, Proceedings of the 6th IFAC Symposium on Advances in Control Education, Oulu, Finland, 2003, pp 287–292.
[18] J. Salt, P. Albertos, S. Dormido, and A. Cuenca, An interactive simulation tool for the study of multirate sampled data system, Proceedings of the 6th IFAC Symposium on Advances in Control Education, Oulu, Finland, 2003, pp 45–50.
[19] Y. Piguet and D. Gillet, Java-based remote experimentation for control algorithms prototyping, Proceedings of the American Control Conference, San Diego, CA, 1999, pp 1465–1469.
[20] C. Schmid, A remote laboratory using virtual reality on the web, Simulation 73 (1999), 13–21.
[21] J. Sánchez, F. Morilla, S. Dormido, J. Aranda, and P. Ruipérez, Virtual control lab using Java and Matlab: A qualitative approach, IEEE Control Systems Mag 22 (2002), 8–20.
[22] S. Dormido and F. Esquembre, The quadruple-tank process: An interactive tool for control education, Proceedings of the European Control Conference, Cambridge, UK, 2003.
[23] E. Esquembre, Easy Java Simulations: A software tool to create scientific simulations in Java, Com Phys Comm 156 (2004), 199–204. See also http://fem.um.es/Ejs.
[24] W. Christian, The open source physics project. http://www.opensourcephysics.org.
[25] C. G. Goodwin, S. F. Graebe, and M. E. Salgado, Control system design, Prentice Hall, Upper Saddle River, NJ, 2001.
[26] E. H. Bristol, On a new measure of interactions for multivariable process control, IEEE Trans Automatic Control 11 (1966), 133–134.
[27] D. E. Sebor, T. F. Edgar, and D. A. Mellichamp, Process dynamics and control, Wiley, New York, 1988.

BIOGRAPHIES

José Sánchez Moreno received his computer sciences degree in 1994 from Madrid Polytechnic University and his PhD in sciences from UNED (Universidad Nacional de Educación a Distancia, Spain) in 2001. Since 1993, he has been working at UNED Department of Computer Sciences and Automatic Control as an assistant professor. His current research interests are the design of new systems for control education, virtual labs, telepresence, and the use of the internet in education.

Sebastián Dormido holds a degree in physics from the Complutense University in Madrid, Spain (1968), and a PhD from University of The Basque Country, Spain (1971). In 1981, he was appointed professor of control engineering at the Universidad Nacional de Educación a Distancia. His scientific activities include computer control of industrial processes, model-based predictive control, robust control, and model and simulation of continuous processes. He has authored and co-authored more than 150 technical papers in international journals and conferences. Since 2002, he has been president of the Spanish Association of Automatic Control CEA-IFAC, where he promotes academic and industrial relations.



Francisco Esquembre received the PhD degree in mathematics in June 1991 from the University of Murcia, Spain, where he has worked since 1986, holding a permanent job as assistant professor since 1994. His academic expertise includes differential equations, dynamical systems, and numerical analysis. He currently teaches at the University of Murcia, and his research includes computer assisted teaching and learning as well as simulation of scientific processes for didactical purposes.