Web-based object-oriented control system design

J. C. MARTÍNEZ-GARCÍA, G. H. SALÁZAR-SILVA, R. GARRIDO

Departamento de Control Automático/Centro de Investigación y de Estudios Avanzados del IPN
A.P. 14-740, 07300 México D.F., México.
E-mail: {martinez,gaston,garrido}@ctrl.cinvestav.mx

Abstract— **We present in this paper some ideas concerning object-oriented control systems design. We propose in particular a Java-based methodology to develop control systems simulations on the World Wide Web, and we illustrate it with an application concerning the simulation of a two degrees-of-freedom robot manipulator.**

Keywords— **Object-oriented control system design, World Wide Web, Java applications, computer assisted control systems education.**

I. Introduction

In this paper, we propose an object-oriented methodology for control systems design, following the discussion started in [15]. We focus our proposal to the application of the Internet World Wide Web for Automatic Control purposes. The Java application which implements our methodology is proposed as a *free* alternative to the commercial tools which perform similar tasks.

In Section II we briefly discuss the basic concepts concerning object-oriented programming and we discuss the main characteristics of the Java programming language. We also discuss in this section how the classic block diagram paradigm, in the automatic control field, can be easily translated in an object-oriented methodology for control systems design.

In Section III we illustrate the proposed methodology concerning a Java application which implements a simulated control of a 2 degrees-of-freedom robot manipulator.

Finally, Section IV is dedicated to some concluding remarks.

II. Basic Concepts

Object-Oriented Control Systems Design (**OOCSD**) mimics the general idea concerning Object-Oriented Programming (**OOP**). Indeed, **OOCSD** is just control design based in object-oriented programming technics. In what follows we recall some ideas concerning object-oriented programming. The interested reader can consult for instance [5], [8] and [16], for a more extensive survey on this topic.

A. Object-oriented programming and Java

Object-oriented programming (**OOP**) is one the most powerful programming paradigm in recent years. This programming idea organizes programs in ways that echo how things are put together in the real world. When we use object-oriented programming, our overall program is made up of lots of different self-contained components (*objects*), each of which has a specific role in the program and all of which can communicate one to another in predefined ways. Object-oriented programming is not limited to combining objects, it also provides many other concepts and features to create and use objects in an easier and flexible way.

One of the concepts employed in **OOP** is that of *class*. A class is a template for multiple objects with similar features, i.e., a class is a (generic) general representation of an object (the concrete representation of an object is called *instance*). In fact, when we write a program in an object-oriented language (like Java or C++), we do not define actual objects. We define classes of objects, which are generally grouped in *class libraries*.

As far as Java is concerned, every class is generally made up of two components: *properties* and *methods*. Properties, defined by variables, are the individual characteristics that differentiate one object from another and determine its appearance, state or other qualities. Class methods determine how the instances of the class change their internal state or react when the instance is asked to do something by another class or object. In fact, they are functions defined inside classes that operate on instances of those classes. As is common in object-oriented languages, there exists in Java some mechanisms for organizing classes and class behaviours: *inheritance*, *interfaces*, and *packages*.

The concept of inheritance is that when we write a class, we only have to specify how that class is different from some other class. Inheritance will give us automatic access to the information contained in the original class. When using inheritance, all classes are arranged in strict hierarchy. Each class has a superclass, and each class have one or more subclasses. Classes further down in the hierarchy are said to inherit from classes further up in hierarchy. Subclasses inherit all the methods and variables from their superclasses. At the top of the Java class hierarchy is the class *object*, all classes inherit from this superclass. Object is the most general class in the hierarchy; it defines behaviour inherited by all the classes in the Java hierarchy. Each class further down in the hierarchy adds more information and becomes more tailored to a specific purpose.

As far as Interfaces and Packages are concerned, both are advances topics for implementing and designing groups and interfaces. An Interface is a collection of method names

Partially supported by Conacyt Research Project 26450-A, México.

without actual definition indicating that a class has a set of behaviours in addition to the behaviours the class gets from its superclasses. Packages in Java are a way of grouping together related classes and interfaces: Package enables modular groups of classes to be available only if they are needed and eliminates potential conflicts between class names in different groups of classes.

B. The World Wide Web as an Automatic Control tool

It is probably unnecessary to tell anyone reading this article about the rapid growth of the Internet, mainly due to the world-wide price reduction in personal computers and connection services. In particular, as the main Internet service, the *World Wide Web*, that we just call the *Web* in the sequel, has now a privileged place in the popular culture. In fact, the Web is never unavailable on today´s university campus, which is very appealing for Automatic Control purposes, including education and remote experimentation (see for instance [6], [10], and [13]).

Because of its accessibility, the Web is a real low-cost alternative to the expensive traditional training services based on laboratory demonstrations. Moreover, the Web as a platform for didactic purposes allows the students to schedule his own learning (see for instance [12] and [14]). The Web also allows 24 hours per day accessibility to virtual and real experimental facilities (see for instance [9]).

Due to the expensive cost of real prototypes, nowadays the academy enhances theoretical teaching with simulation-based demonstrations, and because of its accessibility, the Web can be used as a platform to implement remote experimental facilities with the main advantage that there exists some nice development tools which are free of charge.

C. Java and its possibilities

In order to have a better insight on the possibilities that Java offers, let us briefly describe the main characteristics of this language. For more information see for instance [5] and [8].

Because Java is an object oriented programming language designed to provide a simple, attractive interface to information on the Web, it is a natural tool for the conception on Web-based Automatic Control facilities.

The Java syntaxis structure is very similar to the C++ syntaxis structure, but its virtual machine is completely different; C++ is compiled to the native language of the computer where compilation is performed, which is not the case for Java. As an obvious consequence, the execution time of Java programs is poor as compared with the execution time of an equivalent compiled C++ program (see for instance [7]). In fact, Java is not a genuine compiled language. Indeed, Java development systems simply convert Java programs into a very compact, cross-platform, byte code that can be downloaded and interpreted by a Web browser (this characteristic is what makes Java a very attractive Internet development tool). In fact, almost all the popular browsers are now Java compatible.

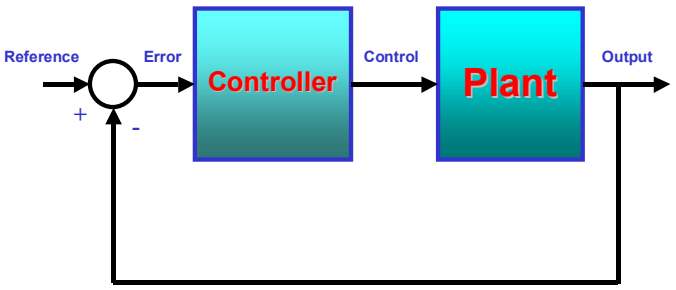

Fig. 1. The Control Problem.

Because of its platform-independency nature, we decided to use Java in our project. It must be pointed out that Java is a Web-oriented general-purpose programming language, not a scientific computation tool, which makes not easy to perform engineering computations. This lack of scientific computing skills, as a consequence of the universal accessibility, dificults the development of Web-based Automatic Control facilities, offering then a number of interesting challenges.

D. The object-oriented nature of the Control problem

Broadly speaking, the control problem, as illustrated in Figure 1, pursues the modification of the *Plant* behaviour in order to influence its *output*, also called *actual output*, in a desired way. This modification is attained through the action of the *Controller*, on the Plant *input*, which reacts to the *error signal*, just called *error*. The error is equal to the *actual output* minus the *desired output*, also called *reference*.

In terms of the object-oriented paradigm, both, the controller and the plant belong to the family of dynamical systems. Indeed, both systems are dynamical blocks interacting with their environment through theirs corresponding inputs and outputs. If we use the block diagram paradigm, we can say that both systems belong to the Block class. Thus, both, the controller and the plant can be defined as subclasses of the Block superclass. The instances corresponding to the controller and the plant classes (for a particular application), are specified by the parameters of the concret controller and the concret plant. In order to illustrate these ideas, we present in the next section a particular application concerning the simulation of a closed-loop control scheme including an industry oriented controller and a well known two degrees-of-freedom robot manipulator (see [15]). Let us remark that object-oriented programming has in fact its roots in simulation. Indeed, the first object-oriented programming language, Simula, was developed to provide simulation facilities within a general purpose programming language (see for instance [4] and [11]).

III. An illustrative example

With respect to Figure 2, the control of a virtual two degrees-of-freedom robot manipulator can be described as follows:

1. The Client uses a Graphic User Interface (**GUI**) in order

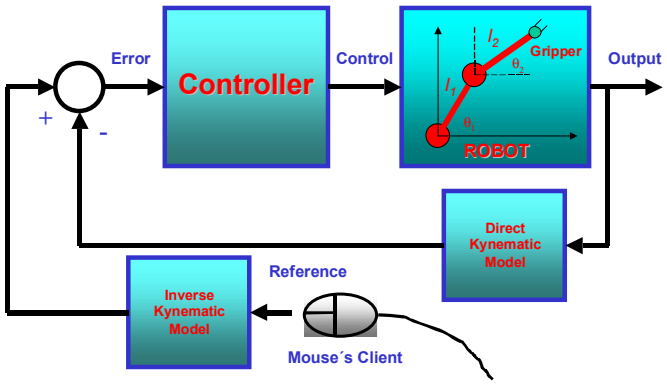

Fig. 2. Main Idea.

to obtain a point in cartesian coordinates on the Client´s computer screen. It is the point where the user wants to place the gripper of the virtual robot manipulator. The desired position of the gripper is then generated by the Client's mouse.

2. The desired cartesian position of the gripper is transformed to the manipulator generalized coordinates, i.e., the angular positions α and β of the two links l_1 and l_2, respectively. These coordinates constitute what is usually called the reference input and they are generated by the inverse kynematic model of the robot manipulator.

3. The actual magnitud of the angular position of the two links, i.e., the *output*, are mesured and compared with a *reference input*, in order to obtain the *position error*.

4. The controller receives the position error and generates the *control input*. The parameters of the controller are specified by the Client via the **GUI**.

5. Finally, the virtual robot manipulator reacts to the control input in a dynamic nonlinear way. The final output is generated by the direct kynematic model of the robot manipulator.

With this information flow in mind, we proceed to the synthesis of the Java program which implements the control of the virtual two degrees-of-freedom robot manipulator.

A. Program synthesis

First of all, let us make some comments about the notation: Sans serif characters are used to indicate the name of a particular Java class, always beginning with a capital letter (consider for instance the Java class called Block). As far as an object is concerned, we also use Sans serif characters, but in this case only small letters are used. Consider for instance the object called block. Finally, we use `Typewriter` characters to write the source code of the programs, including the data names and the member methods on the Java classes.

We can now present the modules which conforms our Java application.

A.1 The robot manipulator

The idea behind the module which concerns the virtual robot manipulator is the dynamical model of a well-known 2R manipulator (see for instance [3]). There exist several approaches to model this kind of systems. Because of the nonlinear nature of the robot, the state space approach is usually considered. In this case, the dynamic behaviour of the system is described by a set of differential equations:

$$\begin{cases} \dot{x}(t) &= f(x(t)) + g(x(t))\,u(t) \\ y(t) &= h(x(t)), \end{cases} \tag{1}$$

where: $x \in \mathbb{R}^n$ denotes the state; $u \in \mathbb{R}^{\geqslant}$ denotes the control input, and $y \in \mathbb{R}^{\geqslant}$ denotes the output. f, g, and h are real valued non linear functions. In our case, this dynamical model is implemented via a discretizing Euler integrator (see for instance [2]).

A.2 The Block class

Following the object-oriented approach, we define an interface for the virtual robot manipulator considering the natural functionality of a 2R manipulator. Applying the block diagram paradigm, this interface is constituted by two basic operations, i.e.:

a) To apply a signal at the input, and

b) To measure the signal at the output.

Extending this idea, the block corresponding to the virtual robot manipulator is built using the following components:

- To apply a signal at the input.
- To measure the signal at the output.
- To observe the state.
- To compute both, the inverse and the direct kynematic models.
- To paint the kynematic chain on the graphic plane.

Since both, the robot manipulator and the controller can be considered as blocks (recall the block diagram paradigm shown in Figure), we first define a basic Java superclass calles Block, which is defined in terms of the basic concept of state. This Block is build around the data constituted by the triplet $(x,\ y,\ u)$, i.e., the state, the output, and the input, respectively. The interface corresponding to the basic Java superclass that we are considering is shown in Listing 1.

```
package ctrl;
public abstract class Block {
 double[] x;//state
 double[] y;//output
 Block nextBlock; //reference to the next Block
 public static double t = 0.0001;//sampling time
 public Block(int n, int m);
 abstract double[] dynamics(double[] u);
 abstract double[] output();
 public void connectTo(Block newBlock);
 public boolean connectMe(Block newBlock);
 public boolean setState(double[] x0);
 public double[] getState();
 public double[] setInput(double[] u);
 public double[] getPutput();
 public String toString();
```

```
 public static void setSamplingTime(double st);
}
```

Listing 1. Block class.

The purpose of the dynamics method is to program the dynamics of the model using a nonlinear set of differential equations as the one described by (1). The connectTo method links the output of a Block with the input of one other Block via the reference nextBlock. The setState method assigns the value of the state vector x, which is useful to provide the initial state of the Block. The getState method allows the observation of the state vector x, which is useful when state feedback is being considered. The setInput method gives an input to the Block and the getOutput method measures the value of the output vector y of the Block. Finally, the setSamplingTime method fixes the integration time t.

A.3 The Robot2R class

Now, taking as a base the Block superclass, we define the Robot2R derived class, which models a robot manipulator of type 2R (see [3]). The Robot2R class adds some particular characteristics to the Block superclass, mainly the *lenght* and the *mass* of the links (we assume at this level that the two links have the same physical parameters). The definition of the Robot2R class is specified in Listing 2.

```
package ctrl;
inport java.awt.*;
public class Robot2R extends Block {
 int linkSize;//Lenght of a link
 int linkScale;
 double linkMass;//Mass of a link
 int baseX, baseY;//Coordinates of the Robot Base
 public Robot2R(int size, double mass, int bX, int
bY);
 double[] dynamics(double[] u);
double[] output();
 public double[] setInput(double[] u);
 public double[] getInvKyn(int xd, int yd);
public int[] getPosition();
 public boolean isInside(int xd, int yd);
 public int[] xformToG(int xl, int yl);
 public int[] xformToG(double xl, double yl);
 public int[] xformToL(int xg, int yg);
 public int[] xformToL(double xg, double yg);
 public void draw(Graphics g);
 void drawBase(Graphics g);
 void drawWorkSpace(Graphics g);
 double sin(double u);
 double cos(double u);
 double acos(double u);
 double atan2(double u, double v);
 double sqrt(double u);
}
```

Listing 2. Robot2R class.

The Robot2R builder defines the dimensions of the state vector x and the output vector y. Robot2R also initializes the particular characteristics of a 2R manipulator. In this case the method dynamics models the dynamics of the 2R manipulator as is specified in (1). The state vector $x = (x_0(t), x_1(t), x_2(t), x_3(t))$ is defined as follows:

$$\begin{aligned} x_0(t) &:= \theta_1(t) \\ x_1(t) &:= \theta_2(t) \\ x_2(t) &:= \dot{\theta}_1(t) \\ x_3(t) &:= \dot{\theta}_2(t), \end{aligned}$$

where $\theta_i(t)$ and $\dot{\theta}_i(t)$ denote the angular position and the angular speed of the i-th link, respectively. The model of the 2R manipulator is given by the following differential equations:

$$\begin{aligned} \begin{bmatrix} \dot{x}_0(t) \\ \dot{x}_1(t) \end{bmatrix} &= \begin{bmatrix} x_2(t) \\ x_3(t) \end{bmatrix} \\ \begin{bmatrix} \dot{x}_2(t) \\ \dot{x}_3(t) \end{bmatrix} &= M^{-1}(x(t))[-V(x(t)) - G(x(t)) \\ &\quad -F(x(t)) + u(t)], \end{aligned}$$

where the matrix functions $M(x(t))$, $V(x(t))$, and $G(x(t))$ are defined as in [3]. $F(x(t))$ denotes the viscous friction.

The output method implements the output of the 2R manipulator, i.e., gives a real vector constituted by the angular positions of the links ($x_0(t)$ and $x_1(t)$). The setInput method implements an Euler integrator. The getInvKyn method computes the inverse kynematic model of the 2R manipulator. The parameters associated to this method are the coordinates of the gripper desired cartesian position and the return value is a real vector of dimension 2 constituted by the computed angular positions. If the desired point is not in the workspace, a new point is build on the workspace border.

The getDirKyn method computes the direct kynematic model of the 2R manipulator, receiving as inputs the two angular positions. As far as the actual position of the kynematic chain is concerned (in cartesian coordinates), it is computed by the getPosition method. This method produces a real vector $p(t) = (p_0(t), p_1(t), p_2(t), p_3(t))$ defined as follows:

$$\begin{aligned} p_0(t) &= \overline{x}_1(t) \\ p_1(t) &= \overline{y}_1(t) \\ p_2(t) &= \overline{x}_2(t) \\ p_3(t) &= \overline{y}_2(t), \end{aligned}$$

where $(\overline{x}_1(t), \overline{y}_1(t))$ and $(\overline{x}_2(t), \overline{y}_2(t))$ are the cartesian positions of the robot manipulator joints on the graphic plane.

The isInside method verifies if a given point is inside the manipulator workspace. The xformTog method converts the local coordinates on the graphic plane. The xformTol method converts the global coordinates of the graphic plane to the local coordinates of the manipulator. Finally, the

draw method (and all its associated methods) draws the robot manipulator and its workspace on the graphic plane. As it can be seen, some trascendental functions are also defined in this class in order to facilitate the syntaxis.

A.4 The PID control class

We choose a very well known Proportional Integral Derivative (**PID**) controller (see for instance [1] and [3]), in order to illustrate how a particular controller can be included in our Java application. To obtain the **PID** controller, we built a Block derived class, that we just call Controller. This class has not in fact a real functionality, but a syntactic one. Extending the Controller class, we obtain the ControlPID class described in Listing 3.

```
package ctrl;
public class ControlPID extends Controller {
 double[] q0, q1, q2;//PID gains
 double[] e0, e1, e2;//Memory for the error
 Block msrBlock;
 public ControlPID(int n);
 double[] dynamics(double[] u);
 double[] output();
 public double[] setInput(double[] u);
  public void setGain(int n, double k, double ti,
double td);
 public void measureFrom(Block newBlock);
}
```

Listing 3. ControlPID class.

The ControlPID class implements the dynamics of a discret time **PID** controller. This method counts with variables to drive the **PID** gains as well as the error memories. The class also have the necessary methods to drive these new data. Since the **PID** gains are stocked in vectors, we can have in fact several **PID** controllers in memory. The controlPID builder dimensions the **PID** controller.

The dynamics method implements the discrete **PID** controller given by the following equation:

$$x(k) = x(k-1) + q_0 u(k) + q_1 u(k-1) + q_2 u(k-2),$$

where: u denotes the existing error between the actual coutput of the robot manipulator and the reference input; q_0, q_1, and q_2 denote the **PID** gains.

The output method directly connects the state with the output. The setInput method obtains both, the reference input and the actual robot manipulator output in order to compute the resulting error. The setGain method adjusts the **PID** gains, following the well known standard rules (see for instance [3]):

$$\begin{aligned} q_0[n] &= k\left(1 + \frac{t}{t_i} + \frac{t_d}{t}\right), \\ q_1[n] &= -k\left(1 + 2\frac{t_d}{t}\right), \\ q_2[n] &= k\frac{t_d}{t}, \end{aligned}$$

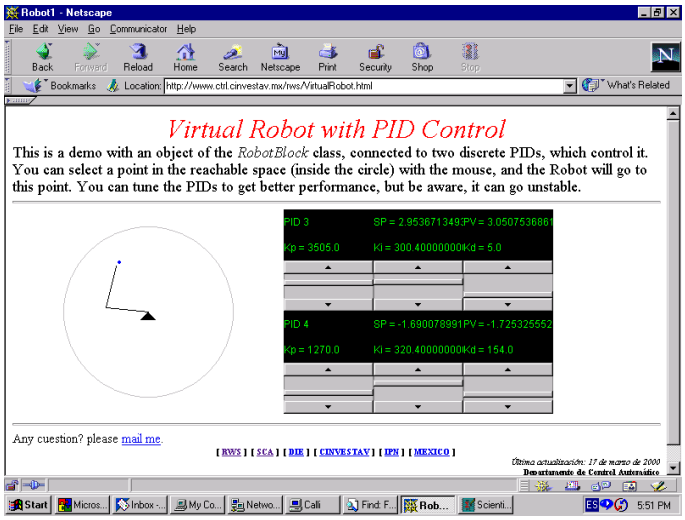

Fig. 3. PID control position of the gripper-Point 1.

where: k denotes the proportional gain; t_i denotes the integral time (also called reset), and t_d denotes the derivative time.

Finally, the measureFrom method connects controlPID to the block which produces the measure.

B. Graphic User Interface

The Graphic User Interface (**GUI**) is compossed of two parts: the first one comprises the Display class, which implements a graphic display generated by the Panel class (which is part of the standard library of Java). The graphic display allows the visualization of the virtual robot manipulator movement. The Display class calls the draw method of Robot2R and obtains a position on the graphic plane via a click from the Client's mouse. This point is then converted in a real vector constituting the reference input. It is important to say that Display uses the Java multithreading resources, in order to give rise to a nice animation of the virtual plant.

The second part of the **GUI** is constituted by an applet which connects the scrollbars with ControlPID, in order to allows the Client to adjust the **PID** gains.

The behaviour of our Java program is illustrated for two differents gripper positions in Figure 3 and Figure 4. Note that the **PID** gains can be modified by the Client using the mouse.

Remark that the classes that we presented here are grouped in a package called ctrl.

IV. Concluding Remarks

A Java-based methodology concerning object-oriented control systems is presented in this paper. We have shown that the object-oriented paradigm can be easily applied to the synthesis of Automatic Control applications. We illustrated it with an example concerning the simulation of a popular control scheme of a two degrees-of-freedom robot manipulator. The structure of the Java application mimics the diagram of blocks corresponding to the closed-loop sys-

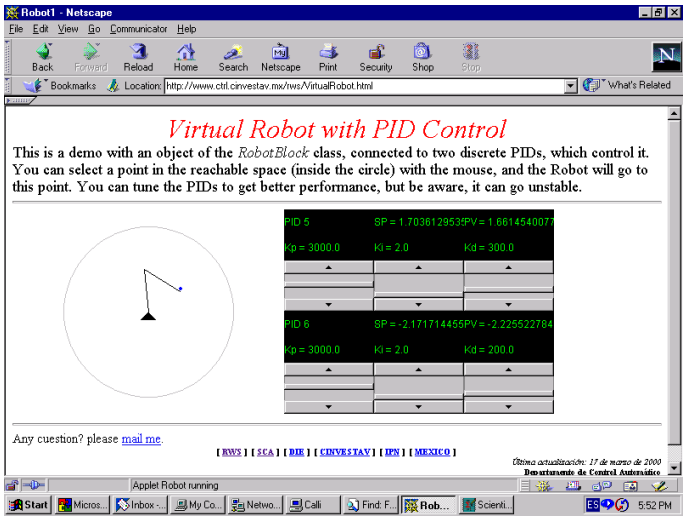

Fig. 4. PID control position of the gripper-Point 2.

tem. The derived class for any arbitrary dynamical system can be obtained from the Block class, which makes possible the simulation of a huge diversity of control schemes.

We can affirm that even if Java was not conceived for developping engineering applications, its object-oriented nature and its platform independency make of Java an excellent tool to develop Automatic Control applications. Indeed, Java allows the developer to easily implement several control strategies. In our last section we illustrate this possibility including a discret time **PID** controller (see the application at *http://www.ctrl.cinvestav.mx/rws/VirtualRobot.html*), which can be easily changed to include a more sophisticated controller, with no extra time conception cost. Let us remark that the Euler method we applied to discretize the nonlinear dynamics of the plant, can be substituted by a Runge-Kutta method, in order to insure better numerical properties in our Java application.

We are developping nowadays a control library which will include more sophisticated control strategies, including real-time control. Our Web-based service will also include in the short term a Matlab based tutorial.

References

[1] K. J. Aström and W. Witenmark, *Computer Controlled Systems*. Prentice Hall, New Jersey, USA, 1989.

[2] J. -L. Chabert, E. Barbin, M. Guillemot, A. Michel-Pajus, J. Borowczyk, A. Djebbar and J.-C. Martzloff, *Histoire d'Algorithms: du caillou à la puce*. Belin, Paris, 1994.

[3] J. J. Craig, *Introduction to Robotics*. 2nd Edition, Addison-Wesley Publishing Company, 1989.

[4] O. J. Dahl and K. Nygaard, "Simula - an algol-based simulation language", *Communications of the ACM*, 9(9), pp. 671-678, September 1966.

[5] B. Eckel, *Thinking in Java*. Prentice-Hall PTR, 1998.

[6] D. Etter, C. R. Johnson, Jr., and G. Orsak, "Experiences using the Internet and WWW to facilitate remote teaming of students and faculty", *Proceedings of the IEEE Conference of Decision and Control*, San Diego, CA, USA, 1997.

[7] M. Felton, *CGI Internet Programming with C++ and C*. Prentice Hall, New Jersey, USA, 1997.

[8] D. Flanagan, *Java in a nutshell*. O´Really & Associates Inc., 1996.

[9] D. Gillet, C. Salzman, R. Longchamp, and D. Bovin, "Telepresence: an opportunity to develop real-world experimentation in education", *European Control Conference (ECC´97)*, Brussels, belgium, July 1-4, Track L No. 439, 1997.

[10] Y. C. Ho, W. B. Gong, C. Cassandras, J. Q. Hu, and P. Vakili, "Experience with developing and dispensing results and advanced course materials on the World Wide Web of the Internet", *Proceedings of the IEEE Conference of Decision and Control*, San Diego, CA, USA, 1997.

[11] B. Kirkerud, *Object-oriented programming with Simula*. Addison-Wesley, Reading, MA, USA, 1989.

[12] J. Linfors, L. Yliniemi, and K. Leiviskä, "Hypermedia based learning for process automation", *European Control Conference (ECC´97)*, Brussels, belgium, July 1-4, Track L No. 924, 1997.

[13] J. Luntz, W. Messner, and D. Tilbury, "Web technology for control education", *Proceedings of the IEEE Conference of Decision and Control*, San Diego, CA, USA, 1997.

[14] D. Munteanu, F. Michau, and S. Gentil, "Autodidact: a simulation-based learning environment in automatic control", *European Control Conference (ECC´97)*, Brussels, belgium, July 1-4, Track L No. 925, 1997.

[15] G. H. Salázar-Silva, J. C. Martínez-García, and R. Garrido, "Enhancing Basic Robotics Education on the Web", *American Control Conference (ACC´99)*, San Diego, CA, 1999.

[16] P. Winters, D. Olhasso, L. Lemay, and C. L. Perkins, *Visual Java in 21 days*. Sams Net, Indianapolis, Indiana, USA, 1996.