maestra y se activan los de la simulación esclava que posee la tiza para poder controlarla.

Para la sincronización, no es necesario que la simulación con la tiza tenga la lista de alumnos, ya que todas las acciones que realice son comunicadas al maestro, que se encarga de retransmitirlas al resto de clientes conectados. En el caso del *step*, el esclavo que posee la tiza se espera a que el maestro le indique que todos los alumnos han dado el paso de simulación.

5.3.4. Conclusiones

La combinación de los laboratorios virtuales y los entornos colaborativos *on-line* es hoy en día la unas de las mejores alternativas para la educación a distancia. Los laboratorios virtuales son herramientas de auto-aprendizaje que ayudan a comprender mejor los conceptos a los alumnos. Y los entornos colaborativos *on-line* permiten compartir ideas y experiencias entre el profesor y los alumnos en tiempo real.

Muchos entornos colaborativos existentes en la actualidad incluyen laboratorios virtuales y remotos. Sin embargo, muy pocos de ellos son de carácter síncrono. Motivados por esta carencia, se ha desarrollado un primer prototipo de un sistema de comunicación *on-line*, usando *TCP Sockets*, entre simulaciones interactivas desarrolladas con *EJS*.

El sistema se ha incluido como una opción más del software *EJS*, con la ventaja de que cualquier simulación creada con esta herramienta y usada en un ordenador conectado a Internet, sirve para realizar una clase virtual. La sincronización de las simulaciones permite al profesor enseñar en tiempo real a todos los alumnos conectados.

CAPÍTULO 6

Resultados Obtenidos

En este capítulo se expondrán los distintos laboratorios virtuales y remotos desarrollados, así como las aportaciones realizadas a *EJS*, como resultado durante el periodo de investigación. Es importante comentar que parte de las aplicaciones que se van a describir se han presentado en congresos nacionales e internacionales (ver sección 7.3).

6.1. Laboratorios Virtuales

Laboratorio virtual de un robot de 6gdl tipo PA-10.

Para el primer laboratorio virtual desarrollado se utilizó la versión antigua de *EJS* y se usó la conexión con *Matlab* [Sanchez y otros, 2005]. Por lo tanto, el modelo del robot se definió mediante una serie de archivos en código *Matlab* (archivos *.m*). *EJS* se utilizó para el desarrollo de la interfaz (vista del robot y controles), declarar las variables y conectarlas con las variables de salida del modelo robótico. Cada cambio en la interfaz de usuario se envía al *workspace* de *Matlab* y es leído por el modelo. De igual modo, las salidas del modelo se escriben en el *workspace* que *EJS* se encarga de leer para actualizar la vista de la aplicación. Como vemos en la Figura 6.1, el intercambio de datos entre *EJS* y *Matlab* se realiza a través del *workspace*.

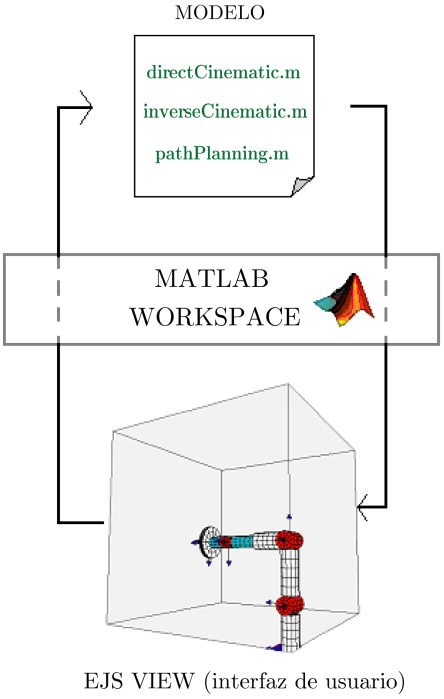

Figura 6.1: Conexión entre *EJS* y *Matlab*

La vista principal del laboratorio virtual se muestra en la Figura 6.5. La parte superior contiene una representación 3D del espacio de trabajo donde se visualiza el robot de 6gdl. Los diálogos que observamos se utilizan para mover el robot. En la parte inferior se visualiza el valor de cada una de las articulaciones, de la posición (X, Y, Z) y orientación $[n, s, a]$ del punto final. Las distintas opciones que implementa el laboratorio virtual son las siguientes:

- Cinemática directa e inversa. Tal y como se comentó en la sección 4.2, el movimiento del robot se basa en la implementación del modelo cinemático directo e inverso del mismo. De este modo, es posible mover el robot dando valores a cada una de sus articulaciones (controles $q1 \ldots q6$) o especificando la posición (controles X, Y, Z) y orientación (controles $X°, Y°, Z°$) del punto final. Con respecto a la orientación, los valores de $(X°, Y°, Z°)$ corresponden a los ángulos *Yaw*, *Pitch*, *Roll* [Barrientos y otros, 1997].

- Parámetros geométricos. En la aplicación desarrollada es posible cambiar el valor de los parámetros geométricos del robot. La simulación se

actualiza dinámicamente mientras el usuario cambia dichos parámetros (Figura 6.2).

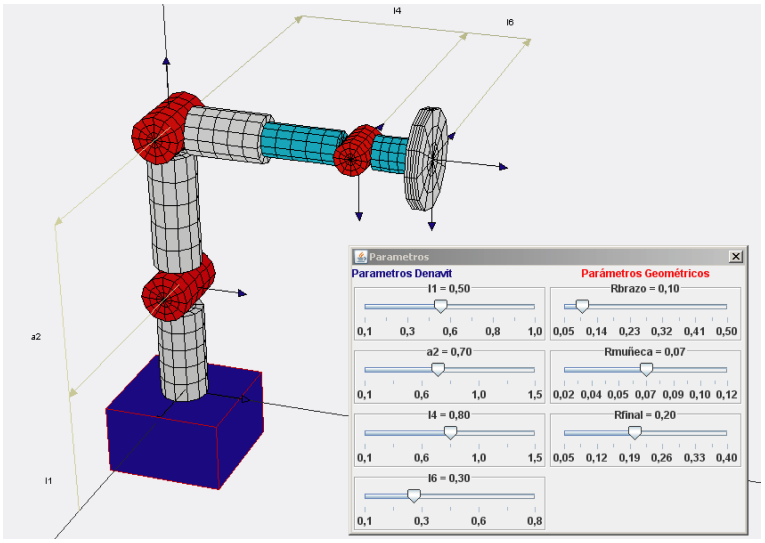

Figura 6.2: Parámetros Geométricos

- Planificador de trayectorias. La aplicación implementa un planificador de trayectorias en el espacio articular. En concreto, la trayectoria polinómica 4-3-4 (sección 4.4). Como vemos en la Figura 6.3, la aplicación grafica la trayectoria 3D en el espacio de trabajo y la evolución temporal de las posiciones, velocidades y aceleraciones.

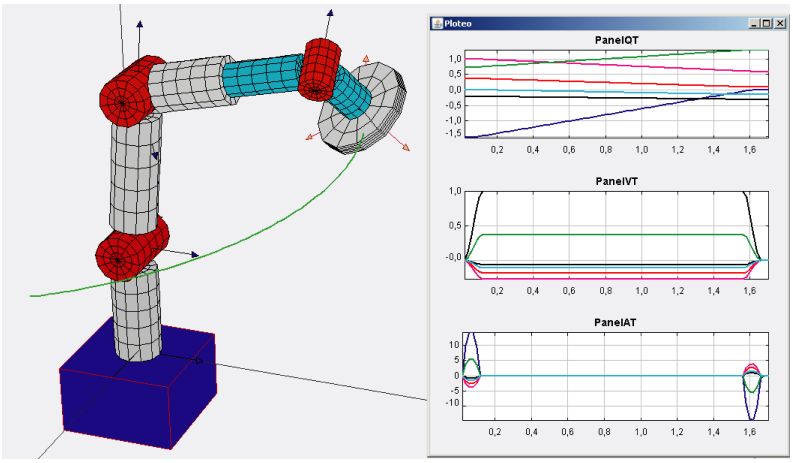

Figura 6.3: Planificador de trayectorias 4-3-4

- Configuración del entorno de trabajo. Es posible introducir objetos prediseñados en el entorno (Figura 6.4), tales como cubos, conos, esferas,... Esta opción de la aplicación se encuentra todavía en desarrollo, ya que se pretende la introducción de tales objetos no sólo para configurar el entorno de trabajo, si no también para poder manipularlos mediante el robot.

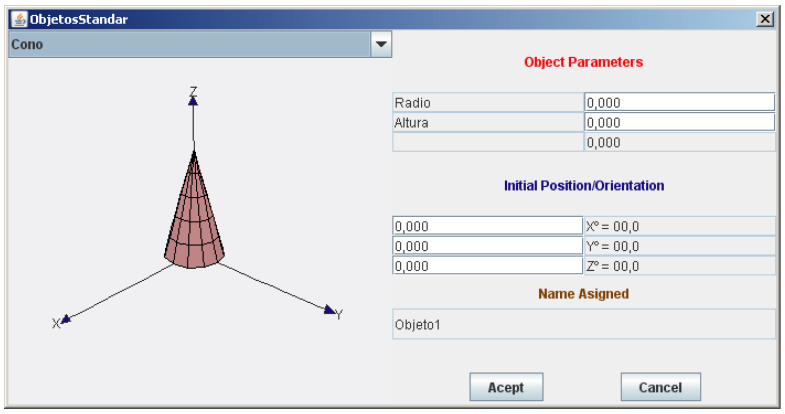

Figura 6.4: Diálogo para insertar objetos en el entorno

6.2. Laboratorio Remoto

Nuestro grupo de investigación desarrolló un sistema de teleoperación denominado *Robolab* [Candelas y otros, 2003a]. Dicho sistema es capaz de teleoperar robots industriales a través de Internet localizados en un laboratorio físico. Aprovechando la arquitectura de este sistema, se ha desarrollado un laboratorio remoto mediante *EJS* capaz de controlar remotamente un robot tipo Scorbot-ER IX.

El sistema de teleoperación desarrollado puede utilizarse desde cualquier ordenador que posea conexión a Internet. La aplicación puede ser ejecutada como un programa *stand-alone* o como un *applet* desde un explorador. Puede accederse a una versión reciente de la aplicación en forma de *applet* a través de la dirección web `http://www.aurova.ua.es/robolab/index.html`. Además, la interfaz proporciona un entorno virtual muy real con el que el usuario puede simular y practicar trayectorias antes de teleoperar el robot. A continuación se exponen más detalles del laboratorio remoto desarrollado.

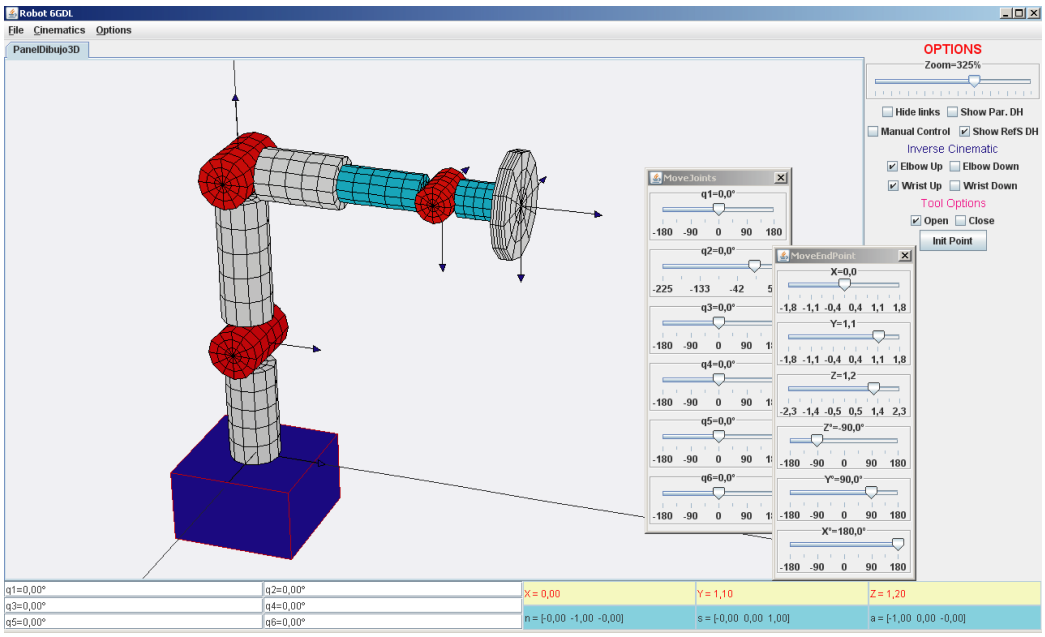

Figura 6.5: Interfaz de usuario del laboratorio virtual

Arquitectura del sistema

Los diferentes elementos que componen el sistema se muestran en la Figura 6.6. Como se puede observar, existen dos partes bien diferenciadas: el usuario conectado a Internet con la aplicación *EJS* y el laboratorio remoto con todos sus elementos *hardware*.

Para ejecutar la aplicación, el usuario tan sólo necesita poseer acceso a Internet, un explorador web, y tener instalado la *JVM* (*Java Virtual Machine*) 1.6 y *Java 3D Runtime* 1.4 o mayor.

Los componentes principales del laboratorio remoto son el robot Scorbot y su controlador RS-232. Respecto al resto de elementos *hardware*, el servidor principal es el que actúa como servidor web del laboratorio remoto. Se encarga de controlar el acceso de los usuarios y de contener el *applet* de teleoperación al que accede el cliente. El servidor de teleoperación se encarga de validar las órdenes de posición que envía el usuario desde la aplicación, traducirlas a lenguaje ACL (*Advanced Control Language*), lenguaje de programación del robot, y remitirlas al controlador para ser ejecutadas. El servidor de vídeo es un AXIS 2400[1] que permite a los usuarios tener realimentación visual durante

[1] *AXIS Web Site:* `http://www.axis.com/`

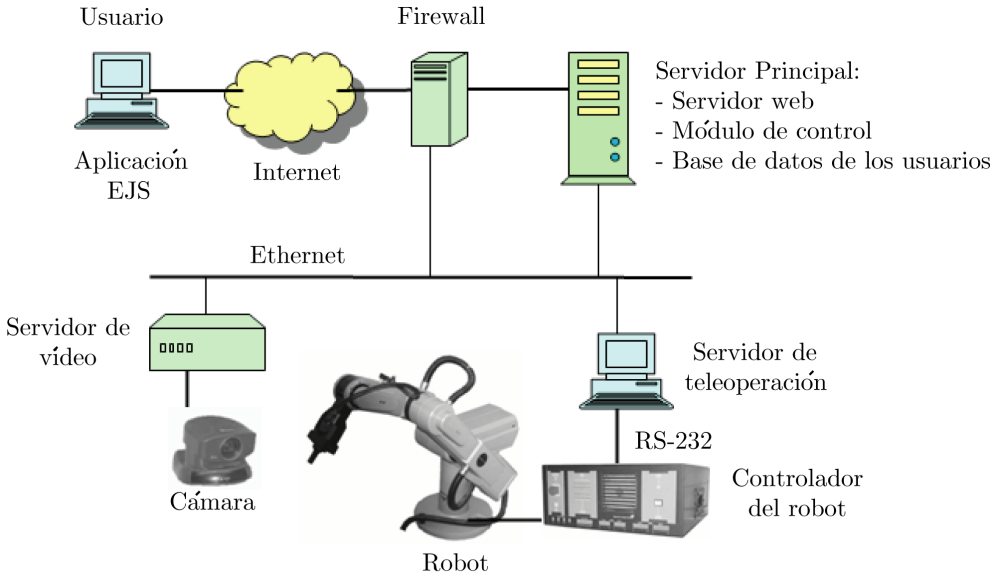

Figura 6.6: Arquitectura del sistema

el proceso de teleoperación. Finalmente, existe un *Firewall* en la entrada de red del laboratorio que aumenta su seguridad.

Interfaz de usuario

Para el desarrollo de la aplicación se ha empleado una versión más actual de *EJS* que la utilizada con el laboratorio virtual descrito en la sección anterior. Mediante esta versión, se consiguen unos gráficos más reales y con mejor apariencia, tal y como se observa en la Figura 6.7 donde se muestra la interfaz de usuario.

La distribución de las distintas partes en la interfaz es muy similar a la del laboratorio virtual explicado en la sección anterior: en el centro, la representación 3D del robot junto con su espacio de trabajo; en la parte inferior, los campos numéricos con el valor de cada una de las articulares y del punto final; y a la derecha, controles que implementan funciones como zoom, mostrar/ocultar los sistemas D-H, etc, ... Además, las opciones que incorpora son prácticamente las mismas: cinemática directa e inversa y el planificador. La única diferencia es que el tipo de planificador no es el mismo: aquí es de tipo lineal (ver sección 4.4). Los tipos de trayectorias que puede simular el usuario con el interpolador lineal son dos: *síncronas* y *asíncronas*.

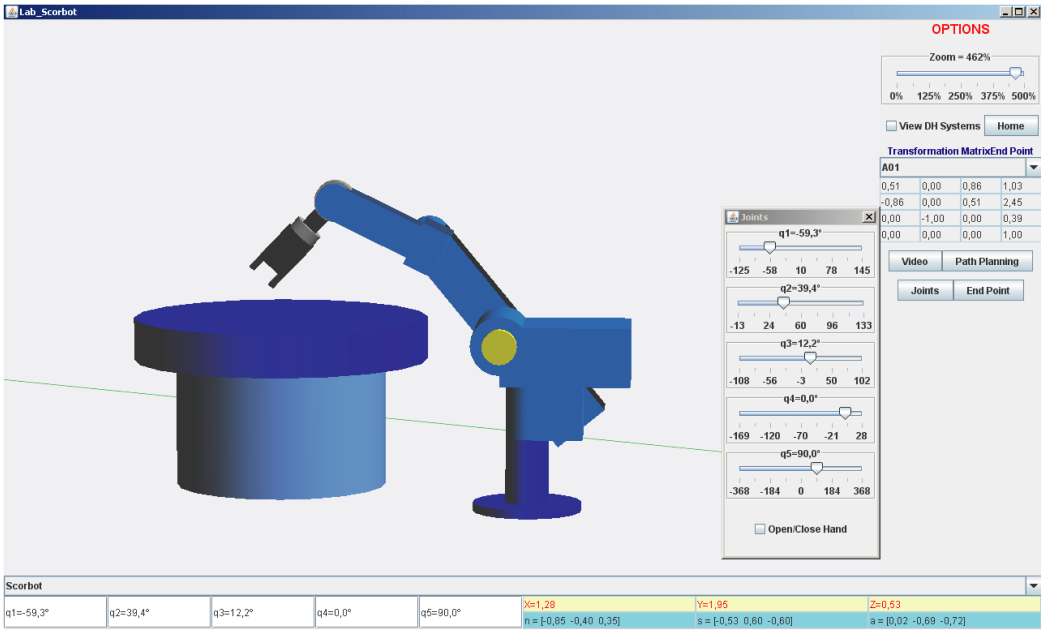

Figura 6.7: Interfaz de usuario del laboratorio remoto

Cada vez que el usuario ejecute una trayectoria, puede guardarla en una lista para su posterior ejecución. Si existe más de una trayectoria en la lista, ya sean síncronas o asíncronas, el usuario puede ejecutarlas secuencialmente (Figura 6.8). La simulación también pinta la trayectoria 3D realizada por el extremo del robot y grafica la evolución de los valores articulares con respecto al tiempo.

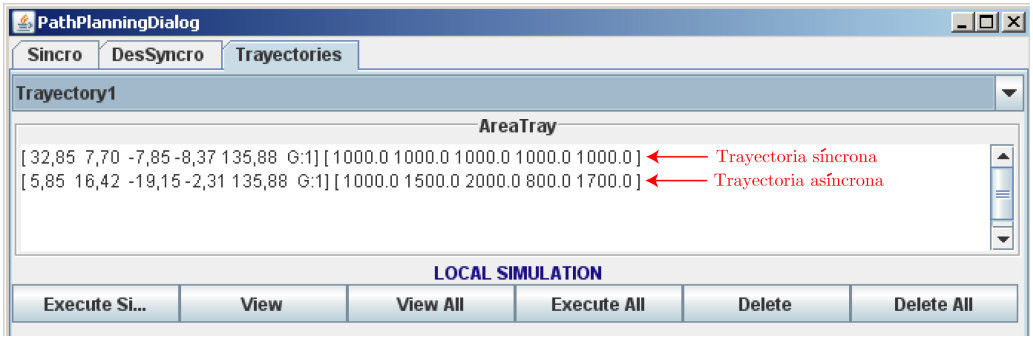

Figura 6.8: Lista de trayectorias de la interfaz

Teleoperación.

Después de que el usuario haya realizado una trayectoria en la simulación, puede teleoperar el robot localizado en un laboratorio remoto desde la misma interfaz. Para ello, es necesario que el usuario esté registrado en una base de datos que se encuentra en el servidor principal.

La comunicación entre la aplicación *EJS* y el servidor principal del laboratorio remoto es mediante el protocolo HTTP. El usuario envía a través de una URL los datos de acceso (usuario y contraseña) y los valores articulares de la trayectoria. Una vez que los datos llegan al servidor, un módulo de ASP (*Active Server Pages*) comprueba la identidad del usuario en una base de datos y si se encuentra registrado, envía la trayectoria articular al servidor de teleoperación para ser ejecutada por el robot remoto.

El servidor de teleoperación es una programa en Java que se comunica mediante *TCP sockets* con el servidor principal. El intercambio de datos entre ambos consta de las trayectorias a ejecutar (servidor principal → servidor teleoperación) y de la realimentación de los valores articulares durante la ejecución de la trayectoria (servidor teleoperación → servidor principal). Antes de ejecutar una trayectoria en el robot físico, el servidor de teleoperación comprueba en una simulación local que dicha trayectoria es correcta. Esta acción se realiza para garantizar el uso correcto del robot físico y de esta manera, no dañar los elementos del laboratorio.

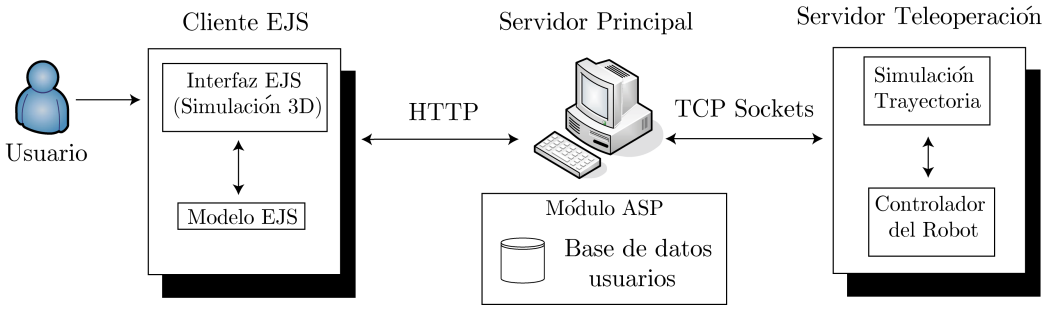

Figura 6.9: Comunicación Cliente EJS-Robot Teleoperado

Opciones de realimentación.

Para visualizar el movimiento real del robot desde la interfaz cliente, la aplicación dispone de dos opciones de realimentación. La primera es mediante

un flujo de vídeo comprimido que genera el servidor AXIS 2400 y se transmite a la aplicación *EJS* para ser mostrada de forma *on-line*. La segunda consiste en una actualización de la simulación con información recibida desde el controlador del robot de los valores articulares reales (Figura 6.10). De esta manera, el usuario observa una representación gráfica del estado real del robot empleando menos ancho de banda que en el caso del vídeo.

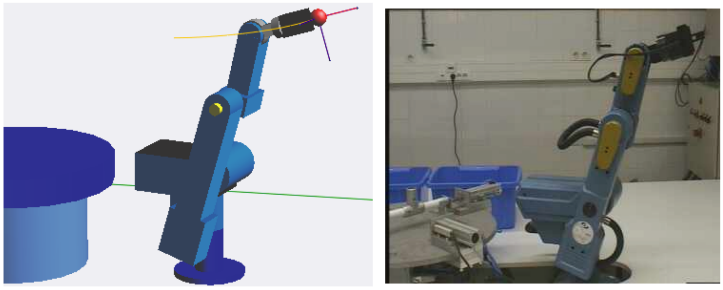

Figura 6.10: Opciones de realimentación (simulación-vídeo)

6.3. Aportaciones a *EJS*

Tal y como se comentó en la sección 5.3, se ha incorporado a una versión experimental de *EJS* un sistema de comunicación síncrona a través de Internet entre simulaciones creadas con este software. El sistema es un primer prototipo, cuyos resultados experimentales que se muestran a continuación, han sido satisfactorios.

En primer lugar, se probó el sistema de comunicación ejecutando las simulaciones sobre el mismo ordenador. Como vemos en la Figura 6.11, se puede observar la sincronización entre tres simulaciones de un tiro parabólico. La simulación más a la izquierda es la maestra (controles activos), mientras que las otras dos son las esclavas (controles desactivados). El campo numérico que se encuentra marcado es el tiempo de simulación, que como podemos observar es el mismo en todas.

Posteriormente, el sistema de comunicación se probó en una red local, con un número de tres ordenadores con simulaciones esclavas conectadas a un ordenador maestro. La sincronización resultó ser más rápida que realizándola sobre un mismo ordenador, ya que en este último caso, la sobrecarga de todos

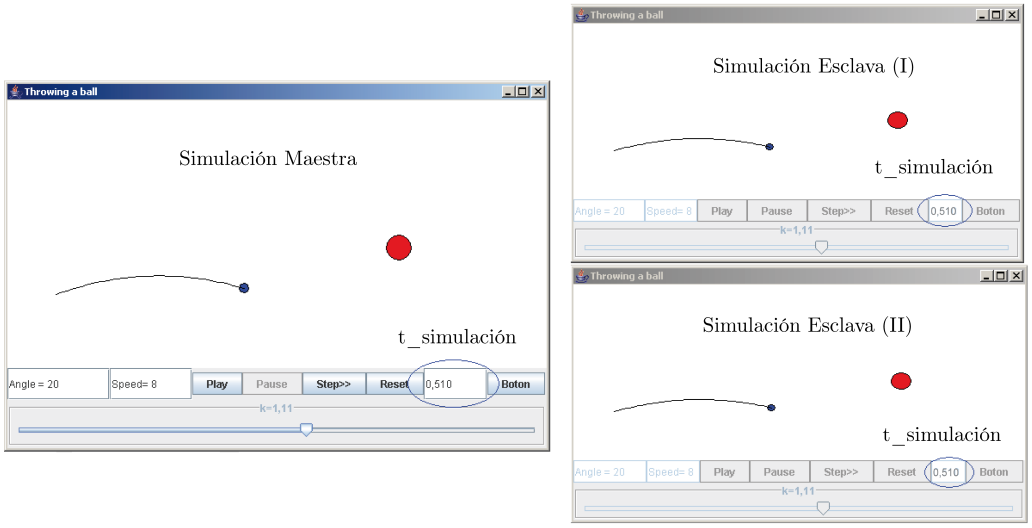

Figura 6.11: Simulaciones sincronizadas

los procesos de las simulaciones ejecutándose sobre la misma CPU ralentizaba la comunicación.

CAPÍTULO 7

Discusión y Conclusiones

Este trabajo se ha centrado fundamentalmente en el desarrollo de laboratorios virtuales y remotos en el campo de la robótica industrial. Las aplicaciones desarrolladas proporcionan entornos innovadores para la simulación y teleoperación de robots industriales. Están principalmente orientadas a la enseñanza y práctica de robótica, permitiendo al usuario aprender mediante el uso de dichos entornos virtuales y remotos.

Para el desarrollo de las aplicaciones se ha empleado *EJS*, software de distribución libre para la creación de simulaciones interactivas en *Java*. Dicho programa ha proporcionado las herramientas necesarias para la creación de las interfaces de usuario y de los entornos virtuales, para el desarrollo del modelo matemático del sistema a simular, y para la programación de las tareas de teleoperación.

Tal y como se ha comentado durante el desarrollo de este documento, la combinación de los laboratorios virtuales y los entornos colaborativos *on-line* es hoy en día una de las mejores alternativas para la educación a distancia. De este modo, y acogiéndonos a uno de los objetivos del trabajo ("Mejorar la interacción colaborativa *on-line* de este tipo de arquitecturas") se ha comenzado a desarrollar un sistema de comunicación síncrono entre simulaciones interactivas a través de *TCP Sockets* (capítulo 5). Además, se pretende incluir este sistema como una opción más del software *EJS* y de este modo, cualquier simulación que se cree con esta herramienta desde un ordenador conectado a Internet, pueda realizar una clase virtual.

7.1. Análisis de los resultados

Realizando una síntesis de este breve período de investigación, se puede deducir que el software escogido para el desarrollo de las aplicaciones presentadas ha sido el correcto. *EJS* ha permitido la creación de laboratorios virtuales y remotos de dispositivos robóticos en muy poco tiempo y sin necesidad de ser un experto programador en *Java*. Sus herramientas han proporcionado la capacidad suficiente para desarrollar entornos virtuales 3D con un gran realismo, además de poder implementar funciones de programación remota. Sus simulaciones pueden ejecutarse como *applets* desde cualquier explorador web, y además, pueden ser integradas con otras simulaciones *EJS* en entornos como *eMersion*.

Los laboratorios virtuales y remotos desarrollados se están utilizando como plataformas de aprendizaje a distancia de robótica industrial. Son aplicaciones que incorporan una interfaz muy sencilla apta para cualquier usuario, con simulaciones gráficas de dispositivos robóticos muy reales y con la posibilidad de experimentar con una gran variedad de conceptos de robótica (cinemática directa, inversa, planficación de trayectorias, dinámica, etc,...). Asimismo, resultan novedosas entre la gran lista de aplicaciones desarrolladas con *EJS*. Además, presentan una posible solución a los problemas existentes en los laboratorios experimentales universitarios (sección 2.1), ya que su empleo permite al alumno ser adiestrado de una forma efectiva y económica [Candelas y Moreno, 2005].

Con respecto a las resultados obtenidos del primer prototipo del sistema de comunicación entre simulaciones, se puede afirmar que han sido satisfactorios. Se ha conseguido una sincronización en tiempo real a través de la red de la evolución de una simulación *Java*. De esta manera, es posible realizar una clase virtual donde un grupo de alumnos, coordinados por un profesor, pueda experimentar con la misma simulación de forma *on-line*.

7.2. Trabajos Futuros

Actualmente existen diferentes líneas para la continuación del presente trabajo. Se han dividido en dos grandes grupos: las relacionadas en la mejora de los laboratorios virtuales y remotos, y las relacionadas con nuevas aportaciones al software *EJS*.

Con respecto a las primeras, se pretende desarrollar las siguientes ideas:

- Implementar la planificación de trayectorias en el espacio cartesiano. Para ello será necesario modelar el jacobiano del dispositivo robótico y de este modo, el movimiento del robot en la simulación podrá ser comandado por velocidades en el extremo del robot.

- Incluir la manipulación de los objetos insertados en la vista de la simulación. Así, el robot podrá realizar operaciones de agarre y traslado de objetos (*pick & place*).

- Implementar el modelo dinámico completo. Dada la complejidad de este trabajo, se recurrirá a la comunicación que posee *EJS* con softwares orientados al modelado de sistemas, tales como *Modelica* [Elmqvist y otros, 1999] o *Matlab*.

- Creación de herramientas para facilitar la programación de tareas de alto nivel del dispositivo robótico tanto en la simulación local como en la teleoperación. Un ejemplo de este tipo de herramientas puede ser el desarrollo de un módulo para el envío de órdenes por medio de comandos de voz.

- Desarrollo de un modelo de cámara para insertarlo en la vista de la aplicación. De este modo, se podrá simular la imagen de una cámara situada en el entorno virtual. Esta aportación podrá ser utilizada para realizar control visual en la simulación, programando el algoritmo de control a partir de las características obtenidas de la cámara virtual modelada. La trayectoria conseguida se enviaría al robot real a través de Internet.

- Creación de interfaces para el uso de dispositivos tales como *joysticks* u otros dispositivos de interacción háptica de bajo coste principalmente, para controlar la simulación y el robot remoto.

- Inclusión de objetos virtuales en el entorno remoto mediante realidad aumentada. De este modo, si el usuario introduce un objeto en la simulación, podrá ser visualizado superpuesto virtualmente en el laboratorio remoto.

- Reconocimiento 3D. Desarrollo de un módulo de reconocimiento de objetos 3D en el laboratorio remoto real y su inclusión como modelos virtuales en el entorno de la simulación.

- Creación de una *toolbox* de robótica en *Java* mediante la librería *vectmath.jar* para que no sea necesario utilizar *Matlab* y de este modo, facilitar la tarea de programación en la realización de simulaciones de robots.
- Creación de cursos HTML para el aprendizaje de robótica donde las simulaciones puedan descargarse en forma de *applets*.

Referente a las nuevas aportaciones al software *EJS*, se pretende:

- Mejorar el sistema de comunicación síncrono entre las simulaciones, ya que uno de los inconvenientes que presenta es que el alumno conectado a una clase virtual necesita poseer el archivo de la simulación y las librerías para su ejecución. Por tanto, se pretende que el sistema de comunicación genere una *URL* con un *applet* que contenga la simulación sincronizada. De este modo, el alumno se conectará a esta dirección web sin necesidad de poseer en su PC la simulación ni las librerías para su ejecución.
- Desarrollar una nueva interfaz gráfica para *EJS*. Tal y como se ha comentado en la memoria, actualmente *EJS* sólo permite utilizar dos tipos de paneles. El *Simple 3D*, cuyos objetos provienen de una librería gráfica incorporada en *EJS* y el *Display 3D*, que carga objetos de la API de *Java 3D*. La primera biblioteca funciona correctamente, pero los gráficos que genera son muy simples y con escasa calidad gráfica. Respecto al panel *Display 3D*, su calidad es mayor, sin embargo suele dar muchos problemas en la generación del código y provocando errores de ejecución. Además, para ambos paneles sólo es posible introducir objetos prediseñados, tales como cubos, cilindros, conos y esferas. Por lo tanto, se pretende la creación de un nuevo panel que pueda cargar objetos *VRML*. Los gráficos en dicho formato poseen una mayor calidad gráfica que los existentes en *EJS*. Además, mediante esta nueva aportación se podrá cargar objetos de cualquier forma y dimensión, siempre y cuando su formato sea *VRML*.
- Incluir una librería en *EJS* de dispositivos para el modelado de entornos robotizados en la vista de la simulación. Es decir, al igual que existen elementos como partículas, vectores, cilindros, etc,... se pretende crear una librería de objetos para construir celdas de trabajo para los robots, como por ejemplo cámaras, sensores, herramientas de manipulación para el robot,etc,....

7.3. Publicaciones

Como resultado de este periodo de investigación (Octubre 2006-Junio 2007), se han obtenido las siguientes publicaciones:

Congresos Internacionales

"Flexible virtual and remote laboratory for teaching Robotics"
Francisco A.Candelas, Fernando Torres, Carlos A.Jara
IV International Conference on Multimedia and Information & Communication Technologies in Education
m-ICTE 2006

Este artículo trata sobre el laboratorio virtual y remoto *Robolab*. También se comentan brevemente los nuevos laboratorios virtuales desarrollados con *EJS*.

"Practical Training of Robotics Concepts Using Interactive Tools"
Carlos A.Jara, Francisco A.Candelas, Fernando Torres
IFAC International Workshop on Intelligent Assembly and Disassembly
IAD'07

Este artículo describe en su totalidad el laboratorio virtual visto en la sección 6.1 y lo enfoca como una posible futura herramienta para la simulación de operaciones de ensamblado y desensamblado.

Congresos nacionales

"Comunicación síncrona de simulaciones interactivas desarrolladas con *Easy Java Simulations*"
Carlos A.Jara, Francisco A.Candelas, Fernando Torres
II Congreso Nacional de Informática
V Jornadas de Enseñanza a través de Internet/Web de la Ingeniería de Sistemas y Automática
EIWISA'2007

Este artículo describe el sistema de comunicación síncrono entre simulaciones *EJS* visto en el capítulo 5. Como ya se comentó, este sistema es sólo un primer prototipo, por lo que se pretende mejorar e incluirlo definitivamente en *EJS*.

"Herramientas interactivas para la enseñanza de robótica"
Carlos A.Jara, Francisco A.Candelas, Fernando Torres
II Congreso Nacional de Informática
V Jornadas de Enseñanza a través de Internet/Web de la Ingeniería de Sistemas y Automática
EIWISA'2007

Este artículo expone las herramientas desarrolladas con *EJS* realizadas durante el período de investigación. Describe tanto el laboratorio virtual visto en la sección 6.1, como el remoto de la sección 6.2.

Revistas (en proceso de revisión)

"Virtual and Remote Laboratory for Robotics e-Learning"

Carlos A.Jara, Francisco A.Candelas, Fernando Torres
IEEE Transactions on Industrial Electronics
e-Learning and Remote Laboratories within Engineering Education

Este artículo trata sobre el laboratorio remoto descrito en la sección 6.2. Además de lo visto en esta sección, se incluyeron una serie de resultados experimentales sobre los retrasos en la transmisión del sistema a través del protocolo HTTP y sobre la similitud entre las trayectorias reales del robot y las simuladas por la aplicación.

APÉNDICE A

Manual de usuario de *Easy Java Simulations*

A.1. ¿Qué es *Easy Java Simulations*?

Easy Java Simulations (*EJS*) es un entorno de simulación que ha sido diseñado y desarrollado por el profesor Francisco Esquembre[1] .Este entorno se ha ideado para el desarrollo de aplicaciones docentes, permitiendo a profesores y alumnos crear de forma sencilla sus propios laboratorios virtuales, sin que para ello requieran de conocimientos avanzados de programación.

El entorno de simulación de *EJS*, así como su documentación y algunos casos de estudio, puede ser descargado gratuitamente de la página web de *EJS*: `http://fem.um.es/Ejs`. Este pequeño manual ha sido extraído de la información que nos proporciona dicha dirección web.

A.2. Paradigma Modelo-Vista-Control

La metodología para la creación de laboratorios virtuales de *EJS* está basada en una simplificación del paradigma "modelo-vista-control" (M-V-C). Este paradigma establece que el laboratorio virtual se compone de las tres partes siguientes:

[1]Profesor Dr. Francisco Esquembre, Dpto. de Matemáticas, Universidad de Murcia

1. El modelo: describe los fenómenos bajo estudio. Está compuesto por un conjunto de variables y por las relaciones entre estas variables.
2. El control: define las acciones que el usuario puede realizar sobre la simulación.
3. La vista: representación gráfica de los aspectos más relevantes del fenómeno simulado.

Estas tres partes están interrelacionadas entre sí:

- El modelo afecta a la vista, ya que debe mostrarse al usuario cuál es la evolución del valor de las variables del modelo.
- El control afecta al modelo, ya que las acciones ejercidas por el usuario pueden modificar el valor de las variables del modelo.
- La vista afecta al modelo y al control, ya que la interfaz gráfica puede contener elementos que permitan al usuario modificar el valor de las variables o realizar ciertas acciones.

Simplificación del paradigma M-V-C realizada por *EJS*

EJS se basa en una simplificación del paradigma M-V-C, suprimiendo la parte del control como tal, e integrando sus funciones tanto en la vista como en el modelo. Esta simplificación se basa en el hecho de que el usuario puede usar la interfaz gráfica del laboratorio virtual (es decir, la vista) para interaccionar con la simulación, empleando para ello el ratón, el teclado, etc. Así pues, en un laboratorio virtual programado usando *EJS*, el usuario interacciona con el modelo a través de la vista. Por tanto, al programar el modelo es preciso especificar de qué forma las acciones realizadas por el usuario durante la simulación sobre los componentes gráficos o los controles de la vista afectan al valor de las variables del modelo. Las propiedades de los elementos gráficos de la vista pueden asociarse con las variables del modelo, dando lugar a un flujo de información bidireccional entre la vista y el modelo. Cualquier cambio en el valor de una variable del modelo es automáticamente representado en la vista. Recíprocamente, cualquier interacción del usuario con la vista de laboratorio virtual, modifica el valor de la correspondiente variable del modelo.

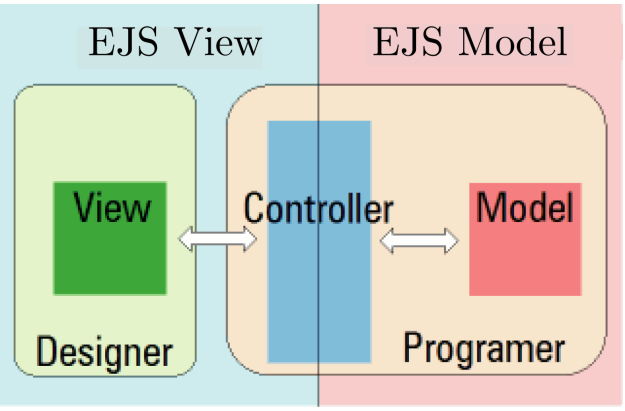

Figura A.1: Paradigma simplificado de *EJS*

Además del modelo y la vista, *EJS* permite incluir en el laboratorio virtual páginas HTML que realicen las funciones de documentación, informando acerca de la finalidad del laboratorio, sus instrucciones de uso, recomendaciones pedagógicas, etc. Este conjunto de páginas recibe el nombre de Introducción.

Resumiendo lo anterior, la definición de un laboratorio virtual mediante *EJS* se estructura en las siguientes tres partes:

- Introducción: páginas html que incluyen los contenidos educativos relacionados con el laboratorio virtual.
- Modelo: modelo dinámico cuya simulación interactiva es la base del laboratorio virtual.
- Vista: interfaz entre el usuario y el modelo. La vista del laboratorio virtual tiene dos funciones. Por una parte, proporciona una representación visual del comportamiento dinámico del modelo. Por otra parte, proporciona los mecanismos para que el usuario pueda interaccionar con el modelo durante la simulación.

En la Figura A.2 se muestra la interfaz de usuario de *EJS*. Se trata de la pantalla que aparece al arrancar *EJS*. Como puede verse, en la parte superior hay tres botones: Introducción, Modelo y Vista. Mediante estos botones puede seleccionarse el panel para la definición de las páginas de la introducción, el panel para la definición del modelo o el panel para la definición de la vista.

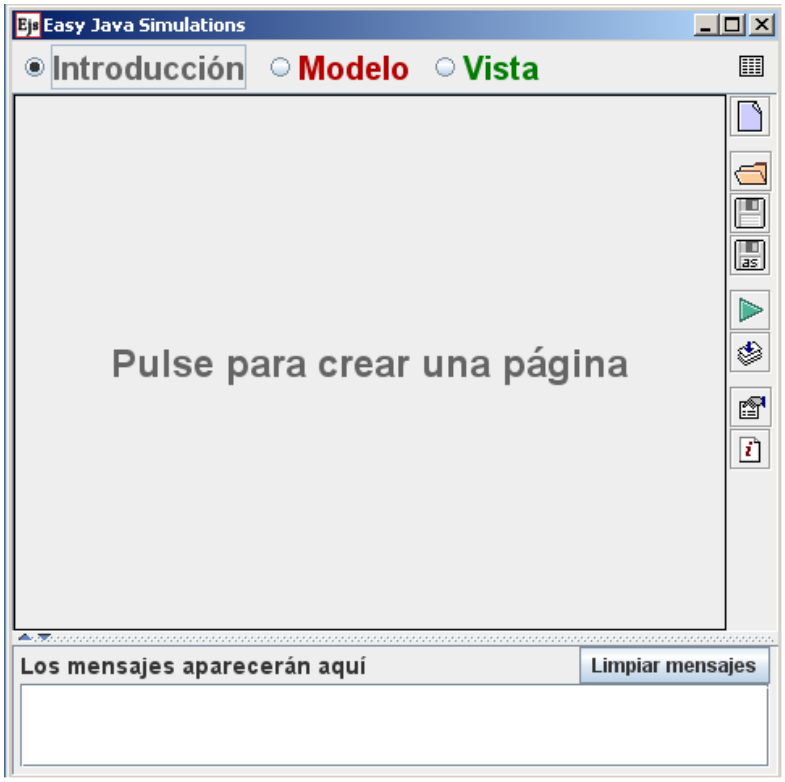

Figura A.2: Interfaz de usuario de *EJS*

A.3. Definición del modelo

Para definir un modelo interactivo en *EJS*, es preciso proporcionar la información siguiente:

- Declarar las variables que intervienen en el modelo.
- Describir los algoritmos necesarios para calcular el valor de las variables: en el instante inicial de la simulación, en función del tiempo y cuando el usuario realiza sobre la vista del laboratorio virtual alguna acción interactiva.

EJS proporciona un procedimiento sencillo para introducir esta información. En concreto, la definición del modelo en *EJS* se compone de las partes siguientes:

- La inicialización de las variables realizada al declararlas. A tal fin, la

expresión cuyo valor debe asignarse a la variable debe escribirse en la columna Valor del panel *Variables*, en la fila correspondiente a la declaración de la variable.

- Los algoritmos para la inicialización de las variables, que se definen en el panel *Inicialización*.
- Los algoritmos escritos en el panel *Evolución*.
- Los algoritmos escritos en el panel *Ligaduras*.
- Los métodos definidos en el panel *Propio*, que pueden ser invocados desde cualquier punto de la descripción del modelo o de la vista. Se emplean típicamente para describir las acciones interactivas del usuario.

En la siguiente imagen podemos ver el entorno de *EJS* con sus respectivos paneles para la definición del modelo.

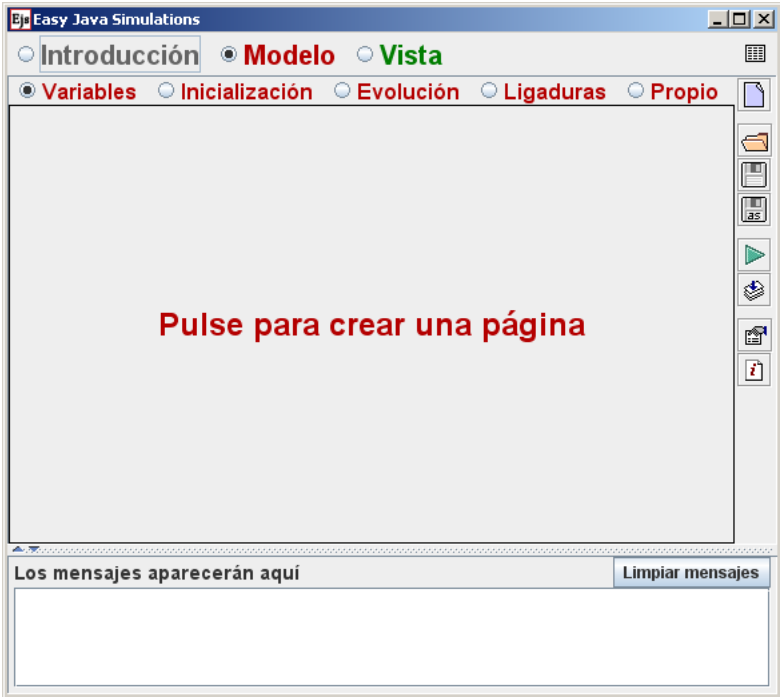

Figura A.3: Definición del modelo en *EJS*

Conceptos de modelado de sistemas

Los modelos matemáticos están compuestos por ecuaciones, que describen la relación entre las magnitudes relevantes del sistema. Estas magnitudes reciben el nombre de variables. El punto de partida para la simulación del modelo consiste en clasificar sus variables de acuerdo al criterio siguiente:

- Parámetros. Son aquellas variables cuyo valor permanece constante durante la simulación.
- Variables de estado. Son las variables que están derivadas respecto al tiempo.
- Variables algebraicas. Son las restantes variables del modelo. Es decir, aquellas que no aparecen derivadas en el modelo y que no son constantes.

En la Figura A.4 se muestra un algoritmo para la simulación de modelos de tiempo continuo. Puede comprobarse que la clasificación de las variables del modelo en parámetros, variables de estado y variables algebraicas constituye la base para la simulación del modelo:

- Al comenzar la simulación, se asignan valores a los parámetros. Estos valores permanecen constantes durante toda la simulación.
- Las variables de estado son calculadas mediante la integración numérica de sus derivadas. Por ejemplo, la función de paso del método explícito de Euler para la ecuación diferencial ordinaria

  $$\frac{dx}{dt} = f(x,t)$$

  es la siguiente:

  $$x_{i+1} = x_i + f(x_i, ti) \cdot \Delta t$$

  donde x_i y x_{i+1} representan el valor de la variable de estado x en los instantes t_i y $t_i + \Delta t$ respectivamente, y $f(x_i, t_i)$ representa el valor de la derivada de x (es decir, $\frac{dx}{dt}$) en el instante t_i.
- El valor de las variables algebraicas y de las derivadas de las variables de estado (variables desconocidas del modelo) se calculan, en cada instante de tiempo, a partir de valor de las variables de estado en ese instante y del valor de los parámetros.

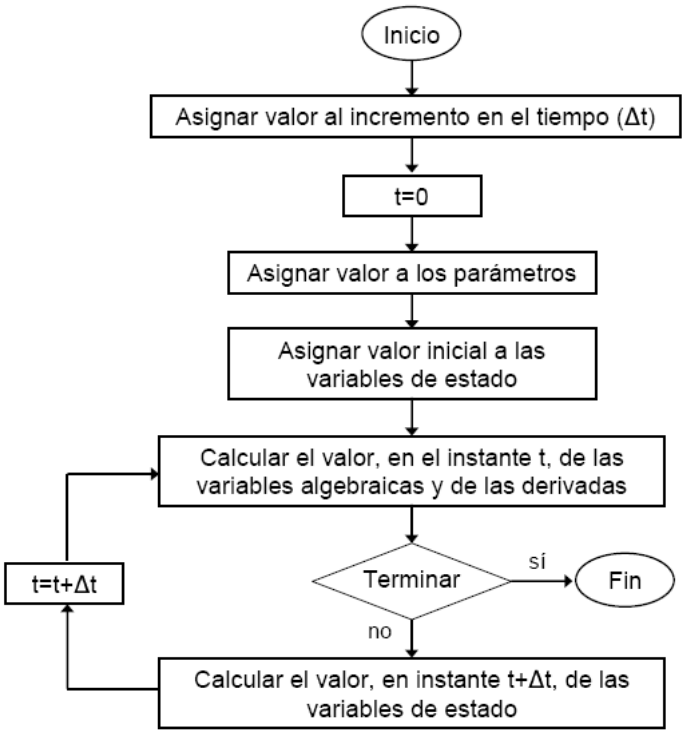

Figura A.4: Algoritmo de simulación

La condición de terminación de la simulación depende del estudio en concreto que vaya a realizarse sobre el modelo. Puede ser, por ejemplo, que se alcance determinado valor de la variable tiempo, o que una determinada variable satisfaga cierta condición.

El valor del tamaño del paso de integración (Δt) debe escogerse alcanzando un compromiso entre precisión y carga computacional. Cuanto menor sea el valor de Δt, menor es el error que se comete en el cálculo de las variables del modelo, pero mayor es el tiempo de ejecución de la simulación.

Para realizar el cálculo de las variables algebraicas y las derivadas, es preciso despejar de cada ecuación la variable a calcular y ordenar las ecuaciones, de modo que sea posible resolverlas en secuencia. Para plantear el algoritmo de la simulación de un modelo, es preciso realizar las tareas siguientes:

- Decidir qué variable debe calcularse de cada ecuación y cómo deben ordenarse las ecuaciones del modelo, de modo que puedan ser resueltas en secuencia. A esta decisión se la denomina *asignación de la causalidad*

computacional.

- Una vez se ha decidido qué variable debe evaluarse de cada ecuación, debe manipularse simbólicamente la ecuación a fin de despejar dicha variable.

Descripción algorítmica del modelo

El modelo se describe en *EJS* mediante fragmentos de código en lenguaje Java, los cuales deben ser escritos en los diferentes paneles que proporciona *EJS* para la descripción del modelo: *Inicialización*, *Evolución*, *Ligaduras* y *Propio*. Esto implica que el modelo matemático debe ser manipulado por el usuario, previamente a escribirlo en los paneles de *EJS*, con el fin de formularlo como una secuencia ordenada de asignaciones. Para ello, es conveniente asignarle una causalidad computacional al modelo.

El algoritmo de simulación de *EJS*

Para poder entender cómo estructurar la definición del modelo en los diferentes paneles, es preciso previamente comprender el algoritmo de simulación de *EJS*. En este contexto, se entiende por algoritmo de simulación de *EJS* el orden en el que *EJS* ejecuta los diferentes paneles y las diferentes ventanas dentro de cada panel. El orden es el siguiente:

1. *EJS* ejecuta los diferentes paneles que componen la descripción del modelo en el orden mostrado en la Figura A.5.

2. Si un determinado panel consta de varias páginas, éstas se ejecutan siguiendo el orden relativo en que están dispuestas, empezando por la que está situada más a la izquierda y terminando con la que está más a la derecha.

3. *EJS* ejecuta los algoritmos de una página siguiendo el orden en que están escritos, comenzando por la parte superior de la página y finalizando por su parte inferior, de forma completamente análoga a como si se tratara de la ejecución de un fragmento de código de un programa escrito en Java.

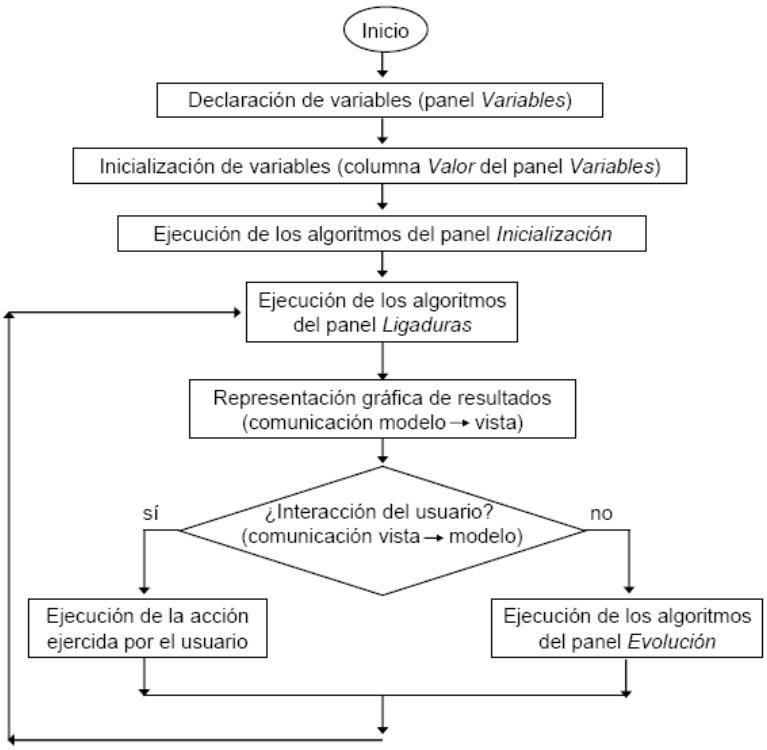

Figura A.5: Algoritmo de simulación de *EJS*

En la Figura A.5 se muestra el algoritmo de simulación de *EJS*, es decir, la secuencia de tareas que realiza *EJS* para ejecutar cualquier laboratorio virtual. Éstas son las siguientes:

1. *EJS* crea las variables y les asigna sus correspondientes valores de inicialización. Para ello, primero realiza las asignaciones descritas en el panel *Variables* y a continuación ejecuta los algoritmos descritos en el panel *Inicialización*.

2. *EJS* ejecuta los algoritmos contenidos en el panel *Ligaduras*.

3. *EJS* crea la vista de la simulación y la muestra en pantalla, estableciendo las asociaciones entre las propiedades de los elementos gráficos y las variables del modelo, así como la forma en la que debe reaccionar ante las acciones del usuario sobre el modelo. En este punto, el modelo se encuentra en su estado inicial, y la vista refleja los valores de las variables en este estado.

4. *EJS* examina si el usuario ha interaccionado con la vista. Pueden darse dos situaciones:

   a) Si el usuario interacciona con la vista, *EJS* ejecuta los algoritmos asociados a dicha interacción y, a continuación, ejecuta los algoritmos del panel *Ligaduras*.

   b) Si el usuario no interacciona con la vista, se ejecutan los algoritmos del panel *Evolución* y seguidamente los del panel *Ligaduras*.

5. *EJS* representa en la vista del laboratorio el nuevo estado del modelo que acaba de evaluarse en el paso anterior.

6. *EJS* salta al Paso 4.

Ejemplo. Vamos a realizar el algoritmo de simulación de *EJS* del circuito eléctrico mostrado en la Figura A.6, compuesto por un generador de tensión, dos resistencias y un condensador (Circuito RC). El modelo de este circuito consta de las ecuaciones siguientes:

$$\begin{aligned} u &= u_0 \cdot \sin(w \cdot t) \\ i_{R1} &= i_{R2} + i_C \\ u - u_C &= R_1 \cdot i_{R1} \\ C \cdot \frac{du_c}{dt} &= i_C \\ u_C &= i_{R2} \cdot R_2 \end{aligned}$$

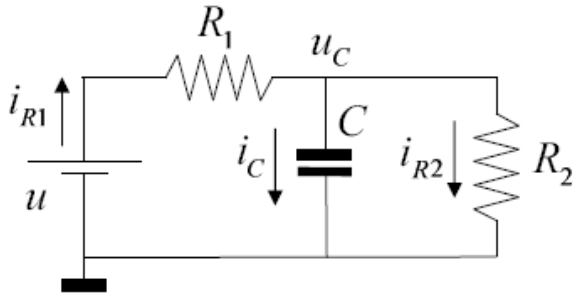

Figura A.6: Circuito RC

La variable u_C aparece derivada, con lo que es una variable de estado del sistema. Con el fin de realizar la asignación de la causalidad computacional, se sustituye en el modelo $\frac{du_c}{dt}$ por la variable auxiliar $deru_c$. Realizando esta sustitución, se obtiene el modelo siguiente:

$$\begin{aligned} u &= u_0 \cdot \sin(w \cdot t) \\ i_{R1} &= i_{R2} + i_C \\ u - u_C &= R_1 \cdot i_{R1} \\ C \cdot deru_C &= i_C \\ u_C &= i_{R2} \cdot R_2 \end{aligned}$$

Y asignándole una causalidad computacional (decidir en qué orden deben evaluarse las ecuaciones y qué incógnita debe evaluarse de cada ecuación) a fin de calcular las incógnitas u, i_{R1}, i_{R2}, i_C, $deru_C$, obtenemos el siguiente modelo ordenado y resuelto:

$$\begin{aligned} [u] &= u_0 \cdot \sin(w \cdot t) \\ [i_{R2}] &= \frac{u_C}{R_2} \\ [i_{R1}] &= \frac{u - u_C}{R_1} \\ [i_C] &= i_{R1} - i_{R2} \\ [deru_C] &= \frac{i_C}{C} \\ \frac{d\,[u_C]}{dt} &= deru_C \end{aligned}$$

EJS proporciona al usuario métodos de integración para el cálculo de las variables de estado. Para ello, es preciso:

- Expresar la derivada de cada una de las variables de estado en función únicamente de variables de estado, parámetros y la variable tiempo. Puesto que se ha asignado la causalidad computacional del modelo,

las manipulaciones necesarias para ello se pueden realizar de manera sencilla. En este ejemplo:

$$\frac{du_C}{dt} = deru_C = \frac{i_C}{C} = \frac{i_{R1} - i_{R2}}{C} = \frac{\frac{u-u_C}{R1} - \frac{u_C}{R2}}{C} = \frac{\frac{u_0 \cdot \sin(w \cdot t) - u_C}{R1} - \frac{u_C}{R2}}{C}$$

Finalmente, la expresión para calcular u_C es:

$$\frac{d\left[u_C\right]}{dt} = \frac{\frac{u_0 \cdot \sin(w \cdot t) - u_C}{R1} - \frac{u_C}{R2}}{C}$$

- Deben escribirse las expresiones calculadas anteriormente para la derivada de las variables de estado en una página EDO (*Ecuación Diferencial Ordinaria*), dentro del panel *Evolución*. En este ejemplo, habrá que escribir la ecuación anterior en una página EDO del panel *Evolución*.

El algoritmo de la simulación se muestra en la Figura A.7 . Se puede observar que:

- Las asignaciones que permiten calcular las variables algebraicas y las derivadas de las variables de estado, se incluyen en el panel *Ligaduras*.
- Es preciso asignar la causalidad computacional a las ecuaciones del modelo para saber cómo deben ordenarse dentro del panel *Ligaduras* y qué variable debe despejarse de cada ecuación.
- El cálculo de las variables de estado, mediante integración de sus derivadas, debe incluirse en el panel *Evolución*. Si se desea emplear uno de los métodos de integración de *EJS*, debe emplearse una página EDO y expresar la derivada de cada variable de estado en función únicamente de variables de estado, parámetros y la variable tiempo.
- Cuando se emplea una página EDO, *EJS* gestiona automáticamente el incremento de la variable tiempo a lo largo de la simulación.

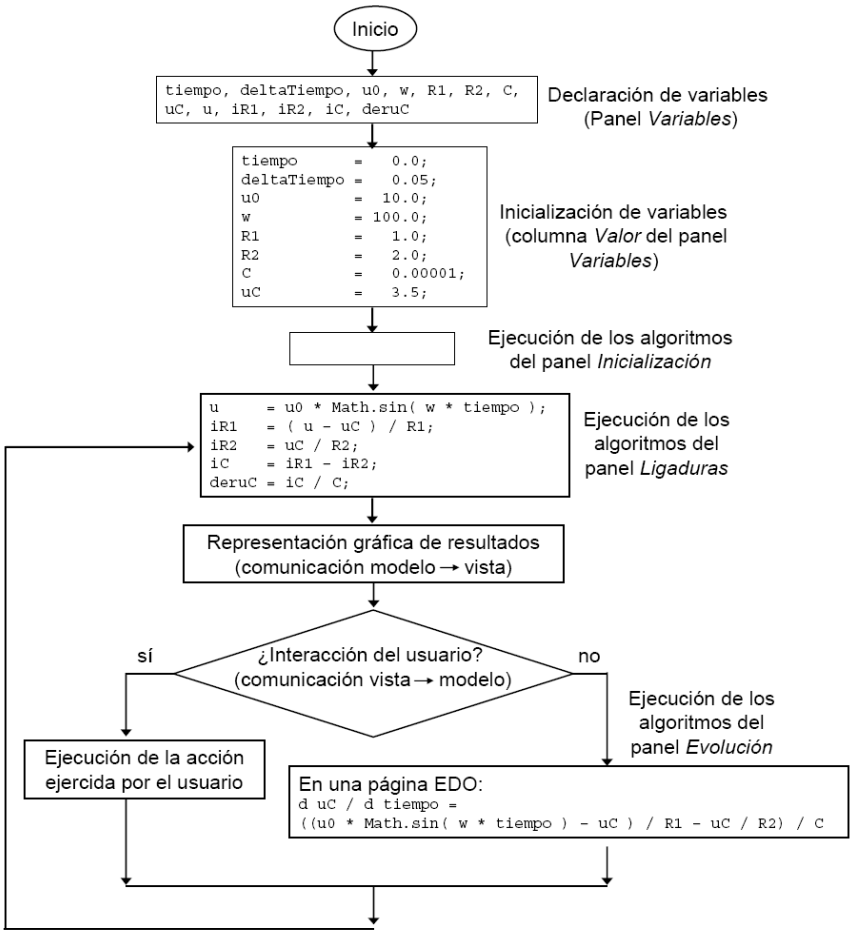

Figura A.7: Algoritmo de simulación de *EJS* del modelo del circuito RC

Declaración e inicialización de las variables

Es preciso declarar y asignar valor inicial a los siguientes tipos de variables del modelo:

- La variable tiempo, que típicamente se inicializará al valor cero.
- Las constantes y parámetros.
- Las variables de estado.

Es preciso declarar, pero no inicializar, los siguientes tipos de variables:

- Las variables algebraicas.
- Las derivadas de las variables de estado (variables auxiliares que se han introducido al analizar la causalidad computacional del modelo).

La inicialización de las variables se realiza en dos paneles: *Variables* e *Inicialización* (vea la Figura A.5):

- En primer lugar, *EJS* ejecuta las asignaciones que el usuario haya escrito en el panel *Variables*. Aquí, el usuario puede declarar variables del tipo *boolean*, *int*, *double*, *String* y *Object* con su respectiva dimensión y su valor inicial.
- A continuación, *EJS* ejecuta los algoritmos que el usuario haya definido en el panel *Inicialización*. En este panel el usuario puede introducir código Java para realizar el cálculo del valor inicial de las variables.

Descripción de la evolución

En la Figura A.8 muestra el panel *Evolución*. Observe que el panel se encuentra dividido en las tres partes siguientes:

- En la parte superior derecha, hay un subpanel con el letrero *Pulse* para crear una página. Pulsando se crea una página en la cual el usuario puede escribir los algoritmos que desee, por ejemplo, puede programar sus propios métodos de integración para calcular las variables de estado.
- En la parte inferior derecha, hay un subpanel con el letrero *Pulse* para crear una página de EDO. Esta página es un asistente para la definición de ecuaciones diferenciales ordinarias. *EJS* genera automáticamente el código para integrar numéricamente las EDO definidas en esta página. *EJS* soporta cuatro métodos de integración, entre los cuales el usuario puede escoger:
  - Euler.
  - Punto medio (Euler-Richardson).
  - Runge-Kutta (4º orden).
  - Runge-Kutta-Fehlberg (4º-5º orden).

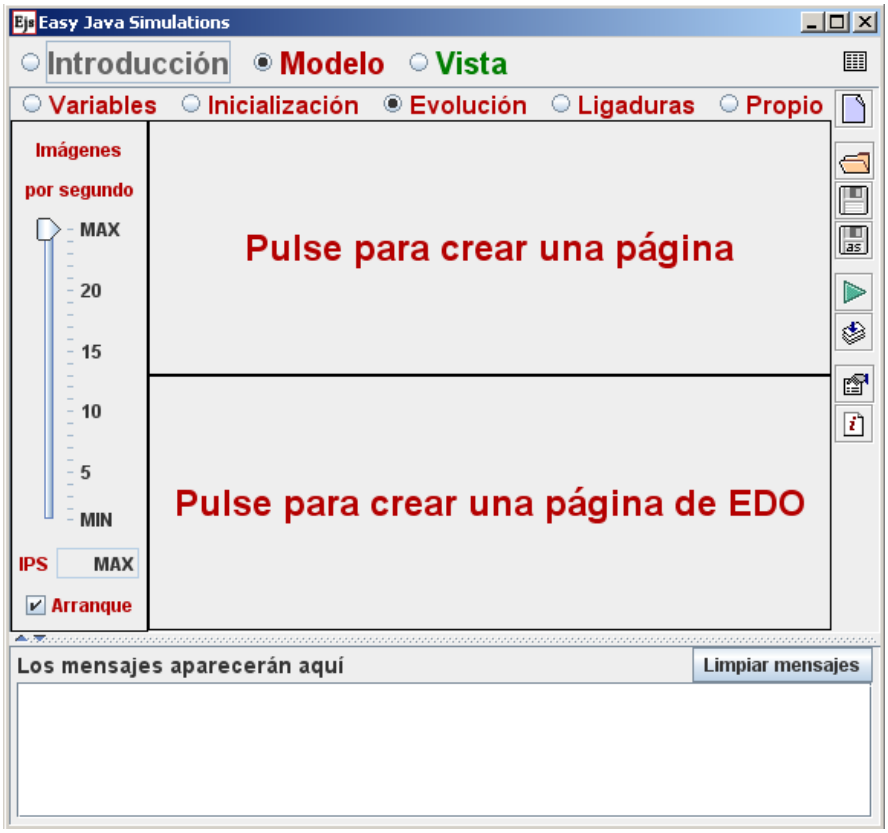

Figura A.8: Panel de *Evolución* de *EJS*

El código que genera *EJS* para una página EDO realiza las dos tareas siguientes:

1. Calcula el valor de las variables de estado, aplicando para ello el algoritmo correspondiente al método de integración que ha seleccionado el usuario. Si el valor actual de la variable tiempo es t, y el tamaño del paso de integración es Δt, este valor calculado de las variables de estado es el valor de las mismas en el instante de tiempo $t + \Delta t$.
2. Incrementa el valor de la variable tiempo en Δt.

Puesto que el código generado por *EJS* para una página EDO incrementa el valor de la variable tiempo, en un laboratorio virtual no puede haber más de una página de EDO. Igualmente, si el laboratorio no tiene ninguna página EDO, el usuario debe gestionar por sí mismo el incremento de la variable tiempo.

- En la parte izquierda, hay un subpanel que contiene el letrero Imágenes por segundo (*IPS*), debajo del cual hay un indicador que el usuario puede desplazar verticalmente arrastrándolo con el ratón. Con ello puede regular la velocidad de ejecución de la simulación. En concreto, determina el número de pasos en el tiempo que avanzará la simulación en un segundo de tiempo real. Cuanto mayor sea el número de imágenes por segundo, mayor será la velocidad con que se visualizará la evolución del sistema. En la parte inferior se encuentra el botón Arranque. Cuando está activado, la simulación comienza tan pronto como es ejecutado el laboratorio virtual. Si está desactivado la simulación no comenzará cuando se ejecute el laboratorio virtual. La utilidad de ello es permitir que el usuario realice ciertas tareas antes de que comience la simulación (por ejemplo, cambiar el valor de ciertas variables). En este caso, es preciso definir un botón de arranque (en la vista del laboratorio virtual) que el usuario deberá pulsar para que se inicie la simulación. Este botón de arranque deberá tener asociada como acción una llamada al método *play()*. Se trata de un método predefinido de *EJS* que hace que comience la ejecución de la simulación.

Descripción de las ligaduras

En general, en el panel *Ligaduras* se escriben los algoritmos para el cálculo de las variables algebraicas del modelo. Como puede verse en la Figura A.5, las ecuaciones de ligadura son aquellas que describen el comportamiento del modelo, no sólo durante su evolución en el tiempo, sino también en el caso de que el usuario realice cualquier cambio interactivo en el valor de alguna variable. Esta es una de las diferencias conceptuales entre las ecuaciones de evolución y las ecuaciones de ligadura.

Métodos propios del usuario

El usuario puede definir, en el panel *Propio*, todos aquellos métodos en lenguaje Java que precise para la definición del modelo o de la vista. Típicamente, los métodos se emplean para definir acciones sobre el modelo que son activadas desde la vista (por ejemplo, cuando el usuario pulsa un botón).

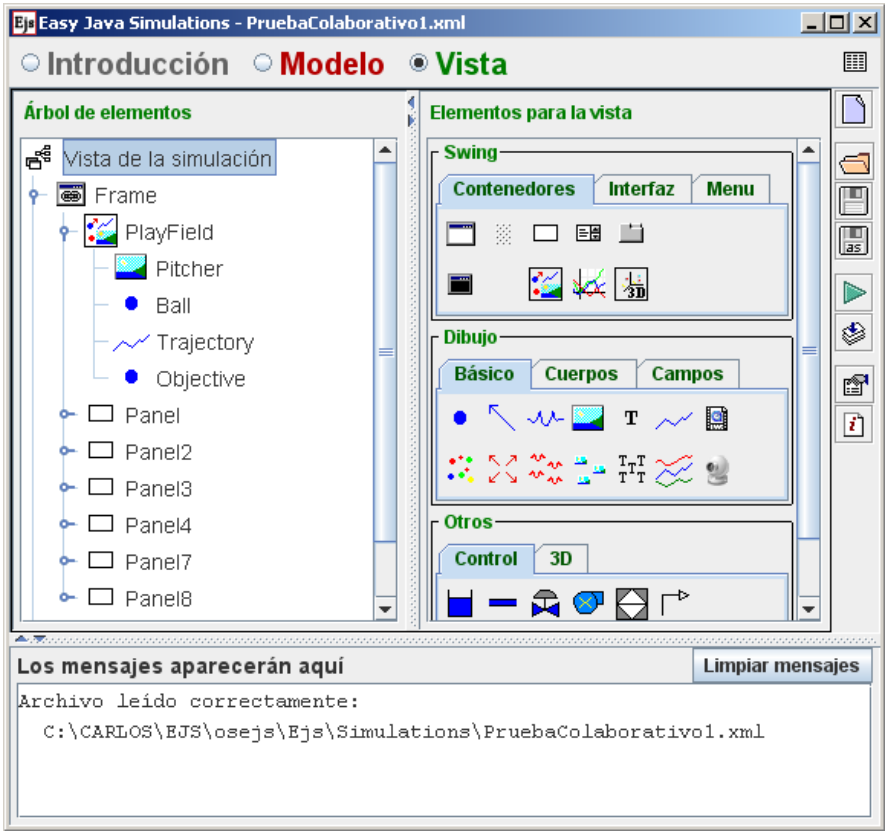

Figura A.9: Panel de *Vista* de *EJS*

A.4. Definición de la vista

La vista del laboratorio virtual es la interfaz entre el usuario y el modelo (ver Figura A.9). Mediante la manipulación de los controles de la vista, el usuario puede:

- Controlar la ejecución de la simulación, por ejemplo, deteniéndola o reiniciándola.
- Cambiar el valor de los parámetros, de las variables de entrada y de las variables de estado del modelo.

La definición de la vista de un laboratorio virtual se realiza en el panel *Vista*. Este panel se subdivide en dos paneles: el panel Árbol de elementos y el panel Elementos para la vista. El panel Elementos para la vista se divide

en tres áreas, donde se encuentran los iconos de todas las clases que se pueden usar para componer la vista. Estas tres áreas se denominan Swing, Dibujos y Otros. A su vez, Swing se divide en Contenedores (elementos que pueden encapsular otros elementos), Interfaz (botones, campos numéricos, campos de texto, ...) y Menú (elementos para crear menús). Dibujos se subdivide en Básicos (partículas, vectores,...) Cuerpos (cubos, esferas, ...) y Campos (vectoriales, gaussianos,...). Y Otros se subdivide en Control (bombas, tuberías,...) y 3D (elementos Java 3D).

El panel Árbol de elementos muestra la composición de la vista del laboratorio virtual. La vista está compuesta por elementos gráficos organizados formando una estructura en árbol.

Hay dos tipos de elementos que se pueden añadir a la vista: contenedor y control. El contenedor (elemento padre) se encarga de encapsular a los controles que tenga asociados (controles hijos) y el control es un elemento que se añade al contenedor. Para poder crear una interfaz, el primer paso es añadir un contenedor tipo *Ventana* al árbol de elementos. Ya teniendo el contenedor principal en la vista, el usuario puede añadir los contenedores y controles que crea oportuno.

Dado que sería muy extenso explicar todos los controles del panel de vista, al final de este capítulo se expondrá un ejemplo de la cómo realizar una simulación (modelo y vista) con *EJS*. Aquí se podrá ver cómo se utilizan algunos controles que se han nombrado en este apartado.

A.5. Ejecución y distribución del laboratorio virtual

Una vez que el usuario ha definido el modelo, la vista y la introducción del laboratorio virtual, *EJS* genera automáticamente el código Java del programa, lo compila, empaqueta los ficheros resultantes en un fichero comprimido, y genera páginas HTML que contienen la introducción y la simulación como un *applet*. Entonces, existen tres posibles formas de ejecutar el laboratorio virtual:

- Como un *applet*, abriendo con un navegador web (Internet Explorer, Netscape, etc.) el documento HTML generado por *EJS* para el laboratorio. Esta opción permite publicar el laboratorio virtual en Internet.

Una vez han sido generadas por *EJS*, se puede editar las páginas HTML del laboratorio virtual y añadirles otros contenidos. Deberá usar para ello un editor de páginas web. Es importante tener en cuenta que, por motivos de seguridad, los *applets* de Java no pueden escribir datos en el disco (pero si pueden leer datos del disco o de Internet). Por ello, si ha programado la simulación de modo que escriba datos en el disco, no podrá ejecutarla como un *applet*, sino que deberá usar cualquiera de las dos formas de ejecución siguientes.

- Ejecución desde el entorno *EJS*.
- Ejecución como una aplicación Java independiente.

A.6. Ejemplo de aplicación

Tiro parabólico

En este apartado vamos a simular el movimiento de un proyectil en el plano XY, lanzado con cierta velocidad inicial y sujeto a la aceleración de la gravedad.

Ecuaciones del movimiento

La trayectoria del proyectil se describe mediante la composición de dos movimientos independientes: un movimiento uniforme en la dirección X, con velocidad constante v_x y un movimiento uniformemente acelerado en la dirección Y, con velocidad inicial $v_y(0)$ y aceleración constante (debida a la gravedad) $-g$, con $g = 9{,}8 m/s^2$. Los valores iniciales de la velocidad dependen de la celeridad del lanzamiento c y del ángulo α de éste respecto de la horizontal de la siguiente forma:

$$\begin{aligned} v_x &= c \cdot \cos(\alpha) \\ v_y(0) &= c \cdot \sin(\alpha) \end{aligned}$$

La posición de la pelota en función del tiempo en la dirección X viene dada por la fórmula:

$$x(t) = x(0) + v_x \cdot t$$

mientras que la posición y la velocidad en el eje Y varían de según las siguientes fórmulas:

$$\begin{aligned} y(t) &= y(0) + v_y(0) \cdot t - \frac{1}{2} \cdot g \cdot t^2 \\ v_y(t) &= v_y(0) - g \cdot t \end{aligned}$$

Por tanto, comparando los valores de estas variables en un instante t y en el instante posterior $t + dt$, donde dt es el intervalo de tiempo que transcurre en cada paso de la evolución de la simulación, obtenemos las siguientes ecuaciones de movimiento:

$$x(t + dt) = x(t) + v_x \cdot dt \tag{A.1}$$

$$y(t + dt) = y(t) + v_y \cdot dt - \frac{1}{2} \cdot g \cdot dt^2 \tag{A.2}$$

$$v_y(t + dt) = v_t(t) - g \cdot dt \tag{A.3}$$

Variables

Las variables que caracterizan el fenómeno son la aceleración de la gravedad, la celeridad del lanzamiento y su ángulo. Añadiendo a éstas las variables cinemáticas del proceso, nos queda la siguiente tabla:

Figura A.10: Variables del modelo

La variable *connect* se utiliza para desactivar momentáneamente la traza que dibuja la trayectoria del proyectil al cambiar el ángulo de inclinación.

Inicialización

Para inicializar correctamente las variables creamos una página de inicialización. En esta página, se iniciarán de nuevo los valores de x e y para poder reiniciar el sistema al cambiar el ángulo sin pasar por la tabla de variables (que pondría el valor del ángulo a 20 grados otra vez). Además, transformamos el valor de *angle* a radianes para poderlo utilizar en las ecuaciones.

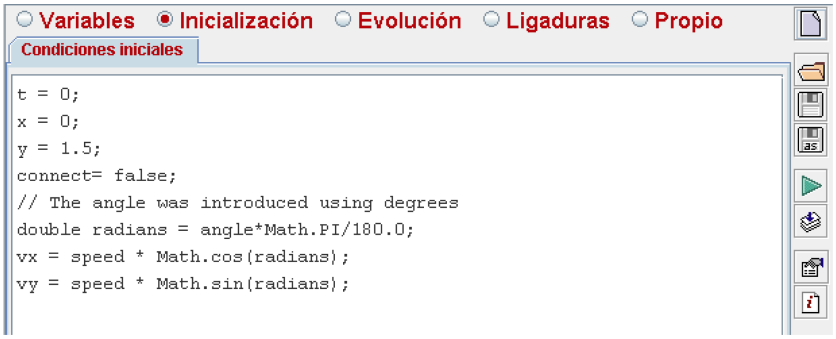

Figura A.11: Página de inicialización

Evolución

Las ecuaciones de evolución consisten simplemente en la implementación de las ecuaciones A.1, A.2 y A.3. Añadiremos la condición de parado de la simulación y de refresco de la traza (variable *connect*).

```
x = x + vx*dt;
y = y + vy*dt - 0.5*g*dt*dt;
vy = vy - g*dt;

connect = true;
if(y<=0)_pause();
```

Figura A.12: Página de evolución

Vista

En primer lugar realizaremos la construcción de la ventana principal y el panel con los controles. Para ello, añadimos un objeto del tipo *Frame* al árbol de elementos. Posteriormente añadimos un *Panel* con distribución `hbox` para añadir los botones y los campos numéricos.

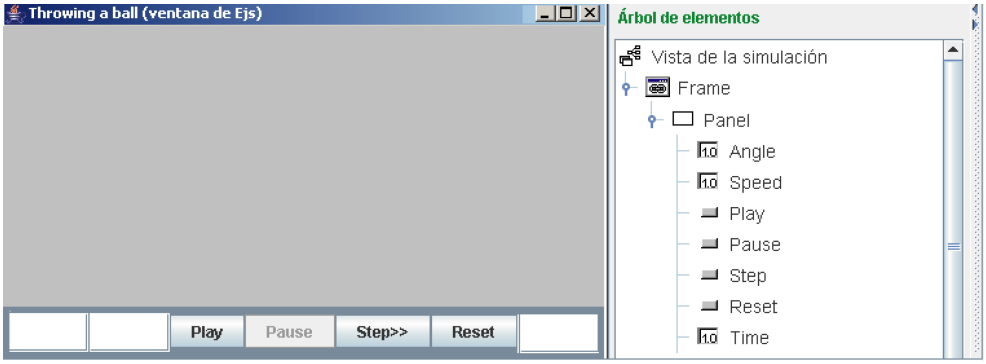

Figura A.13: Configuración de la vista

Los campos numéricos a insertar son: *Angle*, *Speed* y *Time* cuyas variables asociadas son *angle*, *speed* y *t*. Para los dos primeros campos, es necesario añadirles la acción `_initialize()` y el formato del campo, tal y como vemos en la siguiente imagen.

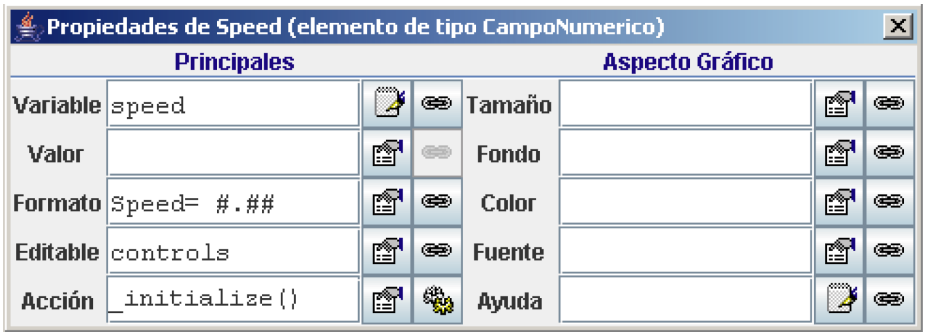

Figura A.14: Configuración del campo numérico

A los botones *Play*, *Pause*, *Step* y *Reset* se les añade las operaciones predefinidas de `_play()`, `_pause()`, `_step()` y `_reset()` para controlar la simulación.

Posteriormente, para la visualización del tiro parabólico, añadimos un panel de dibujo. Aquí, se inserta una partícula que emula al proyectil y una traza para dibujar la trayectoria. Tanto a la partícula como a la traza, se les asocia las variables x e y.

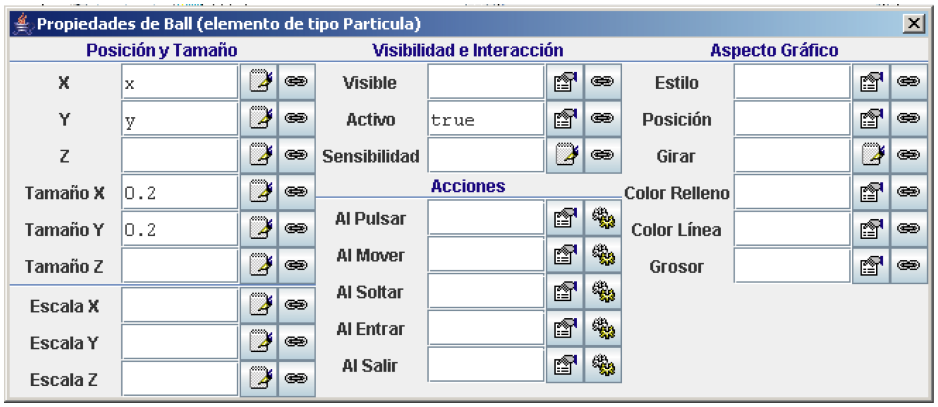

Figura A.15: Configuración de partícula

Ejecutando la simulación

Después de haber completado la simulación (modelo y vista), ésta ya se puede ejecutar. Para ponerla en marcha, hay que pulsar el botón de *Play*. La pelota seguirá una trayectoria parabólica de acuerdo con las ecuaciones de evolución y se parará cuando alcance el suelo (Figura A.16).

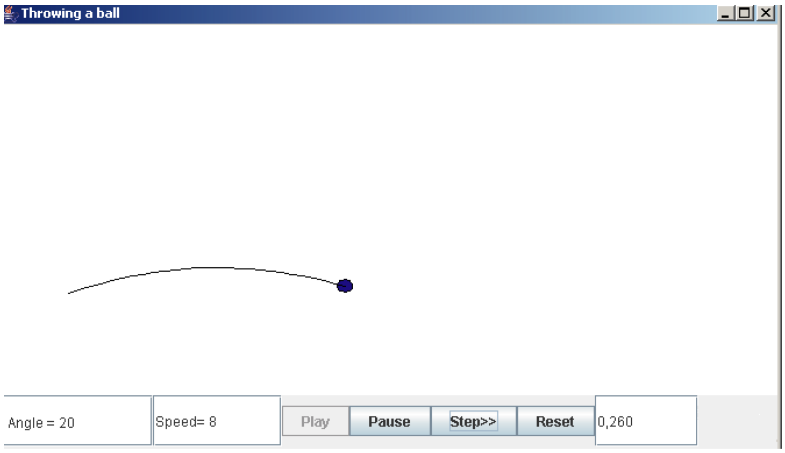

Figura A.16: Simulación

Para lanzar la pelota con un ángulo distinto, tan sólo es necesario escribir un nuevo ángulo en el campo numérico y pulsar "Intro". Esta acción llamará automáticamente al método `_initialize()`, que se encarga de ejecutar el código que escribimos en la página de inicialización sin pasar por la tabla de variables. Si se desea inicializar la simulación completamente, se tendrá que pulsar a *Reset*, que llama a la función `_reset()` que pasa por la tabla de variables y la página de evolución previa una limpieza de pantalla. Aquí vemos la una imagen que muestra la simulación.

APÉNDICE B

Contenido del CD

1. *Easy Java Simulations*. Versión de *EJS* que se ha utilizado para el desarrollo de la mayoría de las simulaciones mostradas en este documento.
2. *Java*. Versión 1.6 del JDK (*Java Development Kit*) y 1.4 de Java 3D.
3. Entornos colaborativos. Clases Java desarrolladas para el sistema de comunicación síncrono.
4. Memoria del documento. Archivo *.pdf* del documento de la memoria.
5. Simulaciones. En esta carpeta se encuentran las simulaciones desarrolladas durante el periodo de investigación. Si se desean ejecutar, tan sólo es necesario hacer doble click sobre el *.jar* de la simulación, previa instalación del JDK y Java 3D.
6. Archivos *Matlab*. Archivos *.m* que se han utilizado para el desarrollo de las aplicaciones que comunican *EJS* con *Matlab*.

Bibliografía

Abler, R. T. y Wells, I. G.: «Distributed Engineering Education: Evolution of the Telecollaboration Stations for Individualized Distance Learning». *IEEE Transactions on Education*, 2005, **48(3)**, pp. 490–496.

Alcazar, J.; F., Cuesta; A., Ollero; C., Nogales y F., López: «Teleoperación de Helicópteros para Monitorización Aérea en el Sistema Multi-Uav Comets». En: *XXIV Jornadas de Automática*, León, 2003.

Alencastre, M.; Munoz-Gomez, L. y Rudomin, I.: «Teleoperating robots in multiuser virtual environments». En: *Proceedings of the Fourth Mexican International Conference on Computer Science*, pp. 314–321, 2003.

Antsaklis, P.; Basar, T.; DeCarlo, R.; McClamroch, N. H.; Spong, M. y Yurkovich, S.: «Report on the NSF/CSS Workshop on new directions in control engineering education». *IEEE Control Systems Magazine*, 1999, **19(5)**, pp. 53–58. 0272-1708.

Aracil, Rafael: «Teleoperación». En: *III Jornadas de Trabajo en la Enseñanza vía Internet*, Alicante, 2002.

Balaguer, C.; Barrientos, A.; Sanz, P.J.; Sanz, R. y Zalama, E.: *Libro Blanco de la Robótica*. Comité Español de Automática (CEA), 2007.

Barnes, B.; Menon, A. S.; Mills, R.; Bruyns, C. D.; Twombly, A.; Smith, J.; Montgomery, K. y Boyle, R.: «Virtual reality extensions into surgical training and teleoperation». En: *4th International IEEE EMBS Special Topic Conference on Information Technology Applications in Biomedicine*, pp. 142–145, 2003.

Barrientos, A.; Peñín, L.; Balaguer, C. y Aracil, R.: *Fundamentos de Robótica*. Mc Graw-Hill, 1997.

Baumgartner, Peter: «The Zen Art of Teaching - Communication and Interactions in eEducation». En: *Proceedings of the International Workshop on Interactive Computer-Aided Learning*, , 2004.

Brady, K. y Tzyh-Jong, Tarn: «Internet-based remote teleoperation». En: *IEEE International Conference on Robotics and Automation*, volumen 1, pp. 65–70, 1998.

Brady, K. y Tzyh-Jong, Tarn: «Internet based manufacturing technology: intelligent remote teleoperation». En: *Proceedings of International Conference on Intelligent Robots and Systems*, volumen 2, pp. 843–848, 2000.

Candelas, F.; Puente, S.; Torres, F.; Segarra, V. y Navarrete, J.: «Flexibe system for simulating and teleoperating robots through the Internet». *Journal of Robotic Systems*, 2005, **22(3)**, pp. 157–166.

Candelas, F. A. y Moreno, J. S.: «Recursos didácticos basados en Internet para el apoyo a la enseñanza de materias del área de Ingeniería de Sistemas y Automática». *Revista Iberoamericana de Automática e Informática Industrial*, 2005, **2(2)**, pp. 93–100.

Candelas, F. A.; Puente, S. T.; Torres, F.; Ortiz, F. G.; Gil, P. y Pomares, J.: «A virtual laboratory for teaching robotics». *International Journal of Engineering Education*, 2003a, **19(3)**, pp. 363–370.

Candelas, F.A.; Torres, F.; Puente, S.T.; Pomares, J. y Ortiz, F.G.: «Teaching and Learning Robotics with Internet Teleoperation». En: *Second International Conference on Multimedia and ICTs in Education (m-ICTE)*, , 2003b.

Cassinis, R. y Rojas, M. T.: «Intelligent Telepresence: Introducing Virtual Reality in Advanced Robots». En: *Intelligent Perceptual Systems: New Directions in Computational Perception*, pp. 368–378. Springer, Berlin, Heidelberg, 1993.

Cassinis, R. y Terceros, M.: «Intelligent Telepresence; Introducing Virtual Reality In Advanced Robots». *Robotics and Autonomous Systems*, 2002.

Corke, P.: *Robotics Toolbox for use with Matlab*. MathWorks, 1996. Página web: `http://www.brb.dmt.csiro.au/dmt/programs/autom/matlab.html`.

Cosma, C.; Confente, M.; Botturi, D. y Fiorini, P.: «Laboratory tools for robotics and automation education». En: *Proceedings of IEEE International Conference on Robotics and Automation*, volumen 3, pp. 3303–3308, 2003.

Dalton, B. y Taylor, K.: «Distributed robotics over the Internet». *IEEE Robotics & Automation Magazine*, 2000, **7(2)**, pp. 22–27.

Denavit, J. y Hartenberg, R.S.: «A kinematic Notation for Lower-Pair Mechanims Based on matrices». *Journal of Applied Mechanims*, 1995.

Dormido, S.; Farias, G.; Sanchez, J. y Esquembre, F.: «Adding interactivity to existing Simulink models using Easy Java Simulations». En: *44th IEEE European Control Conference on Decision and Control (CDC)*, pp. 4163–4168, 2005.

Doulgeri, Z. y Matiakis, T.: «A web telerobotic system to teach industrial robot path planning and control». *IEEE Transactions on Industrial Electronics*, 2006, **49**, pp. 263–270.

Elmqvist, H.; Mattsson, S. E. y Otter, M.: «Modelica-a language for physical system modeling, visualization and interaction». En: *IEEE Symposium on Computer-Aided Control System Design (CACSD)*, pp. 630–639, 1999.

Esquembre, F.: *Creación de Simulaciones Interactivas en Java*. Prentice Hall, 2004.

Fakas, G.; Nguyen, A. y Gillet, D.: «The electronic laboratory journal: a collaborative and cooperative learning environment for web-based experimentation». *Computer Supported Cooperative Work*, 2005, **14(3)**, pp. 189–216.

Foix, C. y Zavando, S.: «Estándares e-Learning». *Informe técnico*, Centro de Tecnologías de la Información, 2002. Página web: `http://www.educ.cl/INTEC-Estandares_e-learning.pdf`.

Fu, K.S.; González, R. y Lee, C.: *Robótica, Control, Detección, Visión e Inteligencia*. Mc Graw-Hill, 2002.

García, C.; Aracil, R.; Madrid, E.; Assandri, A.; Soria, C. y Carelli, R.: «Teleoperación de un Robot Industrial a través de Internet». En: *III Jornadas de Trabajo Enseñanza vía Internet*, Alicante, 2002.

GILLET, D.; NGOC, A. y REKIK, Y.: «Collaborative Web-based Experimentation in Flexible Engineering Education». *IEEE Transactions on Education*, 2005, **48(4)**, pp. 696–704.

GOERTZ, R. C.: «Mechanical Master-Slave Manipulator». *Nucleonics*, 1954, **12(11)**, pp. 45–46.

GOLDBERG, K.; CHEN, B.; SOLOMON, R.; BUI, S.; FARZIN, B.; HEITLER, J.; POON, D. y SMITH, G.: «Collaborative teleoperation via the Internet». En: *IEEE International Conference on Robotics and Automation (ICRA)*, volumen 2, pp. 2019–2024. San Francisco, USA, 2000a.

GOLDBERG, K.; GENTNER, S.; SUTTER, C. y WIEGLEY, J.: «The Mercury Project: a feasibility study for Internet robots». *IEEE Robotics & Automation Magazine*, 2000b, **7(1)**, pp. 35–40.

GOLDBERG, K. y SIEGWART, R.: *Beyond Webcams: an introduction to online robots.* MIT Press, 2002.

GOLDBERG, M.; KUSAHARA, H.; DREYFUS, A.; GOLDMAN, O.; GRAU, M.; GRŽINIC, B.; HANNAFORD, M.; IDINOPULOS, M. y KAC, E.: «The robot in the garden: Telerobotics and Telepistemology in the age of the Internet». *Robotica*, 2001, **19(1)**.

GOLDBERG, S.; BEKEY, G.A. y AKATSUYA, Y.: «DIGIMUSE: An interactive telerobotic system for remote viewing of three-dimensional art objects». En: *Proceedings of the IEEE International Conference on Intelligent Robots and Systems (IROS)*, volumen 98, 1998.

GOLDENBERG, A.; BENHABIB, B. y FENTON, R.G.: «A complete generalized solution to the inverse kinematics of robots». *IEEE Journal of Robotics and Automation*, 1985.

GUANGMING, SONG y AIGUO, SONG: «A novel distributed architecture for building Web-enabled remote robotic laboratories». En: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 144–149, 2005.

GUZMÁN, J.L.; SARABIA, J.F.; RODRÍGUEZ, F.; MORENO, J.C. y BERENGUEL, M.: «Entorno para Programación Remota de Robots Manipuladores en ACL con Realimentación Visual». En: *III Jornadas de Trabajo de Enseñanza vía Internet*, Alicante, 2002.

Kosuge, Kauhiro; Kikuchi, Jun A. y Takeo, Koji: «VISIT: a teleoperation system via the computer network», 2002, pp. 215–226.

Kreutz, R.; Kiesow, S. y Spitzer, K.: «NetChat: Communication and Collaboration via WWW». *Journal of Educational Technology & Society*, 2000.

Kuan, Cheng-Peng y young Young, Kuu: «Challenges in VR-based robot teleoperation.» En: *Proceedings of the 2003 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 4392–4397, 2003.

Mahtani, R.; Béjar, M.; Cuesta, F.; Ollero, A. y Huertas, J.L.: «Prototipo de la Estación de Teleoperación del Instrumento TRIBOLAB en la estación Espacial Internacional». En: *XXIV Jornadas de Automática*, León, 2003.

Marin, R.; Sanz, P. J.; Nebot, P. y Wirz, R.: «A multimodal interface to control a robot arm via the web: A case study on remote programming». *IEEE Transactions on Industrial Electronics*, 2005, **52(6)**, pp. 1506–1520.

Miner, N. E. y Stansfield, S. A.: «An interactive Virtual Reality simulation system for robot control and operator training». En: *IEEE International Conference on Robotic and Automation*, , 1994.

Monferrer, A. y Bonyuet, D.: «Cooperative robot teleoperation through virtual reality interfaces». pp. 243–248. Proceedings of Sixth International Conference on Information Visualisation, 2002.

Nuño, E. y Basañez, L.: *Teleoperación: técnicas, aplicaciones, entorno sensorial y teleoperación inteligente*. Universidad Politécnica de Cataluña, 2002. Página web: `http://www.upc.es`.

Pedreño, J.L.; Guerrero, A. y López, J.: «Módulo de Teleoperación para Acceso vía Internet a Dispositivos Robóticos en Laboratorios Remotos». En: *III Jornadas de Trabajo de Enseñanza vía Internet*, Alicante, 2002.

Pieper, D. y Roth, B.: «The kinematics of manipulators under computer control». En: *Proceedings of the Second International Congress on Theory of Machines and Mechanisms*, volumen 2, pp. 159–169. Polonia, 1969.

Sabater, J.M.; Saltarén, R.; Aracil, Rafael; Yime, E. y Azorín, J.M.: «Teleoperated parallel climbing robots in nuclear installations». *An International Journal on Industrial Robots*, 2006, **33(5)**, pp. 381–386.

Safaric, R.; Truntic, M. y Hercog, D.: «Control and robotics remote laboratory for engineering education». *Interational Journal of Online Engineering (iJOE)*, 2005, **1(1)**.

Saltarén, R.; Azorín, J.; Almonacid, M. y Sabater, J.: *Prácticas de Robótica utilizando Matlab*. Universidad Miguel Hernández, 2000.

Sanchez, J.; Esquembre, F.; Martin, C.; Dormido, S.; Dormido-Canto, S.; Canto, R. D.; Pastor, R. y Urquia, A.: «Easy Java Simulations: an Open-Source Tool to Develop Interactive Virtual Laboratories Using MATLAB/Simulink». *International Journal of Engineering Education*, 2005, **21(5)**, p. 798.

Sebastian, J. M.; Garcia, D. y Sanchez, F. M.: «Remote-access education based on image acquisition and processing through the Internet». *IEEE Transactions on Education*, 2003, **46(1)**, pp. 142–148.

Snow, C.; Pullen, J. y McAndrews, P.: «Network EducationWare: An Open-Source Web-based System for Synchronous Distante Education». *IEEE Transactions on Education*, 2005.

Stein, M. R.: «Interactive Internet artistry: Painting on the World Wide Web with the PumaPaint Project». *IEEE Robotics & Automation Magazine*, 2000, **7(2)**, pp. 28–32.

Taylor, K. y Dalton, B.: «Internet robots: A new robotics niche». *IEEE Robotics & Automation Magazine*, 2000, **7(1)**, pp. 27–34.

Torres, F.; Candelas, F. A.; Puente, S. T.; Pomares, J.; Gil, P. y Ortiz, F. G.: «Experiences with virtual environment and remote laboratory for teaching and learning robotics at the University of Alicante». *International Journal of Engineering Education*, 2006, **22(4)**, pp. 766–776.

Tzafestas, C. S.; Palaiologou, N. y Alifragis, M.: «Virtual and remote robotic laboratory: comparative experimental evaluation». *IEEE Transactions on Education*, 2006, **49(3)**, pp. 360–369.

UNESCO: «Report of the Expert Meeting on Virtual Laboratories». *Informe técnico*, International Institute of Theoretical and Applied Physics (IITAP), 2000. Página web: `http://www.collaborium.org/reports/vlfinal.pdf`.