- Ouija 2000[3] : este proyecto también surgió del grupo de Ken Goldgerg poco después del Telegarden [Goldberg y otros, 2000a]. El sistema telerrobótico emulaba el tablero de una ouija controlada por un robot. El usuario conectado marcaba una letra del tablero y el robot llevaba el cursor hacia esa letra.
- RoboToy[4] : laboratorio web remoto de un pequeño brazo robótico en el que se podían coger y mover objetos, y ver lo que ocurría mediante un stream de vídeo.

Figura 3.8: Robotoy

Finalmente comentar que podemos encontrar más referencias a *On-Line Robots* en la página web `http://ford.ieor.berkeley.edu/ir`.

3.2. Sistemas Actuales

En los últimos años, las aplicaciones robóticas basadas en Web [Dalton y Taylor, 2000] han crecido de tal manera, que la mayoría de los laboratorios importantes tienen aplicaciones para acceder a sus robots desde Internet. Además, gracias al avance de la tecnología en esta área, tales como la realidad virtual y la aumentada, se ha podido mejorar la interacción hombre-máquina y la fiabilidad de estos dispositivos web.

3.2.1. Laboratorios Virtuales

- *Cosimir*. Simulador de aplicaciones con robots industriales desarrollado por FESTO. Mediante este software es posible configurar cé-

[3] *Ouija Web Page:* `http://ouija.berkeley.edu`
[4] *Robotoy Web Page:* `http://robotoy.elec.uow.edu.au/`

lulas de trabajo robotizadas, programarlas y simularlas. Actualmente es uno de los simuladores más completos, existiendo una versión educacional para la docencia y otra profesional para la empresa. Dirección web: `www.festo-didactic.com/int-es/learning-systems/software-e-learning/cosimir/`.

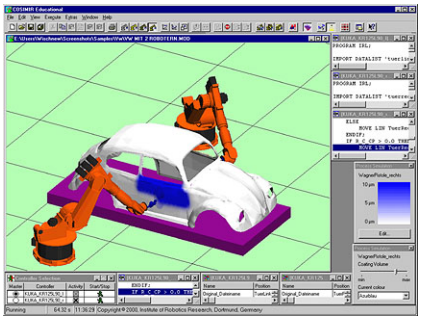

Figura 3.9: Intefaz de *Cosimir*

- *Easy-Rob 3D*. Software de simulación de robots industriales diseñado especialmente para aplicaciones industriales, aunque también ha sido empleado para docencia. La tecnología que usa para los gráficos es OpenGL y la interfaz está realizada en MFC (*Microsoft Fundation Classes*). Dirección web: `http://www.easy-rob.de/`.

Figura 3.10: Intefaz de *Easy-Robot*

- *eM-Workplace PC*. Software empleado en el diseño, simulación, optimización, análisis y programación *off-line* de múltiples robots y procesos de automatización. Proporciona una plataforma apta para optimizar y calcular los tiempos de ciclo de procesos industriales. Dirección web: `http://www.ugs.com/products/tecnomatix/assembly_planning/em_workplace_pc.shtml`

- *Famous Robotic*. Software para la simulación de procesos industriales de alta precisión (pintado, mecanizado,...). Es posible introducirle el

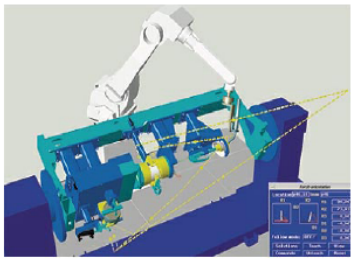

Figura 3.11: Intefaz de *eM-Workplace PC*

archivo CAD o DXF de la trayectoria del robot para simularla. Los gráficos están basados en una tecnología OpenGL. Incluye también varios módulos para el diseño y programación *off-line* de procesos industriales. Dirección web: `http://www.famos-robotic.de/engl/index.htm`.

Figura 3.12: Imágenes del software *Famous Robotic*

- *Internet Robotics*. Simulaciones desarrolladas en *Java 3D* para el control de diferentes tipos de robots: PUMA 560, CRS A465 y Nomadic XR400. Dirección web: `http://www.keldysh.ru/pages/i-robotics/home.html`.

- *Kuka.Sim Pro*. Software de simulación para robots tipo KUKA. Posee muchas opciones para el modelado, simulación, comunicación y programación *off-line* de los robots. Dirección web: `http://www.kuka.com/en/`.

- *PC-Roset*. Software desarrollado por *Kawasaki* para la simulación y programación *off-line* de robots de esta marca. Posee tres versiones dependiendo del tipo de tarea a simular: *Material Handling* (Agarre de objetos), *Arc Welding* (Operaciones de soldado) y *Painting* (Pintado).

- *Roboguide*. *Roboguide* es una familia de simuladores *off-line* desarrollada por FANUC. Posee tres módulos: *Roboguide HandlingPRO*, *Ro-*

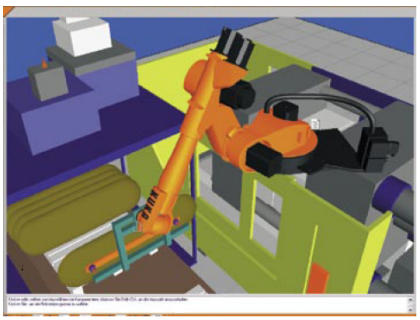

Figura 3.13: Imágenes del software *Kuka Sim Pro*

boguide PalletPRO y *Roboguide WeldPRO* que permiten simular y diseñar celdas robóticas para procesos industriales que impliquen el agarre, paletizado y soldado de objetos. Dirección web: `http://www.fanucrobotics.com`.

- *RobotStudio*. Software que proporciona herramientas para la simulación de procesos industriales utilizando robots ABB. Sus principales características son: importación de archivos CAD, edición de programas, detección de la colisión y planificación de trayectorias. Dirección web: `http://www.abb.com`.

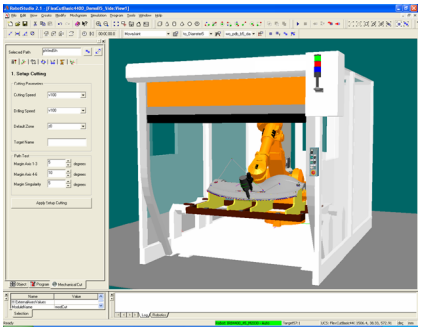

Figura 3.14: Imágenes del software *RobotStudio*

- *RoboWave*. Simulador de robots y procesos industriales. Las principales características de este software son: diseño 3D del espacio de trabajo, simulación 3D de diversos dispositivos robóticos y programación *off-line*. Dirección web: `http://www.erxa.it/prodotti.php?id=111&lang=EN`

- *Virtual Robot Simulator*. Esta aplicación permite la programación *off-line*, la simulación y la monitorización de sistemas multi-robot mediante

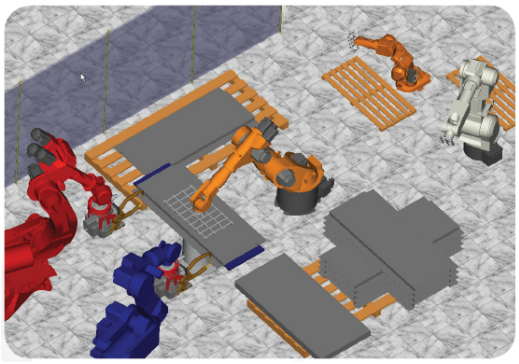

Figura 3.15: Imágenes del software *RoboWave*

una interfaz de usuario gráfico basado en OpenGL. La arquitectura software, puede ser fácilmente ampliada con componentes y aplicaciones externas que el usuario puede realizar. Este software se ha utilizado en diversos proyectos de investigación y desarrollo, como por ejemplo, para el prototipado rápido mediante troquelado robotizado de piezas para el sector del automóvil. Dirección web: `http://robotica.isa.upv.es/virtualrobot/`.

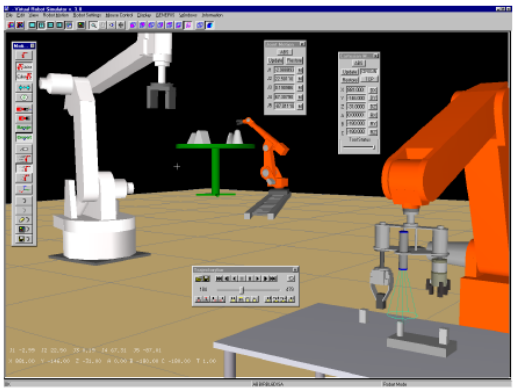

Figura 3.16: Interfaz gráfica de *Virtual Robot Simulator*

3.2.2. Laboratorios Remotos

VISIT

Este dispositivo robótico utiliza modelos predictivos en la comunicación entre la interfaz de usuario y el sistema remoto [Kosuge y otros, 2002]. El usuario conectado es capaz de realizar una tarea de simplemente con el movimiento del ratón. La arquitectura del sistema se encuentra implementada por cinco módulos: procesado de la imagen, comunicación, la interfaz de usuario, el módulo ITEM (*Interactive Task Execution Modules*) encargado de las tareas del telerobot y el módulo de control de movimiento. Aquí vemos una imagen de su interfaz de usuario:

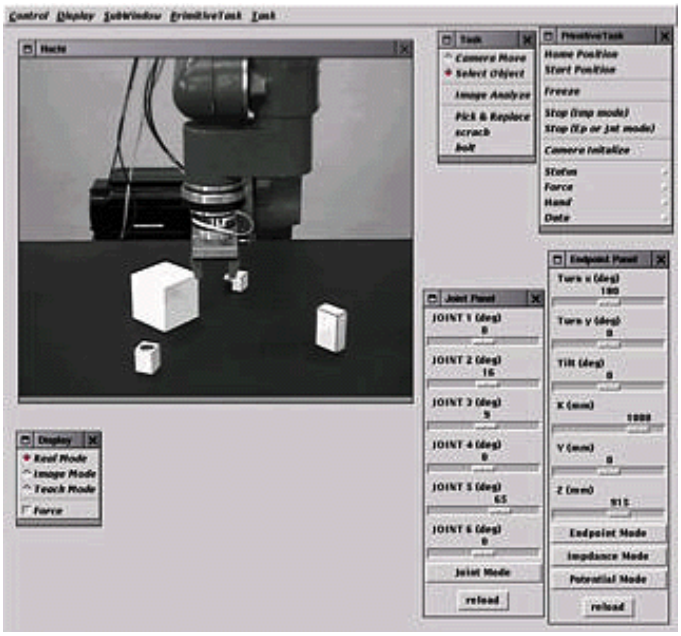

Figura 3.17: Interfaz de usuario de VISIT

ARITI

El sistema ARITI [5] (*Augmented Reality Interface for Telerobotic Applications via Internet*) contiene un robot esclavo de 4 grados de libertad cuyos actuadores son motores paso a paso. Se utiliza una tarjeta de adquisición de

[5] *ARITI Project:* `http://lsc.cemif.univ-evry.fr/Projets/ARITI/index.html`

vídeo Matrox Meteor conectada a una cámara blanco y negro para obtener imágenes en tiempo real. El servidor de vídeo está escrito en C y el servidor de control en C y ASM (Assembler).

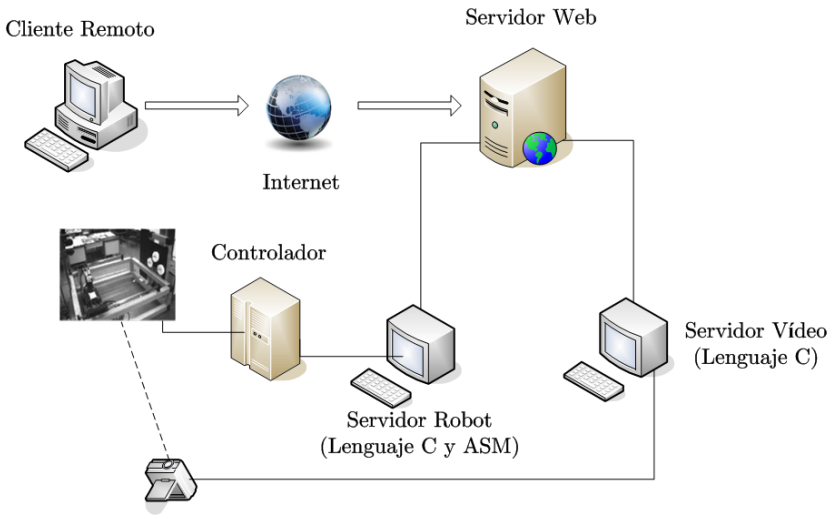

Figura 3.18: Arquitectura del sistema de ARITI

ARITI presenta una interfaz de usuario, escrita en *Java*, basada en realidad aumentada para permitir a los usuarios conectados controlar el robot remotamente desde cualquier navegador web.

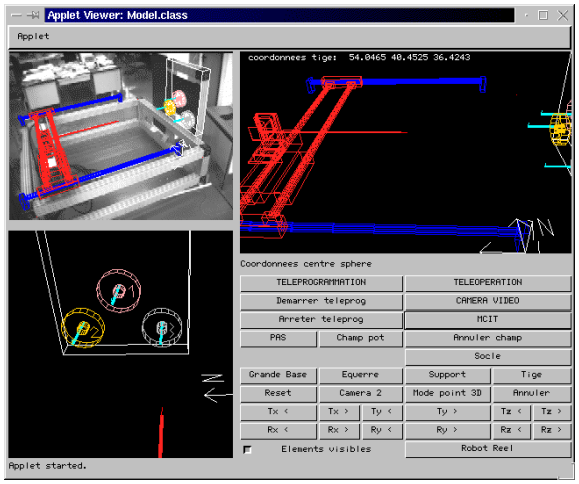

Figura 3.19: Interfaz de usuario de ARITI

Robolab

Este laboratorio remoto desarrollado en la Universidad de Alicante permite teleoperar un brazo robótico Scorbot ER-IX y un Mitsubishi PA-10 a través de Internet con la ayuda de modelos de realidad virtual [Candelas y otros, 2005]. Un usuario puede acceder a todas las funciones del laboratorio virtual a través de una página Web con un *applet Java*. Para la simulación gráfica se emplea *Java3D*, consiguiendo que toda la interfaz de usuario esté contenida en el mismo applet cliente.

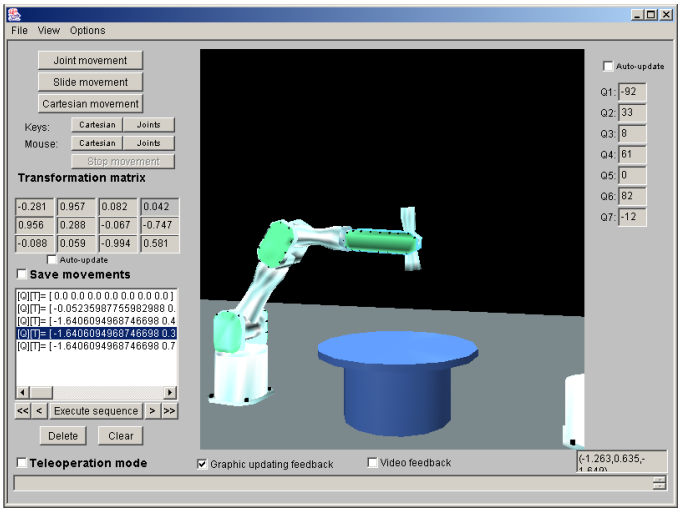

Figura 3.20: Interfaz de usuario de Robolab

Este sistema incorpora la simulación del robot (modo *Off-Line*). De esta manera, el usuario puede determinar si la ejecución de una serie de comandos es válida y no crea problemas en el robot. Después de realizar una simulación y obtener una lista de comandos válidos, el usuario puede ejecutar la opción de teleoperación (modo *On-Line*), solicitando al servidor Web que se ejecuten los movimientos de la lista en el robot real. Además, es posible incorporar nuevos modelos de robots en el sistema.

Posee dos opciones de realimentación: la primera mediante un flujo de vídeo comprimido que genera el servidor de vídeo y la segunda, una realimentación en la simulación local cliente de los valores reales de las articulaciones recibidos desde el controlador del robot.

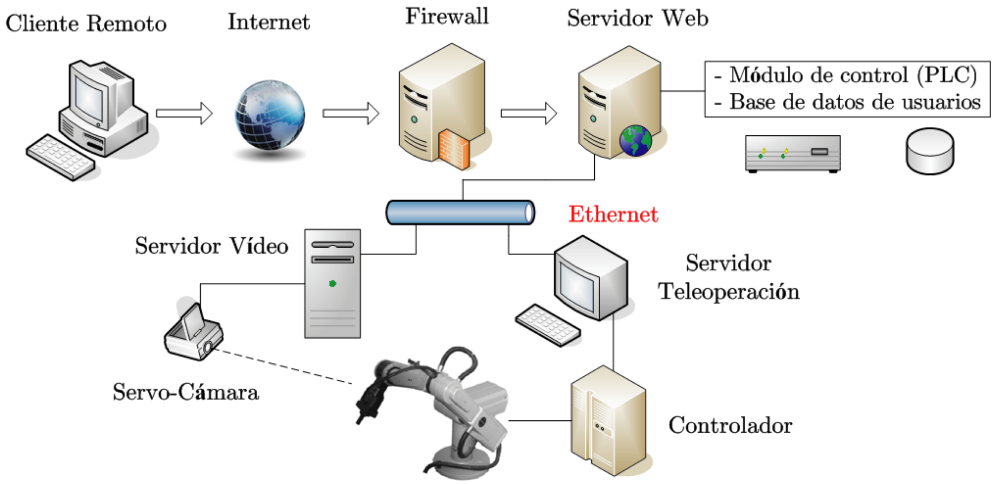

Figura 3.21: Arquitectura del sistema de Robolab

UJI Robot

Este sistema telerrobótico es capaz de controlar los movimientos de un robot vía web mediante comandos de muy alto nivel ("Coge el cubo") [Marin y otros, 2005]. También incorpora un módulo de síntesis y reconocimiento de voz y de reconocimiento de objetos para aumentar la sensación de telepresencia del usuario en el laboratorio remoto. Es uno de los sistemas más innovadores que incorpora varias tecnologías telerrobóticas.

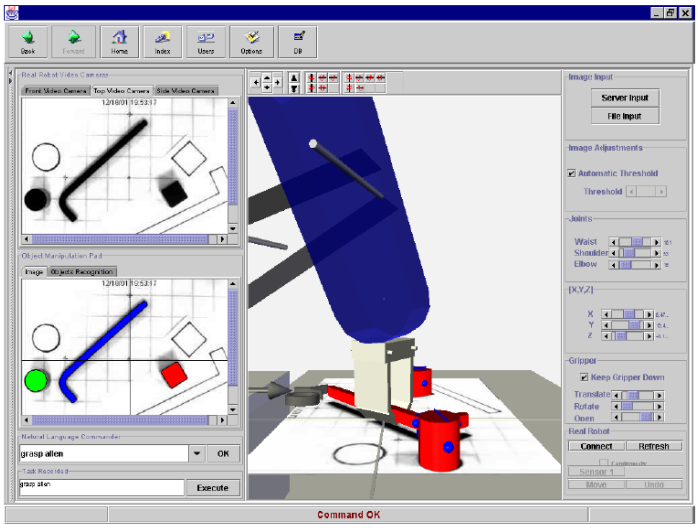

Figura 3.22: Interfaz de usuario del UJI Robot

La arquitectura del sistema está basada en la comunicación vía *Java RMI* (*Remote Method Invocation*) desde el cliente a un servidor central (*Server Manager)* que se encarga de comunicarse con el servidor adecuado en el laboratorio remoto según la petición realizada por el cliente.

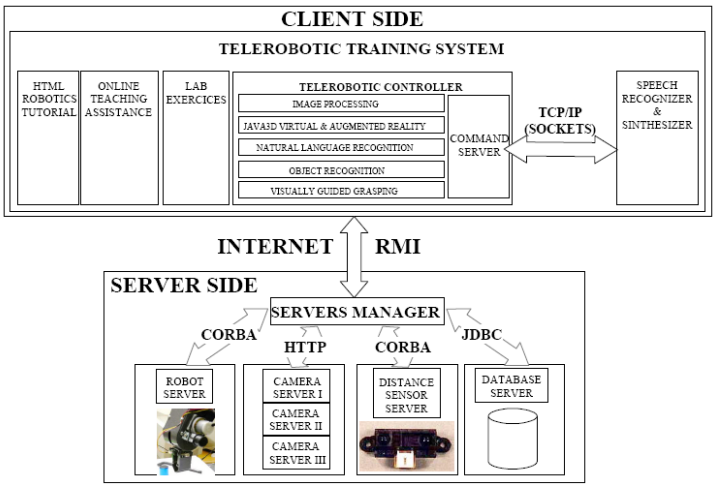

Figura 3.23: Arquitectura del sistema UJI Robot

Otros laboratorios remotos

En este apartado se nombran a otros laboratorios remotos que quizás son menos conocidos pero no por ello menos importantes que los descritos anteriormente. Están expuestos por orden cronológico de publicación y nombre del autor.

- *Cassinis, 2002.* Dispositivo telerrobótico que combina la telepresencia y la realidad virtual [Cassinis y Terceros, 2002]. El sistema se compone de un entorno remoto donde se encuentra el robot con un par estéreo de cámaras CCD montado y el entorno local virtual (emulando el entorno remoto) donde el operador posee unas lentes LEEP (*Large Espanse Extra Perspective*) para la telepresencia. Aquí, un *tracker* o rastreador tipo 3Space-Iso Track, se encarga de obtener la posición y orientación de la cabeza del operador. Estos valores son transformados a valores articulares y mandados al entorno remoto para que el robot actualice su posición y orientación. Mediante el par estéreo, el operador local posee la realimentación visual remota.

- *García, 2002.* Teleoperación a través de Internet de un robot industrial BOSCH tipo Scara [García y otros, 2002]. El robot se encuentra localizado en San Juan (Argentina) y se teleopera desde Madrid mediante un manipulador cartesiano tipo CYBERNET. La teleoperación se realiza bajo la implementación de una estrategia de compensación del retardo de comunicación, para lo cual se dispone de un simulador del entorno remoto. Los protocolos usados en el dispositivo son: TCP/IP para la conexión y NTP para la sincronización de los relojes del sistema en cada extremo.

- *Guzmán, 2002.* Herramienta software basada en la arquitectura cliente-servidor para la programación y ejecución remota de programas escritos en ACL (*Advanced Control Language*), permitiendo a los usuarios realizar sus propias aplicaciones en este lenguaje, compilarlas y probarlas en un robot remoto real [Guzmán y otros, 2002]. La arquitectura es la típica de un sistema cliente-servidor. La aplicación Cliente aporta los mecanismos necesarios para la realización de las diversas operaciones que el usuario puede realizar. Se basa en un navegador cualquiera que permite conectarse a un centro web desde donde se aportan una serie de páginas HTML, un *applet Java* y dos controles *ActiveX* (para la visualización de la realimentación del vídeo). El Servidor se encarga de recibir las peticiones enviadas por el cliente y en su caso interactuar con la célula robotizada. La comunicación entre Cliente/Servidor ha sido realizada haciendo uso de sockets TCP/IP.

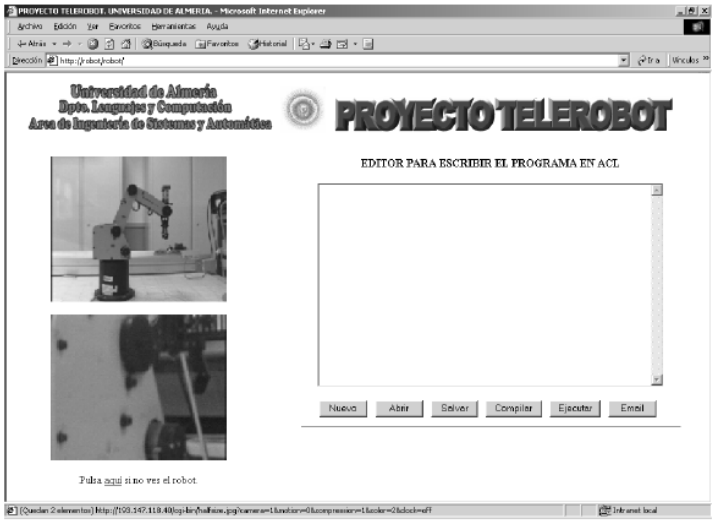

Figura 3.24: Página web del Cliente de *Guzmán*

- *Monferrer, 2002.* Teleoperación de un robot submarino de 6 gdl [Monferrer y Bonyuet, 2002]. La interfaz cliente posee un entorno de realidad virtual con modelos 3D del dispositivo telerrobótico que es actualizado constantemente con los datos de posición que le manda el robot submarino. También posee una serie de controles Active X que se encargan de la realimentación visual de una cámara de alta resolución montada sobre el robot remoto.

- *Pedreño, 2002.* En este trabajo se presenta un módulo de teleoperación basado en tecnologías IP y en una arquitectura cliente-servidor, diseñado para poder actuar remotamente a través de Internet sobre robots de diferentes características y desde cualquier terminal conectado a la red [Pedreño y otros, 2002]. Dicho módulo ha sido probado e implementado en diferentes plataformas robóticas tanto en el Laboratorio NEUROCOR (Murcia, España) a través de una Intranet, como en el ARTS Laboratory (Pisa, Italia), a través de Internet. Ambas plataformas están constituidas por un sistema brazo-mano robot con sensores artificiales de tacto (información sensorial) y sensores internos de posicionamiento (información propioceptiva).

- *Alencastre, 2003.* Entorno multiusuario creado por una interfaz realizada en *Java* con la que es posible teleoperar a diferentes plataformas robóticas (robots móviles e industriales) desde cualquier ordenador conectado a Internet y con la máquina virtual de *Java* instalada [Alencastre y otros, 2003]. También posee un servidor de vídeo con 2 cámaras para la realimentación visual.

- *Cosma, 2003.* Laboratorio virtual y remoto utilizado en la docencia de robótica en Verona (Italia) con la posibilidad de teleoperar un robot industrial PUMA 560 y un robot móvil Nomad200 [Cosma y otros, 2003]. Plataforma basada en tecnología IP e interfaz desarrollada en *Java*.

- *Safaric, 2005.* Herramienta de teleoperación a través de Internet de un robot industrial de 5 gdl, cuya interfaz virtual está construida con tecnología *Java* y *VRML* [Safaric y otros, 2005]. La plataforma remota que está basada en Matlab/Simulink y la librería Real-Time Workshop (RTW), se encarga de realizar el control del dispositivo robótico. El usuario cliente introduce la trayectoria a ejecutar por el robot remoto y la herramienta comprueba si existe algún tipo de colisión entre el robot y el entorno.

- *Song, 2005.* [6] Laboratorio web remoto de un robot Stäubli PUMA 562 [Guangming y Aiguo, 2005]. El cliente está basado en un *applet Java* incluída en un página HTML cuya interfaz está realizada con *Java Swing*. En el laboratorio remoto, un servidor web escucha las peticiones del cliente, y utiliza la tecnología *Java RMI* (*Remote Method Invocation*) para que el usuario pueda visualizar los datos(posición, pares,...) del robot real en el *applet*. Para poder obtener los datos de la cámara y de los sensores del robot(acesso a sus dll's) se ha utilizado *Java JNI* (*Java Native Interface*).

- *Doulgeri, 2006.* Desarrollo de un sistema telerrobótico Web para la planificación y control de un Puma 761 [Doulgeri y Matiakis, 2006]. El laboratorio remoto está formado por el robot y un par de cámaras fijas *pan/tilt*. La arquitectura del sistema consta de un servidor cuya comunicación con el cliente se basa en los siguientes tres puntos:

  - Protocolos de alto nivel (HTTP) para el inicio y cambio de características en el sistema remoto.
  - Protocolo RTP (*Real-time Transport Protocol*) para la realimentación visual de las cámaras remotas al cliente.
  - Protocolo UDP (*User Datagram Protocol*) para los comandos de control del cliente, tales como posición y velocidad.

Figura 3.25: Interfaz Cliente *Doulgeri*

[6] *Web Page:* `http://robot.seu.edu.cn/websensor/`

- *Tzafestas, 2006.* Laboratorio virtual y remoto para la teleoperación a través de Internet de un robot Scara AdeptOne-MV [Tzafestas y otros, 2006]. La comunicación cliente-servidor está basada en el protocolo TCP/IP y la realimentación visual en el protocolo RTP. El *applet* cliente posee una representación virtual realizada en *Java 3D* del laboratorio remoto y el servidor es el encargado de enviar por el puerto serie los comandos recibidos desde el *applet* cliente.

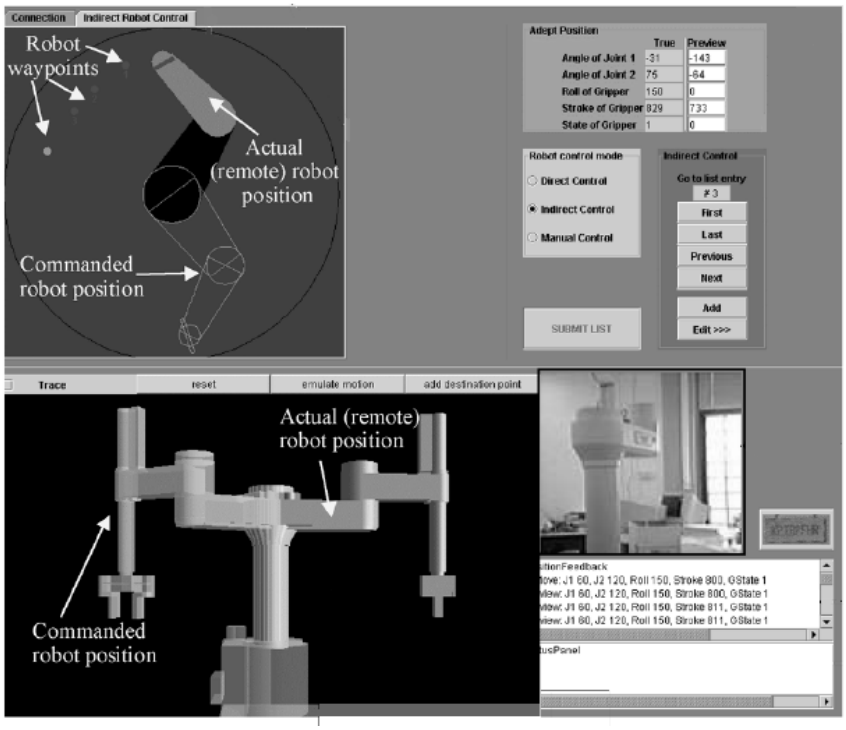

Figura 3.26: Interfaz Cliente *Tzafetas*

Para finalizar esta sección, es importante nombrar a dos puntos de información que pueden servir para conocer las nuevas realizaciones en este campo: *NASA Space Telerobotics Program*, accesible desde la dirección `http://ranier.hq.nasa.gov/telerobotics_page/realrobots.html`, que se encarga de recoger los laboratorios web remotos más importantes de la actualidad, y a *German Aerospace Center*, accesible desde la web `http://www.robotic.dlr.de/Joerg.Vogel/`, donde su principal miembro Jörg Vogel presenta una extensa biblioteca de aplicaciones relacionadas con simulaciones robóticas y control remoto.

3.3. Conclusiones

Para finalizar el presente capítulo, se va a describir de forma breve los métodos más empleados para la construcción de sistemas de control supervisado.

Como se ha visto durante el desarrollo de este capítulo, las aplicaciones telerrobóticas (programación y monitorización remota) se basan en disponer de entornos *software* que modelan y visualizan escenarios 3D, además de ser ejecutables de forma remota. Para ello, la mayoría de dichos sistemas utilizan la combinación de *VRML* (*Virtual Reality Modeling Language*) y *Java* para el desarrollo de sus entornos de programación remota de robots industriales. *VRML* se utiliza para la representación gráfica y *Java* para la implementación de los procesos de cálculo, gestión de la interfaz (*Java Swing* o AWT[7]), la comunicación a través de la web y en ocasiones, también para los gráficos (*Java 3D*).

En el caso de la interfaz remota, las aplicaciones se presentan en forma de *applet*, programas *Java* que se pueden ejecutar desde un navegador cliente. De esta manera, las herramientas se pueden ejecutar desde cualquier computador conectado a Internet. Para la realimentación visual de la imagen de cámaras situadas en el lugar remoto, suele emplearse controles *Active X*, componentes *software* que pueden implementarse en páginas web, permitiendo a los usuarios interactuar con animación, audio y vídeo sin necesidad de abrir programas separados.

El cliente siempre accede mediante el *applet* al servidor web del lugar remoto. La comunicación entre ambos suele realizarse de dos maneras:

- Mediante *TCP sockets*. Suele emplearse cuando en el servidor web se está ejecutando un programa en *Java* que establece el *socket* de comunicación con el cliente.
- Usando el protocolo HTTP. Se emplea para escribir texto en páginas dinámicas que actúan según el mensaje recibido del cliente.

Como se ha comprobado en el apartado donde se expone el estado del arte, generalmente se suelen emplear ambos métodos. Las principales diferencias entre ambos son que los *TCP Sockets* son más rápidos que el protocolo HTTP,

[7]Librerías de *Java* para la creación de interfaces

sin embargo, pueden darse problemas en la comunicación por la existencia de *Firewalls*, lo que no suele ocurrir con el protocolo HTTP.

Una vez que ha llegado la señal de comunicación al servidor web, normalmente éste accede a los servidores de los dispositivos remotos (controlador del robot, cámaras,etc,...) mediante *TCP sockets*, comunicando a cada uno de ellos las órdenes recibidas desde la interfaz cliente. También suele emplearse *JRMI* (*Java Remote Method Invocation*), librería de *Java* que permite ejecutar remotamente objetos desarrollados en *Java*.

Normalmente, las API de control de los dispositivos *hardware* suelen ser librerías dinámicas bajo lenguaje C (*.dll*). Para acceder a ellas desde una aplicación o programa desarrollado en *Java*, se emplea *JNI* (*Java Native Interface*), librería que permite al usuario crear una interfaz entre las funciones de control del *hardware* y la aplicación *Java*.

Finalmente, como se ha podido comprobar, el lenguaje *Java* posee gran relevancia para el desarrollo de las aplicaciones telerrobóticas. Además, su extensa API facilita la programación de sistemas remotos y las tareas de comunicación.

CAPÍTULO 4

Desarrollo de un laboratorio virtual robótico con *EJS*

4.1. Introducción

En este capítulo se va a explicar cómo desarrollar un laboratorio virtual robótico mediante el software *Easy Java Simulations* (*EJS*). Tal y como se comenta en el apéndice A, *EJS* es un software libre desarrollado en *Java* con el fin de simplificar la tarea de la creación de un laboratorio virtual. Desde un punto de vista técnico, esta herramienta facilita la realización tanto de la interfaz gráfica (*vist*a) como de la descripción científica del fenómeno que se desea simular (*modelo*).

Actualmente *EJS* se utiliza para la creación de todo tipo de simulaciones de fenómenos físicos, tales como movimiento de partículas, sólido rígido, transferencia de calor, sistemas de control, etc, ... pero en el campo de la robótica industrial no se han desarrollado muchas simulaciones, y las existentes son muy simples y con robots de pocos grados de libertad. Así, el objetivo de este punto es describir de qué manera realizar una simulación de un robot industrial mediante las herramientas que proporciona *EJS*, desde la construcción de los sistemas Denavit - Hartenberg a partir de su tabla, hasta la dinámica.

Y llegado a este punto, seguramente el lector se preguntará: ¿por qué uti-

lizar *EJS* y no otras herramientas de simulación?. Esta pregunta también se planteó el primer día que se comenzó a trabajar en este proyecto, al desconocerse el funcionamiento de dicha herramienta. La elección quedó justificada tras conseguir, después de sólo unas semanas trabajando con *EJS*, una aplicación que simulaba completamente el movimiento de un robot de 6gdl. Este programa implementa la cinemática directa, la inversa y un planificador de trayectorias. Este hecho puso de manifiesto las ventajas que *EJS* proporciona para la creación de las simulaciones interactivas:

- *EJS* simplifica enormemente la creación de la interfaz y la simulación gráfica del modelo mediante las clases de visualización de 2D y 3D que posee.
- *EJS* proporciona varios controles *Java Swing* para construir la interfaz de usuario fácilmente, sin tener que programarlos.
- *EJS* incorpora la posibilidad de incluir código *Java* de cualquier clase de su API o de una creada por el usuario. De esta forma, al estar todo (herramienta y simulación) basado en *Java*, la tarea de la creación de un laboratorio virtual o remoto se encuentra simplificada gracias el poder de *Java* en temas relacionados con la comunicación a través de Internet.
- *EJS* nos permite resolver numéricamente de manera cómoda complejos sistemas de ecuaciones diferenciales ordinarias mediante el editor de ecuaciones diferenciales que posee.
- *EJS* permite la comunicación con *Matlab*, una potente herramienta para poder desarrollar el modelo del sistema.

Antes de iniciar el siguiente punto, es necesario comentar que como *EJS* está en continuo cambio y van surgiendo versiones nuevas en cortos períodos de tiempo, el contenido de este apartado va a ser explicado tanto en la versión que existe actualmente en la página web del autor [1] (version antigua), como en una más actual obtenida en un curso sobre *EJS* recientemente realizado en la Universidad de Alicante. Además, se mostrarán partes del código desarrollado tanto en *Matlab* como en *Java* para la implementación de las distintas opciones de la simulación, haciendo siempre referencia a su archivo correspondiente del CD.

[1] *EJS web page:* `http://fem.um.es`

4.2. Cinemática Directa

Concepto Teórico.

El problema cinemático directo consiste en determinar cuál es la posición y orientación del extremo final del robot, con respecto a un sistema de coordenadas que se toma como referencia, conocidos los valores de las articulaciones y los parámetros geométricos de los elementos del robot.

$$\begin{aligned} x &= f_x(q_1, q_2, q_3, \ldots, q_n) \\ y &= f_y(q_1, q_2, q_3, \ldots, q_n) \\ z &= f_z(q_1, q_2, q_3, \ldots, q_n) \\ \alpha &= f_\alpha(q_1, q_2, q_3, \ldots, q_n) \\ \beta &= f_\beta(q_1, q_2, q_3, \ldots, q_n) \\ \gamma &= f_\gamma(q_1, q_2, q_3, \ldots, q_n) \end{aligned}$$

En general, un robot de n grados de libertad está formado por n eslabones unidos por n articulaciones, de forma que cada par articulación - eslabón constituye un grado de libertad. A cada eslabón se le puede asociar un sistema de referencia solidario a él y, utilizando las transformaciones homogéneas, es posible representar las rotaciones y traslaciones relativas entre los distintos eslabones que componen el robot. La matriz de transformación homogénea que representa la posición y orientación relativa entre los distintos sistemas asociados a dos eslabones consecutivos del robot se denomina ${}^{i-1}A_i$. Del mismo modo, la matriz ${}^0A_k$, resultante del producto de las matrices ${}^{i-1}A_i$ con i desde 1 hasta k, es la que representa de forma total o parcial la cadena cinemática que forma el robot con respecto al sistema de referencia inercial asociado a la base. Cuando se consideran todos los grados de libertad, a la matriz ${}^0A_n$ se le denomina T, matriz de transformación que relaciona la posición y orientación del extremo final del robot respecto del sistema fijo situado en la base del mismo. Así, dado un robot de 6gdl, se tiene que la posición y orientación del eslabón final vendrá dado por la matriz T:

$$T = {}^0A_6 = {}^0A_1 \cdot {}^1A_2 \cdot {}^2A_3 \cdot {}^3A_4 \cdot {}^4A_5 \cdot {}^5A_6$$

Para describir la relación que existe entre dos sistemas de referencia asociados a eslabones, se utiliza la representación Denavit - Hartenberg (D-H). Denavit y Hartenberg propusieron en 1955 un método matricial que permite establecer de manera sistemática un sistema de coordenadas $\{S_i\}$ ligado a cada eslabón i de una cadena articulada [Denavit y Hartenberg, 1995]. Además, la representación D-H permite pasar de un sistema de coordenadas a otro mediante 4 transformaciones básicas que dependen exclusivamente de las características geométricas del eslabón.

Estas transformaciones básicas consisten en una sucesión de rotaciones y traslaciones que permiten relacionar el sistema de referencia del elemento i con el sistema del elemento $i-1$. Las transformaciones en cuestión son las siguientes:

1. Rotación alrededor del eje z_{i-1} un ángulo θ_i.
2. Traslación a lo largo de z_{i-1} una distancia d_i.
3. Traslación a lo largo de x_i una distancia a_i.
4. Rotación alrededor del eje x_i un ángulo α_i.

Teniendo ya los valores de θ_i, d_i, a_i, α_i, que son los denominados parámetros D-H del eslabón i, la matriz de transformación que relaciona los sistemas de referencia $\{S_{i-1}\}$ y $\{S_i\}$ es la siguiente:

$$^{i-1}A_i = T(z,\theta_i)\cdot T(0,0,d_i)\cdot T(a_i,0,0)\cdot T(x,\alpha_i)$$

Desarrollando esta expresión en términos de los parámetros D-H, nos queda:

$$^{i-1}A_i = \begin{pmatrix} \cos\theta_i & -\cos\alpha_i\cdot\sin\theta_i & \sin\alpha_i\cdot\sin\theta_i & a_i\cdot\cos\theta_i \\ \sin\theta_i & \cos\alpha_i\cdot\cos\theta_i & -\sin\alpha_i\cdot\cos\theta_i & a_i\cdot\sin\theta_i \\ 0 & \sin\alpha_i & \cos\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Desarrollo con *EJS*.

El primer paso para implementar la cinemática directa en *EJS*, es la construcción en la interfaz gráfica de los sistemas D-H. Estos sistemas de referencia llevarán asociados los valores de rotación y traslación de las matrices

de transformación homogénea ${}^0A_k$. Por tanto, será necesario programar un algoritmo que proporcione los valores de dichas matrices ${}^0A_1$, ${}^0A_2 \ldots$, ${}^0A_n$ a partir de la tabla de los parámetros D-H.

La programación de la cinemática directa puede realizarse de dos maneras:

1. Mediante código *Matlab* (archivos *.m*). Como se comentó en la introducción de este punto, *EJS* posee la propiedad de comunicarse con *Matlab* [Sanchez y otros, 2005]. La ventaja de utilizar este software es su gran capacidad de cálculo matricial. El inconveniente, es el uso de otro programa, a parte de *EJS*, para realizar el laboratorio virtual. La función principal de este algoritmo es la que proporciona la matriz ${}^{i-1}A_i$ a partir de los parámetros D-H. Aquí se muestra su código (archivo *denavit.m*):

```
function dh=denavit(teta, d, a, alfa)
dh=[cos(teta)  -cos(alfa)*sin(teta)   sin(alfa)*sin(teta)   a*cos(teta);
    sin(teta)   cos(alfa)*cos(teta)  -sin(alfa)*cos(teta)   a*sin(teta);
           0              sin(alfa)             cos(alfa)             d;
           0                     0                    0             1];
```

   Ya teniendo el valor de ${}^{i-1}A_i$ entre sistemas de referencia consecutivos, para hallar ${}^0A_1 \ldots {}^0A_n$ es necesario posmultiplicar las matrices. Aquí se muestra la función de la cinemática directa de un brazo robot industrial de 6gdl (archivo *cindir6.m*) tipo PA-10:

```
% Parámetros D-H
teta = q;
d    = [l1   0      0     l4     0    l6];
a    = [0      a2   0     0       0    0];
alfa = [-pi/2  0    pi/2  -pi/2   pi/2   0];

% Matrices de transformación homogénea entre sistemas de coordenadas consecutivos
A01 = denavit(teta(1)+pi/2, d(1), a(1), alfa(1));
A12 = denavit(teta(2)-pi/2, d(2), a(2), alfa(2));
A23 = denavit(teta(3)+pi, d(3), a(3), alfa(3));
A34 = denavit(teta(4), d(4), a(4), alfa(4));
A45 = denavit(teta(5), d(5), a(5), alfa(5));
A56 = denavit(teta(6), d(6), a(6), alfa(6));

% Matrices de transformación entre eslabones
A02 = A01*A12;
A03 = A02*A23;
A04 = A03*A34;
A05 = A04*A45;
A06 = A05*A56;
```

2. Mediante código *Java*. Estas funciones pueden ser directamente programadas en el entorno que proporciona *EJS* en lo que se denomina

"Panel Propio" (ver A.3). La biblioteca matemática que trae por defecto el JDK (*Java Development Kit*)es muy limitada para emplearla en el cálculo matricial. Por tanto, es necesario utilizar otra biblioteca que viene con el paquete de Java 3D[2] denominada *vectmath.jar*. Aquí se muestra el código *Java* de la cinemática directa de un robot de 5gdl tipo Scorbot-ER IX[3]:

```
public void directCinematic () {

//Tomamos los valores articulares de la interfaz
q1 = q1d*Math.PI/180;
q2 = q2d*Math.PI/180;
q3 = q3d*Math.PI/180;
q4 = q4d*Math.PI/180;
q5 = q5d*Math.PI/180;

//Creamos la tabla de denavit
DHq[0] = q1;
DHq[1] = q2-Math.PI/2;
DHq[2] = q3+12.4*Math.PI/180;
DHq[3] = q4-18.0*Math.PI/180;
DHq[4] = q5-2.0*Math.PI/180;

//Función denavit. Introduce los valores en un vector At[9]
denavit(DHq[0],DHd[0],DHa[0],DHalpha[0]);

//Introducimos la traslacion inicial p0x,p0y,p0z
At[3] = At[3]+p0x;
At[7] = At[7]+p0y;
At[11] = At[11]+p0z;

//Creamos la matriz A01 a partir de At
Matrix4d A01 = new Matrix4d(At);

//Matrices entre eslabones consecutivos
denavit(DHq[1],DHd[1],DHa[1],DHalpha[1]);
Matrix4d A12 = new Matrix4d(At);
denavit(DHq[2],DHd[2],DHa[2],DHalpha[2]);
Matrix4d A23 = new Matrix4d(At);
denavit(DHq[3],DHd[3],DHa[3],DHalpha[3]);
Matrix4d A34 = new Matrix4d(At);
denavit(DHq[4],DHd[4],DHa[4],DHalpha[4]);
Matrix4d A45 = new Matrix4d(At);

//Matrices de transformación respecto al S.Invercial
Matrix4d A02 = new Matrix4d();
A02.mul(A01,A12);
Matrix4d A03 = new Matrix4d();
A03.mul(A02,A23);
Matrix4d A04 = new Matrix4d();
A04.mul(A03,A34);
Matrix4d A05 = new Matrix4d();
A05.mul(A04,A45);
}
```

[2] http://java3d.dev.java.net

[3] http://www.sti-sl.es/scorboter9.htm

La función denavit(double q,double d, double a, double alfa) que se observa en el código realiza lo mismo que la que se explicó en el apartado anterior, así que no se mostrará.

Habiendo desarrollado ya el algoritmo de la cinemática directa, se pasa a la construcción de los sistemas de referencia en la interfaz. La representación de los sistemas D-H se realiza con un elemento denominado "conjunto de vectores". Para ello, es necesario asociar las propiedades gráficas del elemento en sí (posición y tamaño), con los valores de rotación y traslación de las matrices de transformación que nos proporciona el algoritmo anterior (Figura 4.1).

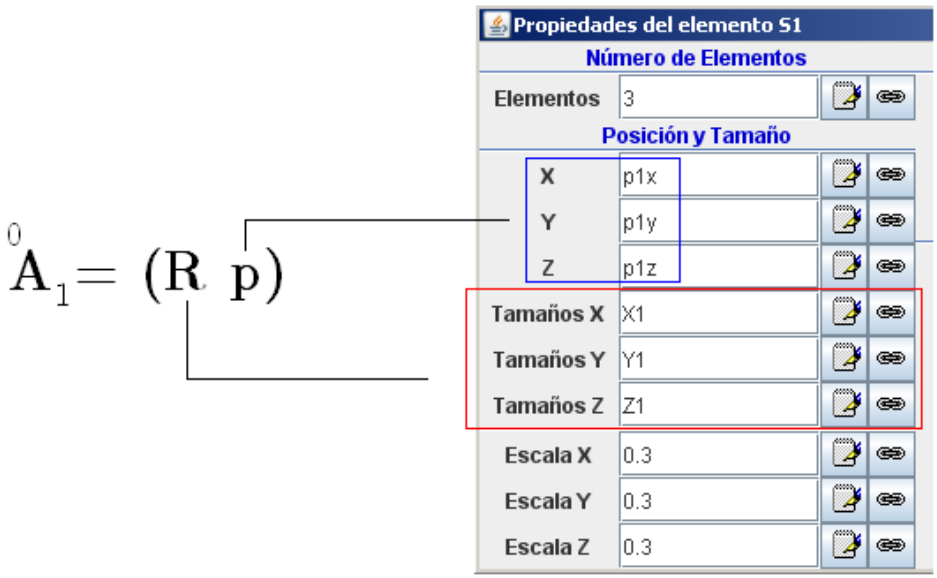

Figura 4.1: Asociación de valores

Como se observa en la figura, se asocia a las propiedades de posición de un sistema de referencia S_K, los valores del vector posición $\vec{p}$ de la matriz ${}^0A_K$. Y a las propiedades de tamaño (X, Y, Z), los valores por filas de la matriz de rotación $R_k = [X_k; Y_k; Z_k]$. Realizando este procedimiento para cada uno de los sistemas de referencia D-H del supuesto robot, se consigue una representación gráfica simple de la dimensión y posición de la estructura robótica. Para conseguir movimiento en la estructura, deben crearse unos controles de deslizamiento (*sliders*) que tengan como variables asociadas los valores articulares del robot $(q_1 \dots q_n)$, y cada vez que su valor sea cambiado, llamar al algoritmo de la cinemática directa mostrado anteriormente. Este

algoritmo calculará las matrices de rotación ${}^0A_1 \ldots$, ${}^0A_n$ que refrescarán la posición y orientación de los sistemas D-H.

Dado que existen dos posibilidades para la programación de la cinemática directa, *Matlab* y *Java*, es conveniente explicar cómo es la llamada a cada una de ellas, ya que se realiza de forma distinta. En el caso de *Matlab*, se deben escribir las variables articulares y geométricas en el *worksapce*, ejecutar el archivo *.m* donde se encuentre el algoritmo y recoger los valores de las matrices de transformación que se han escrito en el *workspace*. Para realizar esta acción, es necesario utilizar la interfaz de comunicación que existe entre *Matlab* y *EJS*, que se basa en una serie de funciones que escriben y leen del *workspace*. Aquí se muestra el código comentado para ejecutar la cinemática directa de un robot de 6gdl:

```
//Escribimos los valores articulares en el workspace
_external.setValue("q1",q1r);
_external.setValue("q2",q2r);
_external.setValue("q3",q3r);
_external.setValue("q4",q4r);
_external.setValue("q5",q5r);
_external.setValue("q6",q6r);

//Escribimos el valor de las variables geométricas al workspacce
_external.setValue("l1",l1);
_external.setValue("a2",a2);
_external.setValue("l4",l4);
_external.setValue("l6",l6);

// Llamada al archivo cindir6.m de Matlab
% BEGIN CODE:
cindir6;
% END CODE

//Leemos el resultado del algoritmo de workspace
//Valores para el sistema de referencia A1
X1 = _external.getDoubleArray("X1");
Y1 = _external.getDoubleArray("Y1");
Z1 = _external.getDoubleArray("Z1");
p1x = _external.getDouble("p1x");
p1y = _external.getDouble("p1y");
p1z = _external.getDouble("p1z");
```

En el caso de ser programado el algoritmo en *Java* dentro del entorno de *EJS*, tan sólo se debe llamar a la función que calcule la cinemática directa. El resultado que se obtiene para ambas formas de programación es idéntica: un movimiento articular de una estructura formada por sistemas de referencia D-H (Figura 4.2)

Para conseguir el aspecto gráfico del dispositivo robótico es necesario asociar a los sistemas de referencia D-H, sólidos (cilindros, cubos,...) que

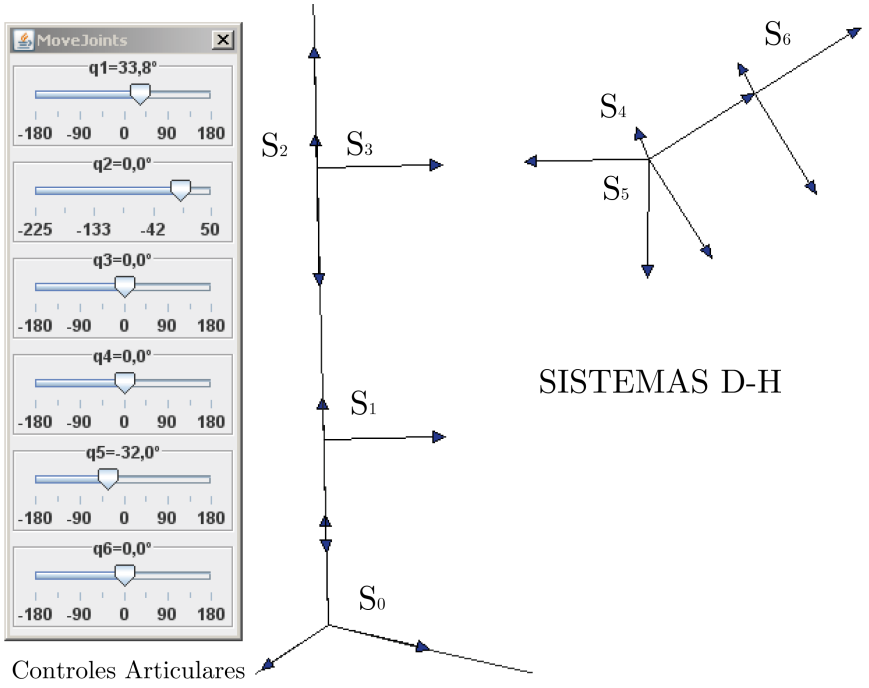

Figura 4.2: Cinemática Directa de un robot de 6gdl

proporciona *EJS*. Actualmente, existen dos posibilidades:

1. Utilizando el panel *Simple 3D* (versión inicial de *EJS*). Los objetos que este panel puede cargar son cilindros, esferas, conos y cajas. Su funcionamiento es bastante sencillo y no da ningún tipo de problema con el código que genera. Sin embargo sólo es posible asociar movimiento de rotación a cilindros y conos, además de que su representación gráfica es muy simple.

2. Utilizando el panel *Display 3D* (versión moderna de *EJS*). Este panel carga objetos de la API de Java 3D, consiguiendo un aspecto más real en la vista de la simulación. Además, es posible asociar movimientos de rotación a todos los objetos. El principal inconveniente de utilizar este panel es que da problemas en la generación del código, resultando errores en la ejecución de la aplicación y en ocasiones, fallando en la representación de la vista (no se ven los objetos en la interfaz). Esto se debe a que la biblioteca 3D incorporada en esta versión de *EJS* no está completamente depurada.

Con la primera opción, tanto los eslabones como las articulaciones del

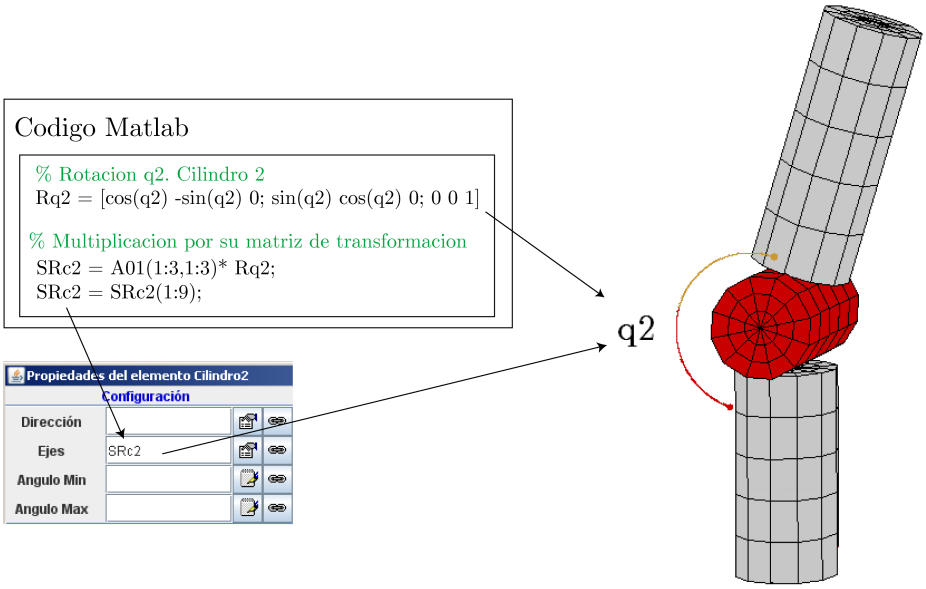

Figura 4.3: Simulación del movimiento de un cilindro en Simple 3D

robot serán cilindros. Para simular su movimiento, basta con asociar a sus ejes (característica del objeto), la matriz de transformación del eslabón que representa. Esta acción, orientará y moverá el cilindro según el sistema de referencia D-H asociado. Además, si se desea que el cilindro simule el movimiento de rotación cuando se cambia el valor articular, se debe multiplicar la submatriz de rotación R_k de su transformación asociada ${}^0A_k$, por otra rotación sobre su eje principal (Figura 4.3).

En el caso de utilizar el panel *Display 3D*, es necesario realizar una transformación de matrices de rotación R_k, que es lo que nos proporciona nuestro algoritmo de denavit, a par de rotación $Rot(\theta, \vec{k})$, ya que es el tipo de dato que aceptan los objetos Java 3D (Figura 4.4). Para realizarlo, se puede programar una función en *Matlab* (archivo *transSRtoQ.m*) o en *Java* para poder dar movimiento a los objetos de este panel. A continuación el código en Java de dicha función:

```
public double[] transSRtoPar (double[] SRc) {

double [] par = new double[4];
double kx1,ky1,kz1,norm, ctemp;
boolean add;

temp = 0.5*Math.sqrt(SRc[0]+SRc[4]+SRc[8]+1);
double kx = SRc[7] - SRc[5];
double ky = SRc[2] - SRc[6];
```

```
double kz = SRc[3] - SRc[1];

if ( (SRc[0]>=SRc[4]) & (SRc[0]>=SRc[8]) )
{
     kx1 = SRc[0] - SRc[4] - SRc[8] + 1;
     ky1 = SRc[3] + SRc[1];
     kz1 = SRc[6] + SRc[2];
     add = (kx>=0);
}
else if( (SRc[4] >= SRc[8]) )
{
     kx1 = SRc[3] + SRc[1];
     ky1 = SRc[4] - SRc[0] - SRc[8] + 1;
     kz1 = SRc[7] + SRc[5];
     add = (ky>=0);
}
else
{
     kx1 = SRc[6] + SRc[2];
     ky1 = SRc[7] + SRc[5];
     kz1 = SRc[8] - SRc[0] - SRc[4] + 1;
     add = (kz>=0);
}
if (add)
{
     kx = kx + kx1;
     ky = ky + ky1;
     kz = kz + kz1;
}
else
{
     kx = kx - kx1;
     ky = ky - ky1;
     kz = kz - kz1;
}
norm = Math.sqrt(kx*kx + ky*ky + kz*kz);
if (norm == 0)
{
      par[0] = 1;
      par[1] = 0;
      par[2] = 0;
      par[3] = 1;
}
else
{
     par[1] = (1/norm)*kx;
     par[2] = (1/norm)*ky;
     par[3] = (1/norm)*kz;
}
par[0] = 2*Math.acos(temp);
return par;
}
```

En las imágenes mostradas (Figuras 4.3 y 4.4) se observa la diferencia entre los tipos de gráficos que generan los dos paneles. Puede verse con más detalle en el capítulo 6 (Resultados Obtenidos), donde se muestran dos aplicaciones, una realizada con el panel *Simple 3D* (sección 6.1) y otra con el panel *Display 3D* (sección 6.2).

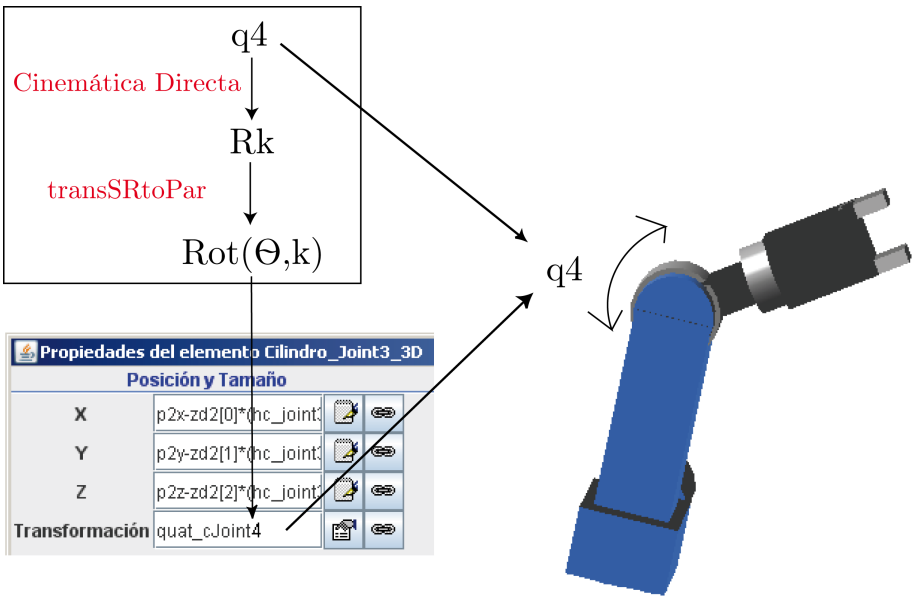

Figura 4.4: Simulación del movimiento de un objeto en Display 3D

Como conclusión de este punto, se puede decir que la simulación del movimiento de un robot en *EJS* se basa en la implementación del modelo cinemático directo. De esta manera, es posible mover el robot mediante controles articulares dando valores a cada una de las articulaciones del robot $(q_1 \dots qn)$. En el siguiente punto, se explicará la implementación del modelo cinemático inverso, cuya finalidad se basa en dar movimiento a un robot mediante valores de la posición y orientación del punto final $(X, Y, Z, X°, Y°, Z°)$.

4.3. Cinemática Inversa

Concepto Teórico.

El problema cinemático inverso consiste en encontrar los valores que deben adoptar las coordenadas articulares del robot $q = [q_1, q_2, \dots, q_n]$ para que su extremo se posicione y oriente según una determinada localización espacial. Al contrario que el problema cinemático directo, el cálculo de la cinemática inversa no es sencilla ya que consiste en la resolución de una serie de ecuaciones fuertemente dependiente de la configuración del robot, además de existir diferentes $n - uplas$ $q = [q_1, q_2, \dots, q_n]$ que resuelven el problema.

En la actualidad existen procedimientos genéricos susceptibles de ser programados [Goldenberg y otros, 1985] para la resolución de la cinemática inversa y obtener la $n-upla$ de valores articulares que posicionen y orienten el extremo final. Sin embargo, el principal inconveniente de estos procedimientos es que son métodos numéricos iterativos, que no siempre garantizan tener la solución en el momento adecuado. De esta manera, a la hora de resolver el problema cinemático inverso es mucho más adecuado encontrar una solución cerrada. Es decir, encontrar una relación matemática explícita de la forma:

$$\begin{aligned} q_k &= f_k(x, y, z, \alpha, \beta, \gamma) \\ k &= 1 \dots n \end{aligned}$$

Para poder conseguir esta relación suele ser habitual emplear métodos geométricos, que consisten en la utilización de las relaciones trigonométricas y la resolución de los triángulos formados por los elementos y articulaciones del robot. La mayoría de los robots suelen tener cadenas cinemáticas relativamente sencillas, y los tres primeros gdl, que posicionan al robot en el espacio, suelen tener una estructura planar. Esta condición facilita la resolución de la $n-upla$. Además, los tres últimos grados de libertad suelen usarse para la orientación de la herramienta, lo cual permite la resolución desacoplada (desacoplo cinemático [Pieper y Roth, 1969]) de la posición del extremo del robot y de la orientación de la herramienta.

Como alternativa para resolver el mismo problema se puede recurrir a manipular directamente las ecuaciones correspondientes al problema cinemático directo. Es decir, a partir de la relación entre la matriz de transformación y las ecuaciones en función de las coordenadas articulares $q = [q_1, q_2, \dots, q_n]$, es posible despejar las n variables articulares q_i en función de las componentes de los vectores **n**, **o**, **a** y **p**:

$$\begin{bmatrix} \mathbf{n} & \mathbf{o} & \mathbf{a} & \mathbf{p} \\ 0 & 0 & 0 & 1 \end{bmatrix} = [t_{ij}(q_i \dots q_n)]$$

Desarrollo con *EJS*.

El método más conveniente para el desarrollo de la cinemática inversa en *EJS* es el geométrico. Por tanto, se tendrá que desarrollar el algoritmo en *Matlab* o *Java*, e implementarlo de la misma manera que se ha realizado con la cinemática directa. Dicho algoritmo se ejecutará cada vez que el usuario

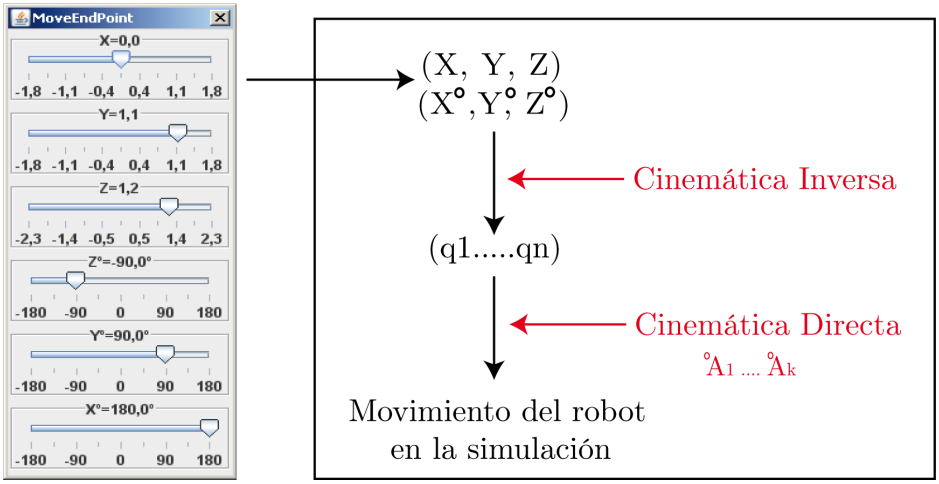

Figura 4.5: Cinemática Inversa

cambie el valor de los controles que dan una posición y orientación al punto final del robot (Figura 4.5).

A la hora de programar la cinemática inversa mediante métodos geométricos, es necesario tener en cuenta las distintas configuraciones del robot: codo arriba o codo abajo para el posicionamiento en el espacio (3 primeros gdl) y muñeca arriba o muñeca abajo para la orientación (3 últimos gdl). El archivo *CinInv6.m*, contenido en el CD, muestra un ejemplo de programación en código *Matlab* del modelo cinemático inverso de un robot de 6gdl. Como ejemplo de programación en *Java*, mostramos a continuación la resolución de la cinemática inversa de un robot de 3gdl, con dos articulaciones rotacionales y una traslacional (RRP):

```
public void cinematicaInversa (px,py,pz)
{
//Calculamos q1. Rotacional
q1=Math.atan2(py,px);

//Calculamos q2. Rotacional
q2=Math.atan2(Math.sqrt(px*px+py*py),pz-l1);

//Calculamos q3. Traslacional
q3= Math.cos(q2)*(pz-l1)+Math.sin(q2)*Math.sqrt(px*px+py*py)-l3;

//Pasamos a grados el valor de las variables rotacionales
q1d = q1*180/Math.PI;
q2d = q2*180/Math.PI;

//Llamamos a la cinemática directa que mueve el robot con los nuevos valores q1,q2,q3
cinematicaDirecta ();
}
```

4.4. Planificador de trayectorias

Concepto Teórico.

El *control cinemático* o la *planificación de trayectorias* consiste en describir el movimiento deseado del manipulador como una secuencia de puntos en el espacio. El control cinemático interpola el camino deseado mediante una clase de funciones polinomiales y genera una secuencia de puntos a lo largo del tiempo.

En el problema de la planificación de trayectorias es necesario tener en consideración las prestaciones reales de los actuadores, de tal manera que el movimiento de un robot sea suave y coordinado. Para obtener un planificador que funcione correctamente, es necesario (Figura 4.6):

1. Estudiar las necesidades de movimiento especificadas por el usuario, evitando colisiones con el entorno. Posteriormente, obtener una expresión analítica en coordenadas cartesianas de la trayectoria deseada en función del tiempo (libre de colisiones).

2. Muestrear la trayectoria anterior en una serie de puntos especificado por sus componentes cartesianas de posición y de orientación $(x, y, z, \alpha, \beta, \gamma)$.

3. Pasar cada uno de esos puntos a coordenadas articulares del robot, utilizando para ello la cinemática inversa (ver sección 4.3).

4. Realizar la interpolación entre los puntos de las coordenadas articulares y obtener para cada articulación una expresión del tipo $q_i(t)$ para cada segmento de control.

Para asegurar la suavidad en las trayectorias articulares, es conveniente añadir restricciones en la interpolación de velocidad y aceleración de paso por los puntos, además de las necesarias condiciones de posición-tiempo.

Existen varios tipos de interpoladores para unir la sucesión de puntos en el espacio articular. Entre los existentes, cabe destacar:

1. Interpoladores lineales. Este interpolador consiste en mantener constante la velocidad de movimiento entre cada dos valores sucesivos (q_{i-1}, q_i)

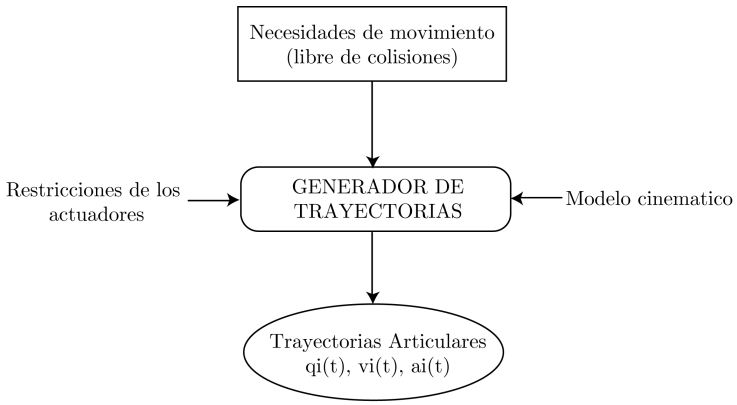

Figura 4.6: Esquema del planificador de trayectorias

de la articulación. La trayectoria entre dos puntos sería:

$$q(t) = \frac{(q_i - q_{i-1})}{T} \cdot dt + q_{i-i} = V_q \cdot dt + q_{i-i}$$

Esta trayectoria asegura la continuidad de la posición, sin embargo origina saltos bruscos en la velocidad V_q de la articulación, provocando aceleraciones de salto infinito. Aunque en la práctica este interpolador no es posible, en el siguiente punto se explicará cómo implementarlo en la simulación.

2. Interpoladores cúbicos o *splines*. Este tipo de interpoladores aseguran que la trayectoria que une los puntos por los que tiene que pasar la articulación presente continuidad en velocidad. La trayectoria está constituida por un polinomio de grado 3 entre cada pareja de puntos adyacentes:

$$q(t) = a + b \cdot (t - t_{i-1}) + c \cdot (t - t_{i-1})^2 + d \cdot (t - t_{i-1})^3$$

De este modo, al tener cuatro parámetros disponibles (a, b, c, d) es posible imponer cuatro condiciones de contorno, dos de posición y dos de velocidad, asegurando de este modo la continuidad en velocidad. Si se desea asegurar la continuidad en aceleración, se debe plantear polinomios de mayor grado.

3. Interpoladores a tramos. Este tipo de interpolador proporciona una alternativa para los de tipo cúbico, ya que en ellos la velocidad de la articulación durante el recorrido está variando continuamente, exigiendo un control continuo de la misma. Este interpolador consiste en descomponer en tres tramos consecutivos la trayectoria de une dos puntos

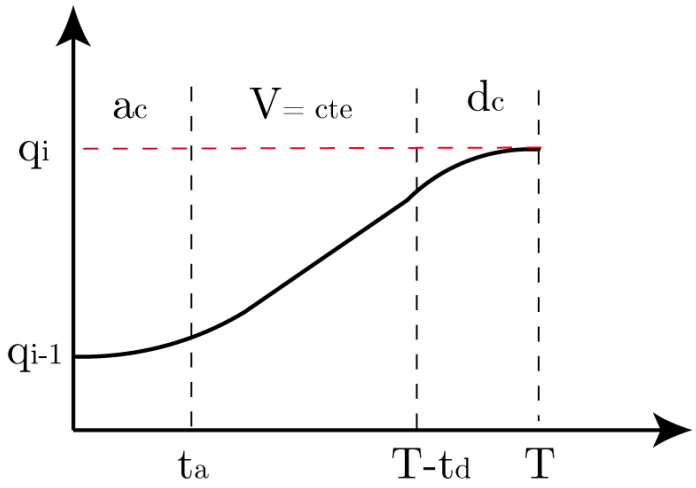

Figura 4.7: Interpolador a tramos

(q_{i-1}, q_i). Un tramo de aceleración, uno central donde la velocidad se mantiene constante y otro de deceleración (Figura 4.7). Este interpolador suele componerse por un polinomio de segundo grado en los tramos de inicial y final (aceleración y deceleración constantes), y un polinomio de primer grado en el central (velocidad constante y aceleración nula).

Desarrollo con *EJS*.

En este apartado se va a explicar cómo implementar distintos interpoladores de trayectorias articulares en la simulación de un laboratorio virtual robótico mediante *EJS*. Al igual que la cinemática directa e inversa, hay dos posibilidades para el desarrollo de los interpoladores:

1. Mediante *Matlab*. En este caso, *EJS* tan sólo se ocuparía de recoger los valores de q_i, v_i y a_i provenientes del *script* donde se encuentra desarrollado el interpolador (archivo *.m*, y de actualizar la vista de la simulación.

2. Mediante las herramientas de *EJS*. En esta opción, será necesario utilizar el panel evolución de *EJS* (ver apéndice A.3). Dentro de este panel se incluye un editor de EDOs (*Ecuaciones Diferenciales Ordinarias*) y una página para el desarrollo de algoritmos en *Java*, todo para la definición de la evolución de las variables del sistema.

A continuación se muestra la explicación del desarrollo de distintos interpoladores, indicándose el método empleado para su implementación.

Interpolador Lineal (Panel Evolución).

Este tipo de interpolador es el más sencillo de implementar. Dentro del mismo, hay que distinguir entre dos variantes:

1. Trayectoria *síncrona*. Todas las articulaciones invierten el mismo tiempo en su movimiento, acabando todas ellas simultáneamente. De este modo, todas las articulaciones se coordinan comenzando y acabando su movimiento a la vez, adaptando su velocidad al tiempo de la trayectoria.

2. Trayectoria *asíncrona*. Las articulaciones comienzan simultáneamente su movimiento, pero cada una de ellas con una velocidad específica dependiendo del tiempo a emplear en su trayectoria. De este manera, cada una de las articulaciones acabará su movimiento en un instante diferente.

Para el desarrollo de la trayectoria síncrona, tan sólo es necesario calcular el valor de la velocidad que tiene que emplear cada articulación para realizar la trayectoria en el tiempo determinado (mismo para todas) e incrementar su valor en el panel evolución según dicha velocidad en un diferencial de tiempo ($q_i = q_i + V_{syn} * dt$).

En el caso de la trayectoria asíncrona, hay que calcular la velocidad específica de cada articulación según su tiempo de ejecución. Aquí se observa el código de dicho cálculo para un robot de 5gdl, el cual se debe implementar en una función dentro del panel "Propio" de *EJS*.

```
//Punto Inicial de la trayectoria
qInit[0] = q1d;qInit[1] = q2d;qInit[2] = q3d;qInit[3] = q4d;qInit[4] = q5d;

//Punto Final de la trayectoria
qFin[0] = q1vfd;qFin[1] = q2vfd;qFin[2] = q3vfd;qFin[3] = q4vfd;qFin[4] = q5vfd;

//Tiempo de cada trayectoria
timePath[0] = t1; timePath[1] = t2; timePath[2] = t3; timePath[3] = t4; timePath[4] = t5;

//Calculo de las velocidades articulares
for (int i =0;i<5;i++)
{
```

```
   //Ponemos una condicion para que no se produzcan valores NaN
   if(timePath[i]==0) velSync[i] = 0;
   else  velSync[i] = (qFin[i] - qInit[i])/(timePath[i]*0.001);//grados/segundo
}
```

Con las velocidades calculadas, tan sólo hay que actualizar el valor de las variables articulares en el panel evolución según dichas velocidades. El código que mostramos a continuación realiza lo comentado:

```
//Contador de las iteraciones del panel evolución
inc=inc+1;

//Actualización de las variables según su velocidad
q1d = q1d+velSync[0]*dt1;
q2d = q2d+velSync[1]*dt2;
q3d = q3d+velSync[2]*dt3;
q4d = q4d+velSync[3]*dt4;
q5d = q5d+velSync[4]*dt5;

//Pasamos a radianes los valores articulares
q1 = q1d*Math.PI/180;
q2 = q2d*Math.PI/180;
q3 = q3d*Math.PI/180;
q4 = q4d*Math.PI/180;
q5 = q5d*Math.PI/180;

//Actualizamos el vector DHq (este vector lo utiliza directCinematic() para mover el robot)
DHq[0] = q1;
DHq[1] = q2-Math.PI/2;
DHq[2] = q3+12.4*Math.PI/180;
DHq[3] = q4-18.0*Math.PI/180;
DHq[4] = q5-2.0*Math.PI/180;

//Incrementamos la variable tiempo
time = time+dt;

//Llamamos a la cinemática directa para mover el robot
directCinematic();

//Para detener a las articulaciones en su valor qfinal
if(inc==(int)(IPS*timePath[0]/1000)) dt1=0;
if(inc==(int)(IPS*timePath[1]/1000)) dt2=0;
if(inc==(int)(IPS*timePath[2]/1000)) dt3=0;
if(inc==(int)(IPS*timePath[3]/1000)) dt4=0;
if(inc==(int)(IPS*timePath[4]/1000)) dt5=0;
```

Con respecto al código anterior, es necesario comentar que la evolución de q_i hay que detenerla en el momento en el que ha llegado a su punto q_{final}. Para esta acción, se ha utilizado un contador de las iteraciones del panel evolución, que se compara con el valor de tiempo de cada articulación haciendo uso de las IPS (*Imágenes Por Segundo*) de la simulación. Por esta razón se ha empleado una variable distinta en el diferencial de tiempo para cada articulación ($dt1$, $dt2$, $dt3$, $dt4$, $dt5$).

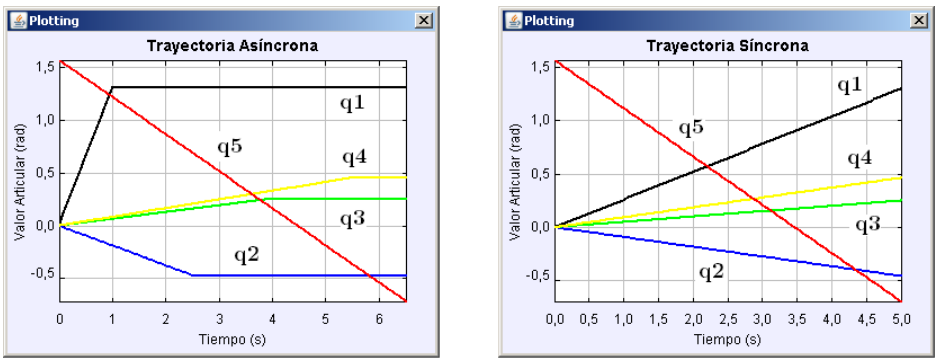

Figura 4.8: Trayectoria asíncrona y síncrona

En la Figura 4.8, se muestra un gráfico obtenido de la simulación del interpolador donde puede verse la diferencia entre la trayectoria síncrona y la asíncrona.

Interpolador 4-3-4 (*Matlab*).

El interpolador 4-3-4 es un interpolador a tramos que divide la trayectoria articular en tres segmentos: la aceleración del motor (fase 1), periodo de velocidad máxima (fase 2) y frenado del motor (fase 3). Los polinomios correspondientes a cada uno de los tramos de esta trayectoria son los que se observa a continuación:

$$\begin{aligned} h_1(t) &= a_{14} \cdot t^4 + a_{13} \cdot t^3 + a_{12} \cdot t^2 + a_{11} \cdot t + a_{10} \\ h_2(t) &= a_{23} \cdot t^3 + a_{22} \cdot t^2 + a_{21} \cdot t + a_{20} \\ h_3(t) &= a_{34} \cdot t^4 + a_{33} \cdot t^3 + a_{32} \cdot t^2 + a_{31} \cdot t + a_{30} \end{aligned}$$

El desarrollo y resolución de estos polinomios se puede encontrar en [Fu y otros, 2002]. Además en el CD está desarrollado el planificador en código *Matlab* (archivo *planificador.m*) de manera que pueda ser implementado en una simulación de *EJS*. Parte de este código se ha extraído de [Saltarén y otros, 2000].

Los parámetros de entrada a este planificador son las velocidades máximas de cada articulación y los tiempos de aceleración y deceleración. Aquí se muestra el código con el que desde *EJS* se obtienen los valores de las posiciones, velocidades y aceleraciones necesarias en la planificación de la trayectoria a partir del punto inicial y final.

```
//Escribimos en el workspace de Matlab la posición inicial de la trayectoria
_external.setValue("q1pi",q1pir);
_external.setValue("q2pi",q2pir);
_external.setValue("q3pi",q3pir);
_external.setValue("q4pi",q4pir);
_external.setValue("q5pi",q5pir);
_external.setValue("q6pi",q6pir);

//Escribimos en el workspace de Matlab la posición final de la trayectoria
_external.setValue("q1pf",q1pfr);
_external.setValue("q2pf",q2pfr);
_external.setValue("q3pf",q3pfr);
_external.setValue("q4pf",q4pfr);
_external.setValue("q5pf",q5pfr);
_external.setValue("q6pf",q6pfr);

//Escribimos en el workspace de Matlab los valores de velocidad máxima
_external.setValue("vq1max",vq1max);
_external.setValue("vq2max",vq2max);
_external.setValue("vq3max",vq3max);
_external.setValue("vq4max",vq4max);
_external.setValue("vq5max",vq5max);
_external.setValue("vq6max",vq6max);

//Escribimos en el workspace de Matlab los valores de tiempo de acel. y frenado
_external.setValue("tacel",tacel);
_external.setValue("tfren",tfren);

//Llamamos al planificador de Matlab
% BEGIN CODE:
planificador;
% END CODE

//Leemos del workspace los valores articulares
pos_plan1 = _external.getDoubleArray("pos_plan1");
pos_plan2 = _external.getDoubleArray("pos_plan2");
pos_plan3 = _external.getDoubleArray("pos_plan3");
pos_plan4 = _external.getDoubleArray("pos_plan4");
pos_plan5 = _external.getDoubleArray("pos_plan5");
pos_plan6 = _external.getDoubleArray("pos_plan6");

//Leemos del workspace las velocidades articulares
vel_plan1 = _external.getDoubleArray("vel_plan1");
vel_plan2 = _external.getDoubleArray("vel_plan2");
vel_plan3 = _external.getDoubleArray("vel_plan3");
vel_plan4 = _external.getDoubleArray("vel_plan4");
vel_plan5 = _external.getDoubleArray("vel_plan5");
vel_plan6 = _external.getDoubleArray("vel_plan6");

//Leemos del workspace las aceleraciones articulares
ace_plan1 = _external.getDoubleArray("ace_plan1");
ace_plan2 = _external.getDoubleArray("ace_plan2");
ace_plan3 = _external.getDoubleArray("ace_plan3");
ace_plan4 = _external.getDoubleArray("ace_plan4");
ace_plan5 = _external.getDoubleArray("ace_plan5");
ace_plan6 = _external.getDoubleArray("ace_plan6");
```

Interpolador splin quíntico (Panel Evolución).

El *splin quíntico* es un interpolador de quinto grado que garantiza la continuidad de las aceleraciones y permite interponer las velocidades articulares como condiciones de contorno [Barrientos y otros, 1997]. Para su implementación en *EJS*, se va a hacer uso del editor de ecuaciones diferenciales ordinales (Panel ODEs, ver Apéndice A).

Como se observa en la ecuación siguiente, el interpolador posee 6 parámetros. Por lo tanto, para hallar el valor de las incógnitas es necesario interponer 6 condiciones de contorno, 3 para el instante inicial, 3 para el instante final.

$$q(t) = a_5 \cdot (t-t_0)^5 + a_4 \cdot (t-t_0)^4 + a_3 \cdot (t-t_0)^3 + a_2 \cdot (t-t_0)^2 + a_1 \cdot (t-t_0) + a_0$$

A continuación, se expone la resolución de los parámetros, necesarios para la implementación del interpolador en *EJS*. Se ha supuesto que $t_0 = 0$ para el instante inicial.

- **Condición 1**. Para $t = 0$, $q = q_i$.

$$\begin{aligned} q(t) &= a_5 \cdot t^5 + a_4 \cdot t^4 + a_3 \cdot t^3 + a_2 \cdot t^2 + a_1 \cdot t + a_0 \\ q_i &= a_0 \\ a_0 &= q_i \end{aligned}$$

- **Condición 2**. Para $t = 0$, $V = V_i$.

$$\begin{aligned} \frac{dq}{dt} &= V = 5a_5 \cdot t^4 + 4a_4 \cdot t^3 + 3a_3 \cdot t^2 + 2a_2 \cdot t + a_1 \\ V_i &= a_1 \\ a_1 &= V_i \end{aligned}$$

- **Condición 3**. Para $t = 0$, $A = 0$.

$$\begin{aligned} \frac{dV}{dt} &= A = 20a_5 \cdot t^3 + 12a_4 \cdot t^2 + 6a_3 \cdot t + 2a_2 \\ A &= 2a_2 \\ a_2 &= 0 \end{aligned}$$

- **Condición 4**. Para $t = t_f$ (tiempo **f**inal), $q = q_f$ (valor articular **f**inal).

$$\begin{aligned} q(t) &= a_5 \cdot t^5 + a_4 \cdot t^4 + a_3 \cdot t^3 + a_2 \cdot t^2 + a_1 \cdot t + a_0 && (4.1)\\ q_f &= a_5 \cdot t_f^5 + a_4 \cdot t_f^4 + a_3 \cdot t_f^3 + a_2 \cdot t_f^2 + a_1 \cdot t_f + a_0 && (4.2)\\ q_f &= a_5 \cdot t_f^5 + a_4 \cdot t_f^4 + a_3 \cdot t_f^3 + V_i \cdot t_f + q_i && (4.3) \end{aligned}$$

- **Condición 5**. Para $t = t_f$, $V = V_f$.

$$\begin{aligned} V &= 5a_5 \cdot t^4 + 4a_4 \cdot t^3 + 3a_3 \cdot t^2 + 2a_2 \cdot t + a_1 && (4.4)\\ V_f &= 5a_5 \cdot t_f^4 + 4a_4 \cdot t_f^3 + 3a_3 \cdot t_f^2 + 2a_2 \cdot t_f + a_1 && (4.5)\\ V_f &= 5a_5 \cdot t_f^4 + 4a_4 \cdot t_f^3 + 3a_3 \cdot t_f^2 + V_i && (4.6) \end{aligned}$$

- **Condición 6**. Para $t = t_f$, $A = 0$.

$$\begin{aligned} A &= 20a_5 \cdot t^3 + 12a_4 \cdot t^2 + 6a_3 \cdot t + 2a_2 && (4.7)\\ 0 &= 20a_5 \cdot t_f^3 + 12a_4 \cdot t_f^2 + 6a_3 \cdot t_f + 2a_2 && (4.8)\\ 0 &= 20a_5 \cdot t_f^3 + 12a_4 \cdot t_f^2 + 6a_3 \cdot t_f && (4.9) \end{aligned}$$

Con las condiciones de contorno del instante inicial, se ha podido hallar el valor de a_0, a_1 y a_2. Con respecto al instante final (ecuaciones 4.1-4.9), ha resultado un sistema de tres ecuaciones (4.3, 4.6, 4.9) con las tres incógnitas a_3, a_4 y a_5. Resolviendo el sistema matricial:

$$\begin{bmatrix} t_f^5 & t_f^4 & t_f^3 \\ 5t_f^4 & 4t_f^3 & 3t_f^2 \\ 20t_f^3 & 12t_f^2 & 6t_f \end{bmatrix} \cdot \begin{bmatrix} a_5 \\ a_4 \\ a_3 \end{bmatrix} = \begin{bmatrix} q_f - V_i \cdot t_f - q_i \\ V_f - V_i \\ 0 \end{bmatrix}$$

el valor resultante de a_3, a_4 y a_5 es:

$$\begin{bmatrix} a_5 \\ a_4 \\ a_3 \end{bmatrix} = \begin{bmatrix} t_f^5 & t_f^4 & t_f^3 \\ 5t_f^4 & 4t_f^3 & 3t_f^2 \\ 20t_f^3 & 12t_f^2 & 6t_f \end{bmatrix}^{-1} \cdot \begin{bmatrix} q_f - V_i \cdot t_f - q_i \\ V_f - V_i \\ 0 \end{bmatrix}$$

$$\begin{aligned} a_5 &= \frac{-6 \cdot q_i - 3 \cdot V_i \cdot t_f + 6 \cdot q_f - 3 \cdot V_f \cdot t_f}{t_f^5} \\ a_4 &= \frac{15 \cdot q_i + 8 \cdot V_i \cdot t_f - 15 \cdot q_f + 7 \cdot V_f \cdot t_f}{t_f^4} \\ a_3 &= \frac{-10 \cdot q_i - 6 \cdot V_i \cdot t_f + 10 \cdot q_f - 4 \cdot V_f \cdot t_f}{t_f^3} \end{aligned}$$

Teniendo ya el valor de los parámetros del interpolador, se dispone para su implementación en *EJS*. Para ello, será necesario utilizar el editor de EDOs para definir las ecuaciones cinemáticas que relacionan la posición, velocidad y aceleración:

$$\begin{aligned} \frac{dq}{dt} &= V\,(rad/s) \\ \frac{dV}{dt} &= A\,(rad/s^2) \end{aligned}$$

Aquí se observa un ejemplo de cómo quedaría el Panel de EDOs de las ecuaciones cinemáticas anteriores para un robot de 3gdl (Figura 4.9):

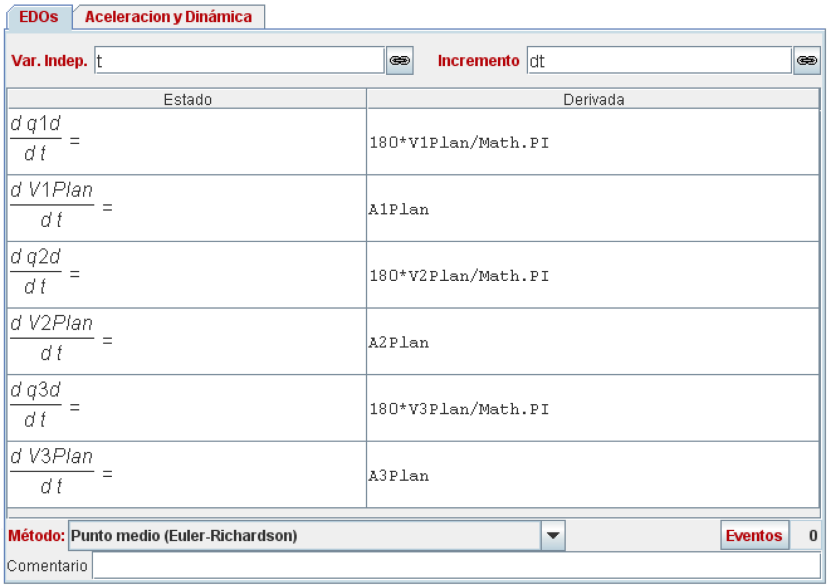

Figura 4.9: Panel de EDOs

La ventaja de utilizar el Panel de EDOs es que tan sólo es necesario calcular el valor de la aceleración articular para cada iteración. *EJS* se encarga de hallar el valor de la velocidad y posición articular a través de la resolución de las ecuaciones diferenciales. De este modo, con el valor de los parámetros a_3, a_4 y a_5 resueltos para unas condiciones de contorno determinadas, se calcula la aceleración cada diferencial de tiempo y se consigue en la simulación el movimiento según un interpolador quíntico. Aquí se muestra el código del cálculo de las aceleraciones para un robot de 3gdl en el panel evolución a partir de a_3, a_4, a_5 previamente calculados:

```
A1Plan = 20*a5[0]*Math.pow(t,3) + 12*a4[0]*Math.pow(t,2) + 6*a3[0]*t;
A2Plan = 20*a5[1]*Math.pow(t,3) + 12*a4[1]*Math.pow(t,2) + 6*a3[1]*t;
A3Plan = 20*a5[2]*Math.pow(t,3) + 12*a4[2]*Math.pow(t,2) + 6*a3[2]*t;
```

A continuación se muestra las gráficas de la posición, velocidad y aceleración con respecto al tiempo del interpolador, donde se puede observar la suavidad y continuidad del movimiento:

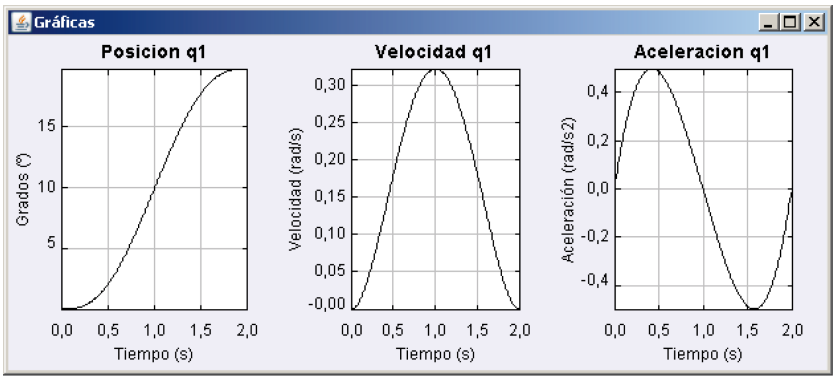

Figura 4.10: Evolución temporal del splin quíntico

Interpolador a tramos (Panel Evolución).

Tal y como se ha comentado en la introducción teórica de este punto, el interpolador a tramos consiste en descomponer la trayectoria articular en tres fragmentos consecutivos: aceleración, velocidad constante y deceleración. Al igual que el caso anterior, se ha empleado el Panel de EDOs y de evolución para su implementación en *EJS*.

Las ecuaciones diferenciales son las mismas que para el *splin quíntico* (Figura 4.9). La diferencia es que la aceleración cambia según el tramo donde se encuentre la trayectoria. Por lo tanto, durante la simulación del movimiento será necesario distinguir entre el tiempo de aceleración, de deceleración y de velocidad constante.

Previamente al inicio de la trayectoria, es necesario calcular la velocidad de sincronización de cada uno de los ejes. De esta manera, todas las articulaciones acabarán simultáneamente invirtiendo el mismo tiempo en su movimiento. A continuación se muestra el código *Java* que realiza dicho cálculo a partir de los puntos inicial y final de trayectoria ($qiPlan$, $qfPlan$) y de las

velocidades máximas de cada eje ($Vq1maxT$, $Vq2maxT$, $Vq3maxT$) para un robot de 3gdl:

```
//Tomamos los valores iniciales de trayectoria
qiPlan[0] = Math.PI*q1d/180;
qiPlan[1] = Math.PI*q2d/180;
qiPlan[2] = Math.PI*q3d/180;

//Tomamos los valores finales de la trayectoria
qfPlan[0] = Math.PI*q1fd/180;
qfPlan[1] = Math.PI*q2fd/180;
qfPlan[2] = Math.PI*q3fd/180;

//Calculamos el tiempo de la trayectoria a partir de las velocidades máximas
taproxT[0] = Math.abs(qfPlan[0]-qiPlan[0])/Vq1maxT;
taproxT[1] = Math.abs(qfPlan[1]-qiPlan[1])/Vq2maxT;
tmaximoT = Math.max(taproxT[0], taproxT[1]);
taproxT[2] = Math.abs(qfPlan[2]-qiPlan[2])/Vq3maxT;
tmaximoT = Math.max(tmaximoT, taproxT[2]);

//Velocidades de sincronización de cada articulación
VsincroT[0] = (qfPlan[0]-qiPlan[0])/tmaximoT;
VsincroT[1] = (qfPlan[1]-qiPlan[1])/tmaximoT;
VsincroT[2] = (qfPlan[2]-qiPlan[2])/tmaximoT;
```

Teniendo ya el valor de la velocidad de sincronización para cada eje, se asociará una aceleración en cada uno de los tramos dependiendo de la velocidad inicial y final de la trayectoria ($ViPlan$, $VfPlan$), y del tiempo de aceleración y deceleración (taT, tdT). Aquí se muestra el código del panel evolución para un robot de 3gdl que distingue los tramos de la trayectoria y asocia un valor de aceleración distinto para cada uno de ellos.

```
//Tramo de aceleración
if(t<=taT){
  A1Plan = (VsincroT[0]-ViPlan[0])/taT;
  A2Plan = (VsincroT[1]-ViPlan[1])/taT;
  A3Plan = (VsincroT[2]-ViPlan[2])/taT;
}

//Tramo de velocidad constante
if(taT<t && t<(tmaximoT-tdT)){
  A1Plan = 0;
  A2Plan = 0;
  A3Plan = 0;
}

//Tramo de deceleración
if(t>=(tmaximoT-tdT)){
  A1Plan = (VfPlan[0]-VsincroT[0])/tdT;
  A2Plan = (VfPlan[1]-VsincroT[1])/tdT;
  A3Plan = (VfPlan[2]-VsincroT[2])/tdT;
}
```

Finalmente, se muestra el resultado de la evolución temporal de la posición, velocidad y aceleración (Figura 4.11).

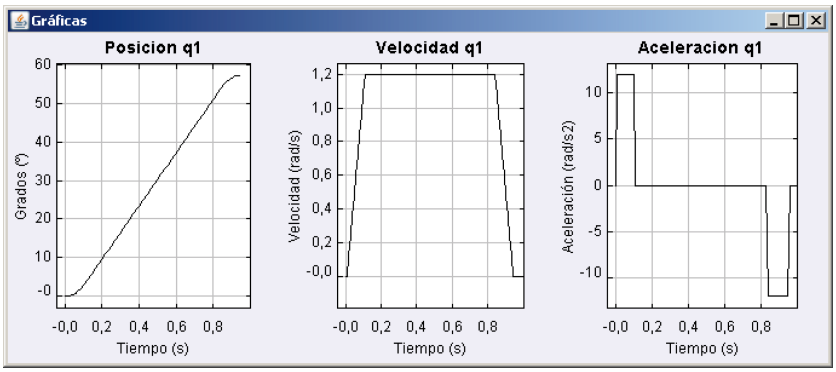

Figura 4.11: Evolución temporal del interpolador a tramos

4.5. Dinámica Inversa

Concepto Teórico.

La dinámica del robot relaciona el movimiento del robot y las fuerzas implicadas en el mismo. El modelo dinámico establece relaciones matemáticas entre las coordenadas articulares (o las coordenadas del extremo del robot), sus derivadas (velocidad y aceleración), las fuerzas y los pares aplicados en las articulaciones (o en el extremo) y los parámetros del robot (masas e inercias).

El *modelo dinámico inverso* expresa las fuerzas y pares que intervienen en función de la evolución de las coordenadas articulares y sus derivadas. Existen varias formulaciones para modelar la dinámica inversa del robot manipulador. Entre ellas está la formulación *Lagrange-Euler*, que presenta un modelo simple y elegante, dando como resultado una serie de ecuaciones diferenciales no lineales de 2º orden acopladas útiles para el estudio de estrategias de control en el espacio de estados de las variables articulares del robot. Sin embargo, este método no se presenta eficaz para aplicaciones en tiempo real dado el elevado tiempo de computación que requieren sus operaciones matriciales. El método de *Newton-Euler* desarrollado por Luh [Fu y otros, 2002], permite obtener un conjunto de ecuaciones recursivas hacia delante de velocidad y aceleración lineal y angular, referidas a cada sistema de referencia

articular. Las velocidades y aceleraciones de cada elemento se propagan hacia delante desde el sistema de referencia de la base hasta el efector final. Las ecuaciones recursivas hacia atrás calculan los pares y fuerzas necesarios para cada articulación desde la mano (incluyendo en ella efectos de fuerzas externas), hasta el sistema de referencia de la base.

El algoritmo *Newton-Euler* se basa en operaciones vectoriales, siendo más eficiente en comparación con las operaciones matriciales asociadas a la formulación Lagrangiana. Además, el orden de complejidad computacional de la formulación recursiva *Newton-Euler* es $O(n)$, mucho menos que el orden de $O(n^4)$ de la formulación Lagrangiana.

El algoritmo computacional *Newton-Euler*, que se encuentra más detallado en las referencias [Barrientos y otros, 1997] y [Saltarén y otros, 2000], se desarrolla en los siguientes pasos:

1. Asigna a cada eslabón un sistema de referencia D-H.

2. Obtiene las matrices de rotación ${}^{i-1}R_i$ y sus inversas ${}^{i}R_{i-1} = ({}^{i-1}R_i)^{-1}$. Dado que son matrices ortonormales, su inversa es la traspuesta ${}^{i}R_{i-1} = ({}^{i-1}R_i)^{T}$.

3. Establece las condiciones iniciales sobre el sistema de la base $\{S_0\}$.

$$
\begin{array}{rcc}
{}^{0}\vec{w}_0 = & [0,0,0]^T & Velocidad\,angular \\
{}^{0}\dot{\vec{w}}_0 = & [0,0,0]^T & Aceleración\,angular \\
{}^{0}\vec{v}_0 = & [0,0,0]^T & Velocidad\,lineal \\
{}^{0}\dot{\vec{v}}_0 = & [g_x,g_y,g_z]^T & Aceleración\,lineal \\
\vec{z}_0 = & [0,0,1]^T & Vector\,director\,\vec{z} \\
{}^{i}\vec{p}_i = & [a_i, d_i \sin q_i, d_i \cos q_i] & Vector\,entre\,sistemas\,D-H \\
{}^{i}\vec{s}_i = & \frac{-{}^{i}p_i}{2} & Vector\,del\,centro\,de\,masas\,del\,sistema\,\{S_i\} \\
{}^{i}I_i = & [I_{xx}; I_{yy}; I_{zz}] & Matriz\,de\,Inercias
\end{array}
$$

   Para el extremo del robot se conocerá la fuerza y par ejercidos externamente ${}^{n+1}f_{n+1}$ y ${}^{n+1}n_{n+1}$ (normalmente este dato proviene de un sensor de fuerzas en el extremo). Teniendo ya los datos iniciales, comienza el algoritmo recursivo hacia delante.

Para $i = 1 \ldots n$ el algoritmo realiza los pasos del 4 al 7:

4. Obtiene la velocidad angular del sistema $\{S_i\}$:

$$ {}^{i}\vec{w}_i = \begin{cases} {}^{i-1}R_i\,({}^{i-1}\vec{w}_{i-1} + \vec{z}_0\dot{q}_i) & i\ rotacional \\ {}^{i-1}R_i\,{}^{i-1}\vec{w}_{i-1} & i\ traslacional \end{cases} $$

5. Obtiene la aceleración angular del sistema $\{S_i\}$:

$$ {}^{i}\dot{\vec{w}}_i = \begin{cases} {}^{i-1}R_i\left({}^{i-1}\dot{\vec{w}}_{i-1} + \vec{z}_0\ddot{q}_i\right) + {}^{i-1}\,\vec{w}_{i-1} \times \vec{z}_0\dot{q}_i & i\ rotacional \\ {}^{i-1}R_i\,{}^{i-1}\dot{\vec{w}}_{i-1} & i\ traslacional \end{cases} $$

6. Obtiene la aceleración lineal del sistema $\{S_i\}$:

$$ {}^{i}\dot{\vec{v}}_i = \begin{cases} {}^{i}\dot{\vec{w}}_i \times {}^{i}\,\vec{p}_i + {}^{i}\,\vec{w}_i \times ({}^{i}\vec{w}_i \times {}^{i}\,\vec{p}_i) + {}^{i}\,R_{i-1}\,{}^{i-1}\dot{\vec{v}}_{i-1} & i\ rotacional \\ {}^{i}R_{i-1}\left(\vec{z}_0\ddot{q}_i + {}^{i-1}\,\dot{\vec{v}}_{i-1}\right) + {}^{i}\,\dot{\vec{w}}_i \times {}^{i}\,\vec{p}_i + 2{}^{i}\vec{w}_i \times {}^{i}\,R_{i-1}\,\vec{z}_0\dot{q}_i + \\ + {}^{i}\vec{w}_i \times ({}^{i}\vec{w}_i \times {}^{i}\,\vec{p}_i) & i\ traslacional \end{cases} $$

7. Obtiene la aceleración lineal del centro de gravedad del eslabón i:

$$ {}^{i}\vec{a}_i = {}^{i}\,\dot{\vec{w}}_i \times {}^{i}\,\vec{s}_i + {}^{i}\,\vec{w}_i \times \left({}^{i}\vec{w}_i \times {}^{i}\,\vec{s}_i\right) + {}^{i}\,\dot{\vec{v}}_i $$

Llegado este punto, se comienza el algoritmo recursivo hacia detrás. Para $i = n \ldots 1$ el algoritmo realiza los pasos del 8 al 10:

8. Obtiene la fuerza ejercida sobre el eslabón i:

$$ {}^{i}\vec{f}_i = \begin{cases} f_{externa} & i = n+1 \\ {}^{i}R_{i+1}\,{}^{i+1}\vec{f}_{i+1} + m_i\,{}^{i}\vec{a}_i & i = n \ldots 1 \end{cases} $$

9. Obtiene el par ejercido sobre el eslabón i:

$$ {}^{i}\vec{n}_i = {}^{i}\,R_{i+1}\left[{}^{i+1}\vec{n}_i + \left({}^{i+1}R_i\,{}^{i}\vec{p}_i\right) \times {}^{i+1}\,\vec{f}_{i+1}\right] + \left({}^{i}\vec{p}_i + {}^{i}\,\vec{s}_i\right) \times m_i\,{}^{i}\vec{a}_i + {}^{i}I_i\,{}^{i}\dot{\vec{w}}_i + {}^{i}\vec{w}_i \times \left({}^{i}I_i\,{}^{i}\dot{\vec{w}}_i\right) $$

10. Obtiene la fuerza o par aplicado a la articulación i:

$$ {}^{i}\vec{\tau}_i = \begin{cases} {}^{i}\vec{n}_i{}^{T}\,{}^{i}R_{i-1}\vec{z}_0 & i\ rotacional \\ {}^{i}\vec{f}_i{}^{T}\,{}^{i}R_{i-1}\vec{z}_0 & i\ traslacional \end{cases} $$

Donde τ es el par efectivo, es decir el par motor menos los pares de rozamiento o perturbación.

Desarrollo con *EJS*.

Dada la simplicidad del algoritmo y que las operaciones no conllevan muchos cálculos matriciales, la implementación del algoritmo en *EJS* puede realizarse tanto en *Matlab* (mediante archivos *.m*) como en *Java* (utilizando la librería *vectmath.jar*). Aquí se mostrará parte del código para su desarrollo en *Matlab* para un robot de 6gdl (archivo *dinamica6.m*).

Los parámetros de entrada al algoritmo son las posiciones, velocidades y aceleraciones articulares que provienen del planificador. También es necesario calcular el valor de las masas, las inercias de los eslabones y las matrices de rotación ${}^{i-1}R_i$ entre sistemas D-H. Para el cálculo de las masas y de las inercias, se puede suponer que los eslabones son cilindros macizos de un determinado material. El cálculo de las inercias debe realizarse con respecto a los ejes de su sistema D-H. Aquí se ve un ejemplo:

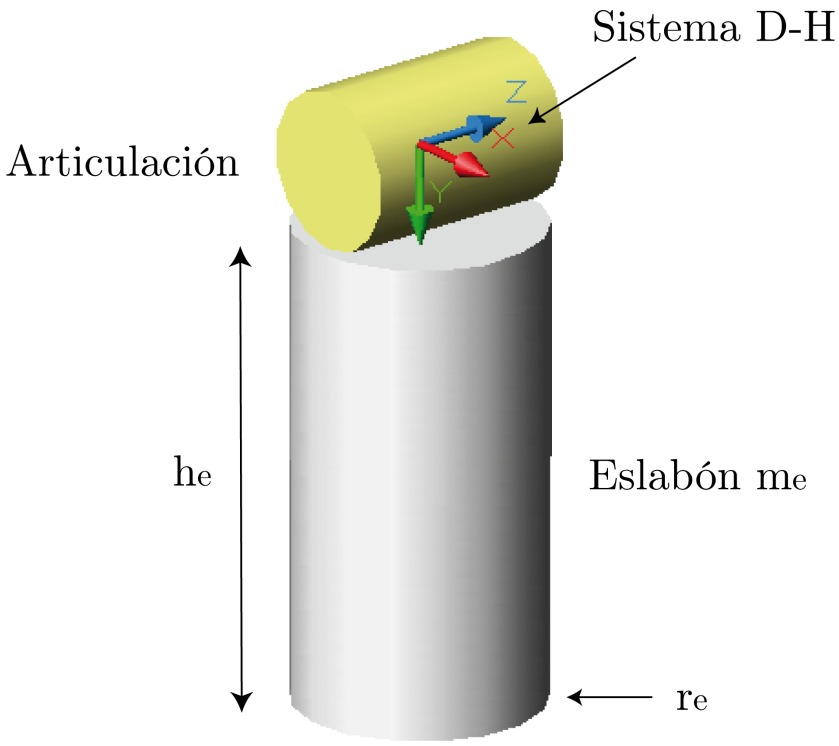

Figura 4.12: Propiedades geométricas del eslabón

Si se supone una densidad de σ para el material del eslabón, el cálculo de la masa y de la matriz de inercias con respecto al sistema D-H de la Figura

4.12 es el siguiente:

$$\begin{aligned} m_e &= \sigma\pi r_e^2 h_e \\ I_{xx} &= \left[\frac{1}{4}m_e r_e^2 + \frac{1}{12}m_e h_e^2\right] + \left(\frac{h_e}{2}\right)^2 m_e \\ I_{yy} &= \frac{1}{2}m_e r_e^2 \\ I_{zz} &= I_{xx} \end{aligned}$$

El valor de los vectores entre sistemas D-H, de la posición del centro de gravedad y de las matrices de rotación ${}^{i-1}R_i$, pueden obtenerse fácilmente de los valores que proporciona la cinemática directa (archivo *cindir6.m*) que se llama cuando se mueve el robot (ver sección 4.2). Aquí se muestra el código en *Matlab* con el que se construyen dichos valores para un robot de 6gdl:

```
% Matrices de Rotación entre sistemas de referencia D-H
% Los valores de Aij provienen del resultado de cindir6.m
r01 = A01(1:3,1:3);                r10 = r01';
r12 = A12(1:3,1:3);                r21 = r12';
r23 = A23(1:3,1:3);                r32 = r23';
r34 = A34(1:3,1:3);                r43 = r34';
r45 = A45(1:3,1:3);                r54 = r45';
r56 = A56(1:3,1:3);                r65 = r56';

% Operaciones para calcular los vectores
r02 = A02(1:3,1:3);                r20=r02';
r03 = A03(1:3,1:3);                r30=r03';
r04 = A04(1:3,1:3);                r40=r04';
r05 = A05(1:3,1:3);                r50=r05';
r06 = A06(1:3,1:3);                r60=r06';

% Cálculo de los vectores entre sistemas D-H.
% Los valores de pix, piy, piz (i=1...n) provienen del resultado de cindir6.m
r10p1 = r10*[p1x p1y p1z]';
r20p2 = r20*[p2x-p1x p2y-p1y  p2z-p1z]';
r30p3 = [0 0 0]'; %todo ceros pq se encuentra sobre el mismo sistema 2
r40p4 = r40*[p4x-p3x p4y-p3y  p4z-p3z]';
r50p5 = [0 0 0]'; %todo ceros pq se encuentra sobre el mismo sistema 4
r60p6 = r60*[p6x-p5x p6y-p5y p6z-p5z]';

% Cálculo de las posiciones de los centros de masas
r10s1 =  r10p1*(-0.5);
r20s2 =  r20p2*(-0.5);
r30s3 =  r30p3*(-0.5);
r40s4 =  r40p4*(-0.5);
r50s5 =  r50p5*(-0.5);
r60s6 =  r60p6*(-0.5);
```

Teniendo ya el valor de las masas, inercias, matrices de rotación y vectores entres sistemas D-H, se comienza el algoritmo recursivo, calculando primero

las velocidades y aceleraciones, y posteriormente las fuerzas y los pares. Dado que son muchas operaciones, no se exponen en este documento y se remite al lector al archivo *dinamica6.m* contenido en el CD, donde parte del código se ha extraído de [Saltarén y otros, 2000] y de [Corke, 1996].

4.6. Conclusiones

En el contenido de este capítulo se ha explicado cómo desarrollar un laboratorio virtual robótico mediante *EJS*. Se ha comprobado que, a pesar de la complejidad presente en el modelo matemático de un robot, no resulta complicado realizarlo mediante este software. *EJS* proporciona las herramientas necesarias para modelar cualquier tipo de sistema físico, desde los más simples hasta los más complejos. Además, una gran ventaja que se ha podido observar en este capítulo ha sido la capacidad de programación de *EJS*, ya que está basado en *Java* y todo la API de este lenguaje puede ser utilizada.

Se ha descrito cómo modelar la cinemática directa e inversa, la planificación de trayectorias de distintos interpoladores y la dinámica inversa de un robot. La cinemática es la que proporciona el movimiento del robot en la vista de la simulación, que puede realizarse dando valores articulares (cinemática directa) o mediante valores al extremo del robot (cinemática inversa). Se ha implementado distintos interpoladores en el espacio articular para describir el movimiento del robot. A través de ellos, es posible visualizar la evolución temporal de las posiciones, velocidades y aceleraciones articulares. Finalmente, la dinámica inversa proporciona el par necesario en las articulaciones durante la ejecución de una tarea.

Como conclusión, comentar que este capítulo representa sólo un primer paso para el desarrollo de laboratorios virtuales de disposotivos robóticos mediante *EJS*. Y aunque sea sólo una base para la creación de este tipo de aplicaciones, se ha presentado algún aspecto como el modelado de la dinámica, que actualmente se encuentra en desarrollo, que es un tema muy novedoso en los simuladores robóticos de distribución libre.

CAPÍTULO 5

Entornos Colaborativos

5.1. Introducción

La irrupción de las *Tecnologías de la Información y la Comunicación* (TIC) en nuestra sociedad ha sido espectacular, y desde el mundo educativo, no podemos ser ajenos a ello. Prueba fehaciente de este hecho ha sido la aparición de nuevas formas de enseñanza y aprendizaje, tales como los *laboratorios virtuales*. Este innovador objeto de aprendizaje (*Learning Object*) ya ha sido evaluado exitosamente por varios investigadores en la enseñanza virtual universitaria [Torres y otros, 2006].

Tal y como se comentó con anterioridad (Capítulo 2), el laboratorio virtual se considera como una herramienta de auto-aprendizaje que permite al alumno comprender mejor los conceptos y tener una visión más intuitiva de los fenómenos. Sin embargo, la mayoría de los laboratorios virtuales son diseñados para ser usados de forma individual (*laboratorio virtual monolítico*), y no permiten el trabajo en grupo ni la colaboración entre los estudiantes y el profesor.

En los últimos años, se han realizado investigaciones para determinar el impacto de los laboratorios virtuales en la docencia universitaria [Candelas y otros, 2003b]. Una de las principales conclusiones de dichos estudios ha sido que el estudiante valora muy positivamente la docencia virtual, además de que los laboratorios virtuales le ayudan a comprender mejor los conceptos. Sin

embargo, consideran los laboratorios virtuales como un apoyo a la enseñanza, no como una alternativa, ya que según las investigaciones, el alumno prefiere asistir al laboratorio real, compartir sus problemas con los demás y tener el apoyo de un profesor.

Durante los últimos años, se han desarrollado muchos entornos web de aprendizaje donde se ha incluido la colaboración y la comunicación. Actualmente, podemos distinguir dos tipos de entornos colaborativos: los *on-line* o *síncronos* y los *off-line* o *asíncronos*. Con respecto a los síncronos, son los entornos que más se aproximan a la enseñanza tradicional, ya que permiten a los estudiantes compartir experiencias en tiempo real durante la clase virtual. En cuanto a los asíncronos, los integrantes de la clase no tienen porque estar conectados al mismo tiempo al sistema, aportando gran flexibilidad en el horario.

La mayoría de los entornos colaborativos *on-line* existentes en la actualidad, utilizan herramientas clásicas para la comunicación entre los integrantes de la clase tales como *chats*, *video streams*, pizarras compartidas [Kreutz y otros, 2000] e incluso paneles de dibujo interactivos [Snow y otros, 2005]. Sin embargo, dichas herramientas limitan el aprendizaje a la hora de explicar conceptos de carácter técnico, que podrían ser observados más fácilmente a través de las simulaciones de un laboratorio virtual. Aun así, el principal problema que existe actualmente es que hay muy pocos entornos colaborativos de carácter síncrono.

Entre uno de los entornos asíncronos más importantes, está *eMersion* [Gillet y otros, 2005]. Este sistema permite el acceso individual a laboratorios virtuales y remotos a través de la web. Además, dentro de este entorno se ha integrado *eJournal* [Fakas y otros, 2005], espacio electrónico donde los usuarios pueden compartir documentos y resultados de experimentos.

A día de hoy, existen entornos síncronos donde se ha intentado combinar laboratorios virtuales y herramientas clásicas de comunicación [Abler y Wells, 2005]. Sin embargo, han resultado ser entornos complejos y la necesidad de utilizar software costoso para implementar los laboratorios virtuales, como por ejemplo *Matlab Web Server*[1].

Considerando lo expuesto, se ha desarrollado un prototipo de un sistema de comunicación síncrono a través del protocolo TCP/IP entre simulaciones *Java* desarrolladas con *EJS*, con el fin de ser integradas dentro de entornos colaborativos *on-line* y realizar la enseñanza en tiempo real a través de In-

[1] *Matlab Web Site:* `http://www.mathworks.com/`

ternet. Dicho sistema de sincronización se ha incluido como una opción más del software *EJS* en una versión experimental, de forma que se aplica a las simulaciones compiladas con esta versión.

Este capítulo se ha divido en tres secciones. En el primero, se comentan las herramientas más destacadas en la actualidad para la creación de entornos web colaborativos. Posteriormente, se explica el sistema de comunicación desarrollado entre las simulaciones interactivas *EJS*. Y finalmente, se expondrán algunas conclusiones de los resultados obtenidos.

5.2. Sistemas de gestión de contenidos

El aprendizaje colaborativo vía web busca propiciar espacios electrónicos de trabajo en los que se dé el desarrollo de habilidades individuales y grupales a partir de la discusión entre los estudiantes y el profesor a la hora de explorar nuevos conceptos. Los sistemas de gestión de contenidos (CMS, *Content Management System*) o *Groupware* [Baumgartner, 2004] facilitan la creación de dichos espacios de trabajo para ayudar al aprendizaje colaborativo virtual.

Los CMS son herramientas poderosas que permiten compartir toda clase de conocimiento relativo a un grupo de personas y facilitan el movimiento y control de la información que se manipula constantemente. Los módulos que integran para ello son, principalmente:

- Calendario y planificación.
- Videoconferencia.
- Sistemas de reunión electrónica.
- Pizarra electrónica y conferencia de datos.
- Conversación (chat).
- Correo electrónico.
- Conferencia y grupos de noticias.

Con la ayuda de estas herramientas citadas y los laboratorios virtuales se podrán crear entornos colaborativos *on-line*, donde un grupo de alumnos

coordinados por un profesor, comparta experiencias y experimentos. Por esta razón, se pensó en crear el sistema de comunicación síncrono entre las simulaciones y de este modo, poder realizar la enseñanza en tiempo real a través de Internet.

Se ha dividido los CMS en dos tipos: los de **uso genérico**, softwares para la creación y personalización de entornos colaborativos, y los **específicos**, entornos colaborativos propios desarrollados en universidades.

5.2.1. CMS de uso genérico

Entre los CMS de uso genérico más destacados podremos nombrar a los siguientes:

- BSCW (*Basic Support for Cooperative Work*). BSCW[2] es una aplicación en la que se pueden almacenar documentos u otros objetos que deseen compartir los miembros de un grupo de trabajo. Esta herramienta proporciona todos los recursos básicos necesarios para experiencias de trabajo cooperativo que se apoyen en las posibilidades de Internet, intranet y extranet, de forma sincrónica y asíncrona. Permite, por tanto, crear espacios de trabajo compartido capacitados para almacenar, gestionar, editar y compartir documentos.

- MOODLE. Es un entorno off-line de muy reciente creación desarrollado por Martin Dougiamas. Moodle[3] es el acrónimo de *Modular Object-Oriented Dynamic Learning Environment* (Entorno de Aprendizaje Dinámico Orientado a Objetos y Modular). Se trata de un software para la creación de cursos y sitios Web basados en Internet. Posee muchas herramientas que gestionan desde la administración de los usuarios y cursos hasta módulos de foro, consulta, cuestionarios y tareas.

- Sinergia[4]. Entorno virtual gratuito desarrollado por Gregorio Jiménez para el aprendizaje colaborativo *on-line* donde el usuario accede como estudiante o profesor, configurando su cuenta como tal y con la posibilidad de utilizar diversas herramientas tales como: pizarras colaborativos, foros, cuentas, correo, etc,...

[2] *BSCW web page:* `http://bscw.fit.fraunhofer.de/`
[3] *Moodle web page:* `http://moodle.org/`
[4] *Sinergia web page:* `http://www.synergeia.info/`

5.2.2. Campus Virtual de la UA

Existen aplicaciones específicas de determinadas organizaciones, especialmente webs o intranets de universidades, que también incluyen funciones de trabajo o docencia en grupo. Cabe destacar el Campus Virtual de la Universidad de Alicante. Es un entorno web para la gestión académica y administrativa de los profesores, alumnos y personal de administración de la Universidad de Alicante. Mediante esta herramienta el profesor puede obtener listas de alumnos, visualizar sus fichas, preparar horarios, contestar a tutorías, proponer y moderar debates, realizar docencia presencial, gestionar su currículum, ver sus nóminas, etc,.... El alumno puede acceder a sus asignaturas, ver sus calificaciones, realizar consultas a profesores, asistir a una clase virtual, etc,.... Además esta aplicación web incluye capacidades de trabajo en grupo *off-line*, como muestra la Figura 5.1.

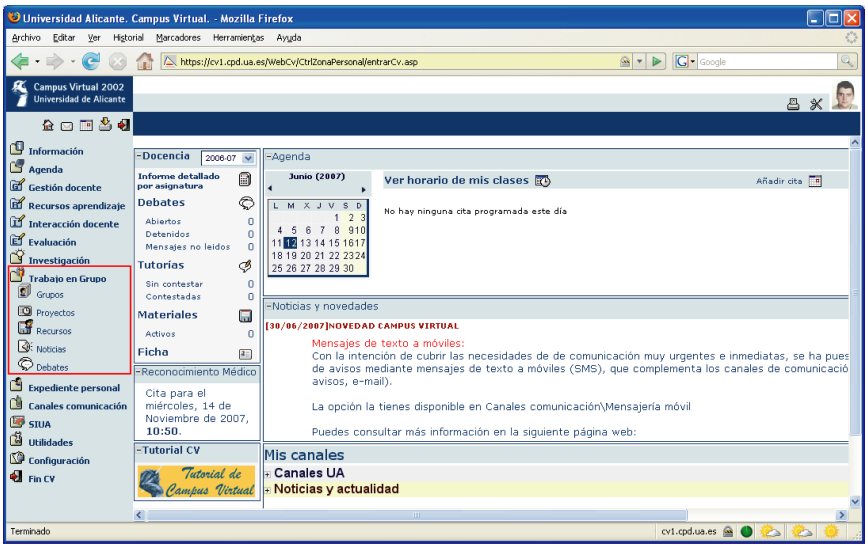

Figura 5.1: Campus Virtual de la Universidad de Alicante

5.3. Descripción del sistema de comunicación

5.3.1. Conceptos previos

Antes de comenzar con la descripción del sistema de comunicación, es necesario que el lector conozca algunos conceptos de las simulaciones inter-

activas de *EJS*. Aunque ya existe un manual del software como un apéndice del documento, para que el lector no tenga que leerlo completamente se redactan aquí unas breves líneas.

Las simulaciones *EJS* constan de dos partes bien diferenciadas: el *modelo* y la *vista*. El modelo está constituido por las variables de la simulación, su valor inicial y por las ecuaciones matemáticas que rigen la evolución del sistema. La vista es la interfaz de usuario donde se muestra la representación gráfica de los diferentes estados del sistema. Ambas partes se encuentran interconectadas. Cualquier cambio en el estado del modelo es visualizado en la vista, y si el usuario interacciona con ella (controles de la interfaz), es capaz de modificar el valor de una variable del modelo.

Durante la ejecución de la simulación, *EJS* crea las variables, las inicializa y ejecuta las ecuaciones del modelo cada cierto diferencial de tiempo para alcanzar un nuevo estado. Esta última fase es conocida como paso de simulación o *step*. Posteriormente, *EJS* actualiza la vista para visualizar el nuevo estado alcanzado y vuelve a ejecutar las ecuaciones para continuar la evolución del sistema.

Unas de las propiedades más interesantes que posee *EJS* es la interactividad de sus simulaciones [Dormido y otros, 2005]. Cada vez que el usuario pulsa un control o interacciona con algún objeto de la vista, se producen eventos que la simulación se ocupa de captar y gestionar. El usuario es el encargado de darle funcionalidad a estos eventos. Para esta tarea, *EJS* proporciona una serie de métodos predefinidos entre los que hay que destacar los siguientes:

- *play()*. Ejecuta la simulación. Es decir, da *steps* cada diferencial de tiempo.
- *pause()*. Detiene la evolución de la simulación.
- *reset()*. Detiene la simulación y la lleva a su estado inicial.
- *update()*. Actualiza las variables del modelo.

Existen muchos más métodos predefinidos, sin embargo los nombrados son los más importantes para la gestión de la simulación. Si el lector desea leer más información, puede ver el manual contenido en la memoria (Apéndice A) o dirigirse a [Esquembre, 2004].

5.3.2. Comunicación entre las simulaciones

Este punto describe cuáles son los *roles* de cada uno de los componentes de la clase virtual: profesor y alumno. También se comenta su funcionalidad y las posibilidades de cada uno. Finalmente, se explica el porqué del protocolo de comunicación utilizado y la arquitectura del sistema.

Componentes y funcionalidad del sistema

Como se ha comentado anteriormente, la clase virtual se compone de la simulación del profesor (*maestro*) y de las simulaciones de los alumnos (*esclavos*).

El maestro es el encargado de gestionar la sesión y la evolución de la simulación. Con respecto a la sesión, el maestro posee una lista de los alumnos conectados (Figura 5.2), a los que en cualquier momento puede desconectar. Referente a la simulación, es el único que puede interactuar sobre ella en un primer momento y comunicar a los alumnos conectados el estado de la misma.

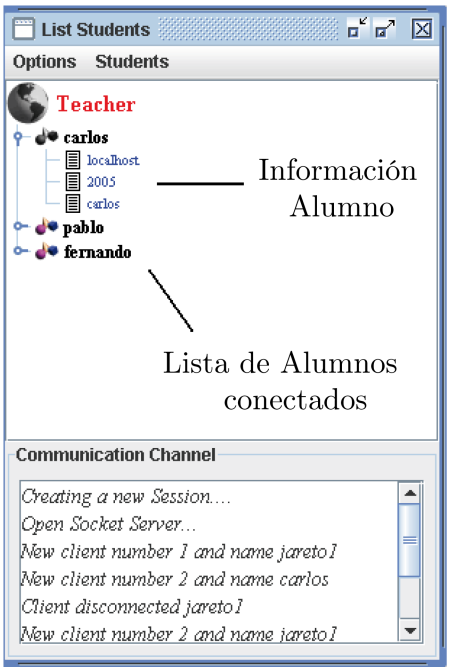

Figura 5.2: Lista de alumnos conectados

Para el comienzo de la sesión, el maestro debe activar el modo colaborativo (*estado de escucha*). A partir de este instante, los alumnos que deseen podrán conectarse al maestro, con la consecuente deshabilitación de todos los controles de su simulación para no poder interactuar con ella (Figura 5.3). Todos los alumnos conectados en la misma sesión, visualizarán en su simulación lo que el profesor realice sobre la suya.

Otra funcionalidad importante es la asignación de la *tiza* (ver punto 5.3.3). Con ella el maestro da a la simulación de un alumno el permiso de llevar el control sobre todas las demás. Tan sólo son permisos para la simulación, ya que la gestión de la sesión siempre queda en manos del maestro.

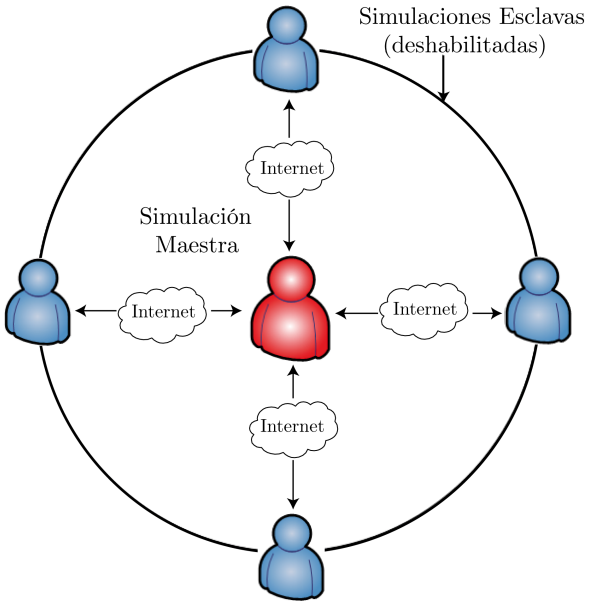

Figura 5.3: Componentes del sistema

Protocolo de comunicación

El sistema de comunicación (Figura 5.4) entre las simulaciones está basado en *TCP sockets* (Figura 5.4). Durante el diseño del sistema, se planteó la posibilidad de utilizar protocolos de más alto nivel, como por ejemplo HTTP. Esta opción permitía evitar posibles problemas con los *firewalls*. Sin embargo, se desechó fundamentalmente por dos razones:

1. Por la necesidad de tener instalado en el maestro un software para dar servicios como servidor de Internet (*IIS*, *Apache*, etc,..).

2. Por el procedimiento de petición-respuesta del protocolo en sí, que complicaba la sincronización entre las simulaciones.

Mediante el protocolo TCP/IP estos problemas son evitados, y si además realizamos la comunicación por el puerto 80, también conseguimos evitar los posibles problemas con los *firewalls*, que era la principal ventaja de HTTP.

El principal problema en el uso de este protocolo para la comunicación entre las simulaciones, es en el caso en que los alumnos se encuentren en redes con conexiones *WAN* deficientes. Esta dificultad aumenta el retardo en la conexión y consecuentemente ralentiza la ejecución de la simulación. En cualquier caso, siempre se mantienen sincronizadas.

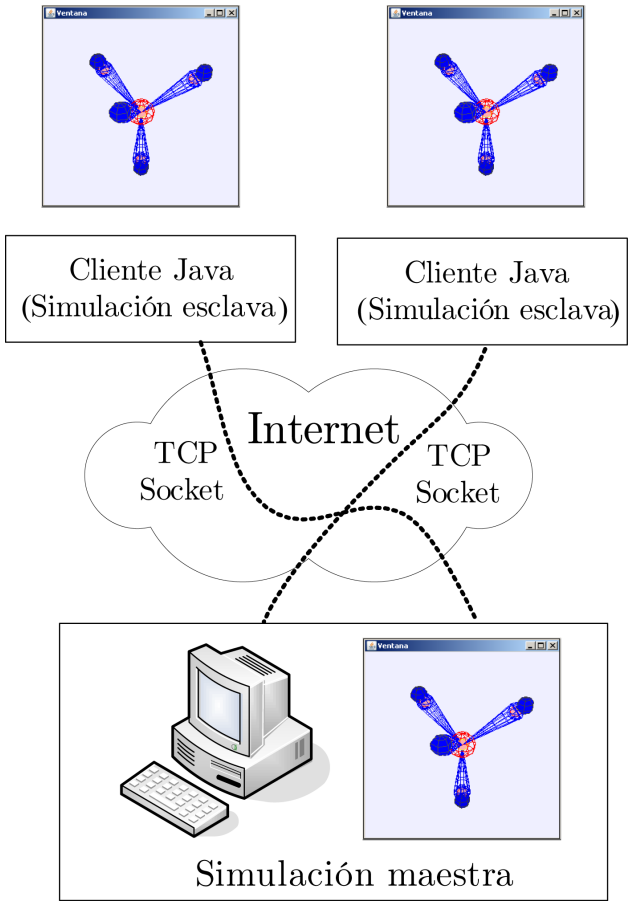

Figura 5.4: Sistema de comunicación

Arquitectura del sistema

Dado que la funcionalidad de la simulación del profesor y la del alumno son distintas, es necesario diferenciar sus arquitecturas.

Para iniciar el sistema de comunicación, el maestro debe activar el modo colaborativo. A partir de este momento un *SocketServer Java* abre un puerto determinado y comienza la escucha de peticiones de los alumnos (Figura 5.5). Este proceso de escucha es el responsable de la conexión y desconexión de las simulaciones de los alumnos.

Por cada petición aceptada, se crea un elemento más en un vector de conexiones (*sockets esclavos*), además de actualizar la lista de alumnos. Dicho elemento, contiene información referente a la procedencia del alumno (IP y puerto) conectado a la clase virtual. El objeto repetidor es el encargado de enviar los datos del estado actual de la simulación a todos los elementos de dicho vector. Cada cambio o evento en la simulación maestra es comunicado al repetidor, que se encarga de ponerlo en la red y hacerlo llegar a los clientes IP.

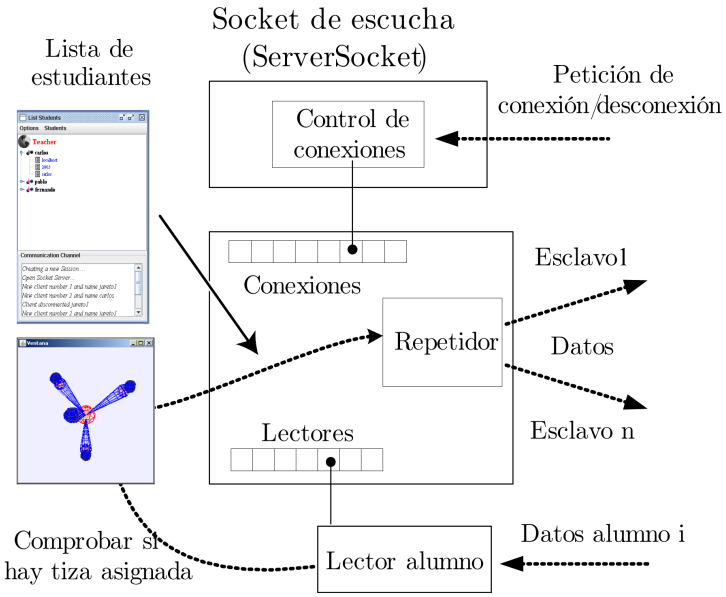

Figura 5.5: Arquitectura del Maestro

Además, para poder conocer el estado de cada una de las simulaciones de los alumnos, se establece un lector por cada conexión aceptada. Dicho objeto se encarga de comunicar al maestro si el esclavo ha realizado correctamente el

paso de simulación que le envió el repetidor. Una vez que se han recibido todas las confirmaciones de los esclavos, el maestro puede continuar con la evolución del sistema. Así, se consigue la sincronización deseada en las simulaciones.

En el caso de existir un esclavo con la tiza asignada, el maestro actúa como si fuera un alumno, además de comunicarle al resto las órdenes que recibe de dicha simulación.

La arquitectura del esclavo es mucho más sencilla que la del maestro (Figura 5.6). Una vez establecido el canal de comunicación (*socket*), sólo existe un intercambio de datos profesor-alumno. Un proceso receptor se ocupa de recibir las órdenes del maestro para controlar y actualizar la simulación. Y un remitente de datos, comunica al lector correspondiente del maestro (ver Figura 5.5), el estado de la simulación.

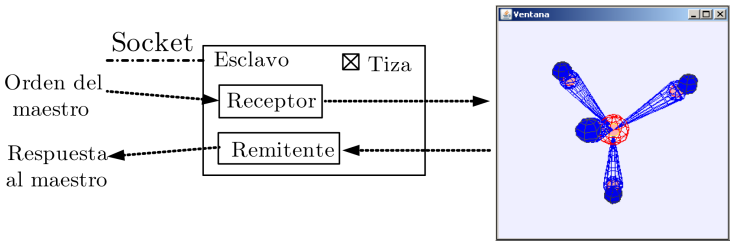

Figura 5.6: Arquitectura del Esclavo

Si el esclavo tiene la tiza asignada, sus controles se habilitan para que el alumno pueda interactuar sobre ella y comportarse como si fuera el maestro de la clase virtual.

5.3.3. Sincronización *on-line*

El sistema de comunicación y sincronización de las simulaciones se ha incluido como una opción más del software *EJS* en una versión experimental, de forma que se aplica a las simulaciones compiladas con esta versión. Desde una simulación *EJS* que se encuentre en un ordenador con conexión a Internet, es posible ponerla a la escucha de peticiones de alumnos o conectarse a una simulación maestra para recibir una clase virtual.

Dado que se está trabajando con simulaciones idénticas, el modelo del sistema es el mismo para todos los componentes de la sesión, facilitando la tarea de actualización de variables y refresco de la vista.

Por el *socket* de comunicación establecido entre el profesor y el alumno, se envía objetos Java que normalmente contienen *Strings*. Aquí, es necesario distinguir entre dos tipos de mensajes:

1. Los correspondientes al paso de simulación.
2. Los correspondientes a eventos como: *pause()*, *reset()*, *update()*, etc,...

Paso de simulación

Tal y como se comentó anteriormente, el paso de simulación o *step* es la resolución del modelo del sistema en un diferencial de tiempo. Por tanto, para sincronizar las simulaciones, es necesario que todas se encuentren en el mismo *step*. Para ello, cuando el maestro comienza la ejecución de la simulación y da el primer paso de simulación, envía a través del repetidor la orden "*step*" para el resto de simulaciones. En este momento, para la ejecución y se pone en espera hasta que todas las simulaciones de los alumnos hayan realizado el *step*. De esta manera se consigue la sincronización, propiedad necesaria para el correcto desarrollo de una clase de aprendizaje colaborativo *on-line*. Además, en el maestro se define un tiempo de espera de las confirmaciones de *step* de sus clientes, y si un cliente no contesta se le ignora. De este modo se evita que un cliente pare la simulación.

Eventos

En el caso de los eventos, el sistema de comunicación es más sencillo. Gracias a que *EJS* capta y gestiona los eventos, tan sólo tenemos que decirle que envíe el mensaje cuando ese evento tenga lugar. Aquí, no es necesario esperar a que los clientes verifiquen que han ejecutado la orden.

Asignación de la tiza

Como se ha comentando en este artículo, el sistema de comunicación puede dar el permiso a un alumno de controlar el estado de la simulación.

El maestro sólo puede asignar la tiza a uno de los alumnos conectados. En el momento de la asignación, se desconectan los controles de la simulación