Robótica Asistida por Teleoperación y Realidad Virtual

Tesis de Maestría realizada por el Licenciado en Informática

Ulises Zaldívar Colado

Para obtener el grado de

Maestro en Ciencias en Ingeniería Eléctrica

Que otorga la Sección de Computación del
Departamento de Ingeniería Eléctrica del Cinvestav

Tesis dirigida por el

Dr. Juan Manuel Ibarra Zannatha
del Departamento de Control Automático del Cinvestav

y por el

Dr. Jorge Buenabad Chávez
de la Sección de Computación
del Departamento de Ingeniería Eléctrica del Cinvestav

Noviembre de 2003

Resumen

Desde hace ya varias décadas la Robótica ha tenido un fuerte impacto en la sociedad y no sólo en el sector manufacturero de la economía, ya que la automatización juega un papel muy importante en muchas disciplinas de la vida diaria. La Robótica logra sustituir el trabajo humano en labores que son peligrosas, insalubres y aún imposibles de realizar por trabajadores humanos. En efecto, los robots suelen soportar más peso que un humano, pueden también ejercer más fuerza que éste, además de que las máquinas realizan de mejor manera tareas repetitivas que a los humanos normalmente les resultan tediosas, molestas y por ende peligrosas. De lo anterior se desprende la importancia que tienen la investigación y la docencia en esta área. Desgraciadamente cada vez es más difícil contar con la infraestructura material adecuada para estas actividades en áreas que, como ésta, tienen una componente tecnológica fuerte, debido a los altos costos de inversión, operación y mantenimiento que demanda.

Esta tesis presenta el diseño e implementación de un Laboratorio Virtual de Robótica que permite a una gran cantidad de gente acceder a un laboratorio de robótica real equipado con una infraestructura experimental especializada, mediante una herramienta tan simple, común y barata como es Internet. Este laboratorio apoyará la formación de recursos humanos y la realización de proyectos de desarrollo distribuidos en el espacio.

Las tres principales funciones de nuestro laboratorio virtual son:

- La generación de **Mundos Virtuales Interactivos**, lo cual permite tener una versión virtual simulada del robot existente en el laboratorio de robótica real. El uso de realidad virtual permite tener un clon virtual del robot real para manipular, programar y simular tareas industriales reales.
- La **Teleoperación**, que permite trabajar con los dispositivos del laboratorio remotamente desde cualquier sitio conectado a Internet y ejecutar en ellos tareas concebidas, programadas y simuladas remotamente en el mundo virtual equivalente.
- La **Programación Automática**, permite generar aplicaciones industriales de manipulación y lograr su correcta ejecución en el robot real.

Los lenguajes utilizados para la creación de los mundos virtuales interactivos fueron VRML (*Virtual Reality Modeling Language*) y Java; para la Teleoperación principalmente se utilizaron Java y Visual Basic; y la programación automática se logró combinando VRML y Java.

Además, se hicieron dos aplicaciones para probar la función correcta del sistema. Un sistema de autoprogramación que permite que el robot virtual pueda organizar objetos en el espacio de trabajo con base en criterios previamente establecidos. Y otro sistema que realiza la teleoperación de un robot móvil hecho en la Universidad de Zaragoza en España.

Nuestro sistema y estas aplicaciones han sido la base para nuevos proyectos desarrollados en México y España.

Abstract

Since a long time, Robotics has impacted in the society and not only in the economy's manufacturing sector, since the automation plays a very important role in many disciplines of the daily life. Robotics replaces the human work on dangerous tasks, and some times, not possible to do by humans. The robots can support a lot of height than humans, and be stronger than, and do the human work of an easy way. That's why the teaching and research had a very importance on this area. Unfortunately, each time, is too difficult to have this kind of infrastructure, with good technological component, because is too expensive.

This dissertation presents a design and implementation of the Virtual Robotics Laboratory that permits to many people to access at the real Robotics Laboratory equipped with an experimental specializing infrastructure through a cheaper tool as Internet. This laboratory will support the formation of human resources and the development project achievement

Three main functions of this virtual laboratory are:

- Generation of **Interactive Virtual Worlds**, which allows having a simulated virtual version of the existing robot in the laboratory. The use of virtual reality to have a virtual clone of the real robot to manipulate, to programming and to simulate industrial real tasks.
- The **Teleoperation**, which allows to work remotely with the laboratory devices from any place connected to Internet, and to execute in them tasks conceived, programmed and simulated remotely in the virtual equivalent world.
- The **Automatic Programming**, allows to generate manipulation industrial applications and to reach execution in the real robot.

The programming languages used to the virtual worlds creation were VRML (Virtual Reality Modeling Language) and Java; for the Teleoperación principally Java and Visual Basic were in use; and the automatic programming was achieved combining VRML and Java.

Besides, several applications were done to prove the well function of the system. An autoprogramming system, that allows that the virtual robot could to organize objects in the working space. Another system fulfils the teleoperación of a mobile robot done at University of Zaragoza, Spain.

Our system and some of these applications have been the base for new projects to be developed in Mexico and Spain.

Contenido

Resumen iii
Abstract v
Contenido vii
Lista de Figuras xi
Lista de de Programas xiii
Lista de de Tablas xv
Agradecimientos xvii
Dedicatorias xix

Capítulo 1
Introducción
1.1. Motivación 1
1.2. Solución propuesta 2
1.3. Trabajos relacionados 4
1.4. Organización de la tesis 8

Capítulo 2
Descripción de la Infraestructura Utilizada
2.1. Robot Industrial UNIMATE S-103 9
2.1.1. Operación del Robot 11
2.2. Estructura de Servidores y Red 14
2.3. Software 17
2.3.1. VRML 17
2.3.2. Interfaz de Comunicación Externa 20
2.3.3. Java 22
2.3.4. Otras herramientas utilizadas 23

Capítulo 3
Diseño del Laboratorio Virtual de Robótica

3.1. Organización del Laboratorio Virtual de Robótica 25
3.2. Mundo Virtual 27
3.2.1. Modelado de la Cinemática de Robots 28
3.2.2. Cinemática Directa del Robot UNIMATE S-103 32
3.2.3. Cinemática Inversa del Robot UNIMATE S-103 34
3.2.1. Interacción con el Robot Virtual 36
3.2.2. Actualización del Espacio Virtual 36
3.2.3. Manipulación de objetos por el Robot Virtual 37
3.3. Interfaz de Usuario 38
3.3.1. Manipulación del Robot Virtual en Coordenadas Articulares 38
3.3.2. Manipulación del Robot Virtual en Coordenadas Operacionales 39
3.3.3. Actualización del Mundo Virtual 40
3.3.4. Enseñanza y Programación del Robot 41
3.4. Autoprogramación del Robot 43
3.5. Teleoperación vía Internet 43
3.5.1. *Teach Pendant* Remoto 44
3.5.2. *Teach Pendant* Virtual 45
3.5.3. Monitoreo Visual Remoto 46

Capítulo 4
Diseño e Implementación del Robot Virtual

4.1. Introducción 47
4.2. Robot Virtual 49
4.2.1. Eslabones Móviles 52
4.3. Interfaz de Comunicación Externa 57
4.4. Interactuando con el Mundo Virtual 58
4.4.1. Rotación de las Articulaciones 59
4.4.2. Rotación con Interpolación 61
4.4.3. Manejo de la Pinza 64
4.4.4. Creación y Supresión de Objetos en el Mundo Virtual 76
4.5. Interfaces de salida de los MV 78
4.5.1. Rotación 78
4.5.2. Sensores 80
4.5.3. Tocando objetos 81

Capítulo 5
Diseño e Implementación de la Interfaz de Teleoperación vía Internet

5.1. Teach Pendant Hardware 84
5.2. Teach Pendant Virtual 84
5.2.1. Conectándose con el Robot 85
5.2.2. Recibiendo Datos del Robot 86

5.2.3. Enviando Datos al Robot 88
5.2.4. Recibiendo Datos Remotos 89
5.3. Teach Pendant Remoto 92
5.3.1. Conectándose con el Teach Pendant Virtual 94
5.3.2. Enviando datos al Teach Pendant Virtual 95
5.4. Teleoperación desde el Mundo Virtual 97

Capítulo 6
Aplicaciones
6.1. Autoprogramación 101
6.2. Clasificación Automática 102
6.3. Robot Móvil 104
6.3.1. Arquitectura de la aplicación 105
6.3.1.1. Hardware y Software disponibles 105
6.3.1.2. Flujo de comunicación 106
6.3.2. Modelado del ambiente virtual en 3D 107
6.3.3. Resultados experimentales 109

Capítulo 7
Conclusiones y Trabajos Futuros
7.1. Conclusiones 111
7.1.1. De la Cinemática del Robot y del Sistema de Visión 112
7.1.2. Del Software Utilizado y Desarrollado 113
7.3. Trabajos en Curso y Futuros 115

Bibliografía y Referencias 117

Lista de Figuras

Capítulo 1
Introducción

Figura 1.1. Esquema del software que constituye el Laboratorio Virtual de Robótica.

Capítulo 2
Descripción de la Infraestructura Utilizada

Figura 2.1. Aspecto del robot industrial UNIMATE S-103
Figura 2.2. Imagen de la botonera externa
Figura 2.3 Esquema del Laboratorio Virtual de Robótica.
Figura 2.4 Diagrama de interacción cliente-servidor.
Figura 2.5 Ejemplo correspondiente al Programa 2.1.

Capítulo 3
Diseño del Laboratorio Virtual de Robótica

Figura 3.1. Esquema del software que constituye el Laboratorio Virtual de Robótica.
Figura 3.2. Modelo virtual del UNIMATE S-103.
Figura 3.3. Diagrama de Denavit-Hartemberg para obtención de la Cinemática.
Figura 3.4. Asignación de los marcos de coordenadas para el robot UNIMATE S-103.
Figura 3.5. Diagrama cinemático del robot UNIMATE S-103.
Figura 3.6. Objetos en el espacio virtual
Figura 3.7. Interfaz de aplicación del sistema.
Figura 3.8. Interfaz de manipulación con cinemática inversa.
Figura 3.9. a) Menú de objetos y b) Menú de programas disponibles en el mundo virtual.
Figura 3.10. Esquema del software del Laboratorio Virtual de Robótica usado en tareas de teleoperación.
Figura 3.11. Página WEB de Teleoperación con el *Teach Pendant* Remoto

Capítulo 4
Diseño e Implementación del Robot Virtual

Programa 4.1. Anidamiento de los nodos correspondientes a las articulaciones del robot.

Figura 4.1. Plataforma de trabajo modelada con VRML.
Figura 4.2. Eslabón fijo del UNIMATE modelado con VRML: a) en piezas, b) ensamblado.
Figura 4.3. Brazo del robot virtual representado sobre el eslabón fijo.
Figura 4.4. Plataforma, eslabón fijo, brazo y antebrazo del robot virtual.
Figura 4.5. Mano del robot virtual y pinza bidigital.
Figura 4.6. Robot virtual.
Figura 4.7. Acceso al Mundo Virtual desde la Interfaz de Usuario a través de la ECI
Figura 4.8. Esquema del flujo de información para realizar rotación con interpolación.
Figura 4.9. Esquema del flujo de información para subir y bajar la pinza.
Figura 4.10. Esquema del flujo de información para abrir y cerrar la pinza
Figura 4.11. Esquema del flujo de información en la ECI para crear y eliminar objetos.
Figura 4.12. Esquema del flujo de los eventos de salida de VRML a Java a través de la ECI.

Capítulo 5
Diseño e Implementación de la Interfaz de Teleoperación vía Internet

Figura 5.1. a) *Teach Pendant Hardware*. b) *Teach Pendant* Virtual.
Figura 5.2. *Teach Pendant* Remoto.
Figura 5.3. Mundo Virtual con la interfaz de teleoperación y de programación.
Figura 5.4. a) Menú de objetos y b) Menú de programas disponibles en el MUNDO VIRTUAL.

Capítulo 6
Aplicaciones

Figura 6. 1. a) Estado inicial, b) Estado intermedio
Figura 6.2. a) Estado final, b) Acercamiento del estado final
Figura 6.3. Esquema de comunicación cliente/servidor de 3 capas.
Figura 6.4. Interfaz de teleoperación del robot móvil.
Figura 6.5. Mundo Virtual del robot móvil.
Figura 6.6. Primera detección de objetos con el láser.
Figura 6.7. Obtención de todo el entorno real. Giro completo.
Figura 6.8. Cruzando la puerta.
Figura 6.9. Del otro lado de la puerta.

Lista de Programas

Capítulo 2
Descripción de la Infraestructura Utilizada

Programa 2.1. Un programa sencillo en VRML.

Capítulo 4
Mundos Virtuales Interactivos con VRML

Programa 4.1. Anidamiento de los nodos correspondientes a las articulaciones del robot.
Programa 4.2. Definición de la Plataforma de Trabajo virtual.
Programa 4.3. Definición del eslabón fijo del robot virtual.
Programa 4.4. Definición del brazo (primer eslabón móvil) del robot virtual.
Programa 4.5. Definición del segundo eslabón móvil del robot virtual.
Programa 4.6. Definición del tercer eslabón móvil del robot virtual.
Programa 4.7. Definición de la pinza bidigital del robot manipulador virtual.
Programa 4.8. Acceso a la función de rotación desde Java.
Programa 4.9. Declaración y cuerpo de la función `RotJoint` en un Mundo Virtual VRML.
Programa 4.10. Especificación de la ruta del flujo de datos para realizar la rotación.
Programa 4.11. Interpolador de rotación de las articulaciones y su sensor de tiempo.
Programa 4.12. Acceso a nodos y eventos desde Java para realizar rotación con interpolación.
Programa 4.13. Función que permite hacer rotación con interpolación en el Mundo Virtual.
Programa 4.14. Flujo de información para realizar la rotación con interpolación.
Programa 4.15. Función en VRML que permite subir y bajar la pinza.
Programa 4.16. Sensores de tacto, tiempo e interpoladores utilizados para subir y bajar la pinza.
Programa 4.17. Flujo de datos para realizar la animación en la pinza.
Programa 4.18. Acceso al control de subir y bajar la pinza desde Java.
Programa 4.19. Función en VRML que permite abrir y cerrar la pinza.
Programa 4.20. Sensores de tacto, tiempo e interpoladores que permiten abrir y cerrar la pinza.
Programa 4.21. Flujo de información que se encarga de abrir y cerrar la pinza.

Programa 4.22. Acceso al control de la pinza para abrir y cerrar desde Java.
Programa 4.23. Nodo (VRML) vacío para crear objetos dentro de él.
Programa 4.24. Código en Java para crear y/o remover objetos en el Mundo Virtual.
Programa 4.25. Código que permite crear parte de un programa en VRML desde Java.
Programa 4.26. Acceso a los nodos que envían información a través de los eventos de salida.
Programa 4.27. Método *callback* que permite recibir eventos de salida desde Java.
Programa 4.28. Acceso desde Java a los nodos que activan la pinza.
Programa 4.30. Declaración VRML de los sensores de tacto para ser accedidos desde Java.
Programa 4.31. Creación de un objeto desde Java en VRML.

Capítulo 5
Teleoperación vía Internet

Programa 5.1. Código que permite conectarse con el robot a través del puerto serie.
Programa 5.2. Sincronización de lectura de información del puerto serie.
Programa 5.3. Recepción y organización de los datos provenientes del puerto serie.
Programa 5.4. Envío de datos al puerto serie.
Programa 5.5. Esperando y aceptando llamadas por *sockets*.
Programa 5.6. Llegada de datos al *socket*.
Programa 5.7. Evaluación de una cadena para enviar datos al robot.
Programa 5.8. Evaluación de las cadenas de un programa y coordenadas para enviar al robot.
Programa 5.9. Conexión al *socket* de VB desde Java.
Programa 5.10. Envío de datos al *socket*.
Programa 5.11. Método de acción de eventos del *mouse* para enviar datos al *socket*.

Lista de Tablas

Capítulo 2
Descripción de la Infraestructura Utilizada

Tabla 2.1. Lista de Comandos disponibles en el modo PGM del Lenguaje C/ROS

Capítulo 3
Diseño del Laboratorio Virtual de Robótica

Tabla 3.1. Parámetros D-H del robot UNIMATE S-103

Capítulo 5
Teleoperación vía Internet

Tabla 5.1. Mecanismo de teleoperación vía Internet basado en el *Teach Pendant* Remoto
Tabla 5.2. Mecanismo de programación y teleoperación vía Internet basado en el MUNDO VIRTUAL

Agradecimientos

A CONACYT, por el apoyo económico brindado para la realización de estos estudios.

Al Cinvestav, principalmente a la sección de Computación del departamento de Ingeniería Eléctrica, por haberme aceptado como alumno de esa sección.

A los profesores de la sección:

Dr. Jorge Buenabad, quien se porto como un amigo más que como profesor durante el tiempo que realice mis estudios.

Al Dr. Arturo Díaz, por haberme brindado las bases principales para la realización de esta tesis.

A los Doctores Adriano de Luca y Jorge Chapa, por las asesorías y la motivación que me brindaron.

A la secretaria de la sección, Sofía Reza, por el apoyo moral que siempre nos brindó, y por atendernos siempre con una sonrisa sin importar la cantidad de trabajo que tuviera.

Un agradecimiento muy especial al Dr. Juan Manuel Ibarra Zannatha, director de mi proyecto de tesis, quien me ha compartido sus conocimientos y fue una gran motivación para realizar este proyecto de tesis. Gracias por brindarme su amistad desde hace tantos años.

A mis compañeros de la sección:

José Manuel (Josela) y Armando (AFI), por haberme brindado su amistad, a pesar de los corajes que llegamos a pasar.

Mizael y Marco, por haber sido grandes amigos y haberme aguantado tantas bromas.

Gabriel (Gabo) y Carlos (Carlota), buenos amigos, por haberme asesorado con una facilidad y paciencia envidiable.

David Araujo (Compatriota), gracias por ayudarme a despejar dudas aun cuando siempre estabas ocupado.

Y a los demás compañeros que, a pesar de ser poco el tiempo que convivimos, logramos crear una buena amistad: Giner, Libertad, Ascelli, Marelino.

Dedicatorias

A mi esposa, Carolina Martínez, por la gran paciencia, apoyo y amor que me ha brindado siempre. Por no haber hecho más difícil todo el tiempo que nos hemos separado para poder realizar estos sueños. Por las palabras de comprensión que siempre sabe decir. Gracias por todo.

A mis hijos Leslie y Gael, discúlpenme por todo el tiempo que han tenido que pasar sin su papá. Los quiero mucho.

A mi mamá, Guadalupe, que siempre esta deseando que sus hijos se superen. Por toda la ayuda que nos ha brindado. Gracias, amá.

A mi papá, Julián Zaldívar, gracias por haberme apoyado económica y moralmente cuando estaba desesperado, gracias por sus palabras de apoyo.

A mis hermanos:

Aníbal y Xiomara, gracias por todo el apoyo moral y económico en los momentos difíciles. Xiomara, gracias también por el apoyo espiritual.

A mis abuelos, papá Juven y mamá Nena, gracias por el afecto brindado, gracias por recibirme feliz cada vez que regresaba.

A mi tía Rossy, al Mendoza y a mi ahijado Erick.

A mis primas, Poly y Chiquita, también para ustedes.

Capítulo 1
Introducción

1.1 Motivación

En el Sistema de Educación Superior de nuestro país es cada vez más difícil que las instituciones cuenten con la infraestructura material y los recursos humanos suficientes para soportar de manera eficiente la formación de nuevos ingenieros e investigadores [Ibarra 00.2]. Esto no significa que no haya algunas instituciones importantes que posean tanto el equipamiento experimental en laboratorios didácticos o de investigación, así como los profesionales expertos en su utilización en el proceso de enseñanza, aprendizaje y en proyectos de investigación. Sin embargo, esto es privilegio de pocas instituciones educativas. Estamos convencidos de que es posible aprovechar de modo más eficiente el equipamiento disponible en los pocos laboratorios que cuentan con la infraestructura adecuada, permitiendo que dicha infraestructura sea puesta a disposición de personas con un cierto nivel mínimo de formación, pertenecientes a otras instituciones. Una manera de lograrlo es haciendo que esta infraestructura sea accesible mediante Internet, aprovechando que ésta última está disponible en la gran mayoría de instituciones de educación superior y aún por muchos particulares [Ibarra 97, Ibarra 00.1, Ibarra 01, Gordillo 98, Sucar 01, Zaldívar 01]. Es en este contexto que se sitúa el proyecto ROVIR, consistente en el desarrollo de un Laboratorio Virtual de Robótica accesible por Internet.

Entre los problemas más interesantes que se resuelven con un Laboratorio Virtual de Robótica está la telepresencia visual y su utilización en la actualización de los espacios virtuales remotos [Chellali 97], la generación de programas para puestos de trabajo robotizados y su

simulación usando modelos gráficos tridimensionales. También destaca la teleoperación vía Internet [Gordillo 98, Chellali 01, Kheddar 97.1] y la utilización de la plataforma en la solución de problemas reales de manipulación robotizada inteligente asistida por visión artificial.

1.2 Solución propuesta

Esta tesis presenta un sistema de Mundos Virtuales Interactivos, el cual genera las propiedades mecánicas y computacionales de los modelos geométricos y cinemáticos tridimensionales del robot UNIMATE S-103 de tres grados de libertad (gdl), y otros equipos de su puesto de trabajo real. El sistema hace un despliegue visual para que el usuario interaccione libremente con los modelos. Estos modelos se crean con VRML 2.0 [VRML 96, VRML 97], el cual es un lenguaje de modelado de realidad virtual de propósito general. Una particularidad esencial del mundo virtual así creado es su realismo y su interactividad: los usuarios tienen acceso a un laboratorio virtual que tiene el mismo funcionamiento del puesto de trabajo y los mismos movimientos que tienen los equipos reales correspondientes.

El uso de VRML facilitó el desarrollo de la simulación del espacio de trabajo del laboratorio real. Además, VRML ofrece la posibilidad de controlar los mundos virtuales a través de objetos programados en algún lenguaje orientado a objetos. Así, con la potencia de un lenguaje como Java se implementó el control y la manipulación de los mundos virtuales.

Esta tesis también resuelve el problema de la teleoperación vía Internet de un robot, así como la generación de programas para el puesto de trabajo del mismo. Durante la manipulación de los mundos virtuales interactivos, se generan programas para el puesto de trabajo, los cuales son entendidos por su modelo virtual como por el robot.

Mientras se realiza la manipulación del mundo virtual en VRML, la aplicación en Java se encarga de generar un registro de los movimientos y acciones que suceden en el mundo virtual. De esta manera las acciones se van registrando como comandos de ejecución del robot real, y así, pueden ser repetidos en el mundo virtual cuando se desee, y a su vez enviados al robot real para su ejecución en el laboratorio. Para lograr la ejecución en el laboratorio, los comandos de ejecución son enviados a través de *sockets* [Internet 1] a una aplicación en Visual Basic que se encuentra ejecutando en un servidor del laboratorio real. Este programa en Visual Basic se

comunica vía puerto serie con el robot real, y éste realiza los movimientos de la misma manera que el robot virtual [Herrera 00, Ibarra 00.1, Ibarra 00.2].

La Figura 1.1 muestra los módulos de software que conforman el Laboratorio Virtual de Robótica y su interacción. El módulo Interfaz de Usuario permite manipular el mundo virtual, además de comunicarse con el laboratorio real. El mundo virtual es el módulo donde se muestra el modelo del robot virtual, el cual respeta el comportamiento cinemática del robot real.

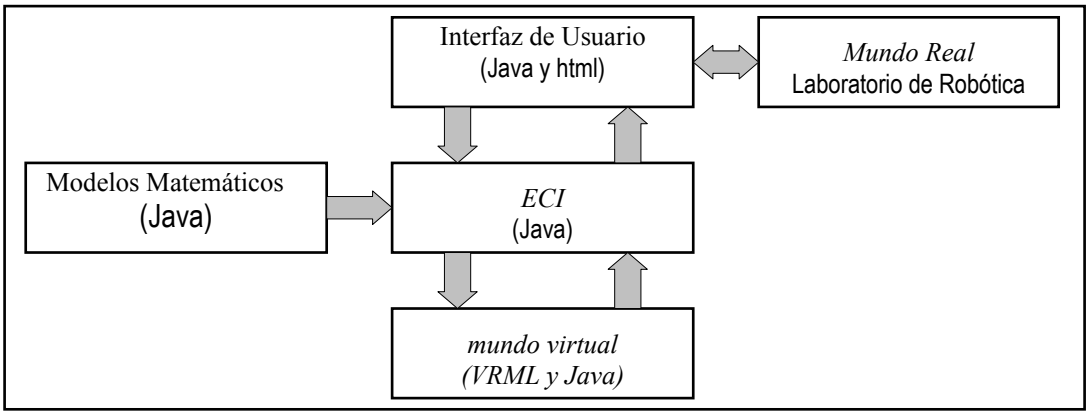

Figura 1.1. Esquema del software que constituye el Laboratorio Virtual de Robótica.

El módulo Modelos Matemáticos contiene los modelos geométricos y cinemáticos del robot real. Estos modelos permiten mover al robot (virtual y real). El módulo mundo real corresponde al Laboratorio de Robótica del Cinvestav. Este recibe de la interfaz de usuario un programa en C/ROS, previamente simulado en esa interfaz. Y el módulo ECI (External Communication Interface) se encarga comunicar a la Interfaz de Usuario con el mundo virtual.

También en esta tesis se desarrollan algunas aplicaciones reales de manipulación (asistidas por un sistema de visión desarrollado por David Araujo [Araujo 02]), como la selección automática de objetos en el puesto de trabajo, aplicando algunos criterios y restricciones y utilizando el algoritmo de selección conocido como selección por cubetas (*bucketsort*) [Bruno 98, Cormen 90, Heileman 98].

Finalmente en esta tesis, se bosqueja la exportación del sistema para incluir una plataforma basada en un robot móvil tomando en cuenta las restricciones cinemáticas que este pudiera tener, así como los modelos geométricos y cinemáticos correspondientes.

La aplicación desarrollada para el robot móvil LABMATE de la Universidad de Zaragoza cumplió satisfactoriamente los objetivos planteados. Fue posible realizar una teleoperación del robot sobre un mapa generado automáticamente en el mundo virtual correspondiente al entorno del robot real. El desarrollo de esta aplicación requirió de hacer simples modificaciones al código utilizado en el prototipo desarrollado para el robot manipulador UNIMATE S-103 para adaptarlo a las propiedades físicas y mecánicas del LABMATE. Esta aplicación se desarrolló en la Universidad de Zaragoza, España, durante una estancia de investigación de tres semanas financiada por el proyecto multinacional PARTI (Programación Asistida de Robots en Tareas Industriales). Dicho proyecto está patrocinado por el CYTED, Programa Iberoamericano de Ciencia y Tecnología para el Desarrollo. Aquí, el problema a resolver, además de la teleoperación, fue la generación automática y en tiempo real de un mundo virtual a partir de la información sensorial de profundidad generada por un sensor láser 3D. El buen resultado obtenido en el desarrollo de esta aplicación muestra la facilidad con que el sistema diseñado y construido en el marco de esta tesis puede adaptarse para resolver diferentes problemas de teleoperación y de programación a distancia.

1.3 Trabajos relacionados

Los casos pioneros de la Robótica por Internet lo constituyen los denominados *telerobots* o *webots* (robots en la WEB) como el de la University of Western, Australia [Taylor] desarrollado por el Dr. Ken Taylor. Desde 1994, fecha en que se puso en la red a disposición de todo mundo, ha tenido más de medio millón de accesos. Barney Daltons [Daltons], en el marco de su tesis doctoral hizo mejoras sustanciales a este prototipo de modo que varios usuarios a la vez pueden desarrollar tareas cooperativamente. Cuando uno de los usuarios tiene el control del robot, los demás pueden ayudarlo midiendo las posiciones de los objetos por manipular y guardando estos datos en la memoria colectiva. Estos usuarios mantienen la comunicación entre ellos mediante una ventana de comunicación (*chat*) [Taylor 97].

El Laboratorio Virtual de Robótica y Manufactura (LVRM) del ITESM-MTY [Gordillo 98], tiene la finalidad de hacer trabajos de maquinado y de ensamble de piezas con la infraestructura material de su laboratorio de robótica. Comandado desde una computadora remota conectada por Internet, utiliza un robot SEIKO de arquitectura cilíndrica con 4 gdl. El robot tiene una pinza de

dos posiciones y está conectado a una estación de trabajo SUN que permite controlar y programar las operaciones del robot localmente o vía Internet. El LVRM cuenta con un espacio virtual para la simulación de los programas hechos remotamente. Para definir los objetos virtuales se construyó un modelador de sólidos, que permite la definición de objetos compuestos y de mecanismos articulados mediante la especificación de restricciones cinemáticas. Al definir las variables articulares para un robot simulado, el modelador de sólidos genera simultáneamente los comandos correspondientes para el robot real; de forma que al aplicar desplazamientos sobre el modelo simulado, estos desplazamientos se aplican simultáneamente o posteriormente sobre el robot real. A diferencia de el trabajo presentado en esta tesis, este laboratorio no permite un monitoreo visual de la ejecuciónd el robot real, además de que siempre se debe de tener al robot en línea para lograr la realización de una práctica, por lo tanto, no es posible atender a más de un usuario a la vez.

En el ITESM-Morelos se diseñó e implementó la herramienta denominada *Laboratorio Virtual de Robótica Móvil* [Sucar 01] que permite la programación y visualización de un robot móvil. Los elementos que forman el sistema son un robot Nomad 200 con un procesador principal Pentium a 166 Mhz, 16 sonares Polaroid, 20 sensores táctiles y dos cámaras de vídeo. La Programación del robot se realiza mediante una interfaz en una pagina WEB, utilizando un lenguaje básico con 10 instrucciones que son codificadas y enviadas por Internet. Este lenguaje permite que el usuario programe al robot mediante una secuencia de instrucciones que le permiten navegar en un cierto ambiente interior (laboratorio), incluyendo desplazamientos y giros. Lamentablemente, en este sistema no es posible obtener información de los diferentes sensores que tiene el robot, por lo tanto las posibilidades de conocer los estados del robot están limitadas.

También se diseñó un módulo (programado en Java) que se encarga de recibir y escribir en un archivo el conjunto de instrucciones introducidas por el usuario remoto. Además, este módulo ejecuta un programa implementado en C, que se encarga de interpretar y ejecutar estas instrucciones dentro del robot móvil. Se tiene un módulo adicional de seguridad que evita que el robot tenga colisiones con las paredes y los objetos en su medio ambiente. Mediante imágenes de vídeo un usuario puede visualizar los resultados de un experimento en el *Laboratorio Virtual*. Debido al tamaño original de las imágenes y la limitante del ancho de banda que se necesita para

transmitir imágenes dentro de la red, se desarrolló un módulo para compresión de imágenes que utiliza el formato JPEG.

En el ITESM-CEM desarrollaron un simulador de una celda flexible de manufactura que consiste de diversos robots y máquinas interactuando para producir productos [Rudomín 97]. El simulador utiliza un mundo virtual que puede ser accedido simultáneamente por varios usuarios quienes pueden hacer modificaciones en línea. El acceso a este sistema es a través de un explorador de Internet, en donde se presenta un ambiente visual en VRML y una interfaz de control realizada en Java. La página que contiene los archivos en Java y VRML para interactuar con el simulador se encuentran en un servidor WEB. Desde esta página se puede poner a funcionar la celda de manufactura, la cual no solo se presenta como un ambiente visual, sino que el servidor cuenta con un programa en Java que funciona como servidor de la celda de manufactura, el cual controla los robots y las maquinas que los usuarios pueden ver en el ambiente virtual. De esta manera, cuando hay más de un usuario accediendo al servidor, todos los clientes pueden ver el comportamiento de la celda, así como hacer modificaciones en ella compartiendo los recursos entre todos estos clientes.

Joerg Vogel [Vogel] del DLR (*Deutsches Zentrum für Luft- und Raumfahrt)* o, en español, Centro Alemán de Investigación Aeroespacial, ha venido desarrollando múltiples proyectos relacionando la Robótica con el Internet mediante tecnologías HTML, VRML, Java, Java3d y CORBA. Martin Rohrmeier, en su tesis doctoral [Rohrmeier 97], desarrollada en el DLR bajo la dirección de Joerg Vogel, presenta un sistema de manipulación y teleoperación de un complejo robot industrial, modelado en VRML e interaccionado a través de *Javascript.* El sistema está construido utilizando modelos geométricos complicados para lograr la apariencia de un robot Kuka KR-6 de 6 grados de libertad. Este robot virtual es manipulado desde el mundo virtual en VRML lo cual tiene sus ventajas y desventajas. Al hacer la manipulación desde el mundo virtual localmente, la velocidad de respuesta es muy alta; pero si la manipulación se hace desde un programa o aplicación externa al mundo virtual, es posible tener errores de comunicación si no se contempla una sincronía adecuada para esta comunicación.

Otro proyecto interesante desarrollado en Alemania es el de la Universidad de Hagen, donde los miembros del proyecto *RichOLD (Enriching Open and Distance Learning by knowledge sharing for collaborative computer-based modelling and simulation)* [Jochheim 00],

desarrollaron varios robots en VRML que pueden ser manipulados mediante *applets* de Java. Estos *applets* permiten introducir datos al mundo virtual tales como los valores de las diferentes variables articulares dependiendo del tipo de robot.

En el instituto de Investigación en Comunicación y Cibernética de Nantes (IRCCyN), en Francia, el Dr. Chellali ha logrado grandes avances en el área de Robótica por Internet, como lo es el control de un robot CRS A456 por medio de un Guante Virtual (*Data Glove*) [Chellali 01]. A través de este guante un usuario puede mover la mano en la realidad, ésta a su vez interactúa con el mundo virtual, el cual de alguna manera realiza la comunicación con el robot real para lograr su teleoperación. Chellali también ha desarrollado un monitoreo virtual de un robot Puma 560. El robot realiza sus movimientos en el laboratorio real y, con el fin de usar el menor ancho de banda posible, una cámara toma imágenes de la posición del robot, y a través de un procesamiento se obtienen los valores de las variables articulares del robot, las cuales se envían al cliente remoto para que el robot en el ambiente virtual adopte la posición del robot real. No es necesario enviar imágenes con tamaños muy grandes y por lo tanto muy costosas para transmitirse por Internet. En otro de sus trabajos [Kheddar 97.1], el equipo del IRCCyN logra hacer una teleoperación en paralelo de 4 robots con distintas cinemáticas, situados en diferentes partes del mundo, desde un solo lugar, todos resolviendo el mismo problema. Otro trabajo es el control de navegación de un robot por medio de sensores en el cuerpo humano del cliente [Kheddar 97.2], donde se busca que los movimientos de la cabeza logren dirigir el robot navegador, y con interpretación de voz se controlen las velocidades, obviamente con el costo de tiempo que produce el reconocimiento de voz.

En sus trabajos, Chellali ha propuesto soluciones utilizando C++ para definir los programas servidor, comunicados directamente con los robots, y Java3D del lado del cliente para la creación de los mundos virtuales. La comunicación entre el cliente y el servidor se realiza por medio de *sockets* [Chellali 97, Chellali 98.1, Chellali 98.2].

1.4 Organización de la tesis

El resto de la tesis está organizada como se describe a continuación.

El capítulo 2 describe los componentes y funciones del robot industrial UNIMATE S-103 utilizado en el proyecto, la infraestructura utilizada, así como las herramientas de software que sirvieron de apoyo para la realización del proyecto. En el capítulo 3 se presenta el diseño del sistema desarrollado. En el capítulo 4 se presentan los mundos virtuales desarrollados y la manera en que se logra que estos mundos sean completamente interactivos y bidireccionales, con el uso de Java.

El capítulo 5 presenta la implementación para lograr la telemanipulación del robot vía Internet. En el capítulo 6 se hace la presentación de aplicaciones de autoprogramación desarrolladas con el Laboratorio Virtual, así como el funcionamiento del sistema aplicado a un robot móvil. En el capítulo 7, finalmente se presentan algunos resultados y conclusiones, así como algunos comentarios acerca de la experiencia obtenida, y los trabajos futuros que pudieran o están desarrollándose sobre el tema.

Capítulo 2
Descripción de la Infraestructura Utilizada

La infraestructura utilizada en el desarrollo de esta tesis fue el robot industrial UNIMATE S-103, algunos periféricos como servidores y cámaras y, software. Esta infraestructura se describe en este capítulo.

2.1 Robot Industrial UNIMATE S-103

El UNIMATE S-103 es un robot industrial cuyas aplicaciones más comunes son el ensamble mecánico y/o electrónico. Sus principales características son: fiabilidad y robustez, su fácil operación, una alta velocidad y gran resolución. Este robot puede concebirse como si estuviera formado por dos sistemas: el mecánico y el electrónico [Malo 93]. La Figura 2.1 muestra el aspecto de este robot.

Sistema mecánico. Se trata de un robot con 3½ GDL (grados de libertad) con arquitectura SCARA (*Selective Compliance Assembly Robot Arm*). Los primeros tres grados de libertad consisten en rotaciones alrededor de ejes verticales, paralelos entre sí, correspondientes a las articulaciones hombro, codo horizontal y muñeca, los cuales le permiten moverse de modo que la muñeca se pueda ubicar en cualquier punto de un plano horizontal; mientras que esta última orientaría la mano en cualquier dirección dentro de dicho plano. La cuarta articulación consiste en un movimiento traslacional a lo largo de un eje vertical coincidente con el eje rotacional de la muñeca. Este movimiento es binario (*on-off*), por lo que, abusando del lenguaje, se le considera como ½ GDL. Un último movimiento es el de apertura y cierre de los dedos de la pinza, el cual

también es binario. Las primeras tres articulaciones están animadas por sendos motores de Corriente Directa (CD) lo que permite tenerlas servocontroladas. Los actuadores de los dos movimientos de la pinza son pistones neumáticos, por lo que su movimiento es binario.

Sistema electrónico. La arquitectura computacional de este robot está constituida por tres tarjetas controladoras basadas en el microprocesador Z-80 para controlar cada uno de los tres movimientos servocontrolados del robot. Una tarjeta general basada en el mismo *chip* se encarga de la coordinación de movimientos, de la programación y de la comunicación con el exterior a través de un sistema de canales binarios de entrada y de salida, así como de una botonera externa (*teach pendant*) conectada a uno de los puertos serie del controlador. La función del *teach pendant* es la de proporcionar una interfaz con el usuario para la programación y el control del robot. En el gabinete de control, además de todo esto, se encuentran las etapas de potencia de los tres servomecanismos del robot. Cabe destacar que la arquitectura computacional, así como la algorítmica y el software de programación y de control de este robot industrial constituyen un "sistema cerrado", es decir, que no es posible hacer la más mínima modificación en sus circuitos o en sus programas. Sin embargo, esto no significa que no podamos utilizarlo como un todo dentro de sistemas complejos de control con retroalimentación sensorial o de programación de alto nivel como el desarrollado en el marco del presente proyecto.

Figura 2.1. Aspecto del robot industrial UNIMATE S-103

2.1.1. Operación del Robot

La operación del robot se hace desde la botonera externa (Figura 2.2), una interfaz *hardware* de programación (también conocida como *teach pendant*), que permite a un usuario interactuar con el robot a través de sus diferentes modos de operación. El *teach pendant* posee varios modos de operación que se describen a continuación.

Figura 2.2. Imagen de la botonera externa

Modo STARTUP (Arranque). Al encender el robot se entra al modo de arranque (STARTUP), en el cual se corren dos rutinas de verificación de la electrónica correspondiente al sistema de control del robot: la primera verifica la electrónica digital y la segunda verifica la electrónica analógica de potencia. Una vez verificado el sistema, C/ROS (Cyber / Robot Operating System) permite al operador que sea energizado el sistema de potencia. El modo de arranque permite además la calibración de los sensores de posición del robot mediante el envío del mecanismo a la posición HOME, pulsando en la botonera la tecla correspondiente. Antes de energizar al robot se debe verificar que el brazo se encuentra a la izquierda de su posición de reposo (HOME).

Modo EXEC (Ejecutivo). Este modo (*executive*) es el principal de C/ROS. Da acceso a los siete diferentes modos de operación del robot, los cuales se describen más adelante. Estando en cualquiera de los modos de operación es posible tener acceso al menú de modos pulsando en la botonera la tecla EXEC, la cual activa el modo ejecutivo.

Modo SET (Inicialización).Si bien se tiene acceso a este modo de operación desde el modo Ejecutivo, también es posible acceder a él directamente después del arranque. En este modo se tiene acceso a los parámetros globales del sistema bajo la denominación de articulación

cero (*axe* 0), tales como el número de ejes controlados, las velocidades máximas y mínimas tanto de operación como de *jogging*, etc. Además, este modo permite la programación de los parámetros del controlador de cada una de las articulaciones servocontroladas de manera individual (*axe i*, con $i = 1,2,3$), tales como ganancias, tiempos de respuesta, corrientes máximas, etc. No debe permitirse el acceso remoto a este modo de operación, pues los usuarios remotos tendrían acceso a la configuración de más bajo nivel del robot..

Modo PGM (Programación del robot). Este modo (*program*) permite crear un programa, en donde la tarea por realizar se especifica como una secuencia de comandos a ejecutar por el robot, con un máximo de hasta 500 líneas de comandos. Para ello se dispone de un conjunto de cuatro comandos de movimiento (MP, MT, MS, GR), seis comandos de control de flujo (LB, GO, IF, CS, RE, DE), cuatro comandos para el manejo de sensores binarios (WT, WF, TT, TF), cuatro comandos misceláneos (SP, SY, ST, NP) y cuatro comandos de ayuda para la edición de programas (INS, DEL, FIX, NUM). Los comandos disponibles se muestran en la Tabla 2.1. Muchos de estos comandos requieren de un argumento, el cual a menudo no debe salir de un cierto rango. Cabe mencionar que los comandos de movimiento utilizan como argumento una posición simbólica denotada con un número entre 0 y 94, en donde el 0 es la posición correspondiente a HOME o posición de reposo, para la cual todos los sensores leen el valor cero. Cuando se entra al modo de operación PGM se tiene un menú con los primeros cuatro comandos de la lista mostrada en la Tabla 2.1, los cuales pueden cambiarse por grupos de cuatro mediante el *scroll* de la botonera.

Modo TCH (Aprendizaje de los puntos). Una vez que, a través del modo PGM, se ha especificado cada uno de los movimientos señalando puntos simbólicos del 1 al 94 (el 0 es *home*), en el modo TCH (*teach*) se guardan en memoria las coordenadas articulares del robot correspondientes a cada uno de los puntos o lugares a los que debe llegar o por los que debe pasar el robot durante la ejecución de la tarea. Existen tres formas de registrar en memoria estos puntos: i) moviendo el robot con las teclas de la botonera para ello destinadas (*jogging*) en lo que se conoce como opción TEACH; ii) moviendo manualmente el extremo del robot, habiendo cortado previamente la energía de los servomecanismos; iii) o editando el valor de cada uno de los tres sensores o coordenadas articulares mediante la opción EDIT. Cada vez que se registra una posición, se debe pasar a la siguiente utilizando la tecla NEXT (posición siguiente). Con la tecla PREV (posición previa) se puede regresar a la anterior posición. Además, desde el modo TCH es posible operar los canales binarios de salida mediante la opción OUT, o bien, leer los canales

binarios de entrada usando la opción IN. En este modo el robot puede aceptar información de posición generada en el exterior, por ejemplo desde una PC remota, a fin de teleoperarlo.

RUN. Para poder ejecutar un programa que ya cuenta con la base de datos correspondiente a las coordenadas articulares de todos los puntos de interés, es necesario entrar a este modo de operación. Ofrece las opciones de ejecución continua y la de paso a paso. En ambas opciones, es posible supervisar visualmente el comando ejecutado en el *display* del *teach pendant*.

Comando	Descripción	Comando	Descripción
MP (*n*)	Ir al punto *n*	CS (*e*)	Llamar subrutina *e*
MT (*n*)	Pasar por el punto *n*	LB (*e*)	Definir etiqueta *e*
MS (*n*)	Inicia movimiento hacia el punto *n*	RE	Retorno de subrutina
GR (0)	Abrir pinza	SY	Sincronizar con evento *n*
GR (1)	Cerrar pinza	WT (*c*)	Esperar que *c* sea verdadero
GR (2)	Levantar pinza	WF (*c*)	Esperar que *c* sea falso
GR (3)	Bajar pinza	DE (*d*)	Retardo de $d * 0.1$ seg
SP (*s*)	Velocidad de trabajo, $1 \leq s \leq 12$	ST	Parar proceso
ON (*c*)	Encender canal (o bandera) *c*	NP	No operar
OF (*c*)	Apagar canal (o bandera) *c*		
TT (*c*)	Probar si *c* es verdadero	INS	Insertar instrucción
TF (*c*)	Probar si *c* es falso	DEL	Borrar instrucción
IF (*e*)	Si es verdadero salta a etiqueta *e*	FIX	Restablecer la declaración
GO (*e*)	Salto incondicional a etiqueta *e*	LIN (*n*)	Desplegar la línea *n*

Tabla 2.1. Lista de Comandos disponibles en el modo PGM del Lenguaje C/ROS

Modo RN (Ejecución del programa). Una vez programada una tarea mediante el modo PGM y registrados los puntos de interés para la misma usando el modo TCH, ya se puede ejecutar dicha tarea. Para ello, se dispone del modo RN (*run*) que permite la ejecución de un programa ya sea de manera continua o paso a paso. En ambos casos es posible supervisar visualmente el comando ejecutado en el *display* del *teach pendant*. Este modo ofrece las siguientes opciones:

RUN: Ejecutar un programa.

RES: Reanudar la ejecución de un programa que haya sido suspendido, a partir de la última instrucción ejecutada.

STP: Ejecución de un programa paso a paso. Cada vez que se pulsa el botón programable correspondiente se ejecuta la instrucción corriente.

STOP: Cuando el programa esta corriendo, esta opción permite detenerlo después de efectuarse la instrucción corriente.

De manera complementaria, mediante el botón BREAK se puede detener inmediatamente la ejecución de un programa en cualquier instante, interrumpiendo la instrucción que se está realizando.

Modo LD (Cargar/Salvar programa). Este modo permite la transferencia del programa corriente y de sus puntos desde la memoria del robot hacia un dispositivo externo o viceversa. Este dispositivo externo puede ser una PC conectada al puerto serie (RS-232-C) auxiliar del robot o bien un dispositivo de almacenamiento de información, por ejemplo un *disc drive*, conectado al controlador del robot. En este modo se tienen las funciones siguientes:

LD: Cargar datos (disco).

SV: Salvar datos (disco).

ULD: Cargar datos de computadora externa.

DLD: Salvar datos a computadora externa.

Modo CLR (Borrar datos). Este modo se usa para borrar el programa corriente o los puntos que utiliza, ambos almacenados en la memoria del sistema. Este modo cuenta con dos funciones opcionales:

PGM: Para borrar programa.

PNTS: Para borrar los puntos.

Modo LOC (Bloqueo de la botonera). Al seleccionar este modo el teclado de la botonera se bloquea, de manera que ya no acepta más comandos. Con el fin de desbloquearla de debe pulsar una cierta secuencia de teclas que sólo debe ser conocida por el administrador del sistema.

2.2 Estructura de Servidores y Red

Además del robot UNIMATE S-103, el sistema computacional que soporta el Laboratorio Virtual de Robótica está constituido por 3 servidores, en los cuales residen los diferentes programas de aplicación. Dichos servidores se comunican entre ellos cuando es necesario, o bien, simplemente realizan comunicación con la aplicación del usuario remoto conectado. En este

sistema, no solo se utilizan comunicaciones cliente-servidor, sino que la comunicación puede establecerse entre el cliente y cualquiera de los tres servidores, o entre los servidores, y aún se contemplan casos en los que los clientes se pueden comunicar entre ellos. En la Figura 2.3 se muestra el esquema general del laboratorio mostrando los servidores y todo lo que conforma al laboratorio. La configuración de los servidores fue diseñada por el autor, tomando en cuenta cuestiones como seguridad, desempeño y flexibilidad, por ejemplo, en caso de que el robot real deje de funcionar, el sistema sigue operando sin problemas. Otro ejemplo sería que, el sistema de visión puede estar procesando mucha información, y este diseño ayuda a independizar procesadores.

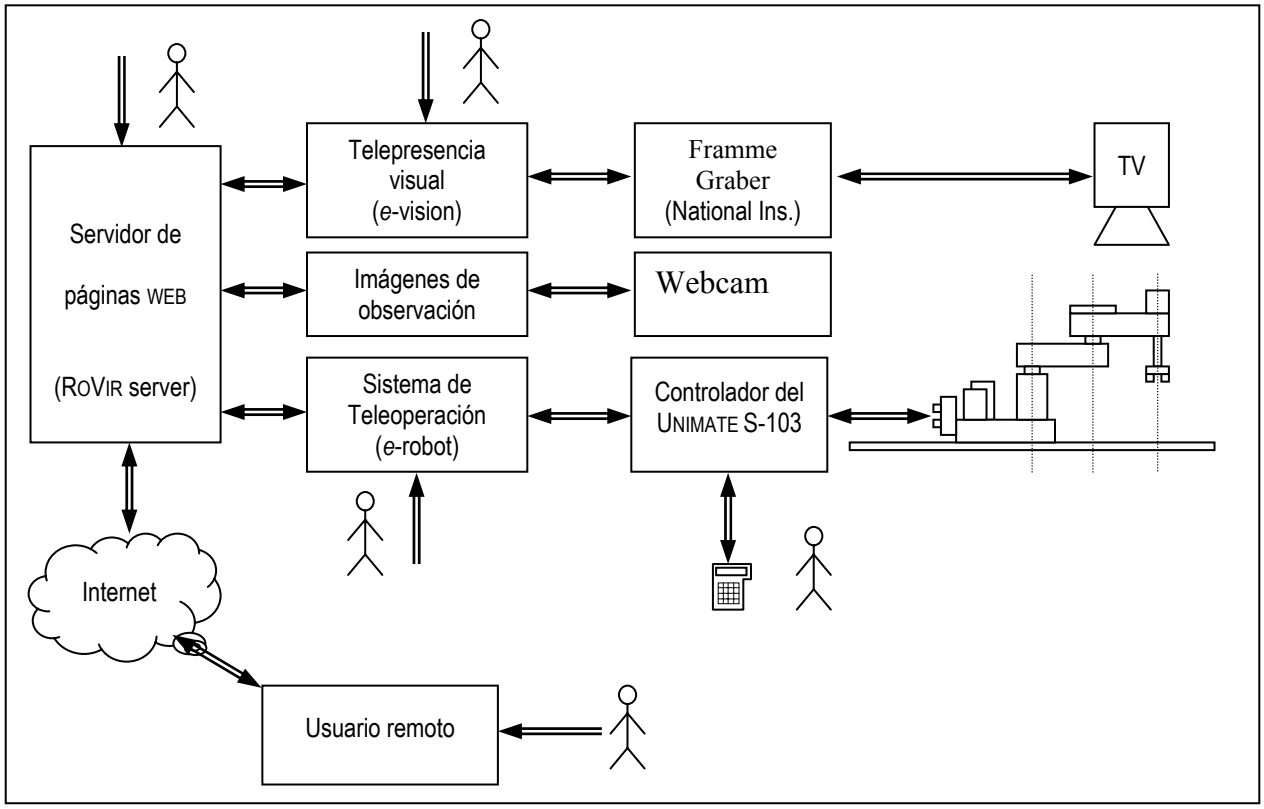

Figura 2.3. Esquema del Laboratorio Virtual de Robótica.

El servidor principal a través del cual los usuarios pueden acceder al Laboratorio Virtual, se denomina ROVIR y su dirección electrónica es *rovir.ctrl.cinvestav.mx*. En este servidor se encuentran las páginas WEB necesarias para que los usuarios remotos puedan enlazarse con el Laboratorio Virtual o con la infraestructura real que posee. Dichas páginas pueden encontrarse en este servidor, o bien puede tenerse acceso a ellas mediante el redireccionamiento a otro servidor.

Un segundo servidor es el que se encarga de establecer comunicación directa entre un usuario remoto y el robot conectado a uno de sus puertos serie. Este servidor fue llamado *e*–robot y su dirección electrónica es *erobot.ctrl.cinvestav.mx*. Aquí se encuentran las aplicaciones que funcionan como servidor de las aplicaciones cliente que se utilizan para teleoperar al robot, o para establecer alguna comunicación con el robot. Las aplicaciones servidor disponibles son TeachPendantServer y RobotServer, las cuales son conocidas también como *teach pendant* virtual. La primera de estas aplicaciones es la que recibe un programa en C/ROS, lenguaje nativo del robot utilizado, para transmitírselo después a dicho robot. El programa en C/ROS se crea con ayuda de la interfaz de programación y más tarde se prueba mediante simulación utilizando el robot virtual. Después puede ser enviado al servidor *e*–robot para que el robot real ejecute dicho programa. La aplicación RobotServer es la que atiende al RemoteTeachPendant, interfaz que utiliza un usuario remoto para teleoperar en tiempo real al robot UNIMATE S-103 conectado al servidor *e*–robot.

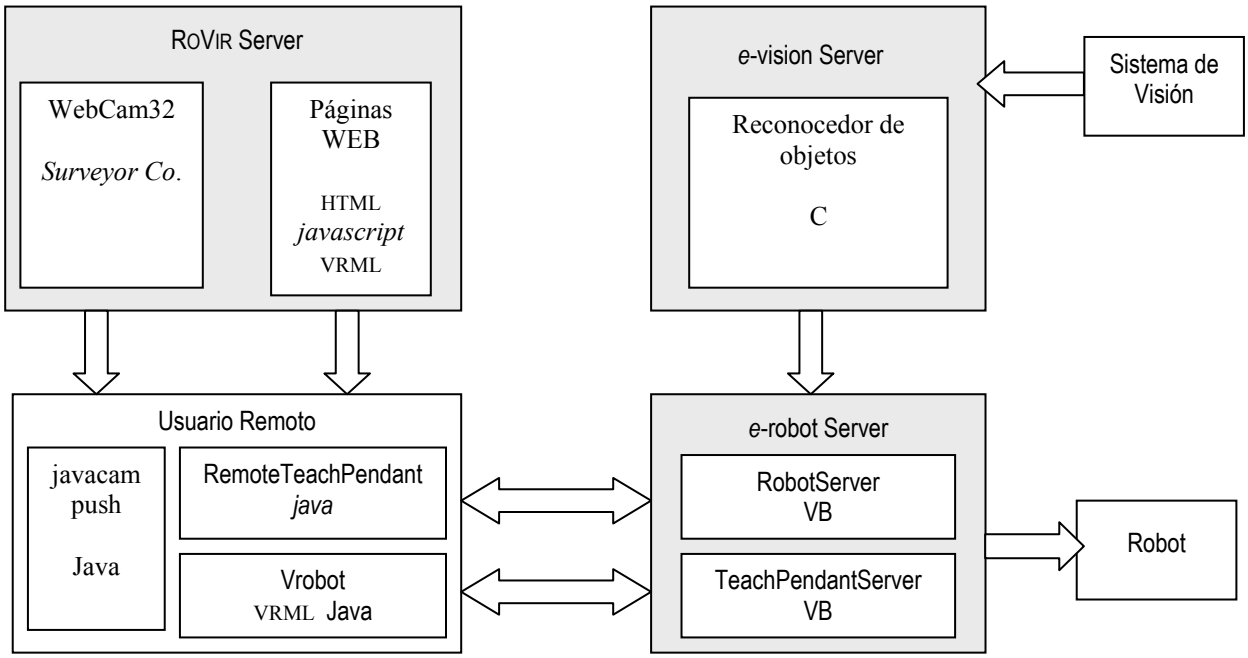

Figura 2.4. Diagrama de interacción cliente-servidor.

Las aplicaciones de visión artificial se encuentran en un tercer servidor llamado *e*–vision, cuya dirección electrónica es *evision.ctrl.cinvestav.mx*. En este servidor se ejecutan las aplicaciones necesarias para hacer el reconocimiento de objetos reales presentes en el área de

trabajo del robot. Otra de las aplicaciones residentes en el servidor *e*–vision es la que permite realizar el modelado 3D de la geometría de dichos objetos utilizando el lenguaje VRML. Dichos modelos serán enviados posteriormente al servidor ROVIR para actualizar el mundo virtual correspondiente al puesto de trabajo robotizado, con el dibujo 3D de todos los objetos por manipular.

En la Figura 2.4 se muestra la interacción que existe entre los servidores del laboratorio y el *software* que reside en ellos.

2.3 Software

El Software utilizado en el desarrollo del sistema ROVIR fue VRML y Java. La evaluación de varios lenguajes para el modelado (OpenGL, Java 3D y VRML) y para teleoperación (CGI's, C y Java) y la selección de los utilizados fue también realizada por el autor.

2.3.1. VRML

La tecnología VRML, cuyo nombre proviene de las siglas inglesas Virtual Reality Modeling Language (Lenguaje de Modelado de Realidad Virtual), nació en mayo de 1994 de la mano de Silicon Graphics, cuando Marc Pesce y Toni Parisi presentaron el artículo que describe el *maze*, prototipo de una interfaz 3D para el WEB [VRML 96]. Este artículo comenzó el desarrollo del lenguaje de modelado de Realidad Virtual VRML. Poco después se liberó una extensión sintáctica llamada VRML 2.0, mientras que la versión VRML97, aparecida en 1997, se convirtió en la norma del ISO (International Organization for Standardization) y del IEC (International Electrotechnical Commission): International Standard ISO/IEC 14772-1:1997. El título completo de esta norma internacional es: *Information technology; Computer graphics and image processing; The Virtual Reality Modeling Language (VRML) Part 1: Functional specification and UTF-8 encoding* [VRML 97].

Básicamente, esta tecnología se puede definir como un formato de archivo que permite especificar objetos 3D que pueden ser visualizados desde un *browser* de Internet (Internet Explorer, Netscape, etc.) añadiendo simplemente un *plugin* para este fin. VRML 2.0 es un poderoso lenguaje para crear mundos virtuales que incluyen objetos 3D, colores, animaciones e

imágenes a partir de archivos de texto; no obstante es un lenguaje simple y fácil de usar. Los archivos que se construyen en este lenguaje corresponden a mundos virtuales (*virtual worlds*) tridimensionales, de ahí que la extensión usada para estos archivos sea *wrl*.

Todos los archivos creado en el lenguaje VRML 2.0 son archivos de texto (ASCII) y, por lo tanto, pueden ser editados con cualquier editor de textos convencional, aunque existen editores comerciales para tal fin que aportan ayudas considerables par la creación de mundos virtuales VRML. Actualmente se está trabajando en la creación de un formato binario para estos archivos, aunque aún no se utiliza. La principal ventaja del formato binario es que tiene una mayor tasa de ocultación del código fuente y, por lo tanto, será más difícil poder copiar los contenidos. También tiene la ventaja de producir archivos de un menor tamaño, por lo que serán más aptos para circular por la siempre congestionada red de Internet.

Antes de proceder a una breve presentación del lenguaje VRML, mencionamos que, para su correcta identificación por parte de un navegador, todos los archivos correspondientes a mundos virtuales escritos en VRML, deben incluir el encabezado siguiente:

VRML V2.0 utf8

La primera parte de este encabezado (#VRML V2.0) denota el tipo y la versión. La segunda parte (utf8), permite utilizar la codificación UTF-8 para así poder trabajar con todos los caracteres especiales (acentos, caracteres japoneses, cirílico, etc.) de acuerdo con el estándar ISO 10646.

Nodos, Campos y Eventos

En los archivos correspondientes a un cierto mundo virtual hecho con VRML, lo que se define es simplemente un grupo de objetos. Estos objetos pueden contener geometría 3D, imágenes, colores, etc., y son llamados nodos (del inglés *nodes*). Estos nodos tienen algunas características variables que se pueden definir mediante los campos (del inglés *fields*), los cuales funcionan como parámetros de los nodos. Para ilustrar esto, en el Programa 2.1 se presenta un ejemplo sencillo de un programa en VRML, mientras que la Figura 2.5 muestra el mundo virtual correspondiente.

```
#VRML V2.0 utf8
Shape {
      geometry Box {
            size 1 1 1 }# Dimensiones X, Y y Z
}
```

Programa 2.1. Un programa sencillo en VRML.

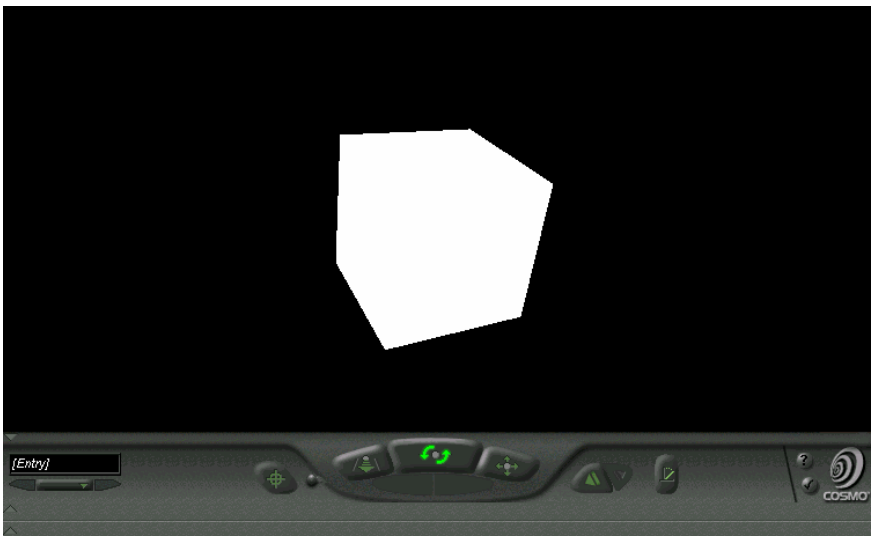

Figura 2.5. Ejemplo correspondiente al Programa 2.1.

En el ejemplo anterior simplemente se crea una caja (*Box*) en VRML. En el código que se muestra en el programa 2.1, lo primero que se hace es definir un encabezado que indica que el archivo es un programa en VRML. Posteriormente se crea un nodo *Shape*, que permite crear figuras por medio de su campo *geometry*. Al campo *geometry* se le asigna un nodo de tipo Box, que indica que la figura a crear será un prisma rectangular con las dimensiones *size* especificadas. El campo *size* recibe 3 valores en unidades de VRML, los cuales indican el largo, ancho y alto de la figura geométrica. Nótese que cuando se asigna el mismo valor para las 3 dimensiones, se genera un prisma cuadrangular o cubo. El cubo visualizado en la Figura 2.5 aparece en las coordenadas origen del mundo virtual y con color blanco, o bien, sin color. Esto se debe a que no se especificaron estos valores en el código del Programa 2.1.

Los campos pueden ser univaluados, cuando se usa un solo valor para definirlos, o pueden ser multivaluados, cuando se necesita una lista de valores para definirlos. Los campos tienen un tipo de dato asociado; es decir, cada campo puede contener datos de un cierto tipo. Por ejemplo,

el campo *size* es de tipo vector de 3 elementos flotante, y por lo tanto se dice que es de tipo *SFVec3f* (Single Valued Field Vector 3 Floating Point). Un nodo puede ser un parámetro de otro nodo y, por lo tanto, puede ser un campo. Esta situación aparece cuando un nodo es un agrupador de varios nodos. Un nodo agrupador sirve para que varios objetos compartan unas mismas propiedades, por ejemplo: mismas transformaciones geométricas, mismo comportamiento, mismo color o textura, etc.

Los eventos (del inglés *events*) pueden ser pensados como mensajes que van circulando entre los nodos y que hacen que se puedan variar los parámetros de un objeto durante la navegación del entorno virtual. En realidad, los eventos son unos campos especiales. Los eventos pueden ser de entrada (*eventIn*), de salida (*eventOut*) o de entrada y salida simultáneamente (*exposedField*).

Los *eventIn* son campos que aceptan eventos dirigidos a ellos, mientras que los *eventOu*t son campos que pueden enviar eventos a *eventIn* de otros nodos o a algún *exposedField* de otros nodos. Por su parte, los *exposedField* son campos que pueden enviar eventos a *eventIn* y a *exposedField* de otros nodos y también aceptan eventos dirigidos a ellos.

Los nodos referentes a primitivas de figuras que se pueden crear en VRML son cajas (*Box*), esferas (*Sphere*), cilindros (*Cylinder*) y conos (*Cone*).

2.3.2. Interfaz de Comunicación Externa

Para establecer una comunicación entre un mundo virtual creado con VRML y un ambiente externo cualquiera es necesario crear una interfaz entre ambos. Esta interfaz ha sido denominada Interfaz de Comunicación Externa o simplemente ECI (*External Communication Interface*). El desarrollo de la metodología de comunicación utilizada por la ECI es una de las partes fundamentales de este proyecto de tesis [Marrin 97].

La ECI está diseñada para permitir que un ambiente externo, programado mediante un lenguaje cualquiera, pueda acceder a los nodos en una escena virtual VRML usando el modelo de eventos VRML existente. En este modelo, el evento de salida de un determinado nodo puede ser enviado a un *eventIn* de otro nodo. Cuando el *eventOut* de algún nodo genera un evento, el *eventIn* del destino es notificado y su nodo procesa el evento. La ECI adapta la interfaz para dar

acceso a los nodos del ambiente externo. Conceptualmente, la ECI permite cuatro tipos de acceso a una escena virtual VRML:

1. Acceso al mundo virtual desde la interfaz de un *script* del *browser*.
2. Envío de eventos a los *eventIn* dentro del mundo virtual.
3. Lectura de los valores enviados por los *eventOut* de los nodos dentro del mundo virtual
4. Obtención de la notificación de cuando los eventos son enviados por los *eventOut* de los nodos dentro del mundo virtual.

Acceso a los nodos. La mayoría de las operaciones en la ECI empiezan por obtener una referencia al nodo. Para lo cual, dicho nodo debe ser nombrado usando un constructor DEF. Cuando se obtiene la referencia, los *eventIn* y *eventOut* del nodo pueden ser accedidos. Ya que un *exposedField* implícitamente contiene un *eventIn* y un *eventOut*, estos también pueden ser accedidos usando los modificadores *set* y *changed*. El modificador *set* permite establecer valores en los campos de los nodos, mientras que *changed* me permite saber los valores que están siendo modificados en los campos.

Ya que se ha referenciado un nodo para hacerlo accesible desde un ambiente externo, un evento puede ser enviado a cualquier *eventIn* de ese nodo. El dato enviado es específicamente del tipo de dato del campo del *eventIn*. Los tipos de datos usados por VRML tienen la misma estructura que los usados por Java, por lo tanto, usando Java no es necesario estar dando formatos específicos a los datos, pero es necesario usando un lenguaje que tenga una estructura de datos diferente.

Lectura de valores de los *eventOut*. Cuando ha sido referenciado desde el ambiente externo, cualquier *eventOut* de un nodo puede ser leído. El valor leído es el último valor enviado por un *eventOut* en el mundo virtual. El dato leído es del tipo de dato del campo del *eventOut* del nodo, y para poder capturar ese dato en el lenguaje utilizado, es necesario hacer un *casting* para obtener el formato correcto.

Notificación de cambios en los *eventOut*. Un ambiente externo puede registrar una notificación cuando se han dado cambios en los *eventOut*. El registro se hace al nodo que contiene dicho *eventOut*. Durante el proceso de registro, el ambiente externo recibe dos datos, el

valor del evento y la notificación. Con la notificación es posible identificar cual de los nodos es el que ha enviado el evento, y de esa manera poder procesar correctamente el valor del evento.

2.3.3. Java

En el contexto de este proyecto se ha considerado que Java representa la mejor opción para programar las interfaces de aplicación del laboratorio virtual, debido a que ofrece una gran conectividad con VRML a través de la ECI [Marrin 97]. Esta decisión se basa en las características principales de Java que se exponen brevemente a continuación.

Java ofrece toda la funcionalidad de un lenguaje orientado a objetos potente, pero sin las características menos usadas y más confusas de éstos [Deitel 98]. Java elimina muchas de las características de otros lenguajes, como C++, para mantener reducidas las especificaciones del lenguaje y añadir características muy útiles como el *garbage collector* (reciclador de memoria dinámica) [Internet 1]. Gracias a esta característica no es necesario preocuparse de liberar memoria.

Java implementa la tecnología básica de C++ con algunas mejoras y elimina algunas cosas para mantener el objetivo de la simplicidad del lenguaje. Java trabaja con sus datos como objetos y con interfaces a esos objetos. Soporta las tres características propias del paradigma de la orientación a objetos: encapsulación, herencia y polimorfismo. Las plantillas de objetos son llamadas, como en C++, *clases* y sus copias, *instancias*. Estas instancias, como en C++, necesitan ser construidas y destruidas en espacios de memoria.

Java incorpora funcionalidades inexistentes en C++ como, por ejemplo, la resolución dinámica de métodos. Esta característica deriva del lenguaje Objective C, propio del sistema operativo NeXT Step. En C++ se suele trabajar con librerías dinámicas (DLL) que obligan a recompilar la aplicación cuando se retocan las funciones que se encuentran en su interior. Este inconveniente es resuelto por Java mediante una interfaz específica llamada RTTI (*RunTime Type Identification*) que define la interacción entre objetos excluyendo variables de instancias o implementación de métodos. Las clases en Java tienen una representación en el *RunTime* que permite a los programadores interrogar por el tipo de clase y enlazar dinámicamente la clase con el resultado de la búsqueda [Lemay 96].

Java se ha construido con extensas capacidades de interconexión TCP/IP. Existen librerías de rutinas para acceder e interactuar con protocolos como *http* y *ftp*. Esto permite a los programadores acceder a la información a través de la red con tanta facilidad como a los archivos locales. La verdad es que Java en sí no es distribuido, pero proporciona las librerías y herramientas para que los programas puedan ser distribuidos, es decir, que se corran en varias máquinas, interactuando.

2.3.4. Otras herramientas utilizadas

Los servidores utilizan el sistema operativo Microsoft Windows 98, mientras que la aplicación de servidor WEB utilizada es OmniHttpd, de *Omnicron Technologies Corporation* [Internet 2]. Este software, de libre distribución, permite tener una administración de usuarios a través del servicio WEB, lo cual es de gran ayuda para la seguridad en el acceso a los programas de teleoperación, pues es muy importante tener cuidado al dejar esperando conexiones en las aplicaciones de servidor, y que no cualquier usuario pueda acceder a estas paginas y hacer teleoperación.

En las aplicaciones de interfaz de programación y TeachPendantRemoto, es posible monitorear al robot para poder visualizar la ejecución de las tareas previamente programadas o bien de la teleoperación en curso. Esto es posible a través de una *webcam* conectada al puerto USB del servidor ROVIR y de una aplicación que permite enviar imágenes a través de la red mediante una página WEB. Con esto los usuarios remotos pueden ver lo que la cámara esté enfocando. El software utilizado para hacer la captura de vídeo es Webcam32 6.0 de *Surveyor Corporation* [Internet 3], el cual tiene una versión *demo* de libre distribución. Para poder visualizar el monitoreo de la cámara a través de la página WEB diseñada, *Surveyor Corporation* ofrece unas clases de Java que funcionan como *applets* que pueden incluirse en una página WEB. Con esto los usuarios remotos visualizan el ambiente del laboratorio.

Los programas en Java fueron hechos en una aplicación que ofrece un agradable ambiente de programación, la aplicación es Kawa IDE Profesional 5.0, la cual esta disponible en versiones de evaluación por la empresa *Allaire Corporation*.

Los programas en VRML se desarrollaron en el ambiente de programación que ofrece la aplicación VRMLPAD 2.0 de *Parallel Graphics*. Esta aplicación también tiene una versión de evaluación disponible en Internet.

El intérprete de VRML que utilizamos en nuestro proyecto es un *plugin* de libre distribución llamado CosmoPlayer, el cual es ofrecido por *Silicon Graphics Incorporation*. Se tienen otros visualizadores (*browsers*) como Platinum y Blaxxun.

El compilador de Java, así como las clases necesarias para poder realizar los programas también son de libre distribución, y se pueden encontrar en las páginas de Sun Microsystems.

Capítulo 3
Diseño del Laboratorio Virtual de Robótica

En este capítulo se presenta el diseño del sistema del Laboratorio Virtual de Robótica. A la infraestructura descrita en el capiítulo anterior, este sistema permite ofrecer el uso del mundo virtual desarrollado, la teleoperación del robot real y autoprogramación del robot.

3.1 Organización del Laboratorio Virtual de Robótica

La Figura 3.1 muestra los módulos de software desarrollados que conforman el Laboratorio Virtual de Robótica. Las flechas muestran la interacción entre los mismos a través del intercambio de información. Dicha figura se encuentra en el Capítulo 1, y es repetida en este por comodidad al lector.

El módulo Interfaz de Usuario permite una completa manipulación del mundo virtual modificando los valores de las variables articulares del robot. Además, permite actualizar el espacio virtual generando objetos en el area de trabajo del robot, los cuales pueden ser manipulados generando automaticamente un programa en C/ROS. Este programa puede ser simulado en el mundo virtual o puede ser enviado al mundo real para ser ejecutado por el robot real.

El mundo virtual es el módulo donde se presenta el modelo tridimensional animado e interactivo del puesto de trabajo robotizado, incluyendo el robot y todos los elementos electromecánicos que lo componen. Este mundo virtual emula no sólo el aspecto geométrico de dichos elementos, sino su comportamiento cinemático así como todas sus características computacionales. Por ello el mundo virtual es muy útil para simular los programas en C/ROS desarrollados con la ayuda de la interfaz de usuario o aún los que se programen con los medios tradicionales al alcance del usuario local (*teach pendant* o simuladores).

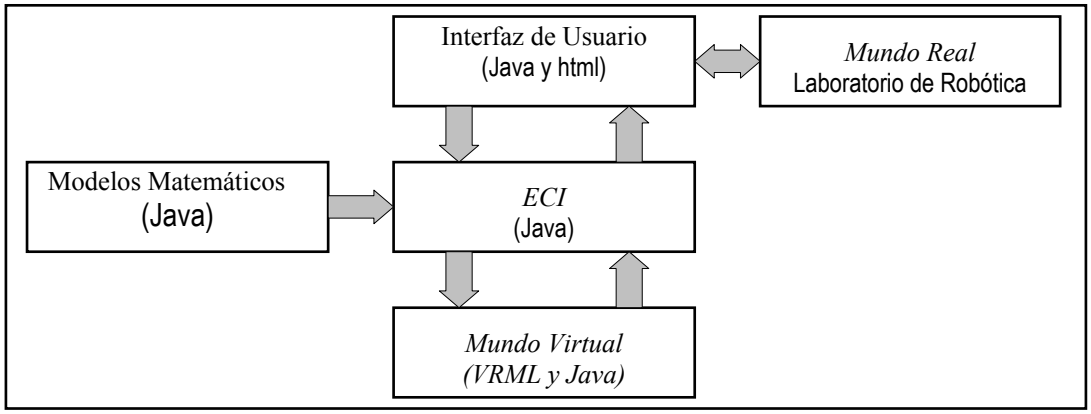

Figura 3.1. Esquema del software que constituye el Laboratorio Virtual de Robótica.

El módulo Modelos Matemáticos, desarrollado en Java, contiene modelos geométricos y cinemáticos con base en la metodología de Denavit-Hartenberg del robot. Estos modelos se usan para mover al robot (virtual y real) evitando que el robot se mueva hacia lugares ocupados por elementos del puesto de trabajo o por él mismo. Esto es necesario pues el robot real no hace la verificación de posibles colisiones cuando se le ordena moverse, lo que podría generar choques consigo mismo o con algún elemento de su puesto de trabajo.

El módulo Mundo Real corresponde al Laboratorio de Robótica del Cinvestav. Este recibe de la interfaz de usuario un programa en C/ROS, previamente simulado en esa interfaz. En el Laboratorio de Robótica del Cinvestav, la recepción de tales programas de usuario se hace a través de una aplicación en Visual Basic corriendo en el servidor e-robot, quien a su vez lo envía al controlador del robot para que la tarea correspondiente sea ejecutada en el robot real. La ejecución de un programa por el robot real puede ser monitoreado desde la interfaz de usuario por

medio de una *webcam* que permite ver el ambiente del laboratorio real a travéz de la página WEB correspondiente.

El módulo ECI se encarga de gestionar entre la Interfaz de Usuario y el mundo virtual, controlando y permitiendo el flujo de información entre los dos módulos. Además envía a los dos módulos los datos provenientes del módulo de Modelos Matemáticos, con el fin de que en el mundo virtual y la Interfaz de Usuario estén contempladas las restricciones cinemáticas del robot.

De todos los módulos descritos, la geometría (apariencia física) del robot y la interfaz de comunicación (vía puerto serie) con el robot fueron desarrollados por Luis Manuel Sastré Leyva y José Andrés Herrera Barragán respectivamente. Todo lo demás es trabajo del autor.

En lo que sigue del capítulo describimos la interacción con el usuario de los módulos.

3.2 Mundo Virtual

La geometría de los objetos que conforman el mundo virtual es representada por el código VRML correspondiente. En la Figura 3.2 se muestra parte de la versión virtual del robot UNIMATE S-103, creado por combinación de primitivas básicas de VRML.

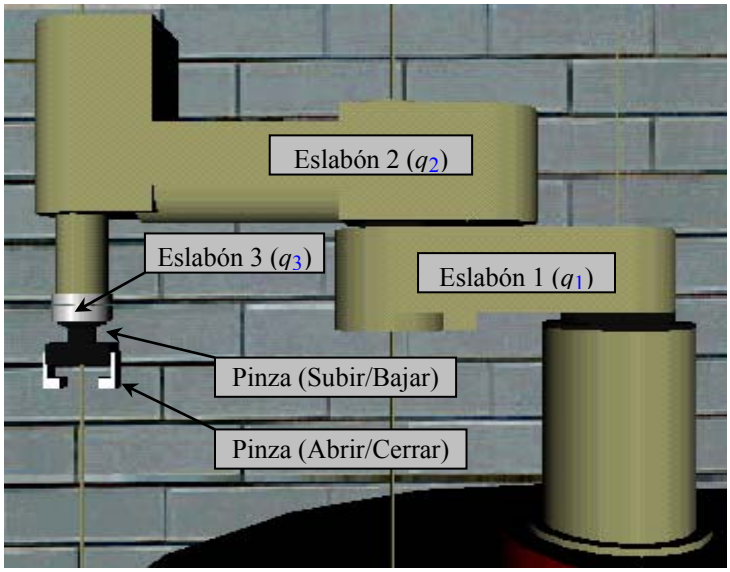

Figura 3.2. Modelo virtual del UNIMATE S-103.

Los objetos (VRML) que representan al robot en el mundo virtual deben tener movimiento de acuerdo a la interacción del usuario mediante el ratón. El programa VRML correspondiente funciona de acuerdo a los movimientos físicos del robot real, calculando la secuencia de posiciones equivalente mediante la cinemática programada en el módulo de Modelos Matemáticos.

El módulo de Modelos Matemáticos contiene el modelado de la Cinemática del Robot UNIMATE S-103. Estos modelos son generales para cualquier robot industrial o de arquitectura similar, por lo tanto se adaptaron para las características particulares del UNIMATE S-103. Esta adaptación fue necesaria, ya que los modelos cinemáticos generales estan descritos para robots de *n* gdl, y el robot utilizado cuenta con 3 gdl.

Existen modelos de cinemática, directa e inversa, a través de los cuales podemos conocer, respectivamente, los valores de las coordenadas cartesianas del robot o las coordenadas articulares del mismo. Las coordenadas cartesianas se refieren a la posición en el plano cartesiano que ocupa el organo terminal del robot, mientras que las cordenadas articulares se refieren al valor angular de cada eslabon del robot con respecto al eslabon anterior.

En esta sección vemos como fueron adaptados estos modelos cinemáticos para el robot UNIMATE S-103. Estos modelos adaptados son la base para el desplegado, en el mundo virtual, del robot cuando se manipulan las variables articulares desde la Interfaz de Usuario, o bien, cuando a través del espacio de cinemática inversa se le especifica al robot que su organo terminal se posicione en una coordenada cartesiana en particular. En las tres subsecciones siguientes se presenta información a más detalle de la cinmática así como la adaptación para el robot UNIMATE S-103.

3.2.1. Modelado de la Cinemática de Robots

Se llama cinemática a la relación que hay entre las variables articulares y las coordenadas cartesianas correspondientes a la posición y orientación del órgano terminal (OT) de un robot.

La cinemática directa básicamente nos ayuda a encontrar la posición en el espacio cartesiano y la orientación de un robot, tomando como referencia las variables articulares. Se puede decir que las variables articulares son los ángulos de orientación de cada eslabón con respecto a un eje en particular, o bien, con respecto a un eslabón anterior.

La cinemática directa, definida como la función $\mathbf{x} = \mathbf{f}(q_1, q_2, ..., q_n)$, permite conocer la posición y orientación del OT del robot, expresada en coordenadas cartesianas, vector **x**, dados los valores de las coordenadas articulares, vector $\mathbf{q} = (q_1, q_2, ..., q_n)$. Un método muy empleado en Robótica para obtener una expresión analítica de la cinemática directa es el que propusieron Denavit y Hartenberg (D-H). En esta metodología se obtiene la caracterización de cada GDL mediante los parámetros del eslabón y la articulación asociadas (el ángulo y la distancia), como los cuatro parámetros D-H. Con cada cuarteta de parámetros D-H se obtiene la matriz de paso generalizado A_i que permite obtener la representación del marco de referencia R_i expresado en el marco de referencia $R_{i\text{-}1}$. Finalmente, multiplicando las n matrices A_i se obtiene una transformación homogénea T_i^n que representa el marco de referencia R_n expresado en el marco de referencia fijo R_0.

En la metodología D-H se comienza por detectar los n ejes articulares q_i, $i = 1, 2, ... n$, numerando los eslabones del robot del 1 al n, de modo que el i-ésimo eslabón tiene en sus extremos a los ejes articulares q_i y q_{i+1}. El segundo paso consiste en caracterizar a cada eslabón por su longitud, definida como la longitud de la normal común a los ejes articulares pasando por sus extremos y designada como a_i. En el tercer paso se asigna un marco de referencia a cada uno de los primeros n-1 eslabones del robot, de modo que el origen del i-ésimo eslabón debe quedar en la intersección de la normal común a_i con el eje articular q_{i+1}; el eje x_i de este marco es la prolongación de la normal común a_i, mientras que su eje z_i se alinea con el eje articular q_{i+1}. Cuando los ejes articulares sucesivos son paralelos, la ubicación de las normales comunes no es única, en este caso, las a_i se ubicarán de tal modo que anulan uno o más parámetros d_i, definidos en el sexto paso de esta metodología. En la Figura 3.3 se muestra el diagrama de dos articulaciones sucesivas.

En el cuarto paso se hace la asignación del marco de referencia asociado al n-ésimo eslabón y al marco de referencia fijo asociado al eslabón cero. El origen del marco del eslabón cero se ubica en la intersección del eje articular q_1 con la superficie de la mesa que soporta al robot, mientras que el origen del marco del último eslabón se pone en el punto del OT del robot que nos interesa posicionar. Generalmente, ese punto es el punto medio entre los dedos de la pinza del manipulador, definiendo el eje z_n paralelo a la dirección de la pinza, mientras que el eje x_i de este marco va del dedo izquierdo al dedo derecho de dicha pinza. Ahora ya se puede definir

la distancia d_n entre los dos últimos eslabones como la distancia entre los orígenes de los marcos de referencia $R_{n\text{-}1}$ y R_n, medida a lo largo del eje z_n.

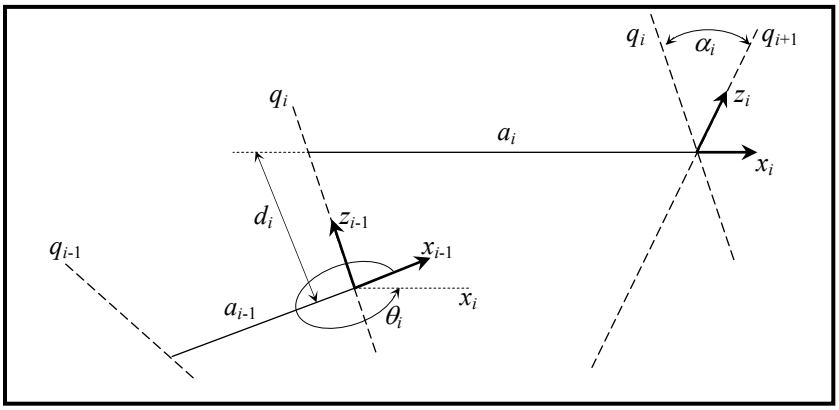

Figura 3.3. Diagrama de Denavit-Hartemberg para obtención de la Cinemática.

En el quinto paso se complementa la caracterización del *i*-ésimo eslabón definiendo su torcedura, α_i, como el ángulo que forman los ejes pasando por sus extremos, q_i y q_{i+1} o bien z_{i-1} y z_i, medido alrededor del eje x_i. Aquí se puede apreciar que el eje x_i coincide con el producto cruz $z_{i-1} \times z_i$. En el sexto paso se caracteriza cada articulación por el ángulo θ_i entre los eslabones que conecta y la distancia d_i entre dichos eslabones. El águlo θ_i se mide alrededor del eje z_{i-1} entre las dos normales comunes que posee, a_{i-1} y a_i o bien entre los ejes x_{i-1} y x_i, mientras que la distancia d_i se mide a lo largo del eje z_{i-1} entre las intersecciones con las dos normales comunes que posee: a_{i-1} y a_i. En el caso de la primera articulación la distancia d1 se mide entre el origen del marco de referencia cero y la intersección entre z_1 y a_1. Resumiendo, los parámetros de Denavit-Hartenberg son aquellos que caracterizan al eslabón (2) y a la articulación (2), de modo que habrá cuatro parámetros D-H por cada GDL del robot, definidos como:

- a_i Longitud de la normal común a los ejes z_{i-1} y z_i, es decir a los ejes articulares q_i y q_{i+1}.
- α_i Ángulo entre los ejes z_{i-1} y z_i medido en un plano normal al eje x_i.
- θ_i Ángulo entre los ejes x_{i-1} y x_i medido en un plano normal al eje z_{i-1}.
- d_i Distancia medida a lo largo del eje z_i entre las intersecciones con sus normales comunes.

Ahora bien, para obtener la matriz homogénea de paso generalizado A_i, es necesario aplicar las transformaciones homogéneas básicas necesarias para confundir los referenciales $R_{i\text{-}1}$ y R_i. Dichas transformaciones elementales son: una rotación de θ_i grados alrededor del eje $z_{i\text{-}1}$;

una translación de d_i centímetros a largo de ese mismo eje, otra translación de a_i centímetros a lo largo del eje x_{i-1} ahora confundido con el eje x_i y, finalmente una rotación de α_i grados alrededor del eje x_i. De modo que la matriz de paso generalizado queda como:

$$
\begin{aligned}
A_i &= Rot(z,\theta_i)\ Trans(0,0,d_i)\ Trans(a_i,0,0)\ Rot(x,\alpha_i) \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\
&= \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}
\end{aligned}
\tag{3.1}
$$

En esta matriz se utiliza la nomenclatura clásica en Robótica, *s* para designar al seno, *c* para el coseno y en subíndice el argumento de estas funciones trigonométricas. Cabe mencionar aquí que, cuando la *i*-ésima articulación es de rotación, θ_i es variable y d_i es constante; mientras que en el caso de articulaciones de translación es al contrario. Con las *n* matrices de paso generalizado se calcula la matriz homogénea que define la cinemática directa del robot de la siguiente manera:

$$
T_0^n(\mathbf{q}) = \prod_{i=1}^{n} A_i(q_i) \tag{3.2}
$$

La cinemática inversa de un robot, expresada por la ecuación $\mathrm{q} = \mathrm{f}^{-1}(\mathrm{x})$, nos permite calcular el valor de las coordenadas articulares del robot, $\mathrm{q}^T = (q_1, q_2, ..., q_n)$, que permiten llevar a su órgano terminal al punto deseado, expresado por sus seis coordenadas cartesianas, tres para representar la posición $t^T = (x, y, z)$, y tres más para representar la orientación. Existen muchos métodos para calcular las coordenadas articulares del robot a partir de la expresión de la cinemática directa dada por la ecuación (3.2). En la subsección correspondiente a la cinemática inversa del robot utilizado se presenta el método correspondiente adaptado a la cinemàtica del robot.

3.2.2. Cinemática Directa del Robot UNIMATE S-103

En la Figura 3.4 se muestra el esquema cinemático del robot UNIMATE S-103, donde aparece la asignación de marcos de referencia a cada uno de los eslabones del robot así como la ilustración de los parámetros D-H correspondientes. Cabe destacar que, en este caso, los orígenes de los marcos de referencia R_1 R_2 y R_3 han sido colocados de manera que anulan los parámetros d_2 y d_3. De manera complementaria, en la Tabla 3.1 se dan los valores numéricos de dichos parámetros a fin de calcular el modelo cinemático directo de este robot.

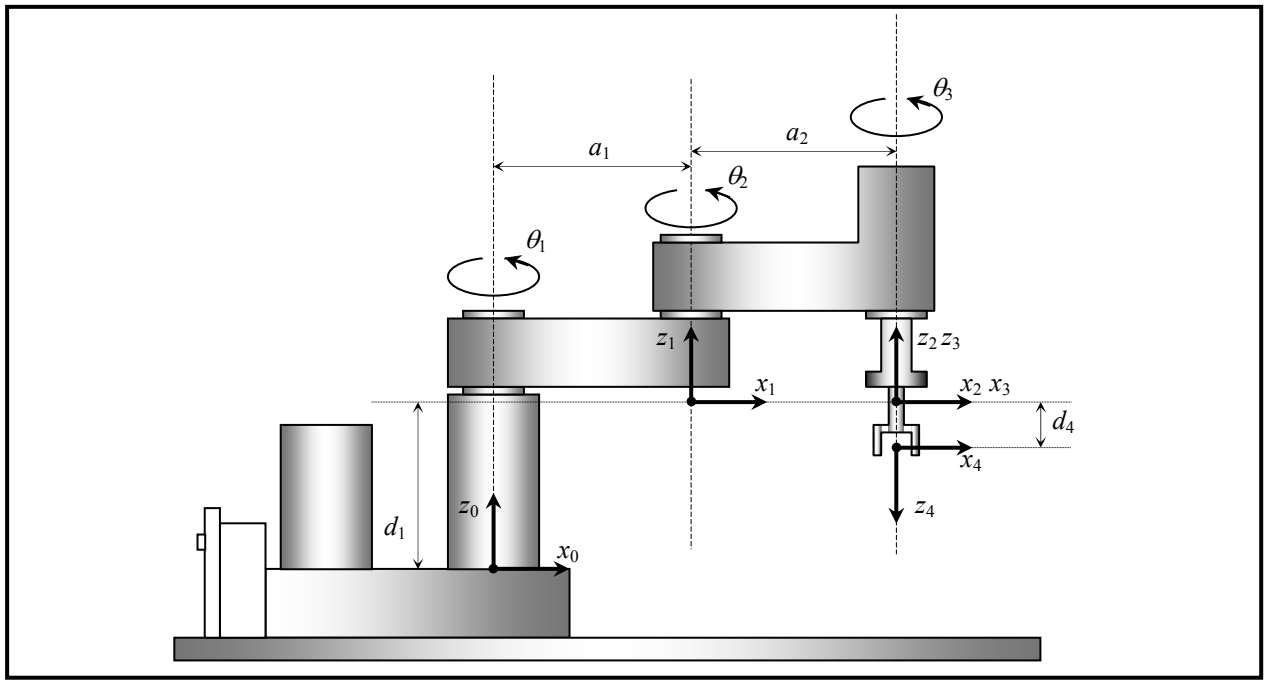

Figura 3.4. Asignación de los marcos de coordenadas para el robot UNIMATE S-103.

i	a_i	α_i	θ_i	d_i
1	a_1	0	θ_1	d_1
2	a_2	0	θ_2	0
3	0	0	θ_3	0
4	0	π	0	d_4

Tabla 3.1. Parámetros D-H del robot UNIMATE S-103

Sustituyendo en la ecuación 2.1 los valores de los parámetros D-H del robot UNIMATE S-103 dados por la Tabla 3.1, y efectuando las multiplicaciones matriciales indicadas por la

ecuación (3.2), se obtiene la matriz de transformación homogénea ${T_0}^4$ que representa la cinemática directa de dicho robot. Dicha matriz se escribe como sigue:

$$T_0^4 = \begin{bmatrix} c_{12}c_4 + s_{12}s_4 & -c_{12}s_4 + s_{12}c_4 & 0 & a_1c_1 + a_2c_{12} \\ s_{12}c_4 - c_{12}s_4 & -s_{12}s_4 - c_{12}c_4 & 0 & a_1s_1 + a_2s_{12} \\ 0 & 0 & -1 & d_1 - d_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

Considerando que toda matriz homogénea está formada por una matriz ortonormal R de rotación, de un vector de translación $t^T = (x, y, z)$, de un factor de escala, en este caso unitario $e = 1$ y de un vector de perspectiva, en este caso nulo $p^T = (0, 0, 0)$, la matriz ${T_0}^4$ se escribe de la siguiente manera:

$$T_0^4 = \begin{bmatrix} R & t \\ p^T & e \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & x \\ r_{21} & r_{22} & r_{23} & y \\ r_{31} & r_{32} & r_{33} & z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

en donde los elementos de la matriz de rotación y del vector de translación se definen como:

$$\begin{aligned} r_{11} &= c_{12}c_4 + s_{12}s_4 \\ r_{12} &= -c_{12}s_4 + s_{12}c_4 \\ r_{21} &= s_{12}c_4 - c_{12}s_4 \\ r_{22} &= -s_{12}s_4 - c_{12}c_4 \\ x &= a_1c_1 + a_2c_{12} \\ y &= a_1s_1 + a_2s_{12} \\ z &= d_1 - d_4 \end{aligned} \tag{3.5}$$

Entonces, aplicando la metodología Denavit-Hartenberg al robot UNIMATE S-103 se ha obtenido que la posición del OT con respecto a R_0 está dada por las siguientes expresiones:

$$\begin{aligned} x &= a_1 \cos\theta_1 + a_2 \cos(\theta_1 + \theta_2) \\ y &= a_1 sen\theta_1 + a_2 sen(\theta_1 + \theta_2) \\ z &= d_1 - d_4 \end{aligned} \tag{3.6}$$

Estas expresiones se obtienen del vector de translación de la transformación homogénea dada por la ecuación (3.4), cambiando la notación condensada por una notación estándar. Finalmente, la orientación de la pinza con respecto al marco de referencia fijo R_0 queda como sigue:

$$\theta = \theta_1 + \theta_2 + \theta_3 \tag{3.7}$$

pues las tres rotaciones se efectúan alrededor de ejes paralelos verticales. Además, esta orientación puede calcularse a partir de los elementos de la matriz de rotación incluida en la matriz que representa la cinemática del robot considerado dada por la ecuación (3.3). Dicha matriz equivale a una rotación elemental de θ grados alrededor del eje z, la cual se expresa como:

$$R = \begin{bmatrix} c_\theta & s_\theta & 0 \\ s_\theta & -c_\theta & 0 \\ 0 & 0 & -1 \end{bmatrix} \tag{3.8}$$

Entonces, conociendo los valores de esta matriz, es posible calcular el ángulo equivalente mediante la función arco tangente de dos argumentos, de la siguiente manera:

$$\frac{\cos\theta}{\sin\theta} = \tan\theta \tag{3.9}$$

Igualando la matriz de rotación dada por la ecuación (3.8) con la que se incluye en la cinemática directa del robot dada por la ecuación (3.3), se puede despejar el valor de la orientación del OT del robot, utilizando la expresión de la función arco tangente de dos argumentos, de la siguiente manera:

$$\theta = \tan^{-1}\frac{c_\theta}{s_\theta} = \tan^{-1}\frac{r_{11}}{r_{12}} = \tan^{-1}\frac{c_{12}c_4 + s_{12}s_4}{-c_{12}s_4 + s_{12}c_4} \tag{3.10}$$

Así, con las expresiones dadas por las ecuaciones (3.6) y (3.7), o bien la (3.10), se obtienen respectivamente la posición y la orientación del órgano terminal del robot como una función de sus variables articulares.

3.2.3. Cinemática Inversa del Robot UNIMATE S-103

En el caso del robot UNIMATE S-103 se puede aplicar un sencillo método de inversión dado que la orientación en el espacio cartesiano se expresa por una sola variable: θ. Para aplicar las leyes de la trigonometría y obtener la cinemática inversa se utilizará el esquema del robot representado en la Figura 3.5.

Proyectando la configuración del robot SCARA considerado sobre el plano *XY*, se puede calcular el valor de la segunda variable articular utilizando la siguiente ecuación:

$$\theta_2 = \tan^{-1}\left(\frac{r}{\sqrt{1-r^2}}\right) \tag{3.11}$$

en donde:

$$r^2 = \frac{x^2 + y^2 - a_1^2 - a_2^2}{2a_1 a_2} \tag{3.12}$$

Una vez calculada la variable articular θ_2 se procede al cálculo de la variable θ_1, utilizando para ello la siguiente ecuación:

$$\theta_1 = \tan^{-1}\left(\frac{y}{x}\right) - \tan^{-1}\left(\frac{a_2 s_2}{a_1 + a_2 c_2}\right) \tag{3.13}$$

Conocidas las dos primeras variables articulares, el valor de la tercera, θ_3, se obtiene fácilmente despejándola de la ecuación (3.7):

$$\theta_3 = \theta_1 + \theta_2 - \theta \tag{3.14}$$

Finalmente, la translación correspondiente a la cuarta variable articular, d_4, se obtiene despejándola de la ecuación (2.6):

$$d_4 = d_1 - z \tag{3.15}$$

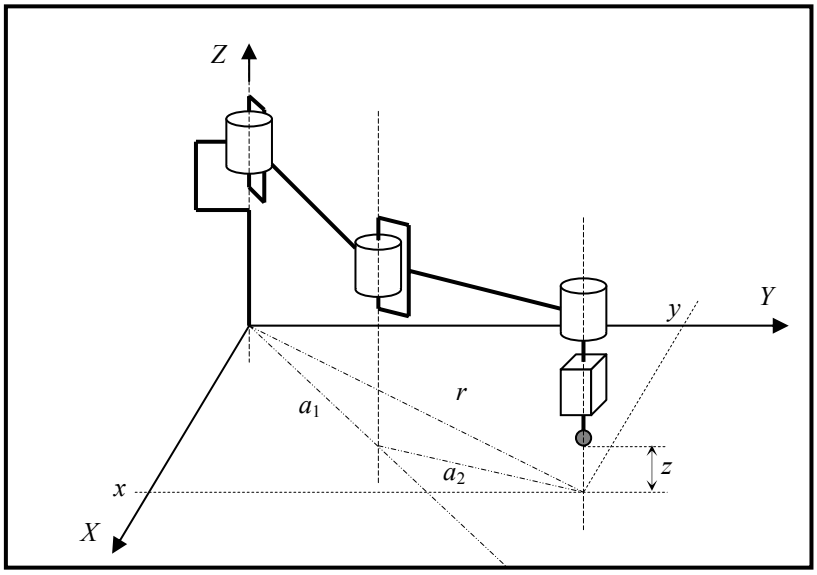

Figura 3.5. Diagrama cinemático del robot UNIMATE S-103.

De este modo tan sencillo, con las ecuaciones de la (3.11) a la (3.15) es posible calcular las cuatro variables articulares correspondientes a un punto deseado especificado como tres coordenadas cartesianas en el espacio de trabajo (x, y, z), más la orientación θ. Este conjunto de ecuaciones determina la cinemática inversa del robot UNIMATE S-103, manipulador de estructura SCARA utilizado en este proyecto.

3.2.4. Interacción con el Robot Virtual

Para comandar manualmente los movimientos de los objetos virtuales del robot es necesario dotar a cada elemento móvil de un sensor apropiado. Si se desea un movimiento rotacional variable cualquiera, se requerirá de un sensor cilíndrico (en VRML `CylinderSensor`), mientras que las translaciones arbitrarias se pueden activar usando sensores planos (en VRML `PlaneSensor`). Para obtener movimientos predefinidos de tipo binario bastará con el uso de sensores de tacto (`TouchSensor`).

Para representar al robot UNIMATE S-103 se construyeron sus cinco elementos móviles y la plataforma estática que los soporta. Estos cinco elementos son: tres eslabones rotacionales y los dos dedos de la pinza. En la Figura 3.2, los tres eslabones tienen movimientos rotacionales representados por las coordenadas articulares q_1, q_2 y q_3, de modo que los valores de estas coordenadas definen la posición en la que se encuentra el robot. Estos valores se pueden modificar desde la Interfaz de Usuario, o bien, dentro del mismo mundo virtual mediante sensores. Los movimiento7s de los tres eslabones son rotacionales, activados mediante sendos sensores cilíndricos; sin embargo el tercer eslabón posee además un movimiento binario de translación a lo largo del eje vertical, activado por un sensor de tacto generando los movimientos de subir y bajar. Finalmente, los dedos de las pinzas pueden abrir y cerrar gracias a otro sensor de tacto. Para que la animación de los movimientos traslacionales de la pinza pueda realizarse con éxito, es necesario utilizar interpoladores de posición (en VRML `PositionInterpolator`).

3.2.5. Actualización del Espacio Virtual

El mundo virtual contiene las máquinas y elementos fijos (móviles o estáticos) que constituyen el puesto de trabajo del robot real. Sin embargo, objetos que han de ser manipulados por el robot constituyen una variable que debe ser incluida en la representación virtual de dicho

puesto de trabajo. Para ello el sistema de visión artificial debe procesar las imágenes de los objetos a manipular, generar un modelo geométrico VRML de cada uno, e incluir cada modelo en el mundo virtual [Araujo 02]. Una vez incluidos, cada objeto es susceptible de ser manipulado manual o automáticamente por el robot virtual, por lo cual deben estar asociados a ellos los sensores adecuados.

Pero algunas aplicaciones no requieren la actualización del mundo virtual con información visual del puesto de trabajo real. Pruebas de manipulación virtual, por ejemplo, pueden utilizar un conjunto de objetos con formas y posiciones generadas aleatoriamente. Estos objetos pueden ser cubos, prismas rectangulares, prismas con base triangular o cilindros, todos de la misma altura, tal como se muestra en la Figura 3.6. El Laboratorio Virtual de Robótica también puede generar un objeto prisma rectangular a la vez en el sitio en que manualmente se ha colocado la pinza del robot virtual. En todos los casos, los objetos virtuales por manipular se crean con un sensor de tacto para que el usuario pueda seleccionarlos usando el ratón de máquina para manipularlos de acuerdo con el programa deseado.

Figura 3.6. Objetos en el espacio virtual

3.2.6. Manipulación de objetos por el Robot Virtual

Es posible manipular cualquier objeto directamente dentro del mundo virtual, siempre y cuando se le hayan asociado los sensores correspondientes, los cuales generan eventos binarios.

Para mover un objeto de una posición a otra se debe tomar con las pinzas del robot y mover alguno(s) de los eslabones del mismo. Para modificar los valores deposición de los eslabones, es necesario que se haga un arrastre con el ratón al momento de pulsar sobre alguno de

los sensores cilíndricos asociados a los tres eslabones. Cuando se desea subir o bajar la pinza o bien abrir o cerrar los dedos, basta con pulsar una vez con el ratón sobre el sensor de tacto asociado a la pinza, para lograr la translación vertical, o al de los dedos para conseguir abrirlos o cerrarlos. Con esto se activará la animación binaria correspondiente.

3.3 Interfaz de Usuario

El módulo Interfaz de Usuario permite la manipulación del robot virtual, actualizar el mundo virtual y emular el sistema de programación del robot de acuerdo con las características computacionales del robot UNIMATE S-103. La Figura 3.7 muestra la Interfaz de Usuario, la cual se compone de los siguientes submódulos de operación: Manipulación de las coordenadas articulares del robot, Manipulación de las coordenadas cartesianas del robot, Actualización del mundo virtual, Enseñanza y Programación del Robot y Autoprogramación.

3.3.1. Manipulación del Robot Virtual en Coordenadas Articulares

La Interfaz de Usuario permite modificar los valores de las coordenadas articulares para obtener una nueva posición del órgano terminal del robot virtual y su representación gráfica dentro del mundo virtual. Esta interfaz despliega el valor actual de las coordenadas articulares de acuerdo con el método de Denavit-Hartenberg (q_1, q_2 y q_3) así como los valores correspondientes medidos por los sensores del robot representados en grados (θ_1, θ_2 y θ_3), identificados en dicha interfaz por las variables t_1, t_2 y t_3. La diferencia entre ambos ángulos es que q es el valor del ángulo con respecto al un eje de referencia (x), y θ es el valor del ángulo con respecto al eslabón anterior.

Una vez modificada la posición del robot con cualquiera de estas opciones, todas las demás se actualizan automáticamente haciendo uso de los Modelos Matemáticos.

En este módulo se tienen también las opciones correspondientes al manejo de la pinza del robot: abrir/cerrar, o subir/bajar. Estos movimientos se comandan a través de dos botones que envían información al mundo virtual para que se activen los sensores correspondientes. Estos botones aparecen con una etiqueta variable que indica la acción que pueden ejecutar si se pulsan con el ratón (Up/Down-Close/Open). Una vez pulsados, estos botones cambian de etiqueta

quedando listos para activar la función complementaria. Se tiene, además, la opción de enviar al robot virtual a su posición inicial (*home*), lo cual en muchas ocasiones es requisito indispensable.

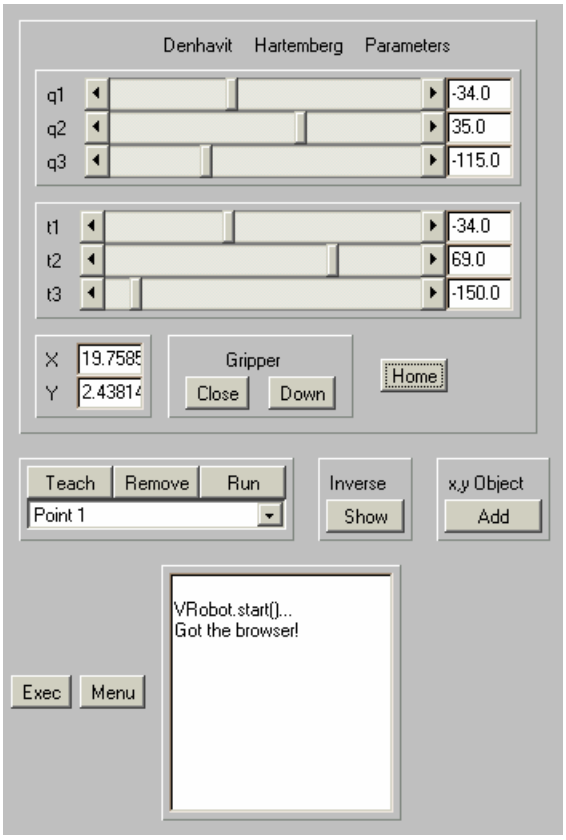

Figura 3.7. Interfaz de aplicación del sistema.

3.3.2. Manipulación del Robot Virtual en Coordenadas Operacionales

El usuario puede también especificar la nueva posición del órgano terminal del robot virtual mediante las coordenadas operacionales, lo cual se logra haciendo uso de la Cinemática Inversa del robot calculada en el módulo de Modelos Matemáticos. Esto se hace escribiendo directamente los valores cartesianos (x, y) deseados en el campo correspondiente o bien haciendo uso de una interfaz gráfica especialmente concebida para ello.

En este último caso, es necesario abrir la interfaz gráfica de esta opción, pulsando el botón *Show* del modulo titulado *Inverse*, con lo cual aparece una interfaz gráfica mostrando la estructura mecánica simplificada del robot, dibujada en planta sobre un círculo representando el espacio de trabajo total del robot, tal como se muestra en la Figura 3.8. En esta interfaz gráfica

basta con pulsar el ratón de la computadora en alguna parte del círculo que se presenta para que el robot virtual se posicione en el punto deseado, siempre y cuando el punto elegido no viole las restricciones cinemáticas especificadas para el robot. Una vez que el robot virtual ha sido enviado a alguna posición deseada, todos los valores de variables de posición se actualizan automáticamente, incluyendo el despliegue gráfico en el mundo virtual.

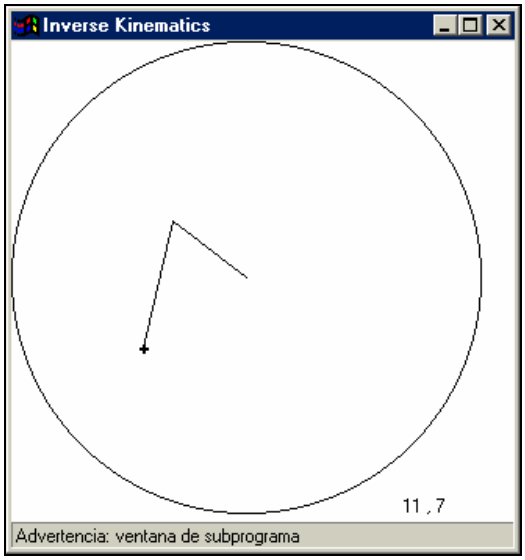

Figura 3.8. Interfaz de manipulación con cinemática inversa.

3.3.3. Actualización del Mundo Virtual

Para actualizar el mundo virtual, la Interfaz de Usuario ofrece 3 opciones: i) con los objetos que se encuentran en el espacio de trabajo real; ii) con un conjunto de objetos en cantidad, forma y ubicación aleatoriamente seleccionadas; y iii) con un solo objeto a la vez, cuya geometría ha sido previamente determinada (un cubo), en la posición donde se encuentre el órgano terminal del robot. Para activar las 2 primeras opciones se debe entrar al *menú* de opciones pulsando con el botón del *mouse* el botón correspondiente (ver Figura 3.7). Una vez hecho esto aparece una ventana con las opciones *Program* y *Object*. Pulsando con el ratón sobre esta última se ofrecen tres opciones: *from workplace*, *random* y *clear*. La primera de ellas trae desde el servidor *e–*vision los modelos 3D de los objetos reales presentes en el puesto de trabajo, la segunda opción genera varios objetos aleatoriamente. En la Figura 3.6 se muestra un ejemplo de espacio de trabajo virtual con objetos creados aleatoriamente. La tercera opción elimina todos

los objetos virtuales presentes en el espacio de trabajo del robot virtual. La Figura 3.9.a muestra el menú de objetos correspondiente.

Figura 3.9. a) Menú de objetos y b) Menú de programas disponibles en el mundo virtual.

La tercera opción para actualizar el mundo virtual se activa pulsando con el *mouse* en la opción *x*,*y Object Add*. Todos los objetos que se añaden al mundo virtual están equipados con un sensor de tacto para que, al ser tocados, se pueda desencadenar la acción de tomar dicho objeto con la pinza del robot virtual.

3.3.4. Enseñanza y Programación del Robot

La Interfaz de Usuario también permite que el robot virtual emule los módulos de operación básicos del robot UNIMATE S-103: el modo *teach* (enseñanza de puntos), el modo program (programación) y el modo *run* (ejecución de programas) del sistema operativo C/ROS de dicho robot. En la fase de enseñanza el robot *aprende* (registra) los valores de las coordenadas articulares del punto en el que se encuentra el órgano terminal del robot. En la fase de programación se escribe el conjunto de instrucciones de movimiento que determinan la tarea a realizar. En la fase de ejecución dicho programa es ejecutado.

En la Interfaz de Usuario (ver Figura 3.7), el botón *Teach* es pulsado cuando se desea que se almacene el punto donde se encuentra el robot actualmente, con lo cual se añade automáticamente un nuevo renglón al campo de texto combinado etiquetado como Point *n*, donde *n* indica el orden del punto almacenado. Cuando se desea eliminar algún punto previamente enseñado, se elige dicho punto en el campo de texto y se pulsa el botón *Remove*, quedando eliminado de la memoria, así como del campo de texto. Si se desea que el robot virtual recorra todos los puntos que previamente se le han enseñado, simplemente se pulsa el botón *Run* y

entonces el robot virtual comenzará a hacer una animación recorriendo estos puntos, a la vez que en el campo de texto se despliega el punto por el cual está pasando el robot. Al seleccionar algún punto del campo de texto combinado, el robot virtual se posicionará inmediatamente en ese punto.

Para realizar un programa de manipulación de objetos se comienza por tocar el primer objeto con el puntero del ratón, luego se lleva el robot al punto destino del objeto tocado (usando la opción de manipulación en coordenadas articulares u operacionales), se registra su posición pulsando el botón *teach*, se baja la pinza y se abren sus dedos. La operación se repite para todos los objetos por manipular y se obtiene automáticamente un programa en C/ROS ejecutable en el mundo virtual, el cual puede también ser enviado al laboratorio real para su ejecución en el UNIMATE S-103. Para esto último, desde el menú de la Interfaz de Usuario se pulsa la opción *send* ofrecida en la opción *Program* (Figura 3.9b). La opción *send* es una alternativa más de comunicación que permite la transmisión de un programa al laboratorio real para ser ejecutado por el robot real.

La opción *New* del menú en la parte relativa a programas, mostrado por la Figura 3.9.b, sirve para empezar a hacer un nuevo programa. En caso de que exista en memoria otro programa previamente realizado, todas y cada una de las líneas de programa se sustituyen con el comando NP (no operación). La memoria se limpiará y quedará lista para ingresar un programa nuevo. En caso de empezar a hacer un nuevo programa y no haber seleccionado la opción *New*, el nuevo programa se añadirá al final del programa residente, pues, tal como es característico en C/ROS, el sistema sólo puede tener un programa residente a la vez. En este caso, el nuevo programa residente estará formado por el programa anterior más el nuevo.

El botón *Exec* ordena que se ejecute en el robot virtual un programa compatible con C/ROS que se haya generado previamente de la manera descrita. Esta programación es del tipo conocido como gestual (tocando y soltando objetos, llevando al robot *de la mano*). Cuando un programa está en ejecución, no es posible pulsar ningún botón de la interfaz de aplicación, pues el robot no podría ejecutar correctamente el programa.

3.4 Autoprogramación del Robot

A manera de demostración de las posibles aplicaciones que se pueden desarrollar con el Laboratorio Virtual de Robótica, se agregó la opción de autoprogramación. Esta opción consiste en la generación automática de programas de manipulación. La aplicación que se tiene es la clasificación de objetos, la cual se selecciona en la opción *Program* del menú eligiendo *demo*, tal como se muestra en la Figura 3.9.b.

En esta aplicación, el sistema genera automáticamente una cantidad aleatoria de objetos escogidos entre cuatro diferentes clases (cubos, cilindros, prismas rectangulares y prismas de base triangular). La posición de cada uno de ellos así como su cantidad son aleatorias. Una vez generados los objetos, el robot debe clasificarlos de manera automática ordenándolos en filas homogéneas, una para los cubos, otra para los cilindros y así sucesivamente. Para ello se desarrolló un módulo de autoprogramación que genera el programa en lenguaje C/ROS que resuelva el problema de clasificación automática. Mediante este módulo de autoprogramación, denominado SORT, el robot se encarga de hacer una selección de los objetos que aparecieron aleatoriamente en su espacio de trabajo, usando un algoritmo de ordenamiento conocido como *ordenamiento por cubetas (Bucket Sort)* [Cormen 90].

3.5 Teleoperación vía Internet

En muchas aplicaciones didácticas de Telerrobótica o de *Internet Robotics* no es necesario contar en el puesto de trabajo remoto con todas las facilidades del laboratorio de Robótica, es decir, con todo el Laboratorio Virtual de Robótica [Chellali 01, Taylor 97]. A menudo sólo interesa tener acceso remoto al sistema de programación del robot industrial, del mismo modo que si se estuviera frente al robot real. Para resolver esta situación se desarrolló un sistema alterno de teleoperación vía Internet del robot UNIMATE S-103 que permite la operación remota del mismo. Este sistema pone a disposición del usuario remoto una versión del *teach pendant* del robot con todas las funciones que éste ofrece [Ibarra 01, Zaldívar 01].

El *teach pendant* desarrollado en esta tesis permite sustituir el *teach pendant* hardware con un programa corriendo en el *e*–server que se encuentra conectado al controlador del robot

mediante un puerto serie. Esta aplicación es accesible por Internet, para lo cual se desarrolló un software de comunicación que permite accesar el *teach pendant* virtual desde la página web de teleoperación. La Figura 3.10 muestra un esquema simplificado de la estructura del software desarrollado. La Figura 3.11 muestra el aspecto de la página web correspondiente al sistema de Teleoperación del robot UNIMATE S-103. En esta página se cuenta con dos secciones: la interfaz gráfica del *teach pendant* remoto y la imagen del Laboratorio de Robótica proporcionada por una *webcam*.

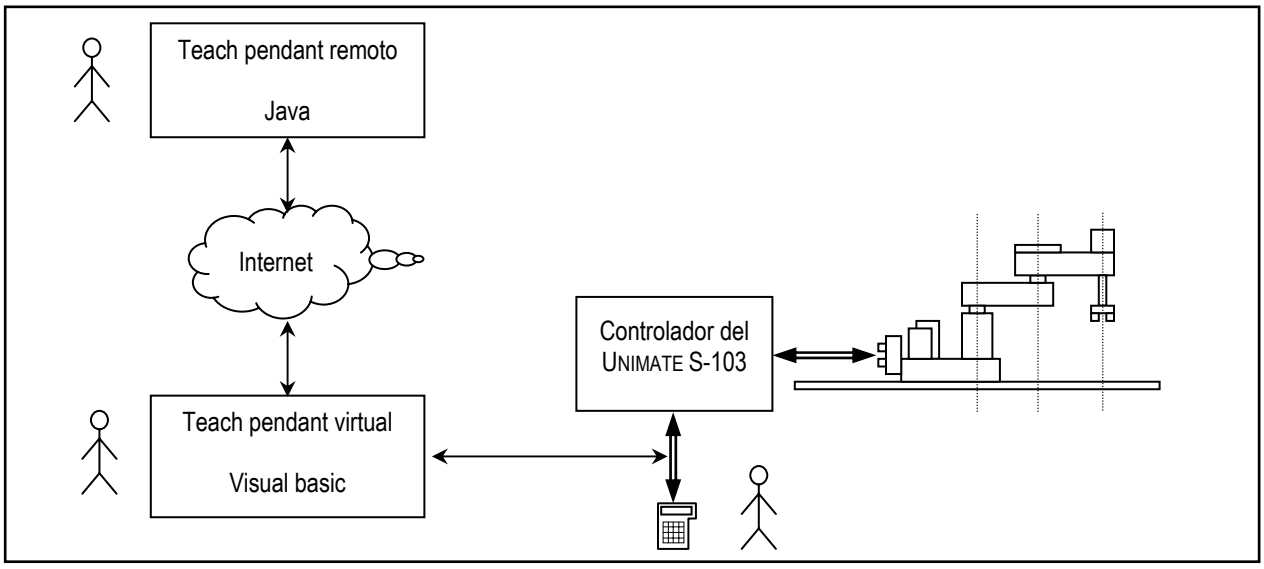

Figura 3.10. Esquema del software del Laboratorio Virtual de Robótica usado en tareas de teleoperación.

El *teach pendant* virtual fue desarrollado por Rodrigo Murguía Alcalá, el cual permite la comunicación vía puerto serie con el robot sustituyendo al *teach pendant hardware*. El autor se encargó de modificar este programa para ponerlo accesible por Internet mediante *sockets*. El autor también desarrolló completamente el *teach pendant remoto* para acceder *al teach pendant virtual* y configuró el monitoreo visual remoto.

3.5.1. Teach Pendant Remoto

Al entrar a la página del Laboratorio Virtual de Robótica (http://rovir.ctrl.cinvestav.mx/) se tiene la opción de elegir la teleoperación del robot UNIMATE S-103, y así recibir la interfaz gráfica correspondiente, que ha sido denominada *teach pendant* remoto. El módulo informático

correspondiente al *teach pendant* remoto está desarrollado en Java y se basa en una interfaz gráfica muy similar al *teach pendant* real, o a su versión virtual (descrita a continuación). Por cada botón pulsado con ayuda del ratón en el *teach pendant* remoto, el *socket* Java asociado transmite un código a través de Internet hacia el *teach pendant* virtual, quien, al recibir el dato, lo interpreta llamando a la rutina correspondiente al botón pulsado.

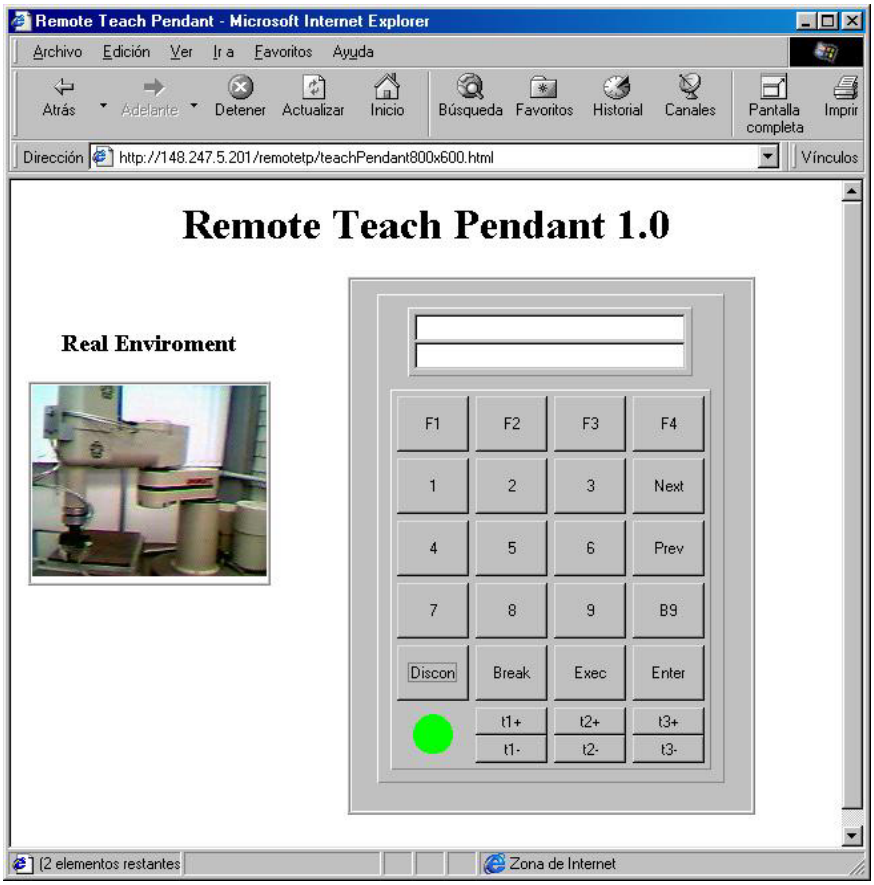

Figura 3.11. Página WEB de Teleoperación con el *teach pendant* remoto

3.5.2. Teach Pendant Virtual

El *teach pendant* virtual [Herrera 00, Ibarra 00.1, Ibarra 00.2], recibe un dato proveniente del botón pulsado en el *teach pendant* remoto y, para interpretarlo, llama a la rutina asignada a dicho botón para enviar el dato correspondiente al controlador del robot real. El *teach pendant* virtual no toma en cuenta restricciones cinemáticas, así que al recibir un dato lo interpreta y lo manda directamente al controlador del robot sin verificar si el movimiento producido puede generar alguna interferencia cinemática entre los elementos móviles del robot.

Por su parte, el controlador del robot recibe los datos que han sido enviados desde el *teach pendant* virtual, y los interpreta de igual manera que si se estuviera trabajando con el *teach pendant* real, asi que todo este sistema permite operar remotamente al robot desde cualquier lugar donde se haya accedido a la página web que contiene el *teach pendant* remoto.

3.5.3. Monitoreo Visual Remoto

En ambas alternativas de conexión (teleoperación y Laboratorio Virtual de Robótica) se usa una *webcam* para monitorear el laboratorio real, la cual se encuentra conectada al servidor *rovir* que atiende a los usuarios remotos (ver Figura 2.9). En esta parte de teleoperación, desde que se accede a la página, el servidor envía constantemente imágenes al usuario remoto, quien las recibe a través de un *applet* de Java y que utiliza el programa *WebCam32*, de uso comercial, que gestiona la comunicación entre la cámara y el usuario.

La rapidez de actualización de las imágenes del laboratorio depende del ancho de banda de comunicación del usuario remoto. El sistema trata de enviar dos imágenes (en formato *JPEG*) por segundo, cuyo tamaño promedio es de 3 kilobytes. Por lo tanto si se quiere observar el monitoreo en “tiempo real” es necesario que el usuario remoto cuente al menos con un ancho de banda aproximado de 48 kbps (2 x 3kb x 8bits). Los modems telefónicos actuales trabajan a una velocidad de 56kbps, por lo tanto no se necesita de equipo especial para contar con este ancho de banda.

Capítulo 4
Diseño e Implementación del Robot Virtual

En el capítulo anterior se mostró el diseño del Laboratorio Virtual de Robótica y se describieron los módulos que lo constituyen así como la interacción entre los mismos. En este capítulo y el siguiente, se presenta la implementación de los módulos correspondientes al mundo virtual, particularmente el Robot Virtual, a la Interfaz de Usuario. Como se mencionó, la apariencia física del robot fue desarrollada por Luis Manuel Sastré Leyva en 1997, y esta versión modificada un poco por José Andrés Herrera Barragán en el verano del 2000, sin embargo, el autor tuvo que modificar esta ultima versión para que cumpliera con las restricciones cinemáticas correspondientes, además de la creación de algunas funciones para realizar una mejor animación.

Se ha creado una aplicación que permite representar los espacios de trabajo del robot real y ambientarlo con los accesorios y detalles que lo rodean, para que un usuario pueda percibir o sentir que se encuentra en el laboratorio real.

4.1. Introducción

Para crear un mundo virtual, primero deben definirse cuáles son los elementos que conforman el mundo real correspondiente para llevarlos a la computadora en una representación lo más realista posible, tanto desde el punto de vista gráfico como funcional. La representación de la forma, es decir, de la parte geométrica de los objetos por representar, no es complicada, basta con aplicar directamente las reglas del lenguaje utilizado, en este caso VRML. La interacción del operador con la forma de estos objetos tampoco resulta problemática, pues VRML proporciona

diferentes tipos de sensores lógicos que permiten al operador hacer la manipulación en pantalla de dichos objetos.

Donde existe dificultad es al momento de representar la funcionalidad de los objetos. Por lo que concierne al mundo virtual que representa el Laboratorio de Robótica, la funcionalidad consiste en las propiedades cinemáticas de las partes móviles de los elementos constitutivos de dicho laboratorio y, sobretodo, en los aspectos computacionales de los elementos programables del puesto de trabajo. Estas características se muestran en toda su importancia, por ejemplo, cuando se trata de implementar la interacción entre el mundo virtual y el usuario, bajo la opción de Programación del Laboratorio Virtual, y al hacer funcionar los programas de teleoperación vía Internet.

Cada eslabón del robot virtual se diseñó como un conjunto de formas elementales debidamente agrupadas y poseedor de los sensores que materialicen mejor el tipo de articulación que lo relaciona con el resto de la mecánica. Al mismo tiempo, en función de la manera en que se comanda el movimiento de cada articulación, se hace uso de la cinemática directa del mecanismo o de la cinemática inversa.

Cuando el movimiento del robot es controlado por su cinemática inversa o por la directa, en base a las ecuaciones presentadas en las secciones 3.1.2 y 3.1.3 de esta tesis, se debe asegurar que el programa cumple con todas las restricciones geométricas presentes en el espacio de trabajo real. La principal restricción tiene que ver con la prohibición de moverse hacia los espacios ocupados por elementos del propio robot. En efecto, como el robot real no tiene manera de verificar si las posiciones consideradas en un programa producen o no problemas de colisión consigo mismo, se desarrolló un sistema de cálculo cinemático que impide las colisiones del órgano terminal del robot con otros elementos del puesto de trabajo y con partes del mismo robot. Además, se tienen programas en Java que calculan la cinemática inversa para obtener las coordenadas articulares del robot, que evitan colisiones con los objetos presentes en el espacio de trabajo, cuyas posiciones han sido definidas en el espacio de la tarea.

4.2. Robot Virtual

En esta sección se presenta el desarrollo del modelado de la parte geométrica del robot industrial UNIMATE S-103, el cual consta de seis partes esenciales: plataforma de trabajo, cuatro eslabones mecánicos, uno de los cuales es fijo y el órgano terminal del manipulador (pinza bidigital). Los cuatro eslabones mecánicos del robot manipulador considerado están unidos en serie mediante tres articulaciones de rotación; además, el cuarto eslabón está unido al tercero por una articulación de translación, tal como aparece en el esquema cinemático de la Figura 2.3.

Para crear los objetos virtuales que componen el robot primero fue necesario adoptar una unidad de medida. Comúnmente se considera que la unidad VRML es el metro, pero en el caso de este trabajo, se consideró cada unidad de VRML como un centímetro; entonces, las medidas de todas las piezas dibujadas se expresaron en centímetros. Después, se genera una escena gráfica jerárquica que contempla el anidamiento de los elementos mecánicos que constituyen el robot, de modo que se garantice la herencia de algunas propiedades y para asegurar que el movimiento de cada uno de los elementos móviles repercuta en los cambios de posición o de orientación de los demás de acuerdo con la cinemática del mecanismo. En el Programa 4.1 se muestra un esquema de dicha jerarquía para comprender mejor el anidamiento mencionado.

El siguiente programa muestra algunos de los nodos que conforman el brazo del robot, en donde se puede observar que el nodo principal (Brazo) es de tipo `Transform`, lo cual indica que el nodo brazo aceptará transformaciones geométricas (rotación, traslación y escala), y esto afectará a todos los nodos anidados dentro de el. El campo `children` indica que dentro del nodo principal es posible crear uno o mas nodos. El siguiente nodo creado es de tipo `Shape`, lo cual permite crear alguna figura geométrica dentro del nodo por medio del campo `geometry`, además de permitir asignarle una apariencia (color, textura) a esta figura a través del campo `appearance`. Se puede notar que al campo `geometry`, se le asigna un nodo de tipo `Cylinder`, el cual permite crear un cilindro de las dimensiones especificadas pos los campos `radius` y `height`. Al campo `appearance` se le asigna un nodo previamente definido, el cual es de tipo `Appearance`, y fue definido como `ColorRobot`.

A continuación se presenta el desarrollo de todas las piezas mecánicas que constituyen el robot virtual agrupadas por elementos.

```
DEF Brazo Transform{
      translation 48.7 42.2 0
      children  [
         Shape {
            geometry Cylinder  {
               radius 6.5
               height 10.9
            }
            appearance USE ColorRobot
         }
         #Aquí van los nodos restantes que conforman
         #la geometría del brazo
         DEF Antebrazo Transform{
            translation 25.65 12.4 0
            children[
                #Aquí van todos los nodos que
                # conforman la geometría del antebrazo
                DEF Mano Transform {
                    translation 35.55 -10.2 0
                    children[
                        #Aquí van todos los nodos que
                        # conforman la geometría de la
                        # mano y de la pinza
                    ]
                }
            ]
         }
      ]
}
```

Programa 4.1. Anidamiento de los nodos correspondientes a las articulaciones del robot.

Plataforma de Trabajo

Se trata de una plataforma circular modelada como un cilindro con un radio de 1.4 metros y 0.05 metros de altura. Esta pieza está construida con acero y tiene un acabado natural, por lo que se eligió un color gris para su versión virtual formado por la combinación RGB (204 204 204). En el Programa 4.2 aparece el código correspondiente a este modelo, mientras que en la Figura 4.1 se muestra el aspecto de este objeto virtual. En dicha figura se observa el grado de aproximación con que se dibuja un cilindro usando el comando (VRML) Cylinder, el cual aparece como un prisma cuya base es un polígono regular de 16 lados. Dicha resolución puede mejorarse utilizando los comandos de figuras libres definidas por puntos.

```
DEF Plataforma Transform{
    translation 43.7 -57.1 0
    children  [ Shape {
            geometry Cylinder  {
                radius 70
                height 5
            }
            appearance USE ColorMetal
        }
    ]
}
```

Programa 4.2. Definición de la Plataforma de Trabajo virtual.

Figura 4.1. Plataforma de trabajo modelada con VRML.

Eslabón Fijo

El eslabón fijo del robot fue desarrollado utilizando cajas (Box) y cilindros (Cylinder), nodos primitivos de VRML que, combinados, permiten crear objetos y figuras más complejas. En la Figura 4.2 se pueden observar todos los objetos que componen el eslabón fijo (9 cilindros y 6 cajas), los cuales le dan la forma adecuada. El Programa 4.3 muestra la definición del eslabón fijo usando el lenguaje VRML.

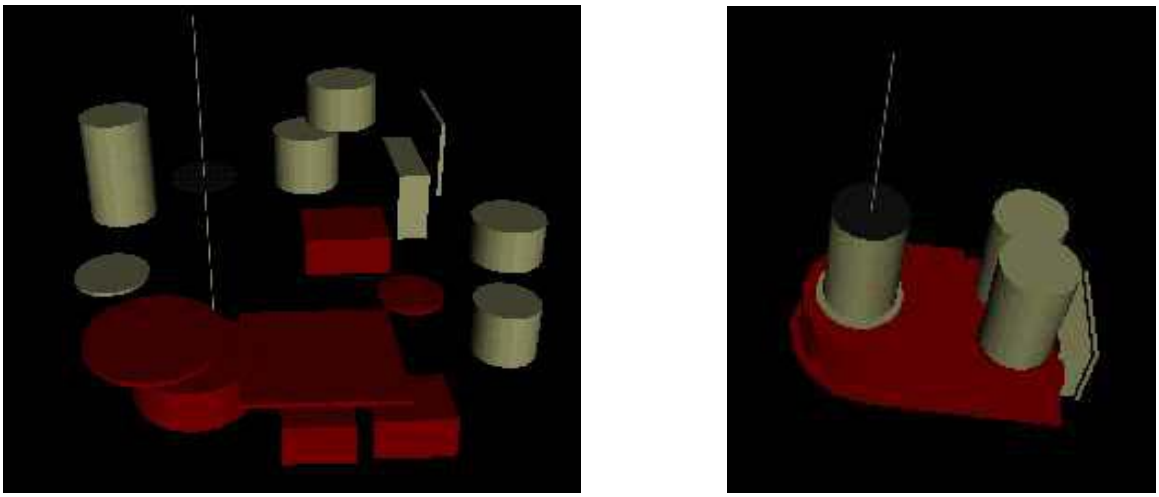

Figura 4.2. Eslabón fijo del UNIMATE modelado con VRML: a) en piezas, b) ensamblado.

```
DEF EslabónFijo Transform{
    translation 0 -54.6 0
    children  [
        Transform {
            translation 43.9 0.75 0
            children  [
                Shape {
                    geometry Cylinder  {
                        radius 20.3
                        height 1.5
                    }
                    appearance USE ColorCafe
                }
            ]
        }
         #Aquí van los nodos restantes que conforman
         #la geometría del eslabón fijo
    ]
}
```

Programa 4.3. Definición del eslabón fijo del robot virtual.

4.2.1. Eslabones Móviles

En el eslabón fijo se inserta el brazo, primer eslabón móvil del robot. En el Programa 4.4 se muestra el código VRML que permite crear el brazo. Al añadir el brazo a la base, el resultado obtenido en el mundo virtual se presenta en la Figura 4.3. De igual manera que el brazo quedó anidado a la base, el siguiente eslabón móvil (antebrazo) se anidó a éste. El antebrazo es la figura más compleja del robot, está formado por más de 10 cilindros y más de una docena de cajas. La Figura 4.4 muestra el antebrazo montado en los elementos anteriores del robot (plataforma,

eslabón fijo y brazo), mientras que el Programa 4.5 muestra su código VRML. Finalmente, la pinza se anidó al tercer y último eslabón móvil. El tercer eslabón móvil es una columna cilíndrica con una caja rectangular en su extremo inferior, cuyo código se muestra en el Programa 4.6, tal como se muestra en la Figura 4.5, donde aparece junto con la pinza. Por su parte, la pinza se modeló como una caja negra sobre la cual se deslizan dos dedos paralelos grises formados por tres cajas cada uno de ellos, tal como se muestra en el Programa 4.7.

```
DEF Brazo Transform{
    translation 48.7 42.2 0
    children [
       Transform {
           translation 0 .75 0
           children  [
               Shape {
                  geometry Cylinder  {
                     radius 6.5
                     height 1.5
                  }
                  appearance USE ColorNegro
               }
           ]
       }
       #Aquí van los nodos restantes que conforman
       #la geometría del primer eslabón móvil
    ]
}
```

Programa 4.4. Definición del brazo (primer eslabón móvil) del robot virtual.

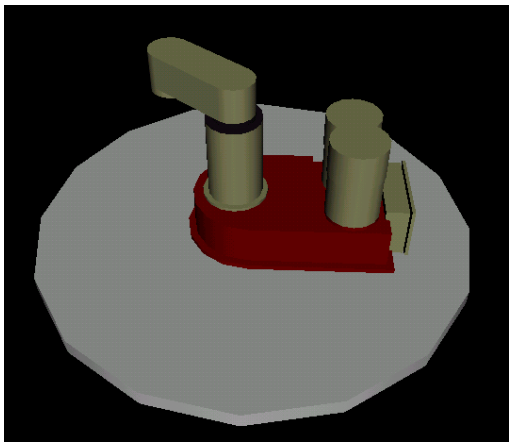

Figura 4.3. Brazo del robot virtual representado sobre el eslabón fijo.

```
DEF Antebrazo Transform{
    translation 25.65 12.4 0
    children[
       Transform {
          translation 0 .5 0
          children  [
             Shape {
                geometry Cylinder  {
                    radius 3.85
                    height 1
                }
                appearance USE ColorNegro
             }
          ]
       }
       #Aquí van los nodos restantes que conforman
       #la geometría del segundo eslabón móvil
    ]
}
```

Programa 4.5. Definición del segundo eslabón móvil del robot virtual.

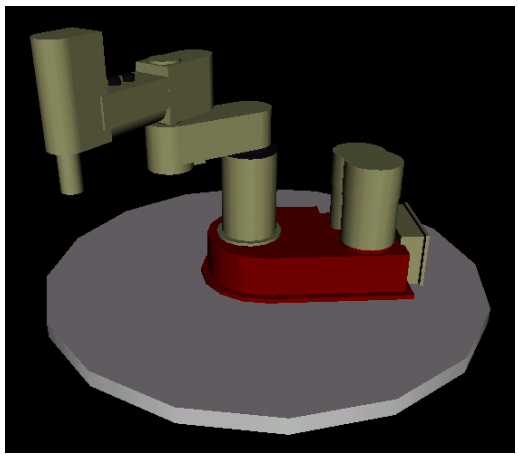

Figura 4.4. Plataforma, eslabón fijo, brazo y antebrazo del robot virtual.

En el siguiente programa se puede observar que existen traslaciones anidadas, para la creación de la mano (pinza) del robot, desde lo que se puede considerar como "muñeca" hasta las extremidades (dedos). Se puede observar también, que existe un nodo definido como `GiraP` y es de tipo `CylinderSensor`. Este nodo permite que la pinza sea sensible al ratón (*mouse*) y se activa al hacer un clic. Utilizando el código necesario, es posible que se gire la pinza sobre su eje por medio de este sensor. El nodo además, cuenta con la posibilidad de establecer restricciones para limitar el ángulo del giro. Estas restricciones se establecen por medio de los campos `minAngle` y `maxAngle`, los cuales, en este caso permiten hacer un giro de 360° de la pinza, ya que este giro va desde $-\pi$ hasta π. En el caso de omitir los valores de estos campos, no se

restringe el giro, de tal manera, que se puede realizar una rotación infinita hacia cualquier dirección.

```
DEF Mano Transform{
    translation 35.55 -10.2 0
    children[
        Transform {
            translation 0 1 0
            children  [
                Shape {
                    geometry Cylinder  {
                        radius 3.2
                        height 1.4
                    }
                    appearance USE ColorMetal
                }
                DEF GiraP CylinderSensor{
                   minAngle -3.1415926 maxAngle 3.1415926
                }
            ]
        }
        #Aquí van los nodos restantes que conforman
        #la geometría del tercer eslabón móvil
    ]
}
```

Programa 4.6. Definición del tercer eslabón móvil del robot virtual.

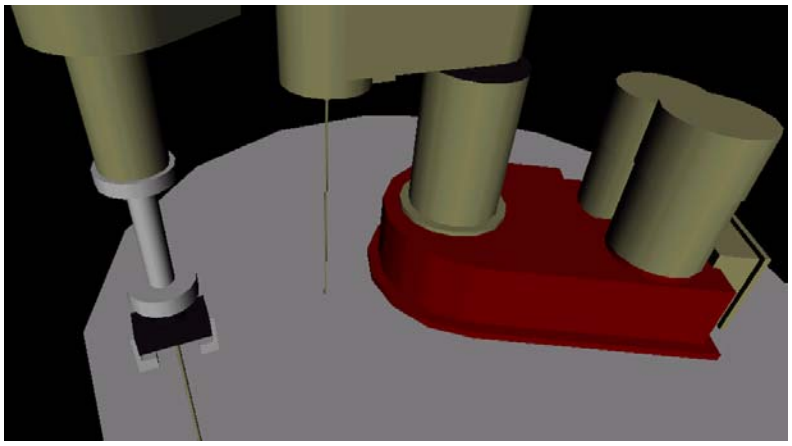

Figura 4.5. Mano del robot virtual y pinza bidigital.

En el código correspondiente a la pinza bidigital (dedos), el cual se presenta en el siguiente programa, se puede observar un nodo definido como `TakeObject`. Este nodo vacío, es el que recibe los objetos que son creados a través de ECI desde la aplicación externa al mundo virtual.

```
DEF PinzaBidigital Transform{
   children[
      Transform {
         translation 0 5.65 0
         children  [
            Shape {
               geometry Cylinder  {
                  radius 1.2
                  height 11.3
               }
               appearance USE ColorMetal
            }
            DEF TakeObject Transform {
               translation 0 -16 0
            }
         ]
      }
      #Aquí van los nodos restantes que conforman
      #la geometría de la pinza bidigital
    ]
}
```

Programa 4.7. Definición de la pinza bidigital del robot manipulador virtual.

Entorno

Todo el modelo virtual del robot se construyó utilizando cajas y cilindros de diferentes dimensiones hasta lograr una representación que se asemeja muchísimo al robot real; para comprobarlo basta con compararlo con la foto del robot que aparece en la Figura 2.1. Para aumentar el realismo de este mundo virtual se insertaron fotografías de las paredes, piso y techo tomadas en el laboratorio, así como fotos del gabinete de control del robot para decorar las cajas que modelan todos estos elementos, logrando que el entorno del robot fuera más parecido al de la realidad. La Figura 4.6 muestra el robot con todos sus detalles.

Figura 4.6. Robot virtual.

4.3. Interfaz de Comunicación Externa

VRML permite interacción, aunque en un solo sentido, entre el mundo virtual y un operador a través del monitor y del ratón de la computadora mediante una utilización adecuada de los diferentes sensores que proporciona este lenguaje. Sin embargo, su utilización en la creación de mundos virtuales interactivos está muy limitada, pues no permite la interacción bidireccional con otros programas.

La principal limitación de VRML es la dificultad para manipular los diferentes objetos del mundo virtual con valores precisos generados por otros programas de cálculo, como los que resuelven la cinemática del robot o los que hacen el control del mecanismo, o bien, con datos provenientes de la realidad. Otra limitación aparece al momento de querer utilizar en el mundo real información interna del mundo virtual. Sin embargo, VRML permite incluir código en *Javascript* dentro de un mismo programa, lo cual da más flexibilidad de manipulación del mundo virtual a los programadores.

Así, al lenguaje VRML se le implementó una interfaz de comunicación externa bidireccional, que denominamos ECI (*External Communication Interface*), la cual permite que se pueda manipular el mundo virtual con datos reales y precisos generados desde algún otro programa. La ECI se desarrolló en Java.

Para garantizar que, tanto el laboratorio real como el mundo virtual que lo representa, estén accesibles desde cualquier parte y casi desde cualquier plataforma, el acceso se hace mediante una página WEB. En este contexto, el uso de páginas WEB también resulta compatible con la ECI y con *applets* en Java, los cuales permiten lograr la comunicación entre módulos y la accesibilidad desde cualquier plataforma computacional, obteniendo así resultados más que satisfactorios. Aunque esta no es la única manera de lograr la accesibilidad y la comunicación bidireccional con un mundo virtual, sí es la más conveniente para lograr los objetivos planteados [Alarcón 00, Deitel 98, Internet 1, Lemay 96, Marrin 97, VRML 97].

El principio básico de la ECI es el siguiente. Un mundo virtual en VRML es esencialmente un árbol cuyo nodo principal o *Browser* contiene toda la información geométrica y de movimiento. A partir del *Browser*, cada nodo hijo es una clase de Java que es susceptible de ser modificada desde el exterior del mundo virtual mediante un programa en Java. Es por ello que con *applets* en Java es posible tener un buen control en línea del comportamiento cinético de todos los elementos geométricos del mundo virtual considerado. La manipulación completa y bidireccional del mundo virtual desde Java comprende las siguientes funciones:

- Obtener el mundo VRML en un objeto de clase *Browser* dentro del programa en Java.
- Leer en un objeto de la clase *Node* el nodo que contiene el objeto o grupo de objetos que se desea manipular, independientemente de su ubicación en el árbol completo.
- Leer el evento que permita realizar la manipulación deseada. Esta lectura debe hacerse en un objeto de clase de evento de entrada o salida y que tenga las propiedades que se desea manipular (rotación, translación, escalamiento, etc.).

4.4. Interactuando con el Mundo Virtual

La Figura 4.7 muestra la manera general en que la ECI funciona como interfaz entre una aplicación y el mundo virtual. La aplicación, por ejemplo la Interfaz de Usuario descrita en el

Capítulo 3, Figura 3.5, permite al usuario modificar los valores de la posición del robot (parámetros *q* y *t* y la posición de pinza) por medio del mouse. Los nuevos valores son recibidos por la ECI, la cual los envía al mundo virtual para actualizar la posición de los diferentes componentes en el mismo.

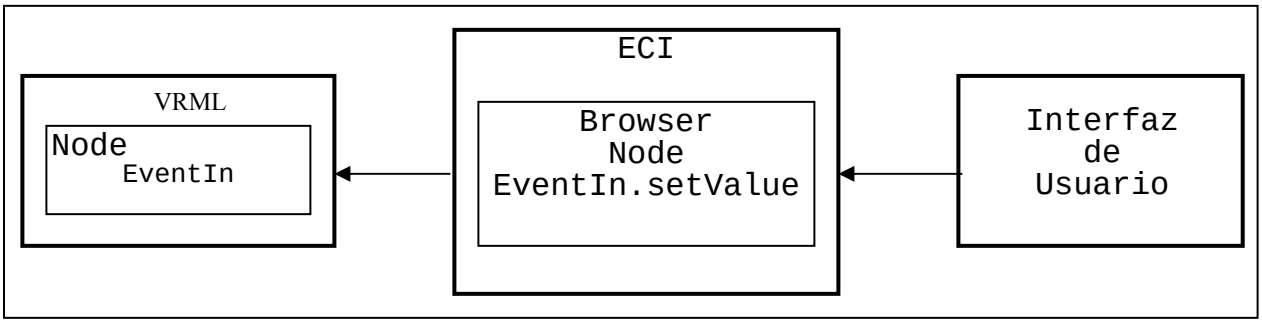

Figura 4.7. Acceso al mundo virtual desde la Interfaz de Usuario a través de la ECI

La interacción entre el mundo virtual, la ECI y una aplicación depende específicamente de los componentes a mover en el mundo virtual. A continuación presentamos la implementación de esta interacción en relación con el movimiento de algunos componentes.

4.4.1. Rotación de las Articulaciones

Los Programas 4.8, 4.9 y 4.10 implementan el movimiento de las articulaciones del robot (brazo, antebrazo y mano). El Programa 4.8 es parte de la ECI y es código Java. En este programa `rotJoin` es un objeto de la clase `Node` donde se lee, a través del objeto `Browser` (de donde podemos hacer referencia a cualquier nodo del mundo virtual), el nodo `RotJoint` del mundo virtual (descrito a continuación). En `rotAll` se lee, también del mundo virtual, el evento de entrada "`InRotJ`". Entonces, al llamar al método `rotAll.setValue` se envían al mundo virtual 3 nuevos valores correspondientes a las posiciones deseadas para el brazo, antebrazo y mano del robot.

En el Programa 4.9, parte del código del mundo virtual (VRML), `RotJoint` es un nodo que permite a cualquier aplicación externa acceder al mundo virtual, a través de su evento de entrada (eventIn) `InRotJ`. Obsérvese que `IntRotJ` es, además, una función javascript que se encarga de recibir los valores enviados por la ECI. Estos valores son asignados a los eventos de salida (eventOut) `RJ1`, `RJ2` y `RJ3`, los cuales envían los valores fuera de la función. Entonces, a

través del código VRML del programa 4.10, dichos valores son recibidos por los nodos de cada articulación del robot para desplegar el movimiento de rotación correspondiente.

Nótese que en la función `IntRotJ` (Programa 4.9) se actualizan también los nodos `SFGiraA`, `SFGiraB` y `SFGiraP`, los cuales permiten acceder a los *Sensores* que aseguran la realización de la rotación en las diferentes articulaciones. Aquí sólo se usan con el fin de especificar el desplazamiento (*offset*) del sensor, pues al realizar la rotación en alguna articulación, el *Sensor* desconoce toda rotación que se haya hecho, y se queda con el valor de la última rotación realizada por el usuario a través del *mouse*. Nótese también que `RJ1`, `RJ2` y `RJ3` son vectores de 4 valores cada uno. Los primeros 3 valores son fijos e indican que la rotación, especificada por el 4to valor (recibido de la ECI), es alrededor del eje *Y*: 0, 1, 0 (*X*, *Y*, *Z*).

```
rotJoin = browser.getNode("RotJoin");
rotAll = (EventInSFVec3f) rotJoin.getEventIn("InRotJ");
rotAll.setValue([θ1 θ2 θ3]);
```

Programa 4.8. Acceso a la función de rotación desde Java.

```
DEF RotJoint Script {
      eventIn  SFVec3f     InRotJ
      eventOut SFRotation  RJ1
      eventOut SFRotation  RJ2
      eventOut SFRotation  RJ3
      field    SFNode      SFGiraA USE GiraA
      field    SFNode      SFGiraB USE GiraB
      field    SFNode      SFGiraP USE GiraP
      url "javascript:
            function InRotJ(v, t){
              RJ1[0]=0; RJ1[1]=1; RJ1[2]=0; RJ1[3]=v[0]; SFGiraA.offset=v[0];
              RJ2[0]=0; RJ2[1]=1; RJ2[2]=0; RJ2[3]=v[1]; SFGiraB.offset=v[1];
              RJ3[0]=0; RJ3[1]=1; RJ3[2]=0; RJ3[3]=v[2]; SFGiraP.offset=v[2];
            }
      "
```

Programa 4.9. Declaración y cuerpo de la función `RotJoint` en un mundo virtual VRML.

```
ROUTE RotJoin.RJ1 TO Brazo.rotation
ROUTE RotJoin.RJ2 TO Antebrazo.rotation
ROUTE RotJoin.RJ3 TO Mano.rotation
```

Programa 4.10. Especificación de la ruta del flujo de datos para realizar la rotación.

4.4.2. Rotación con Interpolación

Con el fin de ahorrar tiempo de comunicacion y brindar una mejor animación en las rotaciones, es posible especificar *rutas* (ROUTE) de rotación en el mundo virtual. Dado que las rutas son especificadas dentro del mismo mundo virtual, y no dependen de constantes envíos por la ECI, se tienen mejores tiempos de respuesta y mejor animación que por medio de la ECI. Esto se logra mediante el uso de interpoladores de posición (`PositionInterpolator`). En estos interpoladores se especifica el valor inicial y el valor final de los ángulos de las articulaciones. Además del interpolador, es necesario declarar un sensor de tiempo (`TimeSensor`) para realizar el cambio de valores desde el inicio hasta el final en un cierto intervalo de tiempo. En el Programa 4.11 se presenta el código en VRML que declara el sensor y el interpolador.

```
DEF TSRotation TimeSensor { cycleInterval 1 }
DEF PIRotation PositionInterpolator{
      key[0,1]
      keyValue[0 0 0, 0 0 0]
}
```

Programa 4.11. Interpolador de rotación de las articulaciones y su sensor de tiempo.

El sensor de tiempo usado es un *Nodo* del tipo `TimeSensor` y es llamado `TSRotation`, cuyo campo `cycleInterval` contiene el valor del tiempo en segundos que dura activo el sensor (el tiempo activo del sensor es el tiempo que dura la ejecución de la animación). Por su parte, el interpolador es un *Nodo* del tipo `PositionInterpolator`, denominado `PIRotation`, en donde el campo `key` es un vector que especifica el número de vectores de datos que serán interpolados y el tipo de interpolación que se realizará, ya sea ascendente o descendente. `[0,1]` indica que son dos vectores y que los valores van a interpolarse iniciando en el primer vector y terminando en el segundo. El campo `keyValue` contiene los valores iniciales y finales para la interpolación.

```
R123 = browser.getNode("R123");
AdjustInterpolator = (EventInMFVec3f) R123.getEventIn("AdjustInterpolator");
tm = (EventInSFTime) R123.getEventIn("Go");
AdjustInterpolator.setValue([θ1i θ2i θ3i][θ1f θ2f θ3f]);
tm.setValue(0.5f);
```

Programa 4.12. Acceso a nodos y eventos desde Java para realizar rotación con interpolación.

Los Programas 4.12, 4.13 y 4.14 implementan el movimiento de las articulaciones del robot desde la posición inicial hasta la posición final, interpolando en todos los valores intermedios que hay en estas dos posiciones. En el Programa 4.12, parte de la ECI, `R123` es un objeto de la clase `Node`, donde se lee, a través del objeto `Browser`, el Nodo `R123` del mundo virtual. En `AdjustInterpolator` se lee, también del mundo virtual, el evento de entrada `AdjustTinterpolator`. En `tm` se lee el evento de entrada Go, también del mundo virtual. Entonces, al llamar al método `AdjustInterpolator.setValue` se envían al mundo virtual 6 valores correspondientes a las posiciones inicial y final del robot. Cuando se quiere iniciar la animación, se llama al método `tm.setValue`. Este método envía cualquier valor diferente de 0 al mundo virtual para activar el sensor.

```
DEF R123 Script {
      eventIn SFVec3f makeRot
      eventIn MFVec3f AdjustInterpolator
      eventIn SFTime Go
      eventOut SFRotation R0
      eventOut SFRotation R1
      eventOut SFRotation R2
      eventOut SFTime T0
      field SFNode PIR USE PIRotation
      field SFNode SFGiraA USE GiraA
      field SFNode SFGiraB USE GiraB
      field SFNode SFGiraP USE GiraP
      url "javascript:
            function AdjustInterpolator(v, t){
                  PIR.keyValue[0][0]=v[0][0];
                  PIR.keyValue[0][1]=v[0][1];
                  PIR.keyValue[0][2]=v[0][2];
                  PIR.keyValue[1][0]=v[1][0];
                  PIR.keyValue[1][1]=v[1][1];
                  PIR.keyValue[1][2]=v[1][2];
            }
            function Go(v, t){
                  T0=t;
            }
            function makeRot(v, t){
                  R0[0]=0;R0[1]=1;R0[2]=0;R0[3]=v[0]; SFGiraA.offset=v[0];
                  R1[0]=0;R1[1]=1;R1[2]=0;R1[3]=v[1]; SFGiraB.offset=v[1];
                  R2[0]=0;R2[1]=1;R2[2]=0;R2[3]=v[2]; SFGiraP.offset=v[2];
            }
      "
}
```

Programa 4.13. Función que permite hacer rotación con interpolación en el mundo virtual.

En el Programa 4.13, parte del código del mundo virtual (VRML), `R123` es un nodo que permite a cualquier aplicación externa acceder al mundo virtual, a través de sus eventos de entrada (eventIn) `AdjustInterpolator` y `Go`. Estos eventos son, además, funciónes

javascript que se encargan de recibir los valores enviados por la ECI. Cuando se accede a la función `AdjustInterpolator`, los valores recibidos de la ECI son asignados al interpolador (`PIRotation`) para actualizar sus valores. Esto se hace a través del campo `PIR`, el cual tiene asignado el nodo `PIRotation`. La función `Go`, al ser llamada, envía hacia fuera del nodo el valor recibido de la ECI, por medio del evento de salida `T0`. Entonces, a través del código VRML del programa 4.14, dicho valor es recibido por los nodos del sensor de tiempo `TSRotation`, el cual a su vez empieza a interpolar los valores de `PIRotation`. Mientras los valores de `PIRotation` estan cambiando, la función `makeRot` del nodo R123, convierte los datos de cada valor del interpolador en vectores de rotación para que puedan ser usados por los nodos de cada una de las articulaciones del robot.

```
ROUTE R123.T0 TO TSRotation.startTime
ROUTE TSRotation.fraction_changed TO PIRotation.set_fraction
ROUTE PIRotation.value_changed TO R123.makeRot
ROUTE R123.R0 TO Brazo.rotation
ROUTE R123.R1 TO Antebrazo.rotation
ROUTE R123.R2 TO Mano.rotation
```

Programa 4.14. Flujo de información para realizar la rotación con interpolación.

La figura 4.8 muestra de una manera esquemática la manera en que se realiza el flujo de información entre Java, la ECI y el mundo virtual para lograr la rotación con interpolación.

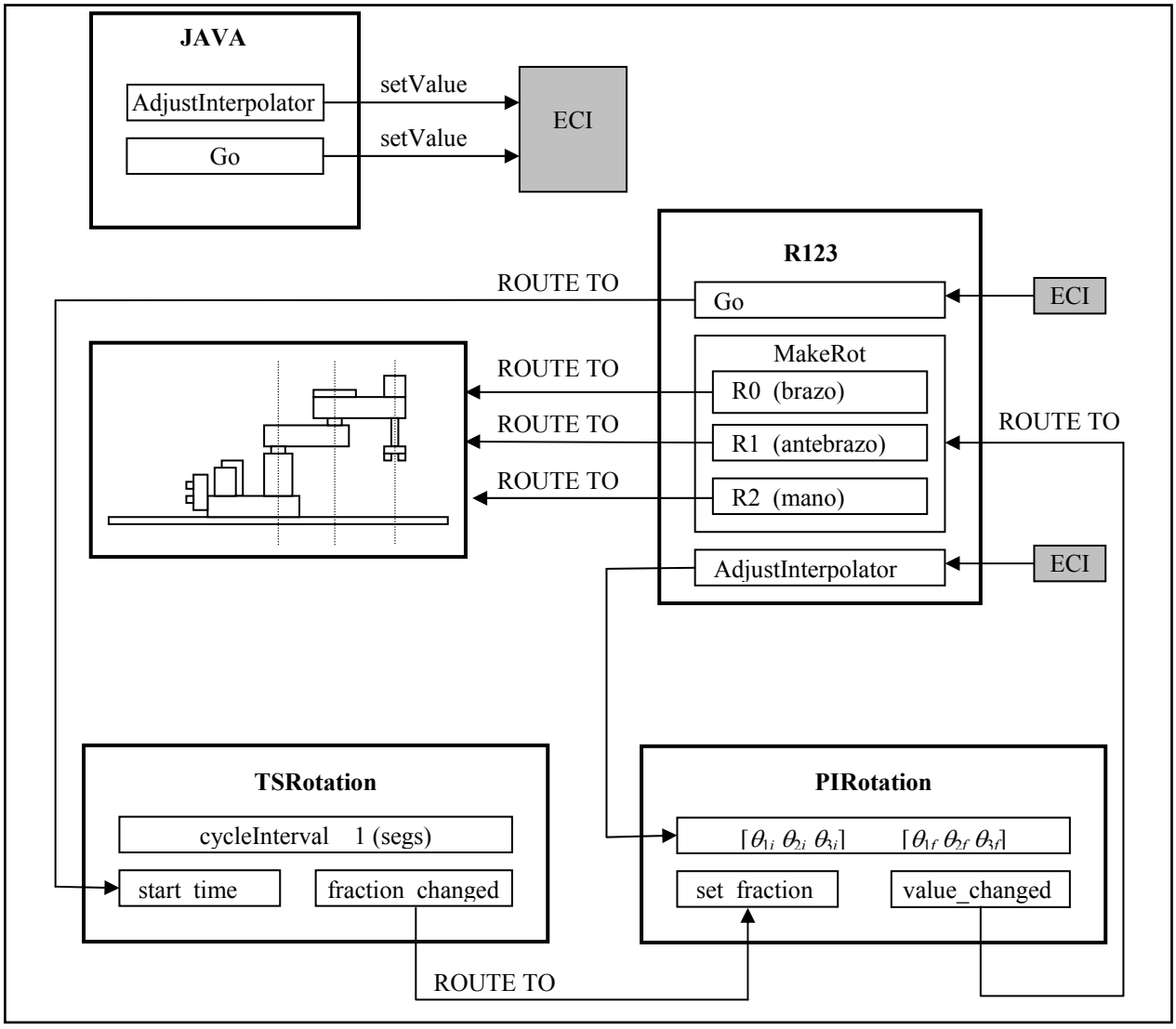

Figura 4.8. Esquema del flujo de información para realizar rotación con interpolación.

4.4.3. Manejo de la Pinza

Del mismo modo que se pueden controlar los movimientos rotacionales de las articulaciones del robot virtual, es posible controlar los movimientos de la pinza del robot. Cabe mencionar que la pinza del robot real tiene dos movimientos binarios: abrir-cerrar y subir-bajar. Para conseguir el movimiento de la pinza en el mundo virtual se hace uso de sensores de tiempo y de interpoladores de posición.

En el Programa 4.15 se presenta el código en VRML de la función que se encarga de tomar la decisión de subir/bajar la pinza.

```
#####Nodo de control subir y bajar
DEF ControlPinza Script {
    field SFBool pinzaAbajo FALSE
    field SFBool pinzaMoviendose FALSE
    eventIn SFTime botonPulsado
    eventOut SFTime subirPinzas
    eventOut SFTime bajarPinzas
    eventIn SFBool seMuevePinza
    url "javascript:
    function botonPulsado(v, t){
        if(!pinzaMoviendose){
            if(pinzaAbajo){
                subirPinzas = t;
                pinzaAbajo = FALSE;
            }
            else{
                bajarPinzas = t;
                pinzaAbajo = TRUE;
            }
            pinzaMoviendose = TRUE;
        }
    }
    function seMuevePinza(v, t){
        if (pinzaMoviendose && !v) pinzaMoviendose = FALSE;
    }
    "
}
```

Programa 4.15. Función en VRML que permite subir y bajar la pinza.

En este procedimiento, el *Nodo* por medio del cual se puede acceder a las funciones del nodo para realizar la animación de subir o bajar la pinza se denomina `ControlPinza`. Aquí, se pueden observar los campos `pinzaAbajo` y `pinzaMoviendose`, ambos del tipo `SFBool`, los cuales se utilizan como banderas para conocer el estado de la pinza.

Los eventos de entrada son `botonPulsado` y `seMuevePinza`; en donde `botonPulsado`, que es de tipo `SFTime`, nos indica que se ha solicitado el movimiento de subir o bajar la pinza, dependiendo de donde ese encuentre la pinza. El otro evento de entrada, `seMuevePinza`, indica que la pinza está en movimiento, lo cual se sabe a través del sensor de tiempo que la controla mediante su campo `isActive`.

Los eventos de salida `subirPinzas` y `bajarPinzas`, ambos del tipo `SFTime` ayudan a activar los sensores de movimiento para subirla o bajarla, dependiendo de su posición. Por otro lado, la función `botonPulsado` se encarga de evaluar si la pinza esta abajo o arriba y, dependiendo de esto, mandarla al lado contrario. Primero se asegura que la pinza no esté en

movimiento, lo cual se hace por medio de la bandera `pinzaMoviendose`. Si la pinza no está en movimiento entonces, por medio de la bandera `pinzaAbajo` se puede saber si se activa el sensor de subir o bajar la pinza.

La función `seMuevePinza` recibe un valor de tipo `SFBool` con el cual se puede saber si el sensor de subir o bajar esta activo o inactivo. En el Programa 4.16 puede verse la declaración de los sensores de tacto y tiempo, así como los interpoladores de posición que ayudan a realizar esta animación.

```
DEF Bajar TouchSensor { }

DEF motorBajar TimeSensor {
    cycleInterval 0.1
}

DEF motorSubir TimeSensor {
    cycleInterval 0.1
}

DEF bajarPinza PositionInterpolator{
    key[0,1]
    keyValue[0 0 0, 0 -11.2 0]
}

DEF subirPinza PositionInterpolator{
    key[0,1]
    keyValue[0 -11.2 0, 0 0 0]
}
```

Programa 4.16. Sensores de tacto, tiempo e interpoladores utilizados para subir y bajar la pinza.

El sensor de tacto (`TouchSensor`) denominado `Bajar` es el que solicita que la pinza cambie de posición, comunicandose con la función `BotonPulsado` del nodo `ControlPinza`. Por su parte, los sensores `motorBajar` y `motorSubir`, de tipo `TimeSensor`, especifican el intervalo de tiempo de la duración del movimiento de la pinza, que en este caso es de 0.1 segundos.

Los interpoladores de posición (`PositionInterpolator`) `subirPinza` y `bajarPinza`, especifican las coordenadas iniciales y finales de la posición de la pinza durante la animación. Para iniciar la animación es necesario especificar las rutas (`ROUTE`) de flujo de información en VRML en la forma indicada en el Programa 4.17.

```
ROUTE Bajar.touchTime TO ControlPinza.botonPulsado

ROUTE ControlPinza.bajarPinzas TO motorBajar.startTime
ROUTE motorBajar.fraction_changed TO bajarPinza.set_fraction
```

```
ROUTE bajarPinza.value_changed TO Pinza.translation
ROUTE motorBajar.isActive TO ControlPinza.seMuevePinza
ROUTE ControlPinza.subirPinzas TO motorSubir.startTime
ROUTE motorSubir.fraction_changed TO subirPinza.set_fraction
ROUTE subirPinza.value_changed TO Pinza.translation
ROUTE motorSubir.isActive TO ControlPinza.seMuevePinza
```

Programa 4.17. Flujo de datos para realizar la animación en la pinza.

Una vez activado el sensor de tacto `Bajar`, el aviso de que fue pulsado se envía a la función `botonPulsado` del nodo `ControlPinza` a través del evento de salida `touchTime`. Cuando la función `botonPulsado` decide si la pinza va a subir o bajar (supongamos que es bajar) activa el sensor de tiempo correspondiente, que en este caso sería `motorBajar`. Esto se hace asignándole un valor al evento de entrada `startTime` al sensor `motorBajar`, mediante el evento de salida `bajarPinzas` del nodo `ControlPinza`.

Ahora, ya que el sensor de tiempo está activado, y que sólo durará 0.1 segundos activado, es necesario activar el interpolador para que se haga la variación en sus valores durante este intervalo de tiempo. Esto se hace asignándole un valor al evento de entrada `set_fraction` mediante el evento de salida `fraction_changed` del sensor de tiempo correspondiente. Los datos que se están interpolando son accedidos mediante el evento de salida `value_changed` del interpolador y se le envían al evento de entrada `translation` del nodo `Pinza`.

Por medio del evento de salida `isActive` del sensor de tiempo `motorSubir` se sabe que el intervalo de tiempo no ha terminado, o bien, si la pinza aún esta moviendose. Este valor se envía a la función `seMuevePinza` del nodo `ControlPinza` para que la bandera `pinzaMoviendose` tome su correspondiente valor. Con esto se realiza la animación, pero para enviar la orden de animación es necesario que se active el sensor de tacto `Bajar`. Desde VRML esto se hace tocando el objeto asociado al sensor; pero también puede hacerse desde Java usando los objetos correspondientes. El Programa 4.18 presenta el código en Java que logra esto.

```
CP = browser.getNode("ControlPinza");
CPinza = (EventInSFTime) CP.getEventIn("botonPulsado");
CPinza.setValue(0.5);
```

Programa 4.18. Acceso al control de subir y bajar la pinza desde Java.

`CP` es un objeto de la clase *Node*, el cual, a través del objeto *Browser* obtiene el *Nodo* `ControlPinza` del mundo virtual VRML. `CPinza` es un objeto de la clase `EventInSFTime`

y obtiene el evento de entrada `botonPulsado` del mundo virtual VRML por medio del objeto `CP`. Por medio de este objeto se envía cualquier valor diferente de 0 para activar el sensor.

La descripción hecha en esta sección explica el funcionamiento del programa para activar la animación de la pinza cuando se desea que suba o que baje. Para tener una visión mas clara de cómo fluye la información desde la Interfaz de Usuario en Java, pasando por la ECI, hasta llegar al mundo virtual, la Figura 4.9 muestra esquemáticamente este flujo de información.

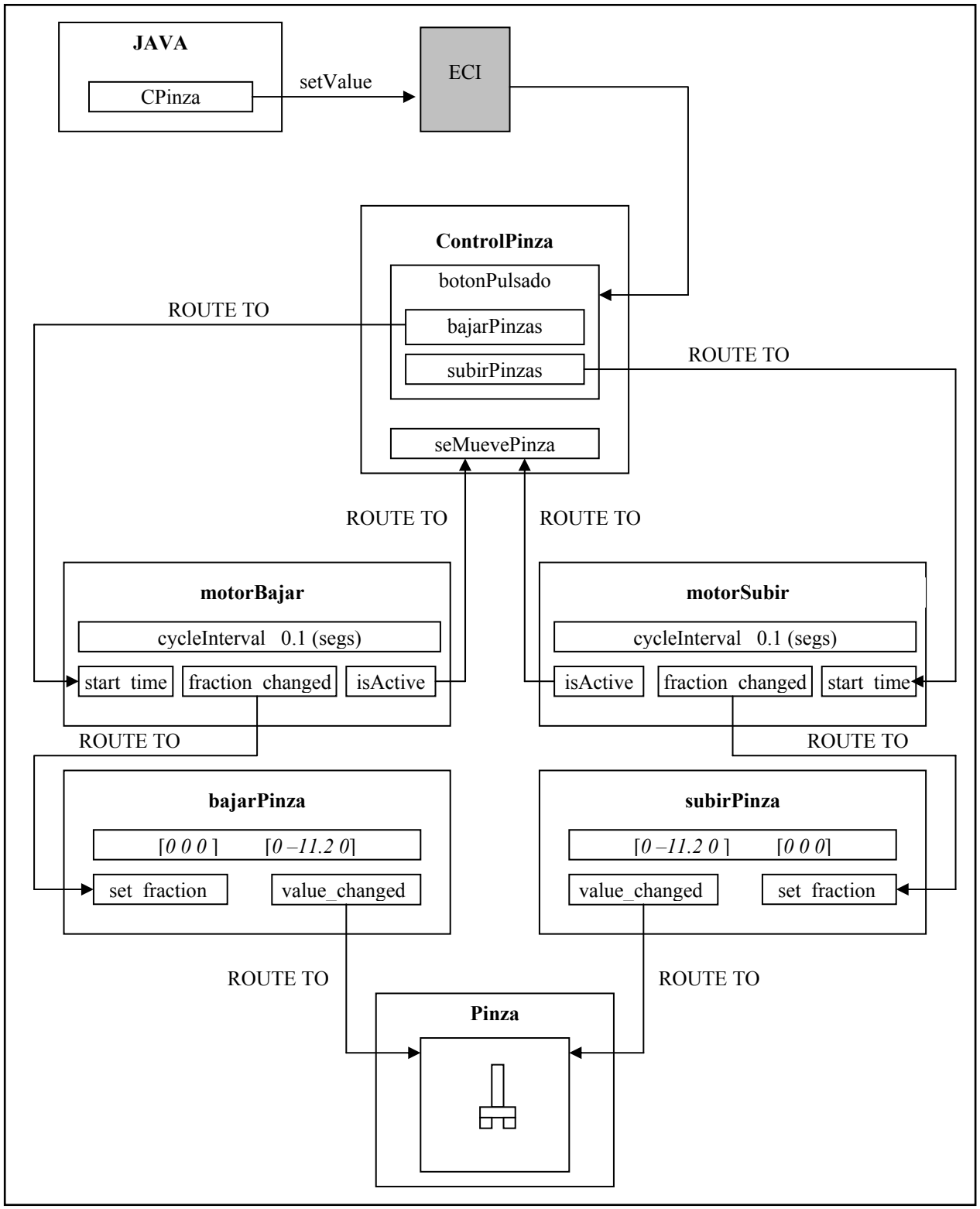

Figura 4.9. Esquema del flujo de información para subir y bajar la pinza.

Ahora, para realizar la animación correspondiente al movimiento de abrir y cerrar la pinza se sigue un procedimiento muy similar al de subir y bajar la piza, sólo que ahora se implementan dos interpoladores, pues la pinza tiene dos "dedos", cada uno de los cuales tiene una coordenada inicial y otra final. La coordenada inicial es cuando la pinza esta abierta, mientras que la coordenada final es cuando esta cerrada.

Así como es necesario escribir algunos *scripts* en Java para poder realizar las animaciones y el control del flujo de la información en los anteriores casos, del mismo modo, dichos *scripts* son necesarios para lograr la animación de apertura y cierre de la pinza. En el Programa 4.19 se presenta el código correspondiente a las funciones *javascript* en VRML aquí descritas.

```
DEF ControlPinzaII Script {
    field SFBool pinzaCerrada FALSE
    field SFBool pinzaMoviendoseII FALSE

    eventIn SFTime botonPulsadoII
    eventOut SFTime abrirPinzas
    eventOut SFTime cerrarPinzas

    eventIn SFBool seCierraPinza

    url "javascript:
    function botonPulsadoII(v, t){
        if(!pinzaMoviendoseII){
            if(pinzaCerrada){
                abrirPinzas = t;
                pinzaCerrada = FALSE;
            }
            else{
                cerrarPinzas = t;
                pinzaCerrada = TRUE;
            }
            pinzaMoviendoseII = TRUE;
        }
    }

    function seCierraPinza(v, t){
        if (pinzaMoviendoseII && !v) pinzaMoviendoseII = FALSE;
    }
    "
```

Programa 4.19. Función en VRML que permite abrir y cerrar la pinza.

Aquí, el nodo se denominó como `ControlPinzaII`, dentro del cual se encuentran declaradas dos funciones en *Javascript*. Las funciones, así como los eventos de entrada se llaman `botonPulsadoII` y `seCierraPinza`; en donde, `botonPulsadoII` es de tipo `SFTime` y `seCierraPinza` de tipo `SFBool`. Dentro de este nodo también hay dos campos que se utilizan como banderas, simplemente para conocer el estado de la pinza cuando se refiere a

apertura o cierre. Estos campos, `pinzaCerrada` y `pinzaMoviendoseII`, son del tipo `SFBool`. Los eventos `abrirPinzas` y `cerrarPinzas`, del tipo `SFTime` , están como datos de salida que posteriormente servirán para activar otros sensores. La función `botonPulsadoII` primero evalúa el valor de la bandera `pinzaMoviendoseII` y, si la pinza no esta moviéndose (abriendo o cerrando), entonces con la bandera `pinzaCerrada` se toma la decisión de abrirla o cerrarla, según corresponda. Para que se active el sensor correspondiente para hacer la animación, se pasa el valor del parámetro `t` a algún evento de salida (`abrirPinzas` o `cerrarPinzas`*)* los cuales se utilizan posteriormente para interpolar los datos necesarios y hacer la animación. También aquí se modifica el valor de la bandera `pinzaCerrada`. La función `seCierraPinza` recibe un valor de tipo `SFBool` por medio del cual se puede saber si el sensor que se encarga de ayudar a los interpoladores a variar los datos está activo o inactivo. Los sensores de tacto y tiempo utilizados para lograr esta animación son presentados y explicados en el Programa 4.20.

```
DEF Cerrar TouchSensor { }
DEF motorCerrar TimeSensor {
   cycleInterval 0.01
}
DEF motorAbrir TimeSensor {
   cycleInterval 0.01
}
DEF cerrarIzquierda PositionInterpolator{
    key[0,1]
    keyValue[0 0 0, .65 0 0]
}
DEF cerrarDerecha PositionInterpolator{
    key[0,1]
    keyValue[0 0 0, -.65 0 0]
}
DEF abrirIzquierda PositionInterpolator{
    key[0,1]
    keyValue[0.65 0 0, 0 0 0]
}
DEF abrirDerecha PositionInterpolator{
    key[0,1]
    keyValue[-.65 0 0, 0 0 0]
}
```

Programa 4.20. Sensores de tacto, tiempo e interpoladores que permiten abrir y cerrar la pinza.

El sensor de tacto (`TouchSensor`) `Cerrar` es por medio del cual se realiza el cambio de posición de la pinza, y este puede ser accedido desde algún nodo que tenga asociado el sensor, o bien, desde una aplicación externa a VRML escrita, por ejemplo, en Java. Desde este nodo, a

través de un ROUTE, se puede acceder a la función botonPulsadoII del nodo ControlPinzaII.

Los sensores de tiempo (TimeSensor) motorCerrar y motorAbrir son activados desde la función botonPulsadoII a través de los eventos de salida abrirPinzas y cerrarPinzas. Cuando son activados los sensores, estos le avisan a los interpoladores que empiecen a hacer la variación de datos durante el intervalo de tiempo especificado en el campo cycleInterval del sensor, que en este caso es de 0.01 segundos.

Los interpoladores de posición denominados PositionInterpolator, especifican la posición inicial y final de cada uno de los "dedos" de la pinza, por lo tanto son 4 interpoladores. Para cerrar la pinza están los interpoladores llamados cerrarIzquierda y cerrarDerecha, mientras que los interpoladores abrirIzquierda y abrirDerecha se utilizan para abrir la pinza. En el Programa 4.21 se presenta la declaración de las rutas (ROUTE) necesarias para llevar a cabo la animación y para controlar el flujo de la información.

```
ROUTE Cerrar.touchTime TO ControlPinzaII.botonPulsadoII

ROUTE ControlPinzaII.cerrarPinzas TO motorCerrar.startTime
ROUTE motorCerrar.fraction_changed TO cerrarIzquierda.set_fraction
ROUTE cerrarIzquierda.value_changed TO PinzaIzquierda.translation
ROUTE motorCerrar.fraction_changed TO cerrarDerecha.set_fraction
ROUTE cerrarDerecha.value_changed TO PinzaDerecha.translation
ROUTE motorCerrar.isActive TO ControlPinzaII.seCierraPinza

ROUTE ControlPinzaII.abrirPinzas TO motorAbrir.startTime
ROUTE motorAbrir.fraction_changed TO abrirIzquierda.set_fraction
ROUTE abrirIzquierda.value_changed TO PinzaIzquierda.translation
ROUTE motorAbrir.fraction_changed TO abrirDerecha.set_fraction
ROUTE abrirDerecha.value_changed TO PinzaDerecha.translation
ROUTE motorAbrir.isActive TO ControlPinzaII.seCierraPinza
```

Programa 4.21. Flujo de información que se encarga de abrir y cerrar la pinza.

Cuando se activa el sensor de tacto Cerrar, éste pasa el valor de su evento de salida touchTime a la función botonPulsadoII del nodo ControlPinzaII, en donde se toma la decisión de abrir o cerrar. Para comprender mejor esta explicación, supongamos que la decisión tomada por la función es la de cerrar. Entonces, por medio del evento de salida cerrarPinzas de la función botonPulsadoII se activa el sensor de tiempo motorCerrar. Esto es a través del evento de entrada del sensor llamado startTime. Cuando se activa el sensor de tiempo, éste sólo durará activo por 0.01 segundos, tiempo durante el cual es necesario estar cambiando los datos de los interpoladores. Esto se hace a través del evento de

salida `fraction_changed` del sensor, pasándole este valor a los interpoladores `cerrarIzquierda` y `cerrarDerecha`, por medio de su evento de entrada `set_fraction`.

Durante este intervalo de tiempo, el interpolador usa el valor actual modificado para realizar la animación haciendo una transformación de translación en cada uno de los "dedos" de la pinza. Esto se hace tomando el valor actual por medio del evento de salida `value_changed` y pasándolo al evento de entrada `translation` del nodo principal de cada uno de los "dedos". A través del evento de salida `isActive` del sensor `motorAbrir`, se accede a la función `seMuevePinza` del nodo `ControlPinzaII`, esto con el fin de saber si hay una animación en ese momento.

Con todo lo que ha sido explicado en los párrafos anteriores se realiza la animación de abrir o cerrar la pinza. Pero antes de esto, es necesario activar el sensor de tacto `Cerrar`, para que inicie el flujo de información. El sensor se puede activar tocando algún objeto que tenga el nodo del sensor asociado, o bien, desde la aplicación externa en Java, lo cual se hace utilizando el código que se muestra en el Programa 4.22.

```
CPII = browser.getNode("ControlPinzaII");
CPinzaII = (EventInSFTime) CPII.getEventIn("botonPulsadoII");
CPinzaII.setValue(0.5);
```

Programa 4.22. Acceso al control de la pinza para abrir y cerrar desde Java.

Aquí, `CPII` es un objeto de la clase *Node*, el cual, a través del objeto *Browser* obtiene el *Nodo* `ControlPinzaII` del mundo virtual VRML. Ahora con `CPII` podemos acceder a las funciones declaradas como eventos de entrada de `ControlPinzaII`. Mientras tanto, `CPinzaII` es un objeto de la clase `EventInSFTime` y obtiene el evento de entrada `botonPulsadoII` del mundo virtual VRML por medio del objeto `CPII`. Por medio de este objeto se envía cualquier valor diferente de 0 para activar el sensor.

La Figura 4.10 muestra el esquema de todo el proceso que se encarga de abrir y cerrar la pinza.

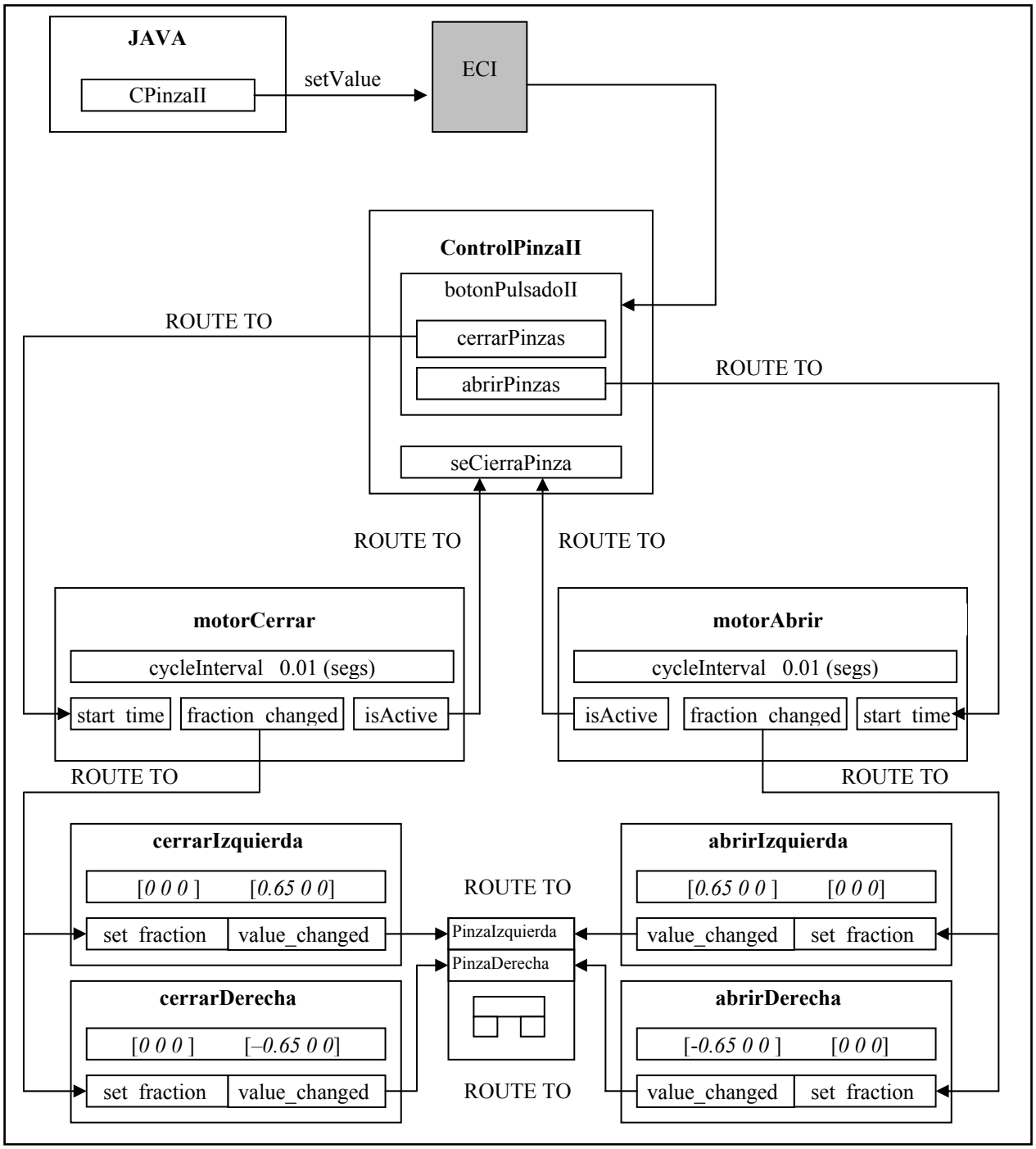

Figura 4.10. Esquema del flujo de información para abrir y cerrar la pinza

4.4.4. Creación y Supresión de Objetos en el Mundo Virtual

Así como es posible acceder al mundo virtual VRML desde Java para manipular los objetos existentes, también es posible crear nuevos objetos desde una aplicación en Java. Los objetos virtuales se crean con el fin de poder visualizar, en el mundo virtual, objetos reales que se encuentran en el área de trabajo del robot real. Aquí es necesario conocer la ubicación y la forma de cada uno de los objetos reales, para lo cual se utiliza un sistema de visión artificial previamente calibrado. Con estos datos se crean los objetos en algún objeto de Java y se envían al mundo virtual para poder verlos en el visulizador o *Browser*. En VRML es necesario declarar un nodo vacío de tipo `Transform` o de tipo `Group` en donde se crearán estos objetos, como el mostrado en el Programa 4.23.

```
DEF Shapes Transform {}
```

Programa 4.23. Nodo (VRML) vacío para crear objetos dentro de él.

En este caso, el nodo es de tipo `Transform` y se llama `Shapes`. `Shapes` será el nodo contenedor de las versiones virtuales de todos los objetos que se encuentren en el área de trabajo del robot real. En el Programa 4.24 se presenta el código en Java para poder tener acceso al nodo `Shape` y a los eventos que permiten crear y eliminar objetos.

```
Shapes = browser.getNode("Shapes");
addChildren = (EventInMFNode) Shapes.getEventIn("addChildren");
removeChildren = (EventInMFNode) Shapes.getEventIn("removeChildren");

addChildren.setValue(shape);
```

Programa 4.24. Código en Java para crear y/o remover objetos en el mundo virtual.

En este procedimiento (Java), `Shapes` es un objeto de la clase *Node*, el cual permite acceder al nodo `Shapes` de VRML. Ahora es necesario acceder a los eventos de entrada `addChildren` y `removeChildren` (ambos de VRML). Estos eventos son de tipo `MFNode;` y tanto el nodo `Transform` como el nodo `Group` contienen estos eventos. En Java se declararon los objetos `addChildren` y `removeChildren`, de la clase `EventInMFNode`, los cuales obtienen los eventos de añadir y eliminar objetos del nodo `Shapes`. Para añadir el objeto al nodo `Shapes`, es necesario hacerlo a través del método `setValue` del objeto `addChildren`. En

este caso el valor que se le está pasando es `shape`, de tipo `String`, en donde `shape` contiene una cadena con código en VRML del objeto deseado. Un ejemplo de este código es el que se muestra en el Programa 4.25:

```
shape=
        "Transform { "+
        "  children [ \n" +
        "    Shape {\n" +
       "      appearance Appearance {\n" +
       "        material Material {\n" +
       "         diffuseColor 0.2 0.8 0.2\n" +
       "        }\n" +
       "      }\n" +
       "      geometry Box { size 3 3 6 }\n" +
       "   } \n"+
       "  ]\n"+
      "}";
```

Programa 4.25. Código que permite crear parte de un programa en VRML desde Java.

Aplicando este procedimiento se almacena en `shape` una cadena que contiene código simple en VRML y queda lista para ser añadida en VRML por medio del objeto `addChildren`. Por otro lado, si se desea eliminar algún objeto es necesario, primero, que el objeto haya sido creado desde VRML y, luego, se usa el método `setValue` del objeto `removeChildren`.

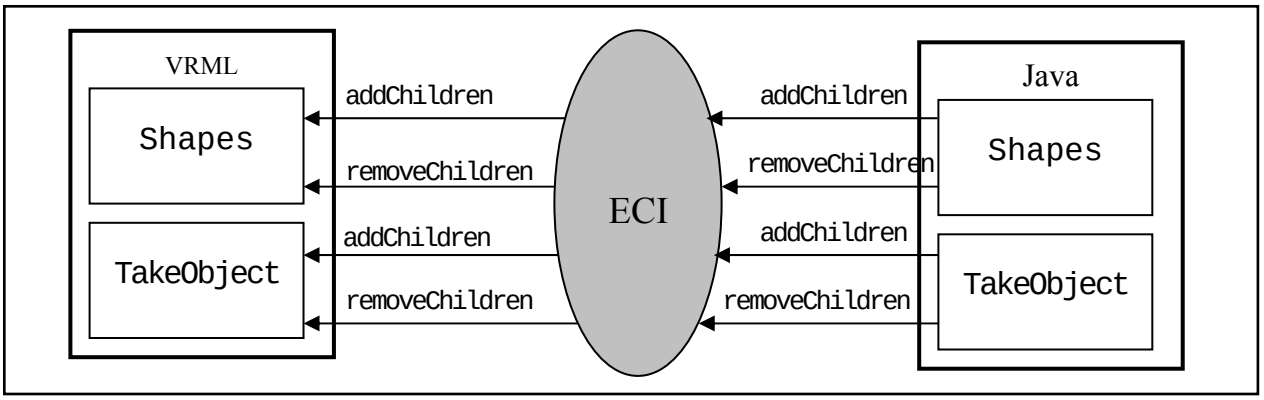

Figura 4.11. Esquema del flujo de información en la ECI para crear y eliminar objetos.

Dentro del programa tenemos dos nodos `Transform` para añadir o eliminar objetos en VRML, uno es el nodo `Shape`, que está ubicado en la parte conocida como `Raíz` del programa VRML, para que los objetos no sean hijos de algún nodo que contenga otros objetos o nodos. El otro lugar donde se pueden añadir o eliminar objetos es en un nodo `Transform` que esta ubicado como hijo de la pinza del robot, para que cuando se le ordene al robot tomar un objeto,

éste se elimine del nodo que se encuentra en la `Raíz` y se cree de nuevo pero ahora asociado a la pinza. Así, cuando el robot se mueva, el objeto se moverá como parte de él. El nodo que se encuentra como hijo de la pinza se llama `TakeObject`. La Figura 4.11 resume todas estas opciones para crear y eliminar objetos.

4.5. Interfaces de salida de los Mundos Virtuales

En nuestro proyecto de Robótica Virtual interesa recibir, en aplicaciones Java, datos de la posición actual del robot, del estado de la pinza en lo que se refiera a su posición (arriba, abajo, abierta o cerrada), e interesa también saber cual de los objetos encontrados en el área de trabajo ha sido tocado, para de esta manera hacer la manipulación correspondiente del mundo virtual. Los datos que se reciben pueden ser de tipo flotante (float), cuando viene el dato de algún sensor, o bien arreglos de flotantes (float[]) cuando se refiere a alguna rotación de cualquiera de las articulaciones.

Para enviar datos del mundo virtual a programas Java externos, es necesario acceder a los eventos de salida de los nodos del mundo virtual deseados. A cada nodo deseado se le asigna un identificador entero, que indica cual es el nodo que está enviando los valores a la aplicación Java y, a través de un método llamado *callback*, se accede a estos datos para utilizarlos en algún programa en Java a conveniencia de la aplicación.

4.5.1. Rotación

Cuando se realiza alguna rotación en las articulaciones del robot, el programa en VRML se comunica con el método *callback* en Java, enviándole el identificador de tipo entero del nodo VRML asociado a la articulación que se ha movido o a la función de la pinza que fue realizada. En el Programa 4.26 se presenta el código en Java para poder establecer comunicación con los nodos de las articulaciones a VRML. En este código, `rotJoin1`, `rotJoin2` y `rotJoin3` son objetos de la clase *Node*, los cuales a través del objeto *Browser* se puede tener acceso a los nodos `Brazo`, `Antebrazo` y `Mano` del mundo virtual para rotarlos

```
rotJoin1 = browser.getNode("Brazo");
rotJoin2 = browser.getNode("Antebrazo");
rotJoin3 = browser.getNode("Mano");
```

```
OutRotB1 = (EventOutSFRotation) rotJoin1.getEventOut("rotation_changed");
OutRotB2 = (EventOutSFRotation) rotJoin2.getEventOut("rotation_changed");
OutRotP = (EventOutSFRotation) rotJoin3.getEventOut("rotation_changed");

OutRotB1.advise(this, new Integer(1));
OutRotB2.advise(this, new Integer(2));
OutRotP.advise(this, new Integer(3));
```

Programa 4.26. Acceso a los nodos que envían información a través de los eventos de salida.

El evento de salida de cada uno de estos nodos es `rotation_changed`, el cual toma un valor siempre que hay una rotación en el nodo. Obtenemos acceso a los valores de estos eventos a través de los objetos `OutRobB1`, `OutRobB2` y `OutRobP` (de la clase `EventOutSFRotation`) que son accedidos a través de los objetos `rotJoin1`, `rotJoin2` y `rotJoin3`; respectivamente. Para poder identificar estos objetos cuando VRML manda un dato con ellos, es necesario asignarles el identificador numérico de tipo entero antes mencionado. Esto se hace a través del método `advise`, usando como primer parámetro "this" y segundo parámetro el número del identificador deseado. La llamada al método advise, está asociada con el método Java *callback* (ver Programa 4.27).

```
public void callback(EventOut who, double when, Object which) {
   Integer whichNum = (Integer) which;
       if (whichNum.intValue() == 1) {
            float[] val = OutRotB1.getValue();
       }
       else if (whichNum.intValue() == 2) {
            float[] val = OutRotB2.getValue();
       }
       else if (whichNum.intValue() == 3) {
            float[] val = OutRotP.getValue();
       }
}
```

Programa 4.27. Método *callback* que permite recibir eventos de salida desde Java.

El objeto `wich` es el valor del identificador del objeto que está enviando los datos, en donde se sabe que 1, 2 y 3 corresponden a los identificadores de los eventos de salida de los nodos de las articulaciones del robot. Una vez identificado el nodo que envía la información, se obtiene esta información través del método getValue, el cual devuelve un dato *float* [] (arreglo de flotantes en el caso de las rotaciones) y es copiado a una variable del mismo tipo. Sólo cuando los valores están asignados a una variable se puede hacer uso de ellos dentro del programa.

4.5.2. Sensores

La comunicación de los sensores del mundo virtual VRML con Java, también se hace de manera similar a la que se hace cuando hay rotación. La única diferencia es el tipo de dato que envía el sensor, pues aquí sólo es un valor flotante (*float*) sencillo. El código en Java para poder acceder a los datos de salida se presenta en el Programa 4.28.

```
Down = browser.getNode("Bajar");
Close = browser.getNode("Cerrar");
DownTouched = (EventOutSFTime) Down.getEventOut("touchTime");
CloseTouched = (EventOutSFTime) Close.getEventOut("touchTime");
DownTouched.advise(this, new Integer(4));
CloseTouched.advise(this, new Integer(5));
```

Programa 4.28. Acceso desde Java a los nodos que activan la pinza.

`Down` y `Close` son objetos del tipo *Node*. En estos se leen, a través del objeto *Browser*, los nodos Bajar y Cerrar del mundo virtual. Dentro del mundo virtual, estos nodos son sensores de tacto (`TouchSensor`) que, al ser activados, se encargan de iniciar el flujo de la animación del movimiento de la pinza. Los sensores de tacto tienen un evento de salida llamado `touchTime`, el cual toma un valor al momento que el sensor es activado. Con los objetos `DownTouched` y `CloseTouched`, que son de la clase `EventOutSFTime`, se puede acceder al evento de salida `touchtime` y, con esto, saber que el sensor está activo.

Como en el caso anterior, también es necesario asignarles un valor de identificador para saber cual es el nodo o evento que está comunicándose con la aplicación en Java. Esto se hace con el método `advise`. Además, es necesario insertar unas líneas de código en el método *callback* de la aplicación Java para identificar el nodo o evento que esta comunicándose con la aplicación. En el Programa 4.29 se puede ver el código insertado en el método *callback*. En VRML se tiene un código ya conocido para permitir las acciones con el ratón en estos objetos; dicho código se presenta en el Programa 4.30.

```
else if (whichNum.intValue() == 4) {
// Se esta subiendo o bajando la pinza
}
else if (whichNum.intValue() == 5) {
// Se esta abriendo o cerrando la pinza
```

```
}
```

Programa 4.29. Código del método *callback* que identifica el objeto que envía el evento.

```
DEF Bajar TouchSensor { }
DEF Cerrar TouchSensor { }
```

Programa 4.30. Declaración VRML de los sensores de tacto para ser accedidos desde Java.

4.5.3. Tocando objetos

Cuando los objetos insertados en el mundo virtual son creados desde Java, se les asigna un sensor de tacto a cada uno de ellos con su respectivo identificador, con el fin de que puedan ser tocados en el mundo virtual y de poder tomar decisiones al respecto desde Java. En el Programa 4.31, se muestra el código que permite añadir un sensor de tacto a un objeto que va a ser creado en el mundo virtual.

```
shape = browser.createVrmlFromString("Cadena de código VRML para
crear un objeto");
String Sensor="DEF ObjectSensor"+String.valueOf(1)+" TouchSensor
{ }\n ";
Node[] Sense = browser.createVrmlFromString(Sensor);
EventOutSFTime              nodeOut              =(EventOutSFTime)
Sense[0].getEventOut("touchTime");
nodeOut.advise(this, new Integer(6));
EventInMFNode nodesIn = (EventInMFNode)
shape[0].getEventIn("addChildren");
nodesIn.setValue(Sense);
addChildren.setValue(shape);
```

Programa 4.31. Creación de un objeto desde Java en VRML.

El objeto `shape`, del cual ya se habló anteriormente, es un objeto de la clase *Node*[] el cual almacena un objeto que va a ser creado en el mundo virtual VRML. Esto se hace a través del método `createVrmlFromString` del objeto *Browser*. El objeto `Sensor`, de la clase `String`, almacena el código de declaración del sensor como una cadena, misma que se va a insertar en el objeto que se va a crear en el mundo virtual. Se concatena un valor numérico convertido en *String* para que sean variables los nombres de los nodos de los sensores que se van a añadir a los objetos. A través del método `createVrmlFromString` del objeto *Browser*, se asigna esta cadena como un nodo al objeto `Sense` de la clase *Node*[].

El objeto `nodeOut` de la clase `EventOutSFTime` obtiene el evento de salida `touchTime` del sensor creado y, a través del método `advise`, se le asigna un identificador (único) al objeto. Todos los objetos tienen identificadores que son mayores de 5, pues los primeros 5 eventos de salida son ocupados por las 3 articulaciones del robot y los 2 movimientos de la pinza. En el objeto `nodesIn` de la clase `EventInMFNode` se obtiene el evento de entrada `addChildren` del objeto que va a ser creado, y con el método `setValue` se añade el sensor al objeto. Finalmente se añade al mundo virtual el objeto creado a través del objeto `addChildren` por medio del método `setValue`. Cuando un objeto en el mundo virtual es tocado con el ratón, éste envía a Java un tipo de información a través de la ECI, el cual es recibido en el método callback como se muestra en la Figura 4.12.

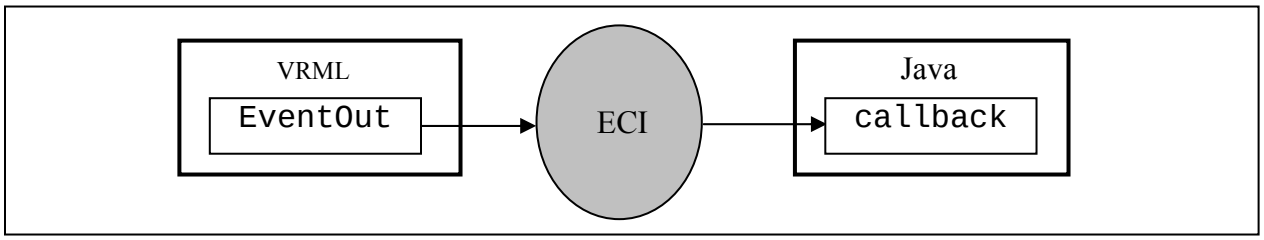

Figura 4.12. Esquema del flujo de los eventos de salida de VRML a Java a través de la ECI.

Capítulo 5

Diseño e Implementación de la Interfaz de Teleoperación vía Internet

Una de las funciones esenciales del Laboratorio Virtual de Robótica es la operación remota de la infraestructura del laboratorio real, en particular a través de Internet. En este capítulo se presenta el desarrollo de la interfaz de teleoperación del proyecto ROVIR encargada de asegurar la comunicación entre las tres diferentes interfaces de operación del robot: el *teach pendant* (TP) *hardware*, el TP virtual y el TP remoto. El primero es la interfaz original de programación del robot industrial utilizado, el UNIMATE S-103 y consiste de una botonera conectada al robot. El TP virtual es una aplicación que fue desarrollada en el verano del 2000 para sustituir al TP *hardware* usando una PC conectada al robot por el puerto serie [Herrera 00, Ibarra 00.1, Ibarra 00.2]. A esta aplicación se le hicieron modificaciones con la finalidad de que quedara accesible desde Internet utilizando la aplicación denominada TP *Remoto*, o bien, desde la interfaz de programación y simulación del robot virtual. Como ya se mencionó anteriormente, estas modificaciónes fueron desarrolladas por el autor.

5.1. Teach Pendant Hardware

La botonera o *teach pendant* es un *hardware* basado en el microprocesador 6801 conectado al sistema de programación y control del robot mediante un enlace serie [Unimate 85]. Este dispositivo cuenta con una serie de botones que permiten que el usuario interactúe con el robot, y además cuenta con un despliegue alfanumérico con dos renglones en donde aparece el menú de opciones con que el usuario cuenta para ello. El *teach pendant hardware* (TPH) permite al usuario configurar, programar y operar al robot. En la figura 5.1a aparece el TPH.

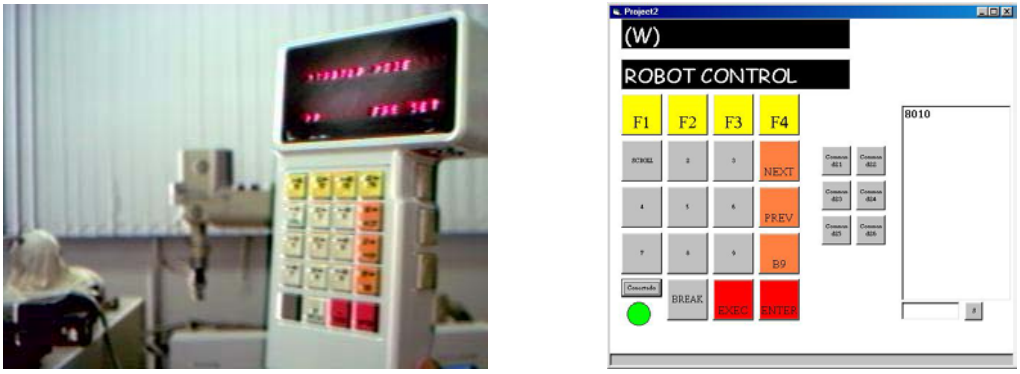

Figura 5.1. a) *teach pendant hardware*. b) *teach pendant virtual*.

Como se mencionó en la sección 2.2.1 de esta tesis, la operación del robot se hace desde el *teach pendant*, el cual posee un modo ejecutivo (*executive*) desde el cual se tiene acceso a los modos operativos de programación (*program*), de enseñanza de puntos (*teach*), de ejecución de programas (*run*), de establecimiento de los parámetros internos del robot (*set*), al modo de borrado (*clear*), al modo de comunicación externa (*load*) y al modo de bloqueo de la botonera (*lock*).

5.2. Teach Pendant Virtual

En esta sección se presenta la descripción del *teach pendant virtual*, sistema desarrollado en Visual Basic que permite la conexión de una computadora personal con el controlador del robot vía puerto serie (RS–232). Además, el TPV permite la conexión por *sockets* desde cualquier otra aplicación para tener un acceso remoto a él.

El *teach pendant virtual* (TPV) emula completamente al *teach pendant hardware*. El TPV muestra al usuario la interfaz gráfica mostrada en la Figura 5.1b, la cual tiene el mismo aspecto visual que el TPH, presentando una serie de botones cuya pulsación se hace mediante el ratón. Cada vez que es pulsado un botón, se envía al robot el identificador correspondiente para que éste responda como si fuera desde el *teach pendant hardware*, de esta manera se tiene un control total del robot desde la PC. A continuación se presentan con más detalle todas las transacciones que llevan a cabo entre el TPH y el sistema de control y programación del robot.

5.2.1. Conectándose con el Robot

En la parte inferior izquierda de la interfaz gráfica del TPV mostrada en la Figura 5.1.b se tiene un botón que sirve exclusivamente para abrir o cerrar la comunicación del TPV con el robot. Inicialmente, este botón tiene por título "Desconectado" y corresponde al color rojo en el indicador. Con el simple hecho de pulsar este botón, se inicia la conexión con el robot, cambiando su título a "Conectado" y el color del indicador a verde. Para finalizar la conexión se pulsa de nuevo el mismo botón, regresando todo a la configuración inicial. En el Programa 5.1 se presenta el código utilizado para establecer la comunicación con el controlador del robot desde la aplicación.

```
Private Sub command20_click()
 MSComm1.CommPort = 1
 If MSComm1.PortOpen = False Then
   MSComm1.PortOpen = True
   Command20.Caption = "Conectado"
   Connected = True
   canReceive = True
 Else
   Connected = False
   canReceive = False
   MSComm1.PortOpen = False
   Command20.Caption = "Desconectado"
 End If
End Sub
```

Programa 5.1. Código que permite conectarse con el robot a través del puerto serie.

El nombre de la rutina correspondiente es **command20_click**, la cual será ejecutada siempre que se pulse el botón llamado command_20. El objeto en Visual Basic que nos permite establecer la comunicación es llamado MSComm1 y es de tipo MSComm. Este objeto cuenta con

varios atributos y métodos que nos permiten hacer un uso eficiente de la conexión con el controlador del robot.

Al atributo `CommPort` se le asigna un valor relativo al número de puerto serie que se va a utilizar (1 para el primer puerto serie). Posteriormente, se evalúa el valor del atributo PortOpen, el cual almacena un valor binario (verdadero o falso) que nos indica si el puerto está actualmente abierto y listo para la comunicación. Si es falso se abre el puerto mediante la asignación del valor verdadero a este atributo. Si el atributo `PortOpen` tiene el valor de verdadero, la conexión con el robot ya está establecida. En este caso sólo se puede cerrar la conexión. Para ello sólo se cambia el valor del atributo `PortOpen` a falso.

5.2.2. Recibiendo Datos del Robot

Una vez que la conexión está establecida, se llama constantemente a la rutina **recibir** (información), desde la rutina **Timer1_Timer** que se ejecuta en todo momento. En el Programa 5.2 se presenta el código de **Timer1_Timer.**

```
Private Sub Timer1_Timer()
   If Connected = True Then
      If canReceive = True Then
         canReceive = False
         recibir
         canReceive = True
      End If
   End If
End Sub
```

Programa 5.2. Sincronización de lectura de información del puerto serie.

En esta rutina se hace uso de las banderas mencionadas anteriormente, pues es necesario asegurar que no se intente recibir datos cuando aún no se haya establecido la comunicación con el robot. Si la bandera `Connected` tiene valor verdadero y la bandera `canReceive` también tiene ese valor, entonces se hace la llamada a la rutina de recepción de datos del robot. Mientras está recibiendo información, se cambia el valor de la bandera `canReceive` por falso, esto con la finalidad de no volver a llamar a esta rutina si no ha finalizado su ejecución. Una vez que finalizó la ejecución de estas rutinas, se asigna el valor de verdadero a `canReceive`, con lo cual pueden ser ejecutadas de nuevo.

La rutina **recibir** se encarga de estar tomando la información que el robot envía al puerto serie, a donde llega como una secuencia de caracteres, los cuales deben de ser divididos en dos cadenas de datos. La rutina **recibir** se encarga de hacer esta división así como la organización de datos para poder representarlos adecuadamente en la aplicación. En el Programa 5.3 se presenta el código de esta rutina, cuyo funcionamiento a grandes rasgos se describe a continuación.

```
Private Sub recibir()
 Dim PauseTime
 Dim cad As String
 Dim cod As String
   ban = 0
   If MSComm1.InBufferCount Then
    Buffer = MSComm1.Input
      cad = ""
      cod = ""
      For i = 1 To Len(Buffer)
         If str(Asc(Mid(Buffer, i, 1))) = 10 Then
            ban = ban + 1
         End If
         If ban = 1 Then
             If str(Asc(Mid(Buffer, i, 1))) > 30 Then
                 cad = cad + Chr$(Asc(Mid(Buffer, i, 1)))
             End If
         End If
         If ban = 2 Then
            If str(Asc(Mid(Buffer, i, 1))) > 30 Then
                cod = cod + Chr$(Asc(Mid(Buffer, i, 1)))
            End If
         End If
      Next i
      List2.List(j) = cad
      List1.List(j) = cod
      j = j + 1
    Else
   End If
    PauseTime = 0.3
    Start = Timer
    Do While Timer < Start + PauseTime
        DoEvents
    Loop
End Sub
```

Programa 5.3. Recepción y organización de los datos provenientes del puerto serie.

La bandera `ban` ayuda a definir la división de las cadenas de datos recibidos. Dentro de los datos recibidos, se encuentra un carácter especial que nos indica donde termina una cadena de datos. Este carácter tiene el valor numérico 10, y cada vez que llega se incrementa el valor de la bandera `ban`. Este carácter llega tres veces: al inicio de la primer cadena de datos, al iniciar la segunda cadena del bloque y al final de la segunda cadena.

Cuando `ban` tiene el valor 1, se asignan los caracteres que van llegando a la cadena cad, hasta que se encuentre el nuevo carácter con el valor numérico 10. Cuando `ban` tiene el valor 2, los siguientes caracteres se asignan a la cadena `cod`. Cuando se encuentra un tercer carácter con el valor numérico 10, el buffer de datos de llegada ha finalizado, entonces se despliegan estas dos cadenas en las listas de texto `List1` y `List2`, las cuales nos muestran en la interfaz la información que ha llegado del robot. Finalmente se hace una pausa de 0.3 segundos para dar tiempo a que finalicen otros eventos que están en ejecución antes de volver a recibir datos.

5.2.3. Enviando Datos al Robot

El envío de datos al robot es relativamente sencillo, solo se necesita definir una rutina por cada botón pulsado, dentro de la cual se asigna el valor deseado a la variable `salida`. La rutina **`manda_dato`**, encargada de enviar el dato a través del objeto `MSComm1`, es llamada desde cada rutina que se invoca al pulsar uno de los botones.

En el Programa 5.4 se presentan las declaraciones de una de estas rutinas con su respectiva explicación. Como ejemplo se utiliza la rutina invocada al pulsar el botón marcado con el número 3, la cual es llamada **`Command7_Click`**, pues el nombre del botón es `Command7`. Se asigna el valor 181 a la variable `salida`, el cual es el número de identificador para éste botón y, en cada rutina de botón pulsado, se asigna el valor correspondiente a esta variable. Posteriormente se invoca a la rutina **`manda_dato`**.

```
Private Sub Command7_Click()
   salida = 181
  manda_dato
End Sub
Private Sub manda_dato()
    If MSComm1.PortOpen = True Then
       MSComm1.Output = Chr$(salida)
    End If
End Sub
```

Programa 5.4. Envío de datos al puerto serie.

La rutina **`manda_dato`** es la que se encarga de enviar el dato al puerto, lo cual se hace asignándole el valor del dato que se desea enviar (`salida`) al atributo `Output` del objeto

MSComm1. En caso de que el puerto no esté listo para comunicarse con el robot, no es enviado ningún dato ayudándose para ello con el atributo PortOpen del mismo objeto.

5.2.4. Recibiendo Datos Remotos

El *teach pendant virtual* también cuenta con el uso de *sockets* para permitir que sea accedido desde una aplicación remota, la cual puede ser, por ejemplo, el *teach pendant remoto*, o bien, la interfaz de programación y simulación del robot virtual. Gracias al uso de *sockets*, la aplicación queda disponible desde cualquier lugar en Internet, siempre y cuando el operador o usuario remoto tenga los privilegios de seguridad para lograrlo.

Para hacer uso de *sockets* en Visual Basic, se utiliza un objeto de tipo MSWinSck, el cual cuenta con todos los métodos y atributos necesarios para establecer comunicación desde otra aplicación, además de usar algunos otros atributos o métodos para validar el estado de la conexión y algunas otras restricciones necesarias. El objeto utilizado es llamado tcpServer.

```
tcpServer.LocalPort = 8010
tcpServer.Listen

Private Sub tcpServer_ConnectionRequest(ByVal requestID As Long)
    tcpServer.Accept requestID
End Sub
```

Programa 5.5. Esperando y aceptando llamadas por *sockets*.

Lo primero que se debe hacer es asignarle un número de puerto al *socket* que se va a quedar en espera de datos y, posteriormente, dejar el *socket* en espera de alguna conexión (ver Programa 5.5). El número de puerto se asigna al atributo LocalPort del objeto tcpServer, y para dejarlo en espera se invoca al método Listen del mismo objeto. La rutina que se debe definir es **tcpServer_ConnectionRequest**, la cual es invocada cuando se trata de hacer una conexión al *socket* establecido en el puerto especificado en donde, al tratar de hacer la conexión, llega automáticamente un identificador de la computadora remota, y se invoca al método Accept enviando como argumento este identificador. Una vez que se estableció la comunicación remota los datos pueden llegar en cualquier momento y son recibidos con la ayuda de una rutina para la llegada de datos, la cual se presenta en el Programa 5.6.

```
Private Sub tcpServer_DataArrival(ByVal bytesTotal As Long)
    Dim strData As String
      tcpServer.GetData strData
      DoEvents
End Sub
```

Programa 5.6. Llegada de datos al *socket*.

Los datos se reciben en la rutina **`tcpServer_DataArrival`**, y son asignados a la variable `strData` por medio del método `GetData` del objeto `tcpServer`. Cuando se tienen los datos en esta variable pueden ser utilizados a la conveniencia necesaria. Además, en esta variable puede llegar un identificador que indica si se ha recibido un programa en C/ROS, coordenadas articulares, o bien, la orden de pulsación de uno de los botones de la interfaz. Lo cual sucede cuando se uyiliza el *teach pendant remoto*.

```
Private Sub Evaluate(cad As String)
Dim PauseTime
   If (cad = "F1") Then Command1_Click
   If (cad = "F2") Then Command2_Click
   If (cad = "F3") Then Command3_Click
   If (cad = "F4") Then Command4_Click
   If (cad = "0") Then Command17_Click
   If (cad = "1") Then Command5_Click
            :
   If (cad = "9") Then Command15_Click
   If (cad = "B9") Then Command16_Click
   If (cad = "Break") Then Command17_Click
   If (cad = "Exec") Then Command18_Click
   If (cad = "Enter") Then Command19_Click
   If (cad = "Next") Then Command8_Click
   If (cad = "Prev") Then Command12_Click
   If (cad = "t1+") Then Command21_Click
   If (cad = "t2+") Then Command23_Click
   If (cad = "t3+") Then Command25_Click
   If (cad = "t1-") Then Command22_Click
   If (cad = "t2-") Then Command24_Click
   If (cad = "t3-") Then Command26_Click
End Sub
```

Programa 5.7. Evaluación de una cadena para enviar datos al robot.

Cuando se recibe un programa en comandos C/ROS, estos se almacenan en un arreglo llamado `commands`, el cual almacena comandos y parámetros de forma numérica, es decir, solamente identificadores y valores del parámetro. Cuando se reciben coordenadas articulares, estas se almacenan en un arreglo llamado `coordinates`, estos datos son los valores de las coordenadas de cada una de las tres articulaciones del robot. Cuando se reciben simplemente

órdenes para la pulsación de algún botón, estas son ejecutadas al tiempo que van llegando haciendo la ejecución en tiempo real.

Para recibir un programa u ordenar que se pulse algún botón, se evalúa una cadena usando la rutina, cuyo código se presenta en el Programa 5.7. **`Evaluate`** recibe como parámetro la cadena que se desea evaluar. Cuando se identifica el contenido de la cadena, se invoca a la rutina que es llamada al pulsar el botón deseado, y con esto se pulsa el botón remotamente.

Cuando lo que se ha recibido es un programa a transmitir al robot, incluyendo sus coordenadas, entonces una rutina se encarga de pulsar los botones necesarios para transmitir el programa al robot como si se estuviera haciendo desde el *teach pendant hardware*. Para que esta secuencia funcione correctamente, es necesario que el robot esté en modo de inicio (home).

Lo primero que se hace es poner al robot en modo de programación (PGM), pulsando el botón **EXEC** y posteriormente el botón **F1**. Posteriormente se evalúa cada elemento del arreglo commands, tomando en cuenta que cada par de datos contiene un identificador de comando en C/ROS y su argumento correspondiente. Cuando el identificador corresponde a uno encontrado en el despliegue del *teach pendant*, entonces se solicita pulsar el botón correspondiente. De lo contrario, se solicita pulsar una secuencia de botones hasta mostrar en el despliegue los comandos correspondientes y se solicita pulsar el botón necesario. Si el comando solicitado no es el de retorno de subrutina (RE), entonces no hay argumento que transmitir, de lo contrario, se solicita pulsar los botones del argumento solicitado.

Cuando se ha terminado de transmitir el programa, es necesario pulsar otra secuencia de botones para llegar al modo de aprendizaje de puntos (modo TCH) en el robot y, posteriormente, transmitirle las coordenadas articulares de cada uno de los puntos. Una vez finalizado todo este proceso de transmisión de información al robot, se pulsa otra serie de botones para llegar al modo de ejecución de un programa (modo RE), y esto hace que el robot empiece a realizar el programa solicitado.

En el Programa 5.8 se muestra el código que transmite un programa.

```
Evaluate ("Exec")           'pone al robot en modo de programación
Evaluate ("F1")
For i = 2 To ncoms - 4
 If (commands(i)="3") Then Evaluate ("F1")       'MP
 If (commands(i)="6") Then Evaluate ("F4")      'GR
 If(commands(i)="10")Then Evaluate("1"):Evaluate("1"):Evaluate("F4")  'CS
```

```
 If (commands(i)="8")Then Evaluate("1"):Evaluate("1"):Evaluate("F3")  'GO
 If(commands(i)="9")Then Evaluate("1"):Evaluate("1"):Evaluate("1"):Evaluate("F1")'LB
 If (commands(i)="18")Then Evaluate("1"):Evaluate("1"):Evaluate("1"):Evaluate("F4")
 If (commands(i)="11")Then Evaluate("1"):Evaluate("1"):Evaluate("1"):Evaluate("F2")
 If (commands(i) <> "11") Then   ' si no fue RE, entonces lee argumento
  For j = 1 To Len(commands(i + 1))
   Evaluate (Mid(commands(i + 1), j, 1))   ' este ciclo transmite los comandos
  Next j                                   ' al robotreal
  Evaluate ("Enter")
 End If
 Evaluate ("Next")
 i = i + 1
Next I
Evaluate ("Exec")
Evaluate ("F2")
For i = 1 To ncoords – 3                   ' este ciclo transmite los valores
 Evaluate ("F2")                           ' q1 q2  y q3 de cada punto conocido
  For j = 0 To 2                           ' por el robot virtual
   For k = 1 To Len(coordinates(i + j))
    Evaluate (Mid(coordinates(i + j), k, 1))
   Next k
   Evaluate ("Enter")
  Next j
 i = i + 2
 Evaluate ("Next")
Next i
Evaluate ("Exec")
   Evaluate ("F4")
   Evaluate ("F1")
```

Programa 5.8. Evaluación de las cadenas de un programa y coordenadas para enviar al robot.

5.3. Teach Pendant Remoto

El funcionamiento del *teach pendant* virtual es muy similar al del *teach pendant hardware*, con lo cual cualquier usuario capacitado para utilizar el TPH puede utilizar también el TPV. El *teach pendant* virtual permite además, la comunicación con otras aplicaciones mediante *sockets*. Así, se desarrolló un sistema que es accesible por Internet y hace uso del TPV, motivo por el cual se denomina *teach pendant remoto* (TPR). La teleoperación, entonces, queda garantizada por el TPR, aplicación que corre en una computadora remota conectada al servidor *e*–robot. La página WEB de teleoperación formada por el *teach pendant* remoto constituye la primera alternativa de conexión remota al Laboratorio Virtual de Robótica. Es la más simple y permite tener acceso a prácticamente cualquiera de los modos de operación del robot industrial utilizado.

El TPR, desarrollado en Java y ejecutado como un *applet* desde una pagina WEB con un *Browser* de Internet, está disponible desde cualquier plataforma y puede ser accedido desde cualquier lugar en Internet siempre y cuando se tengan los permisos y autorización para hacerlo.

El *teach pendant remoto* se conecta a través de un *socket* con el *teach pendant* virtual (el cual se ejecuta en la PC denominada *e*–robot, servidor conectado al robot) y, en la computadora remota, se presenta una interfaz idéntica al *teach pendant* real, la cual se utiliza de la misma forma. Por cada botón pulsado, este sistema envía un identificador al *teach pendant* virtual quien lo interpreta enviándole al robot el mismo dato que se enviaría si se hubiera pulsado el botón correspondiente en el *teach pendant* virtual o en el real.

De este modo, el robot puede ser teleoperado desde cualquier parte del mundo a través de esta página WEB y, para que el usuario remoto pueda percatarse de lo que está sucediendo en el laboratorio real, el sistema de monitoreo visual (implementado en la misma página) permite observar todos y cada uno de los movimientos del robot así como su comportamiento dependiendo de la acción ordenada.

1er paso	Encender el robot • Esperar la inicialización • Presionar el botón hardware *Reset*	Operador del Laboratorio
2° paso	Correr el *teach pendant* virtual en el servidor *e*–robot • Conectarse con el robot • Enviar al robot a la posición *Home* con la interfaz web	Operador del Laboratorio
3er paso	Conectar Cliente/Servidor/Robot: • Conectarse a la página web de Teleoperación • Presionar el botón *Connect* en la interfaz web	Cliente remoto
4° paso	Teleoperar el robot: • Utilizar todas las funciones del *teach pendant* remoto de la misma manera que en el *TP* real o en el virtual	Cliente remoto

Tabla 5.1. Mecanismo de teleoperación vía Internet basado en el *teach pendant remoto*

La Tabla 5.1 muestra las diferentes acciones que deben tomar tanto el operador del sistema ubicado en el Laboratorio de Robótica y Visión del CINVESTAV como el usuario remoto conectado por Internet. La Figura 5.2 muestra el aspecto de la página WEB correspondiente al *teach pendant* remoto, incluyendo la imagen proporcionada por la *webcam* (esta figura es la Figura 3.11 y se repite por conveniencia).

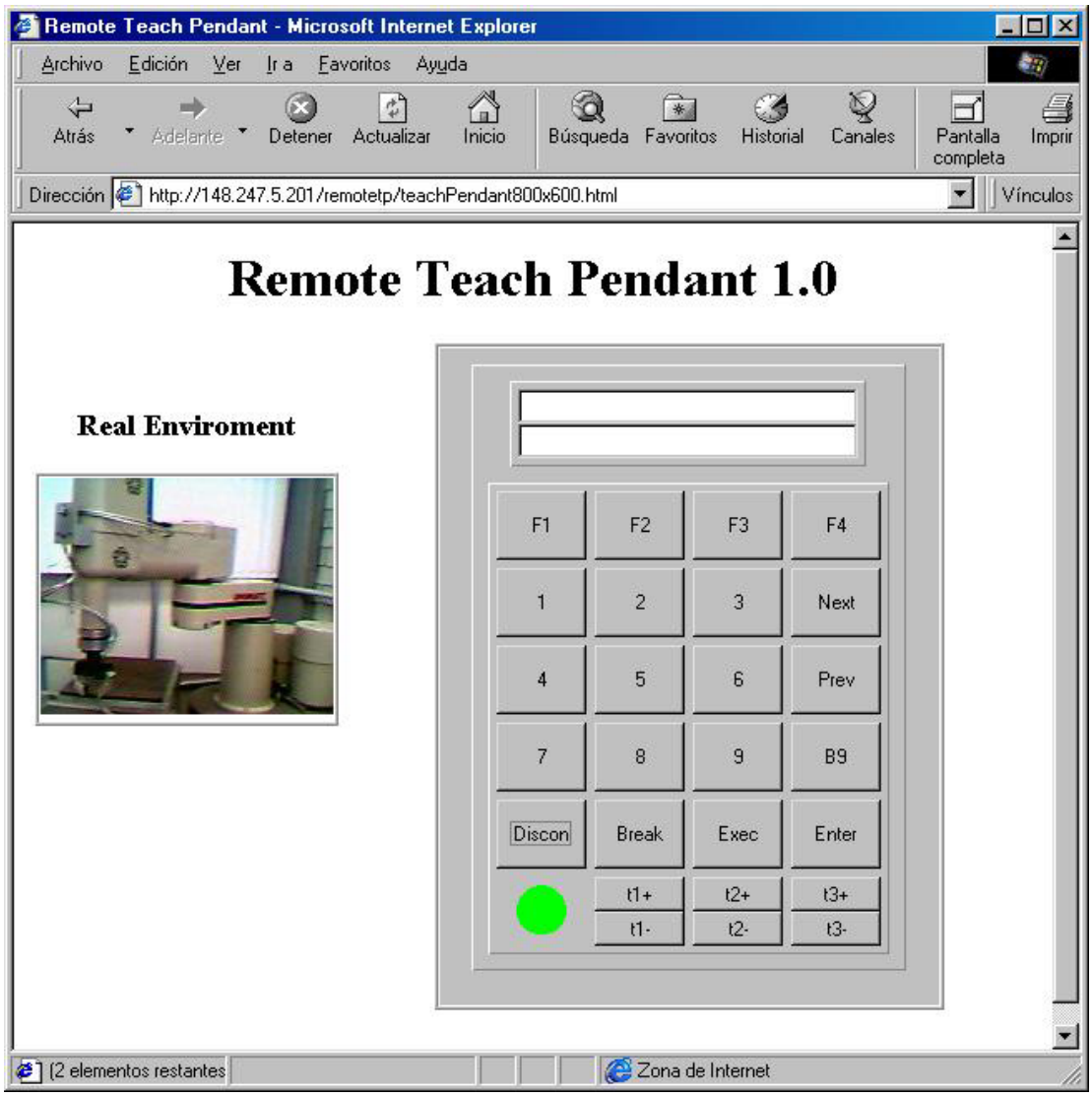

Figura 5.2. *teach pendant remoto*.

5.3.1. Conectándose con el Teach Pendant Virtual

El TPR es un *applet* de Java que se encuentra en el servidor rovir, y es accedido a través de su URL: http://rovir.ctrl.cinvestav.mx/remotetp/teachPendant800x600.html. A continuación se explica la manera en que se comunica con el TPV.

Primero se obtiene la dirección del servidor a través del método `getByName` de la clase `InetAddress`. Esta dirección se asigna al objeto `addr`, de la misma clase. Posteriormente se establece la comunicación con el servidor usando la clase `Socket`, y el objeto `socClient`. En el objeto `oClient`, de la clase `OutputStream`, obtenemos el flujo de salida de datos del

socket, con el método `getOutputStream` del objeto `socClient`. Finalmente se utiliza un objeto a través del cual se envian datos al objeto `oClient`. El objeto debe es de la clase `ObjectOutput` y se le llamó `sClient`. Con esto ya se pueden enviar datos al *socket* a través del objeto sClient.

```
InetAddress addr = InetAddress.getByName("rovir.ctrl.cinvestav.mx");
Socket socClient = new Socket(addr, 8010);
oClient = socClient.getOutputStream();
sClient = new ObjectOutputStream(oClient);
```

Programa 5.9. Conexión al *socket* de VB desde Java.

5.3.2. Enviando datos al Teach Pendant Virtual

Para enviar los datos al TPV se ha definido un método denominado `sendData`. El método `sendData` pertenece a la clase principal de la aplicación encargada de utilizar el objeto que tiene establecida la comunicación con el TPV. El código correspondiente se presenta en el Programa 5.10. Este método recibe como parámetro el objeto de tipo `ObjectOutput`, el cual se encarga de sacar el dato hacia el *socket*. Otro parámetro del método es `S`, de la clase `String`, y es el dato que se desea enviar. En este método primero se limpia el objeto con el método flush, esto con el fin de eliminar algún dato que se encuentre en el buffer de salida. Posteriormente se escribe el dato en el *socket* con el método writeBytes del objeto de tipo `ObjectOutput`.

```
public static void sendData(String S, ObjectOutput O)
     {
                O.flush();
                O.writeBytes((String) S);
     }
```

Programa 5.10. Envío de datos al *socket*.

Ahora bien, para decidir que dato es el que se va a enviar al *socket*, se llama al método `action` cada vez que se pulsa un botón de la interfaz. El método `action` recibe como parámetros el objeto que generó el evento, así como el evento generado. Evaluando este objeto se envían los datos al *socket* a través del método `sendData`, y se muestra en el Programa 5.11. Cada vez que se pulsa un botón de la interfaz, `acton` primero evalúa si la acción emitida viene de un botón y, si esto es verdadero, entonces evalúa el título del botón pulsado. Dependiendo de

esto, se invoca al método `sendData`, dando como parámetro el objeto `sClient` y el dato correspondiente para que el TPV active sus botones y se comunique con el robot.

```
public boolean action(Event event, Object what) {
 if (event.target instanceof Button){
  Button b = (Button) event.target;
  if (connected){
    if (b.getLabel() == "F1")  sendData("F1",sClient);
    else if (b.getLabel() == "F2")  sendData("F2",sClient);
    else if (b.getLabel() == "F3")  sendData("F3",sClient);
    else if (b.getLabel() == "F4")  sendData("F4",sClient);
    else if (b.getLabel() == "1")  sendData("1",sClient);
    else if (b.getLabel() == "2")  sendData("2",sClient);
    else if (b.getLabel() == "3")  sendData("3",sClient);
    else if (b.getLabel() == "4")  sendData("4",sClient);
    else if (b.getLabel() == "5")  sendData("5",sClient);
    else if (b.getLabel() == "6")  sendData("6",sClient);
    else if (b.getLabel() == "7")  sendData("7",sClient);
    else if (b.getLabel() == "8")  sendData("8",sClient);
    else if (b.getLabel() == "9")  sendData("9",sClient);
    else if (b.getLabel() == "Next")  sendData("Next",sClient);
    else if (b.getLabel() == "Prev")  sendData("Prev",sClient);
    else if (b.getLabel() == "B9")  sendData("B9",sClient);
    else if (b.getLabel() == "Break")  sendData("Break",sClient);
    else if (b.getLabel() == "Exec")  sendData("Exec",sClient);
    else if (b.getLabel() == "Enter")  sendData("Enter",sClient);
    else if (b.getLabel() == "t1+")  sendData("t1+",sClient);
    else if (b.getLabel() == "t2+")  sendData("t2+",sClient);
    else if (b.getLabel() == "t3+")  sendData("t3+",sClient);
    else if (b.getLabel() == "t1-")  sendData("t1-",sClient);
    else if (b.getLabel() == "t2-")  sendData("t2-",sClient);
    else if (b.getLabel() == "t3-")  sendData("t3-",sClient);
   }
  }
  return true;
}
```

Programa 5.11. Método de acción de eventos del *mouse* para enviar datos al *socket*.

5.4. Teleoperación desde el Mundo Virtual

Además del TPR, se desarrolló otra alternativa de conexión remota al Laboratorio Virtual de Robótica, consistente en una página WEB con el mundo virtual correspondiente al puesto de trabajo robotizado y la interfaz para su operación. En la Figura 5.3 se muestra el aspecto de dicha página WEB, cuyo mundo virtual ya fue descrito con anterioridad. Esta página permite la manipulación manual del robot virtual usando las interfaces clásicas de VRML (ratón y monitor), o la Interfaz de Usuario del proyecto ROVIR.

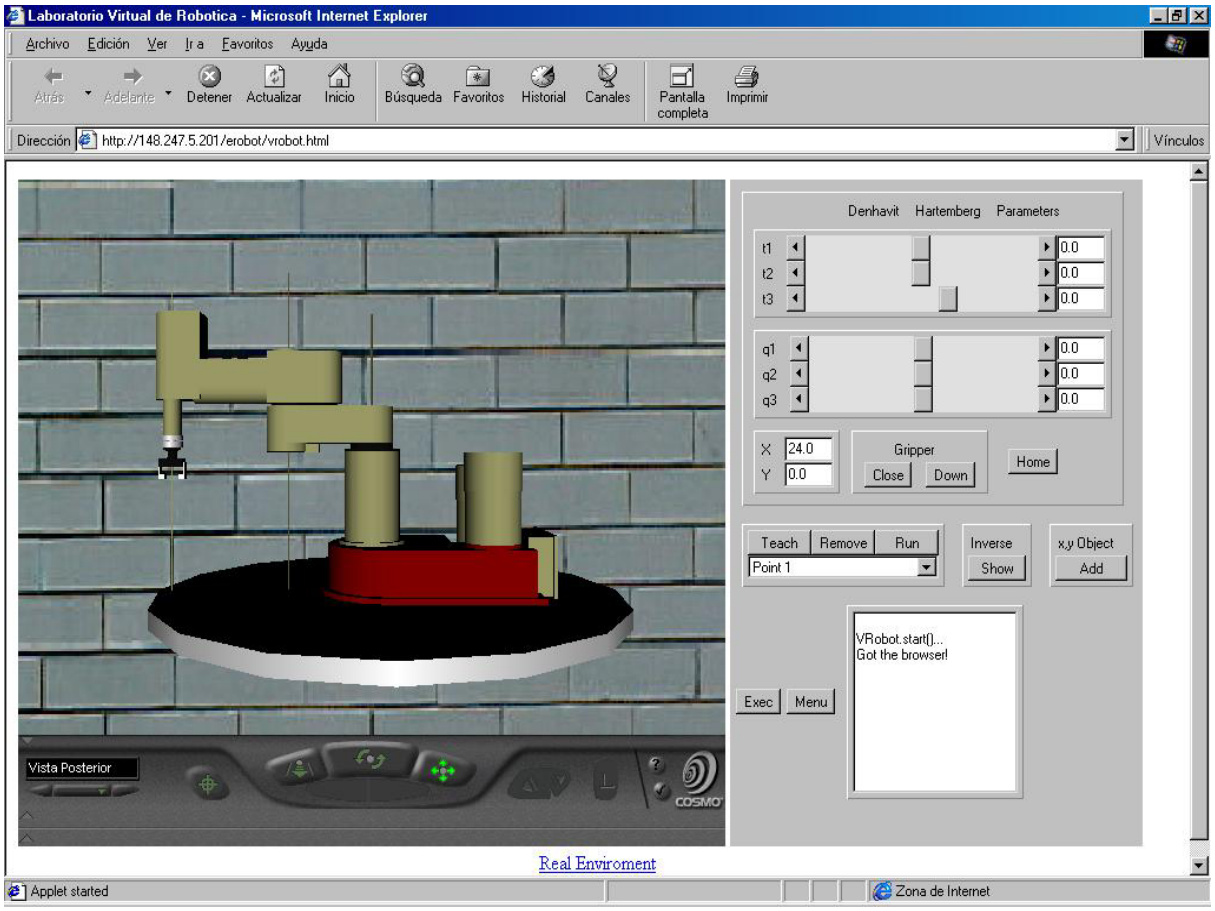

Figura 5.3. Mundo Virtual con la interfaz de teleoperación y de programación.

El mundo virtual que aparece en esta página WEB ha sido desarrollado fuera de línea en lo concerniente a la infraestructura del puesto de trabajo; pero, cuando en el puesto de trabajo real aparecen nuevos objetos a ser manipulados, un sistema de visión local los detecta, los localiza, los reconoce y genera un modelo tridimensional usando VRML. La información generada por el sistema de visión puede utilizarse para actualizar en línea al mundo virtual, siempre con base en el sistema ECI. Para actualizar el mundo virtual se cuenta con un menú de tres opciones: *i*) usar la información generada por el sistema de visión (opción: *from real workspace*), *ii*) generar objetos aleatoriamente en cuanto a cantidad y tipo (opción: *random*), *iii*) generar manualmente los objetos. Las primeras dos opciones se eligen en el menú *object* mostrado en la Figura 5.4.a, el cual se invoca pulsando el botón *menu*. En la tercera opción, el usuario remoto debe posicionar manualmente al robot en cada uno de los lugares en donde quiere que aparezca un objeto y pulsar el botón *add* para que sea creado un objeto virtual en dicha posición.

Una vez que el usuario remoto tiene la página WEB actualizada con la configuración de objetos por manipular, puede utilizarla para realizar un programa de manipulación. Para ello basta con posicionar la pinza del robot en el objeto que deseamos manipular (haciendo click sobre el sensor de tacto asociado a cada uno de dichos objetos) y llevándolo manualmente al lugar de destino deseado. Para ello se utiliza el sensor de tacto que tiene cada uno de los eslabones del robot o bien las interfaces del *frame* de teleoperación y de programación, ya sea con los *sliders* que modifican los parámetros de Denavit Hartenberg del robot (q_i o θ_i) o bien moviendo el robot con la ayuda de la interfaz basada en la cinemática inversa del robot, que puede invocarse haciendo click en el botón *show*. La pinza del robot se maneja con los botones *close*/*open* y *down*/*up*.

Figura 5.4. a) Menú de objetos y b) Menú de programas disponibles en el MUNDO VIRTUAL.

Conforme se van haciendo los movimientos del robot que constituyen la tarea deseada, el sistema va generando automáticamente el conjunto de instrucciones en lenguaje C/ROS que materializan dicho programa, factorizando las tareas repetitivas de tomar y soltar objeto en subrutinas y desplegándose en una ventana de texto para su control por parte del usuario remoto. En esta fase de utilización del Laboratorio Virtual de Robótica es posible llamar la aplicacion (*demo*) incorporada al sistema: SORT., mediante el menú *program*, mostrado en la Figura 5.4.b. El desarrollo de este demo será presentado en el siguiente capítulo.

Ya con la tarea programada, se puede proceder a su simulación en el mundo virtual (botón *run*) para su posterior envío y ejecución en el laboratorio real. El envío se hace utilizando las facilidades que otorga el TPV. Finalmente, pulsando el *link* denominado *real environment* se obtiene una ventana que permite monitorear visualmente la imagen de la *webcam* del laboratorio

real. En la Tabla 5.2 se muestran todas las posibilidades de esta versión del sistema de teleoperación y de programación mediante manipulación del mundo virtual.

1er paso	Encender el robot • Esperar la inicialización • Presionar el botón hardware *Reset*	Operador del Laboratorio
2° paso	Correr el teach pendant virtual en el servidor e–robot • Conectarse con el robot • Enviar al robot a la posición *Home* con la interfaz web	Operador del Laboratorio
3er paso	Conectarse a la página web del robot virtual	Cliente remoto
4° paso	Poner objetos en el mundo virtual (incisos opcionales): • Recibirlos del espacio de trabajo real Seleccionar la opción *from workplace* del menú de objetos • Generarlos aleatoriamente (Seleccionar la opción *Random* del menú de objetos) • Generarlos uno a uno en la posición deseada Llevar al robot a la posición deseada (mouse, sliders, etc.) Presionar el botón *Add*	Cliente remoto
5° paso	Programar al robot virtual: • Programación gestual: Agarrar objetos (con mouse) y llevarlos a la posición deseada. • Programación automática: Seleccionar la opcion de programación automática (Sort by shape) en el menú de programas de demostración	Cliente remoto
6° paso	Simular el programa en el robot virtual • Pulsar el botón *Exec* en la interfaz web	Cliente remoto
7° paso	Enviar el programa al robot (Servidor ⇒ Robot) • Seleccionar la opción *Send* del menú de programas.	Cliente remoto
8° paso	Monitorear la ejecución del programa • Seleccionar el enlace a *Real Enviromet* de la parte inferior de la página web.	Cliente remoto

Tabla 5.2. Mecanismo de programación y teleoperación vía Internet basado en el MUNDO VIRTUAL.

Capítulo 6
Aplicaciones

En los dos capítulos anteriores presentamos la implementación del Robot Virtual UNIMATE S-103 y su sistema de interacción a distancia. En este capítulo presentamos las aplicaciones desarrolladas para probar la correcta funcionalidad de los mismos.

6.1. Autoprogramación

La primera aplicación que se desarrolló fue un sistema de autoprogramación que genera un programa en el lenguaje C/ROS de manera automática siguiendo los movimientos manuales que el operador realiza en una cierta tarea de manipulación tipo *pick-and-place*. Primeramente, el usuario debe generar manualmente un conjunto de objetos virtuales por manipular o bien hacer uso de la opción que permite actualizar el espacio virtual con los objetos presentes en el espacio de trabajo del robot real. Luego se procede a manipular los objetos virtuales, operación que es convertida automáticamente en un programa C/ROS.

Para generar un objeto virtual primero se selecciona el lugar en donde se desea generarlo. Para ello se mueve el *gripper* del robot mediante alguna de las cuatro diferentes alternativas disponibles para ello: deslizadores en coordenadas articulares Denavit-Hartemberg (q), deslizadores en valores de los sensores (θ), usando el diagrama de cinemática inversa que funciona con el modelo inverso de Denavit-Hartemberg (botón *show* en la opción *Inverse* de la interfaz de usuario) o bien atrapando los eslabones del robot mediante el ratón dentro del mundo

virtual. Cada vez que se acciona una de estas alternativas, las otras tres se actualizan automáticamente con base en los modelos cinemáticos del UNIMATE S-103.

Una vez que el órgano terminal del robot ha sido ubicado en la posición deseada se puede generar un objeto pulsando con el ratón tecla *add* de la opción *xy object* en la interfaz de usuario mostrada en la Figura 3.4. Esta operación se repite cuantas veces sea necesario y, ya que han sido generados todos los objetos deseados en el espacio de trabajo del robot bastara con tocar estos objetos con el ratón uno por uno y llevarlos a la ubicación de destino deseado, para que automáticamente sea generado un programa que realiza la función *pick-and-place* correspondiente. Al final de cada operación de manipulación, en el cuadro de diálogo aparece el programa que ha sido generado automáticamente para materializar la acción de manipulación deseada.

Mediante las funciones *teach*, *remove*, *run* es posible editar o ejecutar uno de estos programas accionando los botones correspondientes una vez que ha sido elegida la línea correspondiente al comando por modificar. Finalmente, cuando se ha terminado de manipular todos los objetos es posible visualizar la acción correspondiente al programa de *pick-and-place* generado accionando con el ratón el botón *exec* dentro de la multicitada interfaz de usuario.

6.2. Clasificación Automática

En segunda aplicación, el sistema clasifica automáticamente varios objetos de diferentes clases (cubos, cilindros, prismas rectangulares y prismas de base triangular). Los objetos son generados automáticamente, aunque también podría funcionar con la opción de actualización del espacio virtual basada en visión artificial. La posición de cada uno de los objetos presentes en el espacio de trabajo virtual así como su cantidad son desconocidas por el sistema de clasificación. Una vez generados los objetos, el robot los clasifica ordenándolos en filas homogéneas, una para los cubos, otra para los cilindros y así sucesivamente. Para ello fue necesario desarrollar un módulo de autoprogramación que genera el programa de clasificación en el lenguaje C/ROS del robot.que resuelva el problema de clasificación automática.

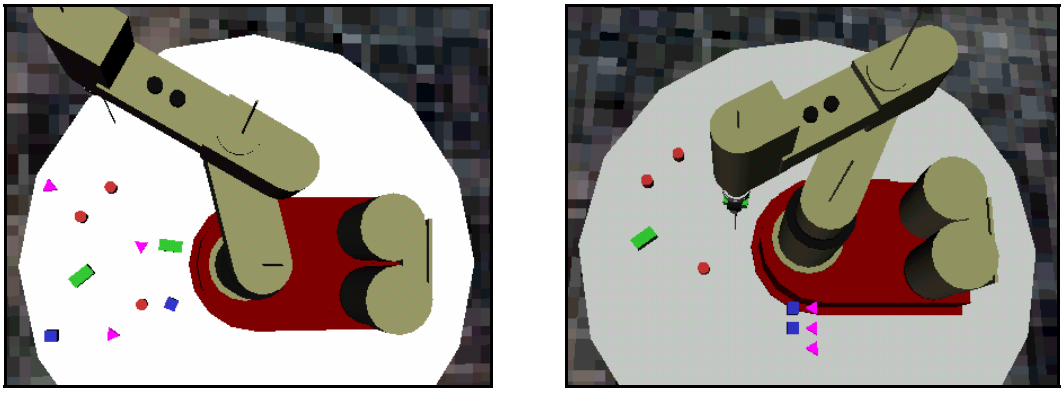

Figura 6. 1. a) Estado inicial, b) Estado intermedio

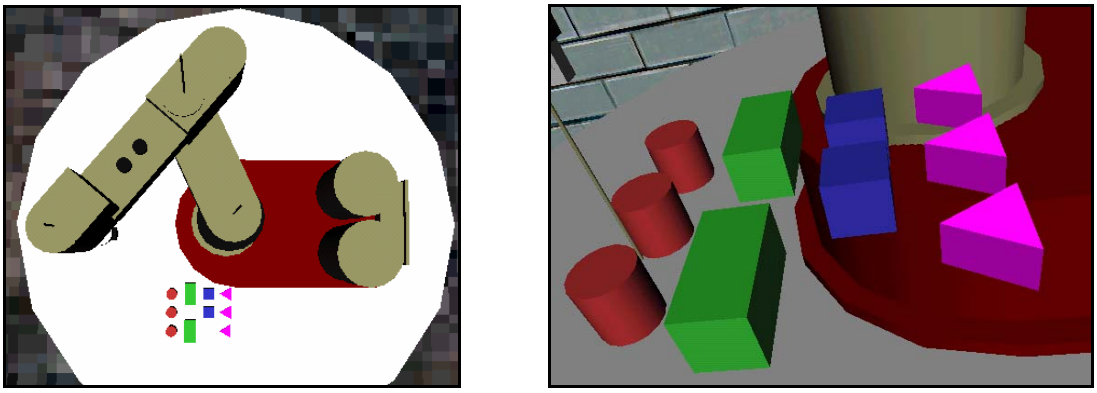

Figura 6.2. a) Estado final, b) Acercamiento del estado final

En esta aplicación, se modificó el algoritmo *Bucket Sort*. La cantidad de diferentes clases de objetos por manipular es relativamente pequeña: se tienen sólo cuatro tipos de objetos diferentes (ver Figura 5.1a). Entonces se necesitan sólo 4 cubetas, en cada una de las cuales habrá objetos de un solo tipo. Con estas características, se sabe que los rangos de elementos son: 1-1, 2-2, 3-3 y 4-4, es decir un tipo de objeto en cada clase, simplificándose de esta manera el algoritmo. Finalmente, los objetos seleccionados son manipulados por el robot y quedan debidamente clasificados y acomodados en filas del mismo tipo, tal como se muestra en las figuras Nº 6.2.a y Nº 6.2.b. En la Figura 6.1b se muestra el programa se selección en ejecución, en donde el robot solamente ha acomodado la mitad de los objetos.

6.3. Robot Móvil

En esta sección se presenta el desarrollo de una interfaz de teleoperación y percepción para el robot móvil Labmate del Laboratorio de Robótica en el Centro Politécnico Superior de la Universidad de Zaragoza en España. Dicha interfaz se presenta como un ambiente de control virtual a través de Internet, la cual comprende una representación virtual en tiempo real del robot y su espacio de trabajo, esto programado en VRML, además de una interfaz de control basada en Java *applets*. El desarrollo de este tipo de interfaces nos permite brindar una simplicidad y aceleración en el trabajo del operador del robot, y facilitar el acceso a los recursos del laboratorio desde redes TCP/IP tomando en cuenta la latencia y/o el ancho de banda de la red desde la cual se pretende acceder. El sistema desarrollado se basa en una arquitectura cliente/servidor de 3 capas o fases, lo cual se explica con un poco de detalle a continuación.

La finalidad del desarrollo de esta aplicación fue desarrollar un ambiente virtual en 3D en tiempo real, que permita representar mapas generados por los sistemas extereoceptivos del robot (láser tridimensional), los cuales recopilan información del ambiente. En el desarrollo de la aplicación se consideró que la información es recopilada en tiempo de ejecución de un programa en el robot y que se creó una interfaz de teleoperación del robot que fácil de utilizar por un operador. El operador le envía comandos de movimiento al robot, y también le solicita información odométrica sobre de la posición y orientación actual del mismo. Esta información se transmite rápidamente desde casi cualquier red, pues no se requiere de un gran ancho de banda porque los datos son representados por unos cuantos bytes. Con esta información se representa al robot en el ambiente virtual con datos reales. El operador puede también solicitarle al robot, a través de la interfaz, que le envíe los datos extereoceptivos que se pueden obtener con el láser, los cuales también se representan en el mundo virtual permitiendo la generación de los mapas del ambiente real. El problema que se tiene con esto, por el momento, es que la información proveniente del láser es muy grande, y no se puede obtener en tiempo real desde cualquier red, solo en las que se cuente con un gran ancho de banda, o bien, desde la red donde se encuentra el robot.

El motivo de esta aplicación fue la representación virtual del funcionamiento del robot. Esta aplicación presenta buen desempeño, ya que no se requiere de cámaras para estar viendo al robot en su espacio de trabajo, lo cual requeriría de un mayor ancho de banda del que se

encuentra a nuestro alcance. El control del robot vía Internet para facilitar el trabajo del operador es una motivación que también fue considerada muy importante para el proyecto. Como se mencionó anteriormente, el sistema ha sido probado en el robot móvil Labmate y ha mostrado una gran eficiencia en su uso para el trabajo del operador. A continuación se presenta la arquitectura del sistema desarrollado, una explicación del desarrollo de los módulos del sistema y los resultados experimentales del funcionamiento en el robot Labmate.

6.3.1 Arquitectura de la aplicación

La arquitectura del hardware que se utilizó y del software que se desarrolló para la creación de esta aplicación, así como el flujo de la información entre las 3 capas o fases del sistema cliente/servidor, que se ilustran en la Figura 6.3.

6.3.1.1. Hardware y Software disponible

En el Laboratorio de Robótica de la Universidad de Zaragoza se cuenta con un robot móvil Labmate, integrado con sensores propioceptivos que permiten medir el estado interno del robot (odométria), además de sensores extereoceptivos utilizados para recopilar información del ambiente (Ultrasonido, cámaras de video, láser tridimensional). Este robot cuenta con una computadora integrada que permite ser accedida a través de TCP/IP. La computadora cuenta con el sistema operativo Unix (Solaris). Aquí se ejecuta un programa servidor que tiene comunicación directa con los sensores del robot y también envía directamente comandos de movimiento al robot. Este programa se queda indefinidamente esperando ser conectado por el cliente.

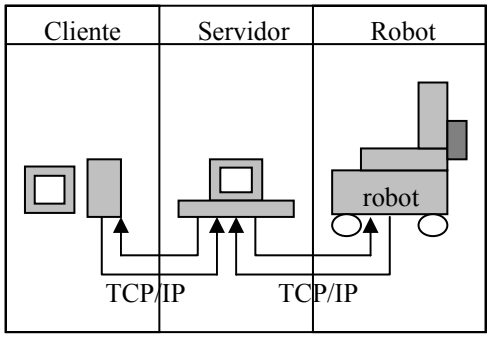

Figura 6.3. Esquema de comunicación cliente/servidor de 3 capas.

El servidor, accesible desde Internet, es una computadora Sun Sparc con sistema operativo Unix (Solaris) que está conectada a la misma red que el robot. Este servidor permite ser accedido vía WEB y envía los programas necesarios al cliente remoto como *applets* de Java. En este servidor se ejecuta el programa cliente/servidor que se encarga de hacer la negociación entre el cliente remoto y el robot. Funciona como servidor para atender al cliente remoto, y como cliente para conectarse con el servidor (robot) y enviar o solicitar la información que requiere el cliente.

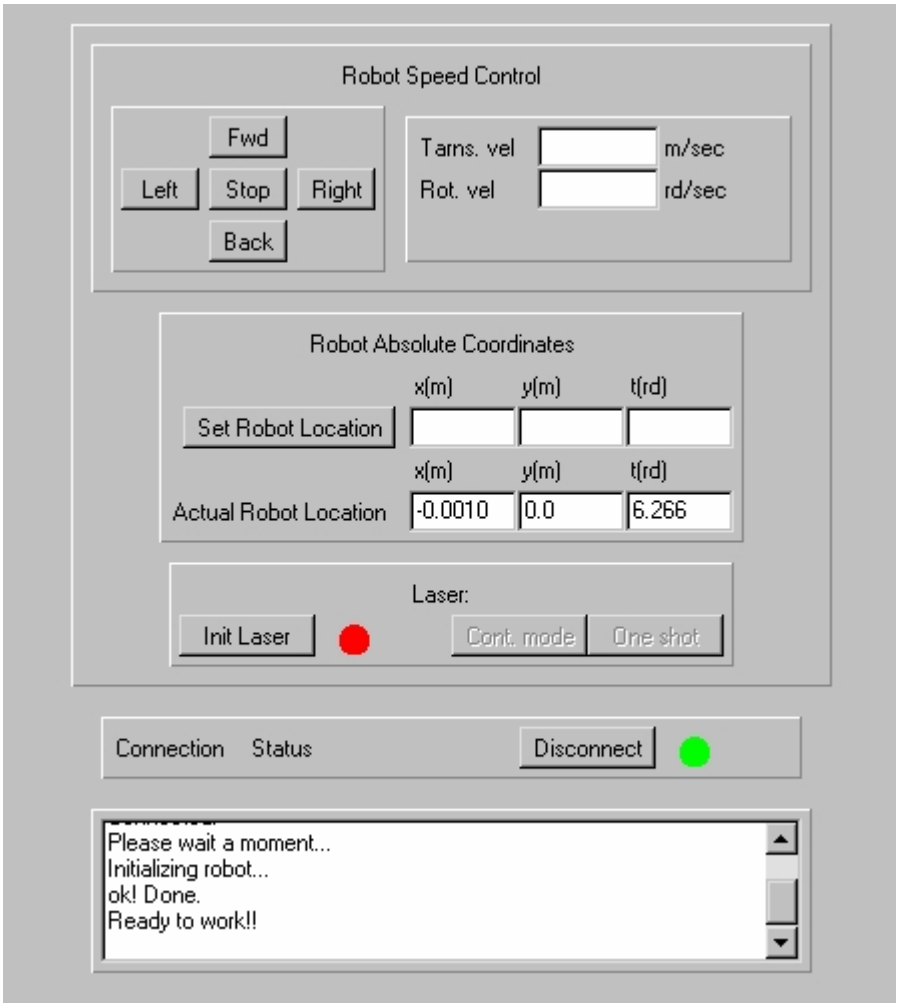

Figura 6.4. Interfaz de teleoperación del robot móvil.

6.3.1.2. Flujo de comunicación

El flujo de comunicación de los datos, desde que el cliente solicita conectarse al robot para iniciar la teleoperación, se explica a continuación. El usuario se conecta al servidor a través

de la interfaz de teleoperación del robot, mostrada por la Figura 6.4. Una vez que el usuario ha solicitado una conexión con el servidor, el servidor envía información indicando la aceptación del usuario, la cual se despliega en la ventana de estado de la interfaz. El servidor, al atender al usuario, se conecta inmediatamente con el servidor que se está ejecutando en el procesador del robot. Cuando esto sucede, el robot esta listo para iniciar su funcionamiento.

Desde la interfaz, el usuario puede manipular completamente al robot. Por ejemplo, puede enviarle comandos de movimiento o solicitar datos odométricos para conocer su posición, o bien, indicarle al robot que tome una lectura láser y usar esos datos para representar los objetos en el mundo virtual.

Cuando el usuario envía un comando de movimiento desde la interfaz, la información se va al servidor principal, el cual lo transmite al robot para que lo ejecute. Al ejecutar el comando, el robot cambia su posición, la cual es posible conocerla recibiendo datos odométricos. Estos datos se están recibiendo en todo momento ya que se solicitan en un ciclo infinito. Al recibir estos datos odométricos con la información de la nueva posición y orientación del robot, la interfaz establece una comunicación con el mundo virtual para que el robot sea posicionado en su nuevo lugar y con su nueva orientación.

El lanzamiento del láser se solicita también desde la interfaz de aplicación, en dos modos: continuo y sencillo. Cuando se hace un lanzamiento sencillo desde la interfaz, el servidor recibe el identificador correspondiente para que se realice este procedimiento, y el servidor lo envía al robot, el cual da como respuesta un conjunto de datos, los cuales representan la distancia a la que se encuentra un objeto en un ángulo determinado. Al llegar los datos a la interfaz de teleoperación, esta los envía al mundo virtual, el cual representa cada uno de los puntos en forma de cilindros de una altura predeterminada.

De esta manera se puede manipular al robot desde cualquier ubicación, simplemente teniendo acceso a Internet. Con los datos odométricos se puede saber en donde se encuentra el robot y visualizarlo en el mundo virtual, como si se tratara del real.

6.3.2. Modelado del ambiente virtual en 3D

La representación virtual del robot móvil, como se mencionó anteriormente está desarrollada en VRML, la cual no requirió de gran complejidad al programarla de manera similar

a la usada con el robot industrial UNIMATE S-103, pues el robot móvil virtual no es manipulado en modo de simulación, sino que simplemente es posicionado en el lugar que indiquen los datos odométricos provenientes de la interfaz en Java. El mundo virtual desarrollado se puede apreciar en la Figura 6.5.

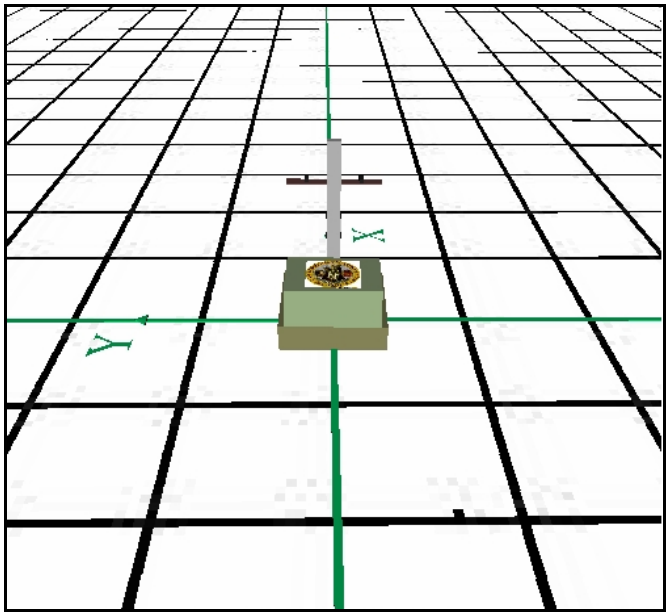

Figura 6.5. mundo virtual del robot móvil.

Modelos en VRML. El robot móvil fue desarrollado de una manera sencilla dado que así fue requerido por su funcionalidad. Simplemente se creó un Nodo principal, y partiendo de éste se desarrollaron todos los nodos que lo conforman como hijos. La única parte de programación en VRML que es diferente a la utilizada en el robot UNIMATE S-103 es la que crea los hijos en tiempo de ejecución. Al representar los nodos correspondientes a los puntos tomados por el láser, se descartó la manera de crearlos desde Java, ya que la cantidad de puntos tomados es importante (480 puntos). Esto requeriría muchas comunicaciones desde Java (480 comunicaciones), lo cual tomaría una gran cantidad de tiempo, haciendo imposible apreciar la teleoperación en tiempo real. Por lo tanto, Java hace sólo una comunicación con VRML, enviándole un arreglo con los 480 puntos, y el programa en VRML se encarga de hacer a creación automática de los puntos de una manera más viable y rápida.

6.3.3. Resultados experimentales

A continuación se presentan los resultados experimentales obtenidos haciendo una teleoperación con el robot móvil.

En el experimento realizado se teleoperó al robot real utilizando la interfaz desarrollada. El operador del robot movió al robot real utilizando la interfaz a través de un mapa que se fue generando en la misma interfaz. En la Figura 6.6 se puede observar al robot real en el área de trabajo donde se desea hacer la navegación, y se muestra el resultado del primer lanzamiento de rayo láser, modelando los puntos tomados en el espacio virtual. En la Figura 6.7, el robot fue girado para obtener los datos del mapa que se desea generar, hasta donde los sensores del robot puedan detectar.

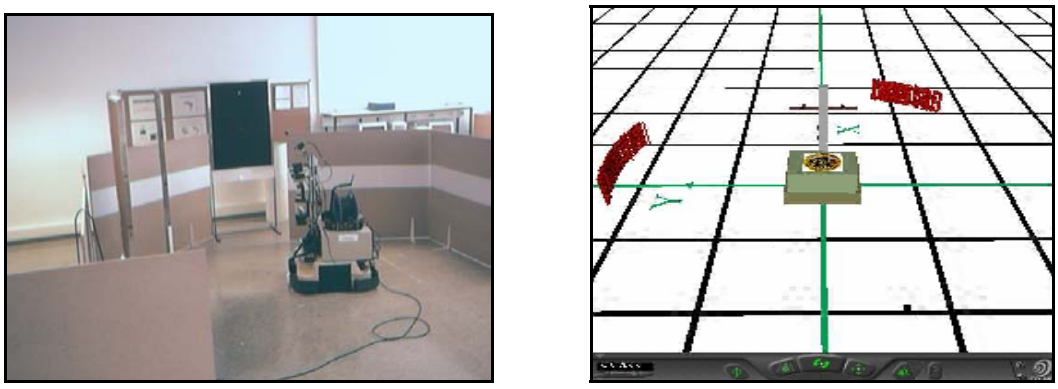

Figura 6.6. Primera detección de objetos con el láser.

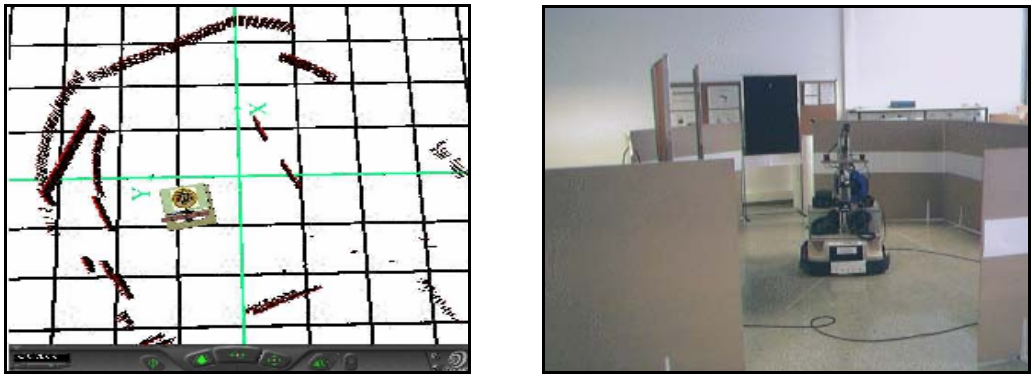

Figura 6.7. Obtención de todo el entorno real. Giro completo.

El robot es manipulado para pasar a través de una puerta que se observa en el mapa virtual, la cual existe en el espacio real de trabajo del robot. Esto se observa en la Figura 6.8.

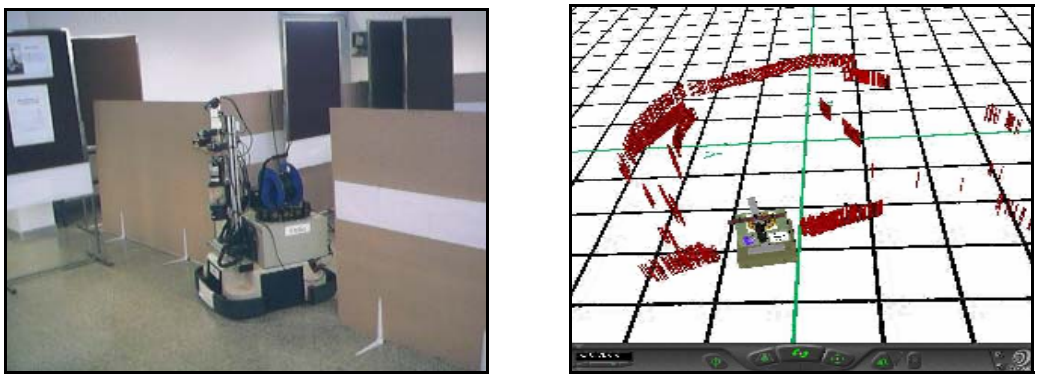

Figura 6.8. Cruzando la puerta.

Finalmente, en la Figura 6.9, el robot ya se encuentra del otro lado del mapa inicial generado, y aun sigue construyendo mapas por si se desea continuar con la teleoperación. También se observa el mapa final generado en el espacio virtual.

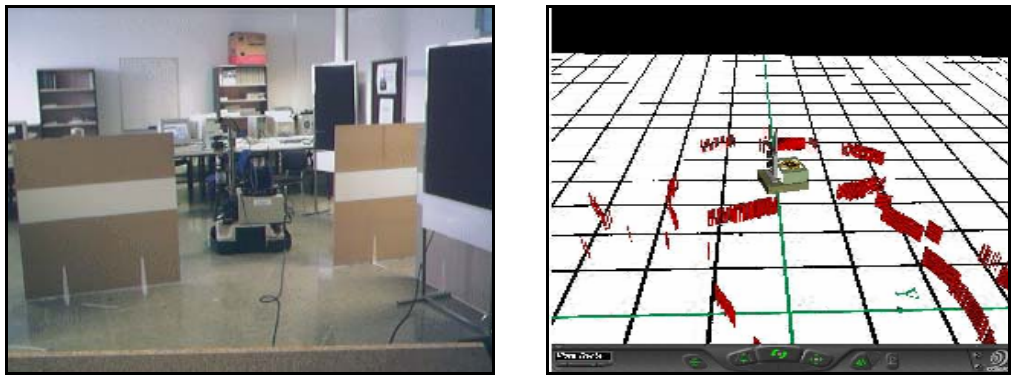

Figura 6.9. Del otro lado de la puerta.

Capítulo 7
Conclusiones y Trabajos Futuros

7.1 Conclusiones

Esta tesis ha presentado un sistema de teleoperación y programación automática de tareas de manipulación para un robot UNIMATE S-103 basado en mundos virtuales, todo ello vía Internet. La solución implementada garantiza la manipulación de los mundos virtuales creados con VRML, basada en la *External Communication Interface*, método de comunicación con programas en Java. Gracias a esta herramienta se pudieron desarrollar mundos virtuales interactivos de gran utilidad en muchas aplicaciones no sólo de Robótica. Otra característica de interés es el uso de un *teach pendant* virtual para garantizar la comunicación vía Internet entre el mundo virtual y el robot.

Se crearon dos prototipos de Laboratorio Virtual, uno para un robot manipulador industrial, el UNIMATE S-103, y otro para un robot móvil comercial, el LABMATE. Para el caso del prototipo del robot manipulador se desarrollaron aplicaciones que hacen uso de técnicas de visión artificial y de programación automática. El prototipo correspondiente al robot móvil LABMATE se desarrolló en la Universidad de Zaragoza, España.

Cabe mencionar que el prototipo desarrollado para el LABMATE se desarrolló cuando el del Unimate S-103 ya estaba terminado. Así, su desarrollo simplemente requirió reprogramar los modelos matemáticos del robot LABMATE, la creación de un mundo virtual más adecuado y unas pequeñas modificaciones a la interfaz de usuario relativas a la cinemática de este robot.

Las principales limitaciones del sistema se deben a las características del robot industrial utilizado pues está basado en tecnologías de hace casi 20 años y a la poca capacidad de Internet para manejar imágenes de ambiente suficientemente detalladas y con una cadencia adecuada para transmitir el movimiento del robot. A pesar de sus limitaciones, los desarrollos producidos en este trabajo de tesis están siendo usados en varios proyectos de *Internet Robotics* que van desde robots para aplicaciones médicas (laparoscopía robotizada) hasta la creación de mundos virtuales interactivos de gran realismo.

Finalmente, uno de los principales resultados de haber trabajado en el desarrollo de este proyecto fue la oportunidad de desarrollar un trabajo multidisciplinario y en equipo, lo cual aportó una componente muy importante de formación profesional.

7.1.1. De la Cinemática del Robot y del Sistema de Visión

Se hizo un estudio de cinemática de robots, tanto directa como inversa para lograr el modelado matemático del movimiento del robot dentro del mundo virtual, así como el control de restricciones cinemáticas en las aplicaciones *Java*. La cinemática también fue utilizada en la manipulación manual del robot virtual en las fases de enseñanza, programación y ejecución. Este es uno de los principales aspectos que garantizan la interactividad adecuada del robot virtual.

La profundización en el estudio de la cinemática del robot permitió que se consiguiera un modelo virtual muy eficiente, así como la planeación en la construcción del mundo virtual para que pudiera ser interactivo con el ambiente exterior. El conocimiento de los parámetros de Denavit-Hartemberg fue de gran ayuda en el diseño, ya que los datos que se obtienen de este procedimiento permiten visualizar y comprender de una mejor manera el comportamiento de un robot, principalmente el utilizado en este caso.

Se estudió la cinemática de los sistemas de visión, con énfasis en los procesos de calibración, para poder determinar los modelos geométricos VRML con que serían codificados los objetos detectados y reconocidos por el sistema de visión desarrollado en el marco de otra tesis de maestría paralela a la presente [Araujo 02]. Si bien aquí no interesan problemas de procesamiento de imágenes ni de reconocimiento, los aspectos cinemáticos ligados a la adquisición de imágenes y a la calibración del sistema de visión son de interés y utilidad en el modelado geométrico de los objetos 3D presentes en la imagen.

En la tesis de Maestría de David Araujo Díaz, se reconocen objetos en el espacio de trabajo del robot real, los cuales son depositados en un servidor como archivos de texto; estos archivos son leidos desde el sistema desarrollado en esta tesis, y a su vez representados en el mundo virtual como objetos tridimensionales. De esta manera, los objetos existentes en el mundo real son representados en el mundo virtual.

7.1.2. Del Software Utilizado y Desarrollado

La organización de los servidores que fue elegida y desarrollada en este trabajo se considera que fue la más apropiada para tener un buen nivel de seguridad en el acceso de los usuarios al proyecto, ya que la estructura modular con que fue desarrollado el sistema permite bloquear algunos de estos módulos sin que deje de funcionar todo el sistema, pues simplemente se puede eliminar el acceso al módulo correspondiente.

El software utilizado para configurar las plataformas así como los lenguajes de programación empleados permitieron que el desarrollo del sistema se lograra satisfactoriamente, tal como fue planteado. Por tal motivo se recomienda que en futuros trabajos de *Internet Robotics* se continúe con la utilización de todos ellos. El uso de software comercial, como el que fue utilizado para transmitir imágenes por Internet con el uso de *applets*, fue lo que se consideró más apropiado, ya que no hubo necesidad de preocuparse por el desarrollo de este módulo del sistema, sino que simplemente se implementó y configuró a las necesidades del proyecto.

Todos los módulos del proyecto se desarrollaron utilizando plataformas *windows*, en particular la versión *Millenium Edition*, pero esto no implica que se necesite usar esta plataforma para acceder al laboratorio. También puede ser accedido desde casi cualquier plataforma, simplemente contando con un *Plug-In* capaz de visualizar programas de VRML y una máquina virtual de Java (JVM) para visualizar los *applets*. El *Plug-In* utilizado fue el CosmoPlayer 2.1, y la JVM fue la que se distribuye con el *Java Runtime Environment* 1.3.1_01 de *Sun Microsystems*.

El que todo haya sido implementado en plataformas *windows*, no significa que sea la manera más viable para el desarrollo de este proyecto. Otros sistemas operativos tipo *Unix*, como *Linux* entre otros, pueden ser mas estables al funcionar como servidores.

La portabilidad del sistema a alguna de estas otras plataformas es relativamente sencilla, ya que algunos de los programas que funcionan como servidores están desarrollados en *Java*, y

por la portabilidad que ofrece, simplemente ejecutando las *Clases* en estas plataformas, el sistema seguiría funcionando. También se considera que si la aplicación *teach pendant* virtual se desarrolla en *Java*, en lugar de estar funcionando en *Visual Basic*, lograría que el sistema funcionara de manera más estable.

La utilización de VRML para la construcción de los mundos virtuales fue la mejor alternativa, ya que su conectividad con otros programas a través de la *External Communication Interface* (ECI), por ejemplo con Java, es completa. Una de sus principales ventajas es que estos lenguajes, VRML y Java, permiten un fácil acceso a través de plataformas WEB, lo que resulta de gran utilidad en el desarrollo de proyectos de *Internet Robotics* como el nuestro.

El uso de la ECI es una de las partes claves del proyecto, ya que constituye el núcleo de comunicación bidireccional del mundo virtual con el exterior, lo que permite considerarlo como realidad virtual, pues los datos con los que se actualiza el mundo virtual provienen de la realidad. Java permite hacer uso de la ECI para comunicarse con VRML, y esto nos da un control completo del mundo virtual, pues la robustez del lenguaje Java permite hacer casi cualquier cosa en programación.

El *teach pendant* virtual utilizado en el Laboratorio Virtual de Robótica es más eficiente que el real, ya que en la PC donde reside se cuenta con todo el poder de un lenguaje de programación con lo cual se tienen menos limitaciones que en la botonera *hardware*. En efecto, es posible expandir el funcionamiento de la versión virtual logrando optimizar el uso de tareas repetitivas, controlar eficientemente errores y darle portabilidad con respecto a otros sistemas.

En este trabajo se desarrollaron dos mundos virtuales en VRML, uno que representa al robot industrial SCARA UNIMATE S-103, y otro que concierne al robot móvil LABMATE. Ambos permiten una interacción del usuario con el mundo virtual a través de *scripts* en *Java*, pero desarrollando una interfaz de control como un *applet* en Java. Esta interfaz de control se comunica con el mundo virtual a través de la ECI. La estructura del sistema diseñado permite un mejor control en el manejo de variables en memoria y restricciones cinemáticas con el robot y, a su vez, permite la comunicación mediante *sockets* con servidores conectados de manera más directa con los robots para lograr la Teleoperación.

7.2. Trabajos en Curso y Futuros

Dentro del proyecto ROVIR, y con base a los desarrollos producidos en el marco de esta tesis, se están desarrollando aplicaciones médicas del robot industrial UNIMATE S-103 en el ámbito de la cirugía laparoscópica, fácilmente transportables a robots diferentes, especialmente a aquellos que hayan sido especialmente construidos para tales aplicaciones. Una de estas aplicaciones pretende el desarrollo de un sistema de control automático que transforme comandos de movimiento simples (adentro-fuera, arriba-abajo, izquierda-derecha) emitidos por el cirujano en movimientos coordinados complejos de un robot equipado con el laparoscopio que genera las imágenes de televisión con las que el cirujano realiza la intervención en la cavidad abdominal del paciente. Contamos con un prototipo virtual en el cual se aplican los esquemas de control mencionados y que puede ser operado a través de interfaces en Java, o bien teleoperado vía Internet mediante las mismas interfaces desarrolladas en el marco de esta tesis. En este proyecto, la parte relativa a la cinemática de robots redundantes y a la solución del sistema de control del robot laparoscopista constituye la tesis de maestría de Braulio Samuel Colmenero Mejía del Departamento de Control Automático del CINVESTAV.

En el ámbito de los robots móviles, en este proyecto se desarrolla un sistema de generación automática de mapas virtuales tridimensionales en VRML a partir de visión monocular. Utilizando las imágenes 2D generadas por la cámara embarcada en un robot móvil se hace la detección de las líneas rectas que constituyen los planos del entorno (piso y paredes); una vez encontrados dichos planos se trata de construir la transformación de perspectiva inversa que permite obtener una imagen rectificada del entorno susceptible de ser modelada mediante VRML, para generar automáticamente una representación tridimensional en forma de un mundo virtual interactivo accesible por Internet. Los aspectos de visión artificial relativos a la obtención de modelos tridimensionales a partir de imágenes monoculares constituyen la tesis de maestría de Eduardo Jesús Iturbe Córdova del Departamento de Control Automático del CINVESTAV. Todos los aspectos de construcción de mundos virtuales interactivos a partir de la información visual procesada en este proyecto se basan en los resultados de esta tesis.

En los proyectos mencionados en los párrafos anteriores, los robots, manipuladores o móviles, se simulan dentro del mundo virtual tomando en consideración exclusivamente la geometría de los puestos de trabajo incluyendo al robot así como la cinemática de sus

movimientos, obteniéndose una representación virtual bastante realista gracias a las técnicas de VRML y a la interactividad de las interfaces ECI programadas en Java en esta tesis. Pero, para aplicaciones del Laboratorio Virtual de Robótica en donde se requiere de una inmersión realista total esto es insuficiente. En efecto, para el desarrollo y evaluación de leyes de control, para realizar tareas de telemanipulación con retorno de esfuerzos y en muchas aplicaciones de teleoperación de robots móviles en ambientes desconocidos o aún en aplicaciones médicas es necesario que el mundo virtual posea características dinámicas regidas por ecuaciones diferenciales y que los objetos manipulados puedan reflejar dichas características sobre la interfaz mecánica del usuario.

En esta línea de investigación y desarrollo se encuentra el proyecto de tesis de maestría que realiza Miguel Ángel Zúñiga, estudiante del Departamento de Control Automático del CINVESTAV, cuyo objetivo es el desarrollo de una plataforma de simulación dinámica para un robot móvil que permita el ensayo y evaluación de leyes de control visuomotor, en donde la salida de información se hace a través de un mundo virtual, el cual puede ser teleoperado mediante una interfaz háptica comercial (*joystick* de Microsoft). En un proyecto más ambicioso, Claudia Marmolejo Rivas realiza su tesis doctoral en ese mismo departamento cuyo objetivo es el desarrollo de un ciclo háptico completo que incluye la visualización virtual realista, el cálculo de colisiones, cálculo de la deformación mecánica debida a la interacción del usuario con los objetos virtuales, el cálculo de las fuerzas de interacción correspondientes y la generación de las fuerzas de retroalimentación que convierten al sistema de simulación virtual en un ambiente realista inmersivo.

Los trabajos que se están realizando para mejorar el prototipo consisten en sincronizar el mundo virtual con el mundo real a fin de utilizar este último para monitorear el comportamiento del robot real, compensando de esta manera la baja capacidad en cuanto a transmisión de imágenes. También se está diseñando la versión Internet II del Laboratorio Virtual de Robótica, en donde se tendrá un ancho de banda suficiente para transmitir imágenes de ambiente con calidad, para una teleoperación que incluya aspectos de control y para aplicar nuestro sistema a toda una celda de ensamble, así como utilizar robots con tecnologías más recientes.

Bibliografía

[Alarcón 00] De Alarcón Alvarez, E. y Parés I. Burgués, N., **"Manual Práctico de VRML 2.0"**, Ed. Biblioteca Técnica de Programación, España, 2000, 187pp.

[Araujo 02] Araujo, Díaz, David, **"Sistema de Visión Artificial para el Laboratorio de Robótica Virtual"**, Cinvestav, México, 2002, 87pp.

[Bruno 98] Preiss, Bruno R. **"Data Structures and Algorithms with Object-Oriented Design Patterns in C++"**, Ed. WILEY, Canadá, 1998, 651pp.

[Chellali 97] R. Chellali**, "Image Understanding for Telepresence Applications"**, ISMCR'97-IMEKO, Tampere, Finland , june1997, 276-283pp.

[Chellali 98.1] Ryad Chellali, **"Visual feedback bandwidth reduction for telerobotics applications"**, IEEE-SMC, CESA'98 , Tunisie, 1998.

[Chellali 98.2] R. Chellali, **"Visual feedback enhancement for telerobotics applications"**, SPIE Aerosense , USA, 1998

[Chellali 01] Chellali, R. y Belousov, Igor R., **"Virtual Reality Tools for Internet Robotics"**, Proc. of the IEEE International Conference on Robotics & Automation ICRA'2001, 2001, Korea, 1878–1883pp.

[Cormen 90] Cormen, Thomas H., Charles E. Leiserson y Ronald L. Rivest, **"Introduction to Atgorithms"**, Ed. MIT Press y McGraw-Hill, 1990, 1025pp.

[Daltons] Daltons, Barney, Internet: http://barney.daltons.net

[Deitel 98] Deitel, H.M. y Deitel, P.J., **"Como programar en Java"**, Ed. Prentice-Hall, México, 1998, 1056pp.

[Gordillo 98] Gordillo, José Luis, **"Proyecto LVRM: Laboratorio Virtual de Robótica y Manufactura"**, Revista Digital *Transferencia*, ITESM Campus Monterrey, México, 1998.

[Heileman 98] Heileman, Gregory L., **"Estructuras de Datos, Algoritmos y Programación Orientada a Objetos"**, Ed. McGraw-Hill, 1998, 305pp.

[Herrera 00] Herrera Barragán, José Andrés, Murguía Alcalá, Rodrigo Francisco y Velarde Ruiz, Jorge, **"ROVIR: Laboratorio Virtual de Robótica",** Reporte Final del Verano de la Investigación Científica AIC-CONACYT; Programa Delfín, México, DF., Agosto, 2000.

[Ibarra 97] Ibarra Zannatha, Juan Manuel, Zaldívar Colado, Ulises, Sastré Leyva y Luis Manuel. **"Hacia un Laboratorio Virtual de Robótica",** Memorias del Simposium Internacional de Computación Nuevas Aplicaciones e Innovaciones Tecnológicas en Computación. México, D.F., Noviembre, 1997.

[Ibarra 00.1] Ibarra Zannatha, Juan Manuel, Murguía Alcalá, Rodrigo Francisco, Herrera Barragán, José Andrés y Velarde Ruiz, Jorge, **"Uso de un Robot Virtual para la Teleoperación de un Robot Industrial"**, Memorias del Segundo Congreso Nacional de Robótica de la AMRob, Instituto Tecnológico de Toluca, Toluca, Edo. Mex., Septiembre, 2000, 53-62pp.

[Ibarra 00.2] Ibarra Zannatha, Juan Manuel, Zaldívar Colado, Ulises, Murguía Alcalá, Rodrigo Francisco, Herrera Barragán, José Andrés y Velarde Ruiz, Jorge, **"Virtual Robotics: A powerfull tool for tele-education"**, Workshop on Advances in Artificial Perception and Robotics AAPR'2000, CIMAT, Guanajuato, México. Octubre, 2000.

[Ibarra 01] Ibarra Zannatha, Juan Manuel, Zaldívar Colado, Ulises y Araujo, Díaz, David, **"Robótica Virtual y Teleoperación de un Robot Industrial"**, Reunión de Primavera de la Corporación Universitaria para el Desarrollo de Internet, México, D.F., Abril, 2001.

[Internet 1] **"The Java Tutorial: Third Edition. A practical guide for programmers"**. Internet: http://java.sun.com/docs/books/tutorial/

[Internet 2] Omnicron Technologies Corporation. Internet: http://www.omnicron.ca/

[Internet 3] Surveyor Corporation. Internet: http://www.surveyor.com/index_webcam32.html

[Jochheim 00] Jochheim, Andreas, Gerke, Michael y Bischoff, Andreas, **"Modeling and simulation of robotic systems"**, RichOLD Project, Hagen, Germany, 2000. Internet: http://icosym-nt.cvut.cz/odl/partners/fuh/course_main/

[Kheddar 97.1] Kheddar, A., Tzafestas, C., Coiffet, P., Kotoku, T., Kawabata, S., Iwamoto, K., Tanie, K., Mazon, I., Laugier, C. y Chellali, R., **"Parallel multi-robots long distance teleoperation"**, Proceedings of the 8th International Conference on Advanced Robotics, ICAR '97, 1997, 1007 –1012pp.

[Kheddar 97.2] Kheddar, A., Chellali, R. y Coiffet, P., **"Implementation of Head-Behaviour Based Control for Navigation Within Virtual Reality Applications"**, Proceedings of the IEEE International Workshop on Robot and Human Interactive Communication, ICAR '97, 1997, 1007 –1012pp.

[Lemay 96] Lemay, L. y Perkings, C. L., **"Aprendiendo Java en 21 días"**. Simon & Schuster Company. Ed. Prentice-Hall, México, 1996, 525pp.

[Malo 93] Malo Tamayo, Alejandro, **"Robot Unimate Serie 100 Modelo 130"**, Ed. Cinvestav, México, 1993, 42pp.

[Marrin 97] **Marrin, Chris,** "Proposal for a VRML 2.0 Informative Annex External Authoring Interface Reference"**, Silicon Graphics, Inc., 1997. Internet: http://www.web3d.org/WorkingGroups/vrml-eai/history/eai_draft.html**

[Rohrmeier 97] Rohrmeier, Martin, **"Telemanipulation eines Roboters via Internet mittels VRML 2.0 und Java"**, Tesis doctoral, Deutsches Zentrum für Luft- und Raumfahrt, Alemania, 1997.

[Rudomín 97] Rudomín, Isaac y Enrique Martínez, **"Simulación en Red de una Celda de Manufactura Compartida"**, Primer Encuentro de Computación ENC '97, Universidad Autónoma de Querétaro, Querétaro, México, Septiembre de 1997.

[Sucar 01] Sucar, Luis Enrique, **"Laboratorio Virtual de Robótica Móvil"**, 1er. Taller de Computación "Robótica y Visión", ITESM Campus Cuernavaca, Morelos, México, Noviembre de 2001.

[Taylor] Taylor, Ken, University of Western, Australia, Internet: http://telerobot.mech.uwa.edu.au

[Taylor 97] Taylor, Ken, Dalton, Barney, **"Issues in Internet Telerobotics"**, Proc. of the FSR'97 International Conference on Field and Service Robotics, Canberra, Australia, December 1997.

[Unimate 85] **"Unimate Industrial Robot Series 100. Equipment Manual and Programming Manual (Draft)"**, Westinghouse Company, Danbury, Conneticut, USA, 1985.

[Vogel] Vogel, Joerg, **"Web Resources, Vogel Joerg homepage"**, Internet: http://www.robotic.dlr.de/Joerg.Vogel/

[VRML 96] Sylicon Graphics, **"Practical Handbook of VRML 2.0"**, Biblioteca Técnica de Programación, 1996, 185pp.

[VRML 97] **"The Virtual Reality Modeling Language. International Standard ISO/IEC 14772-1:1997",** Copyright © 1997 The VRML Consortium Incorporated. Internet: http://www.vrml.org/technicalinfo/specifications/vrml97/index.htm

[Zaldívar 01] Zaldívar Colado, Ulises, **"Uso de un Robot Virtual para la Teleoperación de un Robot Industrial",** Memorias del Congreso Internacional de las Ciencias Computacionales CICC'2001, Instituto Tecnológico de Colima, Colima, México, Marzo, 2001.