81
29.

UNIVERSIDAD NACIONAL AUTONOMA DE MEXICO

FACULTAD DE INGENIERIA

DISEÑO, VISUALIZACION Y CONTROL DE ROBOTS UTILIZANDO HERRAMIENTAS DE REALIDAD VIRTUAL Y ARQUITECTURAS CLIENTE SERVIDOR

T E S I S

QUE PARA OBTENER EL GRADO DE:

INGENIERO EN COMPUTACION

P R E S E N T A :

HERNANDO ORTEGA CARRILLO

DIRECTOR: ING. JOSE CASTILLO HERNANDEZ

MEXICO, D.F. OCTUBRE DE 1998

UNAM – Dirección General de Bibliotecas Tesis Digitales Restricciones de uso

Quiero dedicar este trabajo
a mis padres
Blanca y Everardo
por todo su cariño y apoyo.

Agradecimientos

A mis padres y familiares por brindarme siempre todo su apoyo y cariño.

Al Ingeniero José Castillo Hernández por su amistad y asesoría, pero sobre todo por su paciencia durante el desarrollo de este proyecto. A los Ingenieros Wilfredo Martínez Payán y Miguel Ángel Bañuelos por sus valiosos comentarios y observaciones. Al Dr. José Luís Pérez Silva por apoyar, no solo este proyecto, si no a todos aquellos que se desarrollan en el Laboratorio de Electrónica del Centro de Instrumentos.

A mis amigos Norberto Ortigoza y Andrés Pineda por toda su colaboración en este trabajo, por compartir conmigo sus experiencias, conocimientos e inquietudes. Al Ingeniero Juan José Carreón Granados por su apoyo a lo largo de la carrera, permitiéndome incrementar mis conocimientos y adquirir el gusto por la investigación.

A la Dra. Hanna Oktaba por su apoyo al permitir el desarrollo de gran parte de este trabajo en las instalaciones a su cargo de la Maestría en Ciencias de la Computación.

A todos mis amigos.

Contenido

Introducción

Una de las características que ha distinguido al hombre desde sus inicios, es su capacidad de crear herramientas que faciliten, e incluso hagan posible, el desarrollo de todas sus actividades. En su búsqueda por mejores herramientas ha acumulado una gran cantidad de conocimientos y los ha encauzado a través de la ciencia de la robótica, la cual plantea, el desarrollo y uso de máquinas "inteligentes" capaces de asistir al hombre en sus actividades sirviendo como una extensión de sus aptitudes.

En el laboratorio de Electrónica del Centro de Instrumentos se han desarrollado proyectos de implementación y control de robots, desde 3 hasta 5 grados de libertad. Estos trabajos han enriquecido la experiencia de su personal académico, lo cual permite el planteamiento de proyectos de mayor complejidad con fines de investigación. Actualmente existe el proyecto de implementacion de un robot móvil denominado como "insectoide" con la capacidad de desplazarse a través de terreno irregular. El robot en cuestión utilizará 6 extremidades sensibles al terreno para su desplazamiento, cada una de las cuales es en si misma un manipulador independiente de 3 grados de libertad. A bordo del robot un subsistema compuesto por una pequeña cámara de video y un dispositivo de 2 grados de libertad para su orientación serán utilizados para la adquisición de información visual del medio que lo rodea. Dada la complejidad de este sistema, se ha observado la necesidad de explorar a fondo nuevas áreas del conocimiento que puedan solventarlo. El principal objetivo del presente trabajo es proporcionar una base la implementación de dicho robot; investigar y explorar herramientas de diseño, lenguajes y en general tecnologías computacionales que puedan ser utilizadas en su desarrollo. Permitiendo de esta manera obtener un sistema modular y flexible con el cual se pueda experimentar y efectuar cambios o mejoras de manera sencilla.

A fin de mostar algunas características de las tecnologías exploradas se desarrollan una serie de simuladores que implementan los modelos matemáticos que describen al robot, algoritmos de control del desplazamiento y técnicas de comunicación y colaboración entre sus procesos.Con el fin de sentar una base sólida y completa para la continuación de este proyecto, se describe a detalle el desarollo e implementación de los sistemas mecánico y de decisión, así como los resultados obtenidos en cada una de sus fases.

Los temas a desarrollar se exponen a lo largo de 6 capítulos, abarcando desde la base teórica hasta la implementación de los sistemas constittivos del robot.

En el capítulo 1, se describen los sistemas constitutivos de un robot y la propuesta del proyecto a desarrollar. En capítulo 2 se presentan las características de los lenguajes Java y VRML se discuten temas como las arquitecturas cliente/servidor, comunicación entre procesos a través de sockets y las bibliotecas de Liquid Reálity.

En el capítulo 3 se describe el modelado matemático del robot y con base en este se desarrolla su sistema mecánico describiendo desde el análisis de requerimientos y propuestas de solución hasta el diseño e implementación de cada uno de los componentes.

El capítulo 4 aborda el tema de la interfaz electrónica, a través de la cual interactúa el sistema de decisión y con los sistemas mecánico y de percepción del robot. De igual forma que en el capitulo anterior, se realiza un análisis de requerimientos y propuesta de solución. A lo largo de este capítulo se sientan las especificaciones de acceso a la interfaz electrónica para su comunicación con el software de control.

En el capítulo 5 se describe el desarrollo del sistema de simulación y control del robot, mostrando el análisis, diseño e implementación de cada una de sus partes.

Finalmente, en el capítulo 6 se describen los resultados obtenidos a lo largo del proyecto haciendo un breve análisis en cada uno y concluyendo al respecto.

Capítulo 1

Antecedentes

En este capítulo se describe la propuesta de implementación del robot móvil, en la cual se mencionan los recursos de cómputo y lenguajes a utilizar. Así mismo se hace un breve repaso de los sistemas constitutivos de un robot para que el lector ubique de mejor manera las metas de este trabajo.

Sistemas constitutivos de un robot

Un robot es una entidad programable e idealmente adaptable al medio de trabajo que lo rodea, capaz de organizar sus acciones con el fin de realizar una tarea.

Los robots surgen por la necesidad del hombre de contar con herramientas que le asistan en sus actividades. La robótica se ha desarrollado a través de tres generaciones. En la primera de ellas los robots únicamente repetían movimientos dentro de trayectorias inflexibles desempeñando tareas muy específicas. En la siguiente generación se implementa el control de las partes móviles del robot utilizando retroalimentación con sensores internos. Los errores disminuyen y se incrementa la flexibilidad de los sistemas. Finalmente, la tercera generación de robots cuenta ya con sensores externos a través de los cuales es posible percibir el medio ambiente. Por otro lado tienen la posibilidad de generar su propio plan de trabajo, utilizando para ello la información de sus sensores externos. Como puede observarse, existe la tendencia de dotar a los robots de capacidades sensoriales y motrices que incrementen su adaptabilidad al medio y por ende su flexibilidad y productividad.

Los robots están constituidos básicamente por cuatro sistemas:

- **Sistema mecánico**, formado por el conjunto de eslabones y mecanismos de transmisión de movimiento.
- **Sistema de percepción**, constituido por los sensores que proporcionan al robot información acerca de su estado y del medio que lo rodea.
- **Sistema de decisión**, controla las acciones del sistema mecánico en base a la información generada por los sistemas de comunicación y percepción.
- **Sistema de comunicación**, es el conjunto de elementos que permiten al usuario interactuar con el robot.

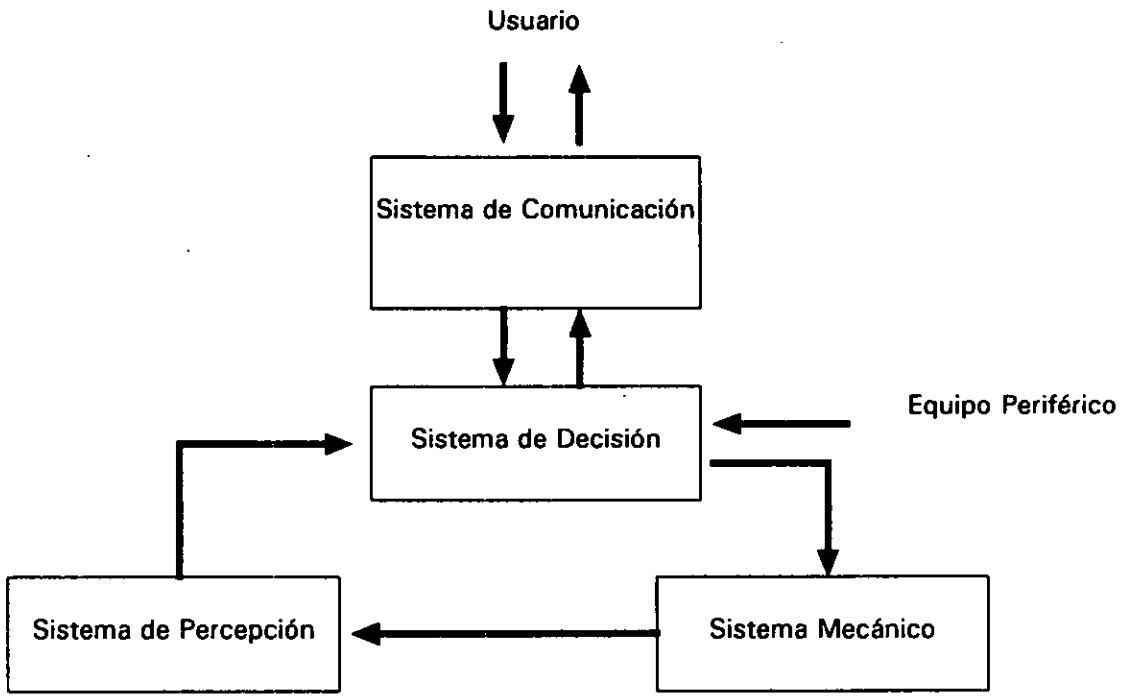


Ilustración 1. Sistemas constitutivos de un robot.

Sistema mecánico

El sistema mecánico es, básicamente, una cadena cinemática constituida por 2 elementos:

- **Eslabones,** conjunto de cuerpos rígidos que pueden acoplarse entre sí por medio de articulaciones. Existen eslabones unitarios, binarios, ternarios, etc., de acuerdo al número de articulaciones con que cuenten.
- **Articulaciones,** son puntos fijos entre dos cuerpos que pueden o no tener movimiento relativo entre ellos. Las articulaciones pueden ser prismáticas o rotacionales, ya sea para trasladar u orientar los eslabones, respectivamente.

Para dotar de movimiento a la cadena cinemática es necesario contar con elemento adicional denominado como actuador. Este elemento genera los esfuerzos necesarios para modificar y mantener la configuración de la estructura mecánica del robot.
Idealmente un actuador debe tener las siguientes características:

- **Inercia pequeña,** que permita una rápida respuesta del robot.
- **Alta rigidez,** para evitar errores por desplazamientos debido a deformaciones causadas por la carga manejada.
- **Acceso al control de sus parámetros,** posición, velocidad, aceleración, etc.

Los actuadores se clasifican de acuerdo al siguiente cuadro:

Actuadores

Neumáticos, utilizan aire a presión para producir movimiento.

Ventajas: Rápidos, económicos, fáciles de instalar, componentes confiables y limpios aún en caso de fuga.

Desventajas: Difíciles de de controlar en posiciones intermedias por el efecto de compresión. Su potencia se encuentra en un rango medio respecto a otros actuadores.

Hidráulicos, Utilizan liquidos para transmitir esfuerzos y transformarlos enmovimiento.

Ventajas: Entre los actuadores, son los que mayor potencia pueden manejar. Es posible controlar su posición y velocidad.

Desventajas: Son costosos, las servovávulas son sensibles al polvo, requieren una planta para generar la potencia hidráulica, los líquidos utilizados pueden ser inflamables.

Eléctricos: Transforman la energía eléctrica en movimiento. Actualmente existen una grán variedad de este tipo de actuadores, entre los más utilizados están los siguientes:

- **Servomotores**, proveen de movimiento continuo de alta presición, grán fuerza y potencia, su costo puede llegar a ser elevado.
- **Motores de pasos**, proveen de movimientos discretos de baja precisión, poca fuerza y potencia, su costo reducido.

Ventajas: Control sencillo, grán versatidad.

Desventajas: Limitación de la potencia.

Sistema de percepción

Los sensores que forman este sistema se clasifican en 2 tipos:

- **Internos**: Proporcionan información sobre la posición, velocidad y aceleración de las partes móviles del robot, y en general acerca del estado interno del robot.
- **Externos:** Provee de información acerca del medio con el cual interactúa el robot.

Dentro de estas clasificaciones existen varios tipos de sensores dependiendo de la variable física a supervisar.

Sensores internos

Sensores de contacto: Estos dispositivos cortan o cierran un circuito eléctrico (0 o 1 lógico) al verificarse una condición, como lo es el contacto físico con un objeto. Existen 2 tipos básicos, los switch o push button y los interruptores magnéticos.

Sensores de posición: Indican la posición de las articulaciones y puede encontrárseles en dos variantes:

- **Analógicos:** Entregan una señal analógica cuya magnitud en voltaje es proporcional a la posición de su cursor. Estos dispositivos pueden ser resistivos o por efecto de acoplamiento magnético.
- **Digitales:** Proporcionan una señal discreta que representa la posición del cursor. Los más utilizados son los codificadores optoelectrónicos, ya sean incrementales o absolutos. Su funcionamiento se basa en la detección de marcas en una regleta o disco.

Sensores de velocidad: Indican la velocidad con que se desplazan las articulaciones. Los más utilizados en la robótica son los siguientes:

- **Optoelectrónicos:** La velocidad es calculada a partir de la frecuencia con que son detectadas marcas en una regleta o disco.
- **Dínamo:** Básicamente es un generador de corriente directa, el voltaje es proporcional a la velocidad con que se mueve su estator.

Sensores externos

Sensores de tacto:

- **Contacto binario:** Utilizan interruptores que al contacto producirán información binaria acerca de la ausencia o presencia de objetos.
- **Piel artificial:** Básicamente es un dispositivo resistivo compuesto por una matriz de electrodos y una placa conductora, ambas separadas por material plástico contaminado de partículas conductoras. La resistencia varía de a cuerdo a la presión ejercida.
- **Deslizamiento:** Detectan el movimiento relativo de un objeto al entrar en contacto con el.

Sensores de esfuerzo:

- **Actuadores del robot:** Los actuadores del robot pueden ser utilizados como sensores de esfuerzo midiendo la energía que es necesaria aplicar a estos para mantener una determinada carga.
- **Galgas tensométricas:** Detectan la fuerza aplicada a estos dispositivos por deformación de un elemento elástico.

Sensores de proximidad:

Ópticos: Utiliza la reflexión de un haz luminoso para determinar la presencia de un objeto.

Magnéticos: Utiliza el acoplamiento magnético para la detección de objetos metálicos.

Neumáticos: Los objetos son detectados por la obstrucción de un flujo de aire.

Acústicos: Efectúan la detección de objetos por medio de la reflexión de señales acústicas.

Sensores de visión: Son los dispositivos más completos, ya que permiten al robot efectuar un reconocimiento global de su entorno. Proporcionan imágenes del área de trabajo que luego serán procesadas con el fin de obtener información como: objetos que la componen, geometría y localización de estos.

Sistema de decisión

El sistema de decisión está formado básicamente por un controlador que administra los recursos del robot para efectuar una tarea. El controlador está formado por las siguientes partes:

- **Memoria:** Lugar donde se guarda la secuencia de tareas a desempeñar.
- **Secuenciador:** Interpreta la información almacenada en la memoria e indica a los demás componentes del controlador las acciones a realizar.
- **Unidad computacional:** Efectúa cálculos y operaciones especificadas por el secuenciador.
- **Interfaz con los sensores internos del robot:** Convierte las señales generadas por los sensores internos a datos numéricos y en general información que pueda interpretar el controlador.
- **Interfaz con los sensores externos del robot:** Convierte las señales generadas por los sensores externos a datos numéricos y en general información que pueda interpretar el controlador.
- **Interface entre el secuenciador y los amplificadores de potencia:** Provee las señales necesarias para activar los actuadores a través de una fase de potencia. Estas señales se basan en la información provista por el secuenciador.
- **Interface con el equipo auxiliar:** Adecua las señales de entrada y salida para entablar comunicación con el equipo periférico.

Sistema de comunicación

Constituye una interfaz a través de la cual el operador interactúa con el robot. Esta comunicación permite al usuario transferir información relativa a la planeación y ejecución de tareas al robot. La comunicación puede llevarse a cabo utilizando de diferentes medios: señales eléctricas, información visual o sonora, etc. Por otro lado la información puede ser transmitida utilizando lenguajes de alto nivel (lenguaje natural) o bien de bajo nivel (señales binarias).

Propuesta de implementación de un robot móvil

Los robots móviles cuentan con la capacidad de desplazarse a través del medio que los rodea, trasladando de esta forma su área de trabajo a diferentes locaciones. La principal ventaja que presentan este tipo de robots es que al no estar sujetos a un área fija, cuentan, potencialmente, con un espacio infinito donde realizar tareas. Idealmente un robot móvil debe portar con los recursos suficientes para ser independiente de medios externos, sin embargo, esta independencia compromete en gran medida los requerimientos de sus sistemas. La propuesta para este proyecto se enfoca en los sistemas de decisión y comunicación del robot apoyándose en tecnologías como realidad virtual y arquitecturas cliente/servidor. En primera instancia, las aplicaciones de realidad virtual han demostrado ser un medio eficiente para la visualización de fenómenos complejos. En nuestra propuesta se utiliza VRML (Virtual Reality Modeling Languaje) como un medio para extender la funcionalidad del sistema de comunicación de forma que no solo se disponga de un medio de control como un jostick o teclado, con el cual indicar las acciones a tomar, si no de todo un sistema interactivo, en el cual se pueda controlar y visualizar a detalle el trabajo del robot. Por otro lado se minimizará la electrónica a bordo del robot dividiendo las funciones del sistema de decisión en una serie de procesos ejecutados en diferentes computadoras. Esta separación de funciones tiene por objeto lograr un diseño modular y flexible capaz de accesar recursos de cómputo externos manteniendo a bordo del robot únicamente la electrónica necesaria para el control directo de los actuadores y la recolección de señales de los sensores (Ilustración 2).

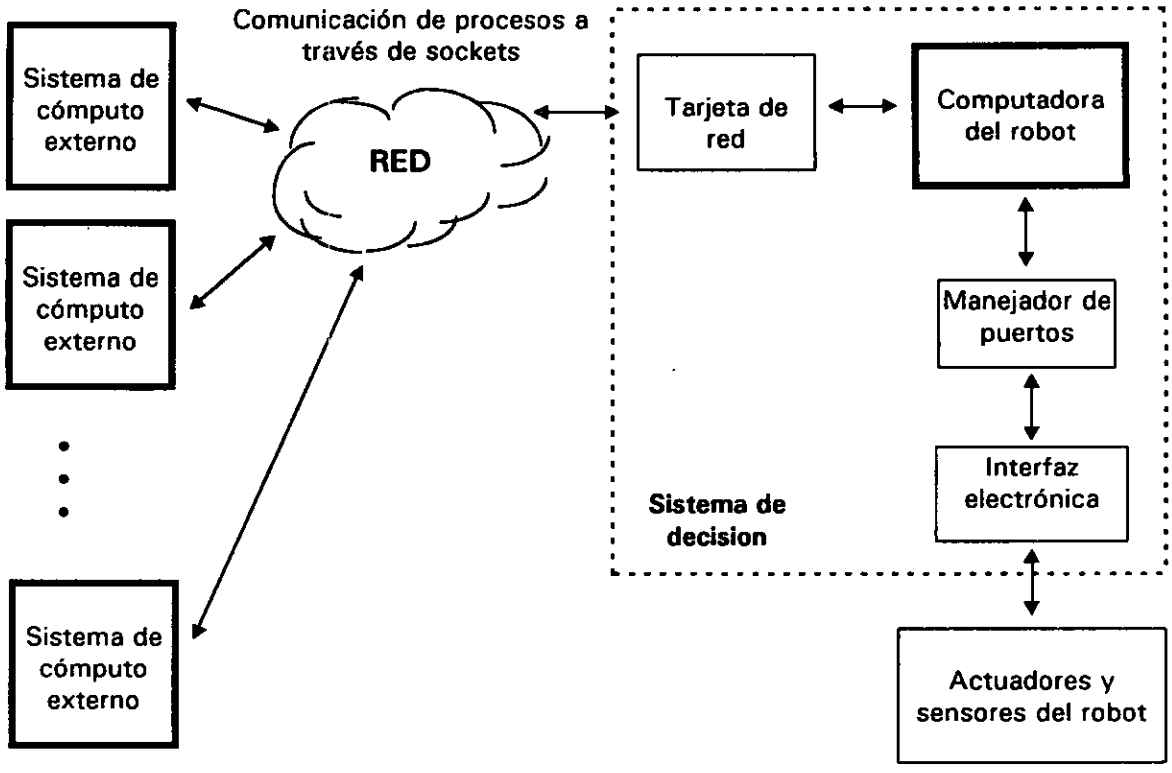


Ilustración 2. División del trabajo del sistema de decisión del robot en procesos externos menores.

La computadora del robot es un sistema basado en un procesador comercial 80X86, ya existen un gran numero de lenguajes de desarrollo y hardware de expansión para este tipo de configuraciones, además de hacer posibles pruebas con equipo PC sin tener que emular un determinado controlador. En esta computadora se ejecuta una aplicación para el control de los actuadores e interpretación de las señales provenientes de los sensores, tareas que no utilizan grandes recursos de cómputo pero que requieren una atención inmediata. Los sistemas externos estarán encargados de proyectar trayectorias a través del terreno y generar el plan de trabajo para el robot entre otros procesos que consumen grandes recursos de cómputo pero cuya frecuencia de requerimiento es menor. Debe considerarse que un robot es un sistema que requiere procesamiento en tiempo real, por lo que el diseño de un esquema distribuido amerita un estudio más a fondo para su correcta implantación. Este trabajo se enfoca a la investigación de los medios para hacerlo posible como son las arquitecturas cliente/servidor y comunicación entre procesos a través de sockets.

El diseño del robot cuenta con 3 pares de extremidades que utiliza como medio de apoyo y desplazamiento. Cada una de estas extremidades, formada por 4 eslabones unidos por 3 articulaciones rotacionales, es en sí misma un robot independiente de tres grados de libertad,. En la siguiente figura puede apreciarse el bosquejo una de estas extremidades y sus características principales.

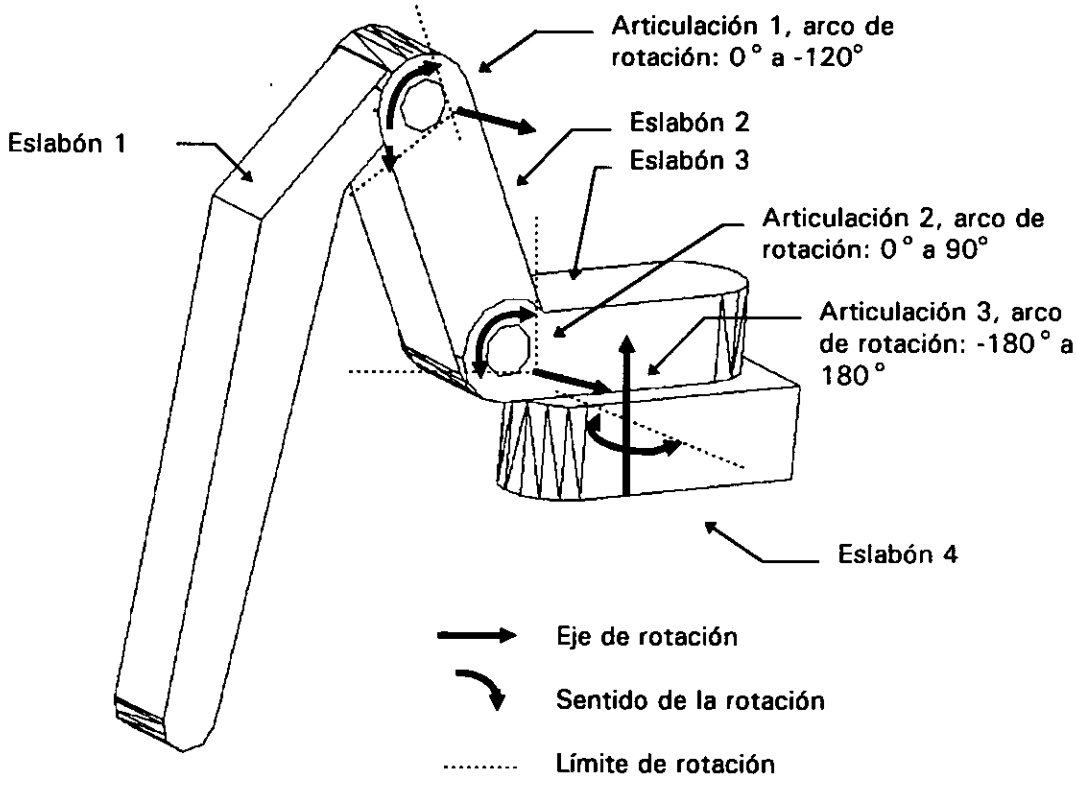


Ilustración 3. Esquema de una extremidad (Diseño ACAD).

Cada articulación cuenta con un control de posición. Este control utiliza un codificador optoelectrónico montado sobre el eje de cada actuador. Las señales son procesadas por la computadora del robot a través del programa de control, que incrementa o decrementa las variables de conteo. Utilizando este esquema de control es posible llevar la extremidad de un punto (x,y,z) a otro utilizando cinemática inversa y seguir una cierta trayectoria generando los puntos intermedios.

Para llevar a cabo la tarea de desplazamiento del robot, las extremidades se agrupan en conjuntos de 3. El "grupo 1" está formado por las extremidades 1,3 y 5, y el "grupo 2" por las extremidades 2,4 y 6.

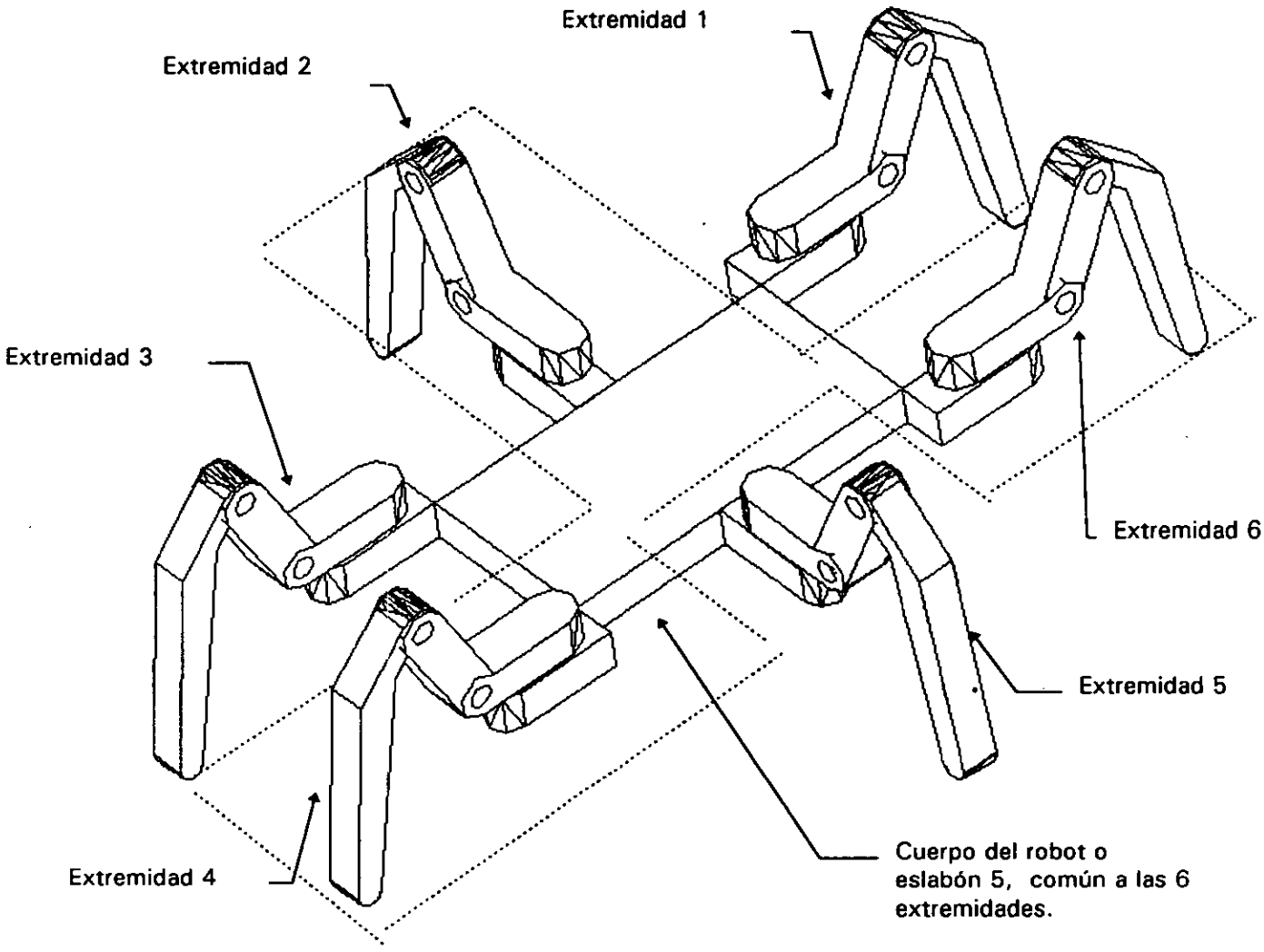


Ilustración 4. Distribución de las extremidades del robot. La zona punteada encierra las extremidades pertenecientes al grupo2, fuera de esta zona pertenece al grupo1 (Diseño ACAD).

Estos grupos adoptan en forma alternada, las tareas de apoyo y desplazamiento o bien de avance. Mientras que las extremidades que forman a uno de los grupos se mantiene en contacto con el terreno y realizan movimientos para desplazar el robot, las extremidades del otro grupo se levantan y avanzan hacia donde se desea dar el siguiente paso. Una vez completada la tarea de cada uno, se intercambian las funciones.

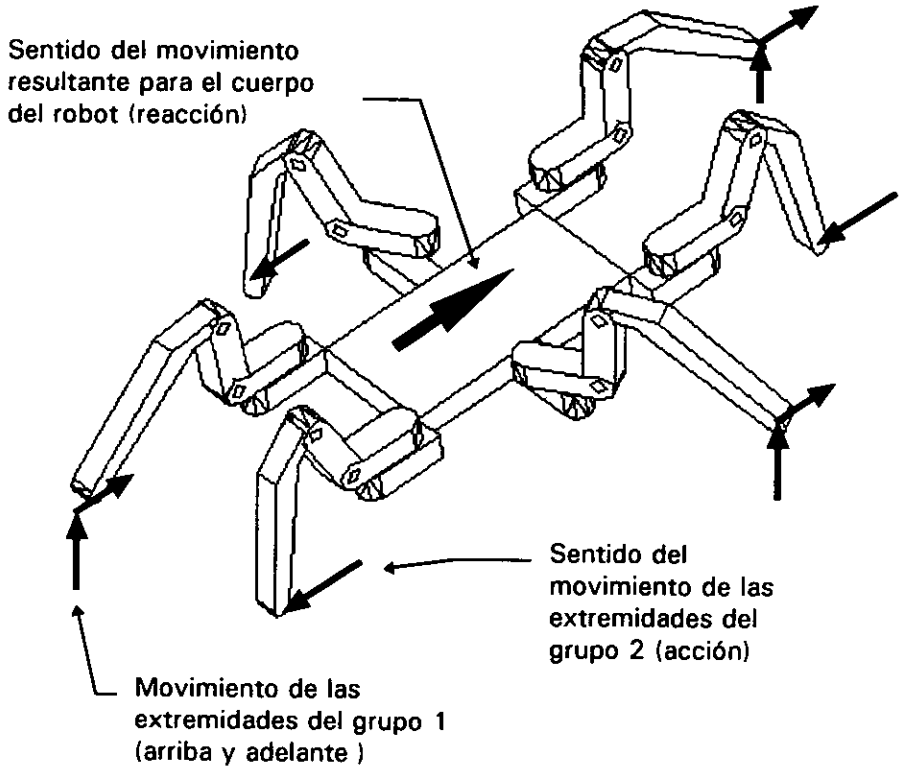


Ilustración 5. Vista superior del robot, grupo2 en contacto con el terreno, apoyando y desplazando; grupo 1 avanzando.(Diseño ACAD).

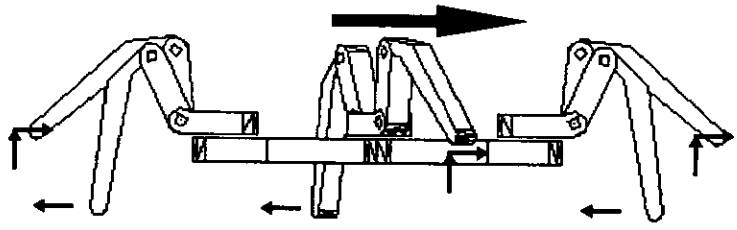


Ilustración 6. Vista lateral del robot, grupo2 en contacto con el terreno, apoyando y desplazando; grupo 1 avanzando.(Diseño ACAD).

Se proyecta que el robot porte un pequeño sistema de video con el cual puedan hacerse tomas del medio lo rodea, identificar objetos e indicar al robot su seguimiento.
El movimiento de la cámara será provisto por un dispositivo de orientación en coordenadas esféricas. Las especificaciones de este subsistema no han sido terminadas aún, sin embargo, su dispositivo de orientación es considerado en el diseño de la interfaz electrónica y del software de simulación y control para su futura implantación. En los capítulos subsecuentes se hará referencia a él considerándolo únicamente como un dispositivo de 2 grados de libertad.

La presente propuesta se apoyará con la programación de una serie de simuladores utilizando los lenguajes Java y VRML, en los cuales se mostrará la visualización y control del robot, el

uso de procesamiento en forma distribuida y conectividad con equipos de cómputo remoto. Adicionalmente se describen la implementación de los sistemas mecánico, de percepción y decisión del robot con el fin de realizar las primeras pruebas de control y desempeño, dejando así una buena base para la continuación de este proyecto.

Capítulo 2

Java, VRML y Arquitecturas Cliente/Servidor

En el presente capítulo se describen en forma introductoria los lenguajes Java y VRML, así como las arquitecturas cliente/servidor, destacando en todos ellos sus características principales y utilidad para el proyecto.

Java

Inicialmente Java comenzó como un lenguaje de programación que facilitaría el desarrollo de aplicaciones e interfaces para dispositivos electrodomésticos tales como televisores, hornos de microondas, etc. Sin embargo, con el crecimiento que tuvo el uso de Internet gracias al surgimiento del WWW y los visualizadores gráficos, se hizo un replanteamiento del uso de este lenguaje para que fuera utilizado como el lenguaje de programación ideal para Internet. Desde su concepción Java fue diseñado para trabajar en red, por lo que fueron diseñadas un conjunto de rutinas especializadas para conformar bibliotecas que permiten la rápida creación de aplicaciones cliente/servidor, utilizando como protocolo de transporte TCP. Java tuvo una rápida aceptación entre la comunidad de Internet; ya que parecía ser la respuesta a muchos de los problemas que se venían arrastrando tales como portabilidad y falta de estándares. Además de que dicho lenguaje fue distribuido de forma gratuita a los usuarios ordinarios. Tal fue su aceptación que corporaciones tan grandes e importantes como Microsoft, Netscape, Silicon Graphics, IBM, HP, entre otras, empezaron a adquirir licencias para usar esta tecnología en sus propias plataformas y aplicaciones.

Netscape fue la primera empresa que incluyo Java en su visualizador, permitiendo la ejecución de pequeñas aplicaciones llamadas applets. Estos applets son programas creados con Java que proporcionan una mayor interacción con la persona que visualiza una página en el visualizador. Siendo ésta la principal causa que impulso a la comunidad al uso de Java en Internet.

Este crecimiento acelerado provocó que el término Java ya no se asocie únicamente a un lenguaje de programación, sino a todo un conjunto de sistemas de hardware, y software (sistema operativo y lenguaje) que están pensados para explotar al máximo el concepto de intranets.

En el caso del sistema operativo tenemos que la compañía SUN ha diseñado un nuevo sistema operativo pequeño y eficiente que puede o no incluir una máquina virtual para la ejecución de programas hechos en Java (esto depende de si el sistema operativo es ejecutado en una máquina Java o en alguna arquitectura diferente). y un visualizador de WWW desde donde se pueden accesar todas las aplicaciones que necesita el usuario final (procesador de palabras, hoja de cálculo, creador de presentaciones, administradores de bases de datos, etc.).

En el lado de hardware se ha desarrollado hardware específico para que las aplicaciones hechas en Java se ejecuten eficientemente. Actualmente existen tres versiones de lo que se conoce como la máquina Java: microjava, picojava y ultrajava. Cada una de estas implementaciones incluye una interfaz de red, una memoria ROM de 4 megas así como 4 megas de RAM. En la memoria ROM se encuentra instalado el sistema operativo JavaOS, esto hace que el sistema operativo sea cargado rápidamente en memoria y la máquina inmediatamente entre en funcionamiento.

El lenguaje Java por sí mismo, ha madurado cada vez más incrementando considerablemente el conjunto de bibliotecas con el que se dispone actualmente. Se han desarrollado bibliotecas de clases para controlar dispositivos electrónicos, para accesar a bases de datos, para telecomunicaciones, para VRML, para multimedios, etc. También ha crecido el numero de plataformas en las cuales pueden ser ejecutados los programas en Java.

Este lenguaje se presenta como una buena opción para la implementación de otras tecnologías como la de agentes, y por lo tanto de sistemas de administración de datos, y de búsqueda y filtrado de información, debido a que proporciona grandes facilidades para desarrollar sistemas distribuidos, además, el lenguaje es sumamente portable. Por portable entendemos que el sistema que desarrollemos en Java podrá trabajar sin ningún cambio en equipos UNIX (Solaris, NextStep, HP-UX, Irix), PC´s (Windows 95, Windows NT), equipos Macintosh y Mainframes. Esto contrasta notablemente con otros lenguajes de programación como C, C++, Pascal y VisualBasic; ya que con estos lenguajes resulta muy difícil portar un programa de una plataforma a otra, principalmente por incompatibilidad con las bibliotecas, con el administrador de ventanas (interfaz gráfica) o debido a que el compilador del lenguaje como tal no está disponible en todas las plataformas.
Esta portabilidad es de gran utilidad ya que permite trabajar con todas las plataformas como si fueran una sola (sobre todo desde el punto de vista del administrador), ahorrándose con esto una gran cantidad de tiempo en el desarrollo de sistemas.

Las plataformas en las que actualmente funciona Java son:

- SPARC Solaris (2.3 o posterior)
- Intel x86 Solaris
- Windows 3.x
- Windows NT/95 (Intel x86)
- Macintosh 7.5
- AIX
- HP-UX
- Digital UNIX
- NCR SysV
- Sony NEWS
- Linux

Los visualizadores que actualmente soportan Java son:

- HotJava
- Netscape Navigator
- Internet Explorer

Actualmente existe un grupo de desarrollo de GNU llamado JOLT que está trabajando para portar la máquina virtual de Java y su compilador a otras plataformas como:

- Amiga
- NeXT
- OS/2

Características de Java

A continuación se enumeran algunas características del lenguaje que lo hacen atractivo para su uso en nuestro sistema.

- **Simple**. Debido a que en Java no es necesario manejar apuntadores de forma explícita y a que incluye un recolector automático de basura Java resulta ser un lenguaje sencillo de aprender y adecuado para desarrollar-sistemas complejos evitando al desarrollador muchos errores comunes en otros lenguajes, como por ejemplo: que la memoria reservada no sea liberada posteriormente, o sea liberada antes de tiempo, o se trate de liberar más de una vez. Este trabajo lo desempeña de forma óptima el recolector de basura de Java debido a que este es ejecutado como un thread (hilo de control) con una prioridad muy baja, siendo imperceptible para el usuario o el programador. Por otra parte, debido a que su sintaxis es muy parecida a la de C o C++, el código escrito en este lenguaje se puede entender fácilmente. Aunque hace uso de la sintaxis de C++ Java elimina muchos de las opciones que están permitidas en C++, como la sobrecarga de operadores, y conversiones automáticas entre tipos.

- **Orientado a objetos**. Java como todo lenguaje orientado a objetos proporciona facilidades como el manejo de la herencia, el polimorfismo y el encapsulamiento. Esto ayuda en gran medida al programador, si anteriormente ha realizado un análisis y un diseño orientado a objetos; ya que el lenguaje le permitirá realizar la implementación de una forma más sencilla y directa. Otra parte importante es que en Java siempre tenemos que utilizar la metodología orientada a objetos (no hay forma de programar en forma estructurada), esto contrasta notablemente con otros lenguajes como C++ que si lo permite. Como Java utiliza únicamente herencia simple, le ahorra al programador todos los problemas que se pueden provocar por el uso de herencia múltiple en sus programas, por ejemplo la resolución de métodos con el mismo nombre pero que provienen de diferentes ramas de la jerarquía de clases. Al mismo tiempo de que proporciona otras formas sencillas de implementar sistemas que anteriormente requerían del uso de la herencia múltiple (interfaces).

- **Distribuido**. Java está diseñado para desarrollar aplicaciones que trabajen en redes. Java soporta diferentes niveles de conectividad de redes a través de un paquete conocido como **Java.net**. Esto implica poder trabajar con protocolos de comunicación tanto de las primeras capas del modelo OSI (TCP/IP), como con protocolos de las capas más altas (HTTP y FTP). Además, los proveedores tienen planeado incluir dentro de las bibliotecas estándar de Java, un conjunto de clases para el manejo de mensajes remotos (algo parecido a las RPC pero con métodos), y la creación de objetos que puedan viajar a través de la red y ejecutarse.

- **Interpretado**. El compilador de Java genera bytecodes, en vez de código de máquina nativo. Para ejecutar una aplicación es necesario suministrar este bytecode a la máquina virtual de Java para que los pueda interpretar. Esto implica una cierta degradación en el tiempo de respuesta de la aplicación. No obstante, actualmente existen implementaciones de una tecnología conocida como just-in-time. Esta tecnología permite al momento de ejecución generar código de la máquina nativa a partir de los bytecodes, esto repercute en un incremento notable en la velocidad de ejecución de los programas. Además, con el surgimiento de arquitecturas de hardware que ejecutan directamente el bytecode se pueden obtener velocidades de ejecución óptimas para las aplicaciones. Además de todo esto el hecho de ser interpretado nos proporciona todas las ventajas para depurar y ejecutar los programas de forma incremental, sin tener que estar sujetos al antiguo ciclo de: escribir-compilar-ligar-ejecutar-depurar-escribir.

- **Robusto**. Java es un lenguaje fuertemente tipificado, lo cual implica que al momento de compilación el compilador de Java es capaz de advertir al usuario de posibles

errores semánticos, los cuales por lo regular son difíciles de depurar. Además el lenguaje tiene interconstruida la capacidad de manejar excepciones, esto permite al programador contar con una forma estándar de manejo de posibles errores que se pueden presentar al momento de ejecución, logrando con esto sistemas más estables al momento de ser usados por los usuarios finales. Además, el esquema con el que cuenta Java para el uso de memoria nos garantiza que no existirá forma de dañar o liberar memoria de forma accidental.

- **Seguro**. Debido a que Java está pensado para ser utilizado en redes es muy importante que el propio lenguaje cuente con varias formas de garantizar la integridad, la consistencia y la privacidad de los programas. De hecho Java cuenta con varios mecanismos y niveles para que se puedan desarrollar sistemas y que éstos puedan viajar por la red de forma confiable, como puede ser la inclusión de algoritmos de cifrado mediante el uso de llave pública. Además, la máquina virtual proporciona todo un mecanismo para verificar y autentificar los bytecodes que está ejecutando en un momento dado, evitando con esto ejecutar programas que puedan dañar los sistemas del usuario o que proporcionen información confidencial a otras personas.

- **Arquitectura neutral**. Debido a que el código fuente de Java es traducido por el compilador en bytecodes, y este código no es exclusivo de ningún hardware, se puede tener la certeza de que un programa funcionará perfectamente en cualquier plataforma que tenga incluida la máquina virtual de Java. Actualmente la gran mayoría de plataformas cuentan con una implementación de la máquina virtual. El surgimiento de las máquinas Java en hardware no implica ningún retroceso en este sentido; ya que aunque el código será ejecutado de manera más eficiente en estas máquinas, todavía podrá el mismo código ser ejecutado en las demás plataformas, gracias a la máquina virtual. Además, el mismo lenguaje estandariza el tamaño de los tipos de datos primitivos, Por ejemplo, un entero siempre ocupara 32 bits. Las bibliotecas que forman parte del sistema definen interfaces portables. Por ejemplo, hay una clase abstracta ***Window*** e implementaciones de ésta para Unix, Windows y Macintosh.

- **Multithreaded**. Java tiene la capacidad de crear múltiples threads, lo cual permite la ejecución en forma concurrente de objetos. Para que esta ejecución se realice adecuadamente Java cuenta con un sofisticado conjunto de primitivas de sincronización que están basadas en el ampliamente usado paradigma de monitor y variable de condición introducido por C.A.R.Hoare. Mucho del estilo de ésta integración viene del sistema Cedar/Mesa de Xerox. Otras ventajas son las de poder obtener una mejor interacción con el sistema y que éste tenga una respuesta casi en tiempo real.

La compañía SUN liberó en marzo del presente año la versión 1.1 de su JDK (Java Development Kit) que incluye las siguientes características:

- **Archivos JAR**. JAR es un formato de archivo independiente de plataforma que contiene varios archivos dentro de uno solo. Múltiples applets y los componentes que requieren (archivos .class, imágenes y sonidos) pueden ser incluidos en un archivo JAR y subsecuentemente ser cargados en el visualizador en una sola transacción HTTP, mejorando significativamente el tiempo de transferencia. El formato JAR también acepta compresión, lo cual reduce el tamaño del archivo y disminuye el tiempo de transferencia. Además, el autor de la aplicación puede firmar digitalmente el archivo para autentificar su origen.

- **Internacionalización**. Permite el desarrollo de applets localizables. Es decir, incluye mejoras para desplegar caracteres UNICODE mediante un mecanismo local, es sensible a la fecha, hora, tiempo y horario local, e incluye un convertidor del conjunto de caracteres, etc.

- **Seguridad**. El JDK incluye APIs para firmas digitales, manejo de llaves, listas de control de acceso, e incluye la posibilidad de firmar clases y otros datos.

- **Mejoras al AWT**. Está conformado por una vasta infraestructura para el desarrollo de GUIs a gran escala, además, contiene APIs para imprimir, para hacer uso del clipboard, manejando diferentes tipos de cursores por componente, un modelo de eventos basado en delegación, mejoras a los gráficos y las imágenes, y un soporte más flexible para el uso de fonts.

- **Mejoras a la interfaz de red**. Lo cual incluye el soporte para seleccionar sockets estilo BSD. Las clases ***Socket*** y ***ServerSocket*** pueden ser extendidas a través de herencia. Se agregaron nuevas subclases a la clase ***SocketException*** para tener mayor granularidad en el reporte y manejo de errores.

- **Invocación de métodos remotos (RMI)**. Lo cual permite a los objetos de Java invocar métodos de objetos que se encuentran en otras máquinas virtuales, inclusive en otras computadoras. Referencias a tales objetos remotos pueden ser pasados como parámetros en llamadas RMI.

- **Serialización de Objetos**. Extiende las clases base de entrada/salida de Java para dar soporte a objetos. Permitiendo la codificación de objetos y la reconstrucción de estos provenientes de un flujo de bytes. La serialización es utilizada para la persistencia de los objetos o para comunicación vía sockets o RMI. La codificación de los objetos protege los datos privados y temporales.

- **Clases *Byte*, *Short* y *Void***. Extiende la biblioteca de clases para incluir clases que permiten manipular a los tipos primitivos como objetos.

- **JDBCTM - Java Database Connectivity**. JDBC es una interfaz de acceso estándar para bases de datos SQL. Éste también proporciona una base común sobre la cual pueden ser construidas herramientas e interfaces de alto nivel. Incluye un bridge ODBC. El bridge o puente es una biblioteca que implementa el JDBC (Conectividad para Bases de Datos de Java) en términos del API estándar para ODBC.

- **Nueva interfaz para métodos nativos Java**. Una interfaz de programación estándar para escribir métodos nativos. El principal objetivo es obtener compatibilidad a nivel binario de bibliotecas de métodos nativos a través de todas las implementaciones de la máquina virtual en una plataforma específica.

- **Documentación para la interfaz del compilador JIT**. Esta documentación es para desarrolladores de herramientas quienes están escribiendo generadores de código nativo u otras utilerías que corren dentro de la máquina virtual de Java.

- **Clases anidadas**. Provee una sintaxis simple para la creación de clases ***Adapter***. Una clase ***adapter*** es una clase que implementa una interfaz (o clase) requerida por un API, y delega el flujo de control de regreso a un objeto "principal".

- **Mejoras en el rendimiento**. Mejoras en el ciclo interno del intérprete ahora escrito en ensamblador sobre Win32 y Solaris/Sparc. Las Ccases ***peer*** del AWT han sido reescritas para Win32. Y un uso opcional de threads nativos sobre Solaris.

VRML 2.0 (Virtual Reality Modeling Languaje)

VRML es un lenguaje para la descripción de ambientes virtuales, permite al usuario interactuar con la representación tridimensional de objetos y entidades. En forma general puede considerársele como un lenguaje de programación para aplicaciones multimedia y como un excelente recurso de visualización comercial y/o científica.

El concepto de VRML fue concebido en la primavera de 1994, en la primera Conferencia Anual World Wide Web realizada en Ginebra, Suiza. Tim Berners-Lee y Dave Ragget organizaron una sesión para discutir las interfaces de Realidad Virtual en World Wide Web. Los participantes de esta sesión describieron algunos proyectos que ya se realizaban para construir herramientas de visualización gráfica tridimensional que interoperaran con el Web. Los asistentes coincidieron con la necesidad de crear un lenguaje común, que representase una especificación para la descripción de escenas tridimensionales e hiperenlaces dentro del WWW; una analogía del HTML utilizando los recursos de la Realidad Virtual. Se sugirió el término VRML (Virtual Reality Modeling Languaje) y el grupo resolvió iniciar el trabajo de especificación al terminar la conferencia.

Poco tiempo después de estas sesiones, se creó la lista de correo www-vrml, en la cual se discutiría el desarrollo de una especificación para la primera versión del VRML. La respuesta a la invitación fue impresionante; al transcurrir una semana, ya había más de mil miembros. Después de un período inicial de consolidación, el moderador de la lista, Mark Pesce del grupo Labyrinth, anunció su intención de tener preparada una versión preliminar de la especificación para la conferencia WWW que habría de realizarse en el otoño de 1994, sólo cinco meses después. Hubo un acuerdo general dentro de la lista,
pues a pesar del poco tiempo disponible, el proyecto podía lograrse si se tomaba en cuenta que los requerimientos para la primera versión no eran tan ambiciosos y que el VRML podía adaptarse con base en una solución existente. La lista creó con rapidez un grupo de requerimientos para la primera versión y comenzó la búsqueda de tecnologías que pudieran adaptarse para satisfacer las necesidades del VRML.

La búsqueda de tecnologías existentes dio como resultado varios candidatos viables. Después de deliberar mucho, la lista llegó a un consenso : el formato del archivo ASCII para Open Inventor de la compañía Silicon Graphics, Inc. El formato del archivo Inventor da soporte a descripciones completas de escenas 3D con objetos, iluminación, materiales, propiedades de ambiente y efectos de realismo transmitidos de manera poligonal. Un subgrupo del formato Inventor, con extensiones para dar soporte al trabajo en red, conforma la base del VRML. Gavin Bell, de la compañía Silicon Graphics, ha adaptado dicho formato para el VRML. SGI ha divulgado que el formato de archivo está disponible para usarse en el mercado abierto y ha hecho del dominio público un analizador sintáctico para dicho formato; de esta manera trata de promover, sin la ayuda de ninguna otra compañía, el desarrollo del visualizador de VRML.

Siguiendo los mismos pasos que en la primera versión VRML2.0 surge del consenso de intereses de usuarios de Internet, entre los que se cuenta una gran cantidad de compañías que darán soporte y comercialización a esta tecnología. La especificación de VRML2.0 cuenta con el registro ISO/IEC DIS 14772-1.

En forma general, un archivo VRML (*.wrl) es tratado en forma similar a uno HTML, el visualizador se encarga de cargarlo en memoria, interpretarlo y generar una representación interna de los objetos y la relación que guardan si. Finalmente se despliega una representación gráfica cuyos elementos pueden interactuar con el usuario. Las características generales de VRML 2.0 son las siguientes:

- **Independiente de plataforma.** El código script es interpretado por una aplicación que servirá de intermediario con cada plataforma en particular, de tal forma que el código en VRML no tendrá que ser alterado
- **Extendible.** Proporciona estructuras sintácticas que permiten incrementar la funcionalidad y potencial del lenguaje definiendo nuevos prototipos de objetos.
- **Puede ser utilizado en ambientes de red con un bajo ancho de banda.**
El formato ASCII y la estructura sintáctica de los archivos VRML permiten el manejo y transferencia de archivos pequeños a través de la red .

Básicamente, la especificación de VRML1.0 consiste en la descripción de objetos estáticos a través de estructuras de datos denominados como nodos. Estos nodos describen la apariencia, geometría y relación de un objeto con otro. La interacción con el usuario está limitada al uso de CGI a través del visualizador. La especificación de VRML2.0 añade 4 conceptos que complementan a la estructura de nodos de VRML1.0:

- **Sensores.** Proveen de un mecanismo a través del cual el usuario puede interactuar con las entidades de un escenario VRML, generando un determinado evento por cada acción efectuada sobre ellos. En el caso de TimeSensor se genera un evento cada determinado tiempo. La siguiente lista muestra los tipos de sensores:
  - ProximitySensor
  - SphereSensor
  - TouchSensor
  - VisibilitySensor
  - CylinderSensor
  - TimeSensor
  - PlaneSensor

- **Interpoladores.** Generan eventos a través de la interpolación lineal de un conjunto de datos. Se utilizan principalmente para proveer animación a las entidades que componen un escenario. Existe un tipo de interpolador específico por cada tipo de dato:
  - color
  - coordinate
  - position
  - normal
  - orientation
  - scalar

- **Script.** Los scripts describen el comportamiento de las entidades que forman el escenario a través de objetos programados en Java o Java Script.

- **Sonido.** Este nodo provee fuentes de sonido con una distribución en el espacio con las cuales el escenario adquiere mayor realismo.

- **Prototipos.** Son entidades cuya especificación interna es descrita por el usuario, permite enriquecer la especificación básica de VRML a través de la creación de bibliotecas. Su implementación utiliza elementos más simples cuya relación y complejidad es encapsulada como un bloque funcional. Los prototipos son utilizados

para simplificar la lógica de un proceso complejo delegando responsabilidades entre varias entidades.

En la Ilustración 6 puede apreciarse la estructura interna de un prototipo, donde la **cabecera del prototipo** define los campos y eventos de entrada donde recibe información del exterior, así como los eventos de salida, a través de los cuales envía información a otras entidades.

La **cabecera del script de control** contiene las conexiones entre la lógica de control y los nodos VRML.

Las conexiones ***ROUTE*** direccionan los eventos de salida de una entidad hacia otra los eventos de entrada de otra.

La **descripción geométrica de la entidad** es una serie de nodos relacionados entre si, a través de los cuales se especifica la apariencia y geometría que tiene una instancia de nuestro prototipo, y será por medio de esta representación, que nosotros visualizaremos las acciones del script de control.

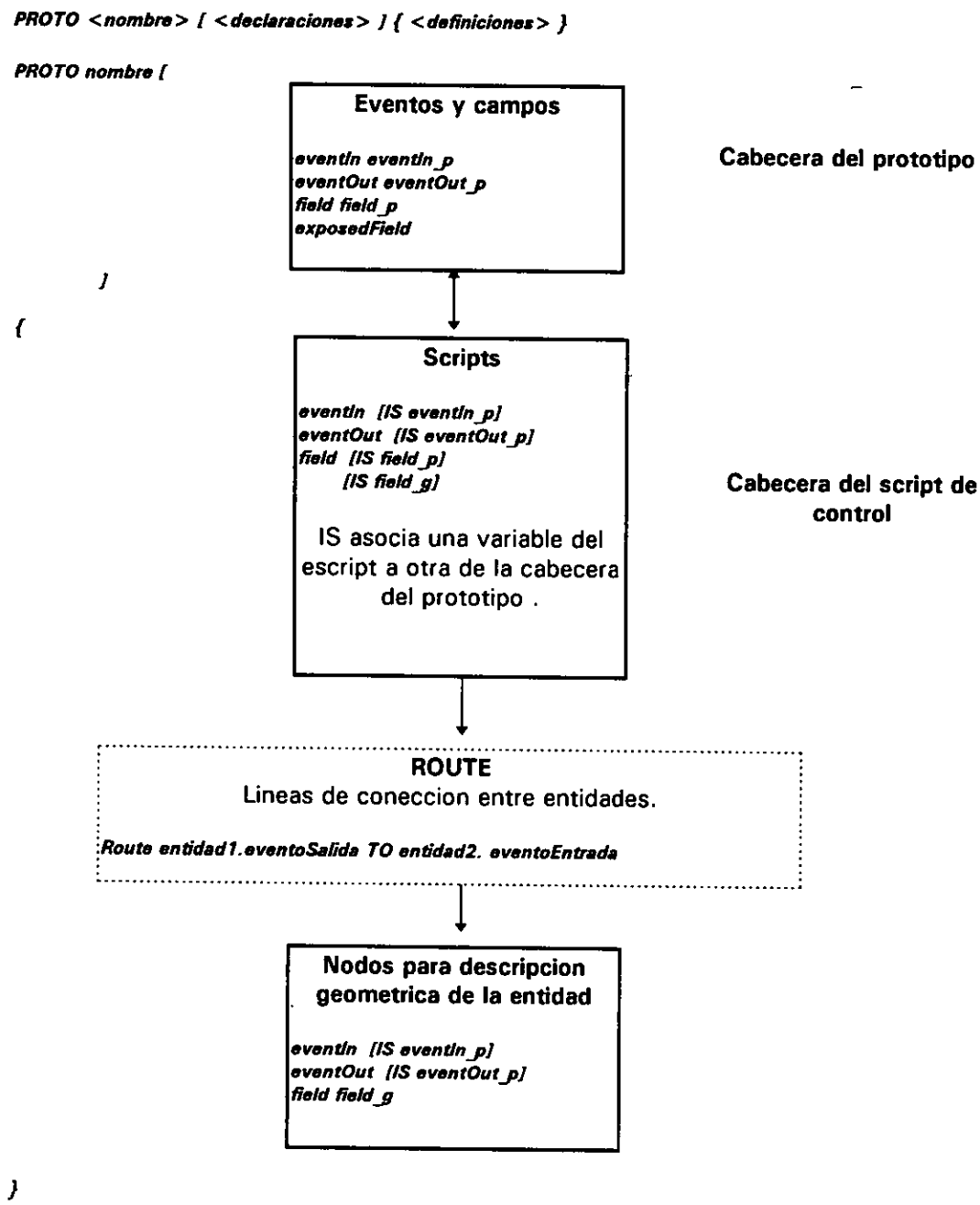


Ilustración 6. Estructura básica de un prototipo.

Las instancias de los prototipos pueden ser vistas como módulos funcionales con sus respectivas conexiones al exterior:

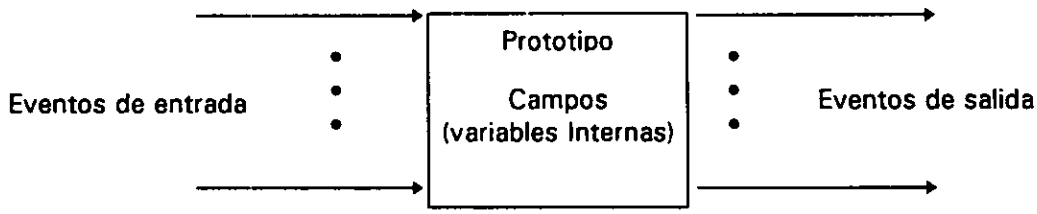


Ilustración 7. Prototipo desde el punto de vista modular.

En forma general, el script interno procesa los datos de entrada y realiza modificaciones en la geometría y apariencia del prototipo. El envío de mensajes a otros módulos completa el ciclo de comunicación. Este esquema de trabajo resuelve de forma sencilla y elegante la complejidad de los sistemas, ya que para su diseño solo se utilizan entidades y conexiones como se muestra en la Ilustración 8.

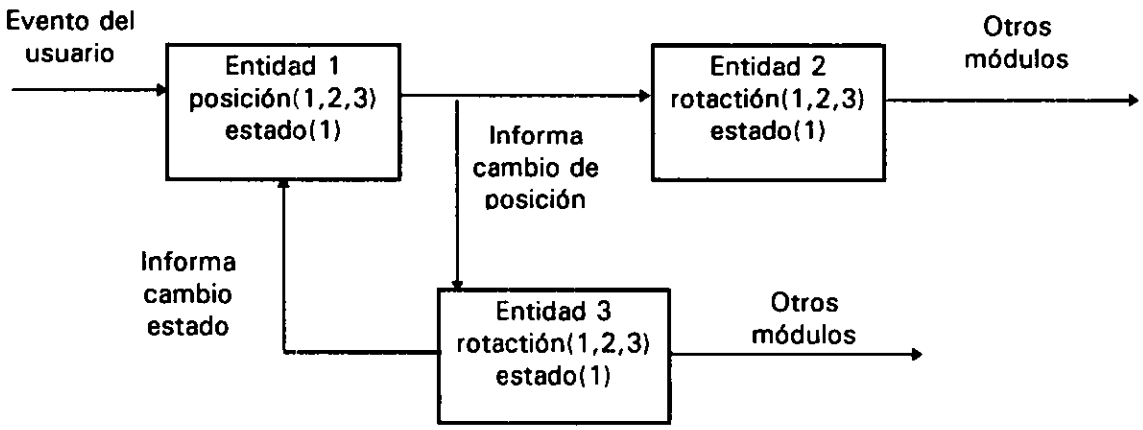


Ilustración 8. Interacción de instancias desde el punto de vista modular.

Para ejemplificar este esquema de programación se muestra parte del prototipo de una de las extremidades del robot.

- En primera instancia se declaran los eventos y campos para el prototipo, los campos son utilizados para inicializar el estado de la entidad, mientras que los eventos de entrada y salida como medio de comunicación al exterior.

```
PROTO Extremidad [

    field SFRotation  rotacionYInicial  0 1 0 1
    field SFVec3f  posicionInicial  0  0 0
```

Los campos son utilizados para la inicialización de posición y orientación de la extremidad.

```
eventIn SFVec3f traslacionRobotIn
eventIn SFRotation  rotXrobotIn
eventIn SFRotation  rotYrobotIn
eventIn SFRotation  rotZrobotIn
```

Los eventos de entrada informan a la extremidad que el robot se está desplazando u orientando, por lo que, si la extremidad pertenece al grupo de apoyo, el script deberá calcular la nueva posicion de sus eslabones..

```
eventIn SFFloat temporizador
```

Reloj de sincronizacion con el robot..

```
eventIn SFVec3f  traslacionExtremoIn
eventIn SFInt32 accionADesarrollar
```

Comandos que recibe la extremidad de su grupo, ya sea en forma codificada en un entero, o bien directa a través de la posición que debe tomar la extremidad.

```
eventOut SFBool  colisionOut
eventOut SFBool  accionFinalizadaOut
eventOut SFVec3f traslacionExtremoOut
```

Eventos generados por la extremidad y que servirán a otras entidades para conocer su estado actual.

```
exposedField SFRotation  roteslabon1f 1 0 0 0
exposedField SFRotation  roteslabon2f 1 0 0 0
exposedField SFRotation  roteslabon3f 1 0 0 0
]
```

Los exposedField tiene las características de campo, evento de salida y evento de entrada, en este caso fueron utilizadas para manipular la geometría del prototipo sin necesidad de recurrir al script.

- A continuación se describe la representación gráfica del prototipo:

```
{
```

Inicio de la definición del prototipo.

```
Transform {
    children [
        Transform {   #cond iniciales
            center  0 0 0
            translation  IS  posicionInicial
            rotation  IS rotacionYInicial
                children [
                    DEF Eslabon_4 Transform {
                        translation  450 0 0
```

Asociacion directa de los campos a los atributos de la geometría.

Definicion de los eslabones, u objetos; descripción de la geometría del prototipo, que será controlado por el escript.

- Finalmente se define a interfaz entre el prototipo y el nodo ***Script*** que provee su funcionalidad:

```
#interfaz entre el cuerpo del robot y una extremidad controlando la cinemática directa e inversa de esta

DEF manejadorExtremidad Script {
  url "manejadorExtremidad.class"
```

Archivo que contiene el código de la clase que controlará el prototipo.

```
  field SFVec3f posicionInicial  IS posicionInicial
  field SFRotation rotacionYInicial IS rotacionYInicial
  eventIn  SFRotation rotacionXRobotIn IS rotXrobotIn
  eventIn  SFRotation rotacionYRobotIn IS rotYrobotIn
  eventIn  SFRotation rotacionZRobotIn IS rotZrobotIn
  eventIn SFVec3f  traslacionExtremoIn IS traslacionExtremoIn
  eventIn SFVec3f traslacionRobotIn IS traslacionRobotIn
  eventOut SFBool  colisionOut IS colisionOut
  eventOut SFBool  accionFinalizadaOut IS accionFinalizadaOut
  eventIn SFInt32 accionADesarrollar  IS accionADesarrollar
  eventIn SFFloat temporizador IS temporizador
  eventOut SFVec3f traslacionExtremoOut IS traslacionExtremoOut
```

Asociacion de las variables de la cabecera del prototipo hacia el script, donde seran procesadas.

```
  eventOut SFRotation rot1Out
  eventOut SFRotation rot2Out
  eventOut SFRotation rot3Out
```

Eventos de salida con los cuales se controla la geometría.

```
},

ROUTE manejadorExtremidad.rot1Out  TO Eslabon_R1.set_rotation
ROUTE manejadorExtremidad.rot2Out  TO Eslabon_R2.set_rotation
ROUTE manejadorExtremidad.rot3Out  TO Eslabon_R3.set_rotation
```

Conexiones explícitas de eventos de salida generados por el script hacia la geometría del robot, cerrando así el circuito de control.

```
} # final del prototipo
```

Liquid Reality

Es un conjunto de bibliotecas desarrollado por la compañía Dimensión X, que permite a las aplicaciones Java soportar la especificación de VRML2.0 y en general hacer uso de representaciones tridimensionales.

La implementación de Liquid Reality incluye las siguientes características:

- Completo soporte a la especificación de VRML2.0.
- API de alto a nivel.
- API de bajo nivel.
- Procesos de rendering específicos para cada plataforma.
- Visualizador de aplicaciones.

Liquid Reality permite el uso de representaciones gráficas conjugando las ventajas del lenguaje Java con el alto desempeño de un lenguaje gráfico de bajo nivel. Su diseño considera una serie de capas que ocultan el manejo de hardware específico a través de métodos nativos programados en C++.

Las bibliotecas que forman la implementación de Liquid Reality son:

- 3D/Multimedia programming interface for Java (***LR-Core***)
- Low-level 3D API for Java (***ICE***)
- 3D Math library and utilities (***Geom***)
- 3D Sound library and utilities (***Sound***)
- Native libraries (***DNXRend***, ***Direct3D***, ***OpenGL***)

Estas bibliotecas conforman la implementación de LR, el desarrollador tiene acceso a cada una de ellas a través de un API. En la siguiente figura se muestra la arquitectura general de Liquid Reality:

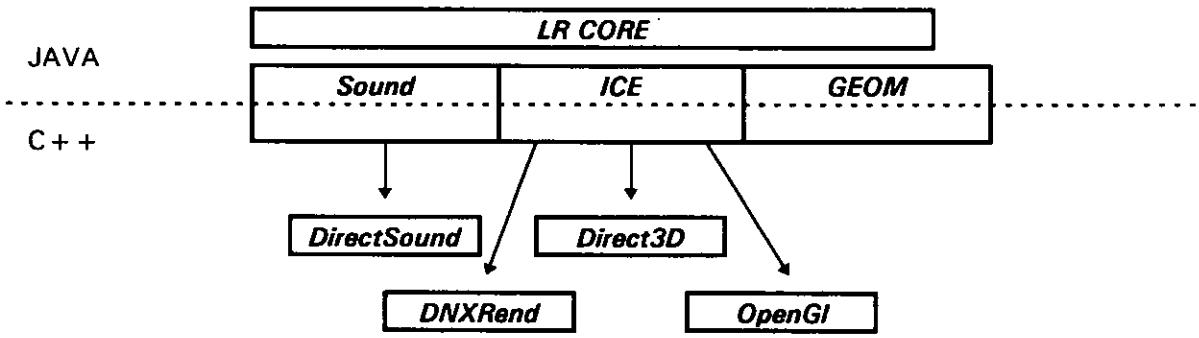


Ilustración10. Arquitectura de LR.

LRCore representa el nivel más alto de la implementación de Liquid Reality, está formado por clases Java independientes de plataforma y es el responsable de sopotar la especificación de VRML2.0. Provee al desarrollador de un API para lenguaje de alto nivel.

ICE es un API de bajo nivel, sus clases soportan la representación de objetos tridimensionales bajo la plataforma Java, sus funciones no dependen de ***LRCore*** y no tiene conocimientos de VRML2.0. Su implementación utiliza métodos nativos que interactúan directamente con el hardware específico de cada plataforma, obteniendo un excelente desempeño gráfico. ***DNXRend***, ***Direct3D*** y ***OpenGL*** son bibliotecas nativas codificadas en C y C++ en las cuales se basan funciones que requieren un alto desempeño como es el proceso de rendering.

Geom es un API que implementa funciones matemáticas para el soporte de representaciones gráficas. Provee al desarrollador el manejo de objetos geométricos como: puntos, vectores y polígonos, además de otros objetos conceptuales como son: matrices, proyecciones, traslaciones y rotaciones.

El API ***Sound*** provee de soporte para la manipulación de sonidos. A través de las bibliotecas de ***DirectSound*** interactúa con el hardware de una plataforma en específico. Las bibliotecas de ***Sound*** son utilizadas por ***LRCore*** para implementar las funciones de audio especificadas en VRML2.0.

La implementación de Liquid Reality permite el acceso a cualquiera de estas capas, la programación en alto o bajo nivel dependerá de las necesidades específicas de cada aplicación.

Arquitecturas Cliente/Servidor

El tema de arquitecturas Cliente/Servidor resulta demasiado extenso para ser tratado adecuadamente dentro de un solo capítulo, es por ello que en esta sección se pretende destacar únicamente el sentido y las ventajas de utilizar una arquitectura cliente/servidor dentro de un sistema como el nuestro.
La terminología cliente/servidor es ampliamente utilizada para definir una serie de sistemas y servicios compartidos, sin embargo no existe un consenso para definir su significado exacto. Sin embargo el concepto de este tipo de arquitecturas es que los clientes y servidores son entidades lógicas separadas que trabajan para realizar tareas en común comunicándose a través de una red. Dentro de este concepto existen una serie de entidades y características que lo complementan:

- **Servicio**: Es la principal relación existente entre procesos que corren en máquinas separadas. El proceso servidor es el proveedor de servicios. El cliente el consumidor de estos. En esencia, una arquitectura cliente/servidor provee una separación transparente de funciones entre entidades, basándose en la idea común y corriente de servicio.

- **Recursos compartidos:** Un servidor puede proveer servicios a muchos clientes al mismo tiempo, por lo tanto este conjunto de clientes tienen acceso a una serie de recursos en forma compartida.

- **Protocolos asimétricos:** Existe una relación entre clientes y servidores de muchos a uno respectivamente. El proceso cliente siempre iniciará el diálogo con la petición de algún servicio. El proceso servidor aguarda en un estado pasivo peticiones de cualesquiera de sus clientes.

- **Transparencia de locación:** El servidor como proceso puede residir en la misma máquina que sus clientes o bien en cualquiera otra dentro de una red. El cliente accesa al servidor utilizando la dirección de la máquina donde reside este último. Un mismo proceso puede ser un cliente, un servidor o ambos.

- **Mix-and-Match**: Idealmente un software cliente/servidor es independiente de plataforma de hardware y sistemas operativos, de esta forma es posible explotar al máximo características propias de cada plataforma sin menoscabo de conectividad con otras.

- **Comunicación basada en mensajes:** Clientes y servidores son entidades separadas que interactúan entre sí a través de un mecanismo de intercambio de mensajes. El mensaje es en si el medio por el cual se hacen peticiones y se da respuesta a ellas.

- **Encapsulación de servicios:** El servidor es un proceso especializado en una tarea. El cliente debe accesar al servidor adecuado al servicio que requiera, enviando los parámetros necesarios para que el servidor pueda ejecutar el trabajo correctamente. Los servidores pueden ser cambiados por nuevas versiones sin que esto implique cambios en los clientes ya que el protocolo de mensajes se conserva.

- **Escalabilidad:** Los sistemas cliente/servidor pueden ser escalados horizontal o verticalmente. La escalabilidad horizontal se refiere a la adición o remoción de clientes que pueden ser atendidos por un servidor, teniendo como resultado leves cambios en el performance del sistema; por otro lado la escalabilidad vertical se

refiere a la migración del servidor del sistema a una o más computadoras para de esta manera mejorar el desempeño.

- **Integridad:** El código y datos que forman el servidor son mantenidos en forma centralizada, garantizando con esto la integridad de los datos, ya que solo existe una sola copia de ellos, por otro lado el mantenimiento resulta más eficiente y barato pues solo debe hacerse en un solo punto.

Como puede apreciarse una arquitectura cliente servidor provee de múltiples características que resultan de gran utilidad a la implementación de un sistema complejo:

- **Procesamiento distribuido:** El mayor problema del sistema que controle al robot, es que este puede resultar demasiado grande y complejo para ser ejecutado en una pequeña computadora a bordo del robot. La propuesta para este caso es separar el sistema de decisión de forma que el cliente universal es el robot, el cual podrá contar con una sencilla computadora que controle parte de la funcionalidad del robot, operaciones básicas como lo es el control de posición de las 6 extremidades podrá ser efectuado localmente, mientras que operaciones de mayor complejidad como lo es la coordinación de las extremidades o bien la planeación de rutas para el desplazamiento serán calculadas por entidades externas.

- **Modularidad:** Dada la independencia entre las entidades cliente y servidor, y una vez definido el protocolo de mensajes a utilizar, es posible hacer cambios de forma local al código de cada servidor y aún del cliente, sin afectar el funcionamiento de otras entidades.

- **Extendible:** Tomando como base el diseño distribuido del sistema, y su infraestructura de comunicación, es posible la implantación de nuevas tecnologías como la teoría de Agentes. con la cual es posible fragmentar y utilizar en forma dinámica el sistema de decisión de acuerdo a las necesidades del robot.

Sockets

Los sockets son un mecanismo de comunicación entre procesos, permiten la transferencia de información en forma de paquetes denominados "mensajes". El intercambio de información puede llevarse a cabo dentro de la misma computadora o bien entre varias de ellas dentro de una red, en ambos casos se establece un canal de comunicación denominado "Circuito virtual".

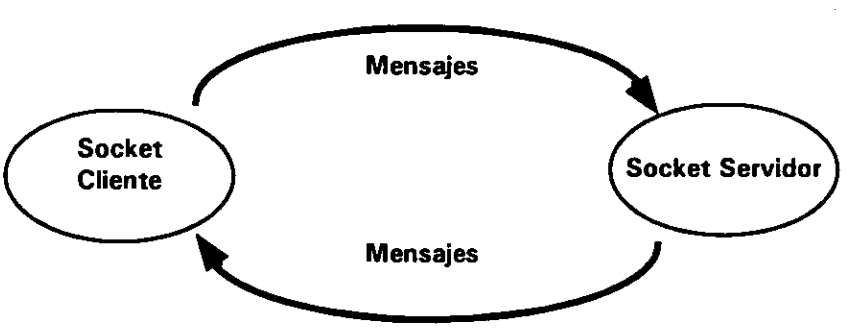


Ilustración 11. Circuito virtual.

La referencia a este socket servidor se establece utilizando 2 identificadores, uno de 32 bits con la dirección IP y otro de 16 bits para el puerto lógico, ambos correspondientes a la computadora donde radica este proceso. Al enviar una petición de servicio un socket cliente, el servidor crea un nuevo socket con el cual el cliente establecerá el intercambio de información (ver Ilustración 12).

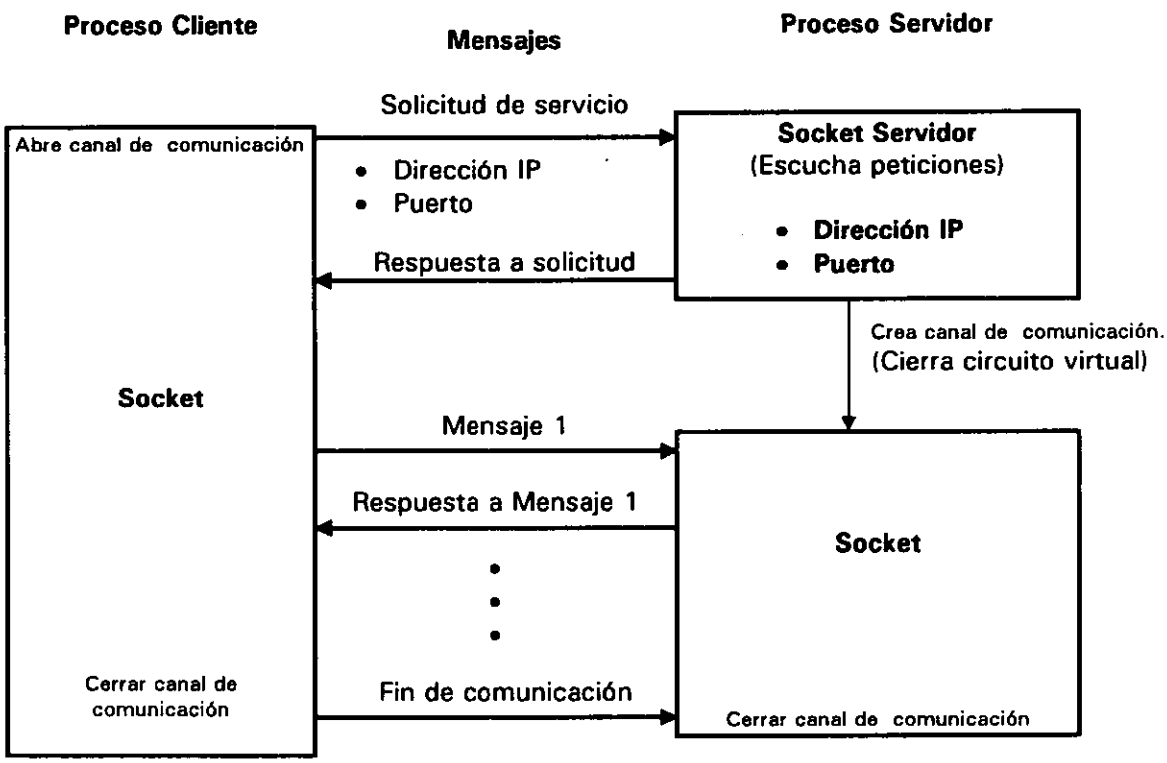


Ilustración 12. Comunicación a través de sockets .

El JDK proporciona clases dedicadas al manejo de sockets, estas se definen en el paquete java.net:

- **ServerSocket**
- **Socket**

Utilizando estas clases es posible crear sockets servidores para atender determinados puertos:

```
java.net.ServerSocket socketServidor = new ServerSocket(puerto);
```

A petición de un proceso cliente, se crea un nuevo socket con el cual establecer comunicación:

```
java.net.Socket socket = listen_socket.accept();
```

De lado del proceso cliente los sockets se crean definiendo la dirección IP y el puerto del socket servidor:

```
java.net.Socket socket = new Socket("uxmcc1.iimas.unam.mx", 10000);
```

Una vez establecido el canal de comunicación se crean las entidades para el control del flujo de datos de entrada y salida:

```
DataInputStream flujoDeEntrada = new dataInputStream(socket.getInputStream());

PrintStream flujoDeSalida = new PrintStream(socket.getOutputStream());
```

A través estas entidades de control es posible el intercambio de mensajes a un nivel alto utilizando variables tipo numérico, carácter o cadena:

```
String mensajeRecibido = flujoDeEntrada.readLine();

flujoDeSalida.println(" Coord. X "+ variableX....);
```

A pesar de existir otros mecanismos de comunicación como OLE o DDE casi de uso exclusivo de Microsoft Windows, pocos son los medios que representen un estándar aplicable en diferentes ambientes y sistemas operativos. Con el creciente uso de Internet esta necesidad de comunicación se ha resuelto haciendo estándar de un mecanismo de comunicación llamado socket, el cual ha sido portado a la mayoría de los sistemas operativos y lenguajes existentes permitiendo el intercambio de información entre ellos. Los sockets cuentan con amplio soporte para una gran cantidad de periféricos como lo son puertos seriales, módems (en todas sus variantes), y por supuesto tarjetas de red en una gran cantidad de topologías de red y protocolos. Este mecanismo por definición permite comunicar procesos a nivel aplicación (modelo OSI de 7 niveles), tanto por canales internos en una misma computadora como por canales externos en múltiples máquinas dentro de una red permitiendo así la implementación de procesamiento distribuido, entre otros.

Capítulo 3

Modelado y Descripción Mecánica

En este capítulo se describe en forma general la arquitectura del robot, el modelo matemático que lo representa y la propuesta para la implementación del sistema mecánico.

Modelo geométrico del robot

El modelo geométrico del robot se utiliza para obtener su cinemática directa e inversa. Este modelo se basa en la longitud de sus eslabones, los ángulos descritos entre ellos y la posición del extremo del robot (un punto en el espacio (x,y,z)). Para simplificar la descripción, el modelado se divide en 2 partes, la primera se refiere al modelo de un robot de 3 grados de libertad que representa a cada una de las extremidades. La segunda abarca el modelado del cuerpo del robot y su relación con cada una de las extremidades.

Modelado de las extremidades

Para obtener el modelo geométrico de las extremidades, es necesario conocer las dimensiones de los eslabones que las forman, cada una de estas representará una traslación sobre nuestra base referencial.

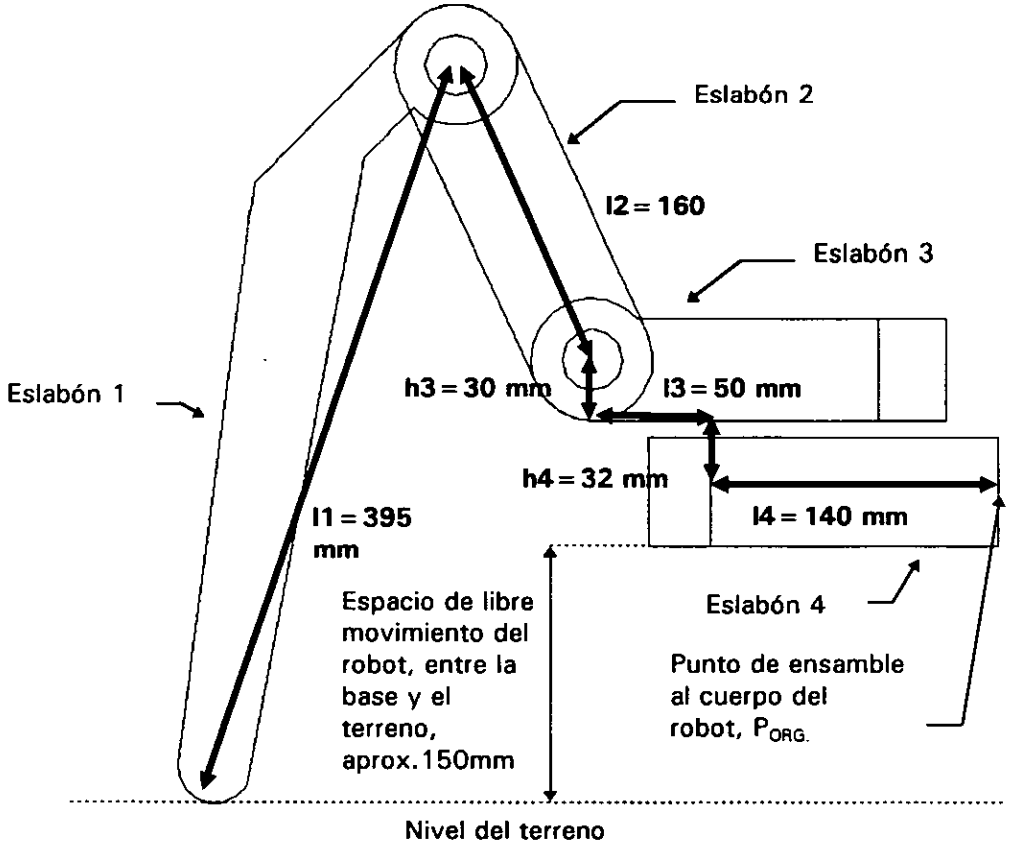


Ilustración 13. Dimensiones de una extremidad (Diseño ACAD).

Las dimensiones de cada eslabón fueron propuestas a lo largo del proceso de diseño. La longitud del eslabón 1 debe ser la suficiente para mantener un cierto espacio entre la base del robot y el terreno. Esta separación ofrece una mayor libertad de movimiento al evitar el contacto de la base del robot con algunas de la irregularidades del terreno que no fueron detectadas por las extremidades.

De igual manera, los eslabones 2, 3 y 4 deben mantener sus dimensiones en el mínimo posible, el suficiente para alojar los actuadores, mecanismos y componentes electrónicos y otro tanto para permitir el acoplamiento con los demás eslabones.

La numeración de los eslabones comienza con el que está al extremo, nombrado como "eslabón1", y así hasta el eslabón 4 el cual cuenta con la base referencial de toda la extremidad, es decir, las bases subsecuentes ({3},{2},{1}) se obtienen de traslaciones y orientaciones con referencia a la base {4}.

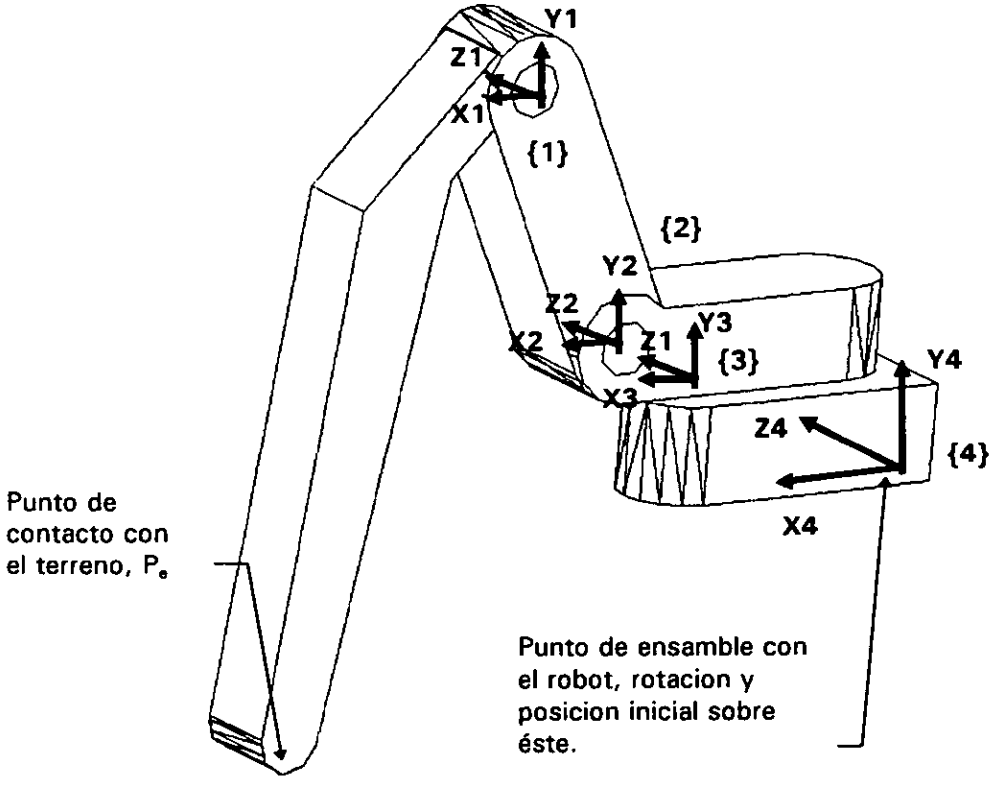


Ilustración 0-14 Ejes de rotación (Diseño ACAD).

Cinemática directa

La detección de obstáculos y el control del desplazamiento requiere de la localización de los puntos de apoyo de cada extremidad. De esta forma se puede localizar e identificar una irregularidad del terreno y calcular la trayectoria para un determinado desplazamiento.

Para conocer la posición del punto P_e de la base referencial {1} descrita en términos de la base {4} (punto de ensamble del robot), es necesario hacer una serie de traslaciones y

orientaciones sucesivas. Es decir para obtener ${}^4P_e$, primero se deberá encontrar ${}^1P_e$, y con base en éste obtener ${}^2P_e$; el proceso se repite sucesivamente hasta llegar a la base requerida.

De forma general, la descripción de un punto de una base {B} en términos de una base {A}, cuando el origen de la primera presenta un traslación, puede obtenerse de la siguiente forma:

${}^AP={}^BP+{}^AP_{ORG(B)}$, donde ${}^AP_{ORG(B)}$, es el vector de origen de {B} medido en {A}.

En términos matriciales:

$$\begin{bmatrix} {}^AP_X \\ {}^AP_Y \\ {}^AP_Z \end{bmatrix} = \begin{bmatrix} {}^BP_X \\ {}^BP_Y \\ {}^BP_Z \end{bmatrix} + \begin{bmatrix} {}^AP_{ORG(B)X} \\ {}^AP_{ORG(B)Y} \\ {}^AP_{ORG(B)Z} \end{bmatrix}$$

Así mismo puede obtenerse la descripción de un punto de una base {B} en términos de una base {A}, cuando el origen de {B} presenta una cierta rotación

${}^AP={}^A_BR\,{}^AP_{ORG(B)}$, donde ${}^A_BR$ es una matriz de rotación.

$$\begin{bmatrix} {}^AP_X \\ {}^AP_Y \\ {}^AP_Z \end{bmatrix} = \begin{bmatrix} (X_B,X_A) & (Y_B,X_A) & (Z_B,X_A) \\ (X_B,Y_A) & (Y_B,Y_A) & (Z_B,Y_A) \\ (X_B,Z_A) & (Y_B,Z_A) & (Z_B,Z_A) \end{bmatrix} * \begin{bmatrix} {}^BP_X \\ {}^BP_Y \\ {}^BP_Z \end{bmatrix}$$

Para simplificar los cálculos de las rotaciones podemos tomar en cuenta solo rotaciones sobre los ejes de referencia, es decir rotaciones canónicas:

$${}^A_BR = ROT(x,\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\operatorname{sen}\theta \\ 0 & \operatorname{sen}\theta & \cos\theta \end{bmatrix}$$, rotación canónica sobre el eje X.

$${}^A_BR = ROT(y,\theta) = \begin{bmatrix} \cos\theta & 0 & \operatorname{sen}\theta \\ 0 & 1 & 0 \\ -\operatorname{sen}\theta & 0 & \cos\theta \end{bmatrix}$$, rotación canónica sobre el eje Y.

$${}^A_BR = ROT(z,\theta) = \begin{bmatrix} \cos\theta & -\operatorname{sen}\theta & 0 \\ \operatorname{sen}\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$, rotación canónica sobre el eje Z.

Ambas operaciones de traslación y orientación se aplican en una sola operación para la obtención de las referencias de cada eslabón. La matriz que representa a la rotación y a la traslación es denominada como matriz de transformación:

$$ {}^{A}_{B}T = \left| \begin{array}{c:c} {}^{A}_{B}R & {}^{A}P_{ORG(B)} \\ \hdashline 0 & 1 \end{array} \right| $$

Esta contiene la información necesaria que especifica la relación que existe entre una base y otra. Al operar un punto (x,y,z) con ella se cambia su base referencial, en este caso de {B} a {A}.

$$ {}^{A}P = {}^{A}_{B}T\,{}^{B}P $$

Basado en éste resultado, la posición que presenta el extremo del robot queda expresada de la siguiente forma:

${}^{2}P = {}^{2}_{1}T\,{}^{1}P_{e}$, descripción de un punto sobre la base {1} en términos de la base {2}.

${}^{3}P = {}^{3}_{2}T\,{}^{2}P_{e}$, descripción de un punto sobre la base {2} en términos de la base {3}.

${}^{4}P = {}^{4}_{3}T\,{}^{3}P_{e}$, descripción de un punto sobre la base {3} en términos de la base {4}.

Sustituyendo términos:

${}^{4}P = ({}^{4}_{3}T\,{}^{3}_{2}T\,{}^{2}_{1}T)\,{}^{1}P_{e} = {}^{4}_{1}T\,{}^{1}P_{e}$, descripción del punto de apoyo de la extremidad de la base {1} en términos de la base {4}.

De esta forma queda resuelto el problema de obtener la cinemática directa para las extremidades.

Cinemática Inversa

Para obtener la cinemática inversa de la extremidad se utiliza el método geométrico. Este se auxilia de una serie de triángulos implícitos en la geometría del robot, cuyas dimensiones están definidas por la longitud de los eslabones y la posición del punto P_e. Con base a las dimensiones del triángulo es posible obtener la magnitud de sus ángulos y con ello los ángulos de rotación de cada uno de los eslabones.

El ángulo denominado como θ_1 se obtiene del triángulo "A" cuyos lados corresponden a los eslabones 1 y 2:

$$ \theta_1 = a\cos\left(\frac{d^2 - l_1^2 - l_2^2}{(-2l_1l_2)}\right) $$

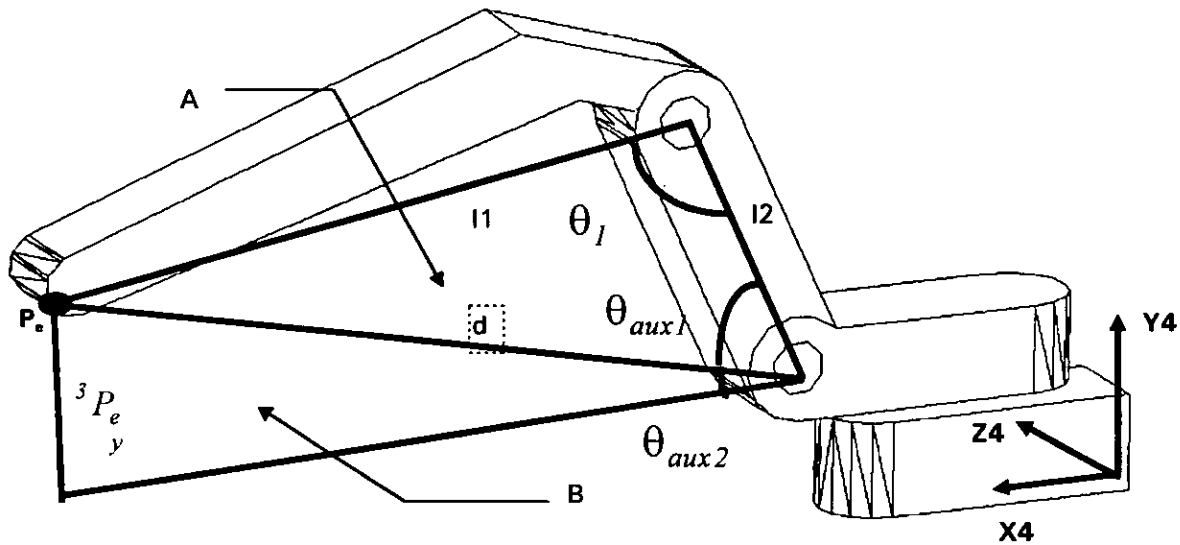


Ilustración 15. Vista lateral del robot, obtención de θ_1 y θ_2 (Diseño ACAD).

Donde el la variable "d" indica la distancia entre el punto P_e y el punto de inserción del eslabón 2.

El ángulo denominado como θ_2 se obtiene en dos fases:

$$\theta_2 = \theta_{aux1} + \theta_{aux2}$$

El ángulo θ_{aux1} pertenece al triángulo "A" cuyas dimensiones I1, I2 y d ya conocemos.

$$\theta_{aux1} = a\cos\left(\frac{l_1^2 - d^2 - l_2^2}{(-2dl_2)}\right)$$

El ángulo θ_{aux2} se obtiene del triángulo rectángulo "B":

$$\theta_{aux2} = atan\left(\frac{^3P_e}{\frac{y}{d}}\right)$$

La magnitud del cateto opuesto es la componente Y de P_e en la base referencial 3 (^3P_e)
$_y$

El ángulo θ_2 queda expresado como:

$$\theta_2 = a\cos\left(\frac{l_1^2 - d^2 - l_2^2}{(-2dl_2)}\right) + atan\left(\frac{{}^3P_{e_y}}{d}\right)$$

Finalmente el ángulo θ_3 descrito por los eslabones 3 y 4 se obtiene del triángulo rectángulo C, y queda expresado como:

$\theta_3 = atan(z'/x')$ para x > 0.
$\theta_3 = 180 + atan(z'/x')$ para x < 0

Donde la magnitud del cateto opuesto es $z' = {}^4P_{e_z} - {}^4P_{ORG(3)z}$ y del adyacente $x' = {}^4P_{e_x} - {}^4P_{ORG(3)_x}$.

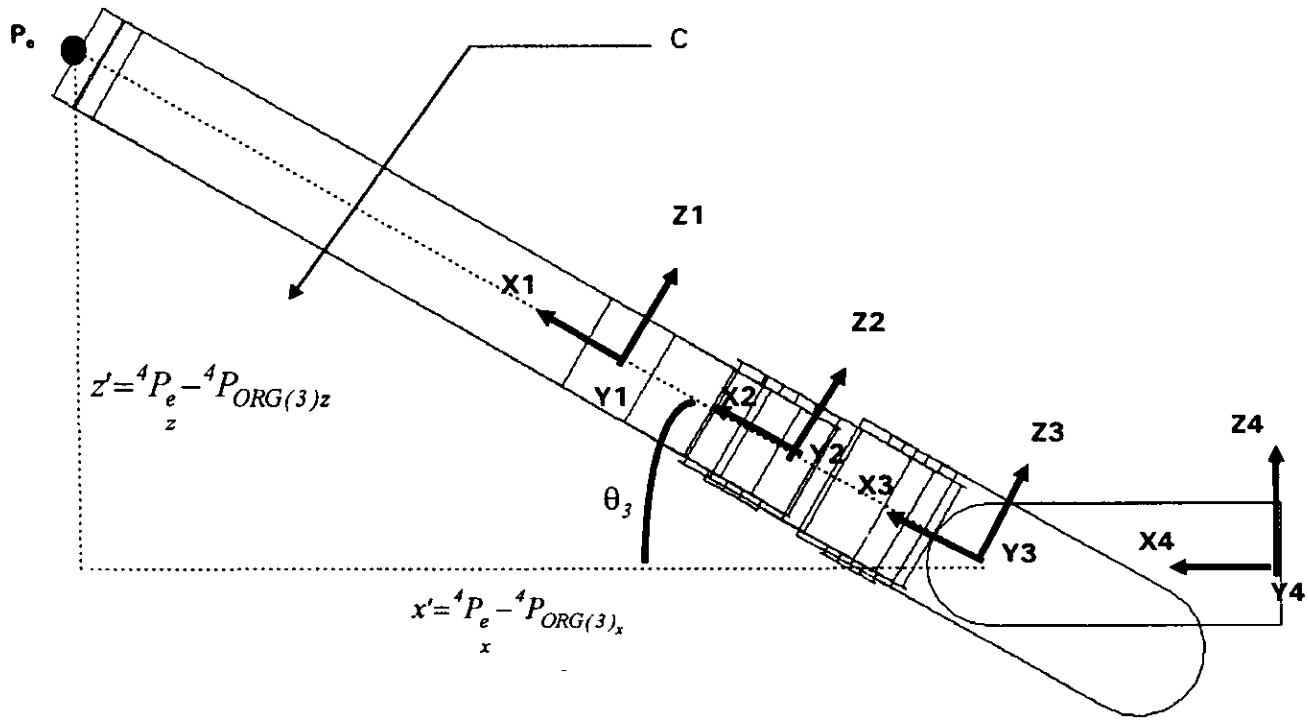


Ilustración 16. Vista superior de la extremidad, obtención de θ_3 (Diseño ACAD).

De esta forma es posible obtener las rotaciones requeridas en cada uno de los eslabones para posicionar P_e en un punto x y z dado.

Modelado del cuerpo del robot

Para el modelo geométrico del cuerpo del robot debemos tener en cuenta que éste es un eslabón común a las extremidades, del cual depende la posición y orientación de estas últimas. Las extremidades son fijadas al rededor del cuerpo, dándoles una posición y orientación diferente a cada una en relación a los ejes del cuerpo del robot {R}. La base {R} es desplazada y orientada de acuerdo al movimiento del robot, con ello las bases referenciales de cada extremidad también son modificadas con relación a la base {Te} del terreno.

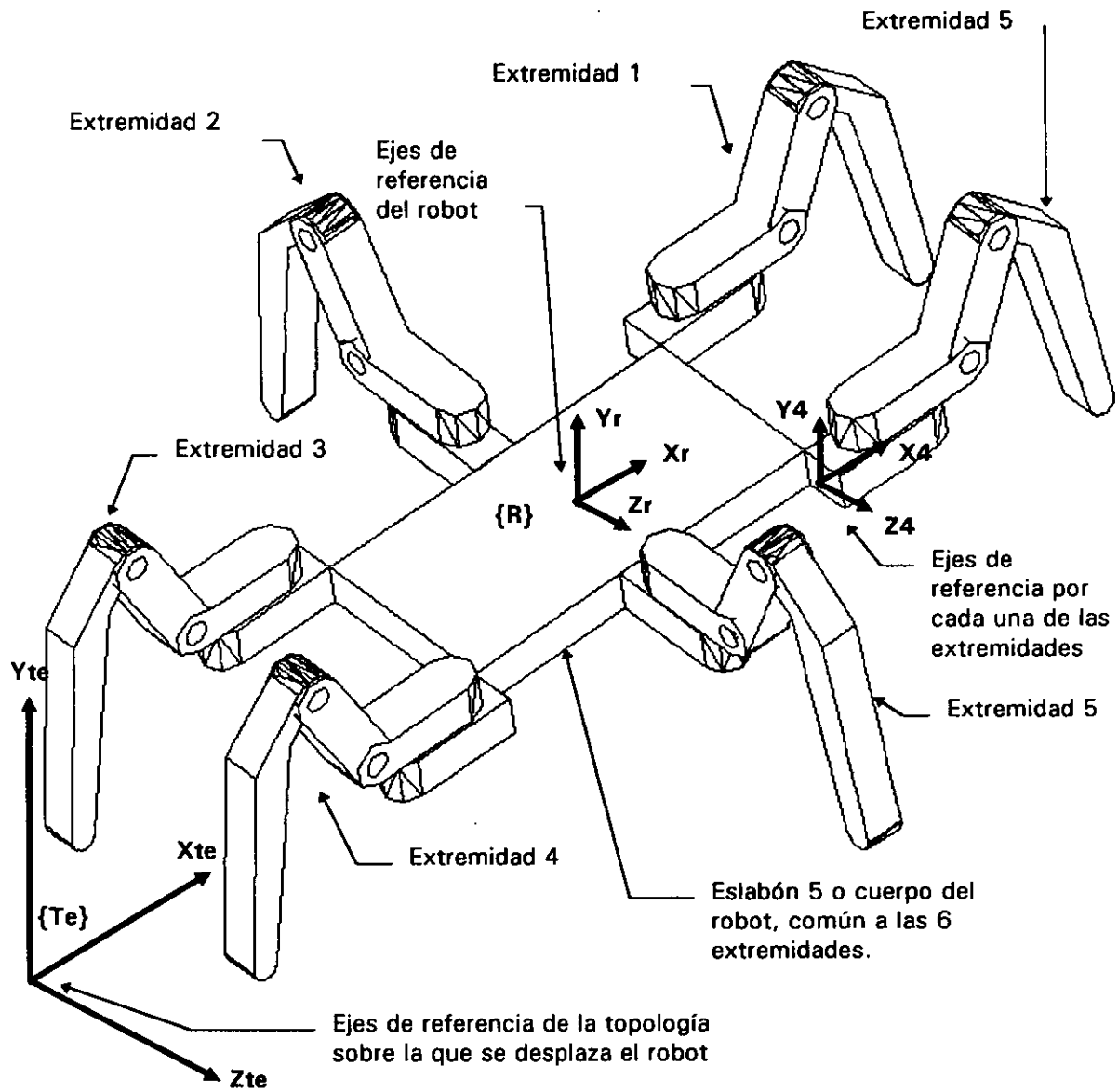


Ilustración 17. Ejes de referencia del robot y sus respectivas extremidades (Diseño ACAD).

El desplazamiento y rotación del robot es relativo al terreno sobre el cual se desplaza éste. El desplazamiento está representado como una traslación de los ejes de la base {R}. Así mismo la

rotación del robot se efectúa sobre el eje Y de la base {R}. Estas transformaciones quedan representadas por las siguientes ecuaciones:

${}^{R}P={}^{R}_{4}T\,{}^{4}P_{e}$, descripción de un punto sobre la base {4} en términos de la base {R}.

${}^{Te}P={}^{Te}_{R}T\,{}^{R}P_{e}$, descripción de un punto sobre la base {R} en términos de la base {Te}.

Extendiendo el modelo de la extremidad (la cual solo contempla de la base {1} hasta la base {4}) :

${}^{Te}P=({}^{Te}_{R}T\,{}^{R}_{4}T\,{}^{4}_{3}T\,{}^{3}_{2}T\,{}^{2}_{1}T)\,{}^{1}P_{e}={}^{Te}_{1}T\,{}^{1}P_{e}$, *descripción del punto de apoyo de la extremidad de la base {1}, en términos de la base del terreno sobre el cual se desplaza el robot.*

La relación entre los modelos se especifica en el código del software de simulación y control, donde el cuerpo del robot y cada una de las extremidades son representadas por una entidad lógica. Los cambios ocurridos por desplazamiento u orientación son comunicados de una entidad a otra, de esta forma se adecuan los parámetros de su modelo geométrico a los nuevas condiciones.

Descripción mecánica

El sistema mecánico que a continuación se describe es una propuesta basada en el modelo geométrico de las extremidades. Este sistema provee de movimiento a los eslabones a través de mecanismos de transmisión acoplados a los actuadores; por otro lado brinda soporte y alojamiento a otros sistemas del robot.

Mecanismos de transmisión

Estos dispositivos adecuan el movimiento de los actuadores para que pueda ser utilizado en rotar los eslabones. Los motores se ubican dentro de cada eslabón de forma longitudinal, bajo de esta disposición el eje del motor queda en forma transversal al eje del eslabón. Por otro lado la fuerza desarrollada por cada uno de estos actuadores no es suficiente para soportar su propio peso y el de la estructura del eslabón.

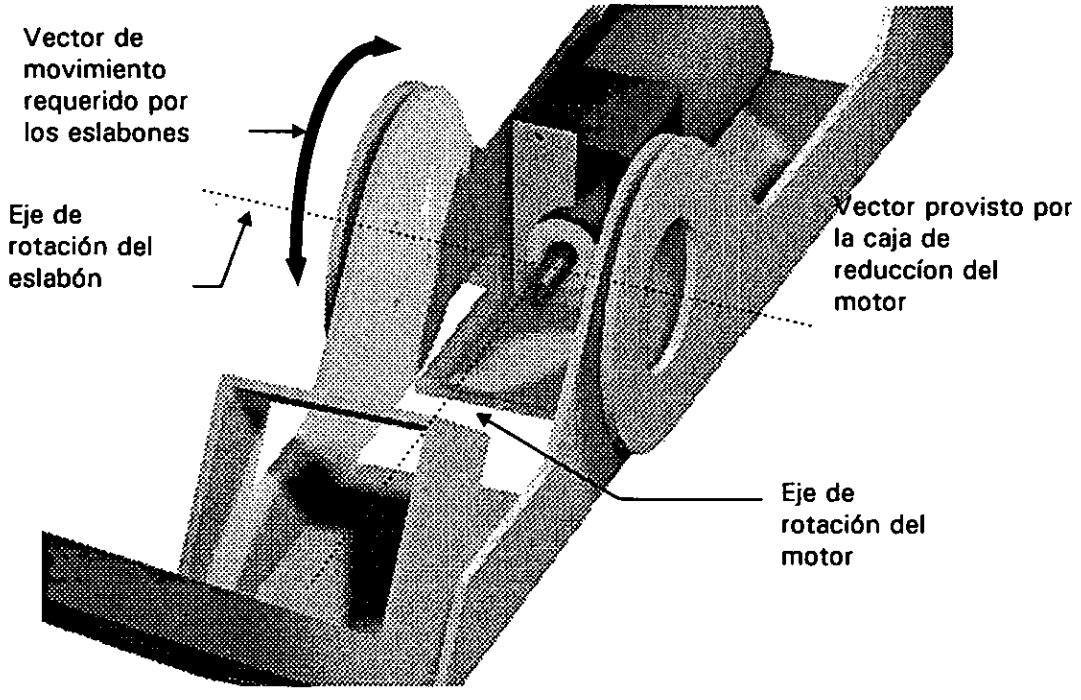


Ilustración 18. Vista del actuador y su disposición dentro de cada eslabón (diseño en ACAD, visualización 3D Studio).

Ambos problemas pueden solucionarse de forma sencilla utilizando una caja de transmisión con un tornillo sin fin y engrane, de esta forma el movimiento se transmite de manera transversal a la vez que se puede multiplicar el torque dependiendo de la relación entre el engrane y el tornillo.

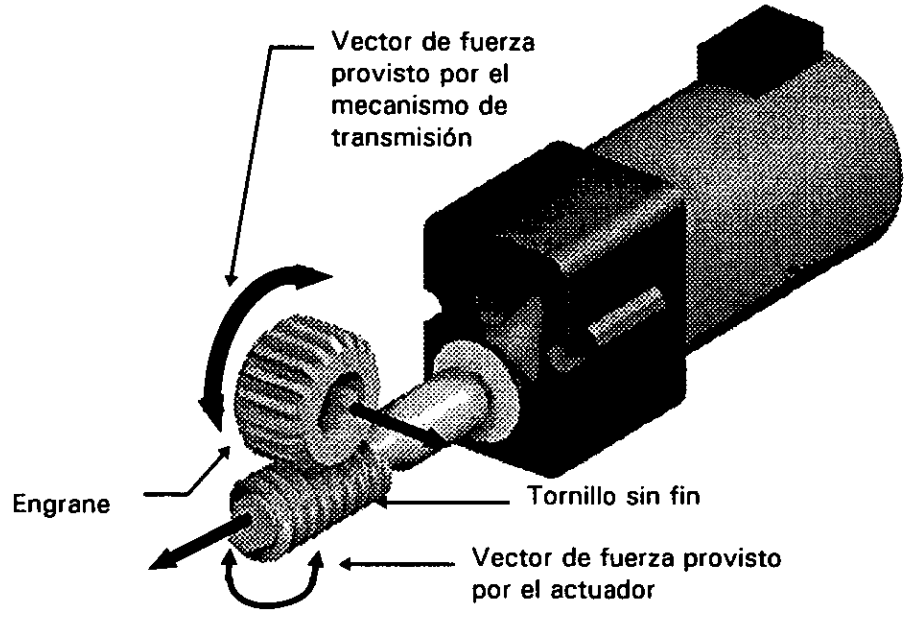


Ilustración 19. Vista del actuador y mecanismo de transmisión (diseño en ACAD, visualización 3D Studio).

Bajo esta configuración el movimiento solo puede ser transferido del tornillo sin fin, cuya fuerza proviene del actuador, hacia el engrane. Esta característica ayuda a anular el

movimiento proveniente del eslabón hacia el actuador, provocado en muchos casos por la inercia de éste o por el mismo peso del dispositivo en reposo. Este último, resulta en el ahorro de la energía que sería necesaria aplicar al actuador para mantener su posición una vez ubicado. Bajo este esquema de transmisión la fuerza del actuador se multiplica veinte veces, así como la velocidad se reduce en la misma proporción. El engrane usado por el mecanismo es fijado a un eje transversal con un par de discos perforados, sobre los cuales se fija la estructura de aluminio del eslabón.

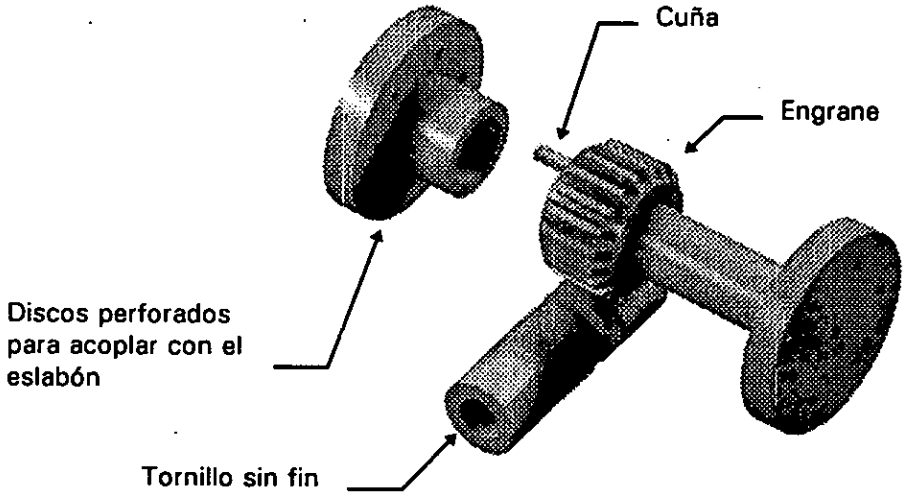


Ilustración 20. Vista engrane, tornillo y soporte (diseño en ACAD, visualización 3D Studio).

Estos componentes son alojados dentro en una caja como la que se muestra a continuación:

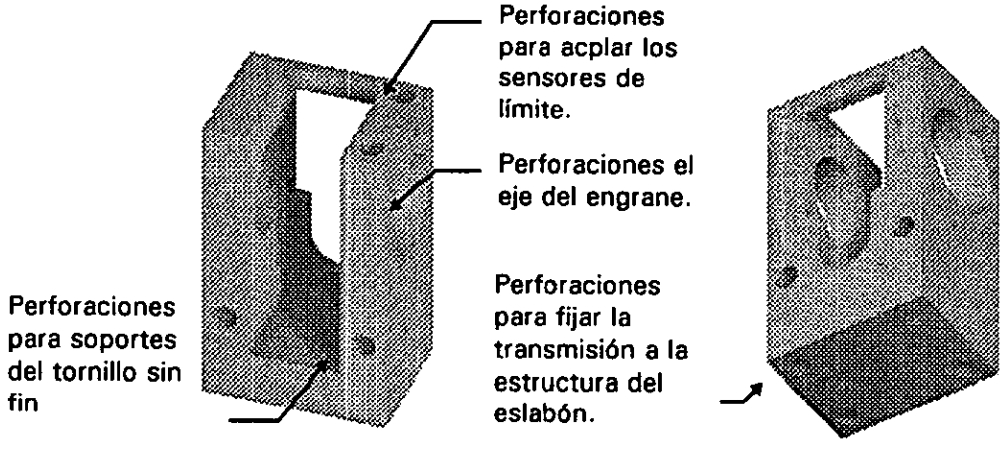


Ilustración 21. Vista caja de la transmisión (diseño en ACAD, visualización 3D Studio).

El tornillo sin fin se coloca sobre un par de soportes que proporcionan un medio para ajustar el empalme entre tornillo y engrane. Se utilizan un par de anillos de material plástico a manera de rodamientos para permitir el libre giro del eje sobre la caja.

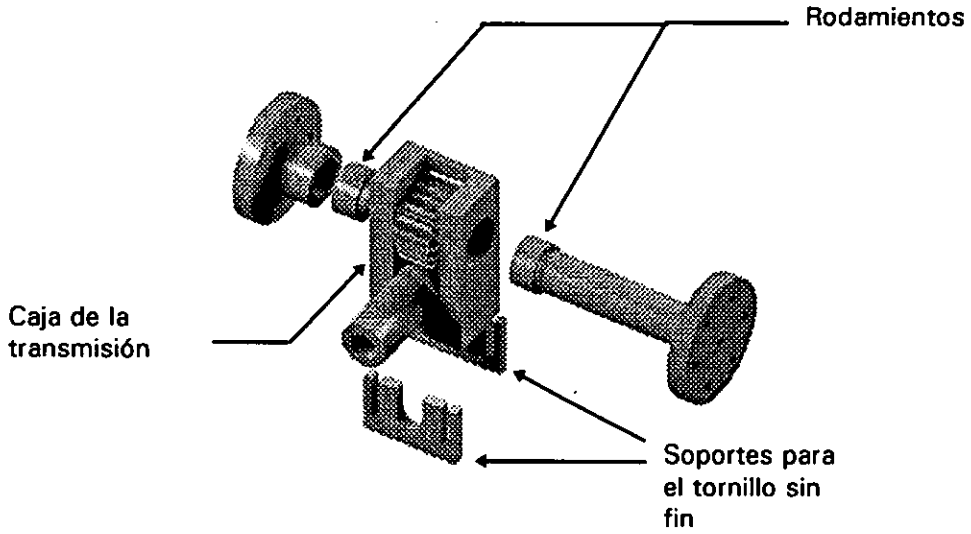


Ilustración 22. Vista de la caja de transmisión, y piezas de ajuste (diseño en ACAD, visualización 3D Studio).

Finalmente en la Ilustración 23 puede observarse la caja de transmisión completa. Los discos de nylon se utilizan para acoplar con la estructura de aluminio de los eslabones. El soporte para el motor fija la caja de transmisión con el actuador.

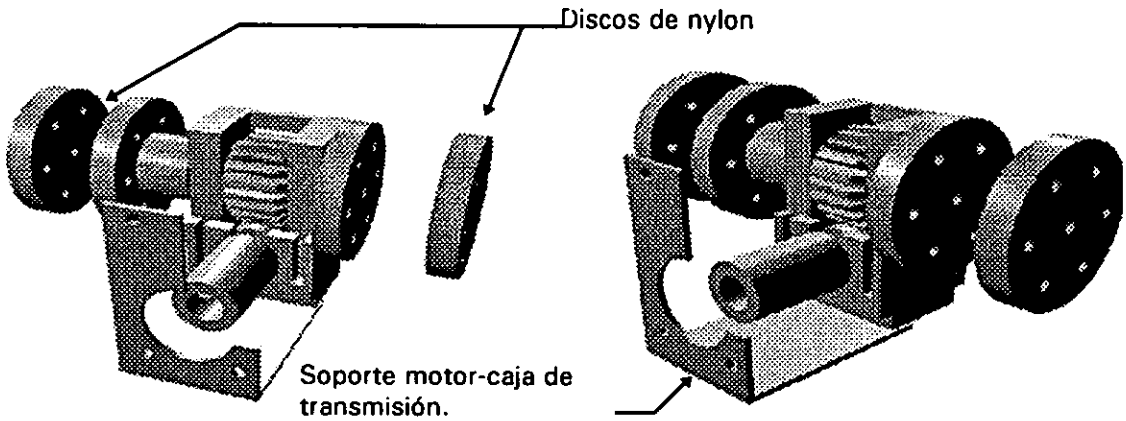


Ilustración 23. Vista de la caja de transmisión ensamblada (diseño en ACAD, visualización 3D Studio).

Soporte mecánico

Como se había mencionado anteriormente, el sistema mecánico de un robot está formado básicamente por una serie de eslabones, estas estructuras tienen la función de dar soporte y alojamiento a los componentes internos, como son el cableado, actuadores, sensores y circuitos electrónicos. Para un buen desempeño, la estructura de cada eslabón debe ser de un material rígido que soporte el peso de los componentes y el esfuerzo aplicado para producir el desplazamiento sin presentar deformaciones que afecten la efectividad del modelo geométrico. Por otro lado estas estructuras deben ser ligeras, con el fin de invertir una cantidad mínima de energía en movilizarlas. Bajo estos requerimirentos se pensó en el uso de canaletas de aluminio

con perforaciones que permitan aligerar el peso. En las siguientes figuras se muestran las estructuras de aluminio de los eslabones de las extremidades:

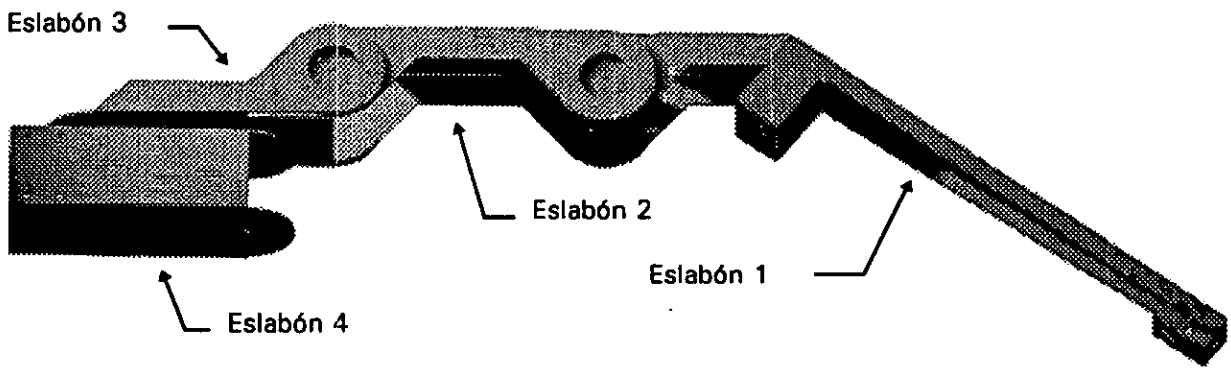


Ilustración 24. Vista de los 4 eslabones que componen la extremidad (diseño en ACAD, visualización 3D Studio).

Los eslabones de aluminio cuentan con una abertura longitudinal, por la cual se introducen y en su caso ajustan las partes internas. Para sellar cada eslabón se utilizan una serie de placas termoformadas de estireno, las cuales simplemente se atornillan a la base de aluminio. De igual forma los mecanismos de transmisión y los motores se ajustan a esta base atornillando cada parte.

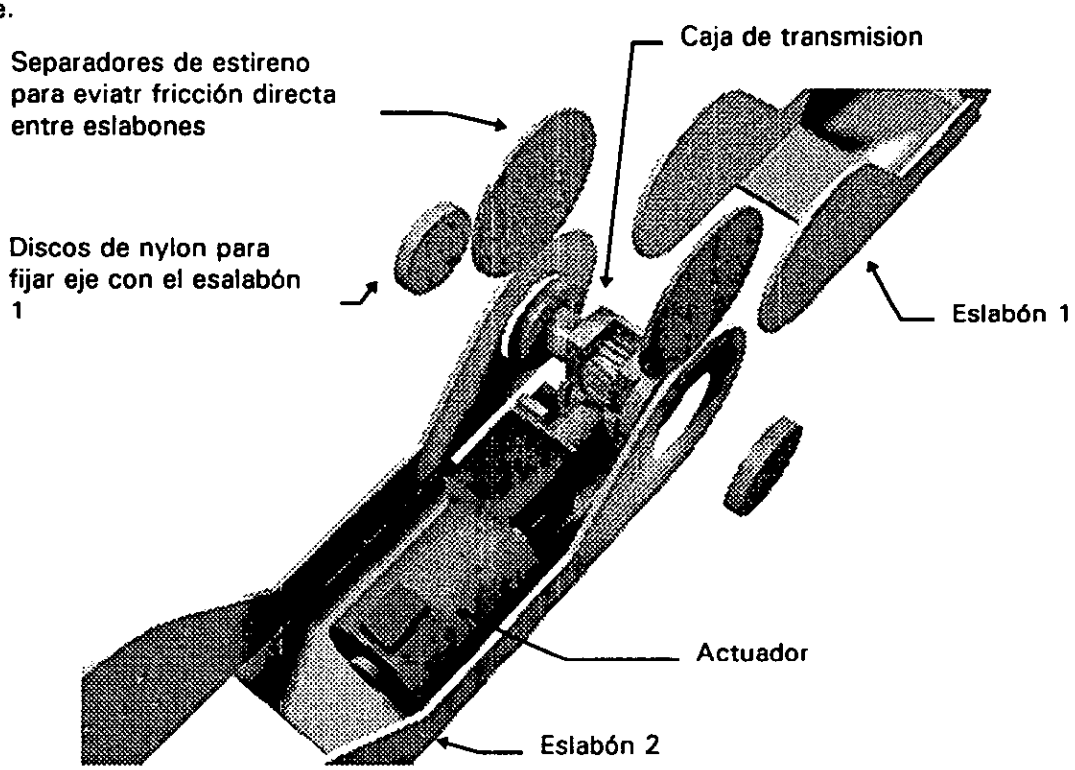


Ilustración 25. Vista del actuador y mecanismo de transmisión montado sobre la estructura del eslabón (diseño en ACAD, visualización 3D Studio).

El conjunto de señales provenientes de los sensores, como las que proporcionan energía a los actuadores son transferidas a través de un bus de cable plano, el cual pasa a través de los eslabones.

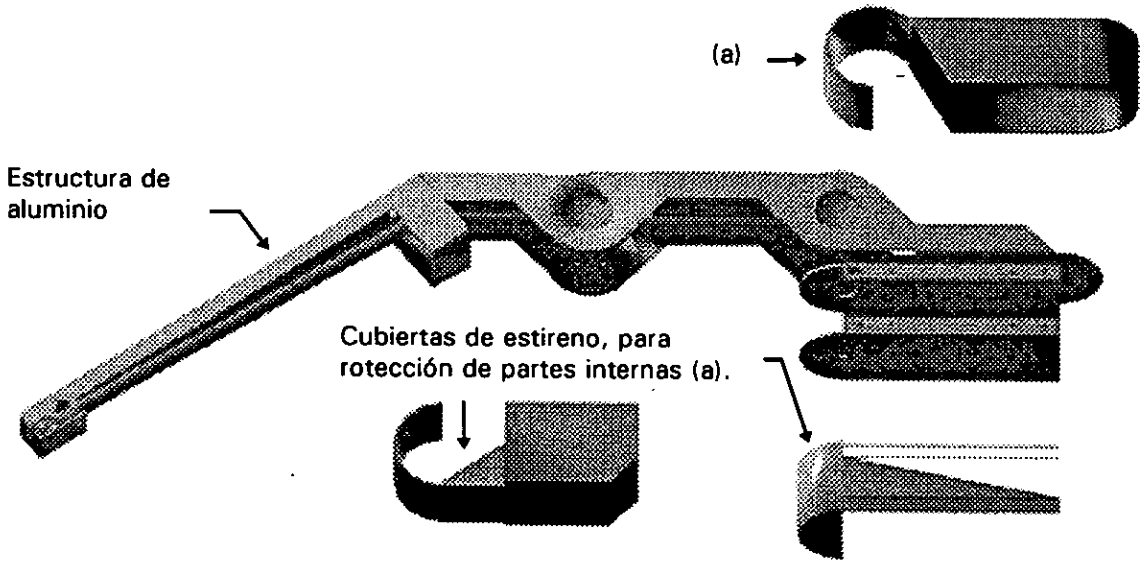


Ilustración 26. Vista de los componentes de la estructura de aluminio y sus protecciones de estireno (diseño en ACAD, visualización 3D Studio).

El diseño del cuerpo del robot (eslabón 5), donde se alojan los circuitos de multiplexión y potencia, se deberá posponer hasta la realización de las últimas fases de implementación del proyecto; esto con el fin de efectuar pruebas de espacio interior, disposición de las extremidades, etc.

Capítulo 4

Interfaz electrónica

La interfaz electrónica permite la comunicación de la computadora con los elementos de percepción y actuadores que constituyen al robot. Su diseño depende en gran medida de las características de estos últimos, por lo que antes de proceder con su descripción trataremos, en forma breve, el tema de los sensores y motores utilizados en la implementación.

Sensores internos

Los sensores internos perciben del estado de las articulaciones del robot. Los límites de rotación se fijan utilizando sensores de contacto. Los pasos intermedios que constituyen la posición relativa se contabilizan utilizando sensores optoelectrónicos. Estos últimos cambian el estado lógico de un circuito si se interrumpe su haz de luz. Acoplando un disco ranurado al eje de rotación del actuador, como se muestra en la Ilustración 27, pueden generarse pulsos que correspondan al desplazamiento de éste. Los cambios de estado (0-1) son contabilizados por la computadora y con base en esto se determina la posición relativa del eslabón.

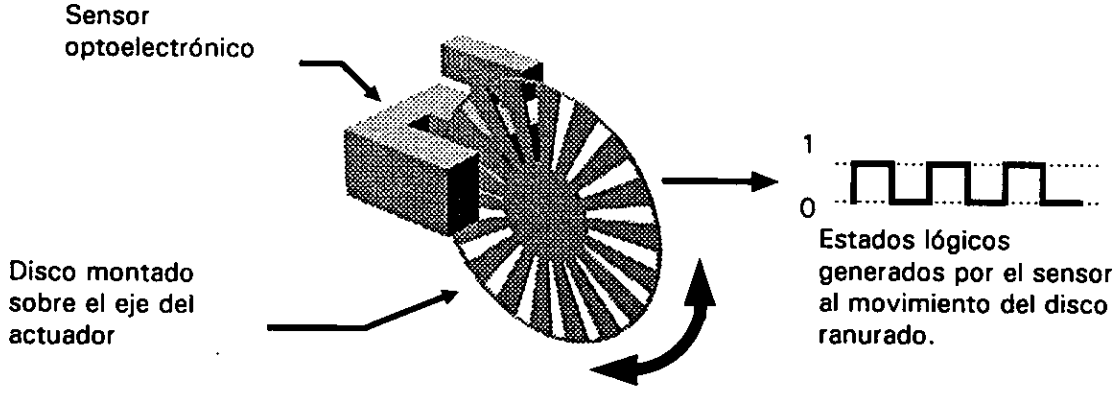


Ilustración 27. Sensor optoelectrónico y disco ranurado (Visualización en 3D Studio)

Los sensores de límite son un par de escobillas que se deslizan sobre el eje de rotación del eslabón, cada una de ellas detecta el límite de rotación en un sentido, identificando los extremos como límite superior e inferior (ver Ilustración 28). Mientras la escobilla este en contacto con el eje se mantiene su estado lógico en 0, al entrar en contacto con una zona no conductora cambia su estado a 1, lo cual indica que se ha llegado al límite de rotación (ver Tabla 1).

Señal de límite superior	Sensor de límite inferior	Descripción
0	0	Rotación libre en ambos sentidos
0	1	Límite superior
1	0	Límite interior
1	1	No válido (mutuamente excluyentes)

Tabla 1. Codificación de las señales de límites de rotación.

Los sensores se acoplan sobre la caja de transmisión de cada eslabón donde se tiene acceso al eje del actuador para detectar su posición y al eje de rotación del eslabón donde se fijan los límites.

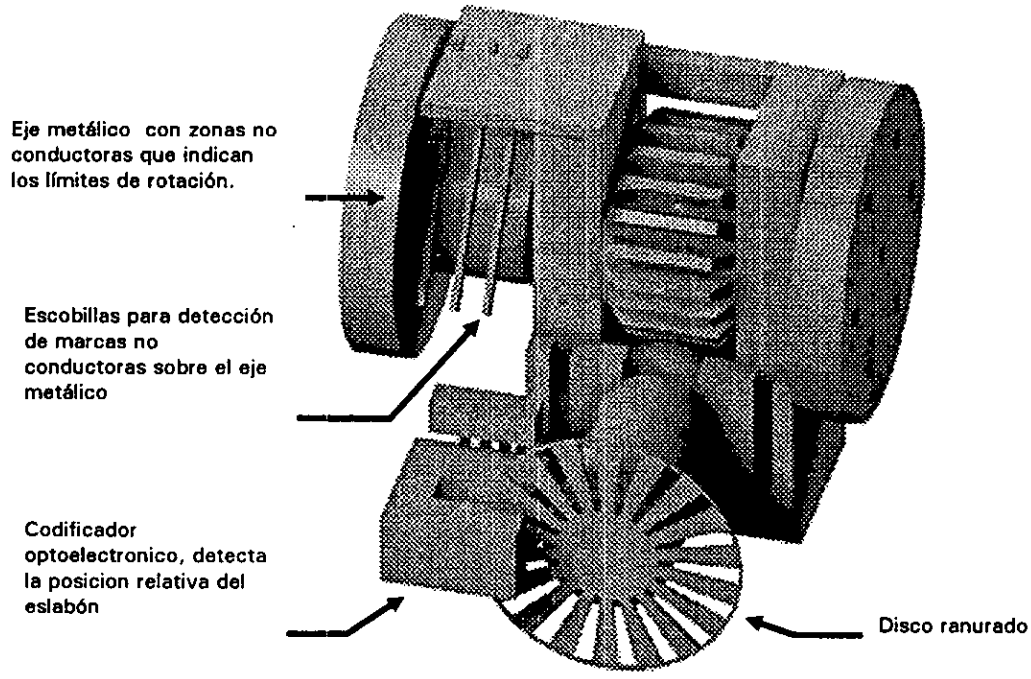


Ilustración 28. Vista de la caja de transmisión junto con sensores internos (diseño en ACAD, visualización 3D Studio)

El conjunto de sensores aquí descritos corresponde a cada una de las articulaciones del robot, por lo que cada una de las 6 extremidades requerirá de 3 sensores optoelectrónicos y 6 para los límites. El dispositivo de orientación en coordenadas esféricas solo tiene 2 articulaciones y por ende solo 2 sensores optoelectrónicos y 4 sensores de límite.

Sensores externos

Los sensores externos aportan información acerca de la topología del terreno sobre el que se desplaza el robot. A través de sensores de contacto es posible detectar irregularidades y adaptar el movimiento de las extremidades del robot para cada caso en especifico. En el extremo del eslabón 1 se disponen 5 sensores en forma ortogonal con el fin de garantizar la detección de objetos en múltiples direcciones. Los sensores están conectados en paralelo (a manera de una AND "alambrada") por lo que solo se produce una señal al contacto de cualquiera de ellos. La localización del obstáculo es estimada por software, teniendo en cuenta el sentido del desplazamiento de la extremidad y considerando que el contacto es provocado por el desplazamiento del robot y no por el movimiento del objeto. Si la extremidad realiza un desplazamiento hacia adelante o hacia arriba y se detecta un obstáculo, el software interpretará su ubicación al frente o arriba respectivamente (ver Ilustración 29).

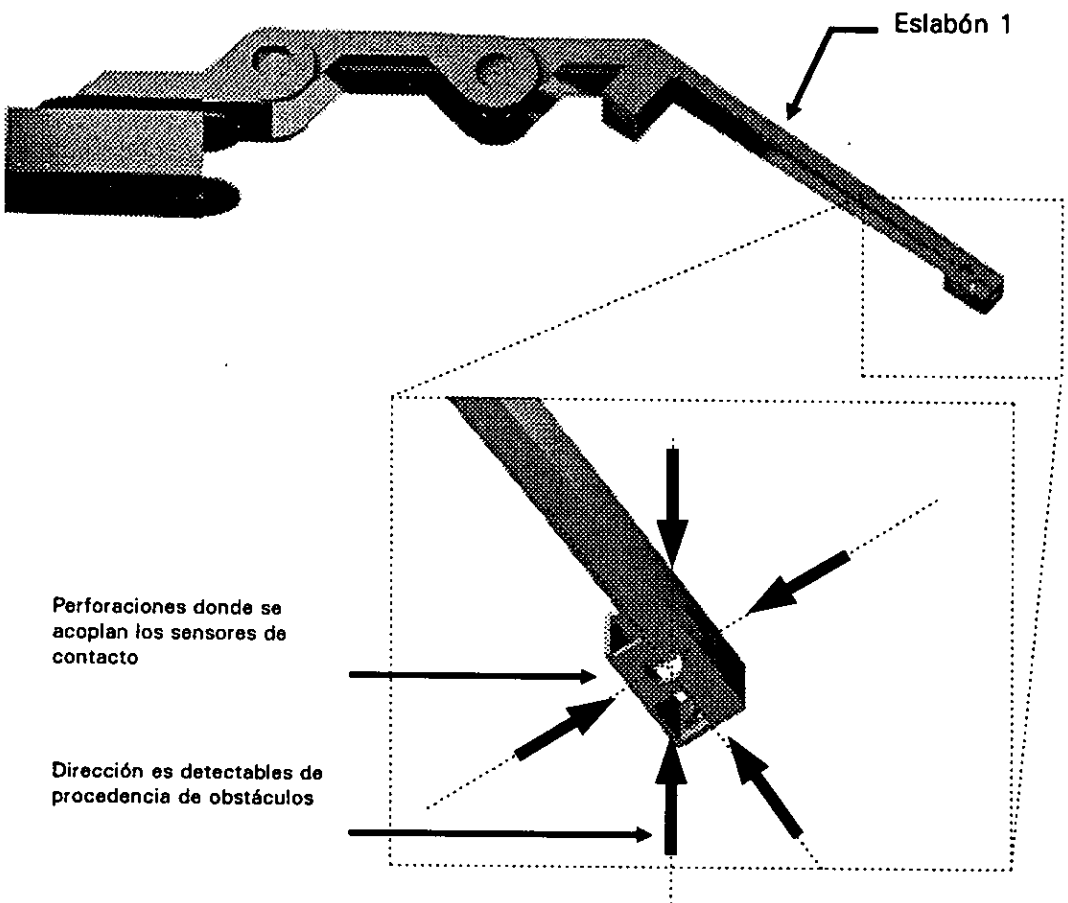


Ilustración 29. Estructura de aluminio de la extremidad. En el acercamiento puede apreciarse la disposición de los sensores, así como el campo de percepción de estos (Visualización en 3D Studio).

La información generada por estos sensores se describe en la siguiente tabla:

Señal del contacto	Descripción
0	Contacto con el terreno o algún obstáculo .
1	Sensor libre, no hay contacto físico.

Tabla 2. Interpretación de las señales de los sensores de contacto

Actuadores

Los actuadores utilizados para proveer de movimiento a los eslabones son motores de DC de la marca ITT con caja de reducción integrada, cuya relación es de 1:10. El peso del conjunto es de 150 g. Su consumo de energía, con una alimentación de 12 V, varía de 0.3 A sin carga a 1 A con una carga máxima. El torque máximo obtenido es de 0.0525 Kg*m. En la siguiente figura se muestra una vista de uno de los actuadores así como de algunos de sus componentes.

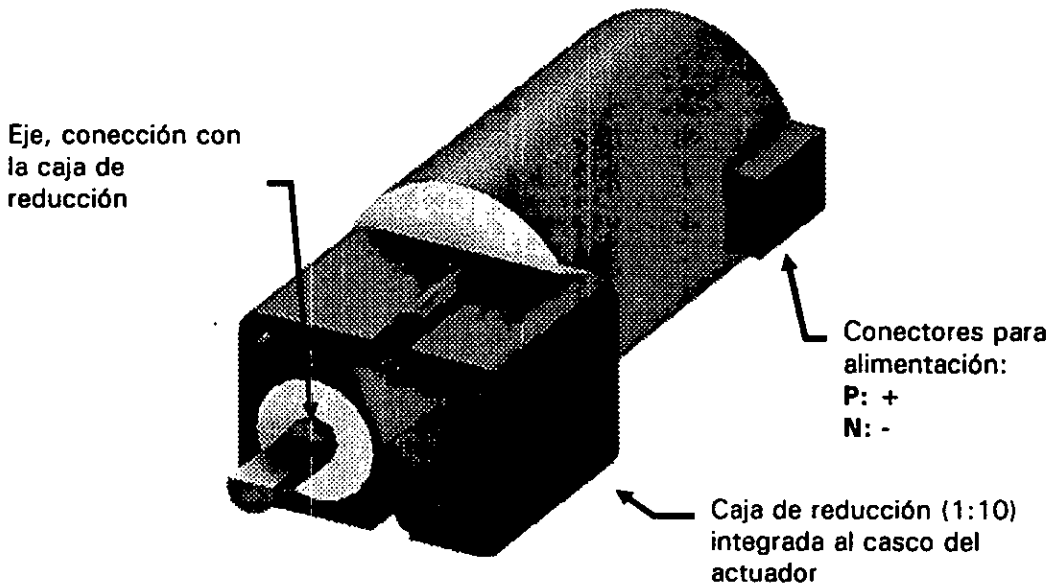


Ilustración 30. Descripción de uno de los motores (diseño en ACAD, visualización 3D Studio).

El control del movimiento del actuador se logra al cambiar la diferencia de potencial en sus terminales (designadas como P y N) , teniendo como resultado: rotación en sentido horario, antihorario o bien sin movimiento (ver Tabla 3)

Terminal P	Terminal n	Código binario (PN)	Descripción
Tierra	Tierra	00	Sin actividad
Alimentación (12 V)	Tierra	10	Rotación en sentido antihorario (N)
Tierra	Alimentación (12 V)	01	Rotación en sentido horario (P)
Alimentación (12 V)	Alimentación (12 V)	11	Sin Actividad

Tabla 3. Códigos de control para los actuadores.

Cada una de las 6 extremidades utiliza 3 actuadores, 1 por cada articulación, mientras que el dispositivo de orientación en coordenadas esféricas solo requiere 2.

Interfaz electrónica

La interfaz electrónica permite a la computadora interactuar con los actuadores y sensores del robot a través de un par de puertos paralelos de impresión. Su diseño contempla el manejo de 7 dispositivos genéricos a través de un circuito independiente por cada uno. Cada una de las 6 extremidades y el mecanismo de orientación de la cámara, son considerados como dispositivos genéricos con 3 actuadores, 3 sensores de posición, 6 de límite y un sensor externo de contacto. Bajo esta consideración el circuito de interfaz del dispositivo es idéntico en cada caso, el control específico de cada uno es delegado al software de control.
La comunicación con los circuitos de cada dispositivo se lleva a cabo mediante un circuito multiplexor. Este circuito utiliza 2 puertos paralelos de la computadora. Uno de ellos, denominado puerto de datos, forma un bus con 8 bits de entrada y 6 bits de salida, al cual tienen acceso a cada uno de los circuitos de los dispositivos. El segundo puerto contiene los circuitos de selección para cada uno de los canales del multiplexor, a través del cual se direcciona el dispositivo que tendrá acceso al bus común (ver Ilustración 31). En la Ilustración 32 donde se muestran los elementos que forman el circuito multiplexor y el circuito de interfaz de cada dispositivo.

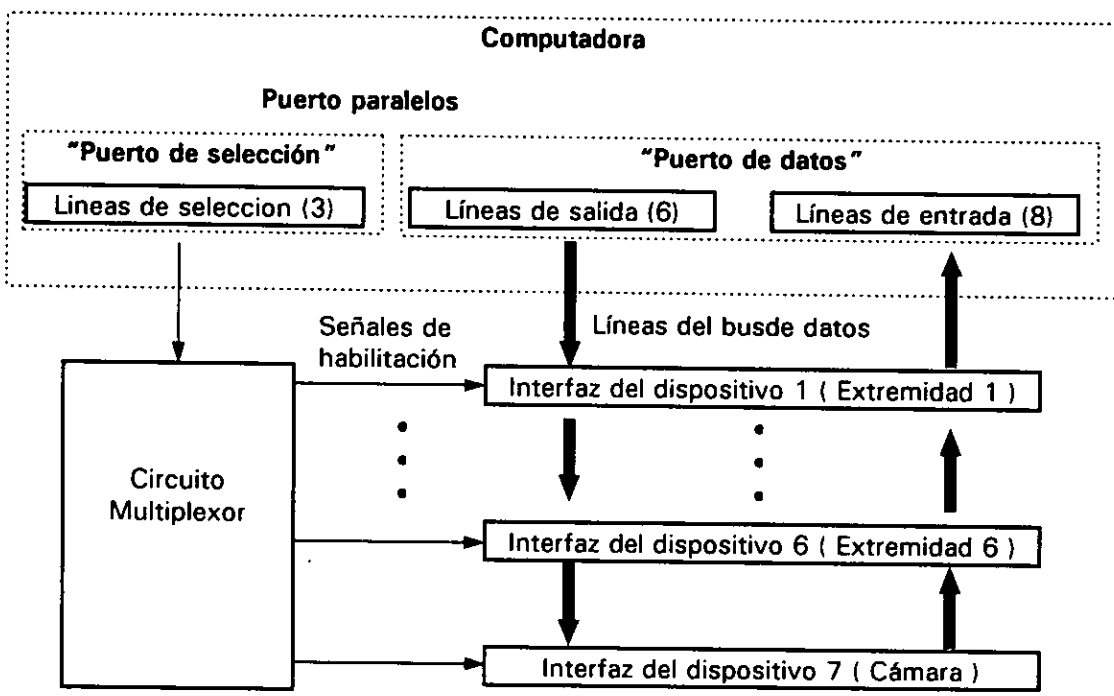


Ilustración 31. Esquema de comunicación de la interfaz electrónica con los dispositivos y la computadora del robot.

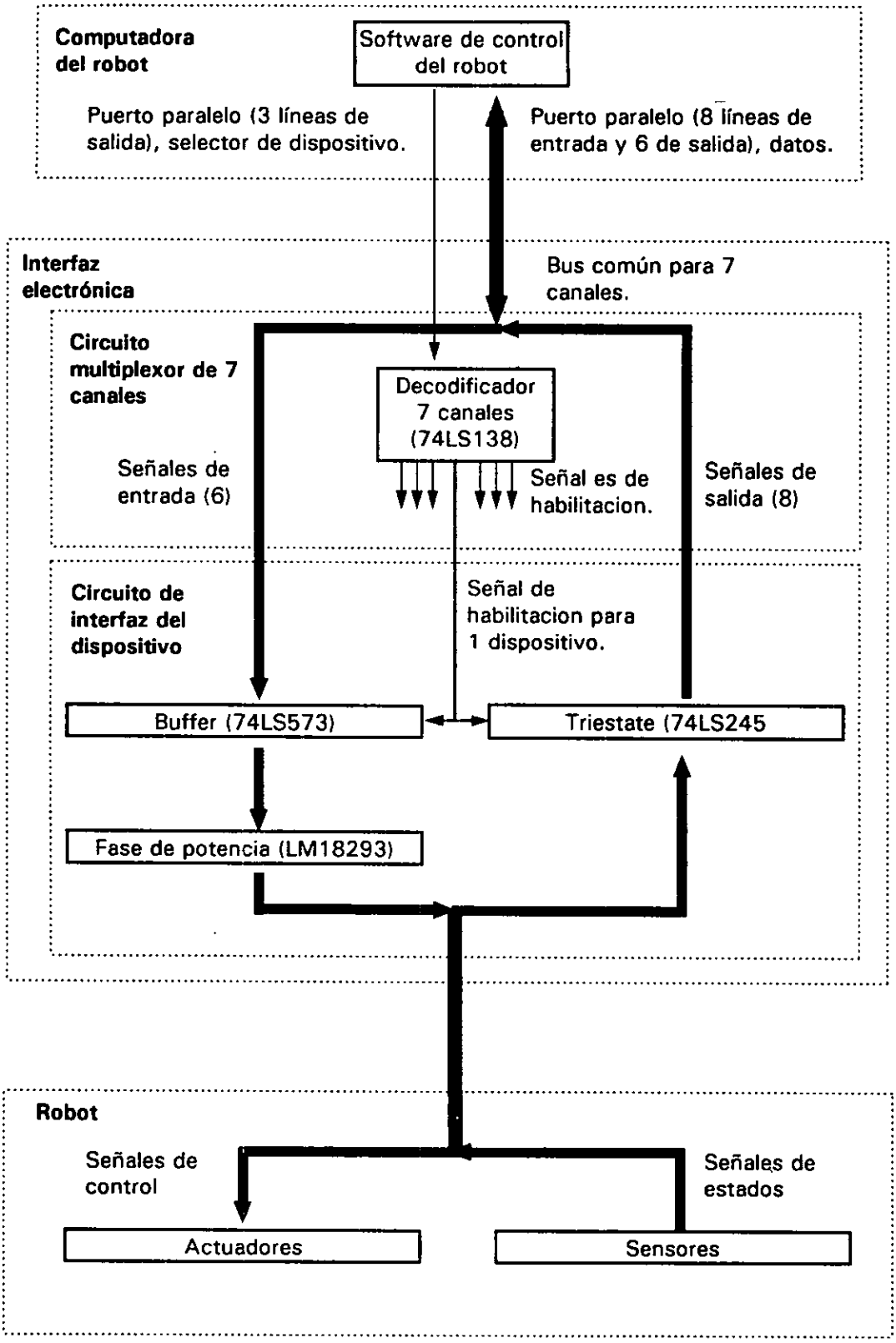


Ilustración 32. Arquitectura general de la interfaz electrónica.

Puertos de selección y datos

La comunicación de la computadora con la interfaz utiliza 3 conectores DB-25, 2 de ellos para los puertos paralelos de la PC y 1 para el circuito de la interfaz, en este último se agrupan las señales de datos y selección (ver Ilustración 31).

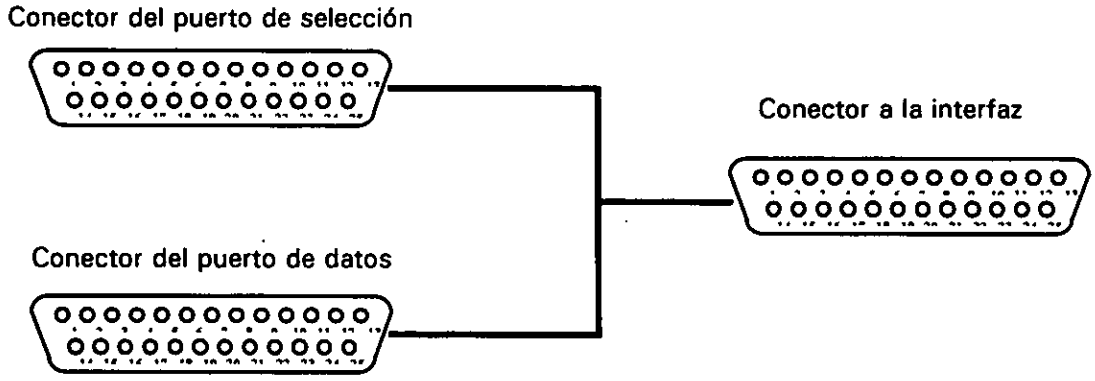


Ilustración 33. Conectores DB-25 de los puertos de la PC y de la interfaz electrónica .

Los puertos paralelos de las computadoras, comúnmente utilizados como puertos de impresión, cuentan con 8 bits de entrada 8 bits de salida. Los bits de entrada representan las señales de estado de la impresora, los bits de salida representan los caracteres de control e información a imprimir (ver Tabla 4). En los siguientes esquemas se muestra la disposición de las líneas de entrada y salida en un puerto paralelo de impresión.

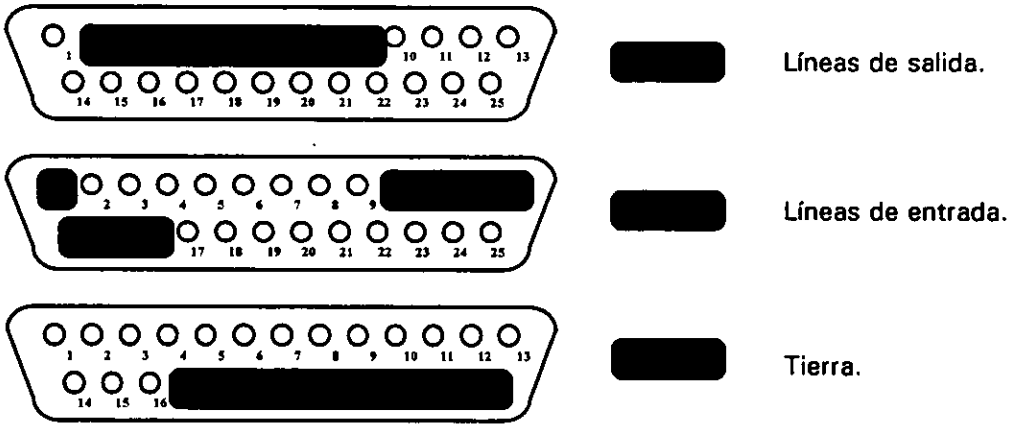


Ilustración 34. Disposición de pins en el puerto paralelo de impresión.

Las señales del puerto paralelo de impresión son reasignadas con el fin de formar el puerto de selección y el de datos de la interfaz. En la siguiente tabla se muestra el conjunto de señales y sus equivalentes para cada caso:

	Señales del puerto paralelo de impresión	Asignación de señales para el puerto de selección	Asignación de señales para el puerto de datos
	Strobe	X	Bit de entrada 0
	Bit de datos 0	Bit de selección 0	Bit de salida 0
	Bit de datos 1	Bit de selección 1	Bit de salida 1
	Bit de datos 2	Bit de selección 2	Bit de salida 2
	Bit de datos 3	X	Bit de salida 3
	Bit de datos 4	X	Bit de salida 4
	Bit de datos 5	X	Bit de salida 5
	Bit de datos 6	X	X
	Bit de datos 7	X	X
	Acknowledge	X	Bit de entrada 6
	Busy	X	Bit de entrada 7
	Paper End	X	Bit de entrada 5
	Select	X	Bit de entrada 4
	Auto Feed	X	Bit de entrada 1
	Error	X	Bit de entrada 3
	Initialize Printer	X	Bit de entrada 2
	Select Input	X	X
	Tierra	Tierra	Tierra

Tabla 4. Asignación de los pines en un puerto paralelo de impresión, la X indica que no se utiliza en la implementación.

La computadora representa los puertos como un conjunto de localidades de memoria de 8 bits denominados puertos lógicos. Cada uno tiene asignado 1 puerto lógico para los datos de salida y 2 para los datos de entrada, estos pueden ser accesados a través de una dirección de memoria.
Los 16 bits que forman los puertos lógicos de entrada no son utilizados completamente, y pueden ser ensamblados para formar un solo byte de información (los números dentro de las casillas corresponden a los pines del puerto):

11	10	12	13	15	0	0	0
7	6	5	4	3	2	1	0

Puerto lógico de entrada, dirección 0379Hex (5 bits útiles)

OR

0	0	0	0	0	16	14	1
7	6	5	4	3	2	1	0

Puerto lógico de entrada, dirección 037ahex (3 bits útiles)

11	10	12	13	15	16	14	1
7	6	5	4	3	2	1	0

Byte de entrada (Puerto de datos).

El byte del puerto de datos corresponde directamente al puerto lógico de salida, de este sólo son utilizados 6 de los 8 bits:

X	X	7	6	5	4	3	2
7	6	5	4	3	2	1	0

Puerto lógico de salida, dirección 0378Hex = Byte de salida (Puerto de datos).

Finalmente del puerto de selección se utilizan solo 3 bits de salida:

X	X	X	X	X	4	3	2
7	6	5	4	3	2	1	0

Puerto lógico de salida, dirección 0278Hex = Byte de salida (Puerto de selección).

Para el conector DB25 de la interfaz se agrupan las líneas de los puertos de datos y de selección como se muestra a continuación:

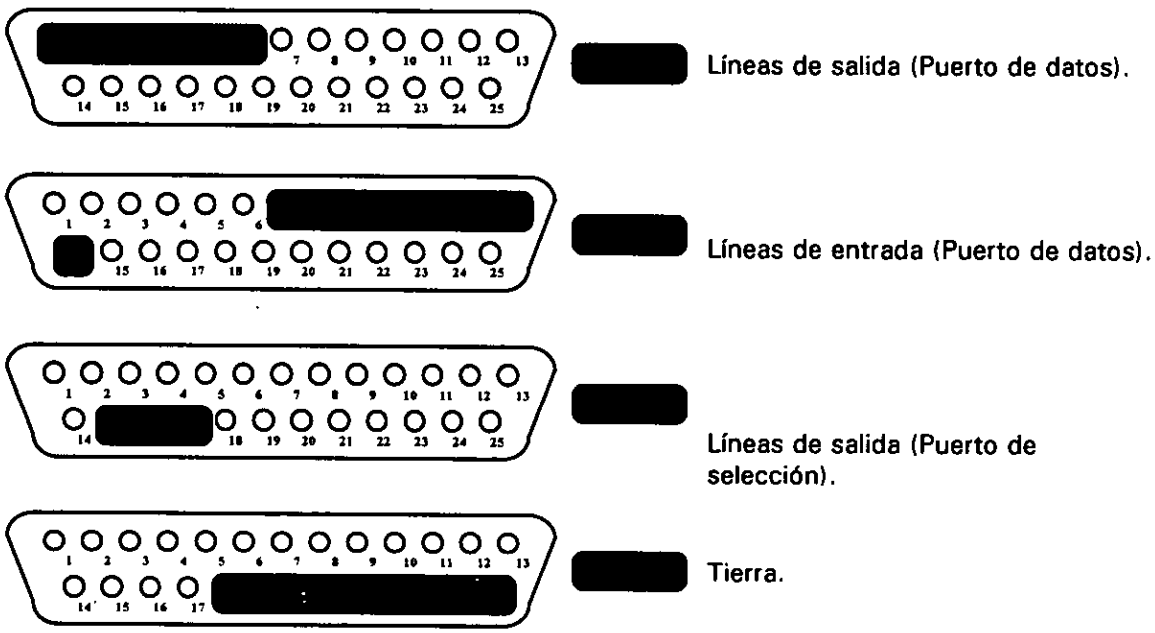


Ilustración 35. Disposición de pines en el conector DB-25 de la interfaz electrónica.

Circuito multiplexor

Este circuito está formado por un bus común con 8 líneas de salida, 6 de entrada, un par de líneas de habilitación para cada uno de los 7 circuitos de los dispositivos, y una serie de líneas de alimentación de 12 V para los actuadores. Cuenta además con un conector DB-25 para comunicarse con la computadora y un banco de conectores para los circuitos de interfaz de los dispositivos (ver Ilustración 41).

El direccionamiento de cada dispositivo se efectúa a través de las líneas del puerto de selección. Estas ultimas, se conectan al circuito decodificador 74LS138 el cual genera las señales de habilitación. En la Ilustración 36 se muestra un diagrama de tiempo con las señales de habilitación de cada dispositivo durante el proceso de direccionamiento.

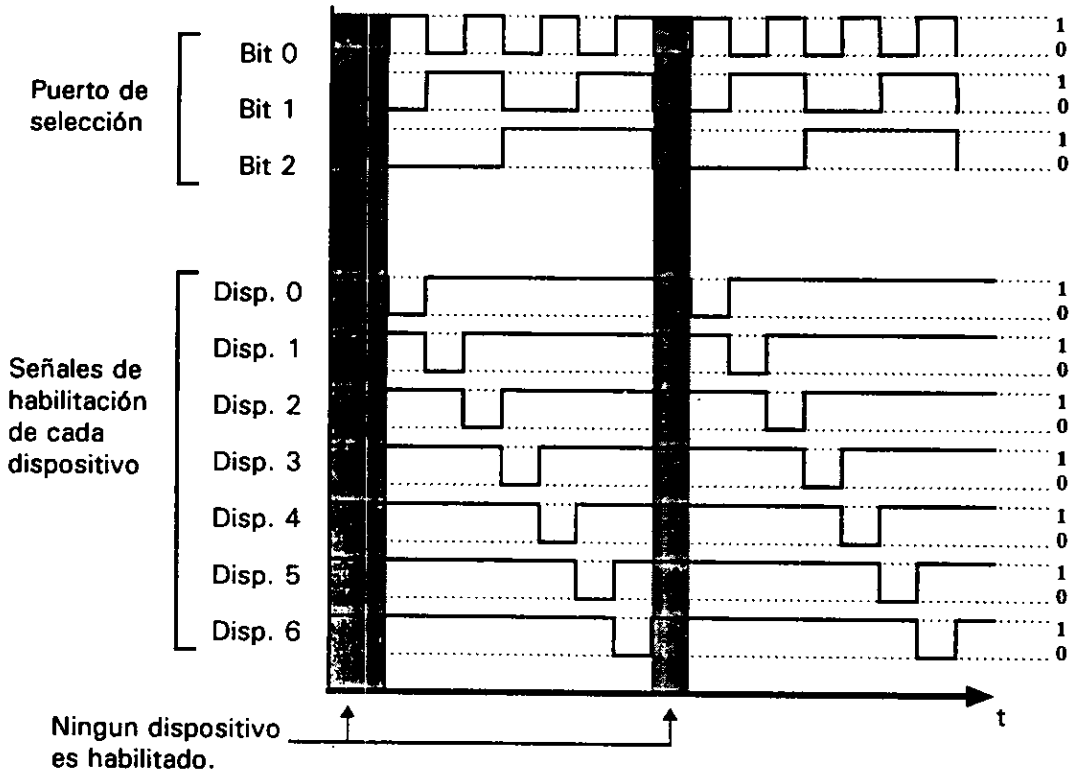


Ilustración 36. Diagrama de tiempos del circuito multiplexor.

En las ilustraciones de las siguientes páginas se muestran los esquemáticos y plantillas del circuito, así como la representación gráfica del mismo.

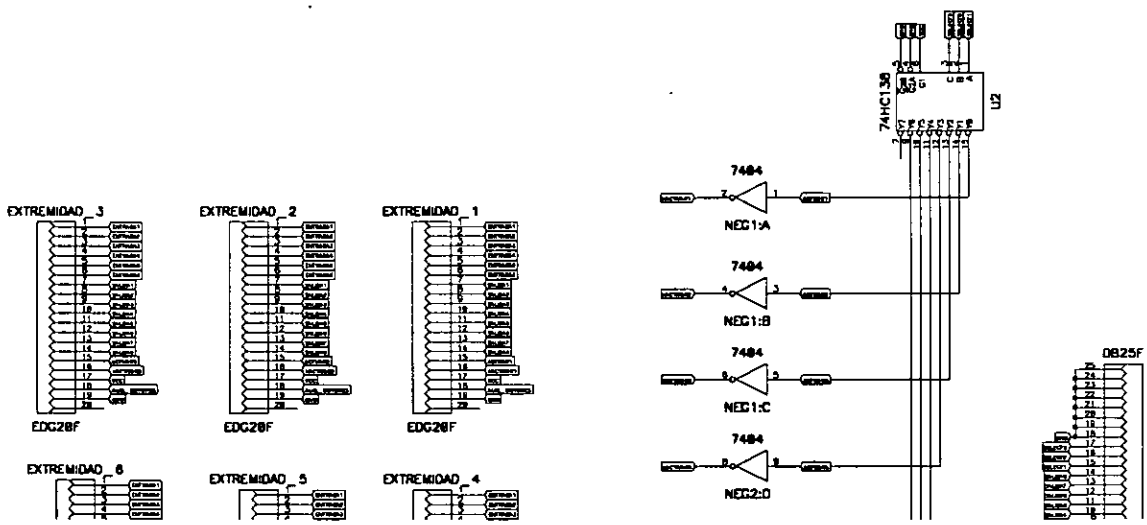


Ilustración 37. Esquemático del circuito multiplexor (diseñado en Tango Pro)

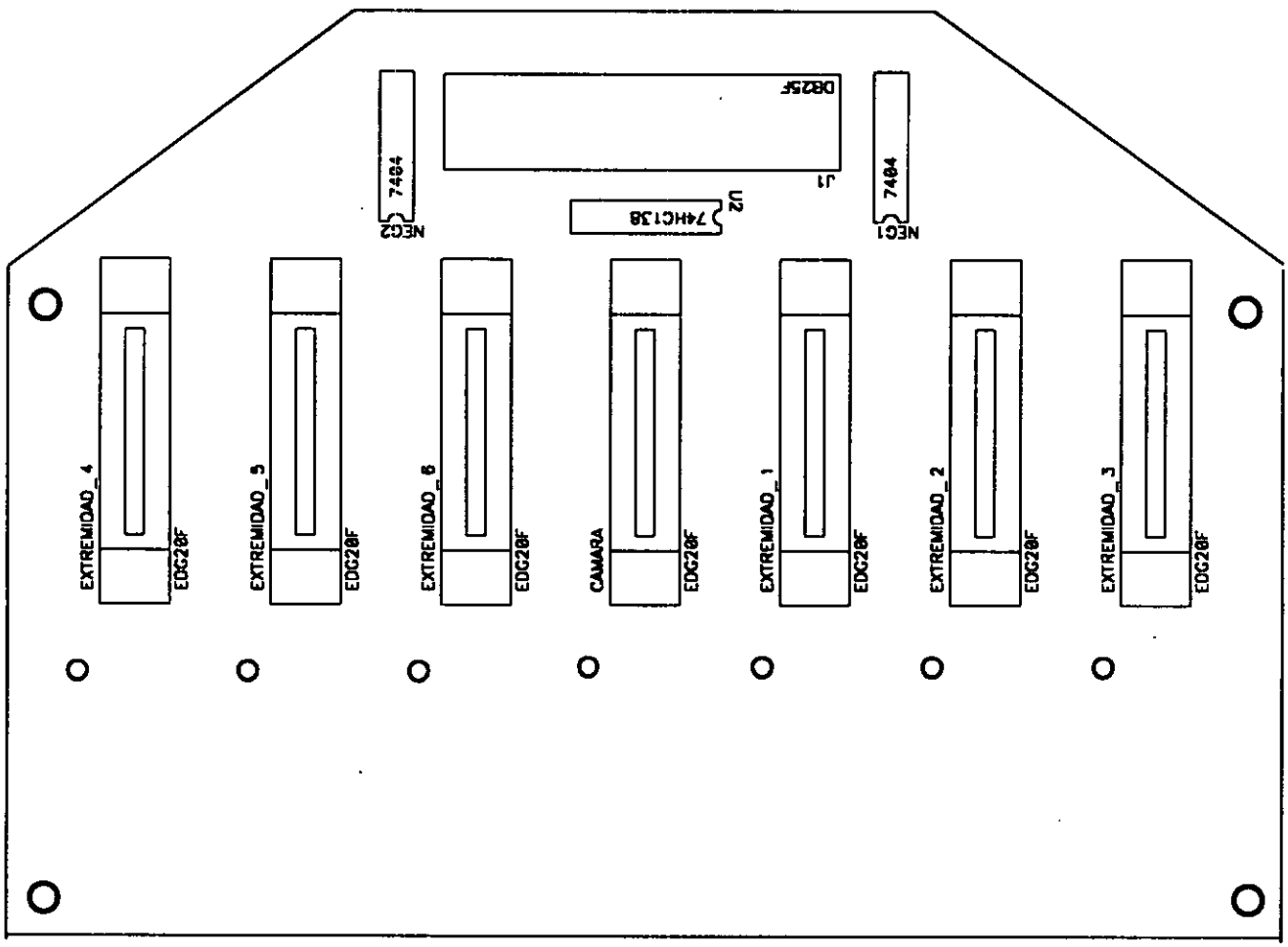


Ilustración 38. Plantilla de componentes del circuito multiplexor (diseñado en Tango Pro)

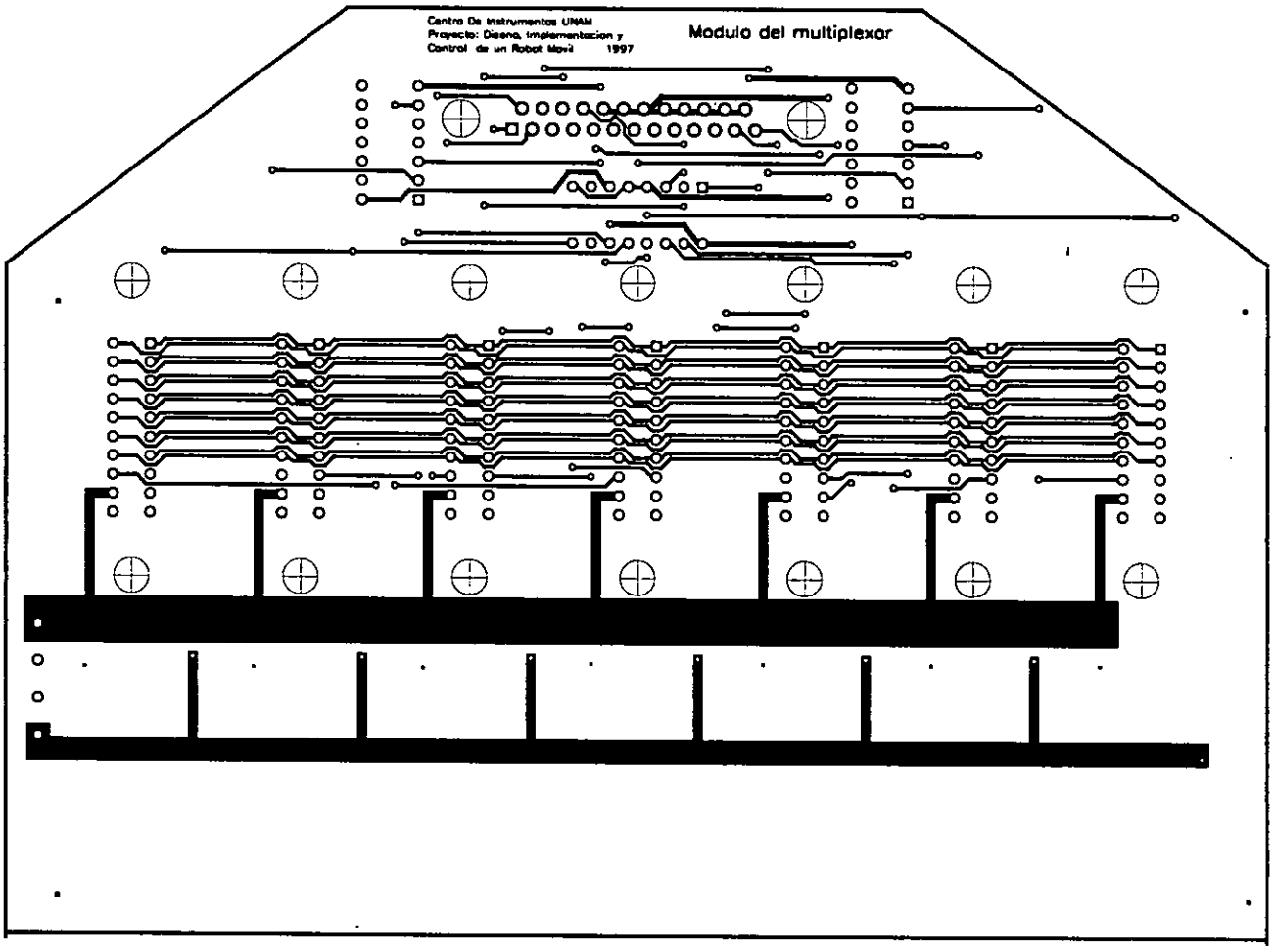


Ilustración 39. Plantilla superior del circuito multiplexor (diseñado en Tango Pro)

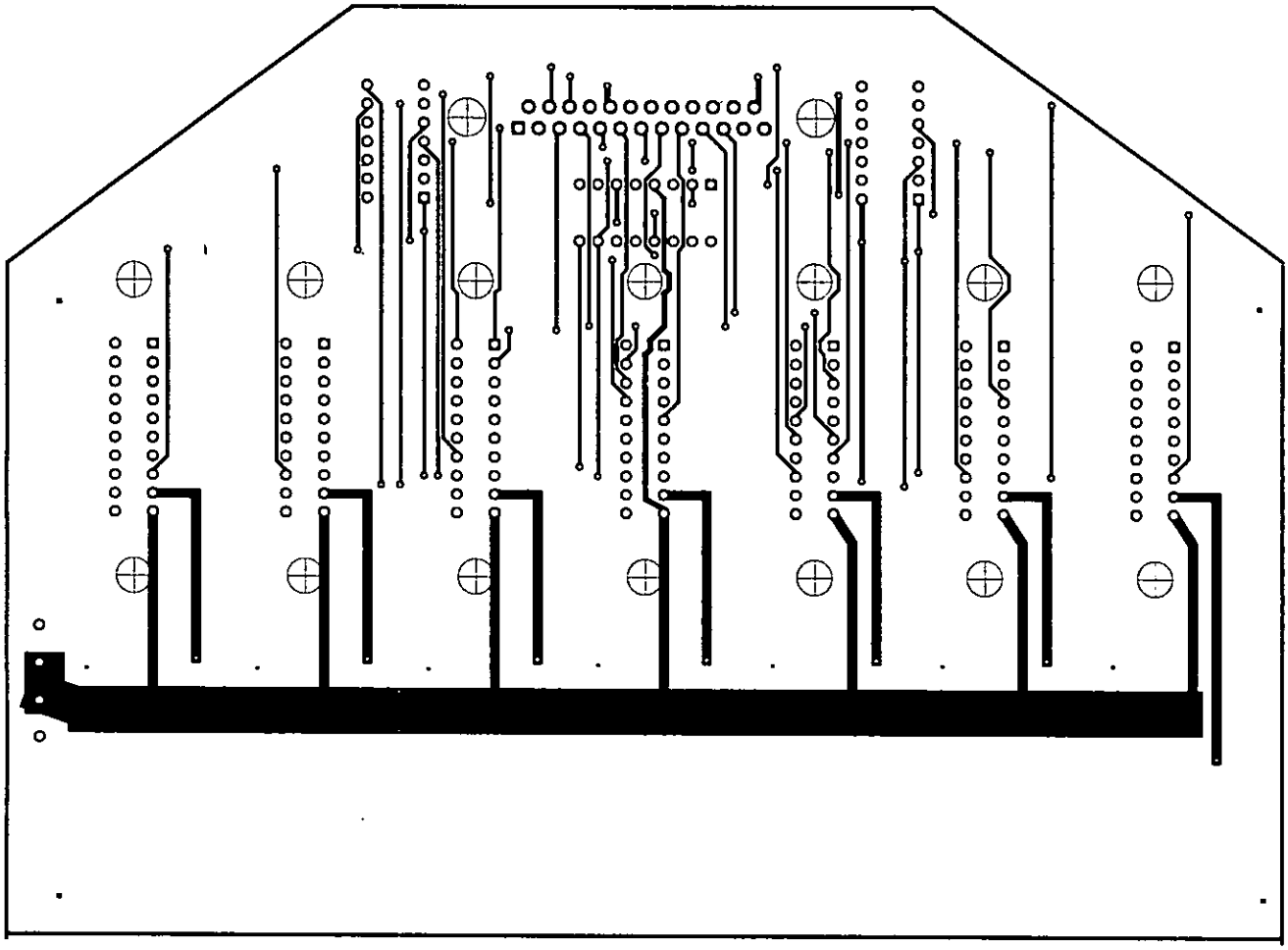


Ilustración 40. Plantilla inferior del circuito multiplexor (diseñado en Tango Pro)

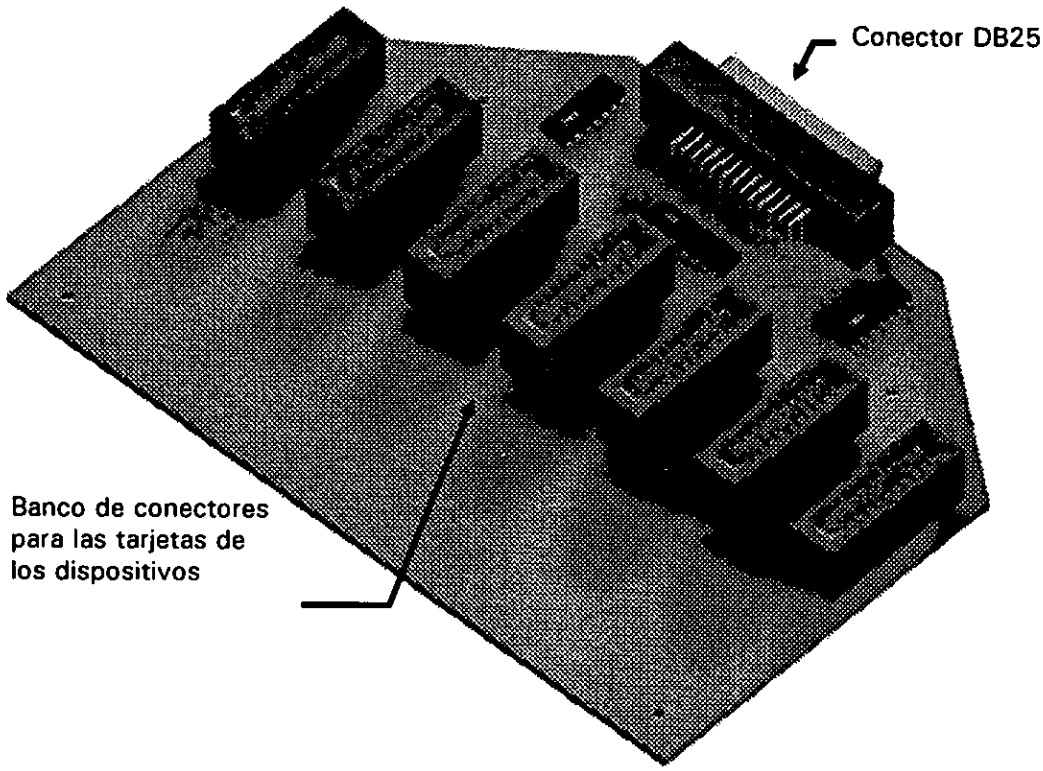


Ilustración 41. Vista del circuito multiplexor (diseñado en ACAD-TangoPro, visualización 3D Studio)

Circuito de interfaz del dispositivo

Este circuito funciona como la interfaz para un solo dispositivo, es un elemento independiente que interactúa con la computadora a través del circuito multiplexor (ver Ilustración 32). Al direccionar, la señal de habilitación permite que el latch (74LS573) tome los 6 bits de entrada residentes en el bus común (ver Ilustración 42), al mismo tiempo el circuito tres estados (74LS245) permite la salida de las 8 señales de los sensores a las líneas de salida del mismo bus (ver Ilustración 46).

Las 6 líneas de entrada contienen señales provenientes del puerto de datos de la computadora. Cada par de bits denominados motor#_P y motor#_N indican la dirección de rotación de uno de los actuadores (ver Tabla 5).

Bit	Módulo	Actuador	Descripción
01	Eslabón 1	motor 1 P	Movimiento del actuador del primer eslabón en sentido horario
02	Eslabón 1	motor 1 N	Movimiento del actuador del primer eslabón en sentido antihorario
03	Eslabón 2	motor 2 P	Movimiento del actuador del segundo eslabón en sentido horario
04	Eslabón 2	motor 2 N	Movimiento del actuador del segundo eslabón en sentido antihorario
05	Eslabón 3	motor 3 P	Movimiento del actuador del tercer eslabón en sentido horario
06	Eslabón 3	motor 3 N	Movimiento del actuador del tercer eslabón en sentido antihorario

Tabla 5. Byte de control para los actuadores de 1 dispositivo.

En el siguiente diagrama de tiempo se muestra la habilitación del circuito y carga de datos del bus común por medio del latch:

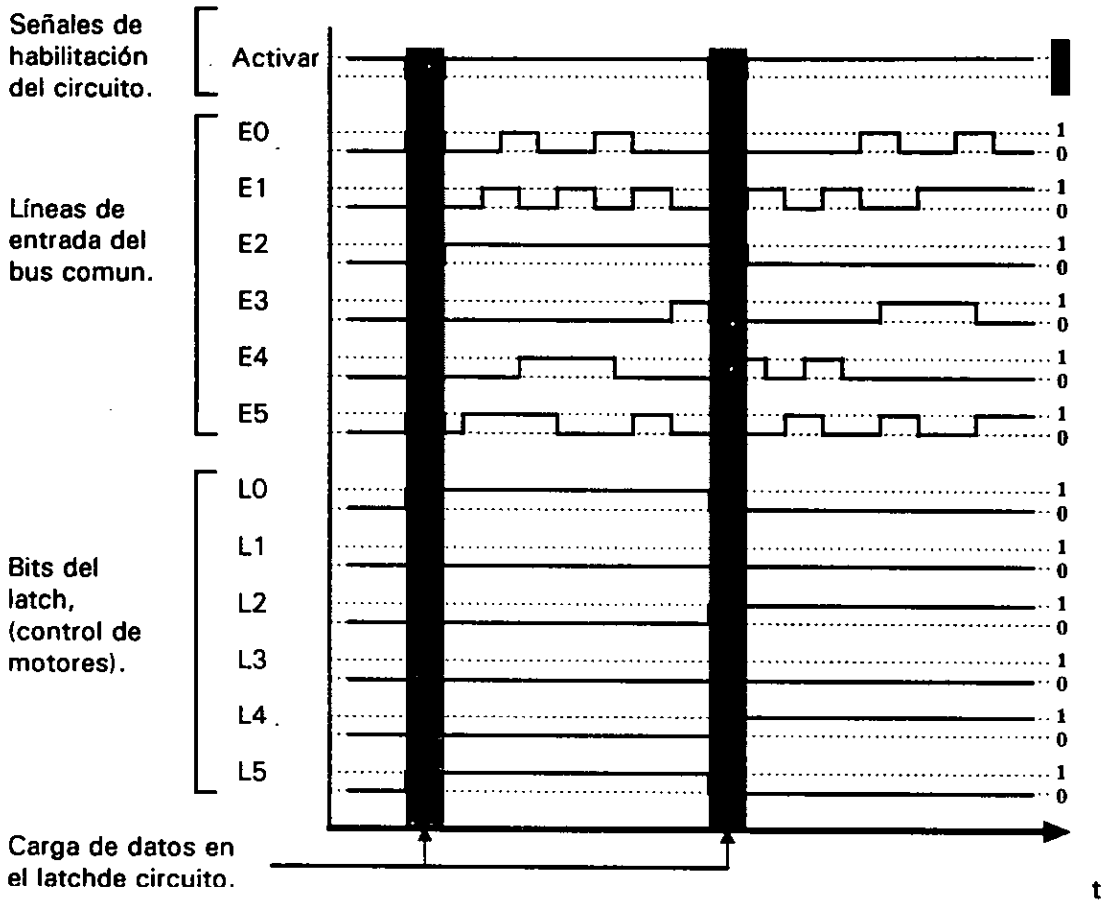


Ilustración 42. Diagrama de tiempo para la carga de datos.

Las señales de control de cada actuador pasan por una etapa de verificación de límites de rotación, de tal forma que si se verifica una de éstas se impide que el actuador continúe su desplazamiento (ver Ilustración 43).

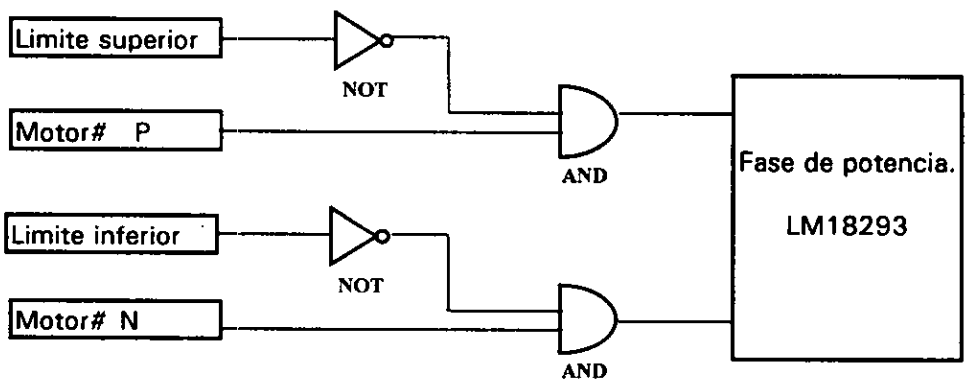


Ilustración 43. Verificación de límites de rotación.

La potencia necesaria para producir el movimiento de los motores se provee a través del circuito integrado LM18293. En forma general este integrado puede configurarse para obtener el equivalente a un par de circuitos "H" , con los cuales se controla el movimiento de 2 motores de DC en forma bidirecional, tal y como lo ilustra la siguiente figura:

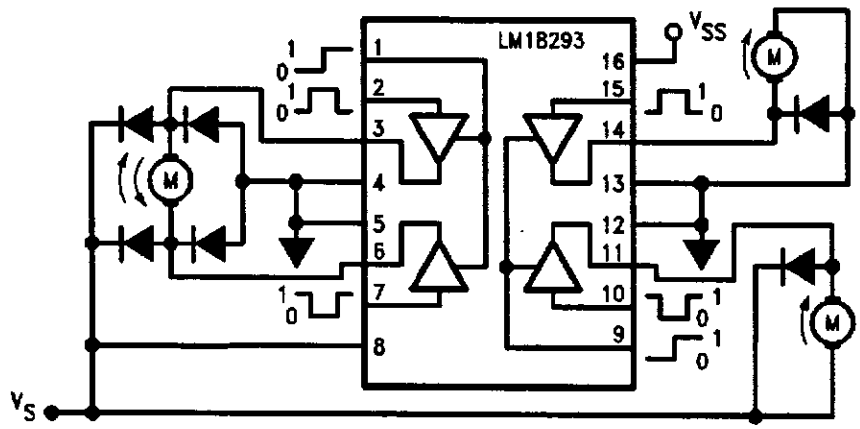


Ilustración 44. Configuración típica del LM18293 para el control bidirecional de motores de DC.

Por otro lado, las líneas de salida de la interfaz corresponden a las señales provenientes de los sensores optoelectrónicos, limites de rotación y sensores externos para detección de obstáculos. Las señales de los 6 sensores de límite son codificadas en 3 bits utilizando la función lógica OR como se indica en la Tabla 6. Dado que el software de control induce el sentido de rotación de los actuadores, no es necesario comunicarle el estado de ambos sensores de límite, solo la verificación de una u otra señal. Con esta información el controlador puede inferir la condición que la produjo (ver Tabla 7).

Señal de límite a la salida	Sensor de límite inferior	Señal de límite a la salida de la interfaz (OR)	Descripción
0	0	0	Rotación libre en ambos sentidos.
0	1	1	Rotación restringida en sentido antihorario.
1	0	1	Rotación restringida en sentido horario.
1	1	1	No válido (mutuamente excluyentes).

Tabla 6. Codificación de las señales de límite de rotación.

Para determinar si la señal corresponde al límite superior o inferior, el software de control toma en cuenta el sentido de rotación inducido en el actuador de acuerdo a la siguiente tabla:

Sentido de rotación del actuador	Señal de límite a la salida de la interfaz	Interpretación del software de control
horario	0	Rotación libre
horario	1	Límite superior
antihorario	0	Rotación libre
atihorario	1	Límite inferior

Tabla 7. Interpretación de las señales de límite a la salida de la interfaz.

Una vez reducido el número de líneas, la información a la salida de la interfaz del dispositivo se codifica en 1 byte como lo muestra la siguiente tabla:

Bit	Módulo	Sensor	Descripción
01	Eslabón 1	Contacto	Señal del límite Superior/inferior del primer eslabón
02	Eslabón 2	Contacto	Señal del límite Superior/inferior del segundo eslabón
03	Eslabón 3	Contacto	Señal del límite Superior/inferior del tercer eslabón
04	Eslabón 1	Contacto	Señal de contacto para detección de terreno
05	Eslabón 1	Optoelectrónico	Señal de cambio de posición
06	Eslabón 2	Optoelectrónico	Señal de cambio de posición
07	Eslabón 3	Optoelectrónico	Señal de cambio de posición
08	Eslabón 2	Contacto	Señal de contacto para detección de terreno

Tabla 8. Codificación de las señales de salida de la interfaz del dispositivo.

Al habilitar el circuito, los 8 bits se transmiten al bus común del multiplexor donde pueden ser accesados por el puerto de la computadora. Este proceso es descrito en el diagrama de tiempo de la Ilustración 45, en el se destaca cómo el bus común sólo transmite las señales de salida del dispositivo en turno. En las ilustraciones de las páginas subsecuentes se muestran los esquemáticos y plantillas del circuito, así como la representación gráfica del mismo.

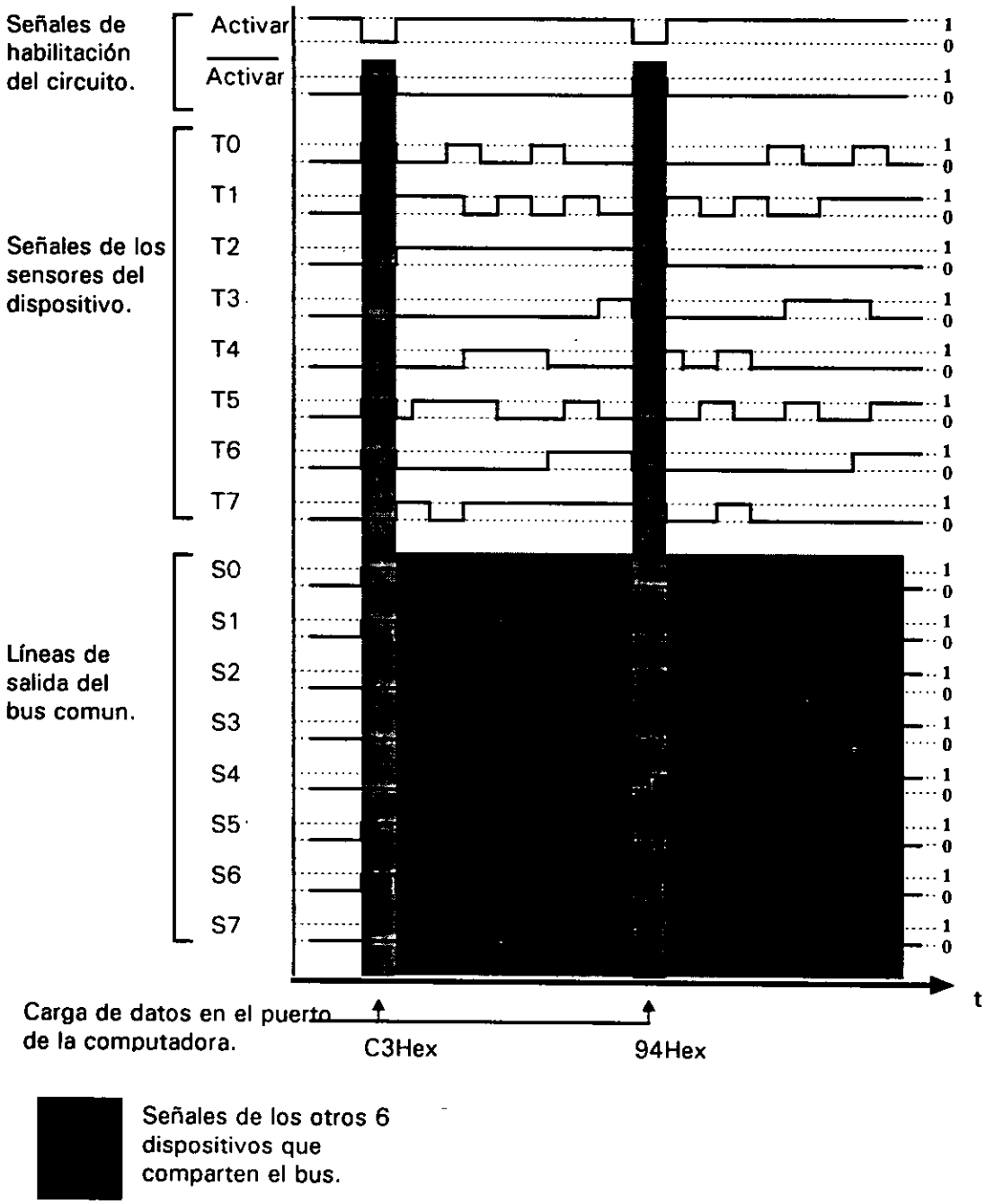


Ilustración 45. Diagrama de tiempo para la descarga de datos al puerto de entrada de la computadora.

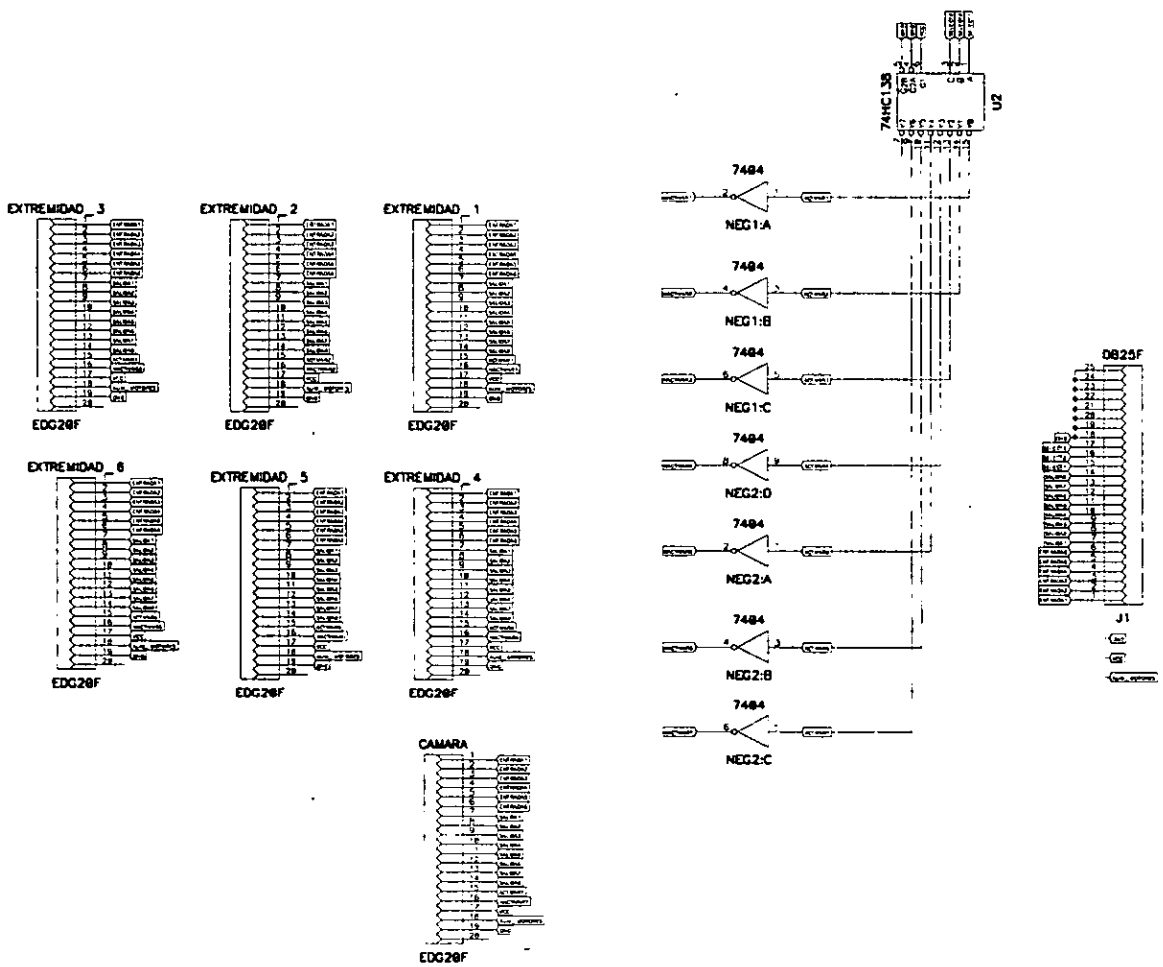


Ilustración 46. Esquemático del circuito de interfaz de dispositivo (Diseñado en Tango Pro).

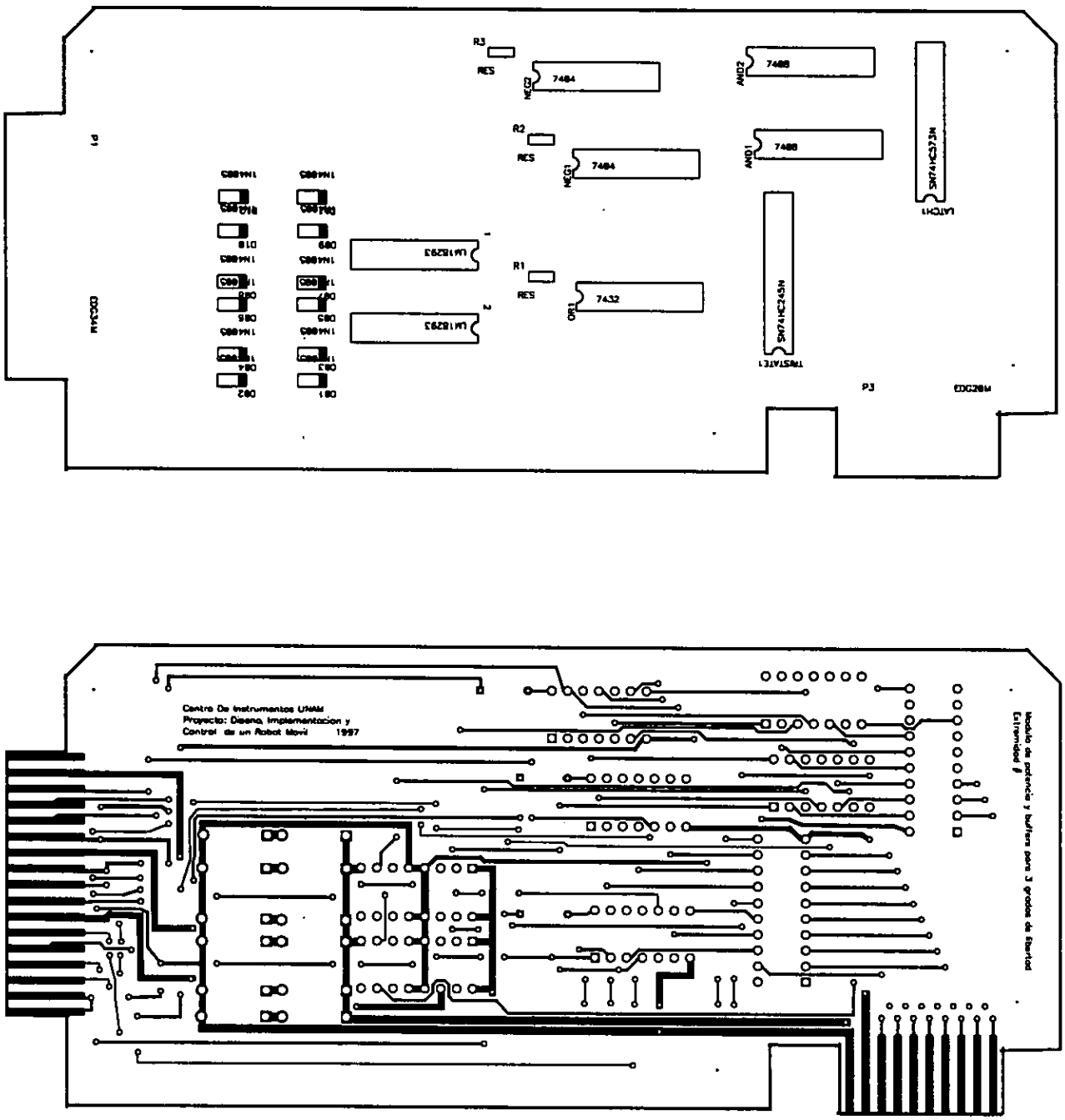


Ilustración 47. Plantilla de componentes y frontal del circuito de interfaz de dispositivo (Diseñado en Tango Pro).

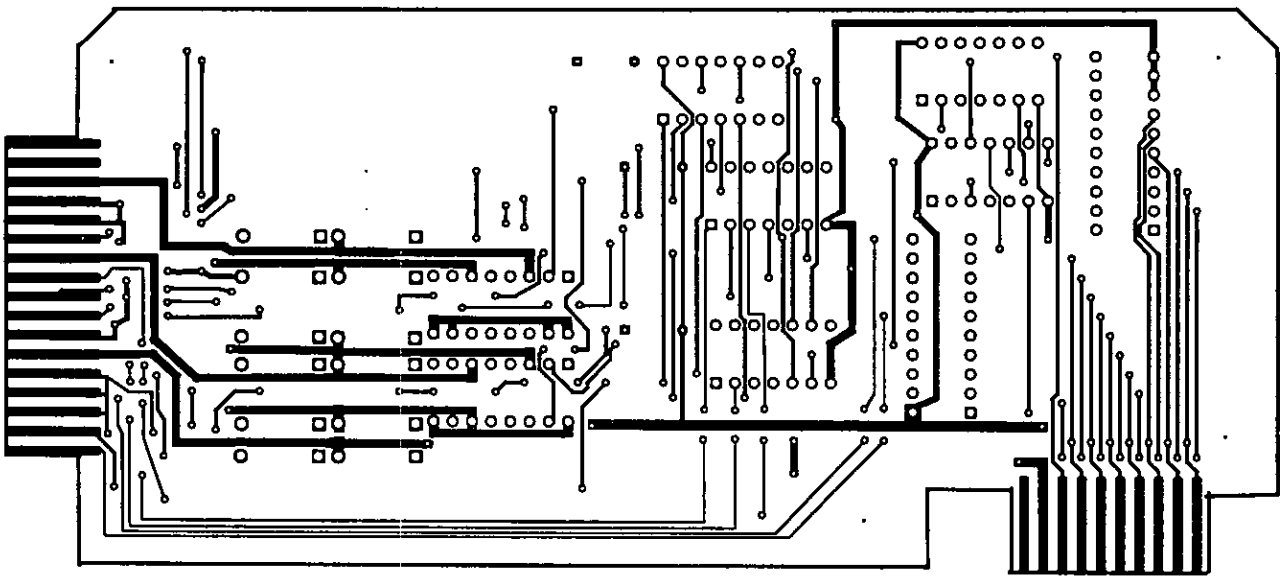


Ilustración 48. Plantilla inferior del circuito de interfaz de dispositivo (Diseñado en Tango Pro).

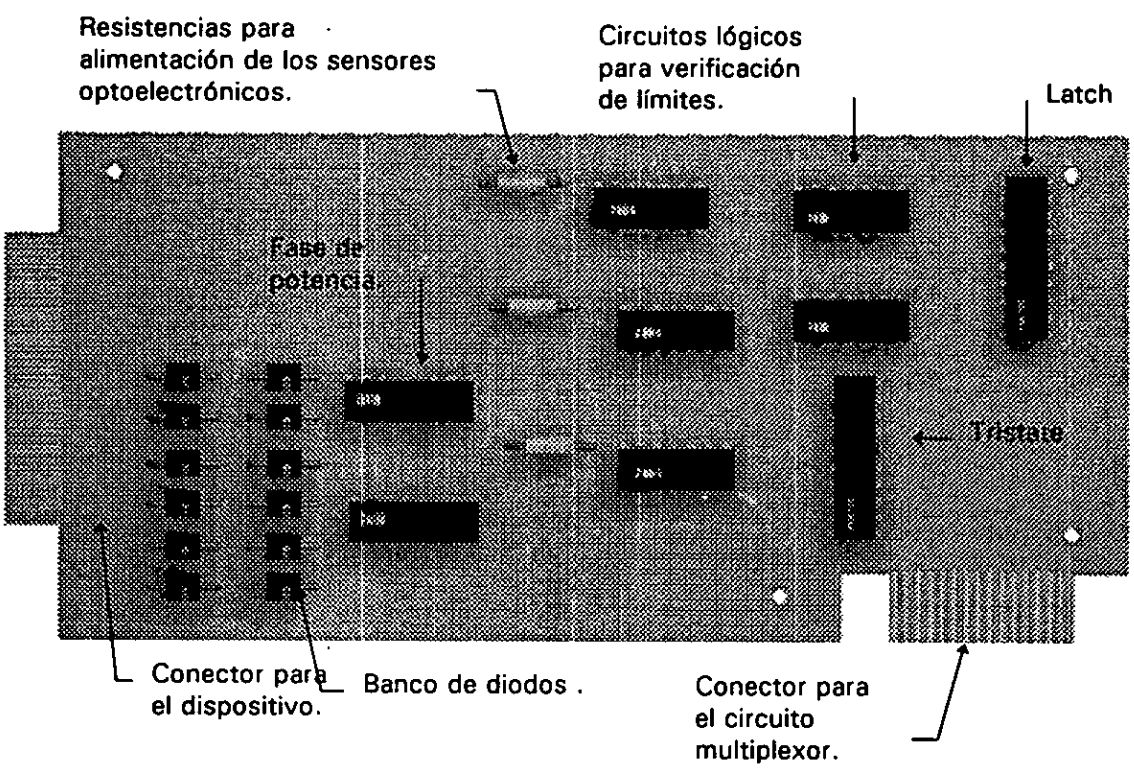


Ilustración 49. Vista del circuito de interfaz de dispositivo (Visualización en 3D Studio).

En la siguiente ilustración se muestra el ensamblado del circuito multiplexor con uno de los circuitos de interfaz de dispositivo.

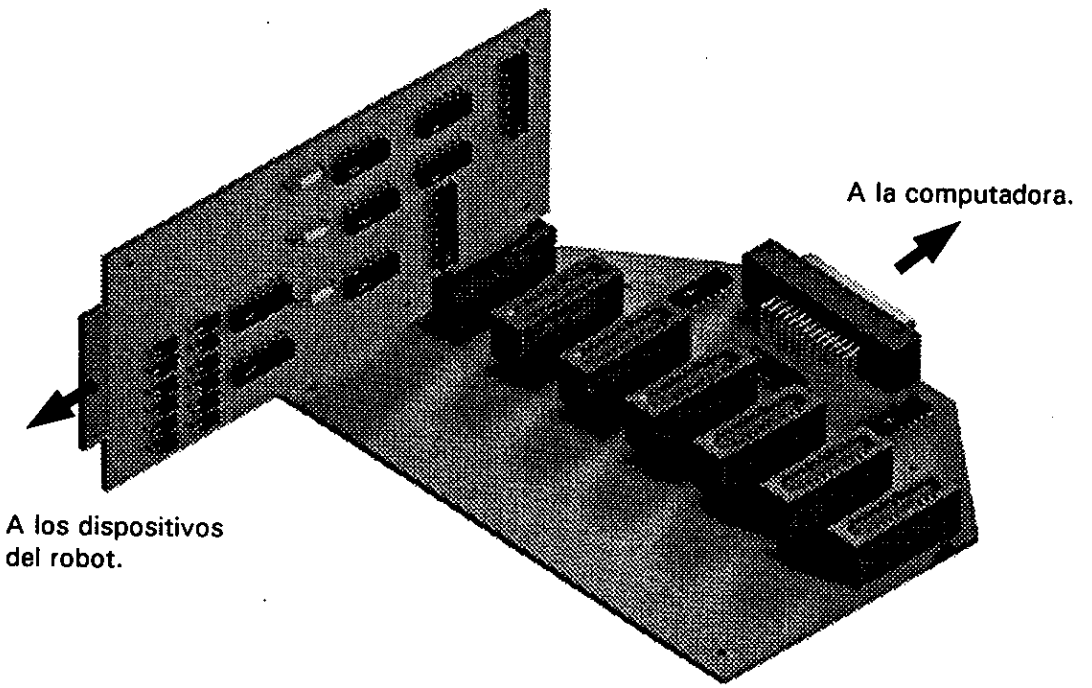


Ilustración 50. Vista del circuito multiplexor y uno de los circuitos de interfaz de dispositivo (Visualización en 3D Studio)

Capítulo 5

Desarrollo del sistema de simulación y control

En el presente capítulo se describen el análisis, diseño e implementación del sistema de simulación y control. Por medio de este software es posible simular el comportamiento del robot al desplazarse por un terreno de topología irregular, controlar cada una de sus acciones y visualizar los resultados.

Con el fin con de dar un mejor seguimiento a cada una de las fases del desarrollo, el capítulo se ha dividido en las siguientes secciones:

- **Especificación de requerimientos.**
- **Bosquejo de la aplicación.**
- **Análisis de las clases provistas por Liquid Reality.**
- **Identificación de clases del sistema.**
- **Definición de atributos y servicios.**
- **Colaboración entre objetos.**
- **Creación de un prototipo.**
- **Extensión del prototipo.**
- **Pruebas y resultados.**

Especificación de requerimientos

El objetivo de la aplicación es simular el desplazamiento del robot móvil descrito en la propuesta del capítulo uno, a través de un terreno de topología irregular. Los resultados obtenidos en la simulaciones deben mostrase al usuario en forma de representaciones tridimensionales, de forma que sea fácil para él hacer evaluaciones cualitativas. Por otro lado la interfaz del sistema ha de permitir al usuario manipular las acciones del robot y algunos otros parámetros durante la simulación.

El simulador hará uso del modelo geométrico del robot para resolver problemas de cálculo de posición de cada extremidad , así como la coordinación de todas ellas a fin lograr el desplazamiento sobre las irregularidades del terreno.

El sistema debe implementar el acceso a recursos de cómputo externos utilizando una arquitectura Cliente/Servidor, de forma que el proceso de simulación pueda ser distribuido en múltiples computadoras. Esta especificación permitirá la incorporación de cálculos de mayor complejidad al simulador y en general dejar abierta la posibilidad de extender el sistema para solventar nuevos requerimientos en el futuro.

El desarrollo de este sistema debe representar una base para la implementación del controlador del robot real, es decir, el diseño de las clases que representan al robot en el simulador serán reutilizadas en el programa de control residente en la computadora del robot.

Finalmente, el sistema deberá contar con un diseño modular, que permita hacer modificaciones de forma sencilla, ya sea con el fin de incrementar su funcionalidad o bien para corregir y depurar el código.

Bosquejo de la aplicación

Esta fase complementa la especificación de requerimientos, definiendo cómo va a interactuar el sistema con el usuario. A manera de un bosquejo se describe en forma general la interfaz del software, indicando las funciones de sus componentes gráficos, la forma de presentar resultados y como recibir comandos. El diseño de esta interfaz auxilia al analista a tener una mejor idea de la funcionalidad que deberá implementar el sistema.

La interfaz con el usuario está constituida por 3 elementos principales:

1. Tableros de control.
Permiten manipular algunos parámetros de simulación, ya sea para modificar el comportamiento del robot, o bien el medio que lo rodea. El visualizador de Liquid Reality provee una serie de controles con los cuales modificar su ambiente gráfico: calidad de representación tridimensional, modo de visualización, iluminación, etc., estos se muestran en la Ilustración 51. Para el control del robot se utiliza un tablero más, en el cual se muestra de forma sencilla e intuitiva las funciones que éste puede realizar, como: desplazamiento hacia adelante, incremento o decremento de altura y la rotación sobre el eje vertical. El prototipo del tablero cuenta con una serie de botones que activan las funciones de rotación y desplazamiento, su forma a manera de vectores indican la acción a desarrollar (ver Ilustración 52). Para el caso del control de la altura se cuenta con un slider con el cual se puede elevar el robot y variar su distancia con el suelo en un rango de 0 a 1 unidades.

El dispositivo de orientación de la cámara se maneja como un dispositivo independiente y su posición angular puede especificarse en forma manual a través de un control esférico sensible al arrastre del puntero del mouse o bien a través de un control automático formado por botones. El resultado del posicionamiento se representa utilizando una luz direccional sobre el robot.

VRML provee un recurso para especificar el punto de vista del usuario dentro del escenario, este es denominado ViewPoint. El tablero incluye una serie de botones que permiten intercambiar el ViewPoint de entre 10 posiciones preestablecidas alrededor del robot, de forma que el usuario pueda apreciar l a simulación desde diferentes puntos de vista.

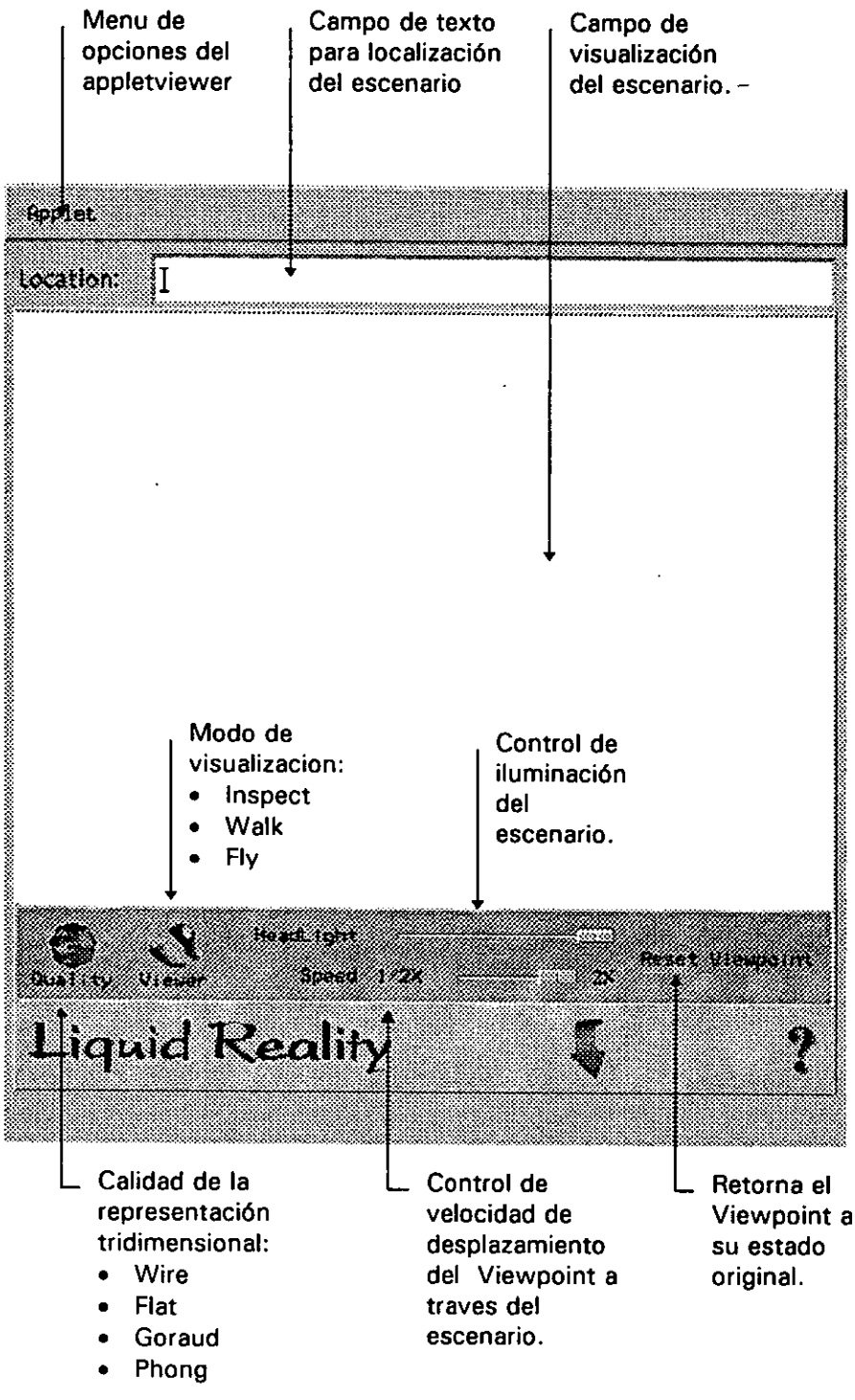


Ilustración 51. Ventana del visualizador de Liquid Reality.

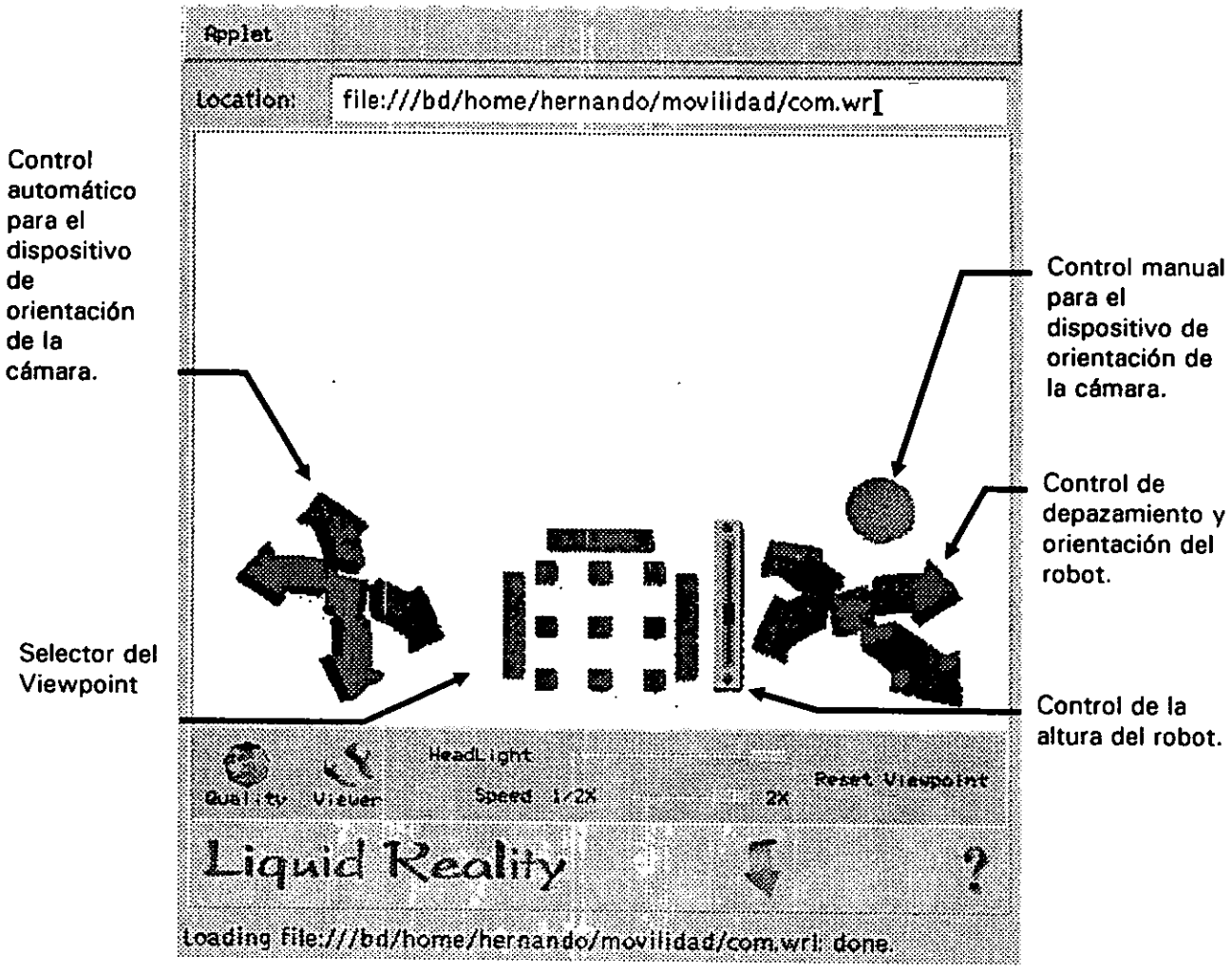


Ilustración 52. Vista del tablero de control y su disposición para el operador.

1. Ambiente de simulación.

Formado por la representación del terreno sobre el cual se desplaza el robot. La descripción de su geometría utiliza una malla de 50X50 puntos y su topología se genera evaluando los puntos de una función f(x,z) (ver Ilustración 53). Esta misma función se utiliza para determinar los puntos de contacto entre el terreno y las extremidades durante el proceso de desplazamiento. La magnitud de la componente en Y (altura) de los puntos obtenidos a través de la cinemática inversa de cada extremidad, es comparada con el valor de f(x,z) correspondiente a la altura del terreno para ese mismo punto.

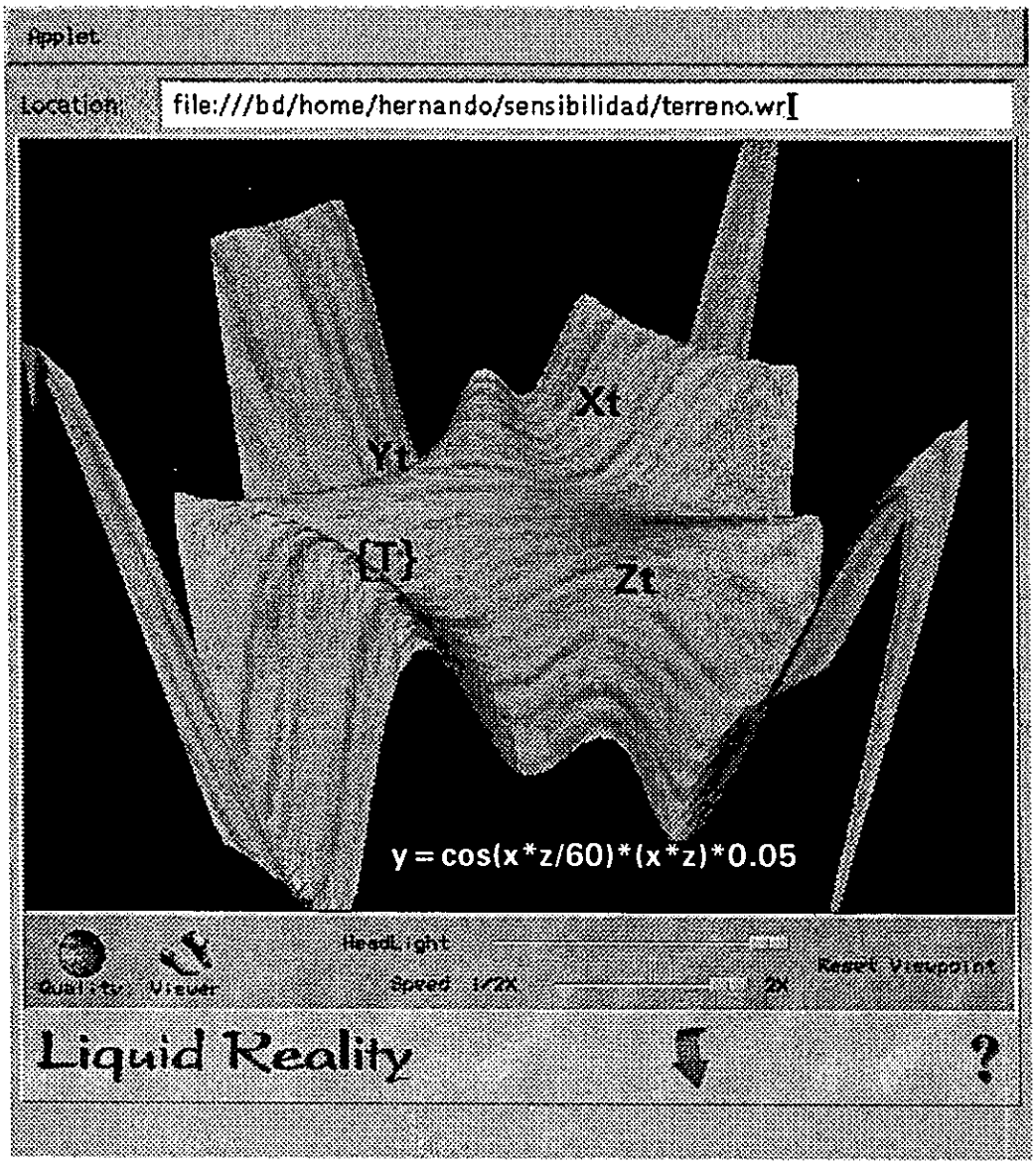


*Ilustración 53. Vista de la representación del terreno generado para las pruebas de desplazamiento, la función empleada es: y=cos(x*z/60)*(x*z)*0.05 .*

1. Representación gráfica del robot.

Muestra al usuario la estructura de eslabones que componen al robot, así como la posición de cada uno de ellos durante la simulación. La representación del robot se obtiene a partir de los planos generados en el diseño de sus eslabones, descrito en el tema "Descripción Mecánica" del Capítulo 2. Estos planos se trasladan de su formato original de AutoCad (*.dxf) al formato de VRML (*.wrl), de forma que los archivos resultantes contienen la descripción geométrica del robot (ver Ilustración 54). La representación gráfica así obtenida se apega al aspecto del modelo original y al del robot real.

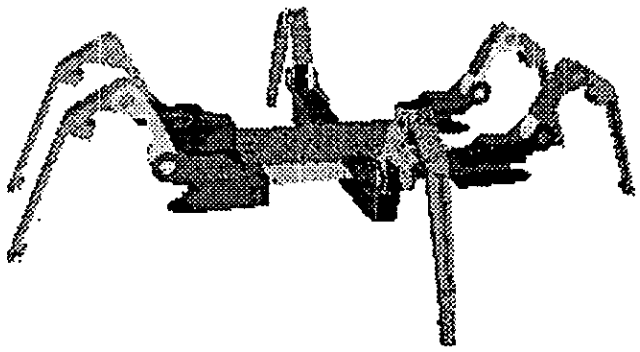


Ilustración 54. Representación gráfica del robot.

Identificación de clases y métodos provistos por el visualizador

Esta fase del desarrollo implica la identificación y estudio de los recursos que provee la implementación de Liquid Reality que sean de utilidad en el diseño de la aplicación. Las clases que a continuación se describen forman parte de la distribución Beta 11 y permiten el mapeo de las entidades en VRML a objetos en Java:

- ***vrml.Browser***
- ***vrml.Field***
- ***vrml.Node***
- ***vrml.Script***

Liquid Reality incluye además el manejo de excepciones para sus objetos a través de las siguientes clases:

- ***vrml.InvalidEventInException***
- ***vrml.InvalidEventOutException***
- ***vrml.InvalidExposedFieldException***
- ***vrml.InvalidFieldChangeException***
- ***vrml.InvalidFieldException***
- ***vrml.InvalidRouteException***
- ***vrml.InvalidVRMLSyntaxException***

En las siguientes secciones se describen brevemente algunas de estas clases y sus respectivos métodos.

Jerarquía de clases de Liquid Reality

```
java.lang.Object
    |
    +- vrml.Event
    +- vrml.Browser
    +- vrml.Field
    |       +- vrml.field.SFBool
    |       +- vrml.field.SFColor
    |       +- vrml.field.SFFloat
    |       +- vrml.field.SFImage
    |       +- vrml.field.SFInt32
    |       +- vrml.field.SFNode
    |       +- vrml.field.SFRotation
    |       +- vrml.field.SFString
    |       +- vrml.field.SFTime
    |       +- vrml.field.SFVec2f
    |       +- vrml.field.SFVec3f
    |       |
    |       +- vrml.MField
    |       |       +- vrml.field.MFColor
    |       |       +- vrml.field.MFFloat
    |       |       +- vrml.field.MFInt32
    |       |       +- vrml.field.MFNode
    |       |       +- vrml.field.MFRotation
    |       |       +- vrml.field.MFString
    |       |       +- vrml.field.MFTime
    |       |       +- vrml.field.MFVec2f
    |       |       +- vrml.field.MFVec3f
    |       |
    |       +- vrml.ConstField
    |               +- vrml.field.ConstSFBool
    |               +- vrml.field.ConstSFColor
    |               +- vrml.field.ConstSFFloat
    |               +- vrml.field.ConstSFImage
    |               +- vrml.field.ConstSFInt32
    |               +- vrml.field.ConstSFNode
    |               +- vrml.field.ConstSFRotation
    |               +- vrml.field.ConstSFString
    |               +- vrml.field.ConstSFTime
    |               +- vrml.field.ConstSFVec2f
    |               +- vrml.field.ConstSFVec3f
    |               +- vrml.ConstMField
    |                       +- vrml.field.ConstMFColor
    |                       +- vrml.field.ConstMFFloat
    |                       +- vrml.field.ConstMFInt32
    |                       +- vrml.field.ConstMFNode
    |                       +- vrml.field.ConstMFRotation
    |                       +- vrml.field.ConstMFString
    |                       +- vrml.field.ConstMFTime
    |                       +- vrml.field.ConstMFVec2f
    |                       +- vrml.field.ConstMFVec3f
    |
    +- vrml.BaseNode
            +- vrml.node.Node
            +- vrml.node.Script
```

```
java.lang.Exception
    |
    +- java.lang.RuntimeException
    |       +- java.lang.IllegalArgumentException
    |               +- vrml.InvalidEventInException
    |               +- vrml.InvalidEventOutException
    |               +- vrml.InvalidExposedFieldException
    |               +- vrml.InvalidFieldChangeException
    |               +- vrml.InvalidFieldException
    |               +- vrml.InvalidRouteException
    |
    +- vrml.InvalidVRMLSyntaxException
```

vrml.Field

Esta clase permite el mapeo y manipulación de tipos de datos definidos en VRML a objetos en Java. Cada tipo de datos está representado por una subclase de ***Field*** (consultar jerarquía de clases) :***SFBool, SFColor, MFColor, SFFloat, MFFloat, SFImage, SFInt32, MFInt32, SFNode, MFNode, SFRotation, MFRotatio, SFString, MFString, SFVec2f, MFVec2f, SFVec3f, MFVec3f, SFTime, MFTime***. Estas subclases implementan métodos de lectura y escritura, sin embargo, existen entidades en VRML2.0, como son los eventos de entrada, que requieren ser mapeados a objetos del tipo "solo lectura" o constantes. Las subclases de ***ConstField*** resuelven este requerimiento, implementando en sus instancias sólo métodos de lectura. La jerarquía de clases es similar a la derivada de ***Field***: ***ConstSFBool, ConstSFColor, ConstMFColor, ConstSFFloat, ConstMFFloat, ConstSFImage, ConstSFInt32, ConstMFInt32, ConstSFNode, ConstMFNode, ConstSFRotation, ConstMFRotation, ConstSFString, ConstMFString, ConstSFVec2f, ConstMFVec2f, ConstSFVec3f, ConstMFVec3f, ConstSFTime, ConstMFTime.***

Las preposiciones SF (Single Field) y MF (Multiple Field) indican si la clase implementa objetos simples o compuestos (uno o más objetos simples). La terminación 2f y 3f implica objetos del tipo vectorial, para (x,y) o (x,y,z) respectivamente.

Las subclases de ***Field*** implementan un conjunto de métodos para el acceso y manipulación de sus objetos: ***getSize(), getValue(), get1Value(), setValue(), set1Value(), addValue(), insertValue(), clear(), delete()*** y ***toString()***. Donde ***getSize(), get1Value(), set1Value(), addValue(), insertValue(), clear() y delete()*** solo están disponibles para las clases MF. A continuación se hace una breve descripción de estos métodos:

getSize()

Retorna el número de elementos de un campo MF.

getValue()

Retorna un objeto Java equivalente del objeto VRML receptor.

get1Value(int index)

Retorna un objeto equivalente en Java del tipo VRML apuntado por ***index***.

setValue(value)

Convierte el objeto de Java***value*** a un tipo VRML y hace una copia de el en el objeto receptor.

set1Value(int index, value)

Convierte el objeto de Java***value*** a un tipo VRML y hace una copia de el en el elemento del objeto receptor apuntado por ***index***.

addValue(value)

Convierte el objeto Java***value*** a un tipo VRML y lo inserta como un elemento del objeto receptor.

insertValue(int index, value)

Convierte el objeto Java***value*** a un tipo VRML y lo inserta como un elemento del objeto receptor en la posición indicada por ***index***.

clear()

Borra los elementos del objeto receptor.

delete(int index)

Borra el elemento apuntado por ***index*** del objeto receptor.

toString()

Retorna el valor (o valores) del objeto receptor en forma de una cadena codificada en utf8.

vrml.Node

Las clases de este paquete implementan la representacion de los nodos de VRML. Los métodos definidos para su acceso y manipulación son los que se describen a continuación:

String getType()

Retorna el tipo de nodo al cual pertenece el objeto receptor.

ConstField getEventOut(String eventOutName)

Retorna una referencia al evento de salida ***eventOutName*** del objeto receptor, el tipo de la referencia debe convertirse al adecuado ya que este método retorna un tipo genérico ***Field***.

Field getEventIn(String eventInName)

Retorna una referencia al evento de entrada ***eventInName*** del objeto receptor, el tipo de la referencia debe convertirse al adecuado ya que este método retorna un tipo genérico ***ConstField***.

Field getExposedField(String exposedFieldName)

Retorna una referencia al campo ***exposedFieldName*** del objeto receptor, el tipo de la referencia debe convertirse al adecuado ya que este método retorna un tipo genérico ***Field***.

Browser getBrowser()

Retorna una referencia a un objeto ***Browser*** donde reside el objeto receptor.

String toString()

Convierte al objeto receptor en una cadena codificada en utf8.

vrml.Script

Esta clase implementa los métodos de acceso a los nodos Script de VRML, los métodos de instancia más importates son citados a continuación:

public void initialize()

Este método es invocado automáticamente por el visualizador antes de que se produzca cualquier evento. Por lo regular es utilizado para inicializar variables u otros procesos.

public void processEvents(int count, Event events[])

Es invocado automáticamente al recibir el nodo ***Script*** un conjunto de eventos.

public void processEvent(Event event);

Es llamado automáticamente al recibir el nodo ***Script*** un evento.

public void eventsProcessed()

Es llamado automáticamente después de haber sido invocado el método ***processEvents()***.

public void shutdown()

Es invocado cuando el nodo ***Script*** es borrado.

Con los métodos de instancia definidos en las clases ***Field***, ***Script*** y ***Node*** es posible manipular los objetos descritos en un escenario VRML. El siguiente ejemplo muestra la forma de interacción entre Java y VRML:

En el archivo de VRML se define un nodo Script especificando los eventos y campos para el acceso a otras entidades.

```
Script VRML2.0

DEF manejadorExtremidad Script {
                    url "ManejadorExtremidad.class"

                    eventIn      SFVec3f traslacionRobotIn  IS      rotXrobotIn

                    eventOut     SFVec3f      xyzExtremoOut

                    field        SFVec3f posicionInicial    IS      posicionInicial
...
}

ROUTE manejadorExtremidad. xyzExtremoOut  TO otroNodo.eventoDeEntrada
```

La comunicación con otras entidades se define en la clase ***ManejadorExtremidad*** . Los identificadores subrayados son utilizados dentro de la implementación de la clase para referenciar a cada campo o evento. Los eventos de entrada se reciben a través del método ***processEvent(Event e)***, donde se deberá filtrar la información para distinguir entre un evento y otro.

```
public void processEvent(Event e){
    if(e.getName().equals("traslacionRobotIn")){
                float  temp = ((ConstSFVec3f)e.getValue()).getY();
        }
    if(e.getName().equals("rotaciónRobotIn ")){
...
```

Para accesar el evento de salida, es necesario crear un objeto que represente al evento:

```
SFVec3f  xyzExtremoOut  =  (SFVec3f) getEventOut("xyzExtremoOut");
....
```

Una vez echo esto podemos accesarlo con cualquiera de los métodos ya mencionados:

xyzExtremoOut.getValue();	Acceso de lectura.
xyzExtremoOut.setValue(tempY);	Cambia el contenido por su equivalente en tempY.

En caso de cambiar el contenido de un evento de salida el visualizador automáticamente enviará este mensaje a ***otroNodo*** a través de la directiva ***ROUTE***.
El acceso a los campos sigue un proceso similar al anterior:

```
SFVec3f posicionInicial  =  (SFVec3f) getField("posicionInicial");
```

posicionInicial.getValue();	Lectura del contenido.

posicionInicial.setValue(temp); Cambia el contenido por su equivalente en tempY.

Los cambios efectuados en los objetos se verán reflejados en los campos dentro del escenario VRML.

Identificación de clases del sistema

En esta fase se describe el análisis y diseño de aquellas clases que tendrán la responsabilidad de satisfacer los requerimientos particulares de la aplicación. Analizando los requerimientos del sistema, la especificación del robot y el bosquejo propuesto, podemos advertir la presencia de algunos de los objetos que a continuación se mencionan:

- Tableros de control
- Robot (cuerpo, grupos, extremidades)
- Terreno

Cada uno de los cuales será representado por entidades llamadas prototipos, con una representación gráfica y una implementación funcional, y ya que esta última en muchos casos funciona como un "manejador", al nombre de cada una de las clases propuestas se les antepone esa palabra:

ManejadorTableroControl
Esta clase implementa la generación de mensajes al robot de acuerdo a las acciones del usuario sobre objetos del escenario, tales como botones, barras deslizables, etc.

ManejadorCuerpoRobot
Implementa las metodologías de desplazamiento así como el control de las extremidades a través de los grupos.

ManejadorGrupo
Implementa el control de un grupo de 3 extremidades.

ManejadorExtremidad
Implementa los métodos necesarios para el movimiento y sensibilidad de cada extremidad, así como el cálculo de la cinemática directa e inversa de cada una de ellas.

Terreno
Implementa la generación del terreno a través de la invocación de los métodos de la clase Topología.

Topología
Esta clase provee los métodos que retornan el valor de la función f(x,y) para un punto en específico.

El tablero de control se descompone en un conjunto entidades :

ManejadorBoton
Genera eventos de valor booleano al activar un botón.

ManejadorSlider
Genera eventos cuyo valor es proporcional a la posición de la barra deslizante.

ManejadorCamaraRobot

Genera eventos cuyo contenido representa la posición angular del dispositivo de orientación.

Las clases que a continuación se describen, implementan el control de una extremidad utilizando una arquitectura Cliente/Servidor. Con ellas se pretende evaluar el desempeño de este esquema de trabajo..

Cinematica

Funge como un cliente a través del cual se calcula la cinemática directa e inversa para un robot de 3 grados de libertad en forma distribuía.

ServidorDeCinematica

Permite la conexión de un proceso cliente con un servidor para el cálculo de la cinemática de un robot de 3 grados de libertad. Espera una petición de servicio y delega su responsabilidad a ***ConecciónCinemática***. Hereda de la clase ***Thread*** pues debe enviar y recibir mensajes en forma independiente al proceso ***Script***.

ConeccionCinematica

Implementa el cálculo de la cinemática de un robot de 3 grados de libertad, el cliente envía un conjunto de coordenadas (x,y,z) y el servidor retorna las posiciónes angulares respectivas. Hereda de la clase ***Thread*** pues debe enviar y recibir mensajes en forma independiente al proceso ***Script***.

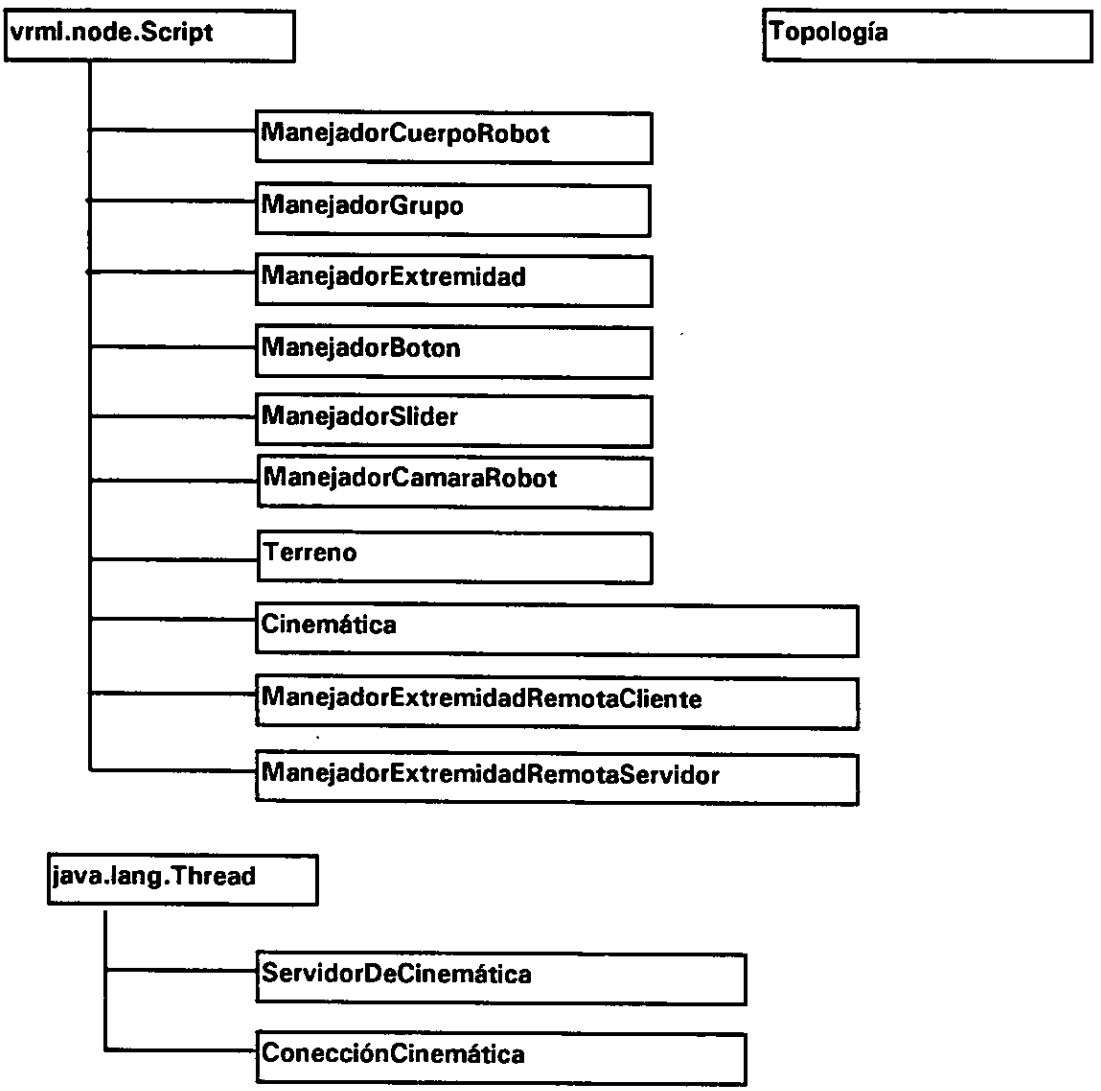
ManejadorExtremidadRemotaCliente

Esta clase implementa el control remoto de un robot de 3 grados de libertad a través de un canal de comunicación por sockets. Permitiendo la manipulación de la posición angular de cada uno de los eslabones en forma interactiva, enviando los comandos de rotación al servidor y esperando la respuesta de este una vez ejecutados.

ManejadorExtremidadRemotaServidor

Implementa la funcionalidad de un robot de 3 grados de libertad que puede ser controlado a distancia por un proceso cliente como, ***ManejadorExtremidadRemotaCliente***.
Controla su representación gráfica del robot para llevar a cabo las peticiones del cliente y retorna una serie de datos que representan la posición angular de cada eslabón.

La jerarquía de las clases antes mencionadas se muestra a continuación:



La estructura del sistema formado por estas clases se muestra en la Ilustración 55 e Ilustración 56.

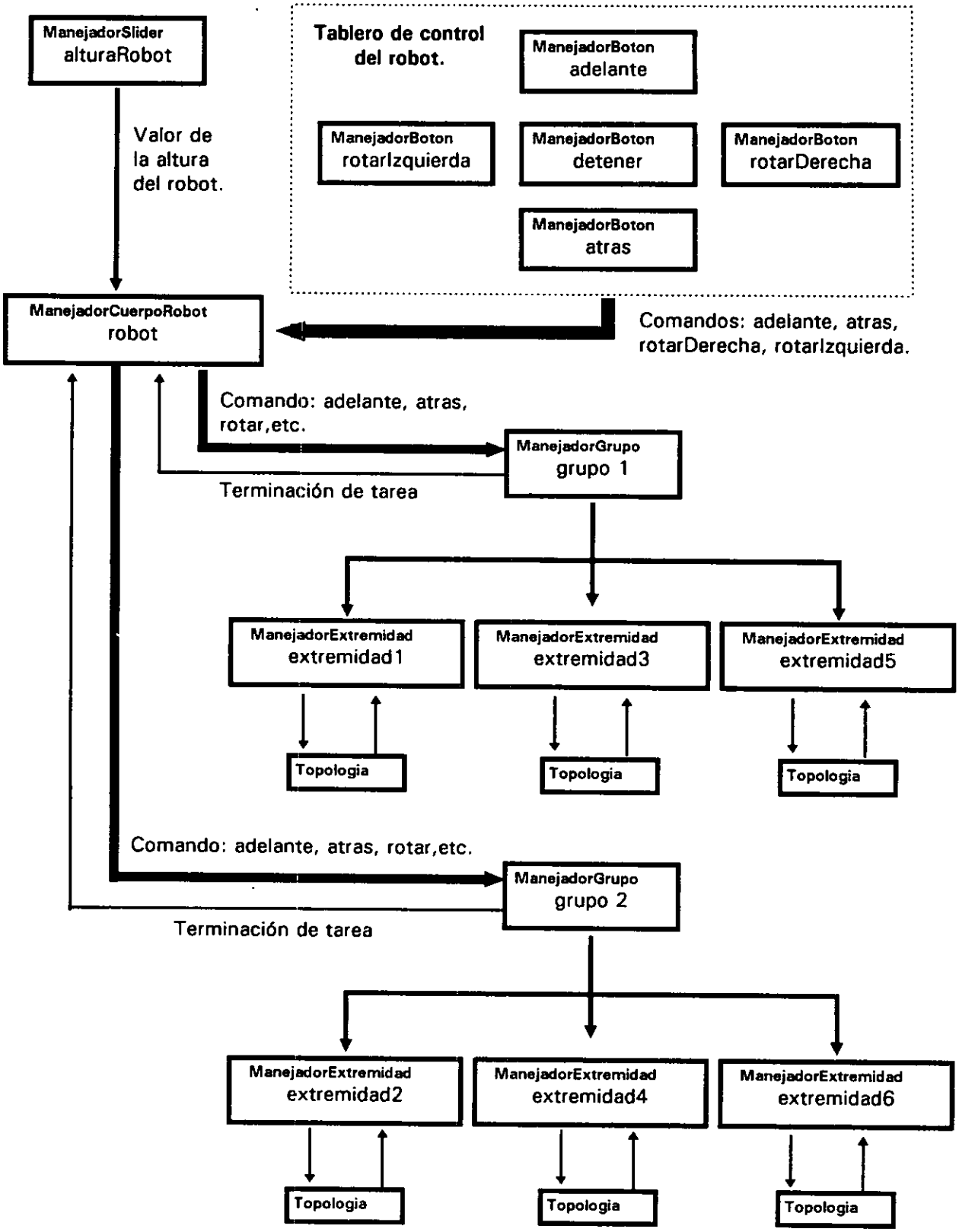


Ilustración 55. Estructura general del sistema de simulación y control.

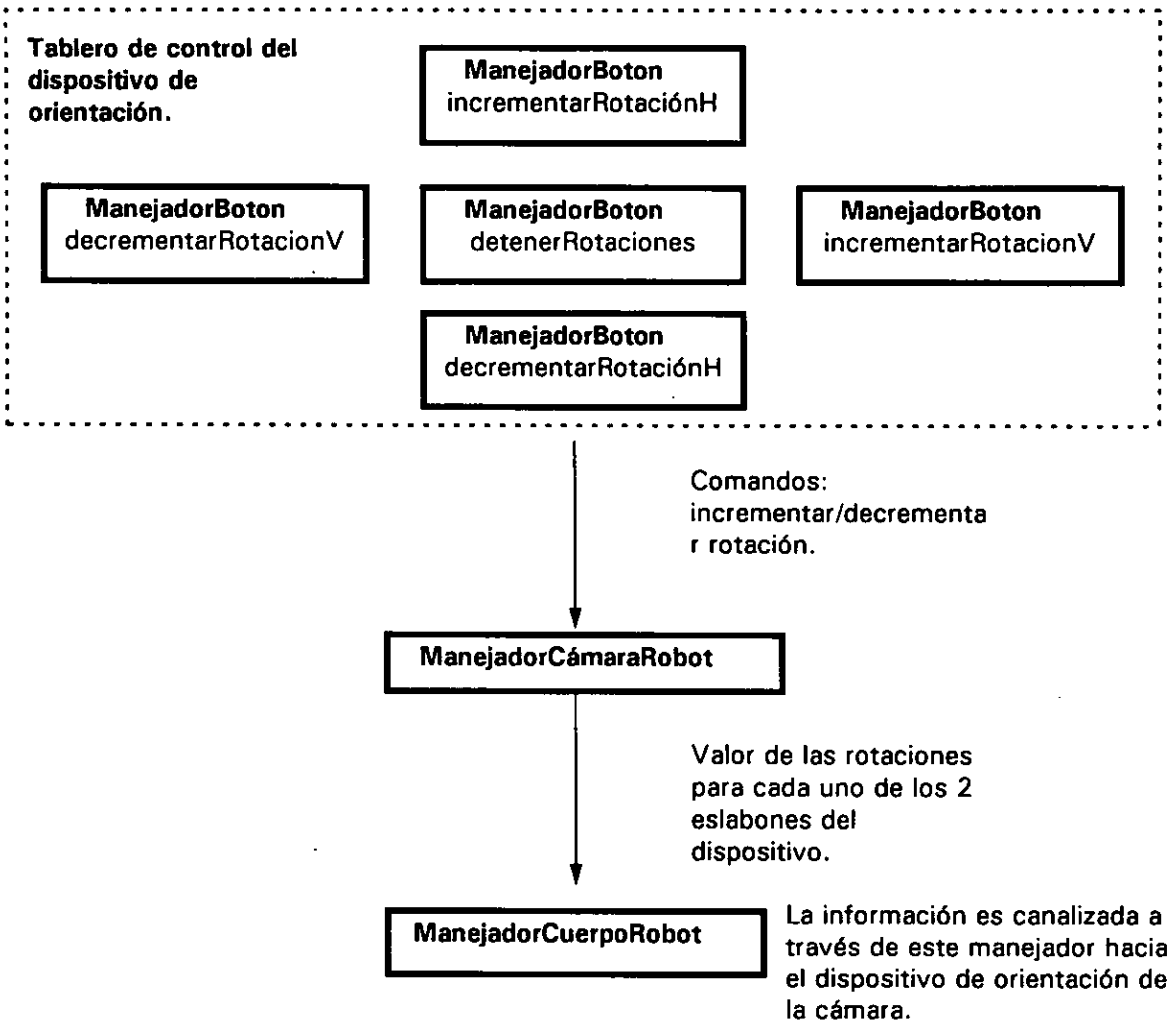


Ilustración 56. Control del dispositivo de orientación, como parte del sistema de simulación y control.

Definición de atributos y servicios

Una vez definidas las clases que conforman a nuestro sistema, es necesario especificar los atributos que tiene cada objeto y los servicios que deberá implementar. Es en este punto donde se delegan responsabilidades a cada entidad.

ManejadorCuerpoRobot

Atributos (variables de instancia):

- Posición sobre el terreno de prueba:
  posicionActual
- Orientación:
  rotacionXActual
  rotacionYActual
  rotacionZActual
- Grupos de extremidades a controlar :
  grupo1AccionOut
  grupo2AccionOut
- Actividad a desarrollar: adelante, atrás, rotar, etc.
  accionActual
- Bandera para indicar el grupo a controlar:
  grupoActivo
- Bandera del estado de actividad de cada grupo:
  grupo1TerminoAccionIn
  grupo2TerminoAccionIn
- Fase de desarrollo del actual comando:
  fase

Servicios que proporciona (métodos de instancia):

- Ejecución de tareas:
  desarrollaAcciones()
- Intercambio de grupo activo:
  intercambiaGrupos()
- Desplazamiento y orientación del cuerpo del robot en función a la actividad a desarrollar:
  movilizaRobot()
- Envío de comandos al grupo activo.
  enviaAccionAGrupoActivo(int accionG)

- Comunicación con otras entidades a través de eventos de entrada y salida.

ManejadorGrupo

Atributos (variables de instancia):

- Las extremidades a controlar:
  accionExtremidadFrontalOut
  accionExtremidadMediaOut
  accionExtremidadTraseraOut

- Tarea a desarrollar: adelante, atrás, rotar, etc.
  accionActual

- Bandera de estado de la actividad de cada extremidad:
  extremidadFrontalActividadFinalizada
  extremidadMediaActividadFinalizada
  extremidadTraseraActividadFinalizada

Servicios que proporciona (métodos de instancia):

- Ejecución del comando, y coordinación de extremidades.
  desarrollaAcciones()

- Comunicación con otras entidades a través de eventos de entrada y salida.

ManejadorExtremidad

Atributos (variables de instancia):

- Posición sobre el cuerpo del robot:
  posicionInicial

- Orientación sobre el cuerpo del robot, en el eje Y:
  rotacionYInicial

- Posición del cuerpo del robot:
  posicionActualRobot

- Orientación del cuerpo del robot:
  rotaciónXActualRobot
  rotaciónYActualRobot
  rotaciónZActualRobot

- Posición del extremo del eslabón 1 (órgano terminal):
  xExtremo
  yExtremo
  zExtremo

- Magnitud del desplazamiento de los eslabones en cada iteración:
  deltaTraslacion

- Dimensiones de los eslabones:
  escala (escala de las dimensiones del robot)
  l1, l2, l3,l4 (longitudes de los eslabones 1,2,3,4, traslación sobre el eje X)
  h1,h2,h3,h4 (altura de los eslabones 1,2,3 ,4, traslación sobre el eje Y)

- Posición angular los eslabones:
  rot1Out
  rot2Out
  rot3Out

- Tarea a desarrollar: movimiento hacia arriba, hacia abajo, rotar, etc.
  accionADesarrollar

- Bandera para indicar colisión con el terreno:
  colisionConTerreno

Servicios que proporciona (métodos de instancia):

- Relaciona el mensaje del tablero de control con una tarea a desarrollar:
  asignaAccion()

- Ejecución de la tarea durante una iteración:
  desarrollaAcciones()

- Cálculo de cinemática directa e inversa:
  cinematicaPuntoExtremo( )

- Determina contacto con el terreno:
  evaluaColision()

- Comunicación con otras entidades a través de eventos de entrada y salida.

Terreno

Atributos (variables de instancia):

- Dimensiones del terreno:
  largo
  ancho

Servicios que proporciona (métodos de instancia):

- Comunicación con otras entidades a través de eventos de entrada y salida.

Topología

Atributos (variables de instancia):

- No requiere variables de instancia.

Servicios que proporciona (métodos de instancia):

- Funciones f(x,y) generadoras de una determinada topología:
  altura (int numero, float x, float z).

ManejadorBoton

Atributos (variables de instancia):

- Bandera de estado (activado/desactivado):
  activo

Servicios que proporciona (métodos de instancia):

- Comunicación con otras entidades a través de eventos de entrada y salida.

ManejadorSlider

Atributos (variables de instancia):

- Nivel marcado por la barra deslizante:
  nivelOut

Servicios que proporciona (métodos de instancia):

- Comunicación con otras entidades a través de eventos de entrada y salida.

ManejadorCamaraRobot

Atributos (variables de instancia):

- Magnitud del desplazamiento por cada iteración:
  deltaRotacion

- Posición angular de los eslabones:
  rotacionAAActual (rotación sobre el eje horizontal)
  rotacionDIActual (rotación sobre el eje vertical)

Servicios que proporciona (métodos de instancia):

- Comunicación con otras entidades a través de eventos de entrada y salida.

Cinematica

Atributos (variables de instancia):

- Dimensiones de los eslabones.
  escala (escala de las dimensiones del robot)
  l1, l2, l3, l4 (longitudes de los eslabones 1,2,3,4, traslación sobre el eje X)
  h1, h2, h3, h4 (altura de los eslabones 1,2,3 ,4, traslación sobre el eje Y)

- Posición angular de cada eslabón:
  rot1
  rot2
  rot3

- Posición del órgano terminal:
  puntoE

Servicios que proporciona (métodos de instancia):

- Implementación del cálculo de cinemática directa e inversa.
- Comunicación con otras entidades a través de eventos de entrada y salida.

ServidorDeCinematica

Atributos (variables de instancia):

- Puerto lógico:
  puerto

- Socket servidor:
  listen_socket

- Socket de comunicación:
  socket

Servicios que proporciona (métodos de instancia):

- Redefinición de los métodos ***run()*** y ***main()*** .

ConeccionCinematica

Atributos (variables de instancia):

- Socket de comunicación con el robot:
  socketRobot

- Flujo del robot.
  datosDelRobot

- Flujo al robot.
  datosAlRobot

- Dimensiones de los eslabones.
  escala (escala de las dimensiones del robot)
  l1, l2, l3, l4 (longitudes de los eslabones 1,2,3,4, desplazamiento sobre X)
  h1, h2, h3, h4 (altura de los eslabones 1,2,3 ,4, desplazamiento sobre Y)

Servicios que proporciona (métodos de instancia):

- Redefinición de los métodos ***run()*** y ***main()*** para inicialización de variables y recopilación de información..

ManejadorExtremidadRemotaCliente

Atributos (variables de instancia):

- Socket de comunicación
  socket

- Puerto lógico del socket:
  puerto

- Acción a realizar por eslabón
  accionActual1 (incrementar/decrementar/detener la actual posición angular del eslabón 1)
  accionActual2 (incrementar/decrementar/detener la actual posición angular del eslabón 2)
  accionActual3 (incrementar/decrementar/detener la actual posición angular del eslabón 3)

- Posición angular de los eslabones
  rotacionEslabon1
  rotacionEslabon2
  rotacionEslabon3

Servicios que proporciona (métodos de instancia):

- Comunicación con otras entidades a través de eventos de entrada y salida .

ManejadorExtremidadRemotaServidor

Atributos (variables de instancia):

- Socket servidor
  listen_socket
- Socket de comunicación
  socket
- Puerto lógico del socket:

puerto (puerto lógico donde colgar el socket)

- Tarea de cada eslabón:

accionActual1 (incrementar/decrementar/detener la actual posición angular del eslabón 1)

accionActual2 (incrementar/decrementar/detener la actual posición angular del eslabón 2)

accionActual3 (incrementar/decrementar/detener la actual posición angular del eslabón 3)

- Rotación actual de cada uno de los eslabones:
  rotacionEslabon1
  rotacionEslabon2
  rotacionEslabon3
- Límites de rotación de cada eslabón, similares a las limitaciones físicas del robot:
  limiteSuperior1
  limiteSuperior2
  limiteSuperior3
  limiteInferior1
  limiteInferior2
  limiteInferior3

Servicios que proporciona (métodos de instancia):

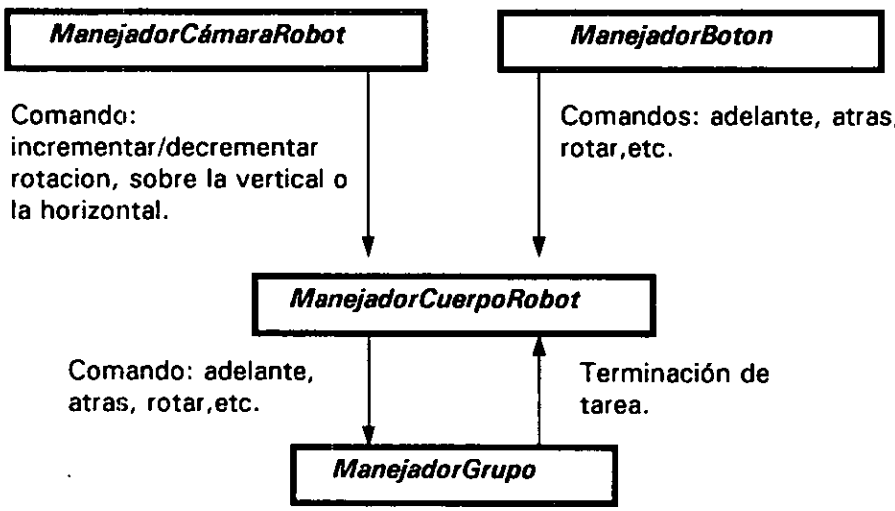
- Comunicación con otras entidades a través de eventos de entrada y salida.

Colaboración entre objetos

Una vez identificadas las clases que intervienen en el funcionamiento del sistema, debemos detallar cómo deberán de colaborar entre sí cada uno de sus objetos. En los siguientes esquemas se muestran los objetos pertenecientes a cada una de las clases implementadas y la relación que guardan entre si.

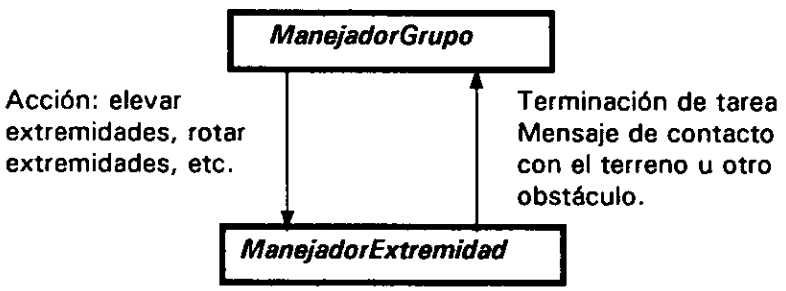
ManejadorCuerpoRobot

Recibe mensajes de los objetos ***ManejadorBoton*** y ***ManejadorSlider***, los interpreta y genera una secuencia de control que envía a cada ***ManejadorGrupo*** para su ejecución. Por otro lado interactúa con un objeto de la clase ***ManejadorCámara*** para el control del dispositivo de orientación.



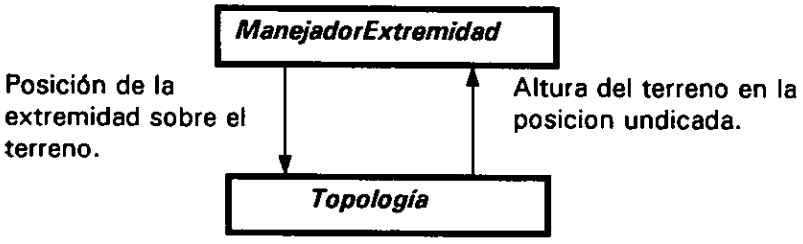
ManejadorGrupo

Implementa el control de un grupo de 3 extremidades a través del ***ManejadorExtremidad*** de cada uno de estas.



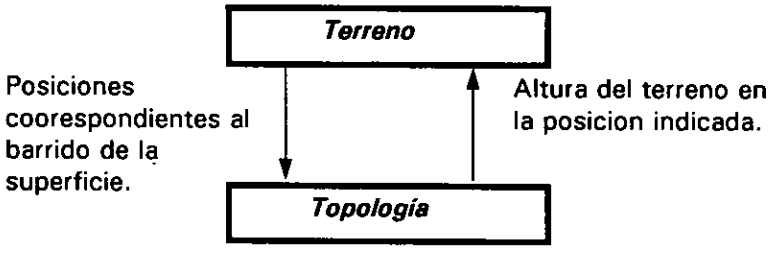
ManejadorExtremidad

Se auxilia de la clase ***Topología*** para determinar el punto en que la extremidad hace contacto con el terreno sobre el cual se desplaza. Puede colaborar con objetos de la clase ***Cinemática*** para el cálculo local o bien de las clases ***ServidorDeCinematica*** y ***ConeccionCinematica*** como procesos auxiliares en sistemas remotos.



Terreno

Genera el terreno invocando los métodos de la clase ***Topología***.



Topología

Implementa métodos para la generación de la morfología del terreno a través de la evaluación de funciones f(x,y) predeterminadas. Interactúa con objetos de la clase ***Terreno*** y de la clase ***ManejadorExtremidad***.

ManejadorBoton

Genera eventos de valor lógico verdadero o falso al activarlo con la acción del usuario sobre una entidad gráfica. Colabora con entidades de las clases ***ManejadorCuerpoRobot*** y ***ManejadorCamaraRobot*** sirviendo como una interfaz de control para el usuario.

ManejadorSlider

Genera eventos en función de la posición de su puntero o barra deslizante, colabora con la clase ***ManejadorCuerpoRobot*** sirviendo como una interfaz de control para el usuario.

ManejadorCamaraRobot

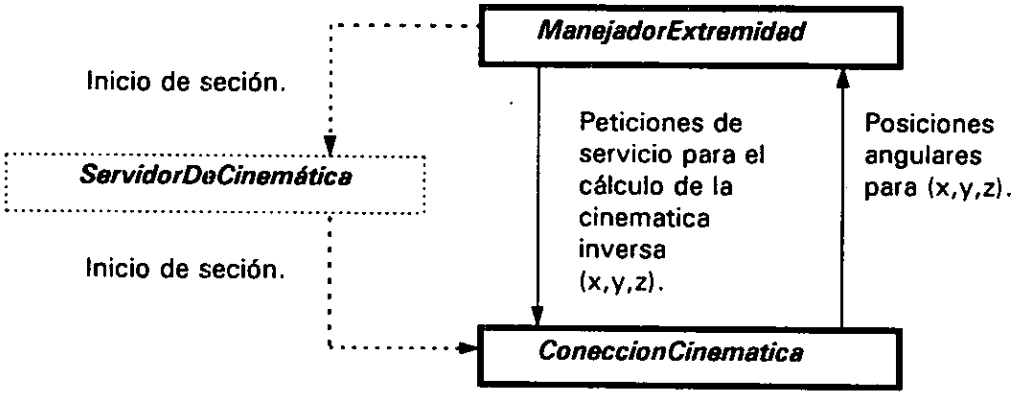
Genera eventos cuyos valores representan la posición angular del dispositivo de orientación de la cámara. Colabora con objetos de la clase ***ManejadorBoton*** a través de los cuales interactúa con el usuario.

Cinemática

Implementa el cálculo de la cinemática directa e inversa para un robot de 3 grados de libertad. Colabora con los objetos de la clase ***ManejadorExtremidad***.

ServidorDeCinemática

Colabora con objetos de las clases ***ManejadorExtremidad*** y ***ConeccionCinematica*** para iniciar una sección.

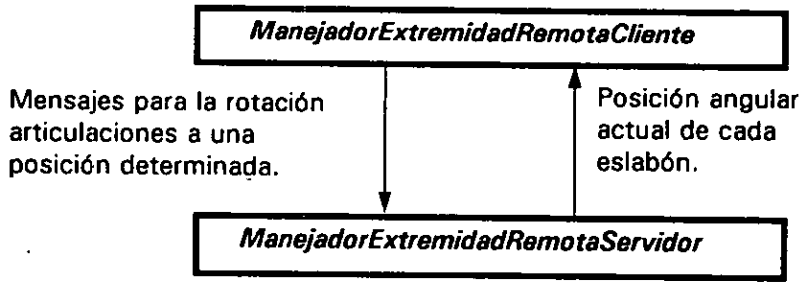


ConeccionCinematica

Implementa el cálculo de la cinemática directa e inversa para un robot de 3 grados de libertad, colabora como un proceso auxiliar con los objetos de la clase ***ManejadorExtremidad***.

ManejadorExtremidadRemotaCliente

Colabora con los objetos de la clase ***ManejadorExtremidadRemotaServidor***, implementando el control remoto de una extremidad.



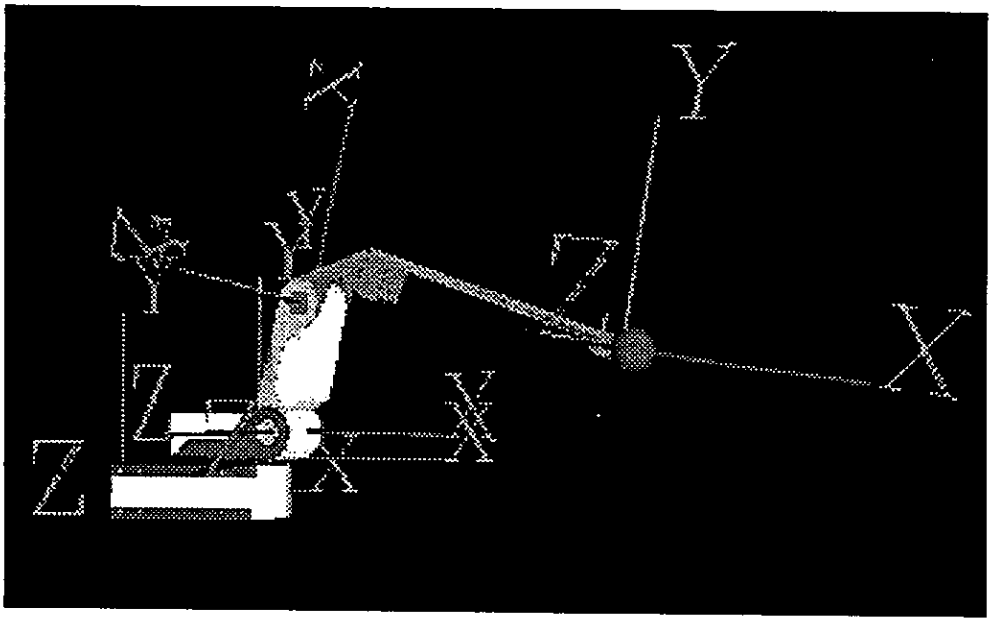
ManejadorExtremidadRemotaServidor

Colabora con los objetos de la clase **ManejadorExtremidadRemotaCliente** implementando el control remoto de una extremidad.

Creación de un prototipo

El objetivo de esta fase es generar un prototipo con el cual probar el funcionamiento de las entidades que forman el sistema. Conforme se avanza en el desarrollo de esta fase se incorporan los objetos diseñados en fases anteriores y se depura su codificación.

Como primer paso, se implementan los prototipos de las extremidades a manera de módulos independientes. La clase ***ManejadorExtremidad*** es incorporada para el cálculo de la cinemática directa e inversa del robot, así como el control de la representación gráfica. Los prototipos de las extremidades son probados utilizando una serie de interpoladores que generan las posiciones angulares de cada eslabón. Con base a esta información se obtiene la cinemática directa, es decir un punto (x,y,z) que se representa con una esfera. Una vez obtenidas estas coordenadas, y a manera de comprobación, se obtiene la cinemática inversa, la cual se representa rotando los ejes de referencia localizados en la base de cada eslabón. En la Ilustración 57 pueden apreciarse los elementos descritos.



Los ejes de referencia son orientados de acuerdo al cálculo de la cinemática inversa.

La posición de la esfera representa el cálculo de la cinemática directa.

Ilustración 57. Vista de una de las extremidades durante una simulación del modelo geométrico del robot.

Como siguiente paso, se implementa un prototipo que genera el terreno de prueba utilizando las clases ***Topología*** y ***Terreno***.. Por otro lado se extiende la funcionalidad del prototipo que representa las extremidades, las clases ***ManejadorExtremidad*** y ***Topología*** colaboran para simular la detección de las irregularidades del terreno. El funcionamiento de estas entidades se comprueba utilizando un "terreno de prueba" sobre la cual el usuario puede desplazar una esfera, la extremidad seguirá su movimiento y el contorno de la superficie, calculando en cada punto la cinemática directa e inversa. En esta prueba es posible desplazar la base de la extremidad, esta última adaptará la posición de sus eslabones para seguir en contacto con la esfera. La Ilustración 58 muestra el resultado de la simulación.

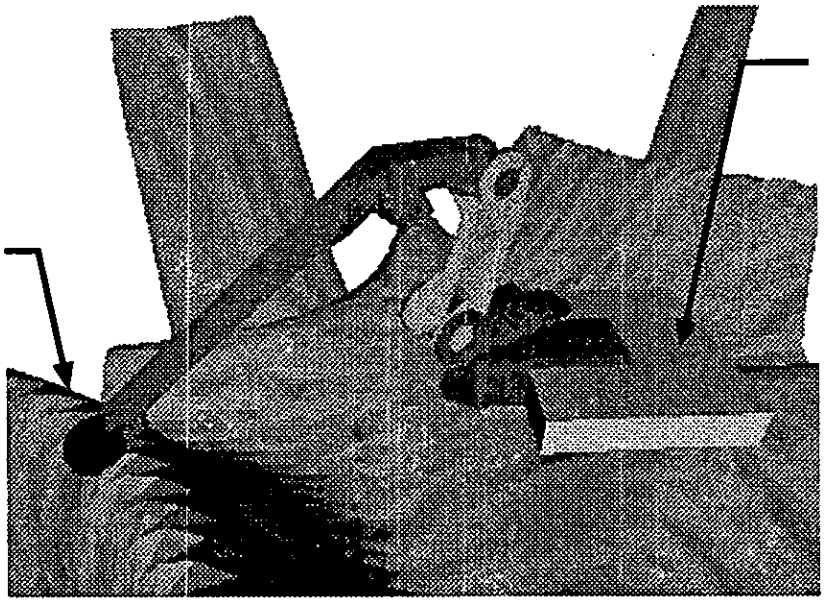
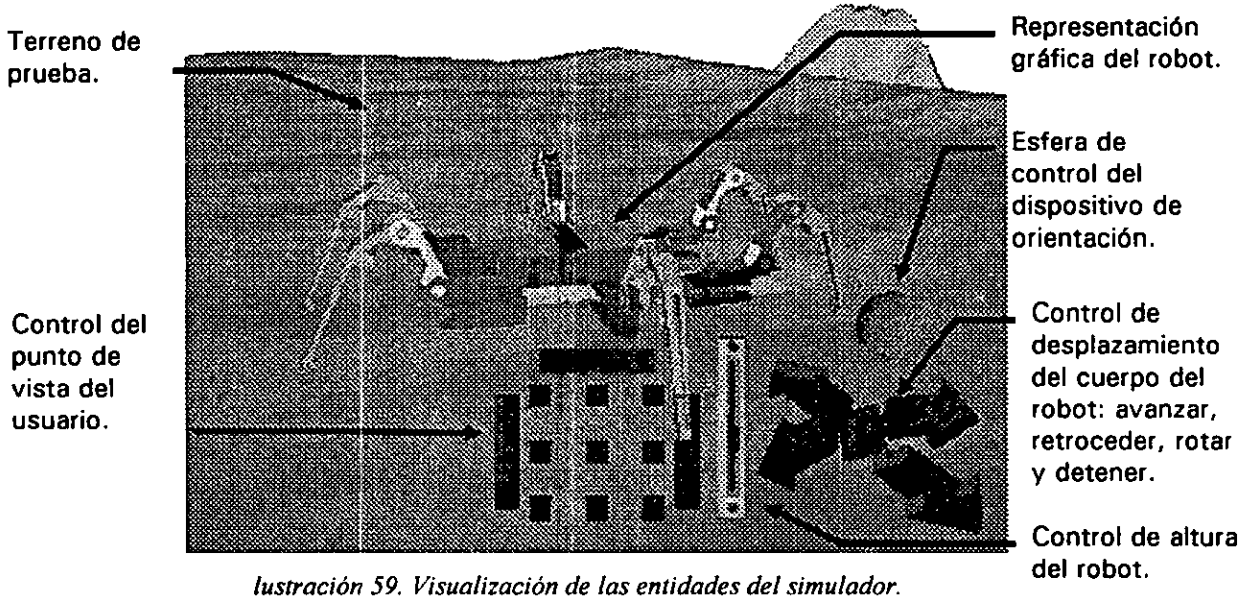


Ilustración 58. Vista lateral del robot durante el proceso de desplazamiento.

Una vez implementadas las entidades se procede a acoplarlas y comprobar la colaboración entre ellas. Las clases ***ManejadorGrupo***, ***ManejadorCuerpoRobot*** y ***ManejadorCamaraRobot*** son incorporadas al prototipo del robot para coordinar las acciones de las extremidades y del dispositivo de orientación. El tablero de control es implementado utilizando las clases ***ManejadorBoton*** y ***ManejadorSlider***. Para comprobar la correcta comunicación entre todas las entidades se implementó parte de las funciones de desplazamiento y orientación del cuerpo del robot. El usuario puede manipularlo por medio del tablero de control y visualizar la simulación a través de las 10 cámaras dispuestas al rededor de este. En la Ilustración 60 y la Ilustración 61 se muestra al robot rotando sobre su propio eje, mientras que en la Ilustración 62 se le puede observar desplazando su cuerpo hacia atras.



lustración 59. Visualización de las entidades del simulador.

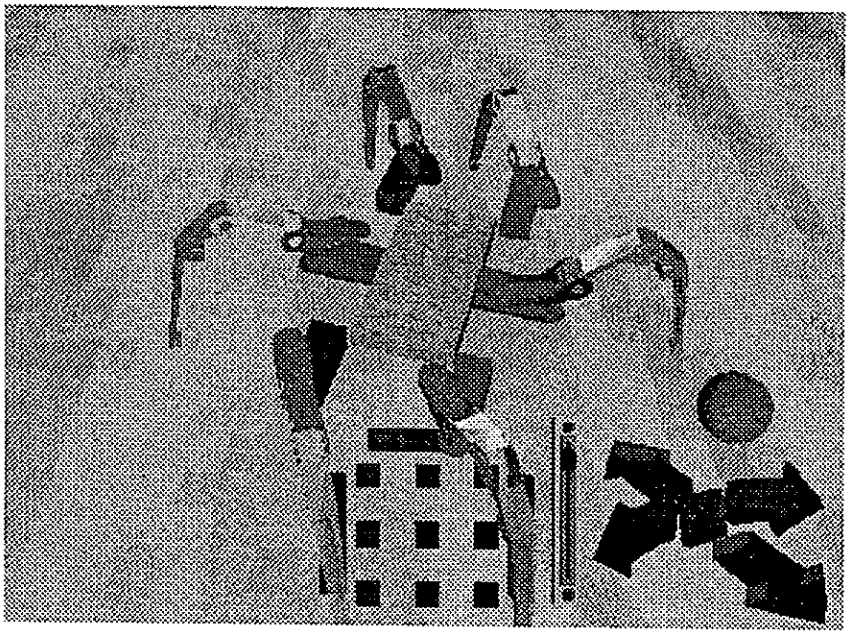


Ilustración 60. Vista aérea del robot durante el proceso de rotación.

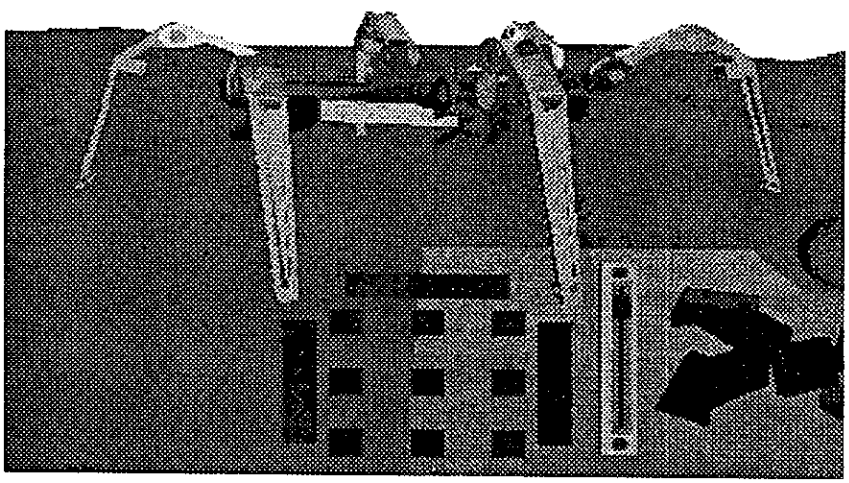


Ilustración 61. Vista frontal del robot durante el proceso de rotación.

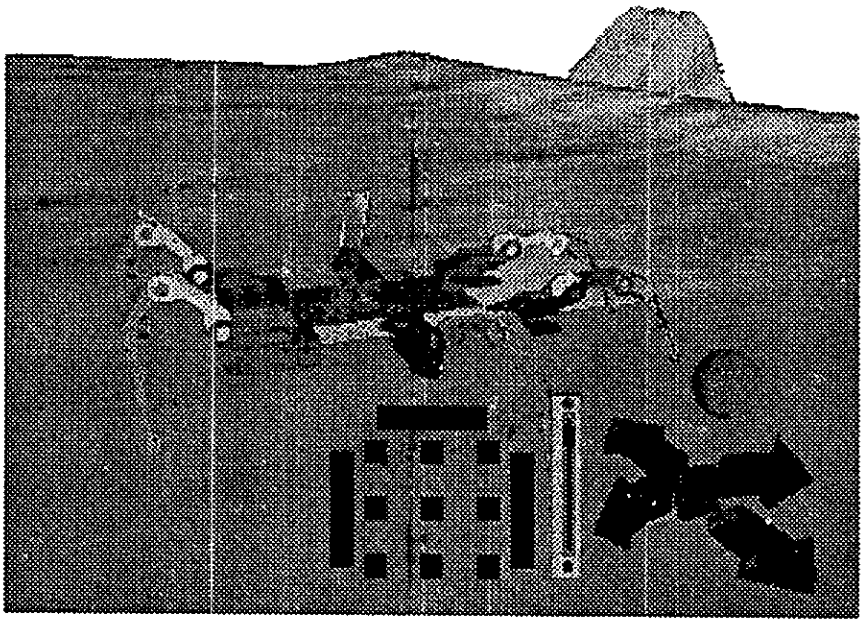
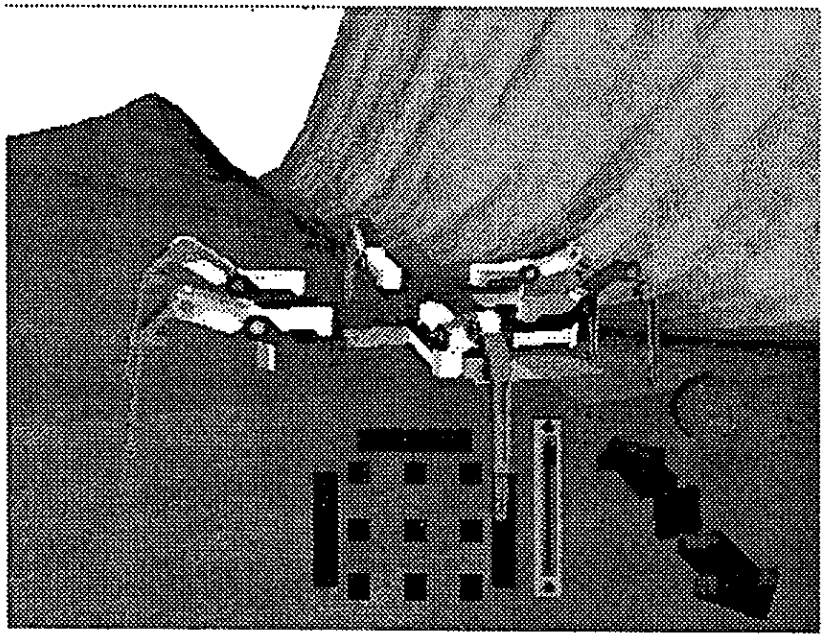


Ilustración 62. Vista lateral del robot durante el proceso de desplazamiento hacia atrás.

Extensión del prototipo

Una vez acopladas las entidades del simulador y comprobada su correcta comunicación se procede a completar la implementación de cada una de ellas. Se incorporan los métodos de desplazamiento de cada una de las extremidades en las respectivas clases ***ManejadorExtremidad***, de forma que podrán responder a mensajes como: elevar extremidad, bajar extremidad, rotar, etc., en los cuales se basan tareas más complejas como lo es el desplazamiento y orientación del robot.
Los 2 objetos de la clase ***ManejadorGrupo*** asumen su papel de coordinador de un conjunto de 3 extremidades, mientras que el objeto ***ManejadorCuerpoRobot*** controla a ambos grupos. La Ilustración 64 muestra la interfaz del simulador al momento de iniciar la aplicación. Los resultados obtenidos de las simulaciones se muestran en la Ilustración 64 e Ilustración 65 donde se observa el robot desplazándose a través del terreno de prueba.



Control de desplazamiento del robot: avanzar, retroceder y detener.

Ilustración 63. Vista lateral del robot en la posición inicial.

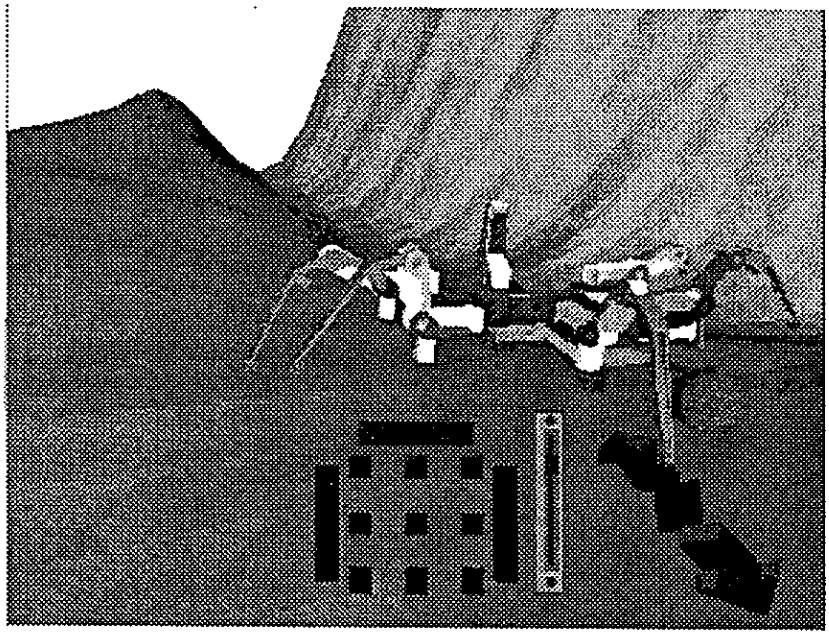


Ilustración 64. Vista lateral del robot durante el proceso de desplazamiento, el vector indica el sentido del movimiento.

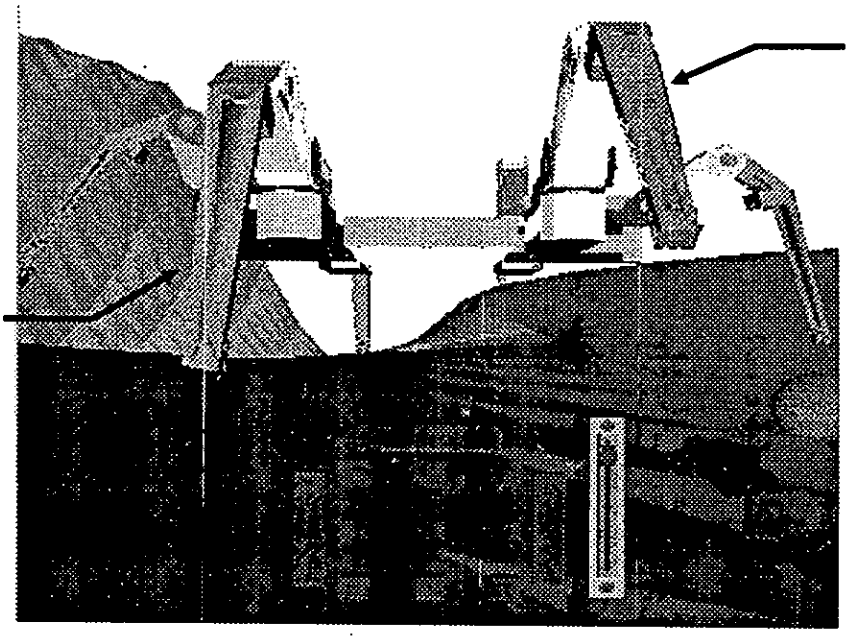


Ilustración 65. Vista trasera del robot con las extremidades de uno de los grupos elevadas durante el proceso de desplazamiento.

Implementación y pruebas del control de una extremidad a través de una arquitectura Cliente/Servidor

Las aplicaciones que a continuación se describen, implementan la comunicación entre procesos siguiendo una arquitectura Cliente/Servidor y servirán para comprobar la eficacia de este tipo de arquitectura en el sistema de simulación.

Para llevar a cabo las pruebas se implementan algunos cambios en los prototipos de las extremidades con el fin de que puedan accesar los servicios de procesos externos. La clase ***Cinemática*** es modificada para fungir como cliente de un proceso externo formado por las clases ***ConeccionCinematica*** y ***ServidorDeCinemática***, al cual envía una serie de puntos para los cuales requiere el cálculo de la cinemática inversa, como respuesta recibe las posiciones angulares respectivas (ver Ilustración 66). Del lado del proceso cliente está el visualizador que muestra las acciones del robot, en esta ocasión no se despliega ningún tablero de control dado que la aplicación no es interactiva, las secuencias de movimientos utilizados se obtienen de interpoladores.

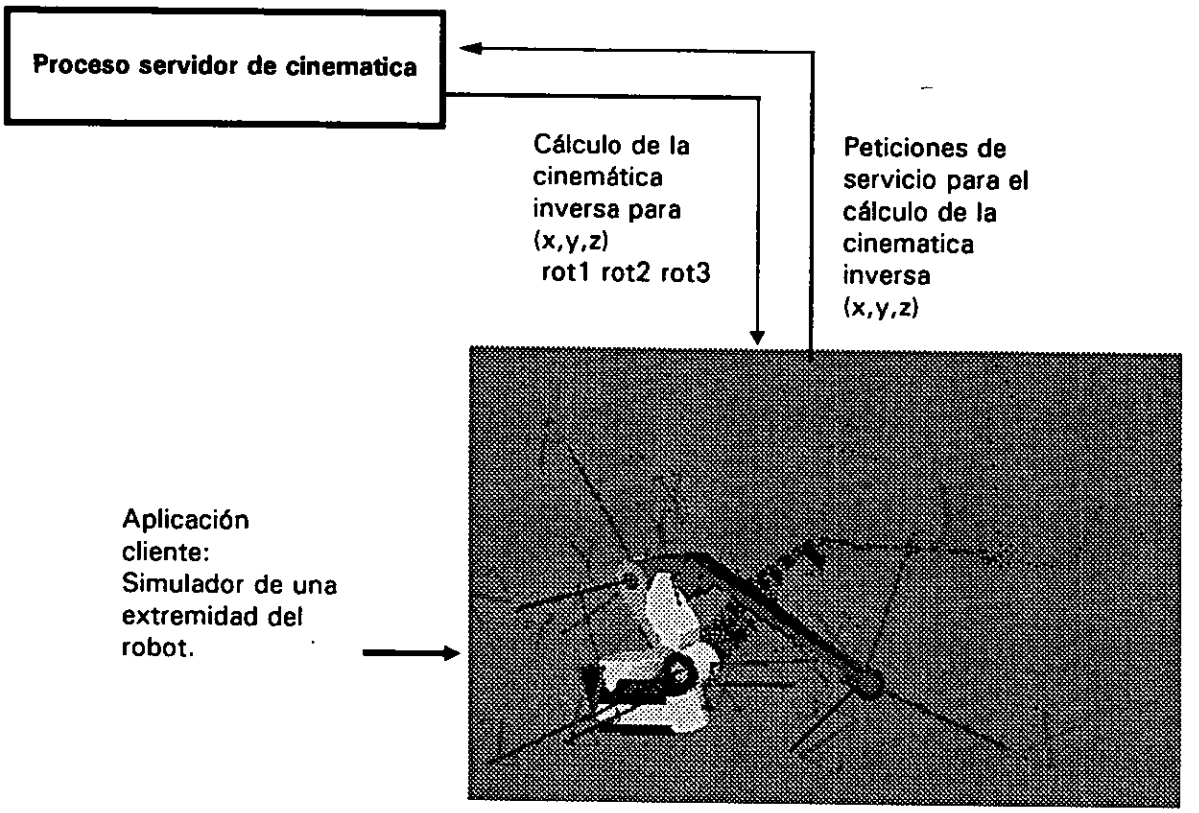


Ilustración 66. Aplicación cliente/servidor para el cálculo de la cinemática inversa de una extremidad del robot.

Este esquema de trabajo permite al proceso servidor interactuar con clientes que bien puede ser un simulador o el controlador del robot real, con la única condición de cumplir con el protocolo de mensajes (ver Ilustración 67).

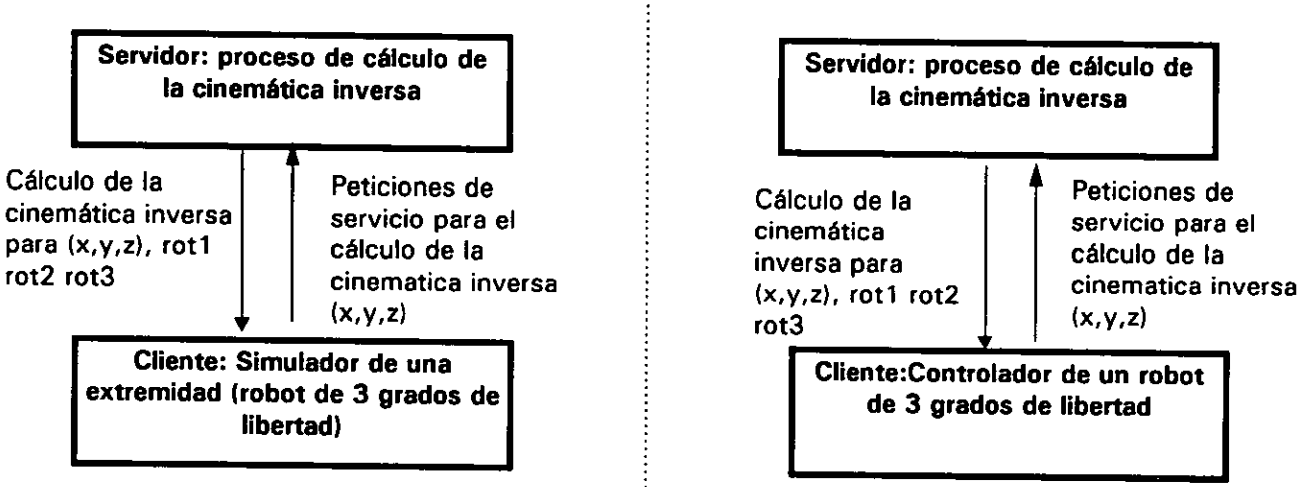


Ilustración 67. Comunicación con procesos de simulación y con procesos de control de dispositivos físicos.

Para complementar las pruebas de comunicación entre procesos se desarrolla otra aplicación que, a diferencia de la anterior, permite la interacción con el usuario. A través de un tablero de control en la aplicación cliente, se manipula la posición angular de cada uno de los eslabones del robot en la aplicación servidor (ver Ilustración 68). El tablero de control envía una serie de

mensajes que indican en que sentido debe rotarse cada eslabón el simulador, este último a su vez ejecuta la tarea y retorna periódicamente la posición angular que toman las partes del robot en cada iteración.
Las aplicaciones cliente y servidor utilizan las clases ***ManejadorExtremidadRemotaCliente*** y ***ManejadorExtremidadRemotaServidor*** respectivamente.

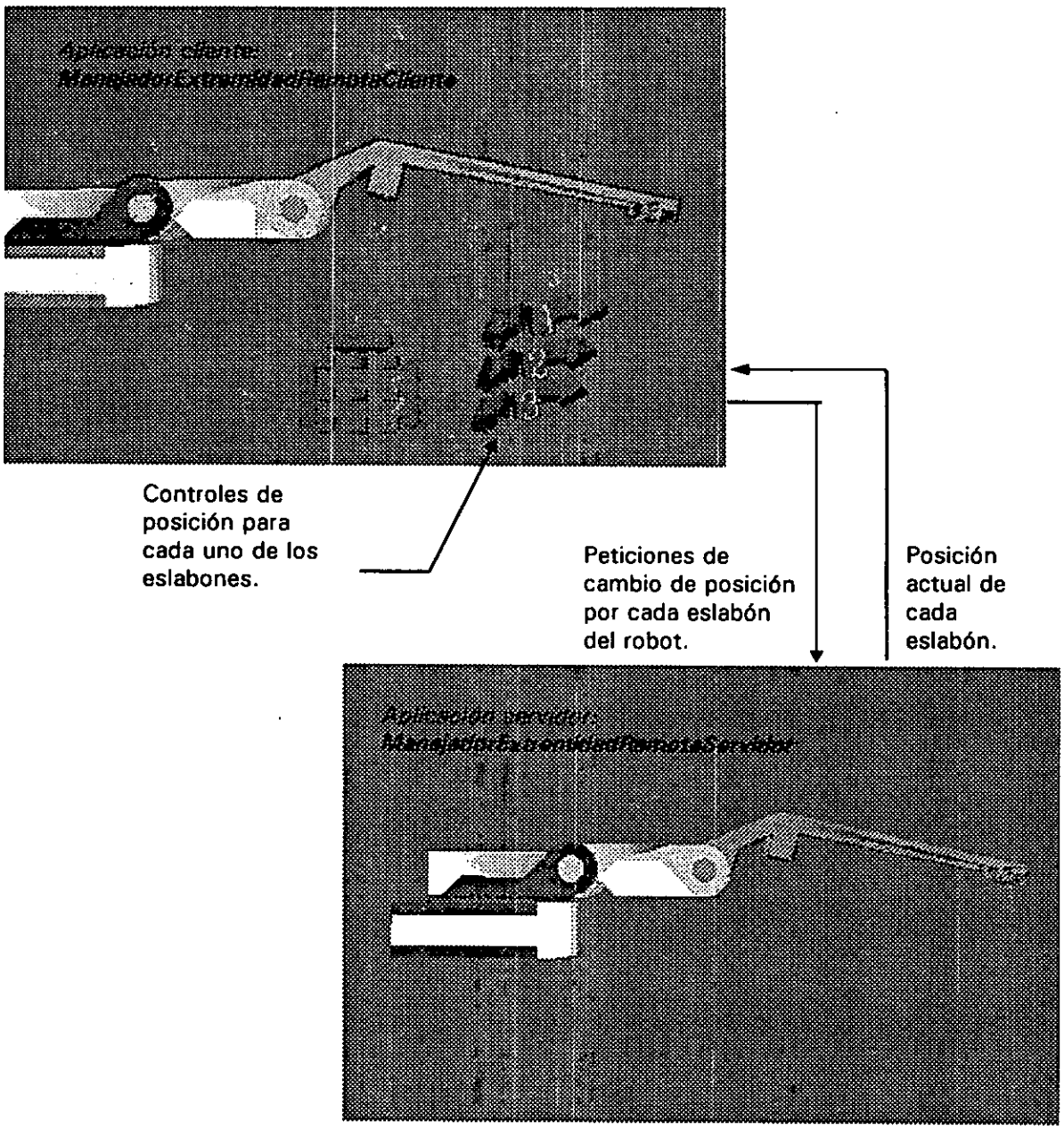


Ilustración 68. Aplicación cliente/servidor para el control remoto de una extremidad.

Pruebas y resultados

Una vez implementada la aplicación se procedió a realizar algunas pruebas de desempeño en computadoras de diferentes características:

- **WS Sparc Ultra Driven**, 170 Mhz, 64 Mb en memoria RAM y 1 Gb en DD local, 8 bits en la paleta de colores del manejador de ventanas .
- **WS Sparc 20 Server**,130 Mhz con 128 Mb en memoria RAM y 1 Gb en DD local, 24 bits en la paleta de colores del manejador de ventanas .
- .**WS Sparc 4**, 80 Mhz con 32 Mb en memoria RAM y 1 Gb en DD local, 8 bits en la paleta de colores del manejador de ventanas .
- **PC Pentium.** 90 Mhz, 32 Mb en memoria RAM y 1Gb en DD, 24 bits en la paleta de colores del manejador de ventanas.

Utilizando las herramientas de medición de recursos de cada sistema se observó una carga del procesador del 100%, sin que por ello se notara una depreciación en su desempeño general. A pesar de existir grandes diferencias en la capacidad de procesamiento entre estas computadoras, la velocidad de simulación y de despliegue no mostraron grandes diferencias. Estos indicadores hacen pensar que la implementación del visualizador hace uso de procesos de baja prioridad que utilizan completamente el procesador, pero que de requerirse, permiten el acceso a otros procesos, evitando así la saturación del sistema. Por otro lado se observó que el rendimiento de la implementación para PC baja excesivamente conforme se añaden nodos ***Script***. a los prototipos.

Las pruebas de los procesos Cliente/Servidor se realizaron implementando conexiones lógicas dentro de la redes locales del IIMAS y del Centro de Instrumentos, y en algunos casos entre ambas redes. Ambos laboratorios cuentan con una red ethernet cuya velocidad de transferencia es de 10Mbits/s. La carga de trabajo en dichas redes durante las pruebas fue variable, sin embargo, la transferencia de mensajes es mínima entre los procesos, se estima una transferencia de 10 paquetes/s con una extensión de 10 a 20 bytes cada uno, por lo que no existe un cambio aparente en el desempeño de la aplicación entre un período de uso intensivo de la red y otro con carga moderada. La distribución de tareas entre varias computadoras utilizando un arquitectura Cliente/Servidor mejora el desempeño de cada una de ellas desahogando su carga de trabajo. Sin embargo para conservar el desempeño en un sistema, debe asegurarse que el tiempo de procesamiento externo, en el cual se requiere un cierto tiempo de transferencia, sea menor que el procesamiento local.

Una de las principales ventajas de trabajar con una arquitectura Cliente/Servidor, es que los procesos son tratados a manera de cajas negras donde sólo importan los formatos de la información de entrada y la de salida. Esta característica permitirá hacer pruebas con un simulador o bien con el software de control del robot, sin que ello implique cambio alguno en el código de los programas. Mediante el simulador del robot podrán depurarse los algoritmos de control y coordinación sin necesidad de utilizar un dispositivo físico, evitándole así posibles daños. Una vez obtenidos los resultados requeridos en las simulaciones, se procede a su implementación en el controlador del robot real.

Desarrollo del controlador de una extremidad del robot

Dentro de los requerimientos del sistema se especifica que las clases generadas para el simulador deben representan una base para la implementación del controlador del robot. Con el fin de apoyar esta especificación y como parte de los resultados obtenidos se muestra a continuación un sencillo controlador programado en C++ para una extremidad robot. El desarrollo del controlador utiliza la clase ***ManejadorExtremidad*** como modelo para el diseño de una nueva clase denominada ***Dispositivo.*** Las instancias de esta última representan a las extremidades y al dispositivo de orientación como entidades lógicas, cuyos métodos implementan: el control de actuadores y la interpretación de señales de los sensores. El acceso a los puertos de la computadora, donde está conectada la interfaz electrónica se implementa en la clase ***ManejadorPuerto*** (ver Ilustración 69).

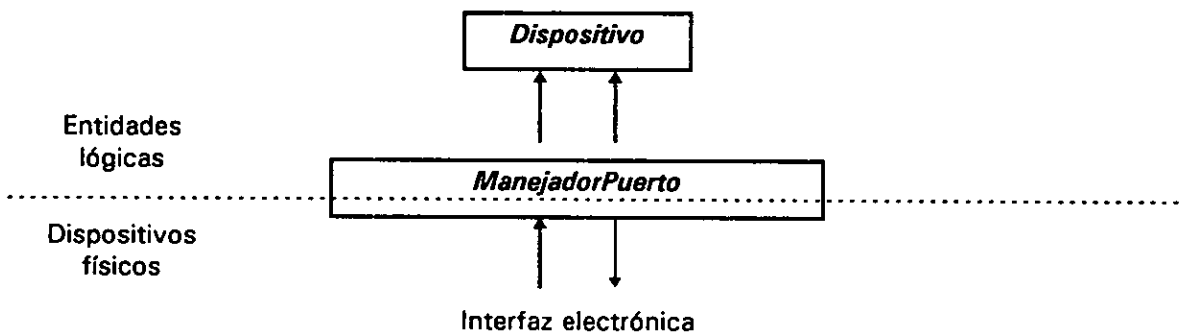


Ilustración 69. Control de los dispositivos a través de la clase ***ManejadorPuerto.***

El control de posición implementado por las entidades de la clase ***Dispositivo*** está representado en la Ilustración 70. Los métodos ***decrementarRotación(), incrementarRotación()*** y ***detenerRotación()*** accesan la variable de instancia ***posiciónRequerida***, con base a la diferencia entre esta última y ***posiciónActual*** se determina en que dirección debe rotar el actuador. Para inducir movimiento en el actuador se envía una señal a la interfaz electrónica utilizando para ello la clase ***ManejadorPuerto***.

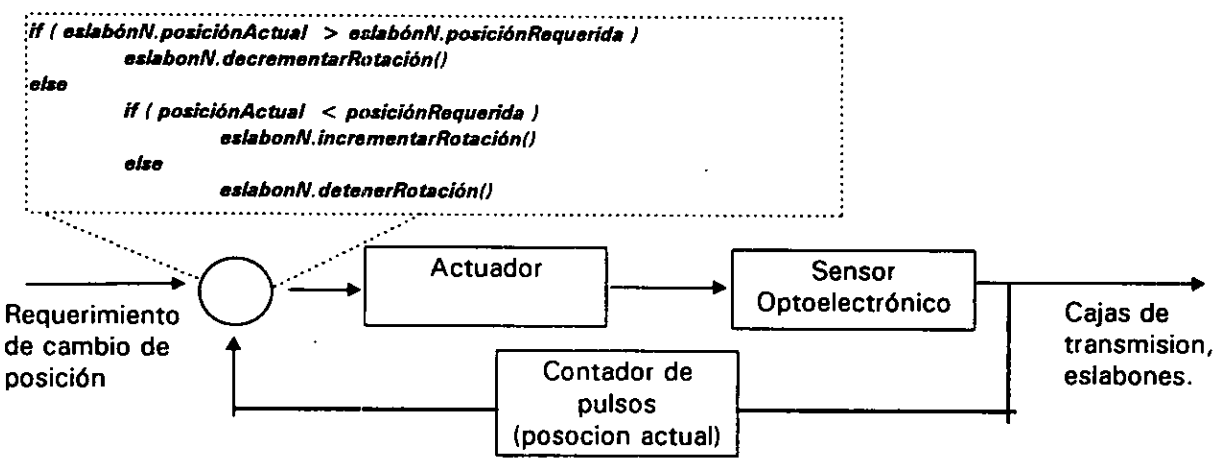


Ilustración 70. Control de posición de las articulaciones del robot.

Capítulo 6

Resultados y Conclusiones

En este capitulo se presentan los resultados obtenidos del desarrollo del proyecto, así como un breve análisis y comentarios del trabajo en general.

Interfaz electrónica

La interfaz electrónica permite controlar la posición de cada una de las partes móviles del robot. Las extremidades y el dispositivo de orientación de la cámara son tratados como dispositivos genéricos, cada uno tiene asignado un módulo electrónico idéntico. La comunicación con los módulos electrónicos se lleva a cabo a través de un circuito multiplexor, el cual permite el manejo de 56 líneas de entrada y 42 de salida utilizando sólo un par de puertos paralelos de la computadora. Este esquema de acceso tiene el inconveniente de ser 7 veces más lento que el acceso en forma directa a cada uno de los dispositivos, sin embargo en pruebas de realizadas utilizando una computadora pentium a 90 Mhz y un programa en C++, se detectó que puede hacerse lectura y escritura a los puertos paralelos a frecuencias de hasta 1 MHz. Considerando que la razón de cambio máxima de nuestros sensores es del orden de 30 Hz, es posible captar sus señales sin pérdida de información.

La separación de los circuitos en módulos independientes permite hacer cambios y mejoras en forma local, sin tener qué cambiar todo el circuito de la interfaz electrónica, por ejemplo:

- Manejar un número mayor número de dispositivos incrementando las líneas de acceso del circuito multiplexor.
- Alimentar actuadores mayor potencia sustituyendo la fase de potencia únicamente en los circuitos de dispositivo que así lo requieran.
- Utilizar sensores con diferentes propiedades eléctricas modificando las etapas del circuito de dispositivo para adecuar las señales a niveles TTL de la computadora.

Esta versatilidad resulta muy conveniente considerando que este circuito forma parte de las primeras propuestas para los sistemas del robot, por lo que debe admitir modificaciones en forma sencilla.

Implementación mecánica

La implementación de una de las extremidades permitió probar el desempeño de los mecanismos de transmisión y soporte, la interfaz electrónica y el software de control. Las prácticas consistieron en la inicialización del robot por medio de la computadora y el software de control; y el desplazamiento de la extremidad sobre una superficie regular, para lo cual la extremidad fue montada sobre una base móvil que permitiera su libre movimiento. En las siguientes ilustraciónes se muestra la extremidad en las ultimas fases de construcción:

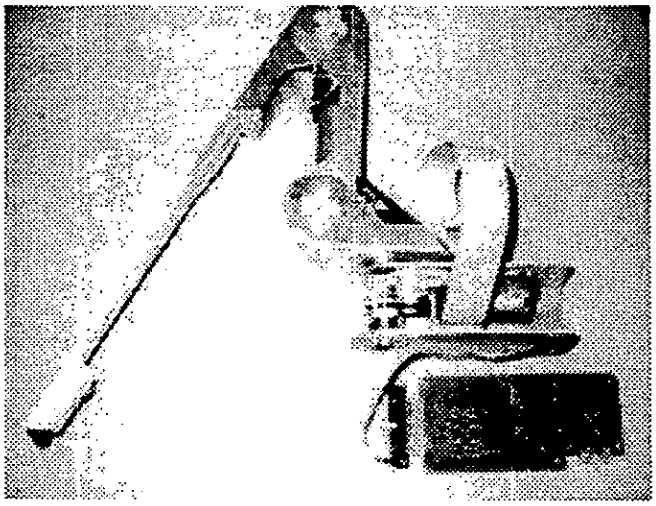


Ilustración 71. Vista lateral de la extremidad, en la parte inferior puede observarse el circuito de interfaz de este dispositivo.

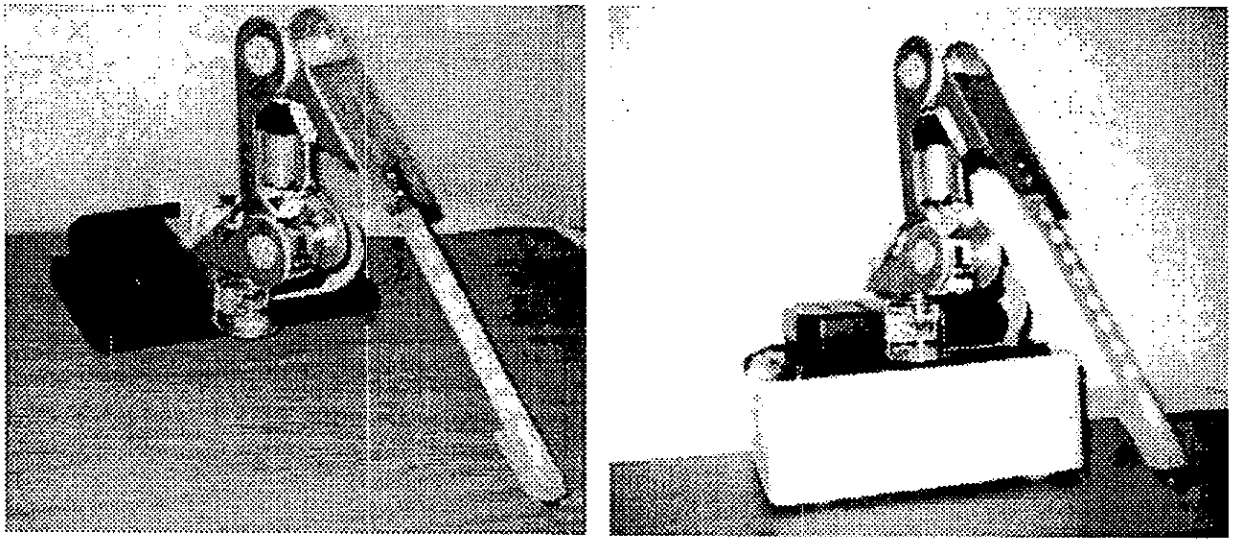


Ilustración 72. A la izquierda puede observarse la extremidad montada sobre una base fija para pruebas de movilidad. A la derecha la extremidad acoplada a una base móvil sobre la cual se efectuaron pruebas de desplazamiento.

El diseño de la extremidad cumple adecuadamente con su función de desplazamiento, sin embargo, durante las fases de implementación y prueba se hicieron algunas observaciones que pudieran mejorar su desempeño:

- Los sensores de límite basados en escobillas son muy susceptibles a dañarse durante la operación del robot, por lo que se procedió a sustituirlos por sensores de contacto.

- Resulta muy lenta y costosa la fabricación de mecanismos de transmisión que no tengan un error más allá de 1 grado, por lo que será recomendable el uso de componentes comerciales que más se ajusten a nuestras necesidades.

- No es posible el uso de contrapesos para facilitar al actuador el movimiento de los eslabones ya que esto incrementaría la carga sobre el robot. Como sustituto para contrarrestar el peso de cada eslabón se optó por el uso del resortes.

Java y VRML

El análisis, diseño y programación orientada a objetos son parte de una filosofía que concibe la abstracción de las características y funcionalidad de objetos y conceptos de la vida real a entidades lógicas. Este esquema de programación permite la conceptualización de procesos muy complejos como sistemas altamente modulares y relativamente sencillos de comprender.

Java es un lenguaje orientado a objetos cuyas características de conectividad y multiprocesamiento lo hacen ideal para la implementación de sistemas complejos. Por otro lado VRML es un lenguaje para la descripción de entidades virtuales, cuya funcionalidad y comportamiento puede ser implementado a través de objetos Java. La conjunción de ambos lenguajes permiten la visualización y simulación de fenómenos o procesos complejos a través del modelado de entidades lógicas, donde la parte funcional está separada de la interfaz gráfica siguiendo un esquema model-view-controller. Algunas observaciones hechas durante el estudio de Java y VRML se citan a continuación:

- **Facilidad de uso y aprendizaje.**
  VRML tiene una sintaxis sencilla, cuenta con un conjunto pequeño de estructuras sintácticas con las cuales es posible definir modelos funcionales. Una vez dominada la sintaxis de este lenguaje el programador puede extender su funcionalidad a través de objetos Java y la definición de prototipos.

- **Conectividad con otras aplicaciones.**
  El JDK implementa clases que soportan la comunicación entre procesos a través de sockets y pipes. El flujo de datos es manejado en forma transparente y se cuenta con la ayuda del manejo de excepciones para la captura de errores durante la ejecución de las aplicaciones.

- **Independiente de plataforma.**
  Al ser interpretados ambos lenguajes, el código puede ser transportado a cualquier plataforma donde exista un intérprete sin hacer cambio alguno al código.

- **Desempeño.**
  Desafortunadamente ambos lenguajes son demasiado jóvenes como para contar con implementaciones bien depuradas y con un alto desempeño, además del hecho de utilizar OOP, lo cual implica un compromiso entre la sencillez del código y su velocidad de ejecución. Sin embargo se ha visto una gran mejora desde las primeras versiones hasta la fecha en el aspecto de consumo de recursos, por lo que se espera pronto dejará de ser un inconveniente. Por otro lado poco a poco el hardware de alto desempeño gráfico para computadoras personales se ha hecho tan común que puede encontrarse en una tarjeta de video desde descompresores MPG hasta aceleradores para gráficos en tercera dimensión.

Simulación, visualización y control

El diseño del simulador consiste en una serie de entidades independientes estrechamente comunicadas, la delegación de responsabilidades y tareas entre cada una de ellas permite realizar procesos complejos bajo un esquema sencillo de participación. Cada entidad cuenta con una representación gráfica (geometría y apariencia física) y uno o más objetos que

controlan su comportamiento. Al ser independientes es posible añadir o modificar entidades como terrenos de prueba, tableros de control para el usuario o incluso diferentes robots móviles.
Los objetos de la clase ***ManejadorExtremidad*** que controlan el comportamiento de las extremidades utilizan el método geométrico para la obtención de la cinemática inversa, este es un método sencillo de utilizar en robots de 3 o menos grados de libertad, implica pocos cálculos y por ende poco tiempo de procesamiento. Desafortunadamente no es un método general, por lo que de requerirse la simulación de robots de diferentes arquitecturas deberán implementarse otros métodos como el de Newton-Raphson.

El uso de procesos auxiliares en equipo remoto es una buena solución cuando se debe realizar cálculos complejos que sobrecargarían la capacidad de la computadora local. El tiempo de procesamiento en equipo remoto asume un cierto retardo por la transferencia de los datos, por lo que se deberá tener cuidado en que este no exceda al tiempo de procesamiento de la computadora local. El desempeño de este esquema de trabajo depende de la elección de los procesos que deben migrar, para ello debe tomarse en cuenta la periodicidad con que son requeridos por el robot, la cantidad de información que requieren y la que generan. Tareas como el análisis de trayectorias a través del terreno y la generación de un plan de trabajo deberán ejecutarse remotamente, mientras que el control de los actuadores y el acopio de señales de los sensores deberán ser locales.

La simulación de procesos donde participan robots es de gran ayuda para la correcta proyección y depuración del plan de trabajo de cada uno de ellos, el usuario puede evaluar el comportamiento de cada entidad y efectuar los cambios convenientes. Esta tecnología está siendo desarrollada y aplicada por compañías como Deneb Robotics y Robot Simulations Ltd. en el ámbito industrial, sin embargo el costo de este software es muy elevado. La implementación de simuladores y visualizadores utilizando los esquemas de diseño aquí propuestos pueden alcanzar los mismos resultados sin requerir de grandes inversiones.

Evolución del proyecto

Los resultados mostrados a lo largo del presente trabajo conforman una base para el proyecto de implementación del robot móvil. Una vez terminada la especificación del robot se procederá a completar su modelo en el simulador, depurar los algoritmos de desplazamiento y en general realizar pruebas de comportamiento. En este punto podrá implementarse la comunicación con equipo de cómputo externo de acuerdo con las necesidades y carga del sistema.

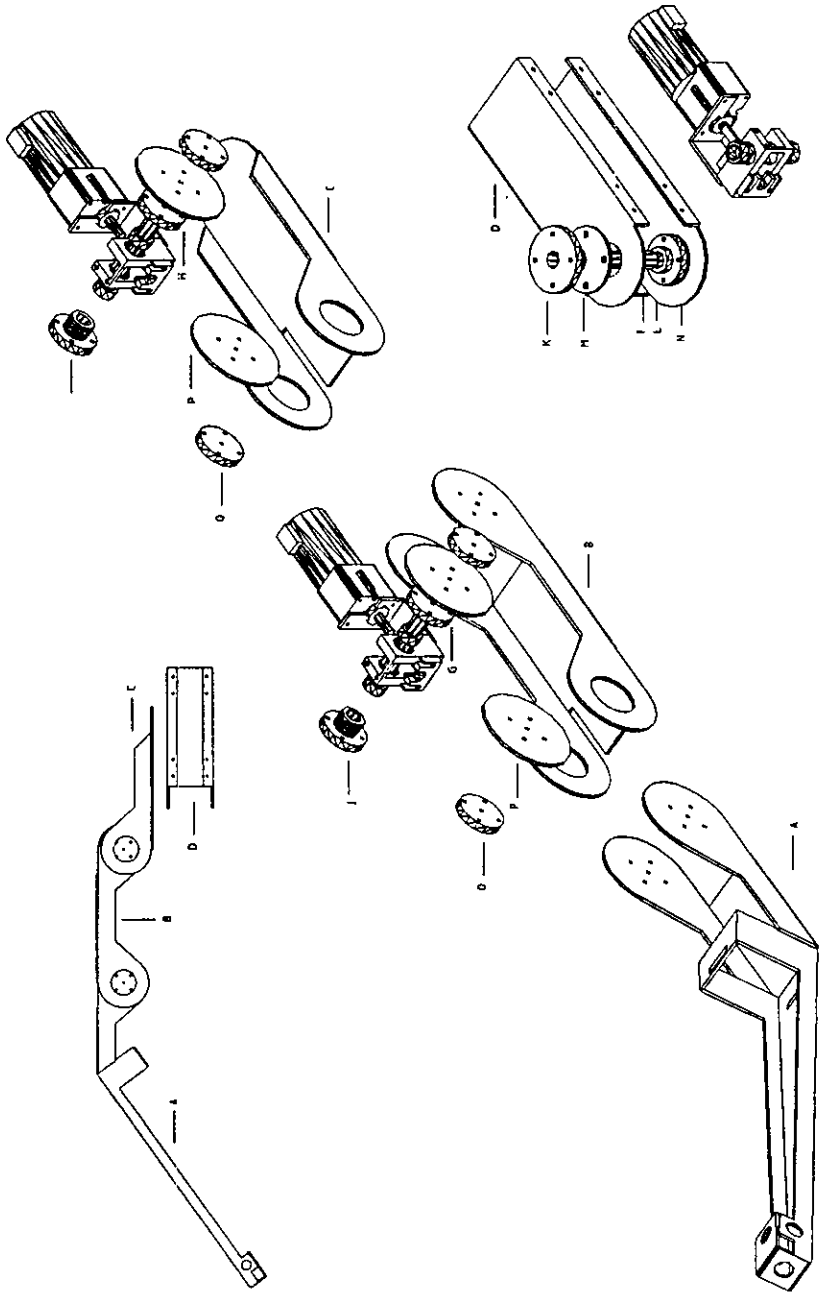
Se pretende que la interfaz gráfica del simulador se utilice como base para el sistema de comunicación con el usuario, de forma que este pueda controlar y observar a detalle los movimientos del robot aún sin estar frente a él. A su vez el controlador del robot implementará los algoritmos de desplazamiento obtenidos de las simulaciones, un protocolo de comunicación con la interfaz del usuario y otro para la comunicación con procesos auxiliares. La infraestructura de comunicación del robot así como el diseño modular del software de simulación y control, permitirá el uso de tecnologías como el procesamiento distribuido a través de Agentes y más adelante la implementación de inteligencia artificial.

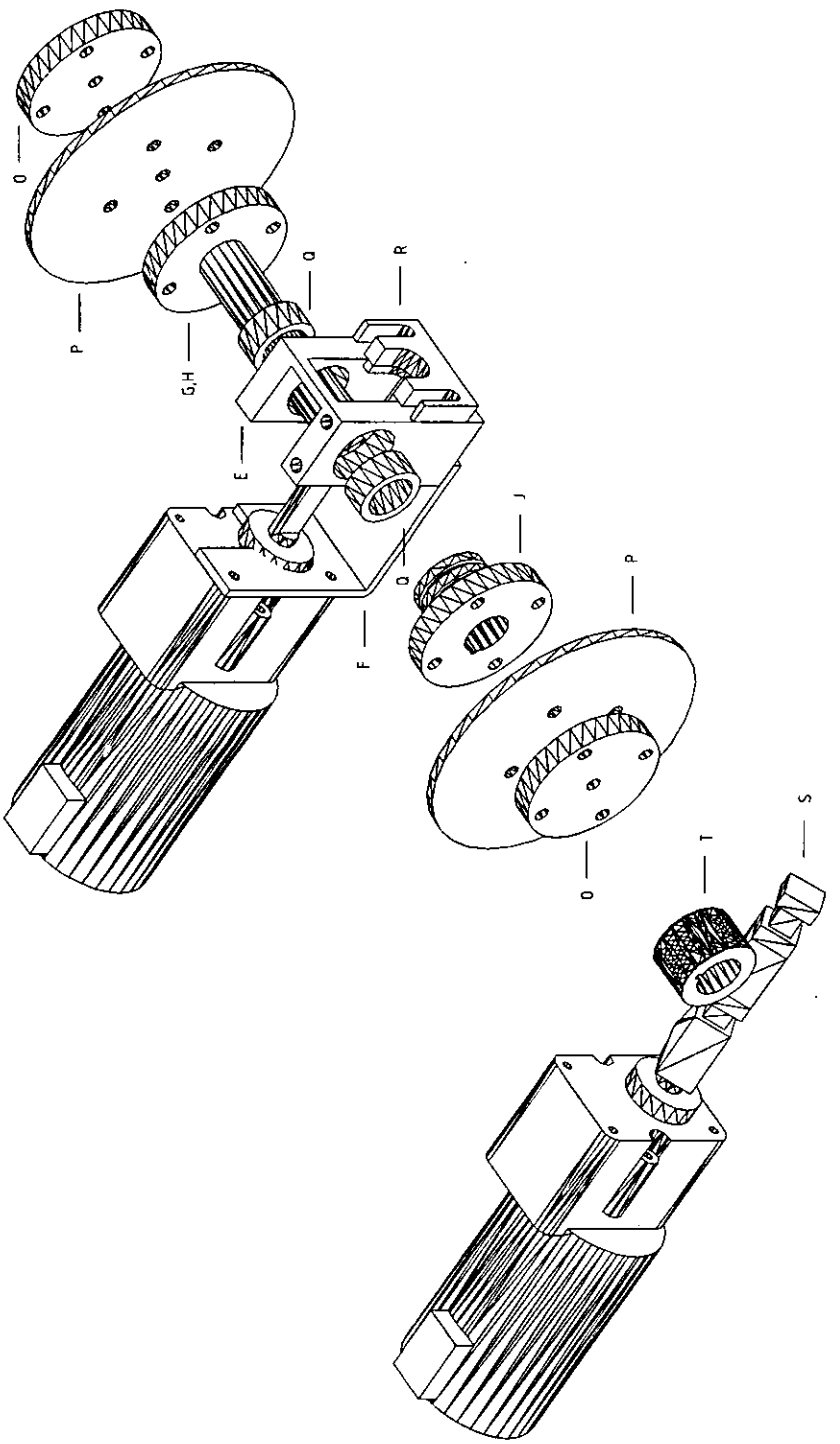
Apéndice A

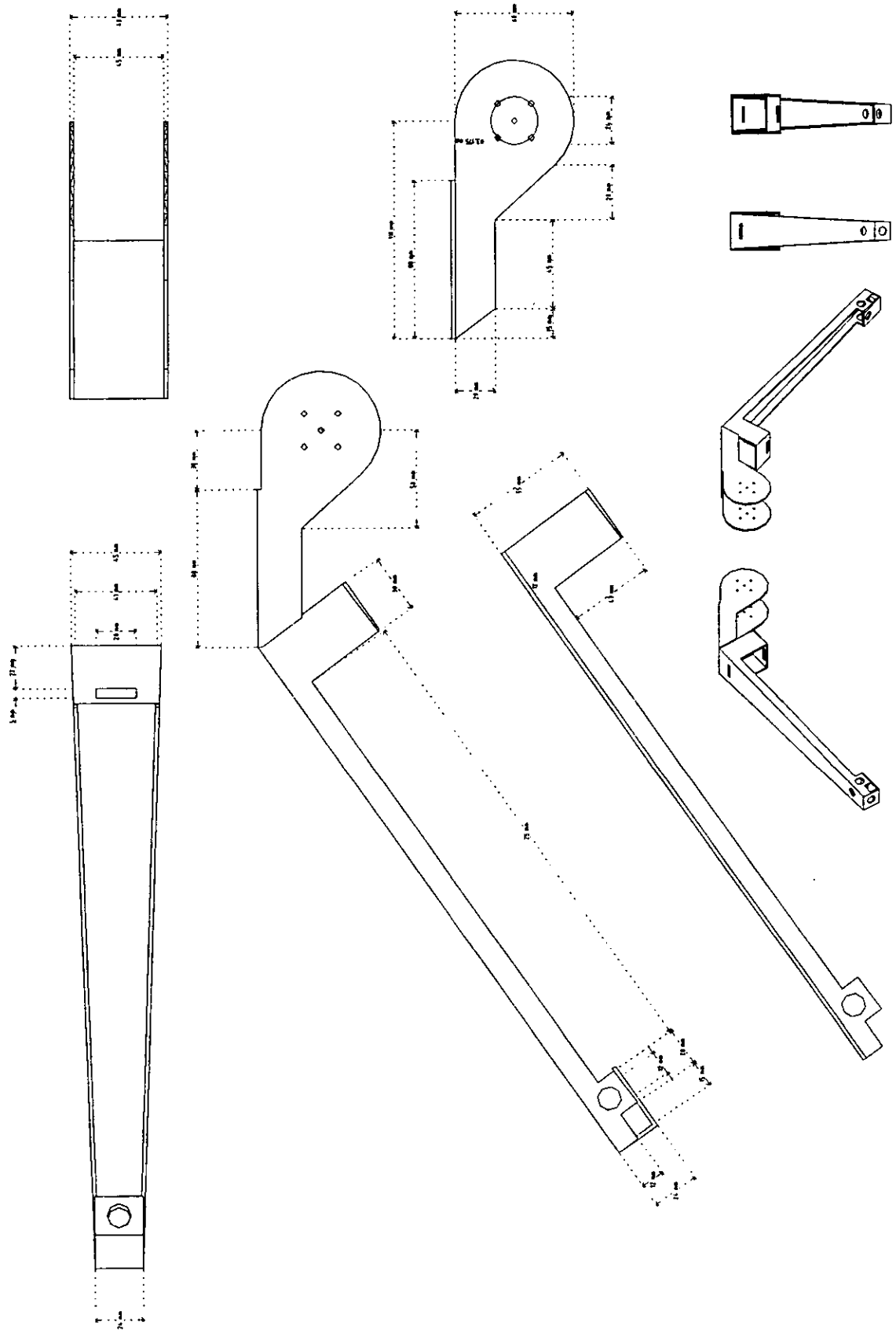
Planos y especificaciones del sistema mecánico

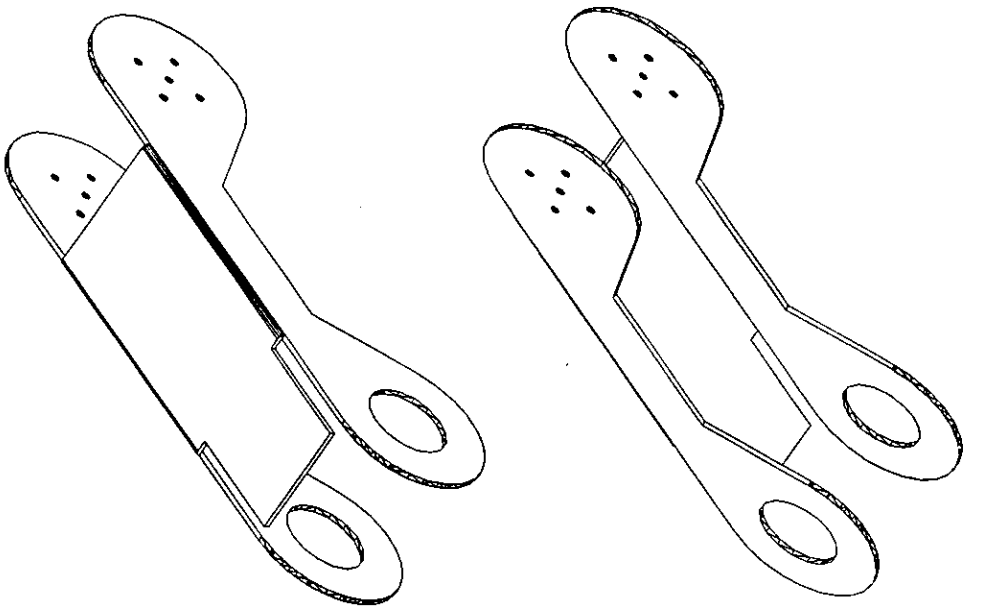
Índice General

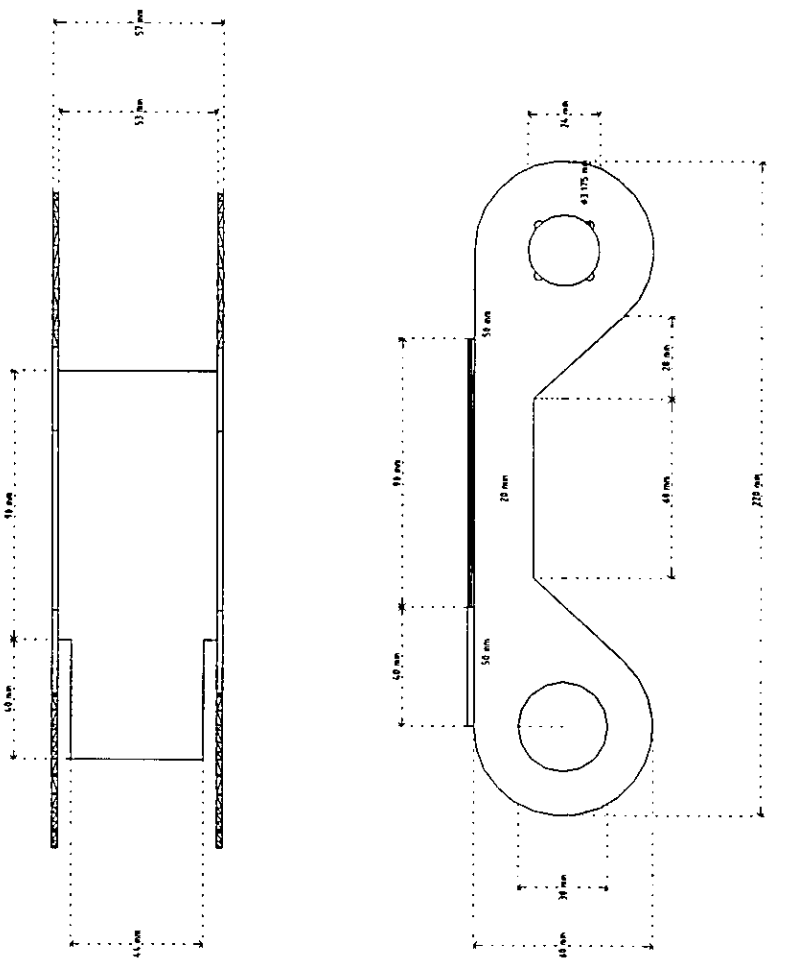
En los planos denominados como "detalle" se describe la localización de las piezas de acuerdo a la su letra de identificación.

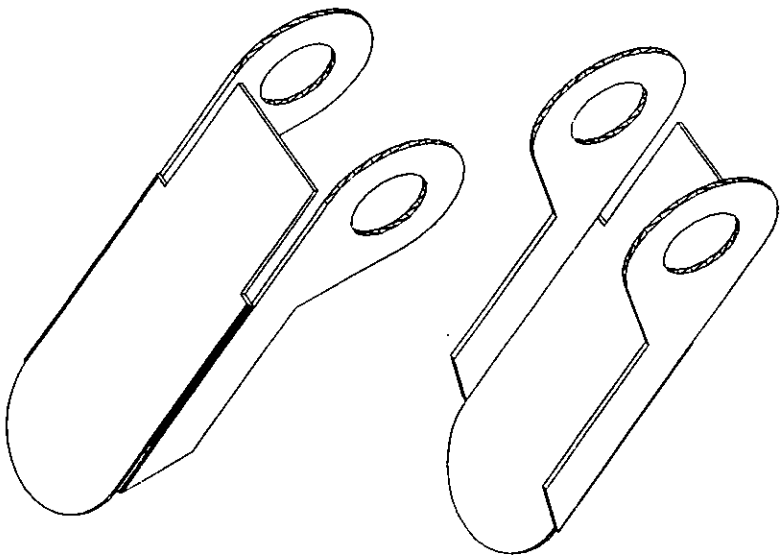


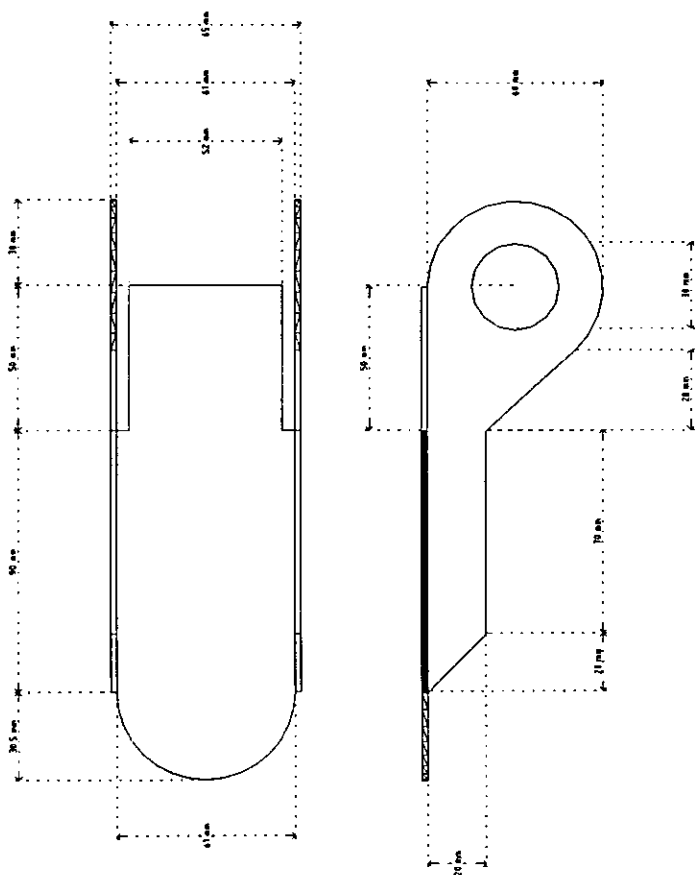

O
P
G,H
E
Q
R
F
O
J
P
O
T
S

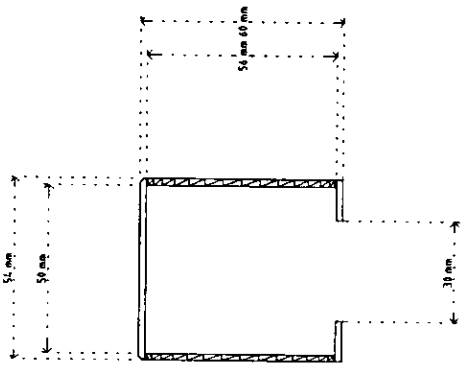


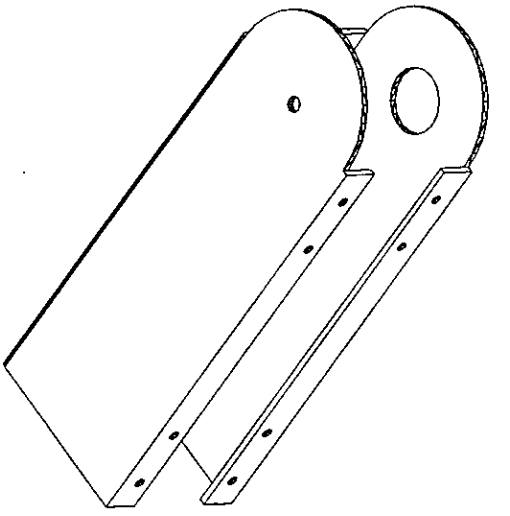


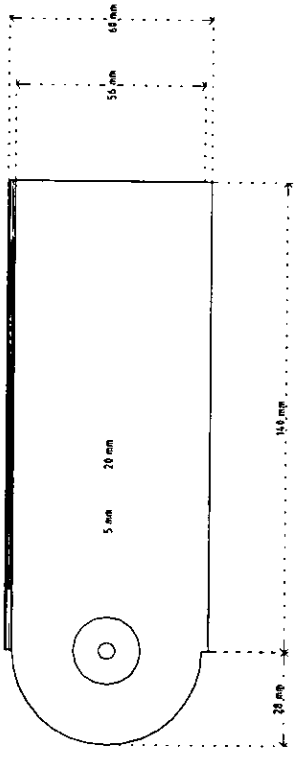


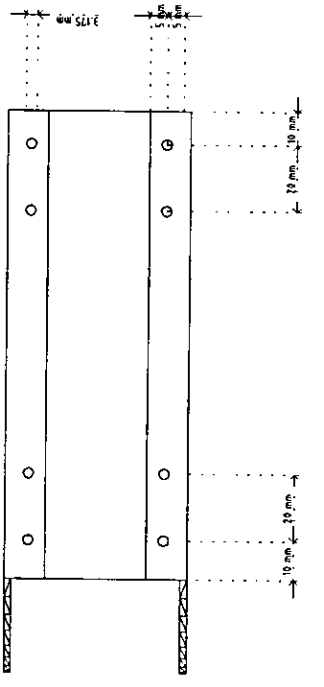


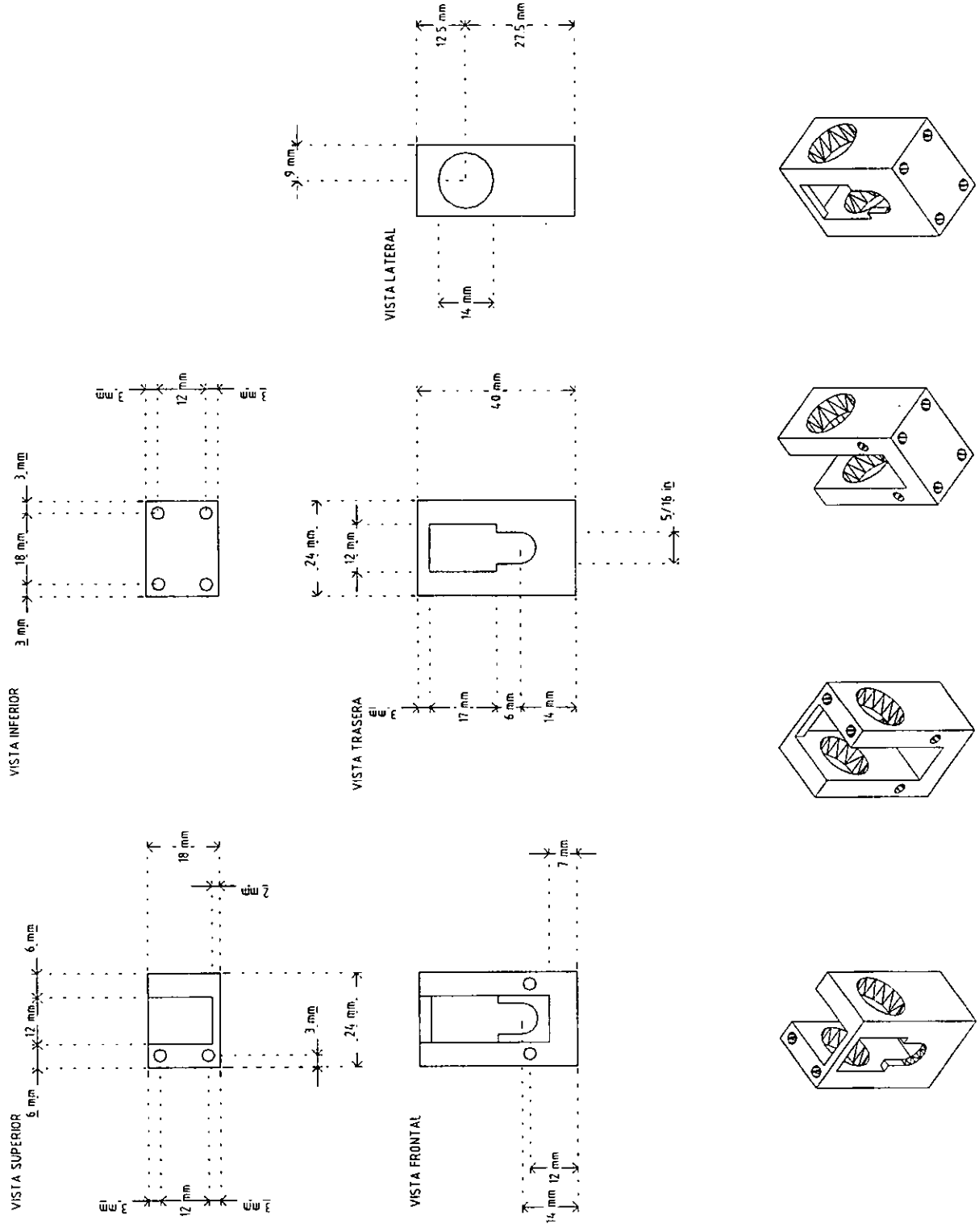


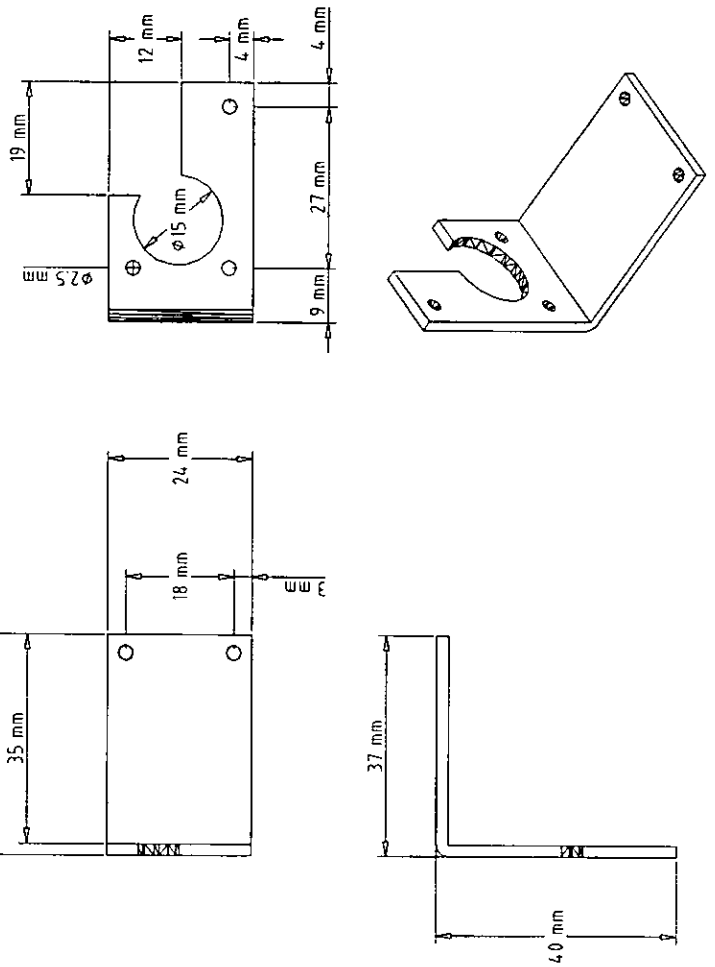

54 mm
50 mm
30 mm

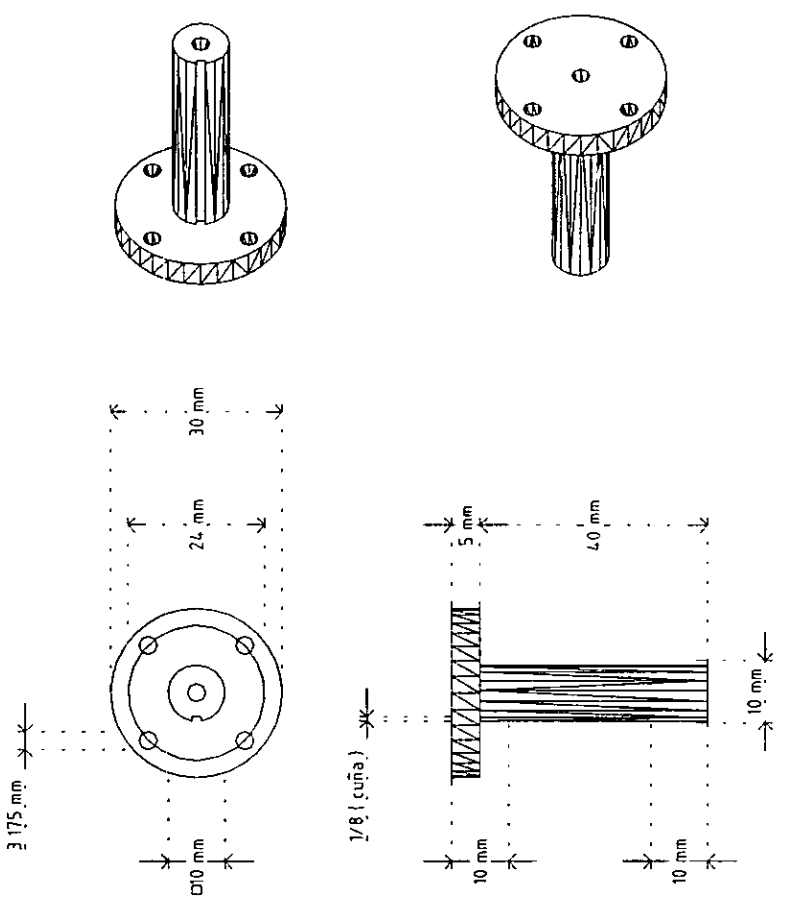


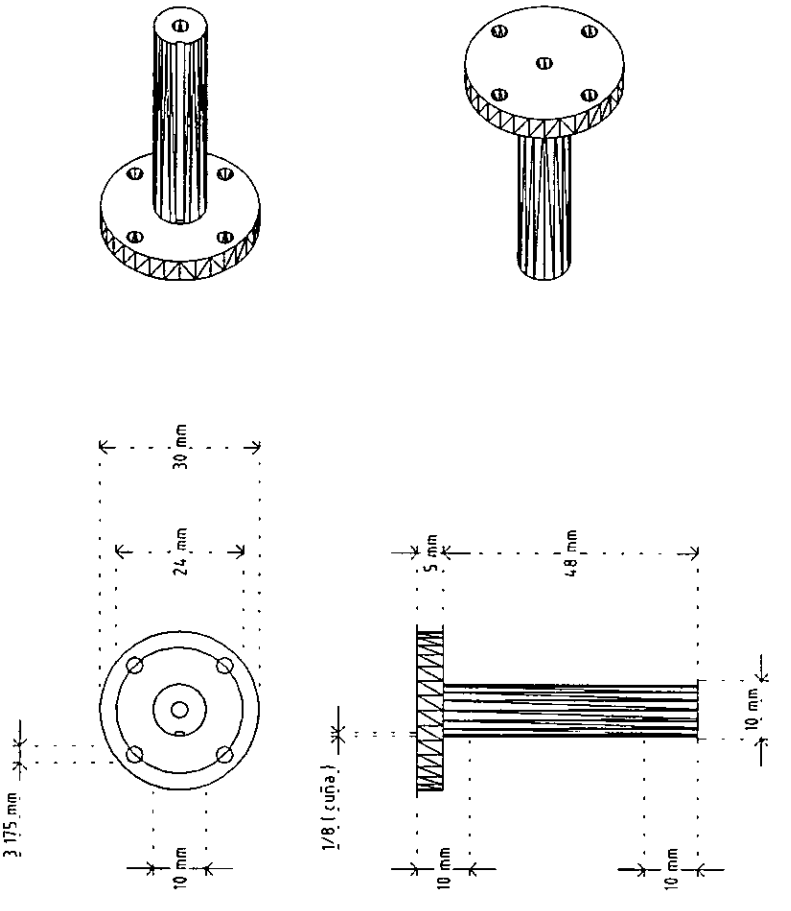

65 mm
55 mm
20 mm
5 mm
140 mm
20 mm

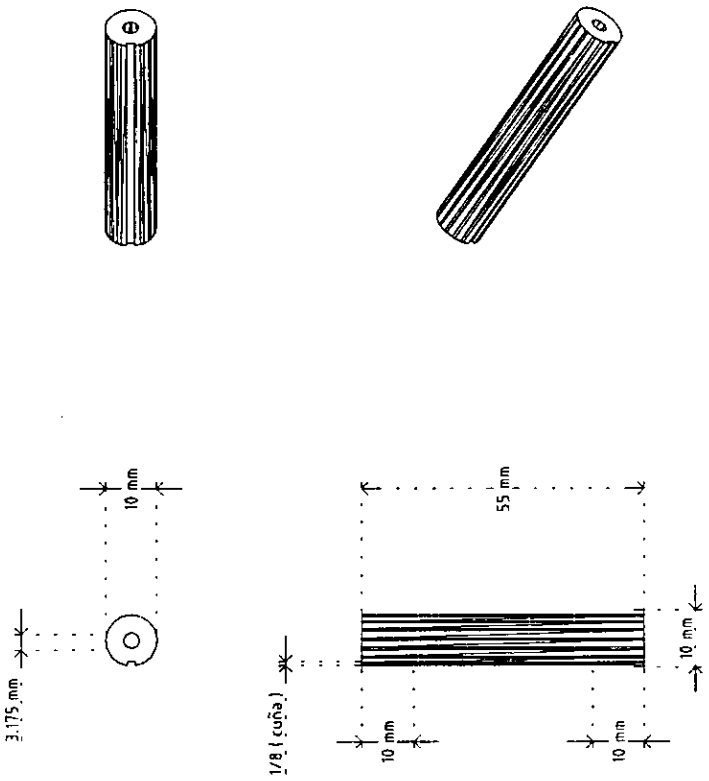

10 mm
10 mm
10 mm
10 mm

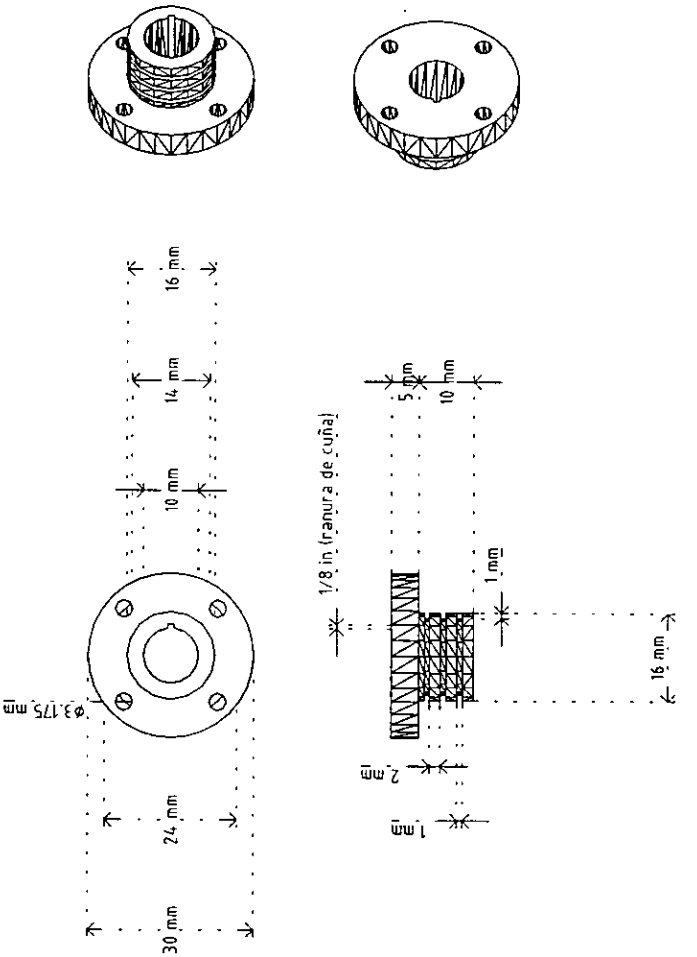

VISTA LATERAL
VISTA INFERIOR
VISTA TRASERA
VISTA SUPERIOR
VISTA FRONTAL

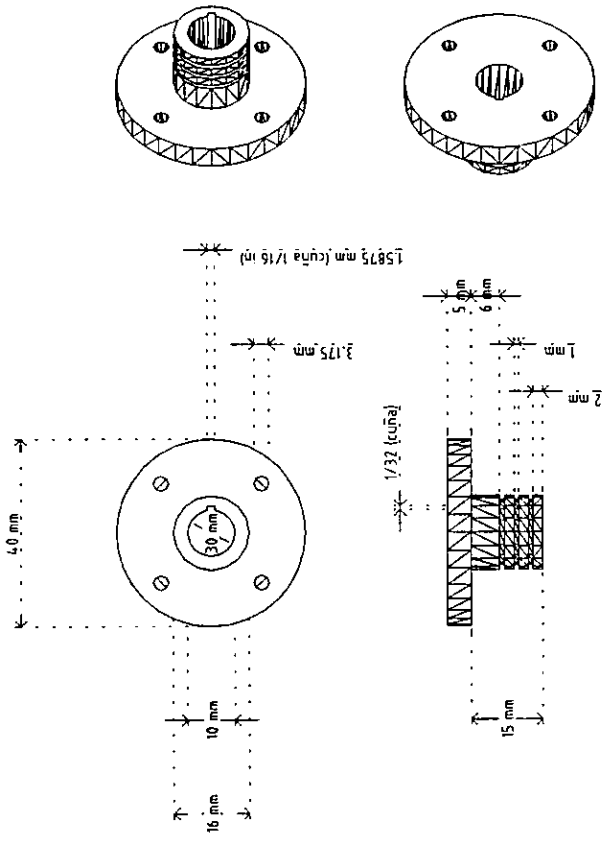

12 mm
4 mm
4 mm
19 mm
27 mm
ø15 mm
ø2,5 mm
9 mm
24 mm
18 mm
37 mm
35 mm
37 mm
40 mm

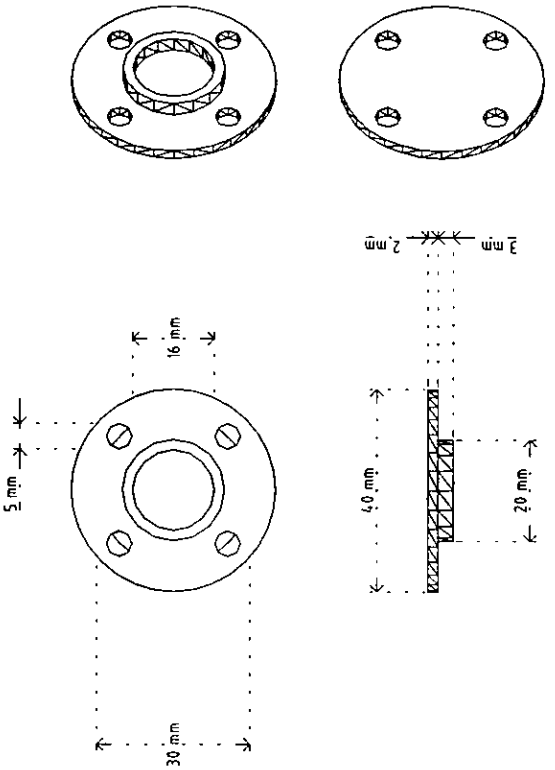

30 mm
24 mm
3.175 mm
Ø10 mm
1/8 (cuña)
5 mm
40 mm
10 mm
10 mm
10 mm

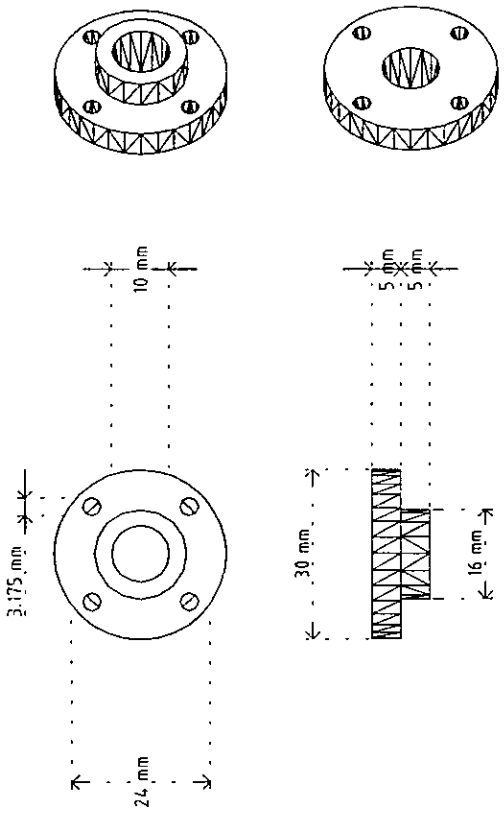

30 mm
24 mm
3.175 mm
10 mm
10 mm
5 mm
48 mm
1/8 (cuña)
10 mm
10 mm
10 mm

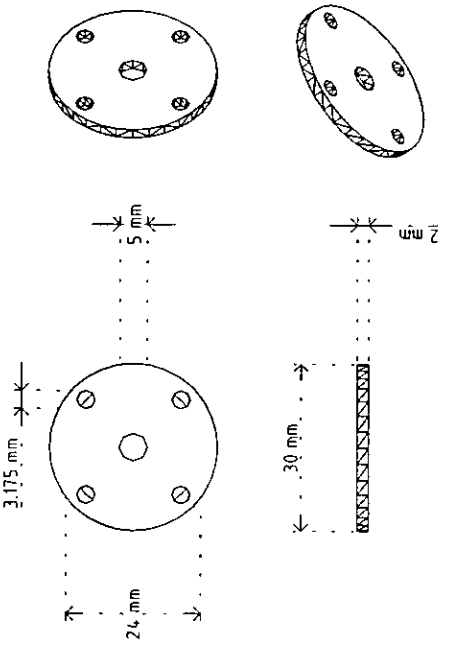

10 mm
3.175 mm
55 mm
1/8 (cuña)
10 mm
10 mm
10 mm

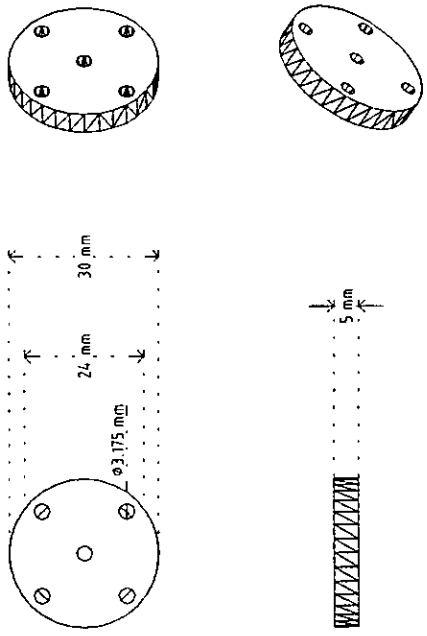

16 mm
14 mm
10 mm
ø3.175 mm
24 mm
30 mm
1/8 in (ranura de cuña)
5 mm
10 mm
1 mm
16 mm
2 mm
1 mm

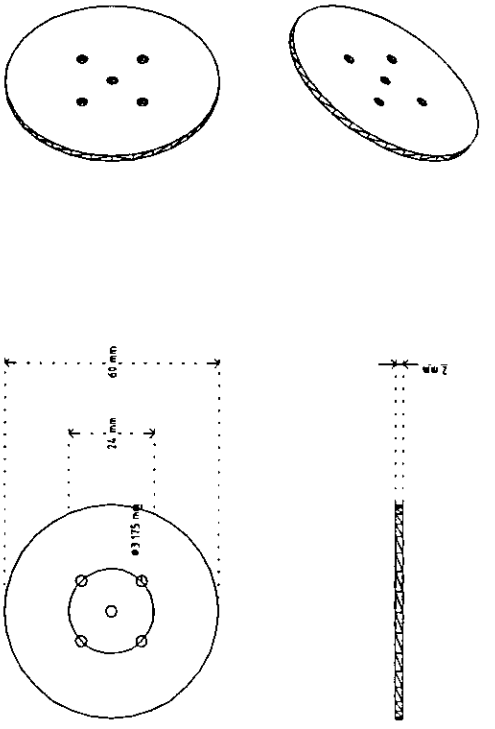

1,5875 mm (cuña 1/16 in)
3,175 mm
40 mm
30 mm
10 mm
16 mm
1/32 (cuña)
5 mm
6 mm
1 mm
2 mm
15 mm

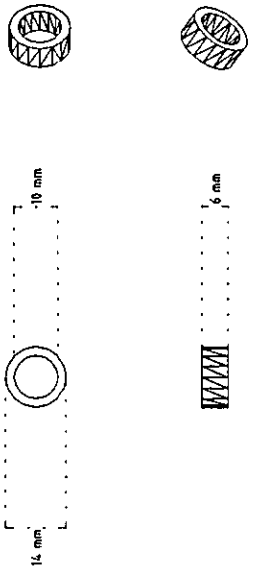

16 mm
5 mm
30 mm
40 mm
20 mm

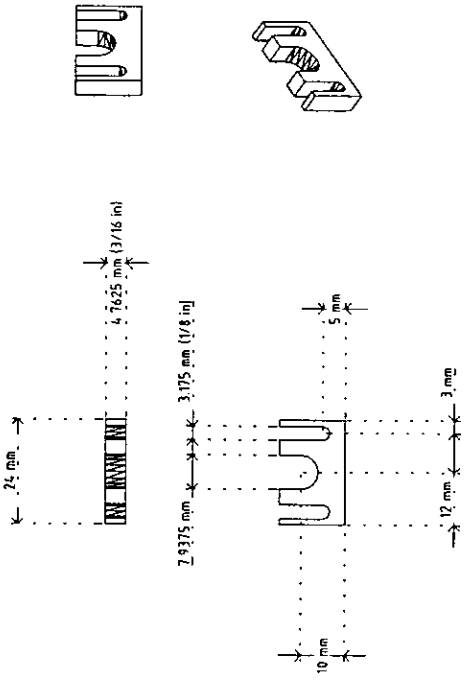

10 mm
5 mm
5 mm
3.175 mm
30 mm
16 mm
24 mm

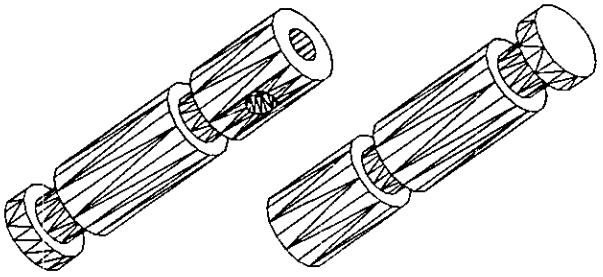

5 mm
2 mm
3.175 mm
30 mm
24 mm

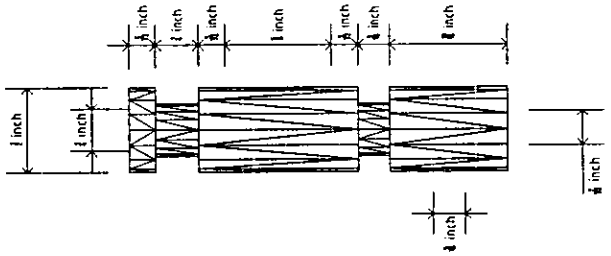

30 mm
24 mm
Ø3.175 mm
5 mm

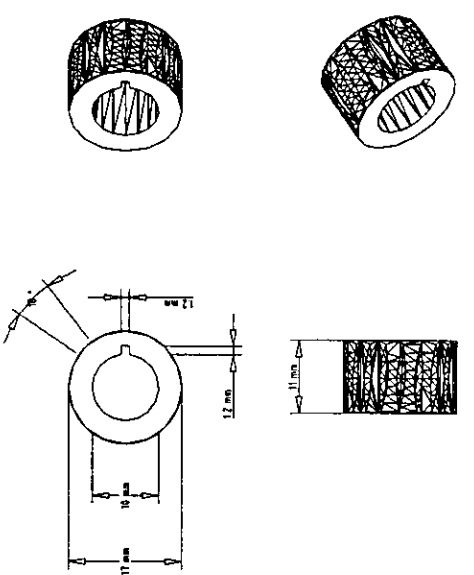

60 mm
24 mm
ø 3.175 mm


10 mm
6 mm
14 mm


24 mm
4,7625 mm (3/16 in)
3,175 mm (1/8 in)
2,9375 mm
5 mm
3 mm
12 mm
10 mm




1 inch
1 inch
1 inch
1 inch
1 inch
1 inch
2 inch
1 inch
1 inch
1 inch
1 inch



Apéndice B

Prototipos de las principales entidades del simulador

Declaración de prototipos

Extremidad

Cada extremidad es un robot de 3 grados de libertad, el cual responde a mensajes explícitos de cambio de posición y mantenerla ante cambios de posición del cuerpo del robot. De la misma forma es capaz de informar si ha terminado de efectuar un determinado cambio, o si este ha sido suspendido por algún evento como puede ser la colisión con el terreno.

Declaración

PROTO Extremidad [

field SFRotation rotacionYInicial 0 1 0 1
field SFVec3f posicionInicial 0 0 0

Campos para inicialización de posicon y orientación de la extremidad.

eventIn SFVec3f traslacionRobotIn
eventIn SFRotation rotXrobotIn
eventIn SFRotation rotYrobotIn
eventIn SFRotation rotZrobotIn

Los eventos de entrada informan a la extremidad que el robot se esta desplazando u orientando, por lo que, si la extremidad pertenece al grupo de apoyo, el script debera calcular la nueva posicion de la extremidad, de tal forma que el robot pueda conservar el equilibrio.

eventIn SFFloat temporizador

Reloj de sincronizacion con el robot.

eventIn SFVec3f traslacionExtremoIn
eventIn SFInt32 accionADesarrollar

Comandos que recibe la extremidad de su grupo, ya sea en forma decodificada en un entero, o bien directa a travez de la posicion que debe tomar la extremidad.

eventOut SFBool colisionOut
eventOut SFBool accionFinalizadaOut
eventOut SFVec3f traslacionExtremoOut

Eventos generados por la extremidad y que serviran a otras entidades para conocer su estado actual, como puede ser si está en contacto con el terreno, o bien si ha terminado la úñtma tarea encomendada.

```
        exposedField SFRotation  roteslabon1f 1 0 0 0
        exposedField SFRotation  roteslabon2f 1 0 0 0
        exposedField SFRotation  roteslabon3f 1 0 0 0
    ]
```

Los exposedField tiene las caracteristicas de campo, evento de salida y evento de entrada, en este caso fueron utilizadas para manipular la geometria del prototipo sin necesidad de recurrir al script.

Interfaz

```
DEF manejadorExtremidad Script
        {
            url "manejadorExtremidad.class"

            field SFVec3f posicionInicial  IS posicionInicial
            field SFRotation rotacionYInicial IS rotacionYInicial
            eventIn  SFRotation rotacionXRobotIn IS rotXrobotIn
            eventIn  SFRotation rotacionYRobotIn IS rotYrobotIn
            eventIn  SFRotation rotacionZRobotIn IS rotZrobotIn
            eventIn SFVec3f  traslacionExtremoIn IS traslacionExtremoIn
            eventIn SFVec3f traslacionRobotIn IS traslacionRobotIn
            eventOut SFBool  colisionOut IS colisionOut
            eventOut SFBool  accionFinalizadaOut IS accionFinalizadaOut
            eventIn SFInt32 accionADesarrollar  IS accionADesarrollar
            eventIn SFFloat temporizador IS temporizador
            eventOut SFVec3f traslacionExtremoOut IS traslacionExtremoOut
```

Referencias a los campos especificados en las declaraciones.

```
            eventOut SFVec3f  xyzExtremoOut
            eventOut SFRotation rot1Out
            eventOut SFRotation rot2Out
            eventOut SFRotation rot3Out
        }
```

Eventos de salida con los resultados de la cinemática directa e inversa, siendo estos últimos utilizados para controlar la posición de la extremidad.

```
ROUTE manejadorExtremidad.rot1Out  TO Eslabon_R1.set_rotation
ROUTE manejadorExtremidad.rot2Out  TO Eslabon_R2.set_rotation
ROUTE manejadorExtremidad.rot3Out  TO Eslabon_R3.set_rotation
```

Grupo

Un grupo es una entidad que coordina el movimiento de 3 extremidades, las cuales están dispuestas en forma alterna al rededor del cuerpo del robot, el robot cuenta con 2 grupos, de tal forma que cada uno de ellos se alterna en ejecutar 2 funciones básicas, la de soporte, y la de desplazamiento.

Declaración

```
PROTO grupo [
        eventOut SFVec3f traslacionExtremoExtremidadFrontal
        eventOut SFVec3f traslacionExtremoExtremidadMedia
        eventOut SFVec3f traslacionExtremoExtremidadTrasera
```

Control de las extremidades a través de una posición explícita que este debe adoptar.

```
        eventOut SFInt32 accionExtremidadFrontalOut
        eventOut SFInt32 accionExtremidadMediaOut
        eventOut SFInt32 accionExtremidadTraseraOut
```

Control de las extremidades a través de la emisión de comandos.

```
        eventIn SFBool extremidadFrontalActividadFinalizada
        eventIn SFBool extremidadMediaActividadFinalizada
        eventIn SFBool extremidadTraseraActividadFinalizada
```

Estatus de las extremidades.

```
        eventIn  SFInt32 accionIn
        eventOut  SFBool terminoAccionOut
```

Comunicación con el cuerpo del robot, recibe comandos y emite señales de estatus.

```
        eventIn SFFloat temporizador
```

Reloj de sincronizacion con el robot.

```
        eventIn SFBool obstaculoExtremidadFrontal
        eventIn SFBool obstaculoExtremidadMedia
        eventIn SFBool obstaculoExtremidadTrasera
```

Señal de obstáculo emitida por cada una de las extremidades.

```
]
```

Interfaz

```
DEF manejadorGrupo Script {
   url "manejadorGrupo.class"

        field SFFloat deltaTraslacion IS deltaTraslacion
        eventOut SFVec3f traslacionExtremoExtremidadFrontal IS
                                        traslacionExtremoExtremidadFrontal
        eventOut SFVec3f traslacionExtremoExtremidadMedia  IS
                                        traslacionExtremoExtremidadMedia
        eventOut SFVec3f traslacionExtremoExtremidadTrasera IS
                                        traslacionExtremoExtremidadTrasera
```

```
    eventOut  SFBool terminoAccionOut  IS terminoAccionOut
    eventIn  SFInt32 accionIn IS accionIn
    eventIn SFFloat temporizador  IS   temporizador
    eventIn SFBool obstaculoExtremidadFrontal IS obstaculoExtremidadFrontal
    eventIn SFBool obstaculoExtremidadMedia IS obstaculoExtremidadMedia
    eventIn SFBool obstaculoExtremidadTrasera IS obstaculoExtremidadTrasera
    eventIn SFBool extremidadFrontalActividadFinalizada  IS
                                                  extremidadFrontalActividadFinalizada
    eventIn SFBool extremidadMediaActividadFinalizada IS
                                                  extremidadMediaActividadFinalizada
    eventIn SFBool extremidadTraseraActividadFinalizada  IS
                                                  extremidadTraseraActividadFinalizada
    eventOut SFInt32 accionExtremidadFrontalOut IS accionExtremidadFrontalOut
    eventOut SFInt32 accionExtremidadMediaOut IS accionExtremidadMediaOut
    eventOut SFInt32 accionExtremidadTraseraOut IS accionExtremidadTraseraOut
}
```

Referencias a los campos especificados en las declaraciones.

Robot

Esta entidad está compuesta básicamente por 2 Grupos, mediante los cuales se sintetiza el control de las 6 extremidades.Responde a comandos de desplazamiento u orientación ya sea del cuerpo del propio robot o bien del dispositivo de orientación de la cámara. Sincroniza las acciones de sus dependencias a través de una señal de reloj.

Declaración

PROTO cuerpoRobot[

```
eventIn SFBool adelanteRobot
eventIn SFBool atrasRobot
eventIn SFBool izquierdaRobot
eventIn SFBool derechaRobot
eventIn SFBool detenerRobot
eventIn SFBool arribaCamara
eventIn SFBool abajoCamara
eventIn SFBool izquierdaCamara
eventIn SFBool derechaCamara
eventIn SFBool detenerCamara
eventIn SFFloat alturaIn
eventIn SFRotation orientacionCamara
```

Comandos para la orientación u orientación del cuerpo del robot y del dispositivo de orientación de la cámara.

```
eventIn SFFloat temporizador
```

Reloj de sincronizacion

```
field SFFloat deltaRotacion 0.02
field SFFloat deltaTraslacion  0.02
field SFVec3f escala 0.001 0.001 0.001
field SFVec3f dimensiones 2 0.2 1
field  SFVec3f posicionInicial 0 1 0
field  SFRotation rotacionXInicial 1 0 0 0
field  SFRotation rotacionYInicial 0 1 0 0
field  SFRotation rotacionZInicial 0 0 1 0
```

Campos para la inicialización de parámetros.

```
field SFRotation  rotacionYInicialE1 0 1 0 1.57
field SFVec3f  posicionInicialE1  0 0 0
field SFRotation  rotacionYInicialE2 0 1 0 1.57
field SFVec3f  posicionInicialE2  0 0 0
field SFRotation  rotacionYInicialE3 0 1 0 1.57
field SFVec3f  posicionInicialE3  0 0 0
field SFRotation  rotacionYInicialE4 0 1 0 1.57
field SFVec3f  posicionInicialE4  0 0 0
field SFRotation  rotacionYInicialE5 0 1 0 1.57
field SFVec3f  posicionInicialE5  0 0 0
field SFRotation  rotacionYInicialE6 0 1 0 1.57
field SFVec3f  posicionInicialE6  0 0 0
```

Rotación y traslación iniciales para cada una de las extremidades.

]

Interfaz

```
DEF manejadorCuerpoRobot Script {
   url "manejadorCuerpoRobot.class"

                    eventIn SFBool adelanteRobot IS adelanteRobot
                    eventIn SFBool atrasRobot IS atrasRobot
                    eventIn SFBool izquierdaRobot IS izquierdaRobot
                    eventIn SFBool derechaRobot IS derechaRobot
                    eventIn SFBool detenerRobot IS detenerRobot
                    eventIn SFFloat alturaIn  IS alturaIn
                    eventIn SFFloat temporizador IS temporizador
                    field SFFloat deltaRotacion IS deltaRotacion
                    field SFFloat deltaTraslacion IS deltaTraslacion
                    field  SFVec3f posicionInicial IS  posicionInicial
                    field  SFRotation rotacionXInicial IS rotacionXInicial
                    field  SFRotation rotacionYInicial IS rotacionYInicial
                    field  SFRotation rotacionZInicial IS rotacionZInicial
                    field  SFVec3f escala  IS escala
                    eventOut SFVec3f escalaOut

                    eventIn SFVec3f traslacionExtremo1In
                    eventIn SFVec3f traslacionExtremo2In
                    eventIn SFVec3f traslacionExtremo3In
                    eventIn SFVec3f traslacionExtremo4In
                    eventIn SFVec3f traslacionExtremo5In
                    eventIn SFVec3f traslacionExtremo6In
                    eventOut SFVec3f traslacionRobotOut
                    eventOut SFRotation rotacionYRobotOut
                    eventOut SFRotation rotacionXRobotOut
                    eventOut SFRotation rotacionZRobotOut
```

Comunicación con las extremidades, información sobre cambios de posición u orientación del cuerpoo del robot.

```
                    eventOut SFInt32 grupo1AccionOut
                    eventOut SFInt32 grupo2AccionOut
                    eventIn SFBool grupo1TerminoAccionIn
                    eventIn SFBool grupo2TerminoAccionIn
```

Comunicación con los Grupos, emisión de comandos y recepción de estatus.

```
}
```

Apéndice C

Código fuente del simulador, VRML2.0

Definición de prototipos.

Caminata.wrl

```
#VRML V2.0 utf8

EXTERNPROTO esl1 [ ]              #Eslabones de las extremidades
 "esl1.wrl#esl1"

EXTERNPROTO esl2 [ ]
 "esl2.wrl#esl2"

EXTERNPROTO esl2_cub [ ]                    #Cubiertas de los eslabones
 "esl2_cub.wrl#cubierta"

EXTERNPROTO esl3 [ ]
 "esl3.wrl#esl3"

EXTERNPROTO esl3_cub [ ]
 "esl3_cub.wrl#cubierta"

EXTERNPROTO esl4 [ ]
 "esl4.wrl#esl4"

EXTERNPROTO esl4_cub [ ]
 "esl4_cub.wrl#cubierta"

############################

EXTERNPROTO com [
            eventOut SFBool cAdelante
            eventOut SFBool cAtras
            eventOut SFBool cDerecha
            eventOut SFBool cIzquierda
            eventOut SFBool cDetener
            eventOut SFRotation orientacionCamara
            eventOut SFFloat altura

            eventOut SFBool  vistaInicial                    # Definicion de vistas basicas del robot
            eventOut SFBool  vistaFrontal
            eventOut SFBool  vistaTrasera
            eventOut SFBool  vistaLateralD
            eventOut SFBool  vistaLateralI
            eventOut SFBool  vistaSuperior
            eventOut SFBool  vistaExtremidadFrontal
            eventOut SFBool  vistaExtremidadLateral

            eventOut SFBool  vistaFrontalDerecha             # Definicion de vistas basicas del robot
            eventOut SFBool  vistaFrontalIzquierda
            eventOut SFBool  vistaTraseraDerecha
            eventOut SFBool  vistaTraseraIzquierda
            eventOut SFBool  vistaSobreExtremidadDelantera
            eventOut SFBool  vistaSobreExtremidadLateral

            ]
 "com.wrl#com"

############################

EXTERNPROTO comc [                    #Control manual del dispositivo de orientación
            eventOut SFBool cAbajo
            eventOut SFBool cArriba
            eventOut SFBool cDerecha
            eventOut SFBool cIzquierda
            eventOut SFBool cDetener

            ]
 "com.wrl#comc"

############################

EXTERNPROTO terreno [ ]                    #Terreno de prueba
 "terreno.wrl#terreno"

############################
```

```
#Definicion de la extremidad
PROTO Extremidad [
                        field SFRotation  rotacionYInicial  0 1 0 1
                        field SFVec3f  posicionInicial  0  0 0
                        eventIn SFVec3f traslacionRobotIn
                        eventIn SFVec3f  traslacionExtremoIn
                        eventIn SFVec3f  trans_ExtremoIn
                        exposedField SFRotation  roteslabon1f 1 0 0 0
                        exposedField SFRotation  roteslabon2f 1 0 0 0
                        exposedField SFRotation  roteslabon3f 1 0 0 0
                        eventIn SFRotation  rotXrobotIn
                        eventIn SFRotation  rotYrobotIn
                        eventIn SFRotation  rotZrobotIn
                        eventOut SFBool  colisionOut
                        eventOut SFBool  accionFinalizadaOut
                        eventIn SFInt32 accionADesarrollar
                        eventOut SFVec3f traslacionExtremoOut
                        eventIn SFFloat temporizador

                ]
{

DEF nodoEscala  Transform {
   children [
DEF nodoPosicionamiento    Transform {   #cond iniciales
                 center  0 0 0
                 translation  IS  posicionInicial
                 rotation  IS rotacionYInicial
                children [


             DEF Eslabon_4 Transform {
                translation  450 0 0

                children [ #children D
                                                          esl4{}
                                                          esl4_cub{}
                Transform { #cond iniciales
                                    center 0 0 0
                              translation  140 320 0
                children
                  DEF Eslabon_R3 Transform {
                  rotation IS roteslabon3f
                  center 200 0 0
                        children [ #children 3
                                                          esl3{}
                                                          esl3_cub{}
                                                            Transform {   #cond iniciales
                                                               center 0 0 0
                                                                        translation  685 10 0
                                                               #rotation 1 0 0 1.6
                                  children
                                          DEF Eslabon_R2  Transform {
                                                  translation 0 0 0
                                                  center -700 10 0
                                                   rotation IS roteslabon2f
                                                                        center -350 0 0
                                                                          children [
                                                                                  esl2_cub{}
                                                                                  Transform  {
                                                                                          translation 1110 -310 0
                                                                                          children
                                                                                            DEF Eslabon_R1 Transform{
                                                                                                    center -765 305 0
                                                                                             rotation IS roteslabon1f
                                                                                                    children [
                                                                                                            esl1{}
                                                                                                            ]

                                                                                          } #transform 1
                                                                                          }
                                                                                  ]
                                                                        }
                                                               }
                                                    }
                                            ]
                                    }
                            ]
                    }
                ]
            }
        ]
    }

DEF manejadorExtremidad Script {      #interfaz entre el cuerpo del robot y una extremidad
   url "manejadorExtremidad.class"
   field SFVec3f posicionInicial  IS posicionInicial
   field SFRotation rotacionYInicial IS rotacionYInicial
   eventIn  SFRotation rotacionXRobotIn IS rotXrobotIn
```

```
  eventIn  SFRotation rotacionYRobotIn IS rotYrobotIn
  eventIn  SFRotation rotacionZRobotIn IS rotZrobotIn
  eventIn SFVec3f  traslacionExtremoIn IS traslacionExtremoIn
  eventIn SFVec3f traslacionRobotIn IS traslacionRobotIn
  eventOut SFVec3f  xyzExtremoOut   #se produce cuando se cambian las rotaciones
  eventOut SFRotation rot1Out
  eventOut SFRotation rot2Out
  eventOut SFRotation rot3Out
  eventOut SFBool  colisionOut IS colisionOut
  eventOut SFBool  accionFinalizadaOut IS accionFinalizadaOut
  eventIn SFInt32 accionADesarrollar  IS accionADesarrollar
  eventIn SFFloat temporizador IS temporizador
  eventOut SFVec3f traslacionExtremoOut IS traslacionExtremoOut
},

ROUTE manejadorExtremidad.rot1Out  TO Eslabon_R1.set_rotation
ROUTE manejadorExtremidad.rot2Out  TO Eslabon_R2.set_rotation
ROUTE manejadorExtremidad.rot3Out  TO Eslabon_R3.set_rotation

} # Extremidad

############################

#Definición del prototipo de un grupo

PROTO grupo [
                        field SFFloat deltaTraslacion 0.1
                        eventOut SFVec3f traslacionExtremoExtremidadFrontal
                        eventOut SFVec3f traslacionExtremoExtremidadMedia
                        eventOut SFVec3f traslacionExtremoExtremidadTrasera
                        eventIn  SFInt32 accionIn
                        eventOut  SFBool terminoAccionOut
                        eventIn SFFloat temporizador
                        eventIn SFBool obstaculoExtremidadFrontal
                        eventIn SFBool obstaculoExtremidadMedia
                        eventIn SFBool obstaculoExtremidadTrasera
                        eventIn SFBool extremidadFrontalActividadFinalizada
                        eventIn SFBool extremidadMediaActividadFinalizada
                        eventIn SFBool extremidadTraseraActividadFinalizada
                        eventOut SFInt32 accionExtremidadFrontalOut
                        eventOut SFInt32 accionExtremidadMediaOut
                        eventOut SFInt32 accionExtremidadTraseraOut
                ]
{
DEF manejadorGrupo Script {
   url "manejadorGrupo.class"

                        field SFFloat deltaTraslacion IS deltaTraslacion
                        eventOut SFVec3f traslacionExtremoExtremidadFrontal IS traslacionExtremoExtremidadFrontal
                        eventOut SFVec3f traslacionExtremoExtremidadMedia  IS traslacionExtremoExtremidadMedia
                        eventOut SFVec3f traslacionExtremoExtremidadTrasera IS traslacionExtremoExtremidadTrasera
                        eventOut  SFBool terminoAccionOut  IS terminoAccionOut     #para informar que ha terminado
                        eventIn  SFInt32 accionIn IS accionIn
                        eventIn SFFloat temporizador  IS  temporizador
                        eventIn SFBool obstaculoExtremidadFrontal IS obstaculoExtremidadFrontal
                        eventIn SFBool obstaculoExtremidadMedia IS obstaculoExtremidadMedia
                        eventIn SFBool obstaculoExtremidadTrasera IS obstaculoExtremidadTrasera
                        eventIn SFBool extremidadFrontalActividadFinalizada          IS extremidadFrontalActividadFinalizada
                        eventIn SFBool extremidadMediaActividadFinalizada IS extremidadMediaActividadFinalizada
                        eventIn SFBool extremidadTraseraActividadFinalizada          IS extremidadTraseraActividadFinalizada
                        eventOut SFInt32 accionExtremidadFrontalOut IS accionExtremidadFrontalOut
                        eventOut SFInt32 accionExtremidadMediaOut IS accionExtremidadMediaOut
                        eventOut SFInt32 accionExtremidadTraseraOut IS accionExtremidadTraseraOut

                       }
} #Grupo


############################

#Definicion del prototipo del dispositivo de orientación
PROTO camara [
                        field SFFloat deltaRotacion          0.1
                        eventIn SFBool arribaCamara
                        eventIn SFBool abajoCamara
                        eventIn SFBool izquierdaCamara
                        eventIn SFBool derechaCamara
                        eventIn SFBool detenerCamara
                        eventIn SFRotation orientacionCamara
                        eventIn SFFloat temporizador
                ]
{
DEF derechaIzquierdaTransform Transform
{

 children
 [

   DEF arribaAbajoTransform Transform
   { set_rotation IS orientacionCamara
```

```
    children
      SpotLight {
                                location 0 1 0
                                direction 2 -1 0
                                color 1 1 1
                                beamWidth 0.3
                                cutOffAngle 0.6
                }
  }
]
}


DEF manejadorCamaraRobot Script {
   url "manejadorCamaraRobot.class"

   field SFFloat deltaRotacion IS deltaRotacion
   eventIn SFFloat temporizador IS temporizador              #Temporizador para simulacion de movimiento
   eventIn SFBool arribaCamara  IS arribaCamara              #Entradas de control
   eventIn SFBool abajoCamara  IS abajoCamara
   eventIn SFBool izquierdaCamara IS izquierdaCamara
   eventIn SFBool derechaCamara IS derechaCamara
   eventIn SFBool detenerCamara  IS  detenerCamara
   eventOut SFRotation rotacionAAOut                         #Salidas de control
   eventOut SFRotation rotacionDIOut

}
ROUTE manejadorCamaraRobot.rotacionAAOut  TO arribaAbajoTransform.set_rotation
ROUTE manejadorCamaraRobot.rotacionDIOut TO derechaIzquierdaTransform.set_rotation

} #Dispositivo de orientación

##############################

#Definicion del cuerpo del robot

PROTO cuerpoRobot[

                        eventIn SFBool adelanteRobot              #Controles del robot
                        eventIn SFBool atrasRobot
                        eventIn SFBool izquierdaRobot
                        eventIn SFBool derechaRobot
                        eventIn SFBool detenerRobot
                        eventIn SFBool arribaCamara               #Controles del dispositivo de orientacion
                        eventIn SFBool abajoCamara
                        eventIn SFBool izquierdaCamara
                        eventIn SFBool derechaCamara
                        eventIn SFBool detenerCamara
                        eventIn SFFloat alturaIn
                        eventIn SFFloat temporizador
                        field SFFloat deltaRotacion  0.05
                        field SFFloat deltaTraslacion  0.05
                        field SFVec3f escala 0.001 0.001 0.001
                        field SFVec3f dimensiones 2 0.2 1
                        field  SFVec3f posicionInicial 0 1 0
                        field  SFRotation rotacionXInicial 1 0 0 0
                        field  SFRotation rotacionYInicial 0 1 0 0
                        field  SFRotation rotacionZInicial 0 0 1 0
                        field SFRotation  rotacionYInicialE6 0 1 0 1.57
                        field SFVec3f  posicionInicialE6  0 0 0
                        field SFRotation  rotacionYInicialE2 0 1 0 1.57
                        field SFVec3f  posicionInicialE2  0 0 0
                        field SFRotation  rotacionYInicialE1 0 1 0 1.57
                        field SFVec3f  posicionInicialE1  0 0 0
                        field SFRotation  rotacionYInicialE5 0 1 0 1.57
                        field SFVec3f  posicionInicialE5  0 0 0
                        field SFRotation  rotacionYInicialE3 0 1 0 1.57
                        field SFVec3f  posicionInicialE3  0 0 0
                        field SFRotation  rotacionYInicialE4 0 1 0 1.57
                        field SFVec3f  posicionInicialE4  0 0 0
                        eventIn SFRotation orientacionCamara
                        eventIn SFBool activaVistaFrontal
                        eventIn SFInt32 numeroDeTerrenoIn
          ]
{

DEF cuerpoRobot Transform {
  scale IS escala
       children
       Transform {
         children [
              Transform{ scale 1000 1000 1000 #transformacion por bug, hgace visible el cuerpo del insecto
                   children[
                                Shape {
                                appearance Appearance {material Material { diffuseColor 1 1 1 }texture ImageTexture {url "water.gif" }
                                        }

                                geometry Box {size IS  dimensiones}
                                    }
```

```
                        DEF frontal Viewpoint {
                                set_bind IS activaVistaFrontal
                                position  6 2 0
                                orientation 0 1 0 1.57
                              }

                    ]
                }

                        DEF camara1 camara{
                                        arribaCamara IS  arribaCamara
                                        abajoCamara  IS abajoCamara
                                        izquierdaCamara  IS izquierdaCamara
                                        derechaCamara  IS derechaCamara
                                        detenerCamara  IS detenerCamara
                                        orientacionCamara IS orientacionCamara

                                      },

            DEF extremidad6 Extremidad{ rotacionYInicial IS rotacionYInicialE6
                                        posicionInicial IS posicionInicialE6
                                      }
             DEF extremidad1 Extremidad{ rotacionYInicial IS rotacionYInicialE1
                                        posicionInicial IS posicionInicialE1
                                      }
             DEF extremidad2 Extremidad{ rotacionYInicial IS rotacionYInicialE2
                                        posicionInicial IS posicionInicialE2
                                      }
             DEF extremidad5 Extremidad{ rotacionYInicial IS rotacionYInicialE5
                                        posicionInicial IS posicionInicialE6
                                      }
              DEF extremidad3 Extremidad{ rotacionYInicial IS rotacionYInicialE3
                                        posicionInicial IS posicionInicialE3
                                      }
             DEF extremidad4 Extremidad{ rotacionYInicial IS rotacionYInicialE4
                                        posicionInicial IS posicionInicialE4
                                      }
          ]
        },
},

DEF temporizador TimeSensor{
cycleInterval 1
 loop TRUE
},

#Script de control
DEF manejadorCuerpoRobot Script {
   url "manejadorCuerpoRobot.class"

   eventIn SFBool adelanteRobot IS adelanteRobot
   eventIn SFBool atrasRobot IS atrasRobot
   eventIn SFBool izquierdaRobot IS izquierdaRobot
   eventIn SFBool derechaRobot IS derechaRobot
   eventIn SFBool detenerRobot IS detenerRobot
   eventIn SFFloat alturaIn  IS alturaIn
   eventIn SFFloat temporizador IS temporizador
   field SFFloat deltaRotacion          IS deltaRotacion
   field SFFloat deltaTraslacion IS deltaTraslacion
   field  SFVec3f posicionInicial IS  posicionInicial rot
   field  SFRotation rotacionXInicial IS rotacionXInicial
   field  SFRotation rotacionYInicial IS rotacionYInicial
   field  SFRotation rotacionZInicial IS rotacionZInicial
   eventOut SFVec3f traslacionRobotOut
   eventOut SFRotation rotacionYRobotOut
   eventOut SFRotation rotacionXRobotOut
   eventOut SFRotation rotacionZRobotOut

   eventOut SFInt32 grupo1AccionOut
   eventOut SFInt32 grupo2AccionOut
   eventIn SFBool grupo1TerminoAccionIn
   eventIn SFBool grupo2TerminoAccionIn
   field  SFVec3f escala  IS escala
   eventOut SFVec3f escalaOut
   eventIn SFInt32 numeroDeTerrenoIn  IS   numeroDeTerrenoIn
   eventIn SFVec3f traslacionExtremo1In
   eventIn SFVec3f traslacionExtremo2In
   eventIn SFVec3f traslacionExtremo3In
   eventIn SFVec3f traslacionExtremo4In
   eventIn SFVec3f traslacionExtremo5In
   eventIn SFVec3f traslacionExtremo6In

}

DEF grupo1 grupo{ }                    #Grupos del robot
DEF grupo2 grupo{ }

ROUTE  manejadorCuerpoRobot.grupo1AccionOut TO grupo1.accionIn
ROUTE  manejadorCuerpoRobot.grupo2AccionOut TO grupo2.accionIn
ROUTE grupo1.terminoAccionOut TO manejadorCuerpoRobot.grupo1TerminoAccionIn
```

```
ROUTE grupo2.terminoAccionOut TO manejadorCuerpoRobot.grupo2TerminoAccionIn
ROUTE extremidad1.traslacionExtremoOut TO manejadorCuerpoRobot.traslacionExtremo1In
ROUTE extremidad2.traslacionExtremoOut TO manejadorCuerpoRobot.traslacionExtremo2In
ROUTE extremidad3.traslacionExtremoOut TO manejadorCuerpoRobot.traslacionExtremo3In
ROUTE extremidad4.traslacionExtremoOut TO manejadorCuerpoRobot.traslacionExtremo4In
ROUTE extremidad5.traslacionExtremoOut TO manejadorCuerpoRobot.traslacionExtremo5In
ROUTE extremidad6.traslacionExtremoOut TO manejadorCuerpoRobot.traslacionExtremo6In

#Rutas de control grupo2
ROUTE grupo2.accionExtremidadFrontalOut TO extremidad6.accionADesarrollar
ROUTE grupo2.accionExtremidadMediaOut TO extremidad2.accionADesarrollar
ROUTE grupo2.accionExtremidadTraseraOut TO extremidad4.accionADesarrollar
ROUTE extremidad6.accionFinalizadaOut TO grupo2.extremidadFrontalActividadFinalizada
ROUTE extremidad2.accionFinalizadaOut TO grupo2.extremidadMediaActividadFinalizada
ROUTE extremidad4.accionFinalizadaOut TO grupo2.extremidadTraseraActividadFinalizada
ROUTE extremidad6.colisionOut TO grupo2.obstaculoExtremidadFrontal
ROUTE extremidad2.colisionOut TO grupo2.obstaculoExtremidadMedia
ROUTE extremidad4.colisionOut TO grupo2.obstaculoExtremidadTrasera
ROUTE grupo1.accionExtremidadFrontalOut TO extremidad1.accionADesarrollar
ROUTE grupo1.accionExtremidadMediaOut TO extremidad5.accionADesarrollar
ROUTE grupo1.accionExtremidadTraseraOut TO extremidad3.accionADesarrollar

#Rutas de control grupo1
ROUTE extremidad1.accionFinalizadaOut TO grupo1.extremidadFrontalActividadFinalizada
ROUTE extremidad5.accionFinalizadaOut TO grupo1.extremidadMediaActividadFinalizada
ROUTE extremidad3.accionFinalizadaOut TO grupo1.extremidadTraseraActividadFinalizada
ROUTE extremidad1.colisionOut TO grupo2.obstaculoExtremidadFrontal
ROUTE extremidad5.colisionOut TO grupo2.obstaculoExtremidadMedia
ROUTE extremidad3.colisionOut TO grupo2.obstaculoExtremidadTrasera
ROUTE grupo1.accionExtremidadFrontalOut TO extremidad1.accionADesarrollar
ROUTE grupo1.accionExtremidadMediaOut TO extremidad5.accionADesarrollar
ROUTE grupo1.accionExtremidadTraseraOut TO extremidad3.accionADesarrollar

#Temporizadores
ROUTE temporizador.fraction_changed TO manejadorCuerpoRobot.temporizador        #Temporizadores hacia la cámara y los grupos
ROUTE temporizador.fraction_changed TO camara1.temporizador
ROUTE temporizador.fraction_changed TO grupo2.temporizador
ROUTE temporizador.fraction_changed TO grupo1.temporizador


ROUTE temporizador.fraction_changed TO extremidad1.temporizador                 #temporizadores hacia las extremidades
ROUTE temporizador.fraction_changed TO extremidad2.temporizador
ROUTE temporizador.fraction_changed TO extremidad3.temporizador
ROUTE temporizador.fraction_changed TO extremidad4.temporizador
ROUTE temporizador.fraction_changed TO extremidad5.temporizador
ROUTE temporizador.fraction_changed TO extremidad6.temporizador
ROUTE manejadorCuerpoRobot.escalaOut TO cuerpoRobot.set_scale
ROUTE manejadorCuerpoRobot.traslacionRobotOut TO cuerpoRobot.set_translation
ROUTE manejadorCuerpoRobot. rotacionYRobotOut TO cuerpoRobot.set_rotation
ROUTE manejadorCuerpoRobot.traslacionRobotOut TO extremidad6.traslacionRobotIn
ROUTE manejadorCuerpoRobot.rotacionYRobotOut TO extremidad6.rotYrobotIn
ROUTE manejadorCuerpoRobot.traslacionRobotOut TO extremidad1.traslacionRobotIn
ROUTE manejadorCuerpoRobot.rotacionYRobotOut TO extremidad1.rotYrobotIn
ROUTE manejadorCuerpoRobot.traslacionRobotOut TO extremidad5.traslacionRobotIn
ROUTE manejadorCuerpoRobot.rotacionYRobotOut TO extremidad5.rotYrobotIn
ROUTE manejadorCuerpoRobot.traslacionRobotOut TO extremidad2.traslacionRobotIn
ROUTE manejadorCuerpoRobot.rotacionYRobotOut TO extremidad2.rotYrobotIn
ROUTE manejadorCuerpoRobot.traslacionRobotOut TO extremidad3.traslacionRobotIn
ROUTE manejadorCuerpoRobot.rotacionYRobotOut TO extremidad3.rotYrobotIn
ROUTE manejadorCuerpoRobot.traslacionRobotOut TO extremidad4.traslacionRobotIn
ROUTE manejadorCuerpoRobot.rotacionYRobotOut TO extremidad4.rotYrobotIn

}#Cuerpo del robot

DEF robot cuerpoRobot{ rotacionYInicialE6 0 1 0  0#-.7854                    #Robot
                       posicionInicialE6 1000 0 700

                       rotacionYInicialE5 0 1 0 -1.57
                       posicionInicialE5 0 0 500

                       rotacionYInicialE4 0 1 0  3.1416#-2.36
                       posicionInicialE4 -1000 0 700

                       rotacionYInicialE1 0 1 0 0#.7854
                       posicionInicialE1 1000 0 -700

                       rotacionYInicialE2 0 1 0  1.57
                       posicionInicialE2 0 0 -500

                       rotacionYInicialE3 0 1 0 3.1416#2.36
                       posicionInicialE3 -1000 0 -700

                       posicionInicial 0 0.5 0
                       escala 0.001 0.001 0.001

        },

DEF controlManualRobot com{}                    #Tablero de control
```

```
ROUTE controlManualRobot.cAdelante TO robot.adelanteRobot
ROUTE controlManualRobot.cAtras TO robot.atrasRobot
ROUTE controlManualRobot.clzquierda TO robot.izquierdaRobot
ROUTE controlManualRobot.cDerecha TO robot.derechaRobot
ROUTE controlManualRobot.cDetener TO robot.detenerRobot
ROUTE controlManualRobot.orientacionCamara TO robot.orientacionCamara

terreno{}.                                          #Terreno de prueba

NavigationInfo { type "EXAMINE" }


DEF inicial Viewpoint {                             # Vista Principal
  position -3 4 8
  orientation 1 1 0 -0.5
}

DEF frontal Viewpoint {                             # Vistas perifericas
   position  6 0.3 0
   orientation 0 1 0 1.67
 }

DEF frontalDerecha Viewpoint {
   position  5 0.3 4
   orientation 0 1 0 0.8
}

DEF frontalIzquierda Viewpoint {
   position  5 0.3 -4
   orientation 0 1 0 2.37
}

DEF trasera Viewpoint {
   position  -6 0.3 0
   orientation 0 1 0 -1.57
 }

DEF traseraDerecha Viewpoint {
   position  -5 0.3 4
   orientation 0 1 0 -0.8
}

DEF traseralzquierda Viewpoint {
   position  -5 0.3 -4
   orientation 0 1 0 -2.37
}

DEF lateralD Viewpoint {
   position  0 0.3 6
   #orientation 0 1 0 1.57
 }

DEF lateralI Viewpoint {
   position  0 0.3 -6
   orientation 0 1 0 3.1416
 }

 DEF superior Viewpoint {
   position  -2 1.5 0
   orientation 0 1 0 -1.57
 }

ROUTE controlManualRobot.vistaInicial TO inicial.set_bind
ROUTE controlManualRobot.vistaFrontal TO frontal.set_bind
ROUTE controlManualRobot.vistaTrasera TO trasera.set_bind
ROUTE controlManualRobot.vistaLateralD TO lateralD.set_bind
ROUTE controlManualRobot.vistaLateralI TO lateralI.set_bind
ROUTE controlManualRobot.vistaSuperior TO superior.set_bind
ROUTE controlManualRobot.altura TO robot.alturaIn

ROUTE controlManualRobot.vistaFrontalDerecha  TO frontalDerecha.set_bind # Definicion de vistas basicas del robot
ROUTE controlManualRobot.vistaFrontalIzquierda TO  frontalIzquierda.set_bind
ROUTE controlManualRobot.vistaTraseraDerecha TO traseraDerecha.set_bind
ROUTE controlManualRobot.vistaTraseraIzquierda TO traseraIzquierda.set_bind
```

###############

terreno.wrl

#Definición del terreno de prueba

```
#VRML V2.0 utf8

PROTO terreno []
{

DEF ROOTNODE Transform {
translation -25 0 -25   #translacion para centrar el terreno
   children [
```

```
      Shape {
        geometry DEF malla ElevationGrid {
          xSpacing 1
          zSpacing 1
          xDimension 50   #/* dimensiones totales del terreno*/
          zDimension 50
          height [ 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1,
                1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1,
               1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1,
               .
               .
               .
                1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1,

              ]
          solid FALSE
        }
        appearance Appearance {
          material Material {
            diffuseColor 1 1 1
                    ambientIntensity 0
                                specularColor          0 0 0
                                emissiveColor          0 0.3 0
                                shininess     .5
                                transparency           0

                  }
          texture ImageTexture {
            url "water.gif"
          }
        }
      }
    ]
  }
DEF Terreno Script {
  url "Terreno.class"
  eventIn SFTime cycleTime
  eventIn SFFloat time_changed
  eventIn SFVec3f position_changed
  eventIn SFRotation orientation_changed
  field SFFloat escalaZ 0.00001        #escacala de la altura del terreno*/
  field SFInt32 xDimension 50          #dimensiones del terreno , se traslada el centro y se comienza en -dim/2*/
  field SFInt32 zDimension 50
  eventOut MFFloat height              #la topologia del terreno se construye valuando una funcion*/
 }

ROUTE Terreno.height TO malla.set_height

} #Terreno

terreno{}

###############
```

eslabonN .wrl

```
#Definición de los eslabones 1, 2,3 y 4, los cambios entre uno y otro radican únicamente en su geometría

#VRML V2.0 utf8

PROTO eslN [ ]
{
          Transform{
          center -755 305 0
          rotation 0 0 1 .41
          children
                              DEF shieldbody Transform {
                                   scale 4.5 4.5 4.5
                                   children    Shape {
                                    appearance  Appearance  {
                                               material Material {
                                                         ambientIntensity         0.254777
                                                         diffuseColor 0.0799639 0.813426 0.9125842
                                                         specularColor            0 0 0
                                                         emissiveColor            0 0 0
                                                         shininess     0.959184
                                                         transparency             0
                                                     }
                                               }
                                    geometry IndexedFaceSet {
                                     coord     Coordinate { #Vertices de la geometría
                                               point       [
                                  195.931535 -85.831297 0.000000,
                                  198.225843 -82.554686 -12.000000,
                                  -138.659899 99.607955 -23.265367,
                                  .
                                  .
                                  .
```

```
                                        -138.659899 99.174406 -23.914213,
                                        -138.659899 98.525554 -24.347758
                                                    ]
                                        }
                                        color     NULL
                                        coordIndex          [          #Relación entre vértices
                                        2, 1, 0, -1,
                                        1, 3, 0, -1,
                                        0, 4, 2, -1,
                                        .
                                        .
                                        .
                                        358, 377, 378, -1,
                                        358, 357, 377, -1
                                                                ]
                                        normal    NULL
                                        solid     FALSE
                                        creaseAngle         0
                                       }
                                      }
                                      center      0 0 0
                                     }
                                }
                        }
                }
} #EslabónV

eslV{}                  #Prueba del prototipo

###############
```

eslabonV .wrl

#Cubiertas de los eslabones 2,3 y 4, los cambios radican únicamente en la geometría y color de estos

```
#VRML V2.0 utf8

PROTO cubierta [ ]
{
                                        DEF shieldbody Transform {
                                         scale .79 .79 .79
                                         translation 140 -6 0
                                         children    Shape {
                                          appearance  Appearance  {
                                                      material Material {
                                                                  ambientIntensity        0.6
                                                                  diffuseColor 1 1 1
                                                                  specularColor           1 1 1
                                                                  emissiveColor           0.6 0.6 0.6
                                                                  shininess    0.9
                                                                  transparency            0
                                                      }
                                                     }

                                          geometry IndexedFaceSet {
                                           coord    Coordinate {
                                                    point       [
                                        268.472791 -158.261254 113.041170,
                                        42.390440 -158.261254 113.041170,
                                        42.390440 5.648443 113.041170,
                                        .
                                        .
                                        .
                                        -343.275696 -173.887655 148.919076,
                                        -341.321409 -175.841987 147.874475
                                                    ]
                                           }

                                           color    NULL

                                           coordIndex          [
                                        2, 1, 0, -1,
                                        3, 2, 0, -1,
                                        3, 4, 2, -1,
                                        .
                                        .
                                        .
                                        85, 86, 89, -1,
                                        86, 87, 89, -1
                                                                ]
                                           normal   NULL

                                           solid    FALSE
                                           creaseAngle         0
                                          }

                                         }
```

```
                                        center      0 0 0
                                       }
        }
}#Cubierta

  cubierta{}  #Prueba del prototipo

##############
```

com.wrl

```
#tablero de control

#VRML V2.0 utf8

EXTERNPROTO boton1 [   eventOut SFTime buttonResult      eventOut SFBool click ]          #Prototipos de los diferentes tipos de  botones
"boton1.wrl#boton1"

EXTERNPROTO boton [    eventOut SFTime buttonResult      SFBool click ]
 "boton.wrl#boton"

EXTERNPROTO boton2 [   eventOut SFTime buttonResult    eventOut SFBool click]
 "boton2.wrl#boton2"

EXTERNPROTO slider [   eventOut    SFFloat sliderResult ]
 "slider.wrl#sliderProto"


PROTO com [
            eventOut SFBool cAdelante                              #Control  del  robot
            eventOut SFBool cAtras
            eventOut SFBool cDerecha
            eventOut SFBool cIzquierda
            eventOut SFBool cDetener
            eventOut SFFloat altura
            eventOut SFRotation orientacionCamara                  #Control del dispositivo de orientación

            eventOut SFBool  vistaInicial                          # Control de las de vistas basicas del robot
            eventOut SFBool  vistaFrontal
            eventOut SFBool  vistaTrasera
            eventOut SFBool  vistaLateralD
            eventOut SFBool  vistaLateralI
            eventOut SFBool  vistaSuperior
            eventOut SFBool  vistaExtremidadFrontal
            eventOut SFBool  vistaExtremidadLateral
            eventOut SFBool  vistaFrontalDerecha
            eventOut SFBool  vistaFrontalIzquierda
            eventOut SFBool  vistaTraseraDerecha
            eventOut SFBool  vistaTraseraIzquierda
            eventOut SFBool  vistaSobreExtremidadDelantera
            eventOut SFBool  vistaSobreExtremidadLateral
          ]
{
          Transform {
           children    Group [
            children   Transform {
              children [
                      DEF prox ProximitySensor {
                       center       0 0 0
                       size         500 500 500
                      }

                      DEF xform Transform {
                       children[
                       ############
                      #Cotrol de la altura del robot
                       Transform{
                         scale 0.012 0.012 0.012
                         translation              0 -0.08 -.4 #-1.7
                         children [

                           Transform { translation 6 -1 0
                                              scale 5 5 5
                                                    children[
                                                                Group {children  slider {sliderResult IS altura} }
                                                    ]
                                             },
                      #Control del dispositivo de  orientación
                         Transform {  translation 13 3 0
                         children[
                          DEF esferaControl Transform {    #esfera control camara
                                    children      [
                                                    Shape {
                                                                appearance Appearance {material Material {
                                                                                                diffuseColor 1 0 1
                                                                                                transparency .4
                                                                                                emissiveColor              .5 0 .5
                                                                                                }
```

```
                                                                texture ImageTexture {url "water.gif" }
                                                                }
                                                geometry Sphere {  radius 1.6}
                                        }

            ]
  }
  DEF sensorEsfera SphereSensor {rotation_changed IS orientacionCamara}
  ]
 },
#Definicion de controles de las vistas
  Transform { translation -4 -2.5 0
          scale 1 5 1
                                        children[
                                                Group {children DEF inicial boton {click IS vistaInicial} }
                                        ]
                        },
  Transform { translation 0 1.5 0
          scale 5 1 1
                                        children[
                                                Group {children  boton {click IS vistaSobreExtremidadDelantera} }
                                        ]
                        },
  Transform { translation 4 -2.5 0
          scale 1 5 1
                                        children[
                                                Group {children  boton {click IS vistaSobreExtremidadLateral} }
                                        ]
                        },
  Transform {
                                        children[
                                                Group {children DEF frontal boton {click IS vistaFrontal} }
                                        ]
                        },
  Transform { translation 2.5 0 0
                                        children[
                                                Group {children DEF frontal boton {click IS vistaFrontalDerecha} }
                                        ]
                        },
  Transform { translation -2.5 0 0
                                        children[
                                                Group {children DEF frontal boton {click IS vistaFrontalIzquierda} }
                                        ]
                        },
  Transform { translation 0 -5 0
                                        children[
                                                Group {children DEF trasero boton {click IS vistaTrasera } }
                                        ]
                        },
  Transform { translation 2.5 -5 0
                                        children[
                                                Group {children DEF trasero boton {click IS vistaTraseraDerecha } }
                                        ]
                        },
  Transform { translation -2.5 -5 0
                                        children[
                                                Group {children DEF trasero boton {click IS vistaTraseraIzquierda } }
                                        ]
                        },

  Transform { translation 0 -2.5 0
                                        children[
                                                Group {children DEF arriba boton {click IS vistaSuperior} }
                                        ]
                        },

  Transform { translation 2.5 -2.5 0
                                        children[
                                                Group {children DEF lateralD boton {click IS vistaLateralD} }
                                        ]
                        },

  Transform { translation -2.6 -2.5 0
                                        children[
                                                Group {children DEF lateralI boton {click IS vistaLateralI} }
                                        ]
                        },
                      ]
          )
############
# Botones para el control del  robot
Transform {
children     Transform{
children     Transform{
children     Transform{
 children   [
  Transform {           #boton Detener
          children      [
                        Group {children DEF detener boton {click IS cDetener} }
```

```
                                    ]
                                translation   0 0 0
                                rotation 1 0 0 -.4
                                scale 1 1 1
                            },
                        Transform { children[
                            Transform {   #boton adelante
                                        children  [
                                          DEF adelante boton2 {click IS cAdelante}
                                        ]
                                rotation 1 0 0 1.5
                                translation   0 0 -2
                                scale 0.5 0.5 0.5
                                }
                            #rotation 1 0 0 1.5
                            ]
                            }
                            Transform { children
                            Transform {   #boton atras
                                        children  [
                                          DEF atras boton2 {click IS cAtras }
                                        ]
                                rotation 1 0 0 1.5
                                translation   0 0 -2 #0 0 -2
                                scale 0.5 0.5 0.5
                                }
                             rotation 1 0 0 3.14
                             }
                            ]
                            #rotation 0 0 1 0.6 #100 —  010 | 001 .
                            rotation 0 1 0  0.2 #0.4
                           }
                            #rotation 1 0 0 -01.0
                            rotation 1 0 0 0.4
                           }

                           }
                            #scale     0.1 0.1 0.1
                             scale 0.012 0.012 0.012
                             translation              0.07 -0.05 -.2 #-1.7
                           }
                           }
                           translation 0 0 0
                           }
                    }
                  }

                 }
                 rotation    0 0 1  0
               }
ROUTE sensorEsfera.rotation_changed TO esferaControl.set_rotation
ROUTE prox.position_changed TO prox.set_center
ROUTE prox.position_changed TO xform.set_translation
ROUTE prox.orientation_changed TO xform.set_rotation
}
Viewpoint {
 position     1.19 4.57 -0.812
 orientation -0.361 0.929 0.0787  0.229
 fieldOfView              0.785
 description "knight"
}

DEF controlManualRobot com{}                    #Prueba del prototipo

###############
```

boton.wrl

#Definición de los botones 1,2 y 3, los cambios entre ellos radican en su geometría y color de estos

```
#VRML V2.0 utf8

PROTO boton [
                        eventOut SFTime buttonResult      #Eventos a los cuales responde el prototipo
                        eventOut SFBool click
             ]
{
 Group {
  children  Transform {
   children [
            Group {
             children    DEF button Transform {
              children  [
               DEF manejadorBoton Script {                              #Script de control
                        eventOut     SFBool       activo IS click
                        eventIn      SFTime       startTime
                        url          "manejadorBoton.class"
               }
               DEF buttonSurface_1 Transform {
                        children     [
```

```
            DEF touchSensorTrigger_2 TouchSensor {}
            Group {
             children [
              DEF buttonAnim Group {
                      children      DEF Time_3 TimeSensor {
                       cycleInterval              1
                       startTime  0
                      }
              }
              DEF buttonSurfaceTranslationInterp_4 PositionInterpolator {
                      key           [ 0, 0.22, 0.48, 0.74,
                                      1 ]
                      keyValue      [ -0.229624 0.778573 -4.51614,
                                      -0.229624 0.5080 -4.51614,
                                      -0.229624 0.720994 -4.51614,
                                      -0.229624 0.749784 -4.51614,
                                      -0.229624 0.778573 -4.51614 ]
              }
              DEF buttonSurfaceScaleFactorInterp_5 PositionInterpolator {
                      key           0
                      keyValue      0.953259 0.953242 0.105989
              }
             ]
            }
            Shape {
             appearance             Appearance {
              material Material {
                      ambientIntensity          0.254777
                      diffuseColor 0.799639 0.0813426 0.126842
                      specularColor             0 0 0
                      emissiveColor             0 0 0
                      shininess     0.959184
                      transparency              0.6
              }
              texture  NULL
              textureTransform      NULL
             }
             geometry Box {
             }

            }
           ]
           translation   -0.229624 0.778573 -4.51614
           rotation      -1 0 0  1.5708
           scale         0.953259 0.953241 0.105989
           scaleOrientation          0 0 1  0
}
DEF buttonCase Transform {
           children      Shape {
            appearance Appearance {
             material   Material {
              ambientIntensity      2.70769
              diffuseColor          0.731633 0.296582 0.279057
              specularColor         0 0 0
              emissiveColor         0 0 0
              shininess             0.979592
              transparency          0.6
             }
             texture    NULL
             textureTransform       NULL
            }
            geometry   IndexedFaceSet {
             coord      Coordinate {
              point     [ -1 1 1,
                          -1 -1 1,
                          1 1 1,
                          1 -1 1,
                          1 1 -1,
                          1 -1 -1,
                          -1 1 -1,
                          -1 -1 -1,
                          -0.833479 0.886951 0.836993,
                          -0.833479 0.886951 -0.830504,
                          0.834017 0.886951 0.836993,
                          0.834017 0.886951 -0.830504 ]
             }
             color      NULL
             coordIndex             [ 0, 1, 3, 2, -1, 4, 5, 7,
                        6, -1, 6, 7, 1, 0, -1, 2,
                        3, 5, 4, -1, 9, 8, 10, 11,
                        -1, 1, 7, 5, 3, -1, 8, 9,
                        6, 0, -1, 10, 8, 0, 2, -1,
                        11, 10, 2, 4, -1, 9, 11, 4,
                        6, -1 ]
             colorIndex             [ ]
             normal     NULL
             texCoord NULL
             creaseAngle            0.5
            }
           }
```

```
                translation   -0.225267 0.602571 -4.5
                scale         1 0.230699 1
              }
             ]
             translation             0.112634 -2.25 -0.325447
             rotation   1 0 0  1.5708
             scale      0.5 0.518051 0.5
             scaleOrientation        0 0 1  0
            }
           }
    ]
   ]
}
 ROUTE touchSensorTrigger_2.touchTime TO Time_3.set_startTime
 ROUTE Time_3.fraction_changed TO buttonSurfaceTranslationInterp_4.set_fraction
 ROUTE Time_3.fraction_changed TO buttonSurfaceScaleFactorInterp_5.set_fraction
 ROUTE buttonSurfaceTranslationInterp_4.value_changed TO buttonSurface_1.set_translation
 ROUTE buttonSurfaceScaleFactorInterp_5.value_changed TO buttonSurface_1.set_scale
 ROUTE touchSensorTrigger_2.touchTime TO manejadorBoton.startTime
}

boton{}      # Prueba del prototipo

DEF VP1 Viewpoint {
 position    -0.913092 0.0176825 6.80882
 orientation 0 0 1  0
 fieldOfView             0.785398
 description "start"
}
DEF NavInfo NavigationInfo {
 type        "EXAMINE"
}

##############
```

slider.wrl

```
#Definición del slider para el control de la altura

#VRML V2.0 utf8

PROTO sliderProto [
              eventOut    SFFloat sliderResult
             ]
{
 Group {
  children  DEF slider Transform {
   children [
            DEF ManejadorSlider Script {
             eventOut    SFFloat     nivelOut IS sliderResult
             eventOut    SFVec3f     posicionApuntadorOut
             eventIn     SFVec3f     posicionApuntadorIn
             field       SFFloat     data        0
             url         "manejadorSlider.class"
            }
            Transform {
             children    Shape {
              appearance             Appearance {
               material Material {
                        ambientIntensity          0.249999
                        diffuseColor 0.425062 0.404868 0.426289
                        specularColor             0 0 0
                        emissiveColor             0 0 0
                        shininess    0.90303
                        transparency              0
               }
               texture  NULL
               textureTransform     NULL
              }
              geometry IndexedFaceSet {
               coord    Coordinate {
                        point          [ -1 1 1,
                                         -1 -1 1,
                                          .
                                          .
                                          .
                                         0.245178 -0.918054 1,
                                         0.245178 0.923502 1 ]
               }
               color    Color {
                        color          [ 0.425062 0.404868 0.426289,
                                         0 0 0 ]
               }
               coordIndex              [ 4, 5, 7, 6, -1, 6, 7, 1,
                                 0, -1, 2, 10, 13, 3, 5, 4,
                                 -1, 6, 0, 8, 2, 4, -1, 1,
                                 7, 5, 3, 9, -1, 0, 1, 9,
                                 12, 11, 8, -1, 10, 2, 8, 11,
                                 15, -1, 9, 3, 13, 14, 12, -1,
                                 13, 10, 15, 14, -1 ]
```

```
    colorIndex          [ 1, 0, 0, 0, 0, 0, 0, 0,
                0 ]
    normal  NULL
    texCoord            TextureCoordinate {
            point       [ 0 1,
                          0 0,
                            .
                            .
                            .
                          0.622589 0.961751,
                          0.622589 0.961751 ]
    }
    colorPerVertex      FALSE
    texCoordIndex       [ 0, 1, 3, 2, -1, 0, 1, 3,
                  2, -1, 0, 8, 14, 1, 3, 2,
                  -1, 0, 1, 5, 3, 2, -1, 0,
                  1, 3, 2, 7, -1, 0, 1, 6,
                  12, 10, 4, -1, 9, 2, 4, 11,
                  18, -1, 6, 3, 15, 16, 13, -1,
                  15, 9, 16, 17, -1 ]
    solid   FALSE
    creaseAngle         0.5
   }
  }
  translation 0.00175315 -0.0859035 -0.0243123
  rotation    0 0 1  0
  scale       0.0627959 0.627959 0.00627959
  scaleOrientation          0 0 1  0
}
Transform {
  children  [
   DEF _1 PlaneSensor {
    minPosition         0 -0.6
    maxPosition         0 0.43
    offset  2 1 0
    autoOffset          FALSE
   }
   DEF knob_2 Transform {
    children Shape {
            appearance  Appearance {
             material   Material {
              ambientIntensity      0.250001
              diffuseColor          0.152381 0.145141 0.15282
              specularColor         0 0 0
              emissiveColor         0 0 0
              shininess 0.957576
              transparency          0
             }
             texture    NULL
             textureTransform       NULL
            }
            geometry    IndexedFaceSet {
             coord      Coordinate {
              point     [ -0.01 0.0302819 -0.0102819,
                          -0.01 -0.0301506 -0.0101506,
                            .
                            .
                            .
                          0.01 0.0085837 0.01,
                          -0.01 0.00858368 0.01 ]
             }
             color      NULL
             coordIndex [ 4, 5, 7, 6, -1, 6, 7, 1,
                          9, 11, 0, -1, 2, 10, 8, 3,
                          5, 4, -1, 8, 0, 2, 4, -1,
                          1, 7, 5, 3, -1, 9, 1, 3,
                          8, -1, 0, 11, 10, 2, -1, 11,
                          9, 8, 10, -1 ]
             colorIndex -1
             normal     NULL
             texCoord   TextureCoordinate {
              point     [ 0 1,
                          0 0,
                            .
                            .
                            .
                          1 0.929185,
                          0 0.929185 ]
             }

             texCoordIndex          [ 0, 1, 3, 2, -1, 0, 1, 3,
                          7, 10, 2, -1, 0, 8, 5, 1,
                          3, 2, -1, 0, 1, 3, 2, -1,
                          0, 1, 3, 2, -1, 6, 1, 3,
                          4, -1, 0, 11, 9, 2, -1, 11,
                          6, 4, 9, -1 ]
             creaseAngle            0.5
            }
    }
    translation         0.000677751 0.393597 0.0162768
```

```
      scale     5.38382 3.43086 3.43086
     }
    ]
    translation 0 -0.58 0
   }
   Transform {
    children     Shape {
     appearance          Appearance {
      material Material {
               ambientIntensity          0.157472
               diffuseColor 0.67482 0.64276 0.676768
               specularColor             0 0 0
               emissiveColor             0 0 0
               shininess     0.90303
               transparency              0
      }
      texture  NULL
      textureTransform     NULL

     }
     geometry Box {
      size     0.2 0.2 0.2
     }
    }
    translation -0.00763095 -0.0989436 -0.0529028
    rotation     0 0 1  0
    scale        1.15998 7.93125 0.141044
   }
   Transform {
    children     [
     Transform {
      children [
               Transform {
                children     Shape {
                 appearance          Appearance {
                  material Material {
                           ambientIntensity          0.250001
                           diffuseColor 0.152381 0.145141 0.15282
                           specularColor             0 0 0
                           emissiveColor             0 0 0
                           shininess     0.957576
                           transparency              0
                  }
                  texture  NULL
                  textureTransform     NULL
                 }
                 geometry Box {
                  size     0.2 0.2 0.2
                 }
                }
                translation -1.35441 1.22689 0.0242854
                rotation     -3.55891e-07 -3.57628e-07 1  4.70896
                scale        0.264543 1 0.0174844
                scaleOrientation          0 0 1  0
               }
               Transform {
                children     Shape {
                 appearance          Appearance {
                  material Material {
                           ambientIntensity          0.250001
                           diffuseColor 0.152381 0.145141 0.15282
                           specularColor             0 0 0
                           emissiveColor             0 0 0
                           shininess     0.957576
                           transparency              0
                  }
                  texture  NULL
                  textureTransform     NULL
                 }
                 geometry Box {
                  size     0.2 0.2 0.2
                 }
                }
                translation -1.3542 1.22351 0.0207868
                scale        0.264543 1 0.0174844
               }
     ]
     translation              -0.0959868 0.234074 0.169558
     rotation 0 0 1  0
     scale    0.674133 0.674133 0 674135
    }
    Transform {
     children Shape {
               appearance  Appearance {
                material    Material {
                 ambientIntensity          0.25
                 diffuseColor              0.138667 0.138667 0.138667
                 specularColor             0.168 0.168 0.168
                 emissiveColor             0 0 0
                 shininess 0.153696
```

```
            transparency        0
           }
           texture     NULL
           textureTransform        NULL
          }
          geometry    IndexedFaceSet {
           coord       Coordinate {
            point       [ 0 0.1 0,
                          -0.0382683 0.1 -0.092388,
                          .
                          .
                          .
                          0.0707107 0.1 -0.0707107,
                          0.0382683 0.1 -0.092388 ]
           }
           color       NULL
           coordIndex [ 0, 1, 2, -1, 0, 3, 1, -1,
                        0, 2, 4, -1, 0, 4, 5, -1,
                          .
                          .
                          .
                        0, 15, 16, -1, 0, 16, 3, -1 ]
           colorIndex  [ ]
           normal      NULL
           texCoord    TextureCoordinate {
            point       [ 0.5 0.5,
                          0.308659 0.96194,
                          0.146447 0.853554,
                          .
                          .
                          .
                          0.853553 0.853553,
                          0.691342 0.96194 ]
           }
           texCoordIndex           [ 0, 1, 2, -1, 0, 3, 1, -1,
                        0, 2, 4, -1, 0, 4, 5, -1,
                          .
                          .
                          .
                        0, 15, 16, -1, 0, 16, 3, -1 ]
           ccw         TRUE
           solid       TRUE
           convex      TRUE
           creaseAngle             0.5
          }
     }
     translation             -1.01473 1.05608 0.0812525
     rotation 1 0 0  1.5708
     scale    1 0.999999 0.999999
    }
   ]
   translation 0.36039 -0.0560347 -0.13177
   rotation    0 0 -1  0.269323
   scale       0.520684 0.520684 0.520684
}
Transform {
 children    [
  Transform {
   children [
            Transform {
             children    Shape {
              appearance              Appearance {
               material Material {
                        ambientIntensity        0.250001
                        diffuseColor 0.152381 0.145141 0.15282
                        specularColor           0 0 0
                        emissiveColor           0 0 0
                        shininess     0.957576
                        transparency            0
               }
               texture  NULL
               textureTransform      NULL
              }
              geometry Box {
               size      0.2 0.2 0.2
              }
             }
             translation -1.35441 1.22689 0.0242854
             rotation    -3.55691e-07 -3.57628e-07 1  4.70688
             scale       0.264543 1 0.0174844
             scaleOrientation        0 0 1  0
            }
            Transform {
             children    Shape {
              appearance              Appearance {
               material Material {
                        ambientIntensity        0.250001
                        diffuseColor 0.152381 0.145141 0.15282
                        specularColor           0 0 0
                        emissiveColor           0 0 0
```

```
                    shininess       0.957576
                    transparency            0
                  }
                  texture NULL
                  textureTransform      NULL
                }
                geometry Box {
                  size      0.2 0.2 0.2
                }
              }
              translation -1.3542 1.22351 0.0207868
              scale       0.264543 1 0.0174844
            }
          ]
          translation           -0.0959668 0.234074 0.169558
          rotation 0 0 1  0
          scale     0.674133 0.674133 0.674135
        }
        Transform {
          children Shape {
            appearance  Appearance {
              material    Material {
                ambientIntensity        0.25
                diffuseColor            0.138667 0.138667 0.138667
                specularColor           0.168 0.168 0.168
                emissiveColor           0 0 0
                shininess 0.153696
                transparency            0
              }
              texture     NULL
              textureTransform        NULL
            }
            geometry    IndexedFaceSet {
              coord       Coordinate {
                point       [ 0 0.1 0,
                              -0.0382683 0.1 -0.092388,
                              .
                              .
                              .
                              0.0707107 0.1 -0.0707107,
                              0.0382683 0.1 -0.092388 ]
              }
              color       NULL
              coordIndex [ 0, 1, 2, -1, 0, 3, 1, -1,
                           0, 2, 4, -1, 0, 4, 5, -1,
                           .
                           .
                           .
                           0, 13, 14, -1, 0, 14, 15, -1,
                           0, 15, 16, -1, 0, 16, 3, -1 ]
              colorIndex  [ ]
              normal      NULL
              texCoord    TextureCoordinate {
                point       [ 0.5 0.5,
                              0.308859 0.96194,
                              .
                              .
                              .
                              0.853553 0.853553,
                              0.691342 0.96194 ]
              }
              texCoordIndex           [ 0, 1, 2, -1, 0, 3, 1, -1,
                           0, 2, 4, -1, 0, 4, 5, -1,
                           .
                           .
                           .
                           0, 13, 14, -1, 0, 14, 15, -1,
                           0, 15, 16, -1, 0, 16, 3, -1 ]
              ccw         TRUE
              solid       TRUE
              convex      TRUE
              creaseAngle             0.5
            }
          }
          translation           -1.01473 1.05608 0.0812525
          rotation 1 0 0  1.5708
          scale     1 0.999999 0.999999
        }
      ]
      translation 0.00776305 -1.56966 -0.122399
      rotation    0 0 -1  0.754638
      scale       0.520684 0.520684 0.520684
    }
  ]
 }
}
 ROUTE _1.trackPoint_changed TO ManejadorSlider.posicionApuntadorIn
 ROUTE ManejadorSlider.posicionApuntadorOut TO knob_2.set_translation
}#Slider
```

```
DEF slider1 sliderProto {}

DEF VP1 Viewpoint {
  position     -0.00763095 -0.0989437 3.24576
  orientation 0 0 1  0
  fieldOfView                0.785398
  description "start"
}
```

Apéndice D

Código fuente del simulador, Java

Definición de clases de control.

manejadorCuerpoRobot.java

```
import java.io.*;
import java.net.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.lang.*;

/*************************************************************/
public class manejadorCuerpoRobot extends Script {
public float deltaTraslacion,deltaRotacion;
public SFVec3f posicionActual,traslacionRobotOut;
public SFInt32  grupo1AccionOut,grupo2AccionOut;
public SFVec3f escala,escalaOut;
public SFRotation rotacionXActual,rotacionYActual,rotacionZActual;
public SFRotation rotacionXRobotOut,rotacionYRobotOut,rotacionZRobotOut;
public float altura = 1;
public int accionActual = 0;                    /*Selector para efectuar alguna accion en cada ciclo: 0 nada, 1 adel, 2 atras,  3 derecha, 4 izq,*/
public int accionRecienEspecificada = 100;      /* Del robot debe llegar una senal de termino de accion en cuyo caso accionActual toma ahora el
                                                   valor de accionRecienEspecificada la cual está expuesta a constante cambio por el usuario*/
public boolean grupoActivo = false;             /*Indican si el actual grupo esta activo o no, true grupo1 false grupo2*/
public boolean grupo1TerminoAccionIn = false,grupo2TerminoAccionIn = false;
int fase = 4;                                   /*Fase en la que se encuentra la accion, 0 nada, 1 arriba, 2 despA, 3 despB, 4 abajo*/
int accionRobot = 0;                            /*Rotaciones y desplazamientos a efectuar  por el robot*/

/*************************************************************/
/*Toma los campos especificados en vrml como field y los envia a sus respectivos extremidades como mensajes de inicializacion*/
public void initialize() {
            deltaRotacion = ((SFFloat) getField("deltaRotacion")).getValue();
            deltaTraslacion = ((SFFloat) getField("deltaTraslacion")).getValue();
            escala = (SFVec3f) getField("escala");
            escalaOut = (SFVec3f) getEventOut("escalaOut");
            escalaOut.setValue(escala);
            System.out.println("Robot: campo escala" + escala);
            posicionActual = ((SFVec3f) getField("posicionInicial"));
            traslacionRobotOut = (SFVec3f) getEventOut("traslacionRobotOut");
            traslacionRobotOut.setValue(posicionActual);
            rotacionXActual  = (SFRotation) getField("rotacionXInicial");
            rotacionXRobotOut = (SFRotation) getEventOut("rotacionXRobotOut");
            rotacionXRobotOut.setValue(rotacionXActual);
            System.out.println("Robot: campo rotacionXInicial" + rotacionXActual);
            rotacionYActual  = (SFRotation) getField("rotacionYInicial");
            rotacionYRobotOut = (SFRotation) getEventOut("rotacionYRobotOut");
            rotacionYRobotOut.setValue(rotacionYActual);
            System.out.println( "Robot: campo rotacionYInicial" + rotacionYActual);
            rotacionZActual  = (SFRotation) getField("rotacionZInicial");
            rotacionZRobotOut = (SFRotation) getEventOut("rotacionZRobotOut");
            rotacionZRobotOut.setValue(rotacionZActual);
            System.out.println( "Robot: campo rotacionZInicial" + rotacionZActual);
            grupo1AccionOut = (SFInt32) getEventOut("grupo1AccionOut");
            System.out.println( "Robot: envia evento inicial 100, inicializar extrem" +  grupo1AccionOut);
            grupo2AccionOut = (SFInt32) getEventOut("grupo2AccionOut");
            System.out.println( "Robot: envia evento inicial 100, inicializar extrem" +  grupo2AccionOut);
}

/*************************************************************/
public void processEvent( Event e ) {

      if ( e.getName().equals( "grupo2TerminoAccionIn" ) )
        { /*La accion actual no debe cambiar hasta recibir la senal de termino del robot */
                    if(accionActual! = 0  ) fase + + ;
        }
       if ( e.getName().equals( "grupo1TerminoAccionIn" ) )
        { /*La accion actual no debe cambiar hasta recibir la senal de termino del robot */
          if(accionActual! = 0 ) fase + + ;
        }
       /*Comandos del usuario*/
       if ( e.getName().equals( "adelanteRobot" ) ) {
          accionRecienEspecificada = 1;
          System.out.println( "Robot: evento adelanteRobot" );
        }
       if ( e.getName().equals( "atrasRobot" ) ) {
```

```
        accionRecienEspecificada = 2;
        System.out.println( "Robot: evento atrasRobot" );
      }
     if ( e.getName().equals("derechaRobot" ) ) {
        accionRecienEspecificada = 3;
        System.out.println( "Robot: evento derechaRobot" );
      }
     if ( e.getName().equals( "izquierdaRobot" ) ) {
        accionRecienEspecificada = 4;
        System.out.println( "Robot: evento izquierdaRobot" );
      }
     if ( e.getName().equals( "detenerRobot" ) ) {
        accionRecienEspecificada = 0;
        System.out.println( "detenerRobot" );
       // accionOut.setValue(0);
      }
     if ( e.getName().equals( "alturaIn" ) ) {
        System.out.println( "Robot: evento cambio de altura " + e.getValue() );
         posicionActual.setValue(posicionActual.getX(),((ConstSFFloat)e.getValue()).getValue(),posicionActual.getZ());
         traslacionRobotOut.setValue(posicionActual);          /*Envia mensaje al robot de cambio de posicion*/
      }
      if ( e.getName().equals( "traslacionExtremo1In" ) ) {
        //System.out.println( "$$$$$$$$$$$$$$Robot: evento movimiento de extremo1 " + e.getValue() );
      }
      if ( e.getName().equals( "temporizador" ) )
      {
        //System.out.println( "temporizador" );
        /*La traslacion implica la rotacion del delta sobre Y y la posterior suma del */
        desarrollaAcciones();
        movilizaRobot();
      }
      //System.out.println( "Robot: estatus  grupoActivo " + grupoActivo + "  fase" + fase );
}

/*************************************************************/
public void intercambioGrupos()
{
        if( grupoActivo )
        {
                grupoActivo = false;
        }
        else
        {
                grupoActivo = true;
        }
}

/*************************************************************/
public void enviaAccionAGrupoActivo(int accionG )
{
      /*Dependiendo del grupo selecciona la accion conveniente, 0 detener 1 arriba 2 abajo 3 avanzar, 4 retroceder, 5 rotar derecha 6 rotar
          izquierda*/
      if (accionG = =0)
      if(grupoActivo) grupo1AccionOut.setValue(0);          /*grupo1*/
            else      grupo2AccionOut.setValue(0);          /*grupo2*/
          else if (accionG = =100)
      if(grupoActivo) grupo1AccionOut.setValue(100);        /*grupo1*/
            else      grupo2AccionOut.setValue(100);        /*grupo2*/
          else if (accionG = =1)
      if(grupoActivo) grupo1AccionOut.setValue(1);          /*grupo1*/
            else      grupo2AccionOut.setValue(1);          /*grupo2*/
          else if (accionG = =2)
      if(grupoActivo) grupo1AccionOut.setValue(2);          /*grupo1*/
            else      grupo2AccionOut.setValue(2);          /*grupo2*/
          else if (accionG = =3)
      if(grupoActivo) grupo1AccionOut.setValue(13);         /*grupo1*/
            else      grupo2AccionOut.setValue(23);         /*grupo2*/
          else if (accionG = =4)
      if(grupoActivo) grupo1AccionOut.setValue(14);         /*grupo1*/
            else      grupo2AccionOut.setValue(24);         /*grupo2*/
          else if (accionG = =5)
      if(grupoActivo) grupo1AccionOut.setValue(5);          /*grupo1*/
            else      grupo2AccionOut.setValue(5);          /*grupo2*/
          else if (accionG = =6)
      if(grupoActivo) grupo1AccionOut.setValue(6);          /*grupo1*/
            else      grupo2AccionOut.setValue(6);          /*grupo2*/
}


/*************************************************************/
public void desarrollaAcciones()
{
      if(fase = =10 )
       {
            accionActual = accionRecienEspecificada; fase = 0;
       }
      if(accionActual = =0)          /*Adelante*/
       {
            fase = 10;
       }
```

```
        if(accionActual = = 1)          /*Adelante*/
         {
        if (fase = = 0)                 /*Eleva extremidades*/
            {
              enviaAccionAGrupoActivo(1);
              accionRobot = 1;
            }
        if (fase = = 1)                 /*Avance de extremidades*/
            {
                        enviaAccionAGrupoActivo(3);
                        accionRobot = 1;
            }
        if (fase = = 2)                 /*Baja las extremidades*/
            {
                        enviaAccionAGrupoActivo(100);
                        accionRobot = 0;
            }
        if (fase = = 3)
            {
                enviaAccionAGrupoActivo(0);
                intercambiaGrupos();
                accionRobot = 0;
                accionRecienEspecificada = accionActual = 0;
                fase = 10;
            }

          }

        if(accionActual = = 2)  /*Atras*/
         {
           if (fase = = 0)              /*Eleva extremidades*/
            {
                enviaAccionAGrupoActivo(1);
                acccionRobot = 2;
            }
           if (fase = = 1)              /*Retrocede de extremidades*/
            {
                enviaAccionAGrupoActivo(4);
                accionRobot = 2;
            }
           if (fase = = 2)  /*baja las extremidades*/
            {
                enviaAccionAGrupoActivo(100);
                accionRobot = 0;
            }
           if (fase = = 3)
            {
               enviaAccionAGrupoActivo(0);
                intercambiaGrupos();
                accionRobot = 0;
                accionRecienEspecificada = accionActual = 0;
                fase = 10;
            }

          }
        if(accionActual = = 100)
         {
           if (fase = = 0)
            {
                enviaAccionAGrupoActivo(100);
            }
           if (fase = = 1)
            {
                enviaAccionAGrupoActivo(0);
                intercambiaGrupos();
                //fase + + ;
            }
           if(fase = = 2)
            {
                enviaAccionAGrupoActivo(100);
            }
          if(fase = = 3)
             {
               enviaAccionAGrupoActivo(0);
               intercambiaGrupos();
                accionActual = 0;
                fase = 4;
               }
        }
}

/*****************************************************************/
public void movilizaRobot()
{
        if(accionRobot = = 1 )
         {  float rotacionY[] = {0f,0f,0f,0f},x,y,z;
            rotacionYActual.getValue(rotacionY);
               x = (float) (deltaTraslacion*(Math.cos(rotacionY[3]))) + posicionActual.getX();
               y = (float) posicionActual.getY();
               z = (float) -(deltaTraslacion*(Math.sin(rotacionY[3]))) + posicionActual.getZ();
```

```
            posicionActual.setValue(x,y,z);
            traslacionRobotOut.setValue(posicionActual);                    /*Evia mensaje al robot de cambio de posicion, para inicializar*/

            System.out.println( "Robot: evento cambio de posicion actual " + posicionActual.getX() + " " + posicionActual.getY() + "
                                " + posicionActual.getZ() );
        }
        if(accionRobot == 2)
        {   float rotacionY[] = {0f,0f,0f,0f},x,y,z;
            rotacionYActual.getValue(rotacionY);
              x = (float) (-deltaTraslacion*(Math.cos(rotacionY[3]))) + posicionActual.getX();
              y = (float) posicionActual.getY();
              z = (float) -(-deltaTraslacion*(Math.sin(rotacionY[3]))) + posicionActual.getZ();
              posicionActual.setValue(x,y,z);
            traslacionRobotOut.setValue(posicionActual);                    /*Envia mensaje al robot de cambio de posicion, para inicializar*/
            System.out.println( "Robot: evento cambio de posicion actual  " + posicionActual.getX() + " " + posicionActual.getY() + "
                                " + posicionActual.getZ() );
        }
        if(accionRobot == 3)
        {   float rotacionY[] = {0f,0f,0f,0f},x,y,z;
            rotacionYActual.getValue(rotacionY);
            rotacionYActual.setValue(0, 1,0,rotacionY[3]-deltaRotacion);
            rotacionYRobotOut.setValue(rotacionYActual);
            System.out.println( "Robot: evento cambio de rotacion actual " + rotacionYActual);
        }
        if(accionRobot == 4)
        {   float rotacionY[] = {0f,0f,0f,0f},x,y,z;
            rotacionYActual.getValue(rotacionY);
            rotacionYActual.setValue(0, 1,0,rotacionY[3] + deltaRotacion);
            rotacionYRobotOut.setValue(rotacionYActual);
            System.out.println( "Robot: evento cambio de rotacion actual " + rotacionYActual);
        }
}
} /*Clase manejadorCuerpoRobot */

/*************************************************************************/
```

manejadorGrupo.java

```
import java.io.*;
import java.net.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import Connection;
import java.lang.*;

/*****************************************************/
public class manejadorGrupo extends Script
{
public SFVec3f traslacionExtremoExtremidadFrontal,traslacionExtremoExtremidadMedia,
traslacionExtremoExtremidadTrasera;
public SFInt32 accionExtremidadFrontalOut,accionExtremidadMediaOut,accionExtremidadTraseraOut ;
public boolean extremidadFrontalActividadFinalizada = false, extremidadMediaActividadFinalizada = false,  extremidadTraseraActividadFinalizada = false;
public float deltaTraslacion;
float xe = 2, ye = 0, ze = 0;
float xFrontal, yFrontal, zFrontal;   /*Coordenadas de los extremos de las
            extremidades*/
float xMedia, yMedia, zMedia;
float xTrasera, yTrasera, zTrasera;
boolean obstaculoExtremidadFrontal = false;
float avance = 0.5f;                                /*Magnitud de desplazamiento + o - a partir del
                                                    punto inicial de la extremidad*/
float distanciaAvanceFrontal = 0;          /*Desplazamiento frontal de una extremidad*/
boolean grupoActivo;                       /*indica al grupo que tiene la responsabilidad de
                                                    avanzar*/
SFBool  terminoAccionOut;
int accionActual = 0;                      /*0 inactivo, 1 avanzar adelante, 2 retroceder, 3
                                                    avance derecha, avance izaquierda, solo es
                        posible cambiarla cuando la actividad ha
                                                    terminado*/
int accion = 0;
int fase = 0;                                       /*Secuencia de movimientos para cada
                                                    actividad*/
int identificadorGrupo;

/*****************************************************/

public void initialize() {
  traslacionExtremoExtremidadFrontal = (SFVec3f) getEventOut("traslacionExtremoExtremidadFrontal");
  traslacionExtremoExtremidadMedia = (SFVec3f) getEventOut("traslacionExtremoExtremidadMedia");
  traslacionExtremoExtremidadTrasera = (SFVec3f)      getEventOut("traslacionExtremoExtremidadTrasera");
accionExtremidadFrontalOut = (SFInt32) getEventOut("accionExtremidadFrontalOut");
  accionExtremidadMediaOut = (SFInt32) getEventOut("accionExtremidadMediaOut");
  accionExtremidadTraseraOut = (SFInt32) getEventOut("accionExtremidadTraseraOut");
deltaTraslacion =  ((SFFloat)getField("deltaTraslacion")).getValue();
  terminoAccionOut = (SFBool) getEventOut("terminoAccionOut");
  //terminoAccionOut.setValue(true);
}
```

```
/*************************************************/
public void processEvent( Event e ) {

        if ( e.getName().equals( "accionIn" ) ) {
         accion = ((ConstSFInt32)e.getValue()).getValue();
         //System.out.println( "Grupo: evento accion " );
        }

       if ( e.getName().equals( "extremidadFrontalActividadFinalizada" ) ) {    /*Bandera de estatus, true = terminado, false = coupado desarrollando*/
            extremidadFrontalActividadFinalizada = ((ConstSFBool)e.getValue())
                                                                                                    .getValue();
        //System.out.println( "Grupo: evento extremidadFrontalActividadFinalizada ");
       }

            if ( e.getName().equals( "extremidadMediaActividadFinalizada" ) ) {          /*Bandera de estatus, true = terminado, false = coupado
desarrollando*/
            extremidadMediaActividadFinalizada = ((ConstSFBool)e.getValue())
                                                                                                    .getValue();
         //System.out.println( "Grupo: evento extremidadMediaActividadFinalizada ");
       }

       if ( e.getName().equals( "extremidadTraseraActividadFinalizada" ) ) {          /*Bandera de estatus, true = terminado, false = coupado
desarrollando*/
            extremidadTraseraActividadFinalizada = ((ConstSFBool)e.getValue())
                                                                                                    .getValue();
         //System.out.println( "Grupo: evento extremidadTraseraActividadFinalizada ");
       }

            desarrollaAcciones();      /*Verifica la accion a realizar y ejecuta cada una
                                                    de sus fases*/
            //System.out.println( "Grupo: estatus " + " accion " + accion + " accionActual " + accionActual + " activo " + grupoActivo);


 } /*Eventos*/

/*************************************************/
 public void desarrollaAcciones()
 {
            if( accionActual = = 100 )
             {

                        accionExtremidadFrontalOut.setValue(100);
                        accionExtremidadMediaOut.setValue(100);
                        accionExtremidadTraseraOut.setValue(100);
             }

            //System.out.println( "Grupo " + identificadorGrupo + ": estatus de extremidades " + extremidadFrontalActividadFinalizada + " " +
extremidadMediaActividadFinalizada + " " + extremidadTraseraActividadFinalizada);

            if(extremidadFrontalActividadFinalizada  &&
                        extremidadMediaActividadFinalizada  &&
                        extremidadTraseraActividadFinalizada && accionActual != 0)
            {
             accionActual = 0;                                          /*Elimina accionActual*/
             terminoAccionOut.setValue(true);                  /*Informa al robot su termino de
                                                                                    actividades*/
             extremidadFrontalActividadFinalizada = false;      /*Reinicia el estatus de las
                                                    extremidades*/
             extremidadMediaActividadFinalizada = false;
             extremidadTraseraActividadFinalizada = false;
            }

            if(accionActual = = 0)     /*No hay ninguna accionActual especificada , sin                                  embargo el grupo
esta activo, se procede a tomar la
                                                    siguiente accion*/
            {
             accionActual = accion;
            }
            if(accionActual = = 1)                                      /*Elevar extremidades */
            {
                        accionExtremidadFrontalOut.setValue(1);
                        accionExtremidadMediaOut.setValue(1);
                        accionExtremidadTraseraOut.setValue(1);
            }
            if(accionActual = = 2)                                      /*Bajar extremidades */
            {
                        accionExtremidadFrontalOut.setValue(2);
                        accionExtremidadMediaOut.setValue(2);
                        accionExtremidadTraseraOut.setValue(2);
            }
            if(accionActual = = 13)                           /*Avanzar extremidades gaso del grupo1*/
            {
                        accionExtremidadFrontalOut.setValue(5);                 /*Estira extremidad*/
                        accionExtremidadMediaOut.setValue(4);
                        accionExtremidadTraseraOut.setValue(6);                 /*Retrae extremidad*/
            }
            if(accionActual = = 23)                           /*Avanzar extremidades gaso del grupo2*/
            {
                        accionExtremidadFrontalOut.setValue(5);                 /*Estira extremidad*/
                        accionExtremidadMediaOut.setValue(3);
```

```
                accionExtremidadTraseraOut.setValue(6);                    /*Retrae extremidad*/
        }
        if(accionActual == 14)                          /*Retroceder extremidades  del grupo1*/
        {
                accionExtremidadFrontalOut.setValue(6);                    /*Retrae extremidad*/
                accionExtremidadMediaOut.setValue(3);
                accionExtremidadTraseraOut.setValue(5);                    /*Estira extremidad*/
        }
        if(accionActual == 24)                          /*Retroceder extremidades  del grupo2*/
        {
                accionExtremidadFrontalOut.setValue(6);                    /*Retrae extremidad*/
                accionExtremidadMediaOut.setValue(4);
                accionExtremidadTraseraOut.setValue(5);                    /*Estira extremidad*/
        }
        if(accionActual == 5 )                          /*Girar extremidades  del grupo1*/
        {
                accionExtremidadFrontalOut.setValue(3);
                accionExtremidadMediaOut.setValue(3);
                accionExtremidadTraseraOut.setValue(3);
        }
        if(accionActual == 6 )                          /*Girar extremidades  del grupo2*/
        {
                accionExtremidadFrontalOut.setValue(4);
                accionExtremidadMediaOut.setValue(4);
                accionExtremidadTraseraOut.setValue(4);
        }
 }

}/*Clase manejadorGrupo*/

/*****************************************************/
```

manejadorExtremidad.java

```
import java.io.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.lang.*;
import Topologia;

/*****************************************************/
public class manejadorExtremidad extends Script {
public SFVec3f posicionInicial,posicionInicialOut,xyzExtremoOut,traslacionExtremoOut;
public ConstSFVec3f posicionActualRobot,posicionAnteriorRobot;
public ConstSFRotation rotacionYActualRobot,rotacionYAnteriorRobot;
public SFRotation rotacionYInicial, rot1Out, rot2Out, rot3Out; /*rotaciones eslabones 1 2 3*/
public float escala=1000, deltaTraslacion=0.1f;
public float xExtremo=2f, yExtremo=0, zExtremo=0; /*relativas al 0 del eslabon 4 de la extremidad*/
public boolean colisionConTerreno=false;
public Topologia topologia;
float l4=750/escala;      /*Longitud del eslabon 4*/
float h4=180/escala;      /*Altura del eslabon 4*/
float l3=170/escala;      /*Longitud del eslabon 3*/
float h3=150/escala;      /*Altura del eslabon 3*/
float l2=710/escala;      /*Longitud del eslabon 2*/
float h2=0;               /*Altura del eslabon 2*/
float l1=1650/escala;     /*Longitud del eslabon 1*/
float h1=0;               /*Altura del eslabon 1*/
float altura=0.3f;
float xInit=0, yInit=altura, zInit=0;          /*Traslacion de los ejes del escenario hacia el eje del robot*/
float rotXRobot=0,/*rotYRobot=0,*/ rotZRobot=0; /*Rotaciones del eje del robot con referencia a los ejes del escenario*/
float rotYExtremidad=0;                        /*Rotacion para posicionar cada extremidad*/
SFBool  accionFinalizadaOut;                   /*Mensaje de termino de acciones al grupo al que pertenece*/
int accionActual=0, accionADesarrollar=0;      /*Comienza inicializando*/
SFBool colisionOut;                            /*Mensajes de colision al grupo, caracter informativo*/
SFBool levantarRobot;                          /*Una vez que se ha producido colision y se continua el movimiento de la extremidad*/

/*****************************************************/
public void initialize()
{
  topologia=(Topologia) new Topologia();
  colisionOut= (SFBool )getEventOut("colisionOut");
  posicionInicial= (SFVec3f) getField("posicionInicial");
 // System.out.println("Extremidad: campo  posicionInicial " + posicionInicial);
  rotacionYInicial  = (SFRotation) getField("rotacionYInicial");
 // System.out.println("Extremidad: campo  rotacionYInicial " + rotacionYInicial );
  rot1Out  = (SFRotation) getEventOut("rot1Out");
  rot2Out  = (SFRotation) getEventOut("rot2Out");
  rot3Out  = (SFRotation) getEventOut("rot3Out");
  accionFinalizadaOut  = (SFBool) getEventOut("accionFinalizadaOut");
  traslacionExtremoOut  = (SFVec3f) getEventOut("traslacionExtremoOut");
  accionActual=0;
 }

/*****************************************************/
```

```
public void processEvent( Event e ) {
 if ( e.getName().equals( "traslacionRobotIn" ) ) {
     if(posicionActualRobot! = null)
                {if( colisionConTerreno ) extremoFijoAnteTraslacion(e,-1);} /*Mantiene el extremo en movimiento junto con el robot*/
            else {
                        posicionActualRobot = (ConstSFVec3f)e.getValue();
                        posicionAnteriorRobot = new ConstSFVec3f(posicionActualRobot.getX(),
                                                posicionActualRobot.getY(),posicionActualRobot.getZ());
                }
        //System.out.println("Extremidad: evento traslacionRobotIn " + ((ConstSFVec3f)e.getValue()).getX() + " " +
                        (((ConstSFVec3f)e.getValue()).getY() + " " + ((ConstSFVec3f)e.getValue()).getZ() );
            }
 if ( e.getName().equals( "rotacionXRobotIn" ) ) {
        //System.out.println("Extremidad: evento rotacionXRobotIn " + e.getValue());
 //           extremoFijoAnteRotacionX();
     }
 if ( e.getName().equals( "rotacionYRobotIn" ) ) {
        //System.out.println("Extremidad: evento rotacionYRobotIn " + e.getValue());
 if(rotacionYActualRobot! = null)
 {if( colisionConTerreno ) extremoFijoAnteRotacionY(e,-1);}                  /*Mantiene el extremo en movimiento junto con el robot*/
            else { float temp[] = {0,0,0,0};
                        rotacionYActualRobot = (ConstSFRotation)e.getValue();
                        rotacionYActualRobot.getValue(temp);
                        rotacionYAnteriorRobot = new ConstSFRotation(temp[0],temp[1],temp[2],temp[3]);
                }
            rotacionYActualRobot = (ConstSFRotation) e.getValue();
}
if ( e.getName().equals( "rotacionZRobotIn" ) ) {
        //System.out.println("Extremidad: evento rotacionZRobotIn " + e.getValue());
        extremoFijoAnteRotacionZ();
     }
if ( e.getName().equals( "traslacionExtremoIn" ) ) {
        //System.out.println("Extremidad: evento traslacionExtremoIn a evaluar " + ((ConstSFVec3f)e.getValue()).getX() + " " +
((ConstSFVec3f)e.getValue()).getY() + " " + ((ConstSFVec3f)e.getValue()).getZ() );
if ( evaluaColision(((ConstSFVec3f)e.getValue()).getX(),((ConstSFVec3f)e.getValue()).getY(),((ConstSFVec3f)e.getValue()).getZ()) < deltaTraslacion ) //hubo
colision, manda mensaje al controlador
          { colisionOut.setValue(true); /*System.out.println("Extremidad: evento COLISION#######");*/}
else  //no hubo colision, se procede a avanzar
          {
            xExtremo = ((ConstSFVec3f)e.getValue()).getX();
              yExtremo = ((ConstSFVec3f)e.getValue()).getY();
              zExtremo = ((ConstSFVec3f)e.getValue()).getZ();
              //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
            cinematicaPuntoExtremo();
           }
      }
 if ( e.getName().equals( "accionADesarrollar" ) ) {
                        accionADesarrollar = ((ConstSFInt32)e.getValue()).getValue();
                        asignaAccion();
                        accionFinalizadaOut.setValue(false);          /*como recibio un comando cambia la bandera*/
                        //System.out.println("Extremidad: evento accionADesarrollar " + accionADesarrollar );
                }
              desarrollaAcciones();   /*verifica la accion a realizar y ejecuta cada una de sus fases*/
              traslacionExtremoOut.setValue(xExtremo,yExtremo,zExtremo);
}

/*****************************************************************************/
public void asignaAccion()
{
if(accionActual = = 0)
 {
            accionActual = accionADesarrollar; /*Solamente cuando se ha terminado de ejecutar una accion*/
 }
}

/*****************************************************************************/
/*Realiza los movimientos necesarios de a cuerdo a la acción especificada*/
public void desarrollaAcciones()
{
if( accionActual = = 100)  /*posiciona las extremidades a nivel del suelo*/
{
   float deltaTemporal; /*contiene el espacio restante para tocar el terreno*/
 if ( (deltaTemporal = evaluaColision(xExtremo ,yExtremo-deltaTraslacion  ,zExtremo)) < deltaTraslacion ) //Hubo colision, manda mensaje al controlador
          {
            yExtremo- = deltaTemporal;                          /*Avanza el espacio restante para tocar el terreno*/
            cinematicaPuntoExtremo();
            //System.out.println("Extremidad: evento COLISION#######  " + deltaTemporal);
             colisionOut.setValue(true);
            //System.out.println("Extremidad: evento COLISION#######");
            colisionConTerreno = true;
            accionActual = 0;
            accionFinalizadaOut.setValue(true);
              /*Ha finalizado inicializacion*/
           }
      else  //No hubo colision, se procede a avanzar
          {   yExtremo- = deltaTraslacion;
              //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
                        cinematicaPuntoExtremo();
           }
}
```

```
if( accionActual = = 1)  /*Eleva las extremidades*/
{
     if ( yExtremo + deltaTraslacion > =0 ) //ha llegado al lim superior
          {
           //System.out.println("Extremidad: evento LIMITE SUPERIOR#######");
           accionActual = 0;
           accionFinalizadaOut.setValue(true);                                    /*Ha finalizado accion*/
           colisionConTerreno = false;
           }
          else  //No se ha llegado, se procede a continuar
          {
              yExtremo + = deltaTraslacion;
              //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
                        cinematicaPuntoExtremo();
          }
}
if( accionActual = = 2)  /*Baja las extremidades, produciendo que el cuerpo del robot se mueva si se ha llegado al suelo*/
{
           if ( yExtremo-deltaTraslacion < = -1 ) //Ha llegado al limite inferior
          {
           //System.out.println("Extremidad: evento LIMITE INFERIOR#######");
           accionActual = 0;
           accionFinalizadaOut.setValue(true);                                    /*Ha finalizado accion*/
           }
          else  //No se ha llegado, se procede a continuar
          {
              yExtremo- = deltaTraslacion;
              //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
                        cinematicaPuntoExtremo();
          }
}
if( accionActual = = 3)  /*Avanza la extremidad a la derecha*/
{
       if ( zExtremo + deltaTraslacion > =0.5 ) //Ha llegado al limite superior
          {
           //System.out.println("Extremidad: evento LIMITE LATERAL DERECHO #######");
           accionActual = 0;
           accionFinalizadaOut.setValue(true);                                    /*Ha finalizado accion*/
           }

          else  //nose ha llegado, se procede a continuar
          {
              zExtremo + = deltaTraslacion;
              //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
            cinematicaPuntoExtremo();

          }
}

if( accionActual = = 4)  /*avanza la extremidad a la izquierda */
{

          if ( zExtremo-deltaTraslacion < = -0.5 ) //ha llegado al lim superior
          {
           //System.out.println("Extremidad: evento LIMITE LATERAL IZQUIERDO#######");
           accionActual = 0;
           accionFinalizadaOut.setValue(true);                                    /*ha finalizado accion*/
          }
         else  //nose ha llegado, se procede a continuar
          {
              zExtremo- = deltaTraslacion;
              //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
            cinematicaPuntoExtremo();
          }
}

if( accionActual = = 5)  /*estira la extremidad*/
{
          if ( xExtremo + deltaTraslacion > = 2.5 ) //ha llegado al lim frontal
          {
           //System.out.println("Extremidad: evento LIMITE FRONTAL #######");
           accionActual = 0;
           accionFinalizadaOut.setValue(true);                                    /*Ha finalizado accion*/
          }
          else  //No se ha llegado, continua
          {
              xExtremo + = deltaTraslacion;
              //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
            cinematicaPuntoExtremo();
          }
}

if( accionActual = = 6)  /*Retrae la extremidad*/
{
          if ( xExtremo-deltaTraslacion < = 2 ) //Ha llegado al lim TRASERO DEL ROBOT
          {
           //System.out.println("Extremidad: evento LIMITE TRASERO #######");
           accionActual = 0;
           accionFinalizadaOut.setValue(true);                                    /*Ha finalizado accion*/
          }
```

```
        else   //No se ha llegado, continua
        {
            xExtremo- = deltaTraslacion;
            //System.out.println("Extremidad: evento traslacionExtremoIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
          cinematicaPuntoExtremo();
        }
}
}

/*************************************************************************************/
 public void cinematicaPuntoExtremo( )
{  /*Obtiene la cinematica inversa para el punto extremo*/
    float[] tem1 = {0, 0, 1, rot1((float)(xExtremo),(float)(yExtremo),(float)(zExtremo) )};
        rot1Out.setValue(tem1);
    float[] tem2 = {0, 0, 1, rot2((float)(xExtremo),(float)(yExtremo),(float)(zExtremo) )};
        rot2Out.setValue(tem2);
    float[] tem3 = {0,1, 0, rot3((float)(xExtremo),(float)(yExtremo),(float)(zExtremo) )-1.57f};
        rot3Out.setValue(tem3);
}

/*****************************************************************/
/*Mantiene la posicion del extremo al desplazar al robot*/
public void extremoFijoAnteTraslacion(Event e, int direccion)
{
float xt,yt,zt,rt[] = {0,0,0,0} ,xr,yr,zr, re[] = {0,0,0,0},xe,ye,ze,deltaTraslacionX = 0,deltaTraslacionY = 0,deltaTraslacionZ = 0;
 float x,y,z;
      //System.out.println("Extremidad: evento traslacionRobotIn " + ((ConstSFVec3f)e.getValue()).getX() + " " + ((ConstSFVec3f)e.getValue()).getY() + "
      " + ((ConstSFVec3f)e.getValue()).getZ() ) ;
      //System.out.println("Extremidad: evento traslacionRobotIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
      x = ((ConstSFVec3f)e.getValue()).getX();
      y = ((ConstSFVec3f)e.getValue()).getY();
      z = ((ConstSFVec3f)e.getValue()).getZ();
      deltaTraslacionX = x-posicionAnteriorRobot.getX(); /*desplazamiento del robot X*/
      deltaTraslacionY = y-posicionAnteriorRobot.getY(); /*desplazamiento del robot Y*/
      deltaTraslacionZ = z-posicionAnteriorRobot.getZ(); /*desplazamiento del robot Z*/
      //System.out.println("Extremidad: deltas desp " + deltaTraslacionX + " " + deltaTraslacionY + " "+ deltaTraslacionZ );
      /*Pasar el punto extremo a coordenadas del robot*/
      rotacionYInicial.getValue(re);
       xr = (float) (xExtremo*(Math.cos(re[3]*re[1])) + (zExtremo*(Math.sin(re[3]*re[1])) ) + posicionInicial.getX()/escala;
       yr = (float) (yExtremo) + posicionInicial.getY()/escala;
       zr = (float) (zExtremo*(Math.cos(re[3]*re[1]))-(xExtremo*(Math.sin(re[3]*re[1])) ) + posicionInicial.getZ()/escala;
       //System.out.println("Extremidad: coord para el robot "+ xr + " " +  yr+ " "+  zr);
       /*Ahora se rota igual que al robot , dejando las coord como coord relativas al robot*/
       rotacionYActualRobot.getValue(rt);  /*Rotacion del robot */
       xt = (float) (xr*(Math.cos(rt[3]*rt[1])) + (zr*(Math.sin(rt[3]*rt[1])) ) + direccion*deltaTraslacionX;
       yt = (float) (yr) + direccion*deltaTraslacionY;
       zt = (float) (zr*(Math.cos(rt[3]*rt[1]))-(xr*(Math.sin(rt[3]*rt[1])) ) + direccion*deltaTraslacionZ;
       //System.out.println("Extremidad: punto extremo para el robot rotado" + xt + " " + yt + " " + zt );
       /*Finalmente se torna el punto a coordenadas de la extremidad para calcular la cinem inversa*/
       xe = (float) (xt*(Math.cos(-rt[3]*rt[1])) + (zt*(Math.sin(-rt[3]*rt[1])) )-posicionInicial.getX()/escala;
       ye = (float) (yt)-posicionInicial.getY()/escala;
       ze = (float) (zt*(Math.cos(-rt[3]*rt[1]))-(xt*(Math.sin(-rt[3]*rt[1])) )-posicionInicial.getZ()/escala;
       xExtremo =  (float) (xe*(Math.cos(-re[3]*re[1])) + (ze*(Math.sin(-re[3]*re[1])) );
       yExtremo =  ye;
       zExtremo =  (float) (ze*(Math.cos(-re[3]*re[1]))-(xe*(Math.sin(-re[3]*re[1])) );
       posicionAnteriorRobot = new ConstSFVec3f(posicionActualRobot.getX(), posicionActualRobot.getY(),posicionActualRobot.getZ());
       cinematicaPuntoExtremo( );
       //System.out.println("Extremidad: extremo despla por trasl " + xExtremo + " "+ yExtremo + " " + zExtremo );
 }

/*****************************************************************/
/*Mantiene la posición del extremo al rotar al robot*/
public void extremoFijoAnteRotacionY(Event e, int direccion)
{
float xt,yt,zt,rt[] = {0,0,0,0} ,xr,yr,zr, re[] = {0,0,0,0},xe,ye,ze,deltaRotacionY = 0;
float rotYActual,rotYAnterior, rtemp[] = {0,0,0,0};
       //System.out.println("Extremidad: evento rotRobotIn " + e.getValue());
       //System.out.println("Extremidad: evento rotacionyRobotIn " + xExtremo + " "+ yExtremo + " " + zExtremo );
       rotacionYActualRobot.getValue(rtemp);                    /*Obteniendo rotacion Y actual*/
       rotYActual = rtemp[1]*rtemp[3];
       rotacionYAnteriorRobot.getValue(rtemp);                  /*Obteniendo rotacion Y anterior*/
       rotYAnterior = rtemp[1]*rtemp[3];
       deltaRotacionY = rotYActual-rotYAnterior;                /*Rotacion en y del robot*/
       //System.out.println("Extremidad: deltas desp " + deltaRotacionY);
       /*Pasar el punto extremo a coordenadas del robot*/
            rotacionYInicial.getValue(re);

      xr = (float) (xExtremo*(Math.cos(re[3]*re[1])) + (zExtremo*(Math.sin(re[3]*re[1])) ) + posicionInicial.getX()/escala;
      yr = (float) (yExtremo) + posicionInicial.getY()/escala;
      zr = (float) (zExtremo*(Math.cos(re[3]*re[1]))-(xExtremo*(Math.sin(re[3]*re[1])) ) + posicionInicial.getZ()/escala;
      //System.out.println("Extremidad: coord para el robot "+ xr + " " +  yr+ " "+  zr);
      /*Ahora se rota igual que al robot , dejando las coord como coord relativas al robot*/
      rotacionYActualRobot.getValue(rt);  /*rotacion del robot */
      xt = (float) (xr*(Math.cos(rt[3]*rt[1] + direccion*deltaRotacionY)) + (zr*(Math.sin(rt[3]*rt[1] + direccion*deltaRotacionY)) );
      yt = (float) (yr);
      zt = (float) (zr*(Math.cos(rt[3]*rt[1] + direccion*deltaRotacionY))-(xr*(Math.sin(rt[3]*rt[1] + direccion*deltaRotacionY)) );
     //System.out.println("Extremidad: punto extremo para el robot rotado " + xt + " " + yt + " " + zt );
     /*Finalmente se torna el punto a coordenadas de la extremidad para calcular la cinem inversa*/
      xe = (float) (xt*(Math.cos(-rt[3]*rt[1])) + (zt*(Math.sin(-rt[3]*rt[1])) )-posicionInicial.getX()/escala;
      ye = (float) (yt)-posicionInicial.getY()/escala;
```

```
        ze = (float) (zt*(Math.cos(-rt[3]*rt[1]))-(xt*(Math.sin(-rt[3]*rt[1])) ))-posicionInicial.getZ()/escala;
        xExtremo = (float) (xe*(Math.cos(-re[3]*re[1])) + (ze*(Math.sin(-re[3]*re[1])) ));
        yExtremo = ye;
        zExtremo = (float) (ze*(Math.cos(-re[3]*re[1]))-(xe*(Math.sin(-re[3]*re[1])) ));
        float temp[] = {0,0,0,0};
        rotacionYActualRobot = (ConstSFRotation)e.getValue();
        rotacionYActualRobot.getValue(temp);
        rotacionYAnteriorRobot = new ConstSFRotation(temp[0],temp[1],temp[2],temp[3]);
        cinematicaPuntoExtremo( );
        //System.out.println("Extremidad: extremo cespla por trasl " + xExtremo + " "+ yExtremo + " " + zExtremo );
    }

    /*****************************************************************/
    /*Evalúa xyz para posible colision*/
    public float evaluaColision(float x, float y, float z)
    {
        float xt,yt,zt,rt[] = {0,0,0,0} ,xr,yr,zr, re[] = {0,0,0,0},xe,ye,ze;
        /*Pasar el punto extremo a coordenadas del robot*/
        rotacionYInicial.getValue(re);
        xr = (float) (x*(Math.cos(re[3]*re[1])) + (z*(Math.sin(re[3]*re[1])) ) + posicionInicial.getX()/escala;
        yr = (float) (y) + posicionInicial.getY()/escala;
        zr = (float) (z*(Math.cos(re[3]*re[1]))-(x*(Math.sin(re[3]*re[1])) ) + posicionInicial.getZ()/escala;
        /*A coordenadas del terreno*/
        rotacionYActualRobot.getValue(rt);  /*rotacion del robot + posicion actual*/
        xt = (float) (xr*(Math.cos(rt[3]*rt[1])) + (zr*(Math.sin(rt[3]*rt[1])) ) + posicionActualRobot.getX();
        yt = (float) (yr) + posicionActualRobot.getY(),
        zt = (float) (zr*(Math.cos(rt[3]*rt[1]))-(xr*(Math.sin(rt[3]*rt[1])) ) + posicionActualRobot.getZ();
        ye = topologia.altura(1,xt,zt);      /*se obtiene la altura de ese punto en el terreno*/
        //System.out.println("Extremidad: punto extremo evaluar colision" + xt + " " + yt + " " + zt + " punto enz terreno " + ye);
        return (yt - ye) ;                   /* Hay colision, yt del extremo ha quedado mas abajo que el terreno*/
    }

    /*****************************************************************/
    /*Las siguientes funciones obtienen la cinematica inversa para un punto especificado en forma relativa al estabón
    4 del robot*/

    public float rot3(float x, float y, float z)
    {
        return  (float) (Math.atan2((double)(x-l4), (double) z));
    }

    /*****************************************************************/
    public float rot1(float x, float y, float z)
    {
        float long2 ,long1 = 0,long3 = 0;
        float xp,yp,zp,x3,y3,z3;
        x3 = l3;
        y3 = h3;
        z3 = 0;
        xp = (float) (x3*(Math.cos(rot3(x,y,z))) + (z3*(Math.sin(rot3(x,y,z))) ) + l4;  /*punto sobre esl3*/
        yp = (float) (y3 + h4);
        zp = (float) (z3*(Math.cos(rot3(x,y,z)))-(x3*(Math.sin(rot3(x,y,z))) ));
        long1 = (float)Math.sqrt(l1*l1  +  h1*h1);
        long2 = l2;
        long3 =  (float) Math.sqrt( (xp-x)*(xp-x)  +  (yp-y)*(yp-y)  + (zp-z)*(zp-z) );
        return (float) (3.14  +  Math.acos(((long1*long1)+(long2*long2)-(long3*long3))/(2*long1*long2))  ) ;
    }
    /*****************************************************************/
    public float rot2(float x, float y, float z)
    {
        float long2 ,long1 = 0,long3 = 0;
        float xp,yp,zp,x3,y3,z3;
        x3 = l3;
        y3 = h3;
        z3 = 0;
        xp = (float) (x3*(Math.cos(rot3(x,y,z))) + (z3*(Math.sin(rot3(x,y,z))) ) + l4;  /*punto sobre esl3*/
        yp = (float) (y3 + h4);
        zp = (float) (z3*(Math.cos(rot3(x,y,z)))-(x3*(Math.sin(rot3(x,y,z))) ));
        long1 = (float)Math.sqrt(l1*l1  +  h1*h1);
        long2 = l2;
        long3 =  (float) Math.sqrt( (xp-x)*(xp-x)  +  (yp-y)*(yp-y)  + (zp-z)*(zp-z) );
        return (float) ( ( Math.acos(((long3*long3) + (long2*long2)-(long1*long1))/(2*long3*long2))  ) +  Math.atan2(y-yp,Math.sqrt((x-xp)*(x-xp) + (z-zp)*(z-zp))))
;
    }

}/*Clase manejadorExtremidad */

/*****************************************************************/
```

manejadorCamaraRobot.java

```
import java.io.*;
import java.net.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import Connection;
import java.lang.*;
```

```
/*****************************************************************/
public class manejadorCamaraRobot extends Script {
public float
deltaRotacion,rotacionAAActual,rotacionDIActual ;
public SFRotation rotacionAAOut;
public SFRotation rotacionDIOut;
public int accionActual = 0;      /*Selector para efectuar alguna accion en cada ciclo: 0 nada, 1 adel, 2 atras,  3 derecha, 4 izq,*/

/*****************************************************************/
public void initialize() {
/*Toma los campos especificados en vrml como field y los envia a sus respectivos extremidades como mensajes de inicializacion*/
            deltaRotacion = ((SFFloat) getField("deltaRotacion")).getValue();
            rotacionAAOut = (SFRotation) getEventOut("rotacionAAOut");
            rotacionDIOut = (SFRotation) getEventOut("rotacionDIOut");
}

/*****************************************************************/
 public void processEvent( Event e )
 {
     if ( e.getName().equals( "arribaCamara" ) ) {
        accionActual = 1;
       System.out.println( "Camara:  evento arribaCamara " );
      }
     if ( e.getName().equals( "abajoCamara" ) ) {
       accionActual = 2;
       System.out.println( "Robot: evento abajoCamara" );
      }
     if ( e.getName().equals("derechaCamara" ) ) {
       accionActual = 3;
       System.out.println( "Robot: evento derechaCamara" );
      }
     if ( e.getName().equals( "izquierdaCamara" ) ) {
       accionActual = 4;
       System.out.println( "Robot: evento izquierdaCamara" );
      }
      if ( e.getName().equals( "detenerCamara" ) ) {
       accionActual = 0;
       System.out.println( "Robot: evento detenerRobot" );
      }
       if ( e.getName().equals( "temporizador" ) ) {
       if(accionActual = = 1)
         {  float rotacionY[] = {0f,0f,0f,0f};
            rotacionAAOut.setValue(0, 0, 1,rotacionAAActual + = deltaRotacion);
            System.out.println( "Robot: evento cambio de rotacion AA actual " + rotacionAAOut );
         }
        if(accionActual = = 2)
         {  float rotacionY[] = {0f,0f,0f,0f};
            rotacionAAOut.setValue(0, 0, 1,rotacionAAActual- = deltaRotacion);
            System.out.println( "Robot: evento cambio de rotacion AA actual " + rotacionAAOut );
               }
        if(accionActual = = 3)
         {  float rotacionY[] = {0f,0f,0f,0f};
            rotacionDIOut.setValue(0, 1, 0,rotacionAAActual- = deltaRotacion);
            System.out.println( "Robot: evento cambio de rotacion DI actual " + rotacionDIOut );
               }
        if(accionActual = = 4)
         {  float rotacionY[] = {0f,0f,0f,0f};
            rotacionDIOut.setValue(0, 1, 0,rotacionAAActual + = deltaRotacion);
            System.out.println( "Robot: evento cambio de rotacion DI actual " + rotacionDIOut );
               }
      }
   }
}/*Clase manejadorCamaraRobot */
```

Topologia.java

```
import java.lang.*;

/*****************************************************************/
public class Topologia {
/*Método que genera la altura de un terreno definido en xz*/

    public float altura (int numero, float x, float z)
            {
              return 0;
              // (float)( (Math.sin(x*10f) + Math.sin(z*10f))/4 );
            }

}/*Clase Topologia */

/*****************************************************************/
```

Terreno.java

```
import vrml.*;
import vrml.node.*;
import vrml.field.*;
import Topologia;

/*****************************************************************/
public class Terreno extends Script {
   private int WIDTH = 0;
   private int DEPTH = 0;
   private SFInt32 width;
   private SFInt32 depth;
   private MFFloat height;
   float[] heightFields = null;
   private  SFFloat escalaZ;
   public Topologia topologia;

/*****************************************************************/
public void initialize() {
      System.err.println( "init'ing wave Script node" );
      height = (MFFloat)getEventOut( "height" );
      escalaZ = (SFFloat) getField( "escalaZ" );
      width = (SFInt32)getField( "xDimension" );
      depth = (SFInt32)getField( "zDimension" );
      WIDTH = width.getValue();
      DEPTH = depth.getValue();
      heightFields = new float[ WIDTH * DEPTH ];
      topologia = (Topologia) new Topologia();
      for ( int blah = 0 ; blah < WIDTH * DEPTH ; blah++ ) {
         heightFields[blah] = 0;
       }
       for ( int x = 0 ; x < WIDTH ; x++ ) {
            for ( int z = 0 ; z < DEPTH ; z++ ) {
              float x1 = (x-WIDTH/2), z1 = (z-DEPTH/2);
              heightFields[(z * DEPTH) + x] = topologia.altura(0,x1,z1);
             }
         };
       height.setValue( heightFields);
    }
} /*Clase Terreno*/

/*****************************************************************/
```

manejadorBoton.java

```
import java.io.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.lang.*;

/*****************************************************************/
public class manejadorBoton extends Script {
SFBool activo;
   public void initialize() {
      activo = (SFBool) getEventOut("activo");
    }

/*****************************************************************/
public void processEvent( Event e ) {
      activo.setValue(true);
   }

}/* Fin de clase manejadorBoton*/

/*****************************************************************/
```

manejadorSlider.java

```
import java.io.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.lang.*;

/*****************************************************************/
public class manejadorSlider extends Script {
SFFloat nivelOut;
SFVec3f posicionApuntadorOut;

/*****************************************************************/
public void initialize() {
      nivelOut = (SFFloat) getEventOut("nivelOut");
      posicionApuntadorOut = (SFVec3f) getEventOut("posicionApuntadorOut");
   }
```

```
/*****************************************************************/
public void processEvent( Event e ) {
      if ( e.getName().equals( "posicionApuntadorIn" ) ) {
      float  tempY = ((ConstSFVec3f)e.getValue()).getY();
       if(tempY<0) tempY=0f;
       if(tempY>1) tempY=1f;
       nivelOut.setValue(tempY);
       posicionApuntadorOut.setValue(0,tempY,0);
        }
   }
}/* Clase controladorSlider*/

/*****************************************************************/
```

ServidorDeCinematica.java

```
import java.io.IOException;
import java.io.PrintStream;
import java.net.ServerSocket;

/*****************************************************************/
public class ServidorDeCinematica extends Thread
{
   public static final int DEFAULT_PORT = 10000;
   protected int port;
   protected ServerSocket listen_socket;
}

/*****************************************************************/
public static void fail(Exception e, String string)
 {
      System.err.println(string + ": " + e);
      System.exit(1);
   }

/*****************************************************************/
/*Crea un socket para atender las peticiones de los clientes*/
public ServidorDeCinematica(int i)
   {
      if (i == 0) i = DEFAULT_PORT;
            port = i;
      try
      {
         listen_socket = new ServerSocket(port);
      }
      catch (IOException e)
      {
         fail(e, "No se puede crear socket en el puerto " + port );
      }
      System.out.println("Servidor: escuchando en puerto " + port );
      start();
   }

/*****************************************************************/
/*Establece comunicación con el cliente a través de un socket, redirecciona la información a un objeto ConeccionCinematica */
   public void run()
   {
      while (true)
      {
         try
         {
            java.net.Socket socket = listen_socket.accept();
            System.out.println("Servidor:  robot conectado ..... ");
            new ConeccionCinematica(socket);
         }
         catch (IOException e)
         {
            fail(e, "Servidor: error en la coneccion del socket");
         }
      }
   }

/*****************************************************************/


   public static void main(String astring[])
   {
            new ServidorDeCinematica(i);
   }

}/*Clase  ServidorDeCinematica */

/*****************************************************************/
```

ConeccionCinematica.java

```
import java.io.*;
import java.net.Socket;
import java.lang.*;

/*****************************************************************/
/*Calcula en forma remota la cinematica inversa de un robot de 3 grados de libertad*/
class ConeccionCinematica extends Thread
{
public  double tras[] = {0,0,0},x = 0,y = 0,z = 0,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4;
public float rot1 = 0, rot2 = 0, rot3 = 0, rtemp = 0;
public float escala = 100;
float l4 = 750/escala;                                  /*Longitud del eslabón 4*/
float h4 = 180/escala;                                  /*Altura del eslabón 4*/
float l3 = 170/escala;                                  /*Longitud del eslabón 3*/
float h3 = 150/escala;                                  /*Altura del eslabón 3*/
float l2 = 710/escala;                                  /*Longitud del eslabón 2*/
float h2 = 0;                                           /*Altura del eslabón 2*/
float l1 = 1800/escala;                                 /*Longitud del eslabón 1*/
float h1 = 0;                                           /*Altura del eslabón 1*/
float altura = 600/escala;
float xInit = 0, yInit = altura, zInit = 0;             /*Traslación de los ejes del escenario hacia el eje del robot*/
float rotXRobot = 0,rotYRobot = 0, rotZRobot = 0;       /*Rotaciones del eje del robot con referencia a los ejes del escenario*/
float rotYExtremidad = 0;                               /*Rotación para posicionar cada extremidad*/
protected Socket robot;
protected DataInputStream datosDelRobot;
protected PrintStream datosAlRobot;

/*****************************************************************/
public ConeccionCinematica(Socket socket)
  {
    robot = socket;
    try
    {
      IOException e2;
      datosDelRobot = new DataInputStream(robot.getInputStream());
      datosAlRobot = new PrintStream(robot.getOutputStream());
      start();
    }
    catch (IOException e2)
    {
      try  {  robot.close();  } catch (IOException e1) {  }
      System.err.println("Coneccion : error en la coneccion del socket ");
    }
    return;
  }

/*****************************************************************/
public void run()
  {
    String string = "hola";
    try {
     do {
          if (datosDelRobot.available() > 0)
          {
            float x, y, z, rotacion1, rotacion2,rotacion3;
            string = datosDelRobot.readLine();
            System.out.println("Servidor: peticion del robot " + string);
            x = (new Float( string.substring(0,string.indexOf(" ")-1 ) )).floatValue();
            y = (new Float( string.substring(string.indexOf(" ")+1,string.lastIndexOf(" ")-1 ) )).floatValue();
            z = (new Float( string.substring(string.lastIndexOf(" ")+1) ) ).floatValue();
            // System.out.println("Servidor: respuesta "x + " " + y + " " + z);
           datosAlRobot.println(rot1(x,y,z) + " " + rot2(x,y,z) + " " + rot3(x,y,z));      /*La respuesta es el cálculo de la cinematica*/
          }
     } while (string != null);
    } catch (IOException e1)  {  }
    finally { try  {  robot.close(); } catch (IOException e2) { } }
    return;
 }

/*****************************************************************/
/*Métodos para la obtencion de la cinematica*/

public float rot1(float x, float y, float z)            /*Rotación eslabón 1*/
{
    float long2 ,long1 = 0,long3 = 0;
    float xp,yp,zp;
    x3 = l3;
    y3 = h3;
    z3 = 0;
    xp = (float) (x3*(Math.cos(rot3)) + (z3*(Math.sin(rot3)) ) + l4;  /*Punto base sobre esl3*/
    yp = (float) (y3 + h4);
    zp = (float) (z3*(Math.cos(rot3))-(x3*(Math.sin(rot3)) );
    long1 = (float)Math.sqrt(l1*l1 + h1*h1);
    long2 = l2;
    long3 =  (float) Math.sqrt( (xp-x)*(xp-x) + (yp-y)*(yp-y) + (zp-z)*(zp-z) );
    return (float) (3.14 + Math.acos(((long1*long1) + (long2*long2)-(long3*long3))/(2*long1*long2)) ) ;
}
```

```
/*****************************************************************/
public float rot2(float x, float y, float z)              /*Rotación eslabón 2*/
{
      float long2 ,long1 = 0,long3 = 0;
      float xp,yp,zp;
      x3 = l3;
      y3 = h3;
      z3 = 0;
      xp = (float) (x3*(Math.cos(rot3)) + (z3*(Math.sin(rot3)) )) + l4;  /*punto sobre esl3*/
      yp = (float) (y3 + h4),
      zp = (float) (z3*(Math.cos(rot3))-(x3*(Math.sin(rot3)) ));
      long1 = (float)Math.sqrt(l1*l1  +  h1*h1);
      long2 = l2;
      long3 =  (float) Math.sqrt( (xp-x)*(xp-x)  +  (yp-y)*(yp-y)  + (zp-z)*(zp-z) );
      return (float) ( ( Math.acos(((long3*long3) + (long2*long2)-(long1*long1))/(2*long3*long2))  )  +  Math.atan2(y-yp,Math.sqrt((x-xp)*(x-xp) + (z-zp)*(z-zp))))
;
}

/*****************************************************************/
public float rot3(float x, float y, float z)              /*Rotación eslabón 3*/
{
      return  (float) (Math.atan2((double)(x-l4), (double) z));
}

}/*Clase ConeccionCinematica */

/*****************************************************************/
```

Cinematica.java

```
import java.io.*;
import java.net.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.lang.*;
import java.lang.Math;
import Topologia;

/*****************************************************************/
public class Cinematica extends Script {
public SFNode pCinemInversa;
public SFNode pCompCinemInversa;
public SFNode t1;
public SFNode t2;
public SFNode t3;
public SFNode puntero;
public SFNode extremidad;
public SFRotation r1;
public SFRotation r2;
public SFRotation r3;
public SFVec3f traslacion;
public SFVec3f xyzpuntero;
public SFRotation rotacion;
public  double tras[] = {0,0,0},x = 0,y = 0,z = 0,x1,y1,z1,x2,y2,z2,x3,y3,z3,x4,y4,z4;
public float rot1 = 0, rot2 = 0, rot3 = 0, rtemp = 0;
public float escala = 100;
public Topologia topologia;
float l4 = 750/escala;                          /*Longitud del eslabon 4*/
float h4 = 180/escala;                          /*Altura del eslabon 4*/
float l3 = 170/escala;                          /*Longitud del eslabon 3*/
float h3 = 150/escala;                          /*Altura del eslabon 3*/
float l2 = 710/escala;                          /*Longitud del eslabon 2*/
float h2 = 0;                                   /*Altura del eslabon 2*/
float l1 = 1800/escala;                         /*Longitud del eslabon 1*/
float h1 = 0;                                   /*Altura del eslabon 1*/
float altura = 600/escala;
float xInit = 0, yInit = altura, zInit = 0;     /*Traslacion de los ejes del escenario hacia el eje del robot*/
float rotXRobot = 0,rotYRobot = 0, rotZRobot = 0;  /*Rotaciones del eje del robot con referencia a los ejes del escenario*/
float rotYExtremidad = 0;                       /Rotacion para posicionar cada extremidad*
DataInputStream flujoDelServidor = null;        /*Flujo del servidor*/
PrintStream flujoAlServidor = null;             /*Flujo al servidor*/
String mensajeDelServidor = "";                 /*Mensajes del servidor*/
String mensajeAlServidor = "";                  /*Mensajes al servidor*/
Socket socket = null;

/*****************************************************************/
public void initialize() {
      puntero = (SFNode) getField("puntero");
      xyzpuntero   = (SFVec3f) (((Node)puntero.getValue()).getExposedField("translation") );
      extremidad = (SFNode) getField("extremidad");
      r1 = (SFRotation) (((Node)extremidad.getValue()).getExposedField("roteslabon1f") );
      r2 = (SFRotation) (((Node)extremidad.getValue()).getExposedField("roteslabon2f") );
      r3 = (SFRotation) (((Node)extremidad.getValue()).getExposedField("roteslabon3f") );
      float[] xyz = {(float)x,(float)z,(float)-y1};
      xyzpuntero.setValue(xyz);
      topologia = (Topologia) new Topologia();
      try {
            socket = new Socket("uxmcc1", 10000);
```

```
            flujoDelServidor = new DataInputStream(socket.getInputStream());
            flujoAlServidor = new PrintStream(socket.getOutputStream());
            System.out.println( "Robot: activo envia peticiones");
        } catch (Exception e) {
                    e.printStackTrace();
                     System.out.println("Robot: problemas en la coneccion de sockets");
                }
    }

/**************************************************************/
public void processEvent( Event e ) {
    String string = "";                         /*Respuesta del servidor*/
    if ( e.getName().equals( "cuerpoRobotTranslacion" ) ) {
        System.out.println( "Traslacion robot " + ((ConstSFVec3f) e.getValue()).getX() + " " +
                ((ConstSFVec3f) e.getValue()).getY() + " " + ((ConstSFVec3f) e.getValue()).getZ());
        xInit = ((ConstSFVec3f) e.getValue()).getX();
        yInit = ((ConstSFVec3f) e.getValue()).getZ() + altura;
        zInit = ((ConstSFVec3f) e.getValue()).getY();
    }
    if ( e.getName().equals( "posicionXZ" ) ) {
        x = ((ConstSFVec3f) e.getValue()).getX();
        y = ((ConstSFVec3f) e.getValue()).getZ();
        z = ((ConstSFVec3f) e.getValue()).getY();
    }

    if ( e.getName().equals( "rotEslabon1f" ) ) {
        float temp[] = {0,0,0,0};
        ((ConstSFRotation)e.getValue()).getValue(temp);
        rot1 = temp[3]*temp[2];
     }

    if ( e.getName().equals( "rotEslabon2f" ) ) {
        float temp[] = {0,0,0,0};
        ((ConstSFRotation)e.getValue()).getValue(temp);
        rot2 = temp[3]*temp[2];
     }

    if ( e.getName().equals( "rotEslabon3f" ) ) {
        float temp[] = {0,0,0,0};
        ((ConstSFRotation)e.getValue()).getValue(temp);
        rot3 = temp[3]*temp[1];
     }

/*Obtención de los angulos para los eslabones*/
    float y1 = topologia.altura(1,(float)x,(float)z);  /*superficie a seguir*/
    x3 = x-xInit;
    y3 = y1-yInit;
    z3 = z-zInit;
    x4 = (float) (x3*(Math.cos(rotYExtremidad)) + (z3*(Math.sin(rotYExtremidad)) ));
    y4 = (float) (y3);
    z4 = (float) (z3*(Math.cos(rotYExtremidad))-(x3*(Math.sin(rotYExtremidad)) ));

/*Rotación de los eslabones*/
    flujoAlServidor.println( x4 + " " + y4 + " " + z4);                 /*Petición de cinematica para x y z*/
    System.out.println( "Robot: peticion " + x4 + " " + y4 + " " + z4 );

  try
   {
        while(! (flujoDelServidor.available() > 0) );                   /*Espera respuesta*/
        string = flujoDelServidor.readLine();
        System.out.println("Robot: respuesta del servidor " + string );
   }
   catch (IOException e1) {  System.out.println("Robot: error en la lectura del socket ");  }
   float[] tem1 = {0, 0, 1, (new Float(  string.substring(0,string.indexOf(" ")-1 ) )).floatValue()};
   r1.setValue(tem1);
   float[] tem2 = {0, 0, 1, (new Float(  string.substring(string.indexOf(" ")+1,string.lastIndexOf(" ")-1 ) )).floatValue()};
   r2.setValue(tem2);
   float[] tem3 = {0,1, 0, (new Float(  string.substring(string.lastIndexOf(" ")+1) ) ).floatValue()-1.57f};
   r3.setValue(tem3);

/*Movimiento del puntero*/

   float[] xyz = {(float)x,(float)z,(float)-y1};
   xyzpuntero.setValue(xyz);

}

/**************************************************************/
public void shutdown()
{
 try
   {
  socket.close();
   }
   catch (IOException e1) {System.out.println("Robot: error en la lectura del socket ");}

}

}/*Clase Cinematica */
```

/***/

manejadorExtremidadRemotaServidor.java

```
import java.io.*;
import java.net.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.lang.*;

/*****************************************************************/
public class manejadorExtremidadRemotaServidor extends Script {
public float
deltaRotacion,limiteSuperior1,limiteSuperior2,limiteSuperior3;
public float
limiteInferior1,limiteInferior2,limiteInferior3;
public SFRotation rotacionEslabon1, rotacionEslabon2, rotacionEslabon3;
public SFRotation incrementarEslabon1,decrementarEslabon1,detenerEslabon1;
public SFRotation incrementarEslabon2,decrementarEslabon2,detenerEslabon2;
public SFRotation incrementarEslabon3,decrementarEslabon3,detenerEslabon3;
public boolean enviar = false;
public int accionActual1 = 0,accionActual2 = 0,accionActual3 = 0;      /*Selector para efectuar alguna accion en cada ciclo:
                                                                        0 nada, 1 adel, 2 atras,  3 derecha, 4 izq.*/
protected int puerto;
protected ServerSocket listen_socket;
protected Socket socket;
protected DataInputStream mensajeDelControl;
protected PrintStream mensajeAlControl;

/*****************************************************************/
public void initialize() {
/*Toma los campos especificados en vrml como field y los envia a sus respectivos extremidades
como mensajes de inicializacion*/

    puerto = ((SFInt32) getField("puerto")).getValue();
    deltaRotacion = ((SFFloat) getField("deltaRotacion")).getValue();
    limiteSuperior1 = ((SFFloat) getField("limiteSuperior1")).getValue();
    limiteSuperior2 = ((SFFloat) getField("limiteSuperior2")).getValue();
    limiteSuperior3 = ((SFFloat) getField("limiteSuperior3")).getValue();
    limiteInferior1 = ((SFFloat) getField("limiteInferior1")).getValue();
    limiteInferior2 = ((SFFloat) getField("limiteInferior2")).getValue();
    limiteInferior3 = ((SFFloat) getField("limiteInferior3")).getValue();
    rotacionEslabon1 = (SFRotation) getEventOut("rotacionEslabon1");
    rotacionEslabon2 = (SFRotation) getEventOut("rotacionEslabon2");
    rotacionEslabon3 = (SFRotation) getEventOut("rotacionEslabon3");
    System.out.println("Servidor: esperando coneccion del modulo de control");
    try{
        listen_socket = new ServerSocket(puerto);
        java.net.Socket socket = listen_socket.accept();
        mensajeDelControl = new DataInputStream(socket.getInputStream());
        mensajeAlControl = new PrintStream(socket.getOutputStream());
        System.out.println("Servidor:  control de usuario conectado ..... ");
      } catch (IOException e){System.out.println("Servidor: error en la coneccion del socket");}
}

/*****************************************************************/
public void processEvent( Event e )
{
    if ( e.getName().equals( "temporizador" ) )
     {
        try {
            if (mensajeDelControl.available() > 0 )
             {                                      /*espera respuesta*/
                String string = mensajeDelControl.readLine();
                if(string.equals("S") )
                enviar = true;
                else
                  {
                    float temp = (new Float(string)).floatValue();
                    accionActual1 = (int) temp%10;
                    accionActual2 = (int) (temp%100)/10;
                    accionActual3 = (int)  (temp/100);
                    System.out.println("Servidor: comando del control de usuario " + string +
                            " " + accionActual1  + " " + accionActual2 + " " + accionActual3);
                  }
             }
          } catch (IOException e1) {System.out.println("Servidor: error en la lectura del socket ");}

    if(accionActual1 == 1)
      {
        float rotacion[] = {0f,0f,0f,0f};
        rotacionEslabon1.getValue(rotacion);
        if((rotacion[3] + = deltaRotacion) < limiteSuperior1 )
        {
          rotacionEslabon1.setValue(0, 0, 1,rotacion[3] + = deltaRotacion);
          //System.out.println( "Robot: evento cambio de rotacion 1 " + rotacionEslabon1 );
        }
      }
```

```
    if(accionActual1 = = 2)
        {
           float rotacion[] = {0f,0f,0f,0f};
           rotacionEslabon1.getValue(rotacion);
           if((rotacion[3]- = deltaRotacion) > limiteInferior1 )
           {
                 rotacionEslabon1.setValue(0, 0, 1,rotacion[3]- = deltaRotacion);
                  //System.out.println( "Robot: evento cambio de rotacion 1 " + rotacionEslabon1 );
            }
        }

    if(accionActual2 = = 1)
        {
           float rotacion[] = {0f,0f,0f,0f};
           rotacionEslabon2.getValue(rotacion);
           if((rotacion[3] + = deltaRotacion) < limiteSuperior2 )
           {
                  rotacionEslabon2.setValue(0, 0, 1,rotacion[3] + = deltaRotacion);
                 //System.out.println( "Robot: evento cambio de rotacion 2 " + rotacionEslabon2 );
           }
        }

    if(accionActual2 = = 2)
        {
           float rotacion[] = {0f,0f,0f,0f};
           rotacionEslabon2.getValue(rotacion);
           if((rotacion[3]- = deltaRotacion) > limiteInferior2 )
           {
                 rotacionEslabon2.setValue(0, 0, 1,rotacion[3]- = deltaRotacion);
                 //System.out.println( "Robot: evento cambio de rotacion 2 " + rotacionEslabon2 );
           }
         }

    if(accionActual3 = = 1)
        {
           float rotacion[] = {0f,0f,0f,0f};
           rotacionEslabon3.getValue(rotacion);
           if((rotacion[3] + = deltaRotacion) < limiteSuperior3 )
           {
                 rotacionEslabon3.setValue(0, 1, 0,rotacion[3] + = deltaRotacion);
                 //System.out.println( "Robot: evento cambio de rotacion 3 " + rotacionEslabon3 );
           }
        }
    if(accionActual3 = = 2)
        {
           float rotacion[] = {0f,0f,0f,0f};
           rotacionEslabon3.getValue(rotacion);
           if((rotacion[3]- = deltaRotacion) > limiteInferior3 )
           {
                 rotacionEslabon3.setValue(0, 1, 0,rotacion[3]- = deltaRotacion);
                 System.out.println( "Robot: evento cambio de rotacion 3 " + rotacionEslabon3 );
           }
        }
            /*Envía el actual estatus del robot al visualizador del usuario*/
    if(enviar)
        {
           float rotacion1[] = {0f,0f,0f,0f},rotacion2[] = {0f,0f,0f,0f},rotacion3[] = {0f,0f,0f,0f};
                       rotacionEslabon1.getValue(rotacion1);
                       rotacionEslabon2.getValue(rotacion2);
                       rotacionEslabon3.getValue(rotacion3);
           mensajeAlControl.println( rotacion1[3]*rotacion1[2] + " " + rotacion2[3]*rotacion2[2] + " " + rotacion3[3]*rotacion3[1] );
           System.out.println(rotacion1[3]*rotacion1[2] + " " + rotacion2[3]*rotacion2[2] + " " + rotacion3[3]*rotacion3[1] );
           enviar = false;
        }
     }
}

/*****************************************************************/

public void shutdown() {
     try
        {
           socket.close();
        } catch (IOException e1) {System.out.println("Robot: error en la lectura del socket ");}
}
}/*Clase manejadorExtremidadRemotaServidor */

/*****************************************************************/
```

manejadorExtremidadRemotaCliente.java

```
import java.io.*;
import java.net.*;
import vrml.*;
import vrml.field.*;
import vrml.node.*;
import java.lang.*;

/*****************************************************************/
```

```
public class manejadorExtremidadRemotaCliente extends Script {
public SFRotation rotacionEslabon1, rotacionEslabon2, rotacionEslabon3;
public SFRotation incrementarEslabon1,decrementarEslabon1,detenerEslabon1;
public SFRotation incrementarEslabon2,decrementarEslabon2,detenerEslabon2;
public SFRotation incrementarEslabon3,decrementarEslabon3,detenerEslabon3;
public int accionActual1 = 0,accionActual2 = 0,accionActual3 = 0;      /*Selector para efectuar alguna accion en cada ciclo: 0 nada,
                                                                        1 adel, 2 atras,  3 derecha, 4 izq,*/
public int accionAnterior1 = 0,accionAnterior2 = 0,accionAnterior3 = 0;
DataInputStream flujoDelServidor = null;          /*Flujo del servidor*/
PrintStream flujoAlServidor = null;               /*fFujo al servidor*/
String mensajeDelServidor = "";                   /*Mensajes del servidor*/
String mensajeAlServidor = "";
String servidor;
int puerto;                                       /*Mensajes al servidor*/
Socket socket = null;

/*****************************************************************/
public void initialize() {
     servidor = ((SFString) getField("servidor")).getValue();
     puerto = ((SFInt32) getField("puerto")).getValue();
     rotacionEslabon1 = (SFRotation) getEventOut("rotacionEslabon1");
     rotacionEslabon2 = (SFRotation) getEventOut("rotacionEslabon2");
     rotacionEslabon3 = (SFRotation) getEventOut("rotacionEslabon3");
    try {
              socket = new Socket("uxmcc1", puerto);
              flujoDelServidor = new DataInputStream(socket.getInputStream());
              flujoAlServidor = new PrintStream(socket.getOutputStream());
              System.out.println( "Control: activo envia comandos");
      } catch (Exception e) {
                    e.printStackTrace();
                    System.out.println("Control: problemas en la coneccion de sockets");
                }
}

/*****************************************************************/
public void processEvent( Event e )
{
     if ( e.getName().equals( "incrementarEslabon1" ) ) {
          accionActual1 = 1;
          System.out.println( "Control:  evento incrementarEslabon1 " );
      }

     if ( e.getName().equals( "decrementarEslabon1" ) ) {
          accionActual1 = 2;
          System.out.println( "Control:  evento decrementarEslabon1 " );
      }

     if ( e.getName().equals( "detenerEslabon1" ) ) {
          accionActual1 = 0;
          System.out.println( "Control:  evento detenerEslabon1 " );
      }

     if ( e.getName().equals( "incrementarEslabon2" ) ) {
          accionActual2 = 1;
          System.out.println( "Control:  evento incrementarEslabon2 " );
      }

     if ( e.getName().equals( "decrementarEslabon2" ) ) {
          accionActual2 = 2;
          System.out.println( "Control:  evento decrementarEslabon2 " );
      }

     if ( e.getName().equals( "detenerEslabon2" ) ) {
          accionActual2 = 0;
          System.out.println( "Control:  evento detenerEslabon2 " );
      }

     if ( e.getName().equals( "incrementarEslabon3" ) ) {
          accionActual3 = 1;
          System.out.println( "Control:  evento incrementarEslabon3 " );
      }

     if ( e.getName().equals( "decrementarEslabon3" ) ) {
          accionActual3 = 2;
          System.out.println( "Control:  evento decrementarEslabon3 " );
      }

     if ( e.getName().equals( "detenerEslabon3" ) ) {
          accionActual3 = 0;
          System.out.println( "Control:  evento detenerEslabon3 " );
      }

     if ( e.getName().equals( "temporizador" ) )
      {
          if( accionActual1 != accionAnterior1 || accionActual2 != accionAnterior2 || accionActual3 != accionAnterior3)
                {                   flujoAlServidor.println(accionActual1 + accionActual2 * 10 + accionActual3 * 100 );
                                    accionAnterior1 = accionActual1;
                                    accionAnterior2 = accionActual2;
                                    accionAnterior3 = accionActual3;
                }
```

```
        try {
                                if (flujoDelServidor.available() > 0 )                    /*Espera respuesta*/
                                 {
                                        String string = flujoDelServidor.readLine();
                                        System.out.println("Control: respuesta del servidor " + string );

                                        rotacionEslabon1.setValue(0, 0, 1,(new Float( string.substring(0,string.indexOf(" ")-1 ) )).floatValue());
                                        rotacionEslabon2.setValue(0, 0, 1,(new Float( string.substring(string.indexOf(" ") + 1,
                                                        string.lastIndexOf(" ")-1 ) )).floatValue());
                                         rotacionEslabon3.setValue(0, 1, 0,(new Float( string.substring(string.lastIndexOf(" ")
                                                        + 1) ) ).floatValue());
                                 }
            }
          catch (IOException e1) {System.out.println("Robot: error en la lectura del socket ");}
           flujoAlServidor.println("S"); /*envia mensaje de que esta listo para recibir datos*/
     }

}
/*****************************************************************/
public void shutdown()
{
 try
    {   socket.close(); }  catch (IOException e1) {System.out.println("Robot: error en la lectura del socket ");}
}

} /*Clase manejadorExtremidadRemotaCliente */

/*****************************************************************/
```

Apéndice E

Código fuente del controlador del robot

clases.h

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include <stdlib.h>

/*contiene la declaración de todas las clases del sistema*/
/*************************************************************************/
class Robot{
public:
        class Dispositivo extremidad1,extremidad2;
        class Dispositivo extremidad3,extremidad4;
        class Dispositivo extremidad5,extremidad6;
        class Dispositivo camara;
        class ManejadorPuerto  manejadorPuerto;
        int numeroDispositivos;

void inicializar(void);
void adelante(void);
void atras(void);
void rotI(void);
void rotD(void);
};

/*************************************************************************/

class Dispositivo{
public:
        int identificador;
        int rotActual1,rotActual2,rotActual3;
        int rotFinal1,rotFinal2,rotFinal3;
        int limSuperiorRot1,limSuperiorRot2,limSuperiorRot3;
        int limInferiorRot1,limInferiorRot2,limInferiorRot3;
        int sensorObstaculo;
        int localizacionDeObstaculo;        /*Localizacion: 0 ninguno, 1 arriba, 2 abajo, 3 izquierda 4 derecha, 5 interior,6 exterior*/
        int accion;                         /*Accion desarrollar, 0 ninguno, 1 arriba, 2 abajo, 3 roti, 4 rotd, 5 recoger, 6  estirar*/
        int status;   /*0 terminado (fin de comando basico), 1 terminado por usuario 2 terminado por obstaculo, 3 ejecutando */
        int delta;    /*Razon de cambio en cada iteración para las rotaciones*/
/*Solo se puede ejecutar una accion por vez, si se requieren movimientos combinados se definen como un comando basico y definir una localizacion de
obstaculo para cada caso, solo se puede hacer uso de los comandos en forma secuencial y no combinado, ademas debera esperarse a terminar un comando para
comenzar  otro */

/*Comandos básicos*/
void inicializar(int id, float r1, float r2,float r3);
void inicia(float r1,float r2,float r3);
void mueve(float r1,float r2,float r3);
void arriba();
void abajo();
void rotI();
void rotD();
void estirar();
void recoger();
void detener(int causa);
void arriba(float delt);
void abajo(float delt);
void rotI(float delt);
void rotD(float delt);
void estirar(float delt);
void recoger(float delt);
void localizarObstaculo();  /*Localiza la procedencia del obstaculo,debe llamarse antes de detener(int causa) para no perder la variable de accion*/
void mueve(int motor,int direccion);    /*Mueve el actuador "motor" en la dirección indicada*/

};

/*************************************************************************/

class ManejadorPuerto{
public:
        unsigned char senalActual[8], senalAnterior[8],senalDeCambio,j;
        unsigned char vectorSalida[8],vectorSalidaAnterior[8];
        class Dispositivo *dispositivo[8];  /*apunt a un arreglo de dispositivos*/
        int portSalidaDatos ,port1EntradaDatos,port2EntradaDatos,portSelect,value;
        int numeroDispositivos;
```

```
void inicializar( class Dispositivo *e1,class Dispositivo *e2,class Dispositivo *e3,
                  class Dispositivo *e4,class Dispositivo *e5,class Dispositivo *e6,class Dispositivo *c );
void inicializarDispositivos(int numDispositivos);
int inicializaDispositivo(int numDisp,int r1,int r2,int r3);
unsigned char leeDispositivo(int dispositivo);
void  interpretaSenales(void);
};
```

Controlador.cpp

```
#include <stdio.h>
#include <dos.h>
#include <conio.h>
#include "clases.h"

/*Controlador de prueba para una extremidad */

void main(void)
{

  Robot robot;                                  /*Instancia del robot*/
  int tecla=0,j,motor,fase=1;
  robot.inicializar();                  /*Inicializa los componentes del robot*/
for(;;j++)
 {
 robot.manejadorPuerto.interpretaSenales();     /*Se comunica con la interfaz electrónica para
                        llevar a cabo el control de la extremidad */

        if( kbhit() )
        {
                tecla=getch();
                if (tecla==49) motor=1;
                if (tecla==50) motor=2;
                if (tecla==51) motor=3;
                if (tecla=='r')  robot.inicializar();
                if (tecla==72) {robot.extremidad1.mueve(motor,1);
                delay (100);}

                if (tecla==80)
                {
                        robot.extremidad1.mueve(motor,2/*atras*/);
                        delay (100);
                }

                if (tecla==13)
                {
                        robot.extremidad1.mueve(motor,0/*detener*/);delay (100);
                }


        }
/*Secuencia de movimientos preestablecidos*/
        if( robot.extremidad1.rotActual1==robot.extremidad1.rotFinal1
           &&robot.extremidad2.rotActual1==robot.extremidad1.rotFinal2
           &&robot.extremidad3.rotActual1==robot.extremidad1.rotFinal3)
           {
                if (fase==1) {delay(40); robot.extremidad1.mueve(20,0,0);fase=2;}
                else if(fase==2){delay(40); robot.extremidad1.mueve(0,20,0);fase=3;}
                else if(fase==3){delay(40); robot.extremidad1.mueve(0,0,20);fase=4;}
                else if(fase==4){delay(40); robot.extremidad1.mueve(-20,0,0);fase=5;}
                else if(fase==5){delay(40); robot.extremidad1.mueve(0,-20,0);fase=6;}
                else if(fase==6){delay(40); robot.extremidad1.mueve(0,0,-20);fase=1;}
           }
      }
}

/*****************************************************************/
```

Robot.cpp

```
#include "clases.h"

/*Inicializa las entidades que representan a los dinpositivos pertenecientes al robot*/
void Robot::inicializar()
{
 extremidad1.inicializar(1,0,0,0);
 extremidad2.inicializar(2,0,0,0);
 extremidad3.inicializar(3,0,0,0);
 extremidad4.inicializar(4,0,0,0);
 extremidad5.inicializar(5,0,0,0);
 extremidad6.inicializar(6,0,0,0);
 camara.inicializar(7,0,0,0);
 manejadorPuerto.inicializar(&extremidad1,&extremidad2,&extremidad3,&extremidad4,
                             &extremidad5,&extremidad6,&camara);
manejadorPuerto.inicializarDispositivos(numeroDispositivos=7);
}
/*****************************************************************/
```

Dispositivo.cpp

```
#include "d:\tesis\puerto\clases.h"

void Dispositivo::inicializar(int id, float r1, float r2,float r3)
{
        limSuperiorRot1 = 0; limSuperiorRot2 = 0; limSuperiorRot3 = 0;
        limInferiorRot1 = 0; limInferiorRot2 = 0; limInferiorRot3 = 0;
        sensorObstaculo = 0;
        localizacionDeObstaculo = 0;        /*0 ninguno, 1 arriba, 2 abajo, 3 izquierda, 4 derecha,
                                                5 interior,6 exterior*/
        accion = 0;  /*Acción actual a desarrollar, 0 ninguno, 1 arriba, 2 abajo, 3 roti, 4 rotd,
                        5 recoger, 6 estirar*/
   status = 0;                          /*0 terminado (fin de comando basico), 1 terminado por usuario,
                                        2 terminado por obstaculo, 3 ejecutando */
   delta = 3;           /*Razón de cambio de las rotaciones */
   identificador = id;
   rotActual1 = rotFinal1 = r1;
   rotActual2 = rotFinal2 = r2;
   rotActual3 = rotFinal3 = r3;
};

/*************************************************************************/

/*Inicializa la posición actual*/
void Dispositivo::inicia(float r1,float r2,float r3)
        {
                rotActual1 = r1;
                rotActual2 = r2;
                rotActual3 = r3;
        };

/*************************************************************************/

/*Mueve los eslabones del dispositivo r pasos*/
void Dispositivo::mueve(float r1,float r2,float r3)
        {
                rotActual1 + = r1;
                rotActual2 + = r2;
                rotActual3 + = r3;
        };

/*************************************************************************/

/*Movimiento básicos fijo*/
void Dispositivo::arriba() {
        if ((rotActual2 + delta) < limSuperiorRot2 && status = = 0)
        {
          rotFinal2 = rotActual2 + delta;
          accion = 1;
          status = 3;
        }
}

/*************************************************************************/

/*Movimiento básicos fijo*/
void Dispositivo::abajo() {
                if ((rotActual2-delta) > limInferiorRot2 && status = = 0 )
        {
                rotFinal2 = rotActual2-delta;
                accion = 2;
                status = 3;
        }
}

/*************************************************************************/

/*Movimiento básicos fijo*/
void Dispositivo::rotI() {
                if ((rotActual3-delta) > limInferiorRot3 && status = = 0 )
        {
                rotFinal3 = rotActual3-delta;
                accion = 3;
                status = 3;
        }
}

/*************************************************************************/

/*Movimiento básicos fijo*/
void Dispositivo::rotD() {
                if ((rotActual3 + delta) < limSuperiorRot3 && status = = 0 )
        {
                rotFinal3 = rotActual3 + delta;
                accion = 4;
                status = 3;
        }
}
```

```
/*********************************************************************/

/*Movimiento básicos fijo*/
void Dispositivo::estirar() {

        if ((rotActual1 + delta)<limSuperiorRot1 && status = =0 ) rotFinal1 = rotActual1 + delta;
        if ((rotActual2-delta)>limInferiorRot2 && status = =0 ) rotFinal2 = rotActual2-delta;
        accion = 6;
        status = 3;

}

/*********************************************************************/

/*Movimiento básicos fijo*/
void Dispositivo::recoger() {
        if ((rotActual1-delta)>limInferiorRot1 && status = =0 ) rotFinal1 = rotActual1-delta;
        if ((rotActual2 + delta)<limSuperiorRot2 && status = =0 ) rotFinal2 = rotActual2 + delta;
        accion = 5;
        status = 3;
}

/*********************************************************************/

/*Movimiento básicos fijo*/
void Dispositivo::detener(int causa) {
        rotFinal1 = rotActual1;
        rotFinal2 = rotActual2;
        rotFinal3 = rotActual3;
//      if (causa = = 2) localizarObstaculo(); /*localiza autom'aticamente el obst'aculo*/
        status = 0;

}

/*********************************************************************/

/*Movimiento básicos con especificación*/
void Dispositivo::arriba(float delt) {

        if ((rotActual2 + delt)<limSuperiorRot2 && status = =0)
        {
                rotFinal2 = rotActual2 + delt;
                accion = 1;
                status = 3;
         }

}

/*********************************************************************/

/*Movimiento básicos con especificación */
void Dispositivo::abajo(float delt) {
                 if ((rotActual2-delta)>limInferiorRot2 && status = =0 )
        {
                rotFinal2 = rotActual2-delt;
                accion = 2;
                status = 3;
        }
}

/*********************************************************************/

/*Movimiento básicos con especificación */
void Dispositivo::rotI(float delt) {
                 if ((rotActual3-delt)>limInferiorRot3 && status = =0 )
        {
                rotFinal3 = rotActual3-delt;
                accion = 3;
                status = 3;
        }
}

/*********************************************************************/

/*Movimiento básicos con especificación */
void Dispositivo::rotD(float delt) {
                 if ((rotActual3 + delt)<limSuperiorRot3 && status = =0 )
        {
                rotFinal3 = rotActual3 + delt;
                accion = 4;
                status = 3;
        }
}

/*********************************************************************/
```

```
/*Movimiento básicos con especificación */
void Dispositivo::estirar(float delt) {

        if ((rotActual1 + delt)<limSuperiorRot1 && status= =0 ) rotFinal1 =rotActual1 + delt;
        if ((rotActual2-delt)>limInferiorRot2 && status= =0 ) rotFinal2 =rotActual2-delt;
        accion=6;
        status=3;

}

/*********************************************************************/


/*Movimiento básicos con especificación */
void Dispositivo::recoger(float delt) {
        if ((rotActual1-delt)>limInferiorRot1 && status= =0 ) rotFinal1 =rotActual1-delt;
        if ((rotActual2 + delt)<limSuperiorRot2 && status= =0 ) rotFinal2 =rotActual2 + delt;
        accion=5;
        status=3;
}

/*********************************************************************/

/*Identifica la posición del obstáculo, debe llamarse antes de detener(int causa) para no perder la variable de accion*/
void Dispositivo::localizarObstaculo()
        {
        if( accion= =0) localizacionDeObstaculo=100; /*obstaculo debido a una accion exterior*/
        if( accion= =1) localizacionDeObstaculo=1;
        if( accion= =2) localizacionDeObstaculo=2;
        if( accion= =3) localizacionDeObstaculo=3;
        if( accion= =4) localizacionDeObstaculo=4;
        if( accion= =5) localizacionDeObstaculo=5;
        if( accion= =6) localizacionDeObstaculo=6;

};

/*********************************************************************/

/*Mueve el actuador indicado por "motor" en una cierta direcciónn*/
void Dispositivo::mueve(int motor,int direccion)
        {
        //vectorSalida[0]=(vectorSalida[0] | (0x01<<((motor-1)*2)))&~(0x02<<((motor-1)*2));
        if( motor= =1)
        {
        if( direccion= =0) rotFinal1 =rotActual1;
        if( direccion= =1) rotFinal1 =/*rotActual1 + 5;*/limSuperiorRot1;
        if( direccion= =2) rotFinal1 =/*rotActual1-5;*/limInferiorRot1;
        }
        else if( motor= =2 )
        {
        if( direccion= =0) rotFinal2 =rotActual2;
        if( direccion= =1) rotFinal2 =/*rotActual2 + 5;*/limSuperiorRot2;
        if( direccion= =2) rotFinal2 =/*rotActual2-5;*/limInferiorRot2;
        }

        else if( motor= =3 )
        {
        if( direccion= =0) rotFinal3 =rotActual3;
        if( direccion= =1) rotFinal3 =/*rotActual3 + 5;*/limSuperiorRot3;
        if( direccion= =2) rotFinal3 =/*rotActual3-5;*/limInferiorRot3;
        }

};

/*********************************************************************/
```

ManejadorPuerto.cpp

```
#include "d:\tesis\puerto\clases.h"

void ManejadorPuerto::inicializar(class Dispositivo *e1,class Dispositivo *e2,
class Dispositivo *e3,class Dispositivo *e4,class Dispositivo *e5,class Dispositivo *e6,
class Dispositivo *c  )
 {
                int i;
                dispositivo[1] = e1;
                dispositivo[2] = e2;
                dispositivo[3] = e3;
                dispositivo[4] = e4;
                dispositivo[5] = e5;
                dispositivo[6] = e6;
                dispositivo[7] = c;
                numeroDispositivos=7;
                for(i=0;i<=numeroDispositivos;senalActual[i++]=0);
                for(i=0;i<=numeroDispositivos;senalAnterior[i++]=0);
                for(i=0;i<=numeroDispositivos;vectorSalida[i++]=0);
                for(i=0;i<=numeroDispositivos;vectorSalidaAnterior[i++]=0xff);
```

```
                    senalDeCambio = 0;
                    portSalidaDatos = 0x378;
                    port1EntradaDatos = 0x379;
                    port2EntradaDatos = 0x37A;
                    portSelect = 0x278;
          };

/*****************************************************************/

/*Inicializa cada uno de los dispositivos del robot*/
void ManejadorPuerto::inicializarDispositivos(int numDispositivos)
{
 for( int i = 1;i <= 1/*numDispositivos*/;i++) inicializaDispositivo(i,100,50,120);
};

/*****************************************************************/



/*Retoma 0 si no se detecta dispositivo, 1 si lo detecta y puede inicializarlo. Mueve los eslabones del dispositivo hasta localizar límites, entonces posiciona en
r1,r2 y r3 respectivamente*/
int ManejadorPuerto::inicializaDispositivo(int numDisp,int r1,int r2,int r3)
{
int contador[3], obstaculo,contadorR;
unsigned char senalAnterior, senalDeCambio, senalActual,retardo = 200;
int numMotor = 2, r[] = {r1,r2,r3};
//while(1) {
// printf(" puerto %x \n",(( (inp(port1EntradaDatos) & 0xf8) | (inp(port2EntradaDatos) & 0x07))^0x80^0x02^0x01) );
 printf("\nDetectando dispositivo #%d", numDisp );
outport(portSalidaDatos,0x00);                                  /* Inicializa salida a 0*/
outport(portSelect,(numDisp-1) );                               /*Selecciona dispositivo*/
outport(portSalidaDatos,0x08);                                  /* Mueve el eslabón hacia arriba*/
delay(3000);
/*Existe o no el dispositivo*/
if (!  ( (((inp(port1EntradaDatos) & 0xf8) | (inp(port2EntradaDatos) & 0x07))^0x80^0x02^0x01) & 0x80 ) )
{
 printf("\n OK \ninicializando dispositivo #%d", numDisp );

for(numMotor = 0;numMotor<3;numMotor++)
{
 outport(portSalidaDatos,0x01<<(numMotor*2));                   /*Mueve eslabon al límite superios*/
  delay(retardo);                       /*Retraso para comenzar el movimiento, aun estando en el límite */
 senalDeCambio = senalActual = senalAnterior = obstaculo = contador[numMotor] = contadorR = 0;

  while( !obstaculo )                                           /*¡Hasta llegar al límite superior*/
          {
                    senalActual = ( (inp(port1EntradaDatos) & 0xf8) |
                                    (inp(port2EntradaDatos) & 0x07))^0x80^0x02^0x01;
                   senalDeCambio = senalActual ^ senalAnterior;
                   if(senalDeCambio&(0x01<<(numMotor)))          /*Camio de estado del sensor de límite*/
                   {
                              if(senalActual&(0x01<<numMotor))
                              obstaculo = 1;                            /*Detecta límite*/
                              senalAnterior = senalActual;
                   }
  }
  outport(portSalidaDatos,0x00);                                /*Detiene el eslabón*/
  outport(portSalidaDatos,0x02<<(numMotor*2));                  /*Mueve en sentido contrario el eslabon*/
  delay(retardo);
  senalActual = senalAnterior = obstaculo = 0;

  while( !obstaculo )                                           /*Detecta límite inferior*/
  {
          senalActual = ((inp(port1EntradaDatos) & 0xf8) | (inp(port2EntradaDatos) & 0x07))^0x80^0x02^0x01;
          senalDeCambio = senalActual ^ senalAnterior;
          if(senalDeCambio)                                     /*Detecta cambios en las señales de entrada*/
          {
             if(senalDeCambio&(0x10<<numMotor)) contador[numMotor]++;
                                                                /*Cambio de estado en el sensor deposicion*/
             if(senalDeCambio&(0x01<<numMotor) )
               if(senalActual&(0x01<<numMotor))
                         obstaculo = 1;/*obstaculo*/
                    senalAnterior = senalActual;
          }
  }
          outport(portSalidaDatos,0x00);                        /*Detiene al eslabón*/
          printf("\n\t Eslabon %d, amplitud del arco: %d",numMotor,contador[numMotor]);

  printf("\n\t Posicionando ....");       /*Una vez identificados los límites se posiciona el eslabón */
  outport(portSalidaDatos,0x01<<(numMotor*2));                  /*Regresa y centra el  eslabón*/
  delay(retardo);
 senalActual = senalAnterior = obstaculo = 0;
 while( !obstaculo )                    /*Posiciona el eje en rx contando desde el límite inferior*/
 {
          senalActual = ((inp(port1EntradaDatos) & 0xf8) | (inp(port2EntradaDatos) & 0x07))^0x80^0x02^0x01;
          senalDeCambio = senalActual ^ senalAnterior;
          if(senalDeCambio)             /*Detecta cambio de estado en la señal de entrada*/
          {
                    if(senalDeCambio&(0x10<<numMotor) )
```

```
                    {
                     contadorR + +;                    /* Detecta cambio de estado en el sensor de posicion*/
                     if(contadorR = = r[numMotor])              /*Comienza a contar desde el limite
                                                                inferior hasta dar con el especificado
                                                                en rn de la declaraci©n de esta función*/
                              obstaculo = 1;
                    }
                  if(senalDeCambio&(0x01 < < numMotor) )      /*Detecta cambio de estado */
                    if(senalActual&(0x01 < < numMotor))
                                        obstaculo = 1;                  /*Detecta límite de rotacion*/
                  senalAnterior = senalActual;
                 }
                 }
                 outport(portSalidaDatos,0x00);                         /*Detiene el eslabón */
                 printf("\n\t OK");
                 delay(500);
 }/*for */
dispositivo[numDisp]->limSuperiorRot1 = contador[0]-r1;
printf("\nlim sup 1: %d",dispositivo[numDisp]->limSuperiorRot1);
dispositivo[numDisp]->limSuperiorRot2 = contador[1]-r2;
printf("\nlim sup 2: %d",dispositivo[numDisp]->limSuperiorRot2);
dispositivo[numDisp]->limSuperiorRot3 = contador[2]-r3;
printf("\nlim sup 3: %d",dispositivo[numDisp]->limSuperiorRot3);

dispositivo[numDisp]->limInferiorRot1 = -r1;
printf("\nlim inf 1: %d",dispositivo[numDisp]->limInferiorRot1);
dispositivo[numDisp]->limInferiorRot2 = -r2;
printf("\nlim inf 2: %d",dispositivo[numDisp]->limInferiorRot2);
dispositivo[numDisp]->limInferiorRot3 = -r3;
printf("\nlim inf 3: %d",dispositivo[numDisp]->limInferiorRot3);
getch();
}
  else
  {
   printf("\n ERROR: no se ha podido detectar el dispositivo %d",numDisp);
  }
}/*funcion*/

/*************************************************************************/

/*Lee a través del puerto paralelo las señales de la interfaz electrónica, dispositivos disponibles 1-7*/

unsigned char  ManejadorPuerto::leeDispositivo(int dispositivo)
{
  outp(portSelect,dispositivo-1);
          return senalActual[dispositivo] = ((inp(port1EntradaDatos) & 0xf8) | (inp(port2EntradaDatos) & 0x07))^0x80^0x02^0x01;
};

/*************************************************************************/

/*Interpreta las señales leidas del puerto paralelo */
void  ManejadorPuerto::interpretaSenales(void)
{
 int disp;
 for(disp = 1;disp< = 1/*numeroDispositivos*/;disp + + )
 {
   if(leeDispositivo(disp)! = senalAnterior[disp])
    {
//          printf("\n\t\t ambos puertos  = 0x%X\t  ",senalActual[disp]);
          senalDeCambio =  senalActual[disp] ^ senalAnterior[disp];
          if(senalDeCambio&0x10 )               /*Eslabón 1 */
          {
                    if(vectorSalida[disp]&0x01)
                              dispositivo[disp]->rotActual1 + + ;     /*Incremento en la posicion*/
                    if(vectorSalida[disp]&0x02)
                              dispositivo[disp]->rotActual1--;        /*Decremento en la posicion*/
          }
          if(senalDeCambio&0x20 )               /*Eslabón 2*/
          {
                    if(vectorSalida[disp]&0x04) dispositivo[disp]->rotActual2 + + ;//positivo
                    if(vectorSalida[disp]&0x08) dispositivo[disp]->rotActual2--;//negativo
          }

          if(senalDeCambio&0x40 )               //opto3
          {
                    if(vectorSalida[disp]&0x10) dispositivo[disp]->rotActual3 + + ;//positivo
                    if(vectorSalida[disp]&0x20) dispositivo[disp]->rotActual3--;//negativo
          }

/*comprobaci'on de limites si hay una senal de limite y no hay una senal
de mov expresa, dejara de moverse la extremidad, serian necesarias
otras 3 senales de entrada*/
          if((senalDeCambio&0x01) && (senalActual[disp]&0x01) )         //limite eslabon1
          {     printf("\nlimite1");
                    if(vectorSalida[disp]&0x01)
                    {
                              dispositivo[disp]->rotActual1 = dispositivo[disp]->rotFinal1 = dispositivo[disp]->limSuperiorRot1;//
//                            vectorSalida[disp] = vectorSalida[disp]&0xFE;/*detiene motor1*/
                              //extremidad1.mensajeDeLimite();
                    }
```

```
                        if(vectorSalida[disp]&0x02)
                        {
                                dispositivo[disp]->rotActual1 = dispositivo[disp]->rotFinal1 = dispositivo[disp]->limInferiorRot1;//negativ2
//                              vectorSalida[disp] = vectorSalida[disp]&0xFD;/*detiene motor1*/
                                //extremidad1.mensajeDeLimite();
                        }
                }

                if((senalDeCambio&0x02) && (senalActual[disp]&0x02 ) )          //limite eslabon2
                {       printf("\nlimite2");
                        if(vectorSalida[disp]&0x04)
                        {
                                dispositivo[disp]->rotActual2 = dispositivo[disp]->rotFinal2 = dispositivo[disp]->limSuperiorRot2;
//                              vectorSalida[disp] = vectorSalida[disp]&0xF7;/*detiene motor2*/
                                //extremidad1.mensajeDeLimite(); donde se deben igualar los cont con lim
                        }
                        if(vectorSalida[disp]&0x08)
                        {
                                dispositivo[disp]->rotActual2 = dispositivo[disp]->rotFinal2 = dispositivo[disp]->limInferiorRot2;
//                              vectorSalica[disp] = vectorSalida[disp]&0xFB;/*detiene motor2*/
                                //extremidad1.mensajeDeLimite();
                        }
                }

                if((senalDeCambio&0x04) && (senalActual[disp]&0x04) )           //limite eslabon3
                {        printf("\nlimite3");
                        if(vectorSalida[disp]&0x10)
                        {
                                dispositivo[disp]->rotActual3 = dispositivo[disp]->rotFinal3 = dispositivo[disp]->limSuperiorRot3;
//                              vectorSalida[disp] = vectorSalida[disp]&0xEF;/*detiene motor2*/
                                //extremidad1.mensajeDeLimite(); donde se deben igualar los cont con lim
                        }
                        if(vectorSalida[disp]&0x20)
                        {
                                dispositivo[disp]->rotActual3 = dispositivo[disp]->rotFinal3 = dispositivo[disp]->limInferiorRot3;
//                              vectorSalida[disp] = vectorSalida[disp]&0xDF;/*detiene motor2*/
                                //extremidad1.mensajeDeLimite();
                        }
                }
/*deteccion de obstaculos*/
                if(!(senalActual[disp]&0x08) )          //obstaculo
                {
                        dispositivo[disp]->detener(2);/*iguala actuales y finales*/
                        vectorSalida[disp] = 0x00;/*detiene motores*/
                        printf("\n obstaculo %d",disp );
                        //determinaObstaculo();
                }

                senalAnterior[disp] = senalActual[disp];

        }
          printf("\n posicion actual %d,%d,%d",dispositivo[disp]->rotActual1,dispositivo[disp]->rotActual2,dispositivo[disp]->rotActual3);
                /*el procedimiento de control es sencillo, simplemente tratar de igualar el final con el actual*/
                if(dispositivo[disp]->rotActual1 = = dispositivo[disp]->rotFinal1)
                { vectorSalida[disp] = vectorSalida[disp]&0xFC;/*detener motor1*/
                }
                else if(dispositivo[disp]->rotActual1 < dispositivo[disp]->rotFinal1) vectorSalida[disp] = (vectorSalida[disp]&0xFC)|0x01;/*avanzar motor1*/
                else if(dispositivo[disp]->rotActual1 > dispositivo[disp]->rotFinal1) vectorSalida[disp] = (vectorSalida[disp]&0xFC)|0x02;/*retroceder
                                                                                                                                        motor1*/

                if(dispositivo[disp]->rotActual2 = = dispositivo[disp]->rotFinal2)  vectorSalida[disp] = vectorSalida[disp]&0xF3;/*detener motor2*/
                else if(dispositivo[disp]->rotActual2 < dispositivo[disp]->rotFinal2) vectorSalida[disp] = (vectorSalida[disp]&0xF3)|0x04;/*avanzar motor2*/
                else if(dispositivo[disp]->rotActual2 > dispositivo[disp]->rotFinal2) vectorSalida[disp] = (vectorSalida[disp]&0xF3)|0x08;/*retroceder
                                                                                                                                        motor2*/

                if(dispositivo[disp]->rotActual3 = = dispositivo[disp]->rotFinal3)  vectorSalida[disp] = vectorSalida[disp]&0xCF;/*detener motor3*/
                else if(dispositivo[disp]->rotActual3 < dispositivo[disp]->rotFinal3) vectorSalida[disp] = (vectorSalida[disp]&0xCF)|0x10;/*avanzar motor3*/
                else if(dispositivo[disp]->rotActual3 > dispositivo[disp]->rotFinal3) vectorSalida[disp] = (vectorSalida[disp]&0xCF)|0x20;/*retroceder
                                                                                                                                        motor3*/

                /*dado el caso de cambiar el vector de salida se envia al puerto*/
                  if(vectorSalida[disp]! = vectorSalidaAnterior[disp])
                        {     // outp(portSelect,disp);
                        outport(portSalidaDatos,vectorSalida[disp]);
                        vectorSalidaAnterior[disp] = vectorSalida[disp];
                        printf("\n\t\t salida  = 0x%X\t  ",vectorSalida[disp]);
                         }

  }
 };

/*************************************************************************/
```

Glosario

A

Actuador
Dispositivo que transforma la energía (comúnmente eléctrica, o potencial) en energía cinética. Dispositivo generador de esfuerzos a velocidad variable que puede modificar y mantener la configuración de la estructura mecánica del robot.

Agente
Software con características de movilidad y autonomía dentro de un ambiente de cómputo distribuido.

Alfa, versión
Es una liberación de software, la cual puede tener cambios muy radicales entre una liberación y otra. Generalmente las versiones alfa son inestables

API (Aplication Programming Interface)
Es un conjunto de especificaciones que indican la forma exacta en que se debe y puede interactuar o usar un sistema de software.

Applet
Es un programa que debe ser ejecutado por medio de un visualizador de WWW.

Appletviewer
Es un programa que permite visualizar applets. Este programa es proporcionado con el JDK.

Articulación
Punto fijo entre dos cuerpos que pueden o no tener movimiento relativo entre ellos.

AWT (Abstract Windowing Toolkit)
Es un paquete de clases que implementa los componentes de interfaz de usuario más comunes, tales como ventanas. botones, menús, y otros.

B

Beta, versión
Es una liberación de software, cuyos cambios no agregan funcionalidad al sistema sino más bien tratan de corregir posibles errores que pueda presentar.

ByteCodes
Es un conjunto de instrucciones similares al código ensamblador de algunas máquinas pero que no es específico de algún procesador.

C

C++
Lenguaje de programación orientado a objetos híbrido, basado en el lenguaje C.

CAD
De sus siglas en ingles Computer Aided Design (Diseño Asistido por Computadora).

Cadena cinemática
Conjunto de eslabones unidos por articulaciones.

Cadena cinemática abierta
Constituida por eslabones binarios y al menos un eslabón unitario.

Cadena cinemática cerrada
Constituida únicamente por eslabones binarios.

Cinemática directa
Cálculo de la posición del órgano terminal de un robot a partir de sus variables de articulación (posición angular o desplazamiento entre sus eslabones).

Cinemática Inversa
Cálculo de las variables de articulación (posición angular o desplazamiento entre sus eslabones) de un robot a partir a partir de la posición de su órgano terminal.

Clase
Es un objeto que define los métodos y atributos para un tipo particular de objeto. Todos los objetos de una clase dada son idénticos en forma y comportamiento pero contienen diferentes datos en sus variables.

Cliente/Servidor
Esquema de colaboración entre procesos, en el cual ciertos programas de cómputo fungen como administradores de recursos, prestando servicios a otros que así lo requieran (clientes).

Clipboard
Área de memoria especial reservada para compartir información entre procesos. En esta área todos los procesos pueden escribir o leer.

D

Daemons
Programas que son ejecutados en background y que por lo regular prestan algún servicio a otros procesos.

E

Encapsulamiento
Es la técnica que permite que los detalles internos de un módulo no sean accesibles por otros módulos, protegiéndolo de interferencias exteriores y protegiendo a los otros módulos de confiar en detalles de implementación que pueden cambiar con el tiempo.

Escenario
De acuerdo a la terminología de VRML un escenario o mundo (world) es el entorno que forman las entidades definidas en un archivo script (*.wrl).

Espacio de trabajo
Área física donde el robot puede realizar una tarea. Es el espacio formado por todos los puntos donde puede posicionarse el órgano terminal.

Eslabón
Cuerpo idealmente rígido que conectan a una o más articulaciones. El número de articulaciones de un eslabón determina si este es unitario, binario, etc.

F

FTP (File Transfer Protocol)
Es un protocolo que permite la transferencia de archivos entre distintos sistemas de cómputo que utilizan TCP/IP como protocolo de transporte.

Framework
Es un conjunto de bibliotecas especializadas que sirven de apoyo para la creación de nuevos sistemas.

G

Garbage collection
Es un mecanismo que permite la liberación de memoria de manera automática sin la intervención directa del programador.

Gateways
Son dispositivos de red que permiten la transferencia de información entre dos o más redes con igual o distinto protocolo de transporte..
GUI
Acrónimo para Graphic User Interface

Grado de libertad
Número de articulaciones independientes utilizadas para posicionar y orientar los eslabones de un robot.

H

Herencia
Un mecanismo mediante el cual las clases pueden hacer uso de los métodos y variables definidas en todas las clases por encima de ella siguiendo su ruta dentro de la jerarquía de clase.
Html (Hipertext Markup Language)
Es un lenguaje que permite la creación de documentos los cuales pueden tener referencias (hipertexto) hacia otros recursos (documentos, imágenes, programas, etc.) locales o remotos.

HTTP (Hipertext Transfer Protocol)
Es un protocolo a nivel de aplicación que permite la tranferencia de documentos HTML u otros recursos referenciados por dichos documentos.

I

IEC
International Electrotechnical Commission.

Instancia
Un término usado para referirse a un objeto que pertenece a una clase en particular.

Intranets
Un concepto que hace referencia a la creación de una red a nivel corporativo que ofrece los servicios encontrados en Internet, con el objetivo de mejorar la funcionalidad de la empresa.

ISO
International Organization of Standardization.

J

Java
Lenguaje orientado a objetos desarrollado por Sun Microsystems y que ha sido utilizado principalmente para la creación de applets.
Javac
Compilador de Java. Este programa se encarga de generar el bytecode que posteriormente será interpretado por la máquina virtual de Java para ejecutar el programa.

Javadoc
Programa que permite la generación de documentación (API) en formato HTML del código de la aplicación de forma automática.

Javah
programa que permite la creación de métodos nativos que serán invocados desde Java.

JavaOS
Es un sistema operativo compacto que ejecuta aplicaciones Java sobre dispositivos tales como computadoras en red o teléfonos celulares.
JAR
Es un formato de archivo independiente de plataforma que contiene información de varios archivos dentro de uno solo.

JDK (Java Developer´s Kit)
Es un kit de desarrollo que incluye herrramientas para desarrollar aplicaciones y applets hechos en el lenguaje Java.

JIT (just-in-time compilation)
Es una tecnología que acelera la ejecución de programas basados en el concepto de bytecode.

L

LAN
Acrónimo para Local Area Network (red de área local)

Lisp
Lenguaje de programación basado en inferencias ampliamente utilizado en el campo de la inteligencia artificial.

M

Máquina virtual
ver VM.

MicroJava
Especificación de la primera generación de microprocesadores dedicados a la ejecución de aplicaciones hechas en Java.

MIT
Acrónimo para Massachusets Institute of Technology

Movilidad
Número de articulaciones de un robot.

Multicast
Es una técnica que permite enviar de forma asíncrona un mensaje a un grupo específico de procesos

Multithread
Múltiples hilos de control. Es una técnica que permite la ejecución concurrente de varios hilos de control dentro de un proceso.

O

Objeto
Un paquete de software que contiene una colección de datos relacionados (en forma de variables) y métodos (procedimientos) para manipular esos datos.

OSI (Open Systems Interconnection)
Un conjunto de estándares que definen reglas a las cuales se deben de apegar diversos servicios en diferentes niveles de una red .

P

Polimorfismo
La habilidad de ocultar diferentes implementaciones detrás de una interfaz común, simplificando la comunicación entre los objetos.

Proceso
Es una abstración que abarca el manejo de memoria, CPU y recursos de entrada/salida corespondientes a un programa.

R

Realidad Virtual
Es un conjunto de medios que permiten al usuario tener la sensación de interactuar con la simulación de un medio, logrando que este se asuma como un medio real.

Robot
Entidad programable, idealmente adaptable al medio de trabajo que lo rodea, capaz de organizar sus acciones con el fin de realizar tareas.

S

Simulación
Proceso de obtener información a cerca de un fenómeno a partir de un modelo matemático que lo representa.

Script
Conjunto de comandos para un interprete.

ServerSocket
Es un socket que se encarga de monitorear la red por posibles solicitudes de conexión provenientes de otra máquina.

Shell
Un intérprete de comandos y lenguaje de programación en sistemas Unix.

Sockets
Es la unión de la dirección IP de una máquina dada y un puerto de servicio asociado a una aplicación especifica. Un socket sirve como un medio de comunicación entre procesos.

Sparc
Arquitectura de microprocesadores basada en RISC desarrollada por SUN Microsystems.

Stand-alone
Este término se aplica a toda aplicación que no requiere de ninguna otra para su ejecución.

T

TCP/IP (Transfer Control Protocol/ Internet Protocol)
Conjunto de protocolos a nivel de transporte y red respectivamente que son utilizados normalmente en sistemas Unix.

Template
Es un esqueleto que define la estructura básica que debe de cumplir cualquier implementación basada en éste.

Thread
hilo de control que define el flujo de ejecución dentro de un proceso.

U

UNICODE
Es un sistema de codificación de caracteres de 16 bits diseñado para soportar el intercambio, procesamiento y despliegue de los textos escritos en diversos lenguajes del mundo moderno.

URL (Uniformal Resource Locator)
Es un formato estándar utilizado en Internet para la ubicación de algún recurso en la red. El formato es: protocolo://recurso:puerto

V

Visualización
Representación gráfica de los datos numéricos provenientes de una simulación o de la medición directa de las variables de un fenómeno.

Visualizador
Herramienta de software que permite desplegar documentos en formato HTML.

VM (Virtual Machine)
Intérprete que ejecuta el bytecode generado previamente por un compilador.

W

WAN
Acrónimo para Wide Area Network (red de área amplia)

WWW (World Wide Web)
Servicio de información a nivel mundial que se basa en la utilización de hipertexto.

X

XDR (Exchange Data Representation)
Es un estándar para la descripción y codificación de datos. Es útil para transferir datos entre diversas arquitecturas de cómputo. Usa un lenguaje para describir formato de datos. Este lenguaje solamente puede ser usado para describir datos, ya que no es un lenguaje de programación.

Referencias

Bibliografía

Object oriented analisis
Peter Coad
Prentice Hall

The essential client/server survival guide
Robert Orfali
Wiley Computer Publishing

VRML para Internet
Mark Pesce
New Riders Publishing, 1996

The essential client/server survival guide
Robert Orfali
Wiley Computer Publishing

VRML para Internet
Mark Pesce
New Riders Publishing, 1996

Object oriented programing with Smalltalk/V.
Dusko Savic
Ellis herwood Limited, 1993

Diseño digital
Morris Mano
Prentice Hall, 1993

Inside 3D Studio
Steve Elliont
New Riders Publishing, 1994

Inside Acad 13
Phillip Miller
New riders Publishing, 1995

Apuntes de Robótica
Ignacio Juárez Campos
Facultad de Ingeniería

UNIX Power Tools
Jerry Peck
O'Reilly & Associates, 1996

Java in a Nutshell
David Flanagan
O'Reilly & Associates, 1996

Teach yourself Java in 21 days
Laura Lemay
Sams Net

Documentos y artículos

Simulador para la Cinemática Directa e Inversa de Manipuladores de Tres Grados de Libertad.
José Castillo H.
Hernando Ortega C.
Sociedad Mexicana de Instrumentación, Centro de Instrumentos UNAM.

Agentes para la Administración y Seguridad de Equipos de Cómputo en Red
Norberto Ortigoza M.
Andres Pineda R.
Tesis para obtener el grado de Ingeniero en Computación,
Facultad de Ingeniería UNAM.

The Virtual Reality Modeling Languaje, VRML2.0
Chris Marrin
Silicon Graphics Inc.

VRML2.0 Specification Java API
Chris Marrin
Silicon Graphics, Inc, 1997

VRML Script
Chris Marrin
Silicon Graphics, Inc, 1996

External Authoring Interface Specification
Jim Kent
Silicon Graphics, Inc, 1996

Liquid Reality White Paper
DimensionX, 1996

The Java Tutorial
Mary Campione and Kathy Walrath
Sun Microsystems, Inc., 1995

Realistic animation of legged running on rough terrain
John Nogle
Anmuts, 1996

Direcciones URL

Especificación y API de Liquid Reality
http://www.microsoft.com/dimensionx

Especificación de VRML
http://www.vrml.com

Especificación y API de Java
http://www.javasoft.com

Recursos y tutoriales de programación con Java y VRML en arquitecturas distribuidas
http://www.hermética.com

Recursos y tutoriales sobre programación con sockets
http://www.sockets.com
http://www.goodnet.com/~esnible/winsock.html
http://www.alumni.caltech.edu/~dank/trumpet/
http://dark.uwaterloo.ca/wattcp.html

Especificaciones acerca de otros lenguajes gràficos
http://www.ssec.wisc.edu/~brianp/Mesa.html#Systems
http://www.cs.utah.edu/~narobins/opengl.html

Recursos para 3D Studio y AutoCAD
http://cedar.cic.net/~rtilmann/mm/obcat/comp.htm objetos 3dstudio
http://avalon.viewpoint.com/ objetos 3dstudio
http://www.ccyber.com/ar13.html
http://sunserver1.rz.uni-duesseldorf.de/~pannozzo/3ds.html

Otro proyetos sobre robots móbiles
http://www.ai.mit.edu
http://alpha-bits.aimit.edu/projects/mobile-robots/
http://ccn.cs.dal.ca/Science/SAS/sas-robo.html
http://www.ccyber.com/ar13.html
http://ammuts.com