53

UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

INTERFAZ GRÁFICA 3D PARA SIMULAR Y CONTROLAR ROBOTS MÓVILES USANDO REALIDAD VIRTUAL

T E S I S:

QUE PARA OBTENER EL TÍTULO DE:

INGENIERO EN COMPUTACIÓN

P R E S E N T A N:

EMMANUEL HERNÁNDEZ HERNÁNDEZ
GABRIEL VÁZQUEZ RODRÍGUEZ

DIRECTOR:
DR. JESÚS SAVAGE CARMONA

México, D.F. **2002**

DEDICATORIA

A mis padres **Rafael y Gloria** por su cariño, confianza y esfuerzo conjunto que me permitió llegar a este momento.

A mis hermanos **Alberto** y **Eunice** por su comprensión, consejos y paciencia.

A mis **amigos**, por los momentos de alegría que pasamos juntos.

Emmanuel

DEDICATORIAS

A mis padres:

Emma y **Gabriel** quienes depositaron en mí todo su amor y confianza para alcanzar esta meta.

A la Fundación Andrade:

En especial al Sr. **Alberto** y la Sra. **Dolores Andrade** por el apoyo incondicional que me brindaron durante todos mis estudios.

Gabriel

AGRADECIMIENTOS

Al **Dr. Jesús Savage Carmona** por la gran ayuda que nos ha brindado todo el tiempo.

Al equipo del **Laboratorio de Interfaces Inteligentes** por el apoyo recibido y por la amistad.

A todas aquellas personas que contribuyeron a la realización de esta tesis.

CONTENIDO

C

CAPÍTULO I
INTRODUCCIÓN

1.1 INTRODUCCIÓN

El proceso de planear y construir un robot móvil puede ser demasiado lento y costoso; está sujeto a errores tanto en el diseño como en la construcción y algunas veces suceden por circunstancias externas como son los materiales defectuosos. En el peor de los casos un error en el desarrollo conlleva a plantear de nuevo el diseño.

Una vez que la construcción del robot ha finalizado, es necesario someterlo a pruebas que verifiquen su correcto funcionamiento. Pasada la etapa de pruebas, el robot es utilizado para el propósito que fue creado; aún en este punto pueden presentarse situaciones que no se habían tomado en cuenta, modificar el diseño puede ser desde complicado hasta imposible.

Cuando se cuenta con el robot adecuado para realizar la tarea deseada, el robot no tiene aún los programas necesarios que permitan realizar las funciones para las que se creó. En muchas ocasiones cada diseño de robot es distinto y para cada caso en particular es necesario rescribir el código de los programas, ya sea porque el robot no cuenta con la misma configuración que el anterior o simplemente porque no utilizan el mismo lenguaje de programación. Hacer cambios a los programas que controlan a un robot puede ser una labor tardada y sujeta a errores.

Este trabajo tiene como objetivo presentar una alternativa a la etapa de desarrollo y prueba de algoritmos para al menos tres configuraciones de robots móviles. Se desarrollará una aplicación que consiste en una interfaz, que permita al usuario, simular desde uno hasta cinco robot móviles dentro de un espacio tridimensional. Los robots virtuales contarán con diversos tipos de sensores que les permitirán percibir información referente al mundo virtual donde se encuentran, y además, podrán ser programados haciendo uso de un lenguaje de programación desarrollado especialmente para este proyecto. Usar un lenguaje de programación de alto nivel para programar robots móviles será una forma sencilla y rápida para desarrollar programas complicados. El proyecto contará con gráficas 3D (tridimensionales) como medio de visualización de resultados, ya que los robots reales se desenvuelven en ambientes tridimensionales.

1.2 ROBOTS MÓVILES

1.2.1 DEFINICIÓN

El término "robot" fue acuñado por el checoslovaco Karel Capek en su obra R.U.R. (Rossum's Universal Robots), la cual fue presentada en Praga en 1921. El término *Robota*, el cual tiene origen checo, quiere decir labor forzada.

En general, un robot es un dispositivo mecánico que puede ser programado para desarrollar un gran número de tareas que pueden abarcar manipulación y movimiento bajo control automático. Los robots pueden ser controlados por humanos, algunas veces a gran distancia, o bien, pueden ser controlados por computadora; por ello, existen diversos tipos.

1.2.2 CLASIFICACIÓN DE ROBOTS

Una forma de clasificar a un robot tiene que ver con la época en la cual apareció. La primera generación de robots data de los 70's y consiste de dispositivos estacionarios, no programables, sin sensores y electromecánicos. La segunda generación de robots fue desarrollada en los 80's e incluye sensores y controles programables. La tercera generación de robots fue desarrollada entre 1990 y la época actual. Estas máquinas pueden ser estacionarias o móviles, autónomos o no, usan programación sofisticada, reconocimiento o síntesis de voz y otras características avanzadas. La cuarta generación de robots está en la fase de investigación y desarrollo e incluye características como inteligencia artificial, auto ensamblaje y tamaño nanométrico.

Dentro de las clasificaciones de robots tenemos robots fijos y robot móviles. Los robots fijos, como su nombre lo indica, se encuentran adheridos a una base; sin embargo, pueden tener extremidades que les permitan realizar diversas funciones; por ejemplo, los robots manipuladores que se usan en la industria automotriz. Un robot móvil consiste de un dispositivo electromecánico que puede ser programado, desplazarse por sí mismo y puede contar con sensores. Existen diversos tipos de robots móviles, la configuración mecánica de la mayoría de ellos entra en dos categorías: vehículos de ruedas dirigidas y vehículos de control diferencial. La configuración de ruedas dirigidas consiste de un eje trasero y una sola rueda frontal. La configuración de triciclo de muchos robots móviles es un ejemplo de este tipo de configuración, pero en general, una configuración de cuatro ruedas se puede considerar dentro de esta categoría si dos ruedas son aproximadas a una sola rueda en el centro del eje. El ejemplo más común de vehículo de control diferencial es un tanque militar, en donde el control de movimiento se realiza al mover con distintas velocidades las ruedas del tanque. La ventaja de los vehículos de control diferencial consiste en la capacidad de poder dar vuelta sobre su propio eje.

1.2.3 CÓMO CONTROLAR EL MOVIMIENTO DE UN ROBOT

La posición de un robot, regularmente está representada por tres parámetros: su posición (x, y) y su orientación, siempre y cuando se asuman movimientos planares. En un robot de tres ruedas solo hay dos grados de libertad que son los correspondientes a los motores del eje trasero; controlar el movimiento de un robot móvil de esta configuración se limita a controlar los motores.

Los algoritmos de control de movimiento pueden ser divididos en dos partes: movimientos de trayectoria continua y movimientos punto a punto. El control de movimiento de trayectoria continua consiste en trasladar al robot sobre una línea continua, dicha línea no necesariamente tiene que ser recta. El tipo de movimiento punto a punto consiste en desplazar al robot desde su punto actual a otro punto, para ello se debe tener conocimiento de su posición en el mundo.

En general, hacer que un robot móvil se traslade sobre una recta requiere de muchos más cálculos que los movimientos punto a punto, pero el movimiento del robot es más suave cuando se mueve sobre una línea y por lo mismo puede ser más lento, facilitando de esta manera el control.

El principio para controlar un robot móvil se basa en el conocimiento de la posición del vehículo y su ambiente. Sin embargo, en la práctica estas dos características pueden ser

desconocidas. Esta es la razón principal de contar con sensores. Los sensores recopilan información que puede ser procesada para obtener una posición estimada, detectar obstáculos y explorar el ambiente. Un robot puede contar con distintos tipos de sensores, los cuales se determinan por diversas razones como lo es la tecnología existente y el propósito del robot.

Los sensores pueden clasificarse en dos categorías: sensores de posición y sensores de ambiente. Los sensores de posición tienen como tarea proporcionar información al robot para que pueda estimar su posición actual. Los sensores de ambiente permiten al robot determinar las características del ambiente en el que se encuentra. Utilizando la información recopilada por sus sensores, un robot puede crear un mapa con las características del entorno en el que se encuentra, y con base en este mapa, puede planear sus movimientos para evitar obstáculos.

1.3 REALIDAD VIRTUAL

1.3.1 DEFINICIÓN

Atendiendo al significado de las palabras que componen al término Realidad Virtual: realidad es la cualidad o estado de ser real o verdadero, y virtual es aquello que existe o resulta en esencia o efecto, pero no como forma, nombre o hecho real. Para lograr una definición adecuada es necesario introducir la concepción humana y tecnológica del término: un sistema con un ambiente interactivo, tridimensional, generado por computadora, el cual pretende la manipulación de los sentidos humanos, simulando el conjunto completo de datos sensoriales que constituyen la "experiencia real". El resultado de esta simulación es un ambiente artificial que no existe desde el punto de vista físico. Dentro de este ambiente uno o varios participantes acoplados de manera adecuada al sistema interactúan de manera rápida e intuitiva.

Un mundo virtual es un ambiente similar a la realidad que se crea utilizando un medio artificial; es una simulación de un ambiente real o imaginario que provee una experiencia en tiempo real proporcionando estímulos parecidos a los de la realidad.

1.3.2 BREVE HISTORIA

Los primeros dispositivos que intentaron recrear situaciones reales en un modo artificial para uso humano fueron los primeros entrenadores de vuelo patentados en 1930, los cuales intentaban reproducir los movimientos de una nave para simular la sensación de volar. Estos eran aparatos mecánicos sin ningún dispositivo gráfico o interfaz parecida.

A mediados de los sesenta el término Realidad Virtual no existía, pero como concepto surge cuando el Dr. Ivan Sutherland publicó un artículo titulado "The Ultimate Display", en el cual establece los principios y conceptos básicos de la realidad virtual. En este escrito se menciona la posibilidad de proporcionar una interfaz que funcione con cualquier simulación, que produjera no sólo la experiencia de volar en un avión sino la de estar en cualquier espacio artificial, en el interior del cual la computadora tiene control total de los eventos. En 1966 Sutherland creó el primer casco visor de Realidad Virtual al montar tubos de rayos catódicos en un armazón de alambre. Este

instrumento fue llamado "espada de Damocles", debido a que el aparato requería de un sistema de apoyo que pendía del techo. Sus primeros desarrollos ya incluían sensores de seguimiento. En 1968 Ivan Sutherland y David Evans crean el primer generador de escenarios con imágenes tridimensionales, datos almacenados y aceleradores. En este año se funda también la sociedad Evans & Sutherland. Su sistema final consistió en muchos aceleradores de hardware para improvisar el funcionamiento del sistema gráfico y la generación de imágenes estereoscópicas en vez de monoscópicas.

Al empezar los setenta se crean los primeros simuladores de vuelo computarizados los cuales marcan lo viable y práctico de los sistema virtuales. Durante esta década se presentan importantes investigaciones y desarrollos sobre despliegue, presentaciones gráficas y sistemas de comunicación hombre / máquina.

A principios de los ochenta Jaron Lanier introduce el término Realidad Virtual, además de colaborar en el desarrollo de periféricos de realidad virtual como guantes y visores. En 1984 William Gibson publica su novela de ciencia ficción, "Neuromancer" en el que se utiliza por primera vez el término "Ciberespacio", refiriéndose a un mundo alternativo al de las computadoras, este término es utilizado por algunas personas para referirse a la realidad virtual.

Durante este período Thomas Furness III desarrolla la "cabina virtual" y poco después presenta el simulador más avanzado contenido en su totalidad en un casco, creado para la armada de los Estados Unidos. Además se crearon notables avances en los dispositivos de entrada y salida como son los visores a color, sonido tridimensional, guantes de control y sistemas más avanzados de posicionamiento. A finales de la década de los ochenta se exhiben los primeros sistemas de realidad virtual completos.

Actualmente, los sistemas de realidad virtual tienen aplicaciones en tareas científicas, en medicina, astronomía, además de ser una ayuda para entrenamientos, como por ejemplo de astronautas, y presentan una notable atracción para crear sistemas de entretenimiento.

1.3.3 CLASIFICACIÓN DE MUNDOS VIRTUALES

Existen diversas formas de clasificar los mundos virtuales, cada una ellas atiende a sus características como: interacción con el usuario, sentidos humanos involucrados, cercanía con la realidad, etc. Usando la clasificación de mundos virtuales por medio de su interacción con el usuario tenemos:

- Pasivos
- Reactivos
- Activos

Los mundos virtuales pasivos permiten realizar recorridos pero no hay interacción con objetos ni tampoco comportamientos predefinidos en objetos. Los mundos reactivos cuentan con recorridos y objetos con comportamiento reactivo a las acciones del usuario. Los mundos activos permiten

realizar recorridos, los objetos tienen comportamientos propios sin necesidad de interactuar con el usuario y también tienen comportamiento reactivo.

Usando una clasificación de acuerdo a la forma en que éstos presentan la información del mundo virtual, los sistemas de realidad virtual pueden clasificarse en tres tipos:

- Inmersivos. Los sistemas inmersivos tienen por objetivo hacer sentir al usuario realmente dentro del mundo virtual representado.
- Proyección. Los sistemas proyectivos intentan proporcionar la misma sensación de inmersión, valiéndose de un espacio cerrado en el cual se introduce al individuo; estos sistemas por lo general son multiusuario.
- Sobremesa. Los sistemas de sobremesa son aquellos que no se enfocan en la característica de inmersión visual, y el usuario nunca es aislado visualmente del mundo circundante.

Algunos de los sistemas específicos para la presentación de la realidad virtual son:

- Cabina de simulación. Generalmente la cabina recrea el interior del dispositivo o máquina que se desea simular (un carro, un avión, o un tanque), las ventanas de la misma se remplazan por pantallas de computadoras de alta resolución, además existen bocinas estereofónicas que brindan el sonido ambiental y pueden estar fijas o sobre ejes móviles. El programa está diseñado para responder en tiempo real a los estímulos que el usuario le envía por medio de los controles dentro de las cabinas.
- Realidad proyectada. En este tipo de sistema una imagen en movimiento del usuario es proyectada junto con otras imágenes en una extensa pantalla donde el usuario puede verse a sí mismo como si estuviese en el escena. En esencia los usuarios se miran ellos mismos como proyectados hacia el mundo virtual. Un ejemplo actual de este tipo de realidad virtual son los escenarios virtuales que se utilizan en ciertos programas de televisión.
- Realidad aumentada. La realidad aumentada se logra cuando una persona tiene el mundo real como referencia, pero utiliza visores de cristal transparentes u otros medios inmersivos para aumentar la realidad, superponiendo esquemas, diagramas, textos, referencias, entre otros.
- Telepresencia. El sistema proporciona a la persona la sensación de estar físicamente en otro lugar por medio de una escena creada por computadora. Es una experiencia psicológica que ocurre cuando la tecnología de simulación funciona lo suficientemente bien como para convencer al usuario de que está en un mundo virtual.

1.3.4 DISPOSITIVOS DE SALIDA EMPLEADOS EN APLICACIONES DE REALIDAD VIRTUAL

Los dispositivos de salida empleados en las aplicaciones de realidad virtual se dividen en las siguientes categorías: dispositivos de presentación y dispositivos de audio. Los primeros

proporcionan al usuario la imagen de los mundos virtuales, mientras que el segundo es un auxiliar muy importante para darle mayor realismo a las aplicaciones virtuales.

Dispositivos de presentación

Los dispositivos o periféricos de salida tienen por objeto proporcionar al usuario la imagen del mundo virtual, es decir, información de carácter gráfico acerca de los objetos que componen el mundo virtual, sus posiciones y sus características. Los dispositivos que comúnmente se encargan de realizar estas tareas son los monitores y las pantallas de proyección; sin embargo, existen otros dispositivos de presentación especiales como son los HMD (Head Mounted Display/ Dispositivo de visualización que es colocado en la cabeza).

Los cascos (HMD) y los sistemas binoculares se emplean únicamente en sistemas inmersivos para reforzar en el usuario la sensación de realidad, ya que aíslan al usuario completamente de su mundo exterior. Sin embargo, este dispositivo presenta algunas lamentables desventajas. Una de ellas es que el usuario se ve imposibilitado de utilizar dispositivos como el teclado, ya que no lo puede ver, debido a la inmersión en la que se encuentra.

En los sistemas proyectivos y de sobremesa se emplean con mayor frecuencia dispositivos convencionales, como los monitores, que nos brindan una excelente resolución y calidad en la imagen, además de que no dificultan el empleo de interfaces físicas. A cambio, no son inmersivos, el ángulo de visión es pequeño y la imagen no es estereoscópica. Este último inconveniente puede salvarse si utilizamos los lentes estereoscópicos. En los sistemas de sobremesa (no inmersivos), el usuario contempla el mundo virtual como si estuviera viéndolo desde una ventana, pero puede interactuar con el mundo virtual y desplazarse por él.

Los sistemas proyectivos combinan todas las ventajas de los sistemas de sobremesa, como la excelente calidad de imágenes e incluyen la posibilidad de inmersión del usuario en el mundo virtual. Para ello, se emplean cabinas en las que el usuario se introduce y en cierto modo se ve sumergido en el mundo virtual dentro de un vehículo.

A continuación se mencionan los dispositivos especiales de presentación más utilizados en la navegación por los mundos virtuales.

HMD, Head Mounted Display

Los HMD, Head Mounted Display, son dispositivos en forma de casco o visera, que el usuario lleva mientras se encuentra dentro del mundo virtual y en los que se incorporan un par de pantallas de cristal líquido pasivas en color, que le permiten una visión estereoscópica de la escena. El dispositivo se complementa con unos auriculares estereofónicos para el sonido y el sensor de posición, adherido al HMD, que permite conocer la ubicación y orientación de la cabeza del usuario.

El cometido del HMD es doble, ya que además de presentarnos una visión estereoscópica, también nos proporciona la sensación de inmersión impidiéndonos la visión del mundo exterior. La sensación de relieve se consigue mostrando en cada una de las dos pantallas imágenes ligeramente

distintas, que corresponden con la imagen del mundo virtual que cada uno de los dos ojos percibiría. Respecto a la sensación de inmersión, es necesario que el HMD incorpore un dispositivo de localización que le permita determinar la orientación de la cabeza del usuario y así, el sistema pueda calcular de modo continuo la perspectiva correcta del mundo virtual.

Una característica importante de los HMD es que son muy ligeros y poco voluminosos, requisitos indispensables desde el momento en que el usuario debe llevar puesto el dispositivo. Las pantallas tienen lentes de aumento para agrandar el tamaño de las imágenes, y el ángulo de visión que se consigue es de 110° horizontal y 90° vertical. Asimismo, se suelen incorporar elementos de filtrado para reducir la sensación de granularidad que las pantallas de cristal líquido proporcionan. Desafortunadamente, las pantallas del HMD presentan problemas de brillo y contraste, impactando en una escasa resolución y una calidad de imagen muy baja.

Lentes estereoscópicos

Existen dos tipos de lentes estereoscópicos, los pasivos y los activos, los cuales se emplean con frecuencia en los sistemas de sobremesa y proyectivos. Los lentes estereoscópicos pasivos son lentes de polarización circular, donde cada cristal está polarizado en un sentido. Delante del monitor se coloca un filtro que se sincroniza con la señal de refresco de pantalla (la señal que indica que el sistema ha terminado de sintetizar la siguiente imagen). Cuando se muestra la primera imagen, el filtro la polariza en un sentido, de forma que la imagen sólo atravesará uno de los dos cristales de los lentes (el que esté polarizado en el mismo sentido). Al mostrarse la segunda imagen, el filtro conmuta y la polariza en sentido opuesto, con lo que la imagen atravesará el otro cristal de los lentes.

Los lentes estereoscópicos activos tienen un par de pantallas de cristal líquido y un dispositivo receptor que se sincroniza con la señal de refresco de la pantalla. Ambas pantallas de cristal se vuelven opacos alternativamente, de acuerdo con las señales recibidas, de forma que se interrumpe la visión por el ojo correspondiente. Cuando en el monitor se está mostrando la imagen correspondiente al ojo izquierdo, los lentes impiden el paso de la luz hacia el ojo derecho, y viceversa.

Dispositivos de audio

El sonido dentro de las aplicaciones de realidad virtual tiene muy importantes funciones. En primer lugar, tiene la función informativa, común a casi todos los tipos de aplicaciones, informando al usuario acerca de la ocurrencia de determinados eventos, como por ejemplo, la confirmación del pulsado correcto de un botón. Como segunda función está la metafórica, es decir, el sonido puede emplearse para traducir una serie de datos a un formato fácilmente entendible por el usuario, por ejemplo, en visualización científica se emplea el sonido para indicar magnitudes de campos magnéticos, de forma que el sonido se va haciendo más o menos agudo a medida de que el usuario se desplaza en el mundo virtual.

Como tercer función del sonido está la artística, donde el sonido se emplea en forma de música de fondo que contribuye a incrementar el atractivo de una aplicación, e incluso a influir en el estado

de ánimo del usuario. Por último, el sonido desempeña una función descriptiva, es decir, son sonidos realizados que describen sucesos que tienen lugar dentro del mundo virtual, por ejemplo, el sonido de un vaso al romperse.

Para la creación de estas secuencias sonoras existen diversos métodos, por ejemplo, una secuencia sonora puede ser previamente grabada y después reproducida, o bien, puede emplear un sintetizador para su creación. De la primera forma, es como si se reprodujera una melodía en un tocadiscos, alguna melodía no existente sería imposible tocarla. En cambio, en el sintetizador tenemos la posibilidad de ordenarle que en cualquier momento genere una determinada nota con los parámetros que queramos: volumen, duración, instrumento que debe tocarla, etc.

Una vez obtenidos los sonidos que deseamos producir, no bastará con integrarlos a las aplicaciones de realidad virtual, sino que además es necesario que el usuario perciba el sonido, lo identifique y trate de localizarlo en un espacio tridimensional. Para lograr esto se han creado técnicas de sonido 3D que permiten asociar sonidos con imágenes, y ofrecen la posibilidad de que el usuario se percate de la proximidad de los objetos, o de la ocurrencia de eventos, que caen fuera de su campo actual de visión.

La técnica más efectiva que se emplea es la de localización del sonido inventada en 1988 por Scott Foster, y está basada en transmitir el sonido a los oídos en diferentes tiempos y con intensidades ligeramente diferentes, es decir, el sonido llegará con mayor rapidez al oído situado más cerca de la fuente de sonido que al otro. Este pequeño retardo de tiempo es muy pequeño pero suficiente como para extraer una primera información acerca de la posición relativa de la fuente de sonido con respecto a ambos oídos. Además, gracias a los pliegues que tenemos en la oreja, es posible filtrar la señal sonora de una forma que dependa de la dirección de la que el sonido provenga. Por tanto, con hacer escuchar al usuario, mediante unos auriculares, las dos señales ligeramente diferentes, éste será capaz de determinar con precisión la dirección de la fuente virtual del sonido.

Otros dispositivos de salida

Además de los dispositivos mencionados anteriormente, existen otros que se integran a los sistemas de realidad virtual, aumentando el realismo de las aplicaciones y la naturalidad de la interfaz con el ordenador. Algunos de ellos son las plataformas móviles y los dispositivos táctiles.

Las plataformas móviles tienen por objeto, simular los movimientos de navegación para conseguir un mayor realismo y para compensar en parte las causas productoras de mareo. Las cabinas son las plataformas que se usan con mayor frecuencia, ya que dependiendo de los movimientos que el usuario realice, el sistema hará inclinar la plataforma hacia un lado o hacia otro. Es importante observar que las plataformas móviles no simulan las traslaciones por el mundo virtual, sino las posiciones que adopta el usuario. Por ejemplo, supongamos que un usuario se encuentra volando a ras de suelo a través del mundo virtual. Mientras no efectúe una maniobra de ascenso, la plataforma no se moverá; al momento de recibir la orden de ascenso, la plataforma se inclinará hacia atrás para reflejar la nueva posición que toma el usuario. Los mismo sucederá cuando el usuario cambie de ruta hacia la derecha, hacia la izquierda o cuando descienda.

Existe la posibilidad adicional de simular la inercia de un cuerpo a través de su peso. Por ejemplo, si el usuario volando a ras de suelo experimenta un incremento de velocidad de su nave, la plataforma podrá inclinarse hacia atrás. Como la imagen le indica al usuario que va al ras del suelo, su cerebro interpreta la inclinación hacia atrás de la plataforma como una aceleración, lo mismo sucedería con la inclinación de la plataforma hacia adelante simulando el frenado, o inclinándose hacia la derecha o izquierda simulando la fuerza centrífuga de un giro.

Los dispositivos táctiles son también muy importantes en las aplicaciones de realidad virtual, permitiendo sentir al usuario el contacto con algún objeto virtual cercano y una contribución aún mayor de realismo. Desafortunadamente estos dispositivos adolecen de dos efectos fundamentales. El primero es su gran tamaño y baja frecuencia de activación, que impiden simular rugosidades finas de las superficies. La segunda es que la sensación táctil es algo irreal, ya que no está acompañada de realimentaciones de carácter kinestésico como podrían ser: las resistencia de los objetos, su peso, la viscosidad, entre otras.

1.3.5 DISPOSITIVOS DE ENTRADA EMPLEADOS EN APLICACIONES DE REALIDAD VIRTUAL

Los dispositivos de entrada en los sistemas de realidad virtual juegan un papel muy importante ya que son los que permiten al usuario transmitir sus órdenes al sistema, y son los que se encargan de controlar aspectos como la posición y orientación del usuario. Por lo tanto, los dispositivos de entrada se dividen en dispositivos de localización y dispositivos de control.

Dispositivos de localización

La función primordial de los dispositivos de localización es medir la posición y orientación de cualquier objeto o parte del mismo, en el espacio de tres dimensiones. Las aplicaciones básicas para este tipo de dispositivos son las siguientes:

a) Detección de la orientación del usuario: En todo sistema de realidad virtual es necesario conocer la posición y orientación del usuario dentro del mundo virtual, con el fin de sintetizar las imágenes desde el adecuado punto de vista. En los sistemas inmersivos, en los cuales se emplean cascos o sistemas binoculares, el control de la orientación debe ser automático, es decir, si el usuario vuelve la cabeza a un lado, su orientación dentro del mundo virtual debe cambiar correspondientemente. En los sistemas no inmersivos, el cambio de orientación dentro del mundo virtual es un cambio en la orientación de la ventana, no de la cabeza del usuario (el usuario siempre mira hacia la misma dirección).

b) Detección de la posición del usuario: Los dispositivos de localización permiten, además, medir posiciones absolutas, y no sólo orientaciones. Esto es, se pueden conocer los desplazamientos que el usuario realiza, y actualizar su posición dentro del mundo virtual de acuerdo a ellos. El problema que presentan estos dispositivos es que su alcance es muy limitado, restringiendo considerablemente los movimientos del usuario; por ello se opta en emplear otros dispositivos de control para desplazamientos más grandes, como por ejemplo las palancas de mando utilizadas en juegos de video, también llamadas joysticks.

c) Digitalización de objetos: Debido a la función que desempeñan los dispositivos de localización, es posible crear objetos reales en mundos virtuales, midiendo las posiciones en diversos puntos de la superficie del objeto a crear. Las medidas obtenidas son introducidas en el ordenador, el cual de manera automática crea el modelo y lo deja listo para su manipulación. Particularmente, estos dispositivos se conocen como digitalizadores.

Dispositivos de control

Los dispositivos de control son los que permiten al usuario realizar el control explícito de las aplicaciones e interactuar con el mundo virtual, como ejemplos podemos citar a los guantes, ratones 3D y joysticks 3D, los cuales se explican más adelante. Entre las órdenes que permiten transmitir estos dispositivos tenemos las siguientes:

- Comandos de navegación, mediante los cuales el usuario le indica al sistema que desea desplazarse dentro del mundo virtual.
- Comandos de interacción, que le permiten al usuario interactuar con los elementos que componen el mundo virtual.
- Comandos de manipulación del estado del mundo virtual, mediante los que el usuario puede regular determinadas variables de estado que afectan al funcionamiento global del sistema.

Guante (Glove)

Los guantes son dispositivos utilizados en las aplicaciones de realidad virtual para detectar la posición de la mano del usuario. Lo que los guantes miden es la posición relativa de los dedos, detectando el grado de flexión de cada uno y la separación existente entre dedos consecutivos.

Los guantes se suelen realizar en nylon o lycra, e incorporan una serie de dispositivos sensores que permiten determinar el grado de flexión de cada uno y la separación entre los dedos. Los datos suministrados por el guante se recogen en el ordenador, a través del puerto serie, y se procesan para extraer la información sobre las posiciones relativas de cada articulación. Incorporando un dispositivo de localización al guante, se puede complementar dicha información con la relativa a la posición absoluta de la mano.

El concepto de los guantes es fácilmente extensible hacia otras partes del cuerpo, logrando medir flexiones de rodillas, codos, muñecas, hasta lograr un dispositivo que determina la postura del usuario. Estos dispositivos tienen diversas aplicaciones en medicina.

Ratones 3D y Joysticks 3D

Los dispositivos existentes para navegación y control, no bastaban para el entorno virtual; por ello se desarrollaron dispositivos específicos para los mundos tridimensionales. Los ratones tridimensionales (mouse 3D) son una extensión natural de los ratones convencionales, así un ratón

permite controlar el movimiento de cualquier objeto en un espacio virtual. Para determinar la posición del ratón se tienen localizadores ultrasónicos, magnéticos y mecánicos.

Los joysticks tridimensionales (joysticks 3D) combinan la función de un ratón tridimensional además de la posibilidad de medir la fuerza que el usuario aplica al movimiento, logrando de esta manera calcular la dirección del movimiento y la velocidad de éste.

Dispositivos simuladores

Los dispositivos de entrada de muchos simuladores son mecánicos, debido a la necesidad de la similitud de los controles reales, transmitiendo a través de estos dispositivos las órdenes al sistema. Por esta razón son de gran importancia en el preparación y entrenamiento, de personal, así como en el desarrollo de sistemas de entretenimiento.

Otros dispositivos de entrada

Algunos otros dispositivos de entrada a los sistemas de realidad virtual son los dispositivos de adquisición de datos, los cuales se emplean para efectuar mediciones de parámetros del mundo real; dispositivos de reconocimiento de voz, los cuales se encuentran muy desarrollados actualmente; dispositivos para el reconocimiento de imágenes, que pueden ser utilizados para el reconocimiento óptico de gestos y movimientos fáciles; dispositivos midi, para la síntesis de secuencias musicales, empleando un canal midi para el control de secuencias sonoras en tiempo real; y controladores de fuerza/giro, los cuales funcionan como un joystick agregando realimentación kinestésica.

1.3.6 CREACIÓN DE UN MUNDO VIRTUAL

Un mundo virtual pasa por distintas etapas que determinan su ciclo de vida de la misma forma que cualquier otro software. Las etapas son las siguientes:

- Propuesta de ambiente
- Objetivo
- Limitaciones
- Análisis
- Diseño abstracto
- Diseño concreto
- Implementación
- Pruebas
- Mantenimiento

En la propuesta de ambiente se especifica qué se espera del mundo virtual justificando su creación y la necesidad de usar realidad virtual. El objetivo especifica cuáles son las metas y propósitos del ambiente, qué es lo que se espera obtener. En base al objetivo se crea una lista de

limitaciones tanto prácticas como de diseño, que determinan el ámbito del problema y el alcance del sistema.

El análisis busca obtener las necesidades que se tienen (requerimientos) para realizar el sistema; haciendo caso a los recursos con los que se cuentan.

El diseño abstracto consiste en realizar diagramas que especifiquen el funcionamiento del mundo virtual y sus elementos; por otro lado, el diseño concreto aterriza los conceptos dados por los diagramas llevándolos a un contexto más práctico; es decir, preparándolos para ser implementados.

Durante la etapa de implementación y al finalizar la construcción del sistema, es necesario realizar pruebas que verifiquen el correcto desempeño del mismo, para que en caso de que ocurriera algún error, fuera posible realizar correcciones. Todo software lleva una fase de mantenimiento que permite realizar mejoras y correcciones, conforme al uso del sistema.

El desarrollo de un mundo virtual obedece principalmente al objetivo y los recursos con los que se cuenta; es posible que durante su creación no se realizaran todos los pasos anteriormente descritos, ya que por especificaciones del problema algunos de ellos ya se encuentran descritos.

Algunos conceptos relacionados con la realidad virtual.

- Ciberespacio: Término creado por William Gibson, escritor de ciencia ficción, con el cual se describían los espacios tridimensionales sintetizados por computadora. Comúnmente se utiliza este término en lugar de realidad virtual, pero se distinguen uno del otro de la siguiente manera: la realidad virtual incluye experiencias, y el ciberespacio sólo visualiza información sobre el ambiente virtual.
- Estereoscópico: Efecto que proporciona un efecto de tridimensionalidad; se consigue al enviar a cada ojo una imagen ligeramente diferente (como ocurre en la realidad), de tal forma que al verlas juntas da una sensación de profundidad.
- Gráficas inmersivas: Reproducciones multidimensionales que al estimular todos los sentidos hace que se pierda la línea entre la realidad y la ilusión, permitiendo que los usuarios queden inmersos dentro del mundo virtual.
- Realimentación de fuerza: Sensación de resistencia proporcionada al usuario por medio de unos actuadores, que no sólo dan la sensación de contacto, sino que también incluye la sensación de la fuerza aplicada.
- Realimentación táctil: Realimentación dirigida al usuario por medio del sentido del tacto, que se distingue de la realimentación de fuerza porque sólo incluye la sensación de tacto no así la de la fuerza involucrada.
- Simulación: Proceso mediante el cual se generan condiciones de ensayo aproximadas a las condiciones reales o condiciones de operación reales de algún sistema.
- Tiempo real: Tiempo entre la entrada de los datos y la salida de la solución, donde la respuesta a la entrada es lo suficientemente rápida como para afectar las entradas posteriores.
- Virtualización: Proceso mediante el cual un humano interpreta una impresión sensorial como un objeto en un entorno distinto al entorno en el que el objeto existe físicamente.

1.4 COMPILADORES

1.4.1 ¿QUÉ ES UN COMPILADOR?

Un compilador es un programa que traduce un lenguaje en otro. Para ello, el compilador lee el programa escrito en 'X' lenguaje, denominado lenguaje fuente, y obtiene como salida un programa equivalente escrito en otro lenguaje, denominado lenguaje objeto (ver figura I.1).

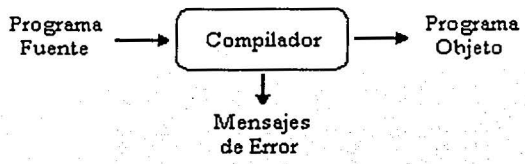


Figura I.1 Un compilador.

Como parte importante de este proceso de traducción, el compilador informa al usuario de la presencia de errores en el programa fuente.

El proceso de compilación consta de dos partes: el análisis; y la síntesis del programa fuente. Durante la parte del análisis, el compilador clasifica cada elemento del programa fuente en varias categorías o clases, y crea con ellos una representación intermedia del programa fuente. La parte de síntesis toma esta representación intermedia y construye el programa objeto.

A primera vista, la diversidad de compiladores puede parecer abrumadora; existen diversos lenguajes de programación y nuevos lenguajes especializados en todas las áreas de la informática hacen su arribo. De igual manera, los lenguajes objeto son variados; un lenguaje objeto puede ser otro lenguaje de programación o un lenguaje de máquina para cierto microprocesador.

1.4.2 CONTEXTO DE UN COMPILADOR

Además de un compilador, se pueden necesitar otros programas para crear un programa objeto ejecutable. Por ejemplo, un programa fuente puede estar dividido en varios módulos almacenados en diferentes archivos. La tarea de reunir el programa fuente a menudo se confía a un programa distinto, llamado preprocesador. El preprocesador también puede expandir abreviaturas, llamadas macros, a proposiciones del lenguaje fuente.

En otras situaciones, el programa objeto creado por algunos compiladores puede requerir procesamiento adicional antes de poderlo ejecutar. Éste es el caso de algunos compiladores que crean código en lenguaje ensamblador, el cual debe ser traducido por un ensamblador a código de máquina y enlazado con algunas rutinas de biblioteca para producir el código que realmente se ejecute en la máquina (ver figura I.2).

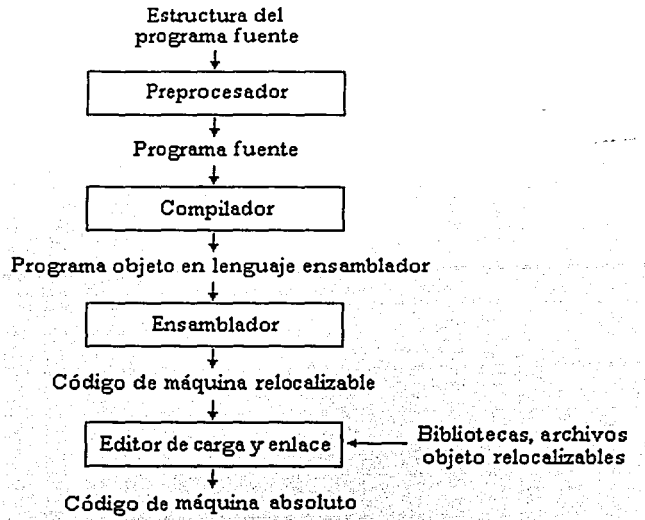


Figura I.2 Sistema para el procesamiento de un lenguaje.

1.4.3 FASES DE UN COMPILADOR

Conceptualmente, un compilador opera en fases, cada una de las cuales transforma al programa fuente de una representación en otra (ver figura I.3).

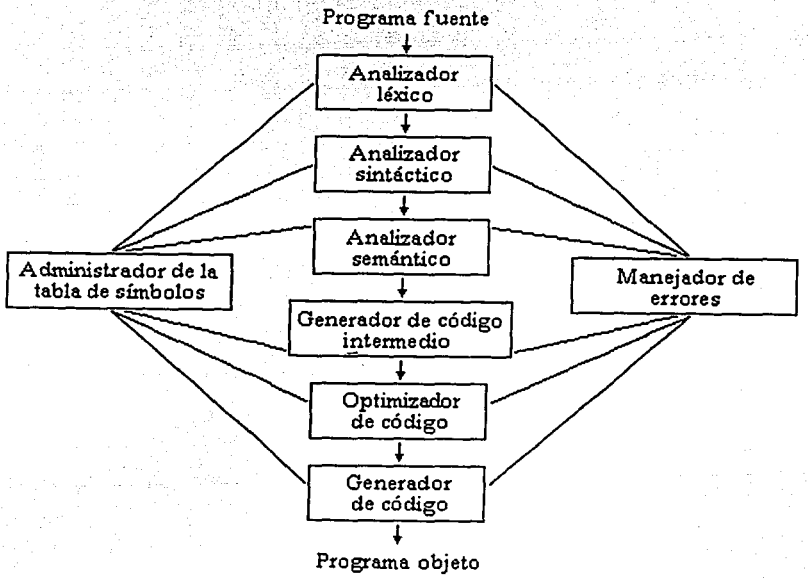


Figura I.3 Fases de un compilador.

En la práctica, se pueden agrupar algunas fases y las representaciones intermedias entre las fases agrupadas no necesitan ser construidas explícitamente.

Análisis del programa fuente

Por lo general, la parte de análisis involucra tres fases: el análisis léxico, el análisis sintáctico y el análisis semántico.

1. Análisis léxico. El programa fuente es revisado carácter a carácter con el fin de reconocer y extraer los símbolos básicos del lenguaje, tales como enteros, decimales, palabras reservadas, operadores e identificadores; dentro de los identificadores encontramos: nombres de variables, de etiquetas o de funciones.

2. Análisis sintáctico. La estructura sintáctica de los símbolos del programa fuente es verificada por el analizador sintáctico, es decir, se revisa la estructura del programa conforme a las reglas básicas que generan al lenguaje fuente.

3. Análisis semántico. El análisis semántico revisa el programa fuente para tratar de encontrar errores semánticos, y reúne la información sobre los tipos para la fase de generación de código. Un componente importante del análisis semántico es el comprobador de tipos. El comprobador de tipos verifica si cada operador tiene operandos permitidos por la especificación del lenguaje fuente. Por ejemplo, indica error cuando se utiliza un número real como índice de una matriz, o aplica conversiones de entero a real cuando se aplica un operador aritmético binario a un número entero y a un número real.

Administración de la tabla de símbolos

Una función esencial de un compilador es registrar los identificadores utilizados en el programa fuente y reunir información sobre los distintos atributos de cada identificador. Estos atributos pueden proporcionar información sobre la memoria asignada a un identificador, su tipo, su ámbito (la parte del programa donde tiene validez) y, en el caso de nombres de procedimientos, cosas como el número y tipos de los argumentos, el método de pasar cada argumento y el tipo que devuelve el procedimiento si es que devuelve algo.

La mayoría de las fases del compilador introducen información sobre los identificadores en la tabla de símbolos; esta información es utilizada por la misma fase o por fases posteriores de diversas formas. Por ejemplo, cuando se está haciendo el análisis semántico y la generación de código intermedio, se necesita saber cuáles son tipos de los identificadores para comprobar si el programa fuente los utiliza de forma válida y así poder generar las operaciones apropiadas con ellos.

Manejo de errores

Cada fase puede encontrar errores. Sin embargo, después de detectar un error, cada fase debe tratar de alguna forma ese error, para poder continuar la compilación y la detección de más errores

en el programa fuente. Un compilador que se detiene cuando encuentra el primer error no resulta tan útil. Las fases de análisis sintáctico y semántico por lo general manejan una gran porción de los errores detectables por el compilador.

La fase léxica puede detectar errores en los caracteres de la entrada si éstos no forman ningún componente léxico del lenguaje. Los errores donde la cadena de componentes léxicos violan las reglas de estructura (sintaxis) del lenguaje son determinados por la fase de análisis sintáctico. Durante el análisis semántico el compilador intenta detectar construcciones que tengan una estructura sintáctica correcta, pero que no tengan significado para la operación implicada. Por ejemplo, existirá un error semántico si se intenta sumar un identificador de tipo entero y el nombre de un procedimiento.

Generación de código intermedio

Después de los análisis léxico, sintáctico y semántico, algunos compiladores generan una representación intermedia explícita del programa fuente. Se puede considerar esta representación intermedia como un programa para una máquina abstracta. Esta representación intermedia debe tener dos propiedades importantes: debe ser fácil de producir, y fácil de traducir al programa objeto.

Optimización de código

La fase de optimización de código trata de mejorar el código intermedio, de modo que resulte un código de máquina más rápido de ejecutar.

Generación de código

La fase final de un compilador es la generación de código objeto, que por lo general consiste en código de máquina relocalizable o código ensamblador.

Agrupamiento de fases

En la práctica, las fases de un compilador suelen agruparse. Por lo regular, se agrupan en dos: una etapa inicial y una final. La etapa inicial comprende aquellas fases que dependen principalmente del lenguaje fuente como el análisis léxico, el sintáctico, el semántico, la creación de la tabla de símbolos y la generación de código intermedio. Además, incluye el manejo de errores correspondiente a cada fase.

La fase final incluye aquellas fases que no dependen del lenguaje fuente, sino sólo de la representación intermedia. Entre estas fases encontramos la fase de optimación de código y la de generación de código, junto con sus manejadores de errores y su vinculación con la tabla de símbolos.

Un adecuado agrupamiento de fases conducirá a la construcción de un compilador con menor número de pasadas. Esta característica es deseable en cualquier compilador dado que cada pasada implica la lectura de un archivo de entrada y la escritura de un archivo de salida, lo cual lleva tiempo.

Por otra parte, un compilador de muy pocas pasadas tampoco suele ser la mejor opción, pues los códigos intermedios y la tabla de símbolos empezarán a aumentar de tamaño y su manejo será cada vez más complejo. Además, será necesario tener en memoria todo el programa fuente, ya que una fase puede requerir información adicional de la proporcionada por fases previas.

1.4.4 PROGRAMAS RELACIONADOS CON UN COMPILADOR

Preprocesadores

Los preprocesadores producen la entrada para un compilador, y pueden realizar las siguientes funciones:

- Procesamiento de macros. Un preprocesador permite al usuario definir macros, que son abreviaturas de construcciones más grandes. Los procesadores de macros tratan dos clases de proposiciones: definición de macros y uso de macros.

  Las definiciones de macros normalmente se indican con algún carácter exclusivo o palabra clave, como *define* o *macro*. Constan de un nombre para la macro que se está definiendo, y de un cuerpo, que constituye su definición. A menudo los procesadores de macros admiten parámetros formales en su definición, esto es, símbolos que se reemplazarán por valores.

  El uso de una macro consiste en dar nombre a la macro y proporcionar parámetros reales, es decir, valores para sus parámetros formales. El procesador de macros sustituye los parámetros reales por los parámetros formales del cuerpo de la macro; después, el cuerpo transformado reemplaza el uso de la propia macro.

- Inclusión de archivos. Un preprocesador también puede insertar archivos de encabezados en el texto del programa. Los archivos de encabezados contienen definiciones de funciones adicionales a las de nuestro programa.

- Preprocesadores racionales. Estos preprocesadores enriquecen los lenguajes antiguos con recursos más modernos de flujo de control y de estructuras de datos. Por ejemplo, se podrían incorporar macros para construcciones *while* o *if* en lenguajes que nos las tuvieran.

Ensambladores

Algunos compiladores producen un tipo de código objeto llamado código ensamblador. El código ensamblador es una versión mnemotécnica del código de máquina, donde se utilizan nombres tanto para operaciones como para direcciones de memoria, en lugar de códigos binarios.

Por lo tanto, para que el código ensamblador pueda ejecutarse, requiere un procesamiento adicional, el cual es efectuado por un programa denominado ensamblador.

Otros compiladores realizan el trabajo del ensamblador, produciendo código de máquina relocalizable[1] que se puede pasar directamente al editor de carga y enlace.

Cargadores y Editores de enlace

Por lo general, un programa cargador realiza las dos funciones: carga y edición de enlaces. El proceso de carga consiste en tomar el código de máquina relocalizable, modificar las direcciones relocalizables y ubicar las instrucciones y los datos modificados en las posiciones apropiadas de la memoria.

El proceso de edición de enlaces permite formar un solo programa a partir de varios archivos de código de máquina relocalizable. Algunos de estos archivos pudieron ser generados tras varias compilaciones, y otros pueden ser archivos de biblioteca de rutinas proporcionados por el sistema.

1.4.5 HERRAMIENTAS PARA LA CONSTRUCCIÓN DE COMPILADORES

- Generadores de analizadores léxicos. Estas herramientas permiten generar de manera automática analizadores léxicos a partir de expresiones regulares. La organización básica del analizador léxico resultante será en realidad un autómata finito[2]. Un ejemplo de este tipo de programas es el compilador de LEX que genera un reconocedor de expresiones regulares mediante un autómata finito eficiente.

- Generadores de analizadores sintácticos. Estos generadores producen analizadores sintácticos a partir de una entrada fundamentada en una gramática independiente del contexto.

- Dispositivos de traducción dirigida por la sintaxis. Estas herramientas producen grupos de rutinas que recorren el árbol de análisis sintáctico generando código intermedio.

---

[1] *Código de máquina relocalizable.* Cuando se ejecutan varios programas en una computadora, la memoria del sistema debe compartirse entre todos ellos. Esto representa un inconveniente para el programador, pues no puede escoger la dirección en memoria donde se cargará su programa. Entonces, el programador hace su programa asumiendo que la dirección en la que se comenzará a cargar el programa es la dirección cero de memoria. Al cargarse el programa para ejecutarse, las direcciones se relocalizan, o sea, se les suma la dirección real de comienzo del programa. Un ensamblador no genera las direcciones absolutas o reales cuando hay relocalización, pero añade información al "código objeto" para que, más adelante, el cargador pueda llevar a cabo la relocalización. Este programa objeto se conoce como un programa relocalizable ("relocatable program"). El ensamblador genera las direcciones relativas al comienzo del programa (dirección 0), y produce instrucciones para que el cargador ajuste o relocalice las direcciones.

[2] *Autómata Finito.* Un autómata finito es un reconocedor de un lenguaje, que toma como entrada una cadena de caracteres y responde 'Sí', si esa cadena es un componente léxico del programa, o 'No', si no lo es. Recuerde que los componentes léxicos (tokens) son secuencias de caracteres que tienen un significado colectivo para cierto lenguaje.

- Generadores automáticos de código. Estas herramientas toman un conjunto de reglas que definen la traducción de cada operación del lenguaje intermedio al lenguaje objeto. Estas reglas deben incluir suficiente detalle para poder manejar los distintos métodos posibles de acceso a los datos.

- Dispositivos para análisis de flujo de datos. Mucha de la información necesaria para efectuar una buena optimación de código implica hacer un análisis de flujo de datos, que consiste en la recolección de información sobre la forma en que se transmiten los valores de una parte de un programa a cada una de las otras partes.

1.5 LENGUAJES DE PROGRAMACIÓN

1.5.1 DEFINICIÓN DE LENGUAJE

Un lenguaje de programación, en informática, es cualquier lenguaje artificial que puede utilizarse para definir una secuencia de instrucciones que son procesadas por una computadora. Es decir, un lenguaje de programación está formado por una secuencia de instrucciones y un conjunto de reglas que definen la manera como se forman esas instrucciones. Para que una computadora pueda realizar las funciones y las operaciones que deseamos, es necesario suministrar las instrucciones adecuadas debidamente agrupadas y ordenadas. Este conjunto de instrucciones constituyen lo que se denomina un programa o aplicación. Para que las instrucciones sean comprensibles por la computadora, y debido a la propia estructura física de la misma, los programas deben estar expresados como combinaciones de ceros y unos, es decir, en lenguaje de máquina. Debido a las graves dificultades que presenta la programación en lenguaje de máquina para el ser humano, se han desarrollado instrucciones y reglas que permiten indicar al procesador la secuencia de instrucciones deseadas, pero en un lenguaje más accesible.

Los lenguajes permiten escribir las operaciones que se requieren realizar para resolver un problema de modo parecido a como se escribiría convencionalmente, es decir, redactando adecuadamente el algoritmo de resolución del problema.

1.5.2 CLASIFICACIÓN DE LOS LENGUAJES DE PROGRAMACIÓN

Lenguaje máquina

El lenguaje máquina es el único lenguaje que entiende directamente la computadora. Utiliza un alfabeto binario que consta de los dos únicos símbolos (0 y 1) denominados bits (abreviatura inglesa de dígitos binarios). Sus instrucciones son cadenas binarias, es decir, series de caracteres de 0's y 1's, que especifican la operación que la computadora realizará. A estas cadenas se les denomina instrucciones de máquina o código máquina.

El lenguaje máquina fue el primer lenguaje utilizado en la programación de computadoras, pero dejó de utilizarse por su dificultad y complicación, siendo sustituido por otros lenguajes más fáciles de aprender y utilizar, que además reducen la posibilidad de cometer errores.

Ventajas del lenguaje máquina: posibilidad de cargar (transferir un programa a la memoria) sin necesidad de traducción posterior, lo que supone una velocidad de ejecución superior a cualquier otro lenguaje de programación.

Desventajas del lenguaje máquina: dificultad y lentitud en la codificación. Gran dificultad para verificar y poner a punto los programas. Los programas sólo son ejecutables para el procesador que fueron pensados. En la actualidad, las desventajas superan a las ventajas, lo que hace prácticamente no recomendables a los lenguajes máquina.

Lenguajes de bajo nivel

A principios de la década de los 50 y con el fin de facilitar la labor de los programadores, se desarrollaron códigos mnemotécnicos para las operaciones y direcciones simbólicas propias del lenguaje máquina. A estos códigos mnemotécnicos se les denominó lenguaje ensamblador.

La computadora sigue utilizando el lenguaje máquina para procesar los datos, pero existen programas ensambladores que traducen los códigos mnemotécnicos a sus equivalentes en el lenguaje máquina.

Un lenguaje de bajo nivel es más fácil de utilizar que el lenguaje máquina, pero es dependiente de la arquitectura del procesador. Los programas en lenguaje ensamblador son más fáciles de modificar que los programas en lenguaje máquina.

Lenguajes de nivel medio

Poseen características de alto y bajo nivel, por lo que se puede obtener velocidades de proceso muy similares al bajo nivel, control total del equipo y además, facilidades de programación.

Lenguajes de alto nivel

Se caracterizan por su similitud con los lenguajes humanos, por lo cual son más fáciles de aprender, entender y usar. Sus principales objetivos son:

- Humanizar las tareas de programación, acercando los lenguajes de programación al lenguaje coloquial (de conversación).
- Hacer compatibles las distintas computadoras por medio de la programación: Un programa escrito en un lenguaje de alto nivel puede ejecutarse en cualquier computadora.

Los lenguajes de alto nivel usan nombres simbólicos para representar datos, variables, direcciones de memoria, etc. Esto nos permite acercar las tareas de programación a los problemas, alejándolos de los detalles técnicos relacionados con el computador. Al usar instrucciones similares al lenguaje humano y dado que la computadora no es capaz de reconocer estas órdenes, es necesario el uso de un intérprete que traduzca el lenguaje de alto nivel a un lenguaje de bajo nivel que el sistema pueda entender.

Dentro de las ventajas que se tienen al usar lenguajes de alto nivel tenemos:

- Puede ser usado, después de algunas modificaciones, en distintos equipos.
- El tiempo de formación de los programadores es relativamente corto, en comparación con el necesario para aprender los lenguajes de nivel inferior.
- El programador no necesita conocer cómo funciona una computadora en específico para poder confeccionar sus programas.
- El tiempo necesario para codificar y poner a punto, es decir, los cambios y correcciones posteriores de un programa en lenguaje de alto nivel, es inferior al necesario en el caso de los lenguajes menos evolucionados.

Algunos de los inconvenientes de usar lenguajes de alto nivel son:

- El tiempo de ejecución es mayor.
- No se aprovechan las posibles ventajas de la arquitectura interna del sistema.
- Se incrementa el uso de memoria debido al uso de un intérprete.

1.5.3 GENERACIONES DE LENGUAJES DE PROGRAMACIÓN

Primera generación. Lenguaje máquina. Características:

- No requiere traducción alguna.
- La computadora es capaz de leerlo directamente.

Segunda generación. Lenguaje ensamblador. Características:

- Dependiente de la máquina.
- Requiere de una traducción.

Tercera generación. Lenguajes de nivel medio. Características:

- Requieren especificaciones de cómo realizar una tarea.
- Se debe especificar todas las posibles opciones.
- Requieren de un número grande de instrucciones.
- Los códigos pueden ser difíciles de leer, entender, mantener y depurar.
- Originalmente desarrollados para operaciones por lote.
- Orientados hacia archivos.
- Requieren de traducción y cada instrucción es convertida en varias instrucciones de máquina.

Cuarta generación. Lenguajes de alto nivel. Características:

- Requiere la especificación de la tarea a realizar (el sistema determina cómo efectuarla).
- Requiere traducción y cada instrucción en convertida en muchas instrucciones en lenguaje de máquina.

- Errores fáciles de localizar.
- Orientados hacia bases de datos.
- Orientados a procedimientos.

1.5.4 COMUNICACIÓN EN LA ROBÓTICA

El lenguaje siempre ha sido una vía eficaz de comunicación, la relación robot-hombre también debe de utilizar mecanismos para una comunicación eficaz. Entre las formas que existen de comunicación con los robots tenemos:

- Reconocimiento de palabras separadas. Actualmente este sistema es bastante primitivo y suele depender de quién habla. Estos sistemas pueden reconocer un conjunto de palabras concretas de un vocabulario limitado.

- Enseñanza y repetición. Es la comunicación más común utilizada en los robots industriales. Implica el enseñar al robot todos los movimientos que necesita realizar.

- Lenguajes de programación de alto nivel. Los lenguajes suministran una solución más general en la comunicación hombre-robot. Los lenguajes clásicos (FORTRAN, BASIC, PASCAL) no disponen de los comandos e instrucciones específicas que se necesitan para la programación en la robótica. Hasta ahora los lenguajes utilizados han sido diseñados para un modelo específico de manipulador o para una tarea concreta, por lo que en estos momentos no existe ningún lenguaje universal.

1.5.5 PROGRAMACIÓN EN ROBÓTICA

La programación que se emplea en robótica tiene diferentes tendencias. La tabla I.5 muestra los tipos de programación en robótica.

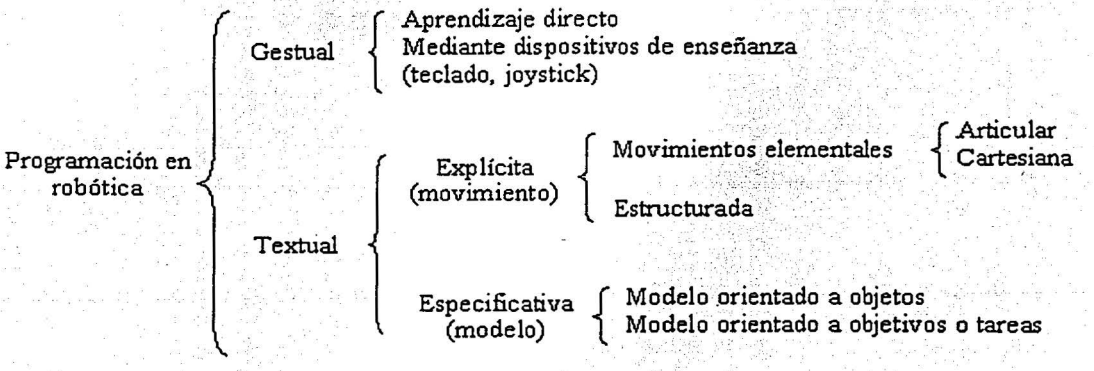


Tabla I.5 Modelos de programación usados en robótica.

Programación Gestual

Este tipo de programación exige el empleo del manipulador en la fase de enseñanza, por lo que se dice que es un proceso online[3].

La programación gestual no necesita que el usuario conozca un lenguaje de programación, simplemente debe habituarse al empleo de los elementos que constituyen el dispositivo de enseñanza. Los lenguajes de programación gestual, además de necesitar al propio robot en la confección del programa, carecen de adaptabilidad en tiempo real con el entorno y no pueden tratar, con facilidad, interacciones de emergencia.

Programación Textual

Las trayectorias del manipulador se calculan matemáticamente con gran precisión y se evita el posicionamiento manual. En esta labor no participa el robot por lo que es un proceso offline[4].

El programa queda constituido por un archivo de instrucciones o sentencias, cuya confección no requiere de la intervención del robot. El operador no define las acciones del brazo manipulador, sino que se calculan dentro del programa mediante el empleo de las instrucciones textuales adecuadas.

Según las características del lenguaje, pueden confeccionarse programas de trabajo complejos, con inclusión de saltos condicionales, empleo de bases de datos, posibilidad de creación de módulos operativos intercambiables, capacidad de adaptación a las condiciones del mundo exterior, etc.

La programación textual está dividida en dos grandes grupos de diferencias marcadas:

- Programación textual explícita. El programa consta de una secuencia de órdenes o instrucciones concretas que van definiendo con rigor las operaciones necesarias para llevar a cabo la aplicación. Se puede decir que la programación explícita engloba a los lenguajes que definen los movimientos punto por punto, similares a los de la programación gestual, pero bajo la forma de un lenguaje formal. Con este tipo de programación, la labor del tratamiento de las situaciones anormales, como colisiones, queda a cargo del programador. Existen dos niveles de programación explícita:
    - Nivel de movimiento elemental. Comprende los lenguajes dirigidos a controlar los movimientos del robot. Existen dos tipos:
        - Articular, cuando el lenguaje se dirige al control de los movimientos de las diversas articulaciones del brazo. Los lenguajes del tipo articular indican los incrementos angulares de las articulaciones; al no tener una referencia general de la posición de

[3] *Online*. En este caso, el término online se utiliza para indicar que el robot se encuentra encendido y funcionando, y procesando las instrucciones, de manera casi inmediata, que el programador le proporciona.

[4] *Offline*. A diferencia del proceso online, el proceso offline no necesita que el robot se encuentre en funcionamiento para poder programarlo.

las articulaciones con relación al entorno, es difícil relacionar al sistema con piezas móviles y obstáculos.

- Cartesiano, cuando el lenguaje define los movimientos relacionados con el sistema de manufactura, es decir, los del punto final del trabajo. Los lenguajes del tipo cartesiano utilizan transformaciones homogéneas, lo que hace que se independice a la programación del modelo particular del robot, puesto que un programa confeccionado para uno, en coordenadas cartesianas, puede utilizarse en otro, con diferentes coordenadas, mediante el sistema de transformación correspondiente.

- Nivel estructurado. Intenta introducir relaciones entre el objeto y el sistema del robot, para que los lenguajes se desarrollen sobre una estructura formal. Se puede decir que los lenguajes correspondientes a este tipo de programación adoptan la filosofía del PASCAL. Describen objetos y transformaciones con objetos, disponiendo, muchos de ellos, de una estructura de datos arborescente. El uso de lenguajes con programación explícita estructurada aumenta la comprensión del programa, reduce el tiempo de edición y simplifica las acciones encaminadas a la consecución de tareas determinadas. En los lenguajes estructurados, es típico el empleo de las transformaciones de coordenadas.

- Programación textual especificativa. Es una programación del tipo no procesal en la que el usuario describe las especificaciones de los objetos que se encontrarán en el entorno de trabajo del robot (universo), y también debe describir las tareas que hay que realizar sobre ellos. El sistema computacional para la programación textual especificativa debe de disponer del modelo del universo, por ejemplo un modelo geométrico del mundo donde se encuentra el robot. Este modelo será, normalmente, una base de datos según la clase de aplicación. El trabajo de la programación consistirá en la descripción de las tareas a realizar, lo que supone poder llevar a cabo trabajos complicados.

1.5.6 LENGUAJES DE PROGRAMACIÓN EN ROBÓTICA

A continuación se realiza una descripción de los lenguajes de programación más usados en robótica.

Gestual punto a punto

Los lenguajes más conocidos en programación gestual punto a punto son el FUNKY, creado por IBM para uno de sus robots, y el T3, original de CINCINNATI MILACROM para su robot T3. Los movimientos pueden tener lugar en sistemas de coordenadas cartesianas o cilíndricas, siendo posible insertar y borrar las instrucciones que se desee. Es posible, también, implementar funciones relacionadas con sensores externos, así como revisar el programa paso a paso, hacia delante y hacia atrás.

A nivel de movimientos elementales

Los lenguajes más conocidos para movimiento punto a punto son: ANORAD, EMILY, RCL, RPL, SIGLA, VAL, MAL. Todos ellos mantienen el énfasis en los movimientos primitivos, ya sea en coordenadas articulares, o cartesianas. En comparación, tienen como ventajas destacables, los saltos condicionales y saltos a subrutina, además de un aumento de las operaciones con sensores, aunque siguen manteniendo pocas posibilidades de programación offline.

Estos lenguajes son, por lo general, del tipo intérprete, con excepción del RPL, que tiene un compilador. La mayoría dispone de comandos de tratamiento a sensores básicos: tacto, fuerza, movimiento, proximidad y presencia. El RPL dispone de un sistema complejo de visión, capaz de seleccionar una pintura y reconocer objetos presentes en su base de datos

Estructurados de programación explícita

Los siguientes son ejemplos de lenguajes de programación explícita: AL, HELP, MAPLE, PAL, MCL, MAL EXTENDIDO. Con excepción de HELP, todos los lenguajes de este grupo están provistos de estructuras de datos del tipo complejo. Así, el AL utiliza vectores, posiciones y transformaciones; el PAL usa, fundamentalmente, transformaciones y el MAPLE permite la definición de puntos, líneas, planos y posiciones. Sólo el PAL y el HELP carecen de capacidad de adaptación sensorial. Los lenguajes AL, MAPLE y MCL, tienen comandos para el control de la sensibilidad del tacto de los dedos (fuerza, movimiento, proximidad, etc.). Además, el MCL posee comandos de visión para identificar e inspeccionar objetos.

Especificativa a nivel objeto

En este grupo se encuentran los siguientes lenguajes: RAPT, AUTOPASS, LAMA.

La filosofía de RAPT se basa en definir una serie de planos, cilindros y esferas, que dan lugar a otros cuerpos derivados. Para modelar a un cuerpo, se confecciona una biblioteca con sus rasgos más representativos. Enseguida, se definen los movimientos que ligan a los cuerpos a ensamblar (alinear planos, encajar cilindros, etc.).

El lenguaje AUTOPASS fue creado por IBM para el ensamblaje de piezas; utiliza instrucciones muy comunes en el idioma inglés. Precisa de un ordenador de varios megabytes de memoria y además, prevé colisiones y genera acciones a partir de las situaciones reales.

LAMA procede del laboratorio de Inteligencia Artificial del MIT (Massachusetts Institute of Technology), para el robot SILVER, orientándose hacia el ajuste de conjuntos mecánicos. Aporta más inteligencia que el AUTOPASS y permite una buena adaptación al entorno. La operatividad del LAMA se basa en tres funciones principales:

- 'Creación de la función de trabajo. Operación inteligente.
- Generación de la función de manipulación.

- Interpretación y desarrollo, en forma interactiva, de una estrategia de realimentación para la adaptación al entorno de trabajo.

En función de los objetivos

La filosofía de estos lenguajes consiste en definir la situación final del producto a fabricar, a partir de la cual se generan los planes de acción tendientes a conseguirla, obteniéndose finalmente, el programa de trabajo. Estos lenguajes prevén, incluso, la comunicación hombre-máquina por medio de la voz. Los lenguajes más conocidos de este grupo son: STRIPS y HILAIRE

STRIPS. Fue diseñado, en la Universidad de Stanford, para el robot móvil SHAKEY. Se basa en un modelo del universo ligado a un conjunto de planteamientos aritmético-lógicos que se encargan de obtener las subrutinas que conforman el programa final. Puede ser interpretado o compilado.

HILAIRE. Procedente del laboratorio de Automática y Análisis de Sistemas (LAAS) de Toulouse, está escrito en lenguaje LISP. Es uno de los lenguajes naturales más interesantes, por sus posibilidades de ampliación e investigación.

CAPÍTULO II
WTK (WORLD TOOL KIT)

2.1 INTRODUCCIÓN

WTK, World Tool Kit, es un API[5] para desarrollar aplicaciones con gráficas 3D en tiempo real; está orientado a la creación de mundos virtuales donde los objetos pueden tener asociados comportamientos y propiedades reales.

Dentro de las tareas que el API permite, tenemos las siguientes:

- proceso de render[6]
- lectura de periféricos (sensores)
- importación de geometrías
- reproducción de archivos de audio
- funciones de simulación
- interfaz de usuario
- conectividad

Debido a su compatibilidad con el lenguaje C, es posible crear una aplicación de WTK en plataformas como Win9x, WinNT, SUN, IRIX o Linux. Para este trabajo, la aplicación fue desarrollada sobre Windows 98 debido a que el API de WTK para Win9x había sido donada a la Universidad en fechas recientes, por lo que se tenía libre acceso a ella.

WTK es un API orientada a objetos; sin embargo, es posible escribir programas utilizando programación estructurada y/o programación orientada a objetos.

Las funciones en WTK han sido agrupadas en clases como a continuación se describe.

- Universe. Es la clase que contiene todos los elementos pertenecientes al mundo virtual, como son geometrías, nodos, puertos de vista, sensores, etc. Aún cuando es posible tener múltiples escenas y simulaciones a la vez, sólo hay un Universo.
- Geometries. Son objetos gráficos visibles durante una simulación, como un cubo, esfera, pirámide, cilindro, etc.
- Nodes. Son las partes más pequeñas dentro de la construcción de una escena gráfica; puede decirse que son sus bloques de construcción.
- Polígonos. Se entiende como polígono a varios segmentos de recta unidos sucesivamente entre sí formando una figura cerrada.
- Vértices. Se denomina vértice a un punto en el espacio 3D que pertenece a la intersección de dos segmentos que forman parte del perímetro de un mismo polígono.
- Lights. Objetos que son fuentes de luz dentro del mundo virtual.
- Viewpoints. Esta clase guarda información acerca de la posición y orientación del observador dentro del mundo virtual; desde esta posición serán proyectados y dibujados todos los elementos geométricos de la escena en la pantalla de la computadora.

[5] *API* (Application Program Interface/Interfaz para desarrollo de aplicaciones). Es un conjunto de rutinas, protocolos y herramientas para desarrollar software.

[6] *Render*. Proceso de generación de una imagen a partir de información tal como modelos matemáticos de objetos tridimensionales, luces, mapas de texturas, etc.

- Windows. Son los elementos donde los viewports realizan su despliegue.
- Sensores. Son los dispositivos de entrada utilizados en las aplicaciones de realidad virtual. Estos dispositivos generan posiciones, orientaciones y otro tipo de información, que es utilizada para controlar o asignar comportamientos a los objetos en el mundo virtual.
- Path. Estos objetos permiten que objetos geométricos sigan una determinada trayectoria.
- Tasks. Tareas asociadas a objetos que permiten modificar sus propiedades y comportamientos.
- Motion links. Los motion links permiten asignar cierto tipo de comportamiento a los objetos del mundo virtual. Este comportamiento consiste en seguir a otros objetos en movimiento.
- Sonido. Utilizado para reproducir archivos de audio.
- Interfaz de usuario. WTK es capaz de hacer uso de una propia interfaz de usuario independiente de la plataforma.
- Networking. Permiten realizar comunicación asíncrona entre mundos virtuales que pueden ser ejecutados en distintos sistemas.
- Puerto serial. Funciones para comunicación a través del puerto serie.

2.2 CICLO DE SIMULACIÓN

Internamente WTK sigue una serie de pasos que le permite llevar el control del mundo virtual realizando operaciones propias de sistema, así como también, operaciones indicadas por el usuario, es decir, cuenta con un ciclo de simulación. En la figura II.1 se muestra el ciclo de simulación de WTK.

Preprocesamiento. Realización de cálculos que permiten optimizar las indicaciones dadas por el usuario, como son geometrías, objetos, condiciones iniciales y mapa de ambiente. Este es un proceso que se realiza en tiempo no real[7] al inicializar por primera vez el ciclo de simulación.

Lectura de sensores. Cada mundo virtual necesita sensores para poder interactuar con el usuario, en este punto se recopila la información de cada uno de los dispositivos indicados como sensores. Dentro de los dispositivos con los que se pueden contar tenemos: mouse, teclado, guante y otros dispositivos de localización (posición/orientación).

Ejecución de tareas de sistema. WTK procesa la información recibida y la prepara para los siguientes pasos del ciclo de simulación.

Actualizar objetos. Con el uso de la lectura de los sensores, los objetos que pertenecen al mundo son actualizados. La actualización consiste en modificar las estructuras lógicas que contienen la configuración de los objetos.

[7] *Tiempo real.* Referente a cálculos cuyo tiempo de realización es tan corto que se considera inmediata la obtención del resultado.

Ejecución de tareas de usuario. En WTK el usuario es capaz de añadir funciones especiales a los objetos, con el fin de que éstos modifiquen su comportamiento o modifiquen el comportamiento de otros objetos; en esta etapa dichas funciones son ejecutadas.

Guardar configuración de geometrías. Es posible guardar la configuración de las geometrías en cada paso del ciclo de simulación para su posterior reproducción, por lo que en este paso es guardada la posición de los objetos seleccionados.

Render. Se muestra gráficamente la nueva proyección de la configuración de los objetos en el mundo virtual.

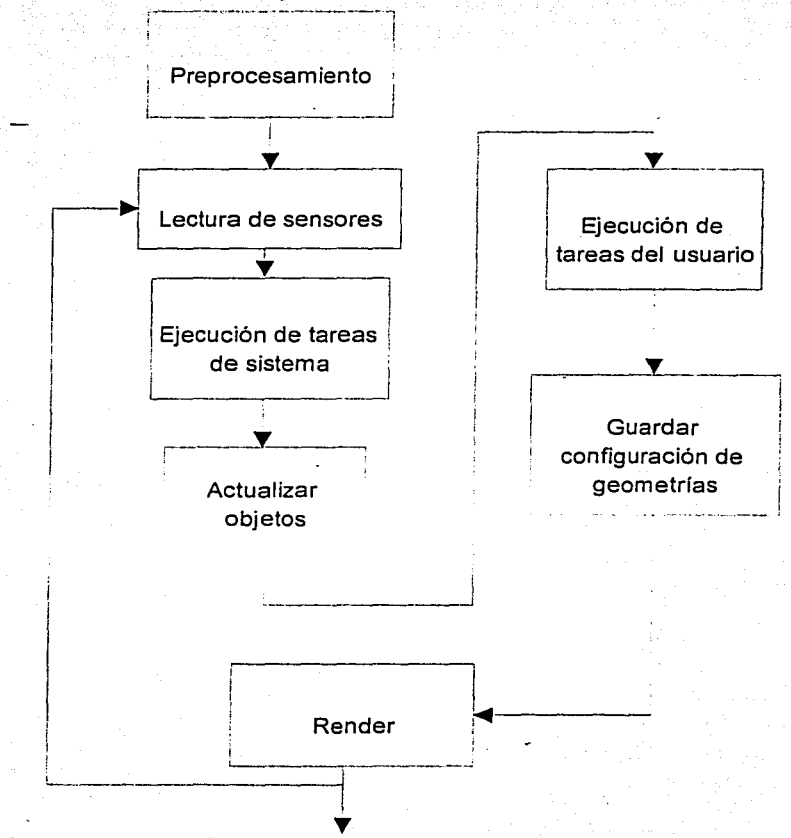


Figura II.1 Ciclo de simulación.

Como puede observarse, el cálculo de cada frame[8] involucra una gran cantidad de pasos; la participación que tiene el usuario en el ciclo de simulación no sólo está dada por la ejecución de las tareas indicadas por él mismo, sino que también interactúa haciendo uso de los sensores y es capaz de modificar el orden de los pasos del ciclo de simulación. Es importante conocer en qué orden se realizan los pasos de la simulación ya que su desconocimiento es fuente potencial de errores.

2.3 CONSTRUCCIÓN DE UNA ESCENA GRÁFICA

Una escena gráfica consiste en la organización de la escena dentro de un modelo jerárquico. En WTK una escena está compuesta por los siguientes elementos:

- Geometrías
- Luces
- Información de posición / orientación de geometrías y luces
- Niebla

2.3.1 MODELADO JERÁRQUICO

Un modelo es la representación de algunas características (no necesariamente todas) de una entidad concreta o abstracta. El propósito de un modelo es permitir que la gente visualice y entienda la estructura o el comportamiento de una entidad, y además provea el vehículo necesario para experimentar con dicha entidad.

En el área de gráficas por computadora, los modelos geométricos describen colecciones de componentes con ciertas propiedades geométricas. Entre estas propiedades encontramos:

- La disposición espacial y forma de los componentes (ejemplo: la geometría de la entidad), y otros atributos que afectan la apariencia de los componentes, tales como el color o el tamaño.
- La conectividad de los componentes (ejemplo: la estructura o topología de la entidad).
- Otros valores y propiedades asociados con los componentes, como características eléctricas o texto descriptivo.

Los modelos geométricos frecuentemente tienen una estructura jerárquica inducida por un proceso de construcción de abajo hacia arriba, de manera que los componentes son utilizados como bloques constructores para crear entidades de nivel superior, las cuales a su vez sirven como bloques constructores para entidades de niveles superiores. En la estructura jerárquica lo menos importante es el proceso de construcción; éste puede ser de abajo hacia arriba o de arriba hacia abajo; lo que importa es la jerarquía final.

Entonces, una jerarquía es creada para los siguientes propósitos:

[8] *Frame.* Una imagen sencilla en una secuencia de animación. Cuadro de animación.

Como puede observarse, el cálculo de cada frame[8] involucra una gran cantidad de pasos; la participación que tiene el usuario en el ciclo de simulación no sólo está dada por la ejecución de las tareas indicadas por él mismo, sino que también interactúa haciendo uso de los sensores y es capaz de modificar el orden de los pasos del ciclo de simulación. Es importante conocer en qué orden se realizan los pasos de la simulación ya que su desconocimiento es fuente potencial de errores.

2.3 CONSTRUCCIÓN DE UNA ESCENA GRÁFICA

Una escena gráfica consiste en la organización de la escena dentro de un modelo jerárquico. En WTK una escena está compuesta por los siguientes elementos:

- Geometrías
- Luces
- Información de posición / orientación de geometrías y luces
- Niebla

2.3.1 MODELADO JERÁRQUICO

Un modelo es la representación de algunas características (no necesariamente todas) de una entidad concreta o abstracta. El propósito de un modelo es permitir que la gente visualice y entienda la estructura o el comportamiento de una entidad, y además provea el vehículo necesario para experimentar con dicha entidad.

En el área de gráficas por computadora, los modelos geométricos describen colecciones de componentes con ciertas propiedades geométricas. Entre estas propiedades encontramos:

- La disposición espacial y forma de los componentes (ejemplo: la geometría de la entidad), y otros atributos que afectan la apariencia de los componentes, tales como el color o el tamaño.
- La conectividad de los componentes (ejemplo: la estructura o topología de la entidad).
- Otros valores y propiedades asociados con los componentes, como características eléctricas o texto descriptivo.

Los modelos geométricos frecuentemente tienen una estructura jerárquica inducida por un proceso de construcción de abajo hacia arriba, de manera que los componentes son utilizados como bloques constructores para crear entidades de nivel superior, las cuales a su vez sirven como bloques constructores para entidades de niveles superiores. En la estructura jerárquica lo menos importante es el proceso de construcción; éste puede ser de abajo hacia arriba o de arriba hacia abajo; lo que importa es la jerarquía final.

Entonces, una jerarquía es creada para los siguientes propósitos:

[8] *Frame.* Una imagen sencilla en una secuencia de animación. Cuadro de animación.

- Para construir objetos más complejos de manera modular, típicamente mediante la repetitiva invocación de bloques constructores que varían de forma y apariencia.
- Para mejorar la determinación de objetos visibles en la escena.
- Para acelerar el proceso de detección de colisiones.
- Para economizar memoria, pues basta con guardar sólo las referencias a objetos utilizados repetitivamente, en lugar de definir cada vez al mismo objeto.
- Permite la sencilla actualización de un objeto, ya que cualquier cambio en un bloque constructor se propaga automáticamente a los objetos de niveles superiores que utilizan ese bloque.

Los modelos geométricos son representados usualmente mediante diagramas de bloques, o mediante otras representaciones como los árboles jerárquicos. Este último método es el que se utilizará a lo largo de esta sección debido a que muestra de manera sencilla la topología de una entidad. Además de que es más fácil implantar esta representación en WTK, ya que también emplea árboles jerárquicos. Un árbol jerárquico en WTK es un diagrama de árbol, donde la raíz se encontraría en la parte superior y las hojas en la inferior y a cada elemento se le denomina nodo. La figura II.2 muestra una escena gráfica sencilla.

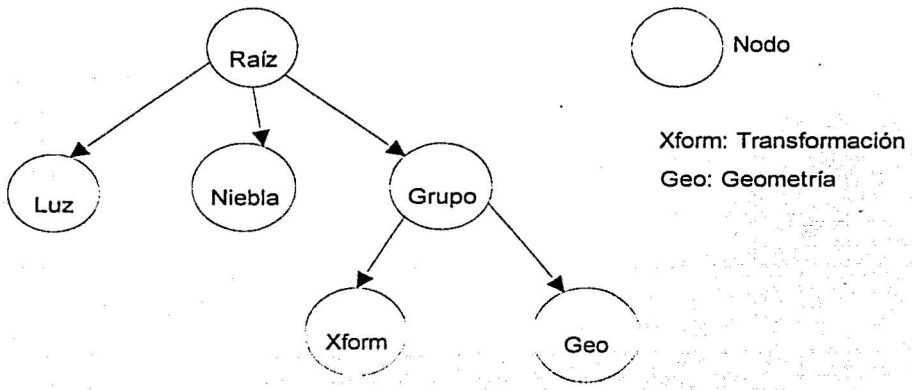


Figura II.2. Ejemplo de escena gráfica.

Las figuras II.3.a y II.3.b muestran cómo se modifican los nodos dentro de una escena gráfica y su resultado, respectivamente.

En la figura II.3.a tenemos una escena gráfica muy sencilla en la que se tiene un nodo raíz, un cuadrado y su descriptor de posición. Después de aplicar una rotación sobre el nodo Xform, que contiene la información de la posición del cuadrado, éste nodo se ve modificado y la transformación es propagada a través del árbol hacia los nodos de la derecha con la misma jerarquía o menor; de esta forma es como el nodo cuadrado se modifica. En la misma figura II.3.a la transformación es aplicada directamente al nodo del cuadrado y de esta forma ningún otro nodo es modificado. En ambos casos el cuadrado es afectado por la transformación, obteniendo la misma escena gráfica

(figura II.3.b), pero los nodos modificados no son los mismos.

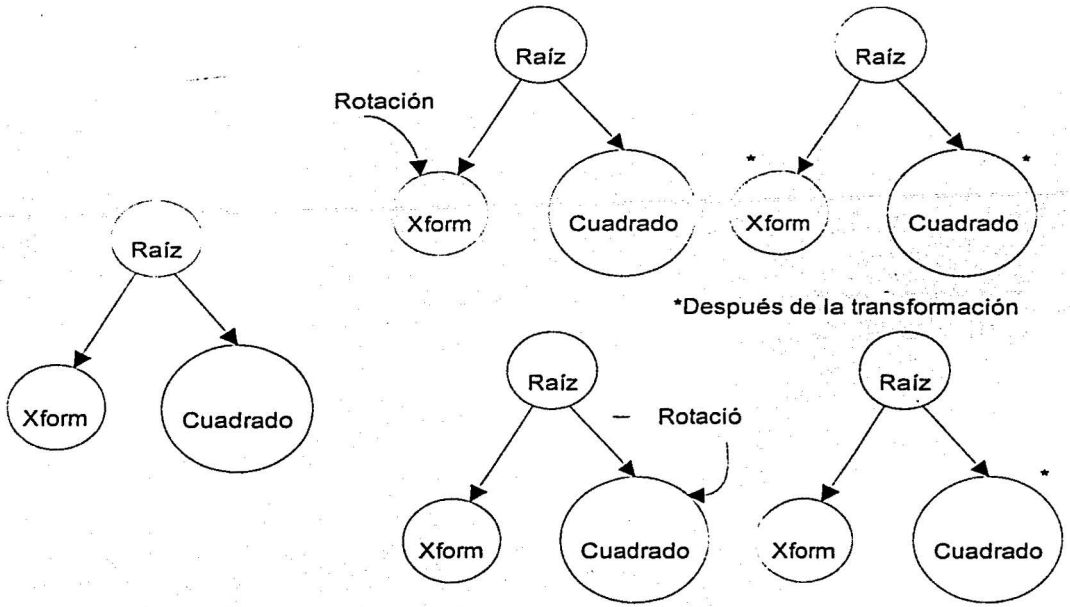


Figura II.3.a Estructura de árbol de la escena gráfica.

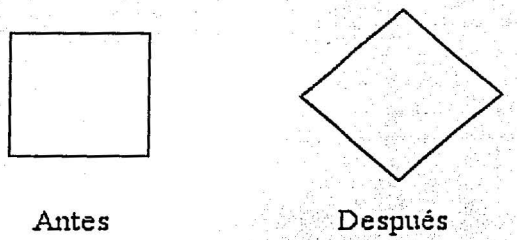


Figura II.3.b Escena gráfica.

2.4 TERMINOLOGÍA DE UNA ESCENA GRÁFICA

En la figura II.4 tenemos:

- Ancestro. El nodo A tiene un subárbol que incluye el nodo E; de esta forma el nodo A es ancestro del nodo E. Hay que hacer notar que el nodo J no es ancestro del nodo I.
- Nodo hijo. Es el descendiente directo de un nodo; por ejemplo: el nodo B y C son nodos

hijos del nodo A. El nodo J no es hijo del nodo A.

- Descendiente. Cualquier nodo que esté contenido en un subárbol de otro nodo se considera un descendiente de dicho nodo. En la figura II.4 el nodo F es uno de los descendientes del nodo A.

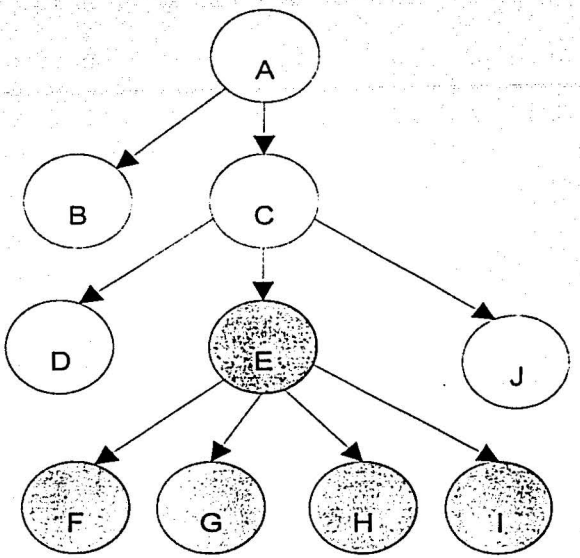


Figura II.4 Ejemplo de escena gráfica.

- Nodos hoja. Un nodo sin hijos. Los nodos B, D, F, G, H, I y J son nodos hoja.
- Nodo padre. Nodo que es ancestro directo de otro. El nodo A es padre del nodo C pero no del nodo E. Un nodo puede tener más de un padre.
- Predecesor. Debido a que el nodo B y C son procesados antes del nodo J, éstos son sus predecesores. Los predecesores de un nodo pueden afectar el proceso de render de dicho nodo.
- Nodo raíz. Cada escena gráfica sólo tiene un nodo raíz. En la figura II.4 el nodo raíz es el nodo A.
- Árbol de la escena gráfica. Son todos los nodos en una escena gráfica organizados de forma jerárquica.
- Subárbol. Un nodo junto con todos sus descendientes. Los nodos sombreados son un subárbol.
- Nodo sibling (hermanos). Hijos con el mismo padre; por ejemplo los nodos F, G, I y H.
- Orden transversal. Se refiere al orden en el que los nodos son procesados durante la simulación. Los nodos en la escena gráfica han sido etiquetados de forma transversal.

2.5 TIPOS DE NODOS

Un nodo es el elemento fundamental con el cual se construyen las escenas gráficas; el cual puede contener información de geometrías, luces, niebla, y datos de posición. Existen dos tipos de nodo:

- Contención
- Agrupamiento

2.5.1 CONTENCIÓN

Como su nombre lo indica, son nodos que contienen elementos en su interior. Los elementos que pueden contener son:

- Geometrías. Movible y no movible
- Luz. Movible y no movible
- Posición
- Niebla

De esta forma los nodos geométricos contienen información geométrica, los nodos de luz contienen información de iluminación, los nodos de niebla contienen información de la niebla y los nodos que contienen información acerca de la posición se llaman nodos de transformación. Los nodos de contención siempre son hojas del árbol jerárquico, es decir, no tienen hijos.

2.5.2 AGRUPAMIENTO

Los nodos de agrupamiento dan forma a la escena gráfica, es decir, dicen cómo están organizados los elementos en la escena; sin embargo, no contienen información sobre lo que existe dentro de ella. Los nodos de agrupamiento están divididos en:

- Nodo de grupo
- Nodo separador
- Nodo separador de transformación
- Nodo procedural
- Nodo especializado
- Nodo separador movible

2.5.2.1 Nodo de grupo

El nodo de grupo tiene como función servir como padre para uno o más nodos de un árbol; no tiene propiedades especiales en la escena gráfica. La figura II.5 muestra un ejemplo de nodo de grupo.

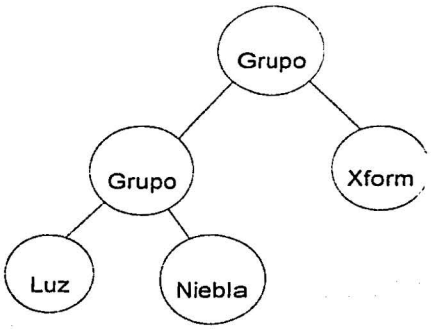


Figura II.5 Nodo de grupo.

2.5.2.2 Nodo separador

Debido a la forma en que funcionan las transformaciones dentro de la escena gráfica, en ocasiones puede ser necesario evitar que se propaguen las transformaciones fuera de cierta rama; éste es precisamente el trabajo de los nodos separadores: evitar que las transformaciones geométricas, los efectos de luz y niebla, se propaguen a través del árbol. En la figura II.6 se muestra un ejemplo haciendo uso de nodos separadores.

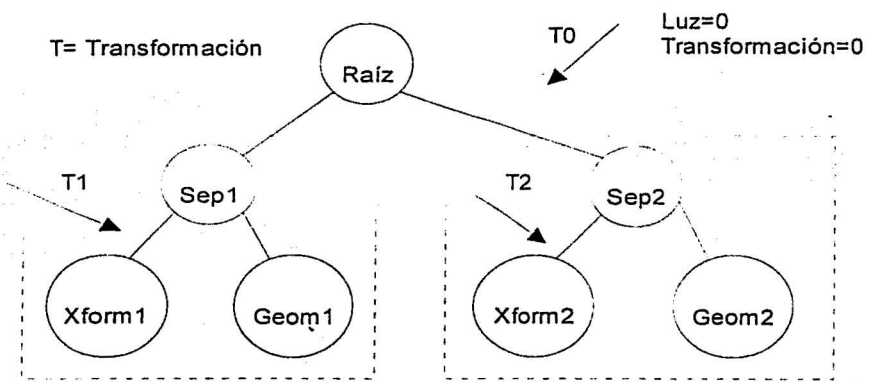


Figura II.6 Nodo separador.

En la figura II.6 tenemos dos nodos separadores (Sep1 y Sep2), al aplicar una transformación (T1) dentro de la rama que separa Sep1, dicha transformación no sale del subárbol formado por Sep1, Xform1, Geom1; es el mismo caso para Sep2 con la transformación T2.

2.5.2.3 Nodo separador de transformación

Este tipo de nodo tiene las mismas funciones que las de un nodo separador, a excepción de que este tipo de nodo sí permite la propagación de luz y niebla. La figura II.7 muestra un ejemplo de nodo separador de transformación.

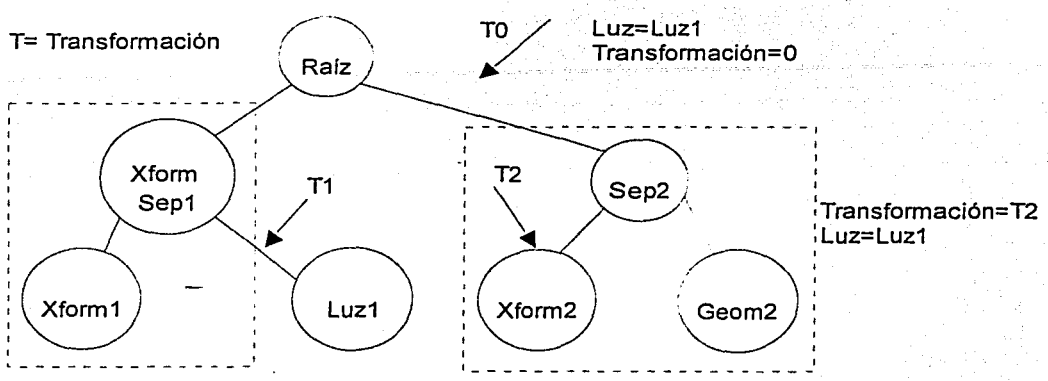


Figura II.7 Nodo separador de transformación.

En la figura II.7, el nodo separador de transformación XformSep1 impide que la transformación (T1) se propague hacia otros nodos; sin embargo, no impide que las propiedades de iluminación del nodo Luz1 se propaguen hacia el subárbol de la derecha.

2.5.2.4 Nodo procedural

Los nodos procedurales tienen información que permite activar un nodo hijo mientras otros nodos hijos son desactivados; fuera de esto, contienen la misma información que los nodos de agrupamiento organizacional. Los nodos procedurales se dividen en:

- Nodo de nivel de detalle
- Nodo switch (nodo selector)

Nodo de nivel de detalle

Para tener un mejor desempeño del sistema, este tipo de nodo permite seleccionar el nivel de detalle (LOD, level of detail) que se mostrará, dependiendo de la distancia que tengan los objetos del punto de vista. El nivel de detalle se divide en 3 categorías: distancia corta, distancia mediana y distancia lejana; es necesario indicar cuáles nodos se presentarán dependiendo del nivel de detalle. La figura II.8 muestra un ejemplo de rste tipo de nodo. El nodo LOD determina cuál grupo debe de ser mostrado.

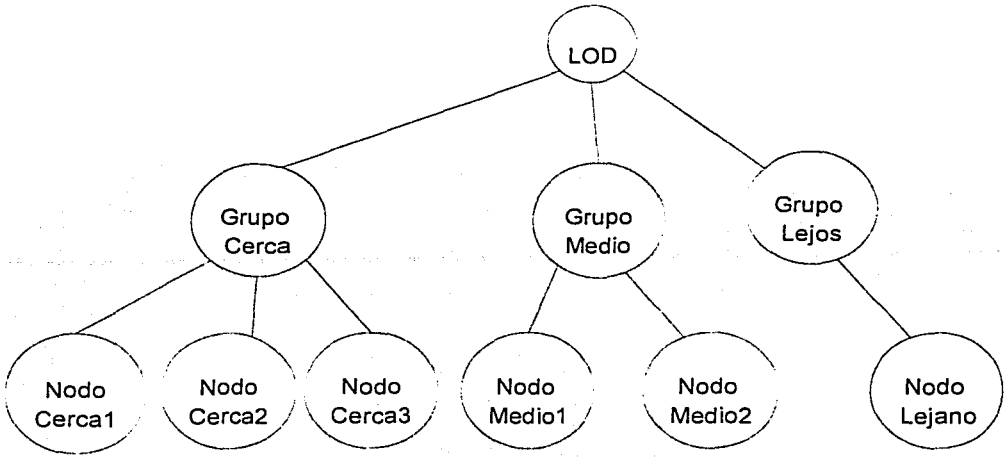


Figura II.8 Nodo de nivel de detalle (LOD).

Nodo switch

Este nodo permite seleccionar, de entre un conjunto de nodos hijos, cuál nodo se encontrará activo durante el proceso de recorrido del árbol. La figura II.9 muestra un ejemplo de nodo switch.

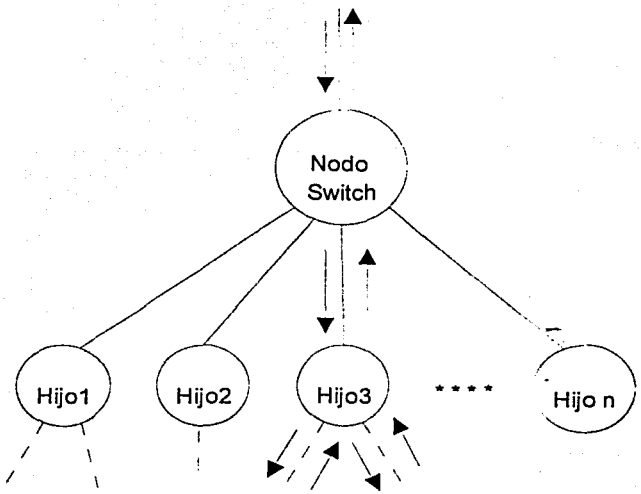


Figura II.9 Nodo switch.

2.5.2.5 Nodo especializado

Los nodos especializados están divididos de la siguiente manera:

- Nodo raíz
- Nodo inline (en-línea)
- Nodo ancla

El nodo raíz es único y actúa como el nodo superior dentro de una escena gráfica; es decir, no tiene nodo padre. El nodo inline y el nodo ancla hacen referencia a un elemento que se encuentra en una dirección de Internet; por ejemplo cargar un modelo geométrico localizado en un equipo remoto. Su diferencia consiste en que el nodo ancla realiza la carga una sola vez y el nodo inline se actualiza.

2.5.2.6 Nodo separador movible

Se le llama nodo separador movible al conjunto formado por un nodo separador, un nodo de transformación y un nodo de contenido que están estructurados como lo muestra la figura II.10. Estos nodos permiten crear escenas más rápido y mejor organizadas, ya que auto-contienen información de su posición y orientación.

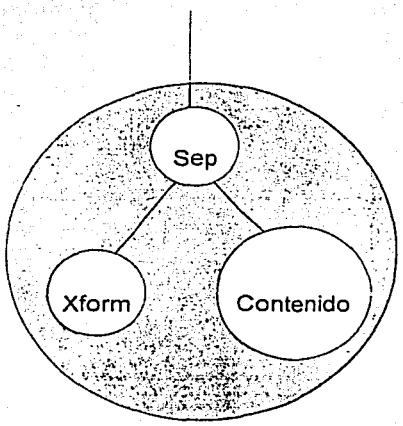


Figura II.10 Nodo separador movible.

2.6 RECORRIDO DE LA ESCENA GRÁFICA

El nodo raíz es el punto inicial de la escena gráfica y es el punto donde el proceso de dibujar la escena comienza; el proceso de recorrido es siempre el mismo partiendo del nodo raíz hacia los nodos inferiores y de izquierda a derecha; es decir, si durante el recorrido del árbol se encuentra un nodo que tenga más de un hijo, el recorrido continúa a través del primer hijo (el hijo que se

encuentre más a la izquierda) hasta finalizar su rama. La figura II.11 muestra un ejemplo de cómo se realiza un recorrido.

Durante el recorrido de la escena, cada vez que se encuentra un nodo, el sistema lo evalúa y procesa dependiendo de su tipo. Cada vez que se encuentra con un nodo geométrico, se dibujan los elementos geométricos con su posición y orientación actual, así como también, la luz y niebla actual. Cuando se encuentra con un nodo de luz, la luz se añade al conjunto de luces actual. Cuando se encuentra con un nodo de transformación, se modifica la orientación y posición actual. Cuando se encuentra un nodo de niebla se activa la niebla en ese momento. Cada nodo de luz, transformación y niebla afecta a las geometrías porque indican cómo deben de ser dibujadas. Las transformaciones que se apliquen a un nodo son el resultado de las transformaciones acumuladas durante el recorrido del árbol hasta ese punto.

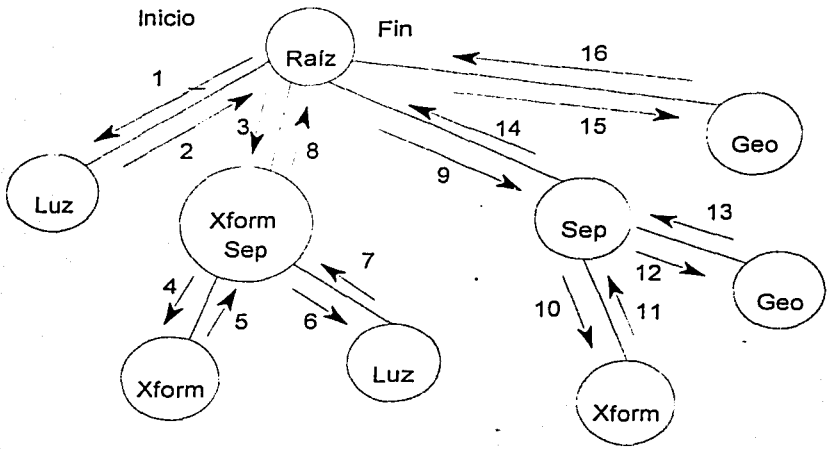


Figura II.11 Ejemplo de recorrido de una escena gráfica.

2.7 EJEMPLO DE CONSTRUCCIÓN DE UNA ESCENA GRÁFICA

La escena que se construirá consiste de una persona parada sobre la calle donde también hay un automóvil y una pared, y son iluminados por el sol. Es necesario que la persona pueda moverse a través de la escena, y también el automóvil.

Los elementos que forman a la escena son:

- Persona
- Automóvil
- Calle
- Pared
- Sol

De las especificaciones sabemos que la persona y el automóvil deben de poder modificar su posición, por lo que notamos que deben ser nodos movibles. La calle y pared son inmóviles, sin embargo, si pudiéramos mover la calle, deberán moverse la pared, la persona y el automóvil. Ahora que si la persona o el automóvil se mueven, la pared y la calle deben de permanecer inmóviles. El sol debe de iluminar todos los elementos. Tomando en cuenta las características antes mencionadas, podemos crear una escena gráfica de la siguiente forma:

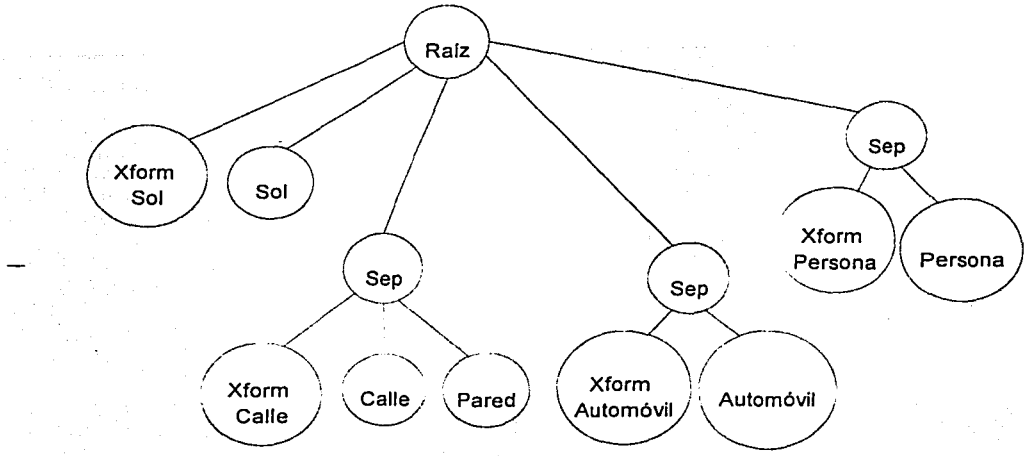


Figura II.12 Ejemplo de construcción de una escena gráfica.

2.8 REQUERIMIENTOS PARA UTILIZAR WTK

Los requerimientos de sistema para ejecutar aplicaciones de WTK en una computadora de escritorio con Win9x instalado son:

Como mínimo:

- Procesador 486/66 o superior
- 8 MB de RAM como mínimo
- DirectX 5.0 o superior
- Tarjeta de video que soporte DirectX con resolución de 640x480 píxeles con 16 bits de color

Opcional:

- Ratón (mouse)
- Palanca de mando (joystick)
- Algún dispositivo 3D (posición) o 6D (posición/orientación)
- Tarjeta aceleradora de video con al menos 4 MB de memoria

Software:

- Microsoft Visual C++ 4.2

2.9 CREACIÓN DE UNA APLICACIÓN DE WTK

Los pasos a seguir para crear una aplicación de WTK son los siguientes:

- Crear proyecto
- Añadir bibliotecas de WTK al proyecto
- Añadir código fuente

2.9.1 CREACIÓN DE UN PROYECTO EN VISUAL C++

En el menú de archivo de Visual C++ se encuentra la opción de crear un nuevo proyecto (ver figura II.13).

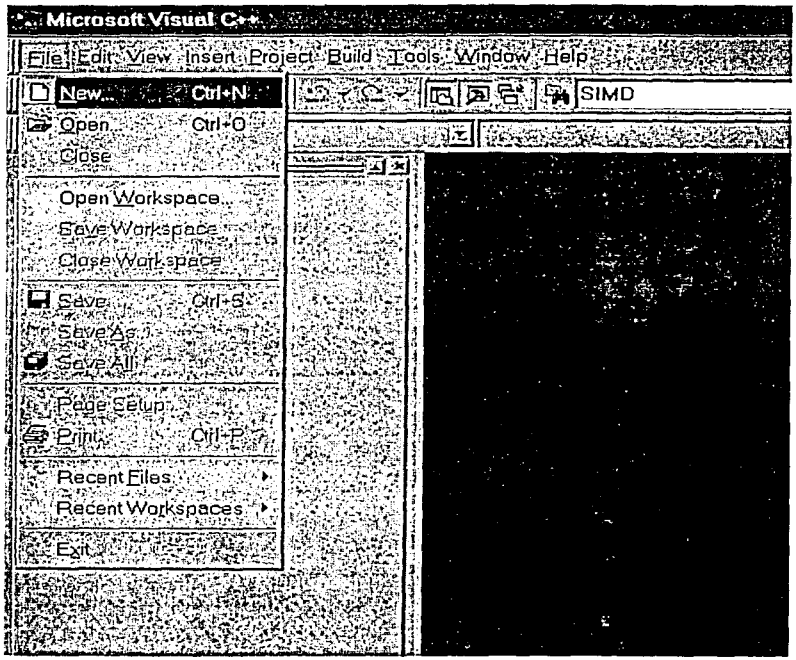


Figura II.13 Cuadro de diálogo para crear un nuevo proyecto.

Dentro de Visual C++ existen diversas formas de crear un proyecto de programación; la opción que utilizaremos es modo consola de 32 bits como se muestra en la figura II.14; sin embargo, es posible realizar aplicaciones de WTK utilizando Win32 application y MFC AppWizard.

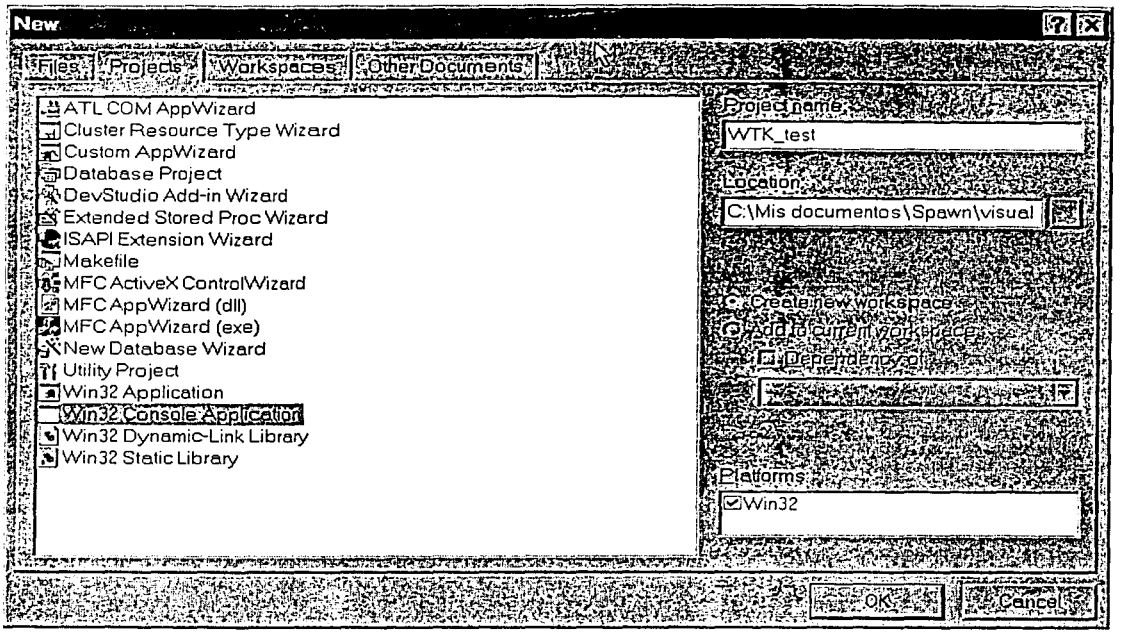


Figura II.14 Crear un proyecto Win32 Console Application.

En este punto es necesario especificar el nombre del proyecto, así como también, el directorio donde será almacenado; en este ejemplo el proyecto tiene el nombre de WTK_test. A continuación seleccione la modalidad de proyecto vacío (ver figura II.15). Un proyecto vacío, como su nombre lo indica, no agrega archivos al proyecto, por lo tanto, el usuario debe incorporarlos explícitamente.

2.9.2 CÓMO AÑADIR LAS BIBLIOTECAS DE WTK DENTRO DEL PROYECTO

Al momento de crear un proyecto win32 de tipo consola, se añaden por defecto las siguientes bibliotecas:

- kernel32.lib
- user32.lib
- gdi32.lib
- winspool.lib
- comdlg32.lib
- advapi32.lib
- shell32.lib
- ole32.lib
- oleaut32.lib
- uuid.lib
- odbc32.lib
- odbccp32.lib

Una aplicación de WTK necesita añadir la biblioteca propia del API: wtk.lib; esta biblioteca se encuentra ubicada en el directorio lib del directorio de instalación de WTK. La figura II.16 muestra la estructura de directorios que se crea al instalar WTK.

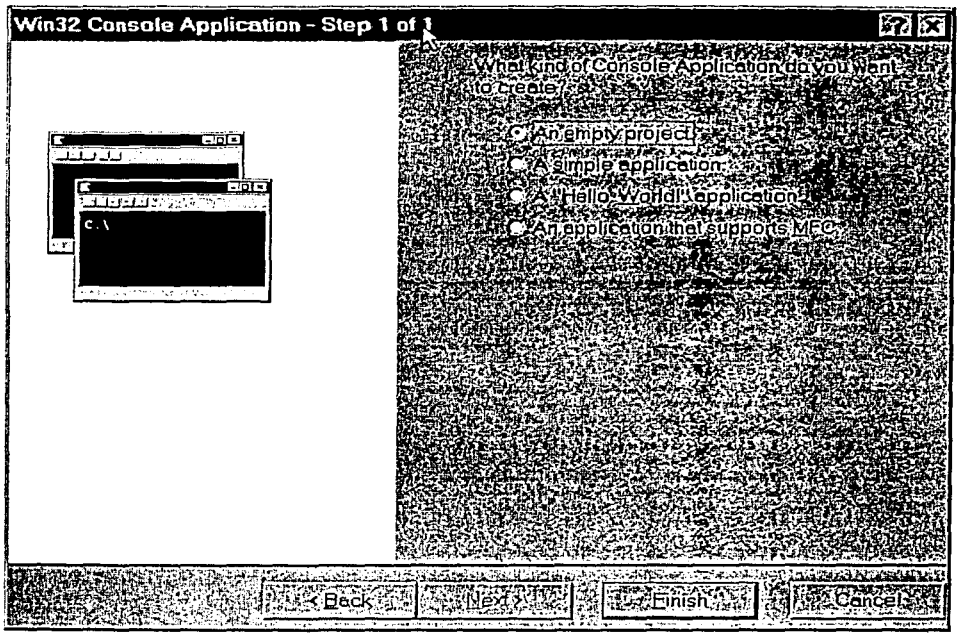


Figura II.15 Crear un proyecto vacío.

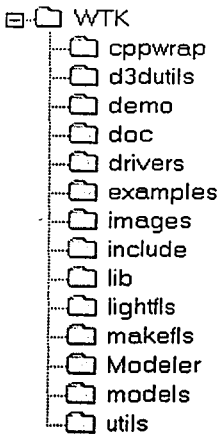


Figura II.16 Estructura de directorios de WTK.

WTK necesita para el despliegue gráfico usar el API de DirectX, por lo que es necesario especificar el uso de estas bibliotecas. Las bibliotecas necesarias son:

- ddraw.lib
- dsound.lib
- dxguid.lib

Durante el proceso de instalación de WTK, las bibliotecas de DirectX fueron copiadas en el mismo directorio lib (ver figura II.16). Si lo desea puede utilizar estas bibliotecas, o bien, utilizar las bibliotecas propias del sistema operativo, ya que seguramente serán versiones más recientes.

Otras bibliotecas que son necesarias para el funcionamiento del API de WTK son:

- wsock32.lib
- winmm.lib

Estas bibliotecas se copian al sistema durante la instalación de Microsoft Visual C++, por lo que supuestamente el sistema cuenta con ellas.

Durante el proceso de ligado de los archivos objeto que forman el programa, WTK causa conflictos con otras bibliotecas del sistema y no se generará un archivo ejecutable; para evitar esto, es necesario especificar que se desea omitir a la biblioteca: libcd.lib. De esta forma se evitan conflictos con el sistema y es posible generar archivos ejecutables. La figura II.17 muestra dónde se indica la exclusión de bibliotecas.

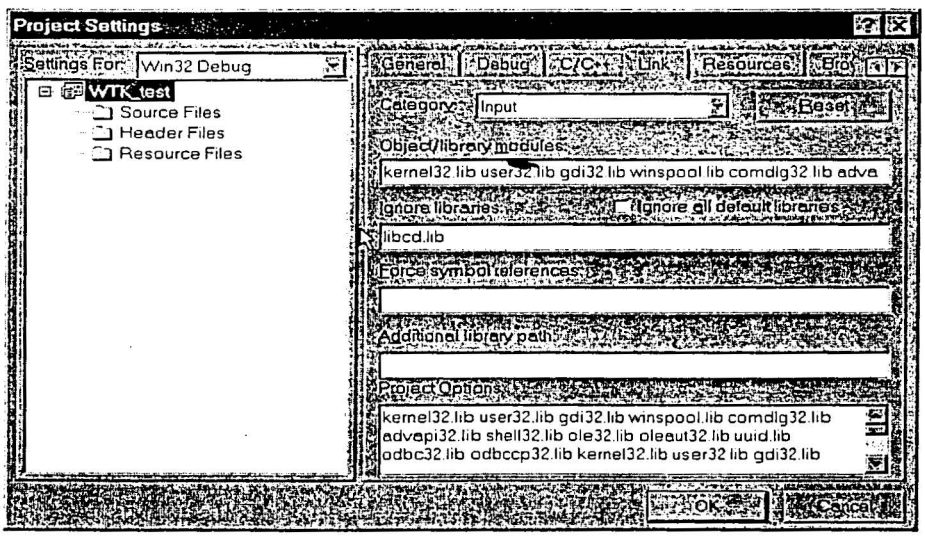


Figura II.17 Exclusión de la biblioteca libcd.lib.

Para añadir las bibliotecas antes indicadas dentro del proyecto de Visual C++, deben de añadirse en Proyect/Settings como lo muestra la figura II.18.

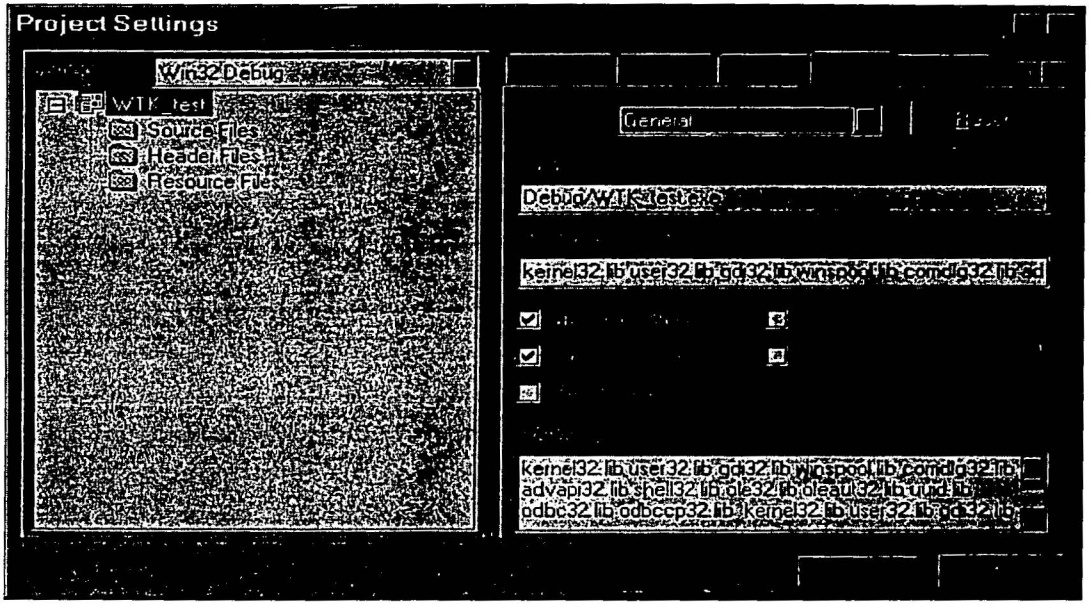


Figura II.18 Bibliotecas para un proyecto de WTK.

Resumen bibliotecas de WTK

El siguiente es un resumen de las bibliotecas mínimas indispensables que una aplicación de WTK debe tener:

- wsock32.lib
- winmm.lib
- kernel32.lib
- user32.lib
- gdi32.lib
- winspool.lib
- comdlg32.lib
- advapi32.lib
- shell32.lib
- ole32.lib
- oleaut32.lib
- uuid.lib
- odbc32.lib
- odbccp32.lib
- ddraw.lib
- dsound.lib
- dxguid.lib
- wtk.lib

2.9.3 EJEMPLO DE PROGRAMA

El siguiente ejemplo muestra un cubo que rota sobre sus ejes respondiendo a las flechas del teclado. La figura II.19 muestra la ejecución del programa.

```
/************************************************
 * Test.c                                       *
 * Usage: Use arrows to rotate the cube. Q to quit.  *
 ************************************************/

#include "wt.h"
#include "stdio.h"

void KeyboardActions(void);

WTnode *cube;

void main(int argc, char *argv[]){
    WTnode *root;
    WTgeometry *geo;
    WTsensor *sensor; /*the Mouse*/
    WTviewpoint *view; /*the Viewpoint*/

    WTuniverse_new(WTDISPLAY_DEFAULT,WTWINDOW_DEFAULT);
    root = WTuniverse_getrootnodes();

    /*Create cube*/
    geo=WTgeometry_newblock(5.0f,5.0,5.0,FALSE);
    cube = WTmovgeometrynode_new(root,geo);

    sensor = WTmouse_new(); /*Active mouse*/
    view = WTuniverse_getviewpoints();
    WTviewpoint_addsensor(view, sensor);

    WTkeyboard_open(); /*Activate keyboard*/
    WTuniverse_setactions(KeyboardActions);
    WTmessage("\nKeyboard enabled");
    WTwindow_zoomviewpoint(WTuniverse_getwindows());

    WTuniverse_ready();
    WTuniverse_go(); /*Starts simulation*/
    WTuniverse_delete(); /*All done*/
}

void KeyboardActions(void){ /*keyboard manager*/
    short key;

    key=WTkeyboard_getlastkey();
    switch(key){
        case 'Q':
                    WTmessage("\nBye!...");
                    WTuniverse_setrendering(WTRENDER_WIREFRAME);
                    WTuniverse_stop();
```

```
                    break;
            case WTKEY_UPARROW:
                    WTmovnode_axisrotation(cube,X,(float)0.1);
                    break;
            case WTKEY_DOWNARROW:
                    WTmovnode_axisrotation(cube, X,(float)-0.1);
                    break;
            case WTKEY_LEFTARROW:
                    WTmovnode_axisrotation(cube, Y,(float) 0.1);
                    break;
            case WTKEY_RIGHTARROW:
                    WTmovnode_axisrotation(cube, Y,(float)-0.1);
                    break;
    }
};
```

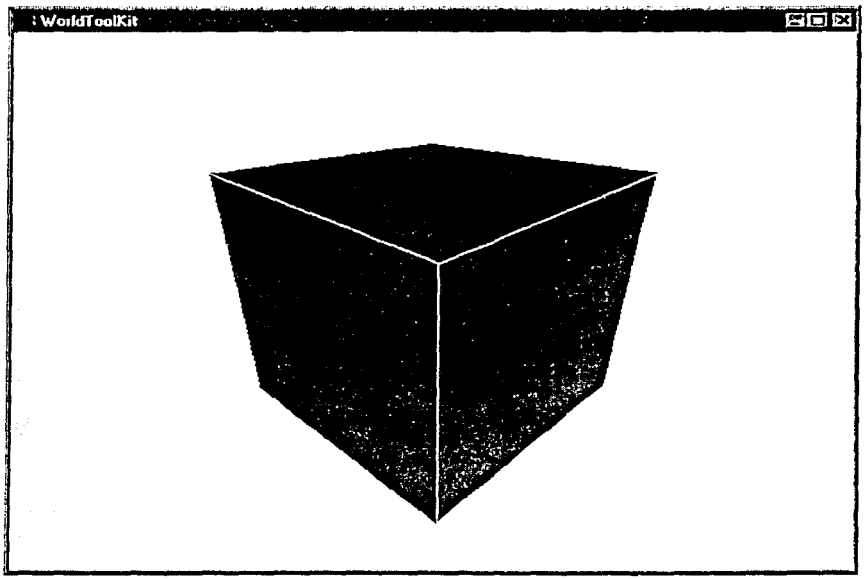


Figura II.19 Salida de programa.

CAPÍTULO III
INTERFAZ 3D

3.1 INTRODUCCIÓN

En la actualidad, las interfaces del hombre con las computadoras, para la mayor parte de los sistemas, utilizan gráficos para interactuar con el usuario, indiferentemente del tipo de aplicación. Por lo regular, los sistemas generales del presente consisten en ventanas, menús descendentes y de entrada momentánea, íconos y dispositivos de presionar, como un mouse o un lápiz óptico para colocar el cursor en la pantalla. Las interfaces gráficas para usuarios comunes incluyen X Windows, Windows, Macintosh, OpenLook, Motif, Gnome y KDE; estas interfaces se utilizan en una gran variedad de aplicaciones.

En los paquetes de gráficas, se diseñan diálogos interactivos especializados para aplicaciones individuales como ingeniería de diseño, diseño arquitectónico, visualización de datos, dibujo mecánico, gráficas empresariales y programas de dibujo. Las entradas en los sistemas de gráficas pueden provenir de muchos dispositivos de hardware diferentes, con más de un dispositivo que ofrece la misma clase general de datos de entrada.

En general, una interfaz de usuario busca tener dentro de sus características, los siguientes conceptos:

- Niveles de capacidad múltiple. Este concepto especifica que es posible realizar una misma acción de dos o más formas diferentes, debido a que existen usuarios con diferentes niveles de capacidad.
- Rapidez de aprendizaje. La rapidez de aprendizaje tiene que ver con el tiempo que le toma a un nuevo usuario aprender a utilizar la interfaz de forma eficiente.
- Consistencia. La consistencia especifica que se deben preservar los significados de los elementos durante toda la interfaz, así como también, su presentación.
- Mínimo de memorización. El mínimo de memorización determina que se debe de planear la interfaz de forma que no sea necesario memorizar una gran cantidad de pasos para realizar tareas comunes o sencillas.
- Respaldo y manejo de errores. El respaldo y manejo de errores indica que debe ser posible cancelar una acción indicada así como también deshacerla, y en los casos que sea imposible deshacer la operación se debe generar un advertencia.
- Retroalimentación. La retroalimentación quiere decir que la interfaz no sólo recibe información del usuario sino que muestra el progreso de las acciones que realiza en cada paso; esto es de especial importancia cuando el tiempo de respuesta es alto.

De manera práctica, los elementos que determinan las características de una interfaz de usuario incluyen cubrir necesidades específicas del problema y los recursos con los que se cuenta.

3.2 ROBOT COMMAND CENTER

La interfaz que se ha desarrollado permite controlar robots virtuales y reales haciendo uso de periféricos dedicados a ello o instrucciones específicas proporcionadas por el usuario; de esta forma,

la interfaz puede utilizarse de dos maneras: como medio de visualización de resultados y como simulador de robots; debido a estas características se le ha llamado Robot Command Center[9] (Roc2).

Roc2 permite interactuar con robots reales y virtuales de forma indistinta, cuenta con diferentes maneras de especificar instrucciones a un robot y hace uso de ventanas con gráficas en 3D para visualizar los resultados de las operaciones.

La siguiente es una lista de características que presenta Roc2:

- Uso de 3 ventanas para visualizar resultados tridimensionales
- Navegación libre del ambiente de simulación
- Consola de comandos
- Control sobre de uno o más robots
- Posibilidad de usar distintos modelos de robot
- Posibilidad de cambiar el ambiente de simulación
- Compatibilidad con el control de juegos del Super Nintendo (SNES joypad)
- Entradas de usuario local y remota
- Compatibilidad con Robel[10]

Funcionamiento

El diagrama de bloques de la figura III.1 muestra la forma en que opera internamente Roc2.

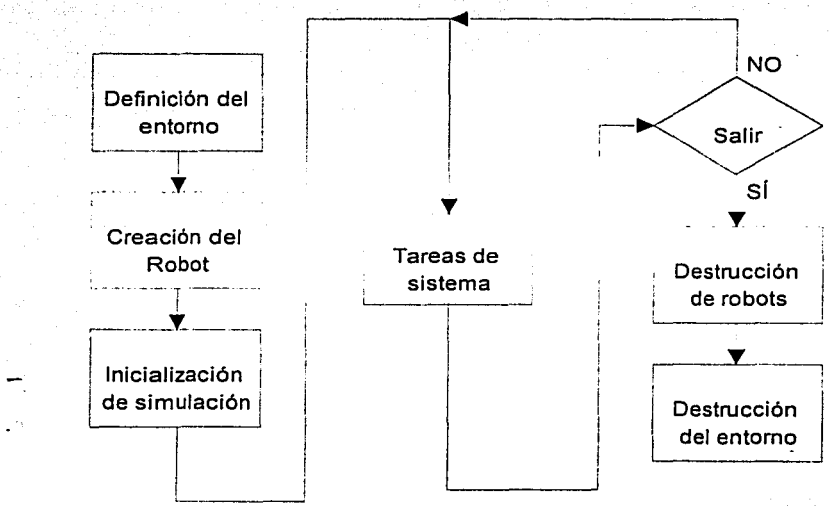


Figura III.1 Diagramas de bloques que muestra la estructura interna de Roc2.

[9] *Robot Command Center*. Centro de comandos para robots.
[10] *Robel*. Lenguaje para comportamientos de robots (Robot behavior language). Para mayor información acerca del lenguaje Robel consulte el capítulo VI.

Definición del entorno

Un mundo virtual es definido por medio de la escena gráfica que lo forma; es decir, por un conjunto de geometrías y luces ordenadas de forma jerárquica que definen la distribución y características del entorno. Roc2 es capaz de utilizar diferentes entornos:

- Definición a través de mapas de descripción
- Definición mediante archivos de modelos
- Entorno vacío

Las ventajas y desventajas de usar cada una de las técnicas anteriores se describen a continuación.

Definición a través de un mapa de descripción. Este formato permite especificar las dimensiones del mundo que se desea cargar así como la ubicación de los obstáculos. En realidad es un archivo de texto con extensión wrl que puede ser creado en cualquier editor; las reglas para poder crear un archivo de descripción aparecen en el Apéndice C. Manual de usuario. La estructura de la escena debe de ser planeada por el desarrollador de forma indirecta y luego implantada en el sistema. Usar esta técnica tiene como ventaja que el mundo será representado haciendo uso de la menor cantidad de geometrías por lo que tendrá buen desempeño; la desventaja es que la construcción de mundos complicados no es sencilla.

Definición mediante archivos de modelos. Esta técnica consiste en crear el mundo virtual utilizando un modelador de mundos tridimensionales; los formatos soportados y el tipo de aplicación con el que pueden ser creados son:

- 3DS (3D Studio)
- BFF (Sense8)
- DXF (AutoCAD)
- GEO (VideoScape)
- NFF (Sense8)
- OBJ (Wavefront)
- SLP (ProEngineer "RENDER")

La ventaja de utilizar un modelador 3D para crear un mundo virtual consiste en la comodidad que proporciona la herramienta, con lo cual es posible crear mundos 3D complicados de forma sencilla. La desventaja consiste en que cargar un modelo disminuye significativamente el desempeño del sistema.

Entorno vacío. El entorno vacío consiste en una escena gráfica vacía, es decir, un mundo 3D sin elementos. La ventaja principal es que se obtiene un desempeño inigualable del sistema. La desventaja es que carece de utilidad, pues es necesario tener elementos con los cuales interactuar.

Las siguientes figuras permiten ver un entorno virtual en tres modos de render distinto. Véanse las figuras III.2, III.3, III.4.

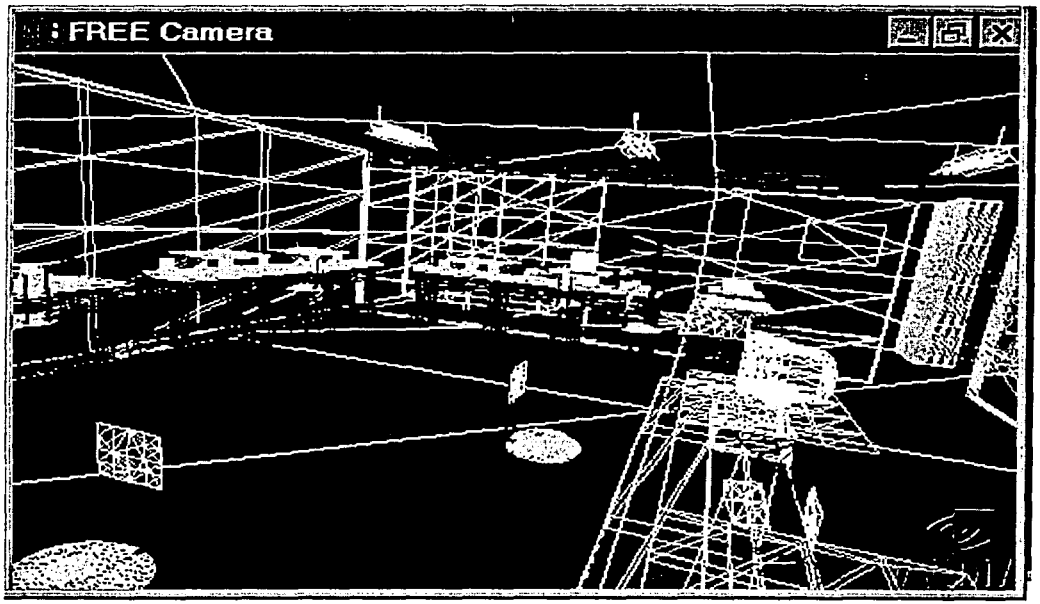


Figura III.2 Modelo de alambres del mundo virtual.

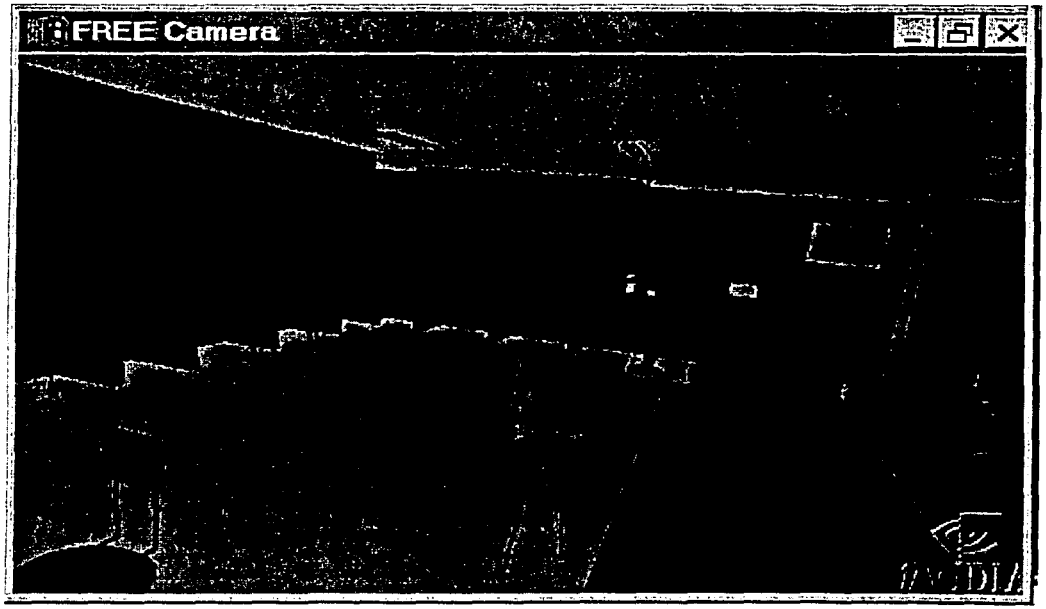


Figura III.3 Modelo en colores sólidos del mundo virtual.

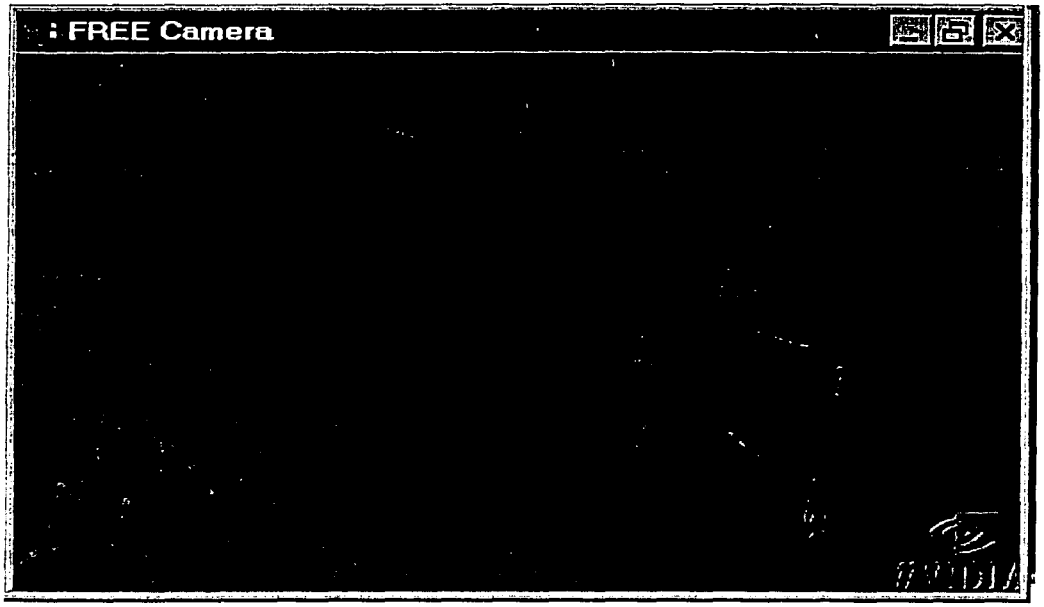


Figura III.4 Modelo con texturas del mundo virtual.

3.3 CREACIÓN DE ROBOTS

En este punto se especifican cuáles serán los robots que pertenecerán al mundo virtual. Existen dos formas distintas de crear un robot móvil:

- Modelo 3D
- Robot Roc2

Modelo 3D

Usando las bibliotecas de WTK es posible cargar un modelo tridimensional para dar forma a un robot. El modelo tridimensional debe ser creado en un modelador 3D, por lo que crear diseños complicados es sencillo. El formato de archivo utilizado es 3ds, propio del paquete 3D Studio.

Robot de Roc2

Roc2 tiene especificada una rutina para crear un robot móvil; este robot es una representación de un robot real utilizando primitivas gráficas propias del API de WTK. Las características de este robot móvil son:

- cuatro sensores de contacto
- cuatro sensores de infrarrojo
- cuatro sensores de ultrasonido
- cuatro sensores reflectivos
- cambio de apariencia
- modificación de su tamaño

3.3.1 CONSTRUCCIÓN GEOMÉTRICA

La construcción del robot móvil sigue las mismas reglas de la construcción de una escena virtual; es decir, está compuesto por primitivas gráficas y luces, organizándolas en una estructura jerárquica. Las operaciones de movimiento propias de un robot móvil se reducen a movimientos de rotación y traslación; la construcción de la escena obedece a estas necesidades: cambiar la posición y orientación del modelo completo en cualquier momento, además de que se debe considerar un movimiento rotacional axial a las ruedas traseras para simular su movimiento. La figura III.5 muestra el modelo del robot móvil dentro de la interfaz.

La figura III.6 muestra la escena gráfica del robot móvil, la figura III.7 muestra la nomenclatura de la escena gráfica y la figura III.8 muestra las dimensiones del robot móvil.

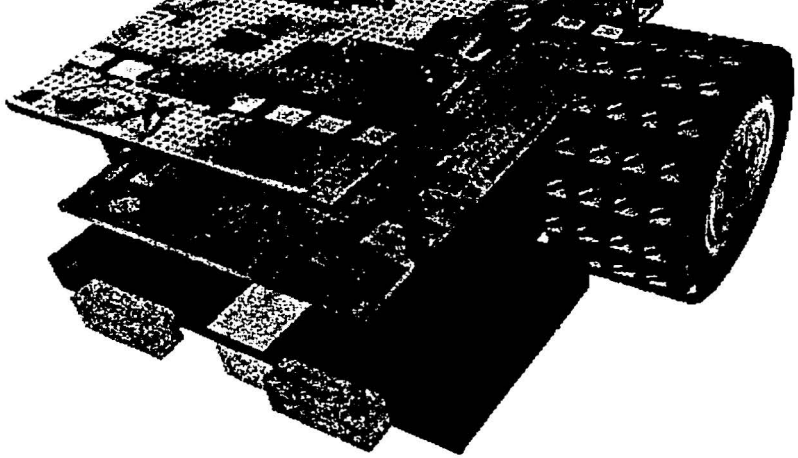


Figura III.5 Robot virtual estándar del Roc2.

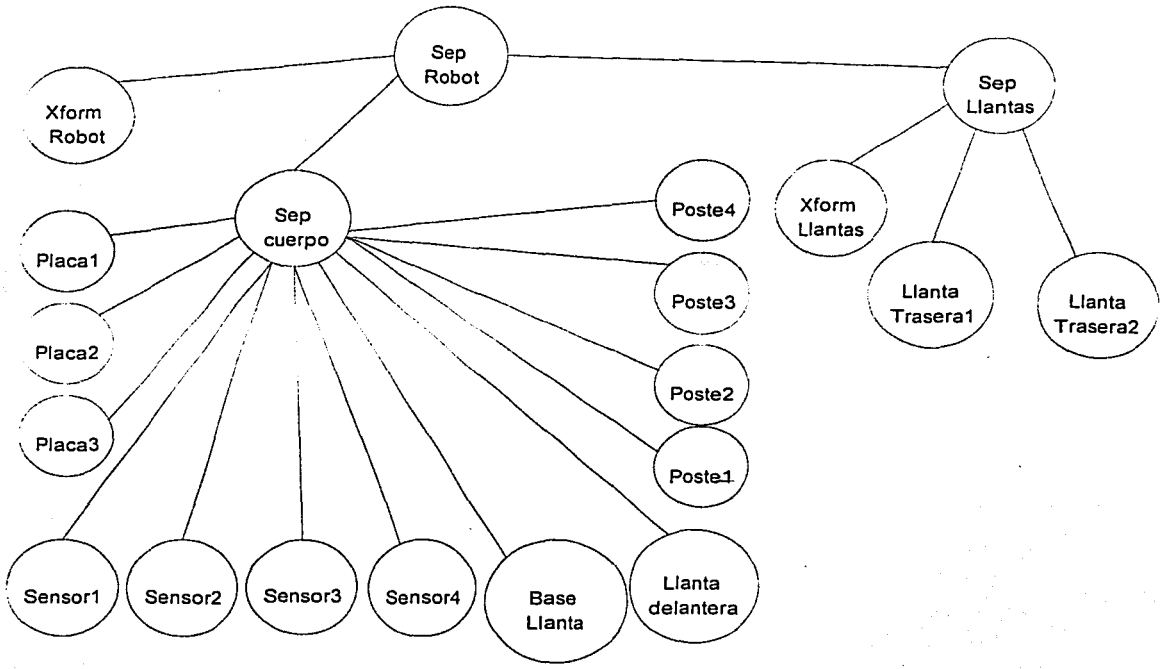


Figura III.6 Escena gráfica del robot móvil.

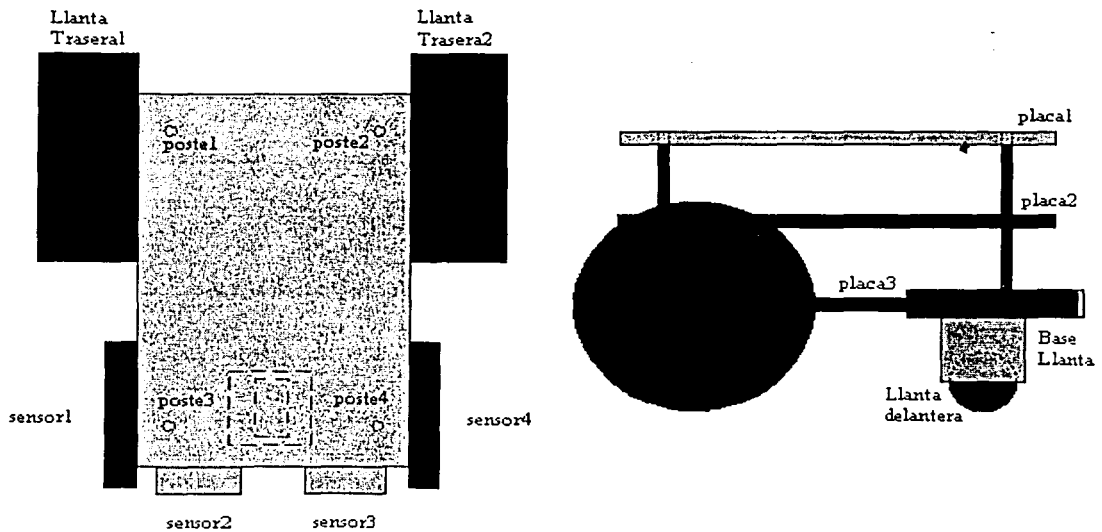


Figura III.7 Nomenclatura de construcción del robot.

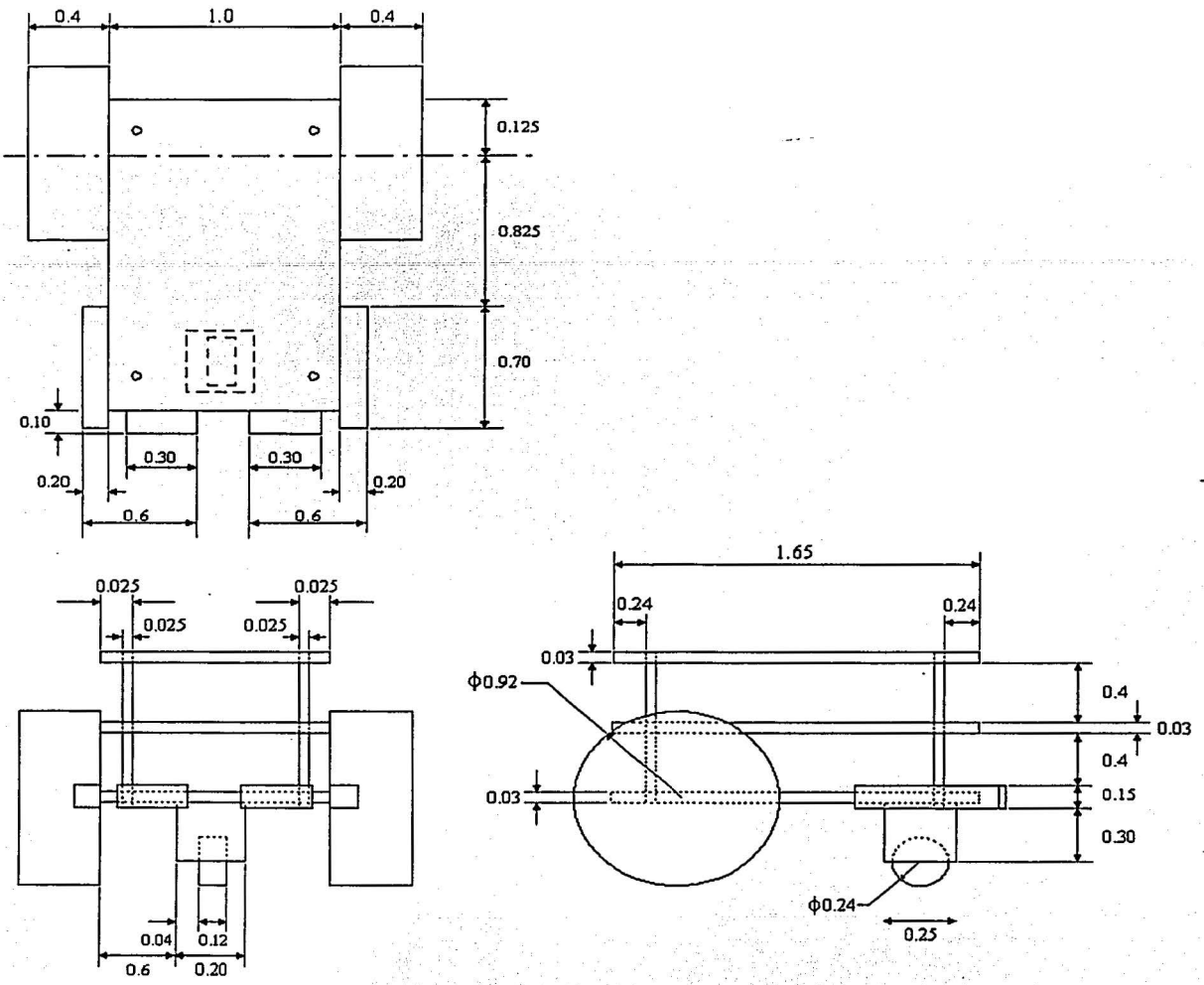


Figura III.8 Dimensiones, en decímetros, del robot móvil.

3.3.2 CONTENIDO LÓGICO

Dentro de cada robot existe un conjunto de datos que permiten controlarlo y conocer el estado en el que se encuentra; dichos datos tienen una estructura lógica interna que consta de las siguientes partes:

- Nombre. El nombre es el identificador del robot. El Roc2 permite tener más de un robot dentro de la simulación, por lo tanto, es necesario un método que permita identificarlo del resto de los robots.

- Máquina virtual. Para que cada robot pueda interpretar las instrucciones pertenecientes al lenguaje Robel necesita contar con la máquina virtual de Robel (RVM, Robel virtual machine). De esta manera, cada robot utilizado en Roc2 será capaz de interpretar instrucciones que le permitan realizar las acciones indicadas por el usuario.

- Registros de datos. Estructura de datos de tipo flotante, entero y cadena para uso interno del robot.

- Posición y orientación relativa. Localización del robot haciendo uso de un punto de referencia especificado por el usuario.

- Posición y orientación absoluta. Localización del robot haciendo uso de los ejes del sistema.

- Banderas de estado del robot. Datos que nos permiten conocer en qué estado lógico se encuentra el robot. Las banderas con las que se cuenta son:

    - Ocupado
    - En movimiento
    - En rotación
    - Destino alcanzado
    - Rotación finalizada
    - Colisión

- Banderas de estado de los sensores. Roc2 permite tener distintos tipos de sensores en un mismo robot y debido a ello, todo valor obtenido por una lectura de sensor se registra en una bandera de estado.

- Sensores. Dentro de cada robot se indican el tipo, posición y número de sensores que tendrá el robot móvil para interactuar dentro del sistema.

3.4 SENSORES

Los sensores disponibles en los robots del Roc2 son los siguientes. Recuerde que estos sensores están integrados en la estructura lógica del robot.

- Contacto
- Reflectivo
- Infrarrojo
- Sonar

3.4.1 SENSORES DE CONTACTO

La función de un sensor de contacto es detectar cuándo un elemento toca al sensor. La acción de tocar nos indica de forma implícita que el sensor debe ser un objeto tridimensional, por lo tanto,

los sensores de los robots están representados por prismas. Si un sensor entra en contacto con algún otro objeto, su estado lógico cambia a uno; si no, adquieren el estado lógico de cero.

Crear un sensor de contacto en un mundo 3D se reduce a determinar cuándo un elemento 3D, por ejemplo un prisma, está siendo tocado por otro, es decir, se están intersecando los cuerpos. Existen diversos métodos computacionales relacionados al cálculo de intersección de cuerpos, cada uno dedicado a casos muy especiales; la gran mayoría demandan una gran cantidad de recursos, pero en el caso particular de la lectura de sensores se han tomado algunas consideraciones que permiten realizar algunas simplificaciones.

Consideraciones:

- Los objetos son prismas de 6 caras
- Las caras son convexas
- Los ángulos de las caras de los prismas son rectos
- La normal de una cara está alineada a un eje coordenado del sistema

La figura III.9 muestra un par de prismas que cumplen con las consideraciones antes descritas.

Las consideraciones nos permiten garantizar que todas las caras del cuerpo pueden ser proyectadas sobre planos cuyas normales son paralelas a los ejes del sistema; cada proyección mostrará 2 caras del prisma, una cara ocultando a la otra ya que son paralelas.

Si un cuerpo se interseca con otro, la intersección debe ser visible en todas las proyecciones. El problema de intersección de cuerpos se ha reducido a un problema de intersección de figuras planas (áreas). La figura III.10 muestra dos prismas y sus proyecciones.

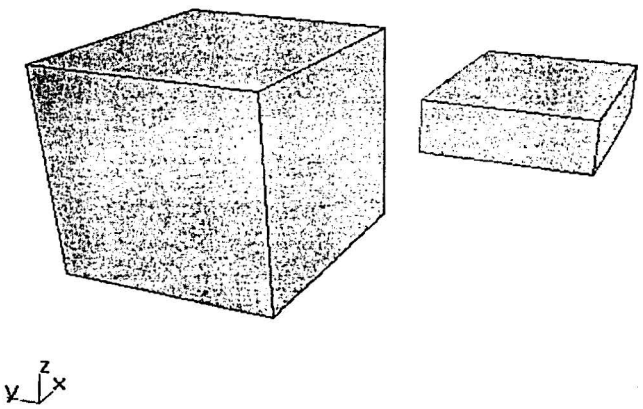


Figura III.9 Prismas utilizados en colisiones .

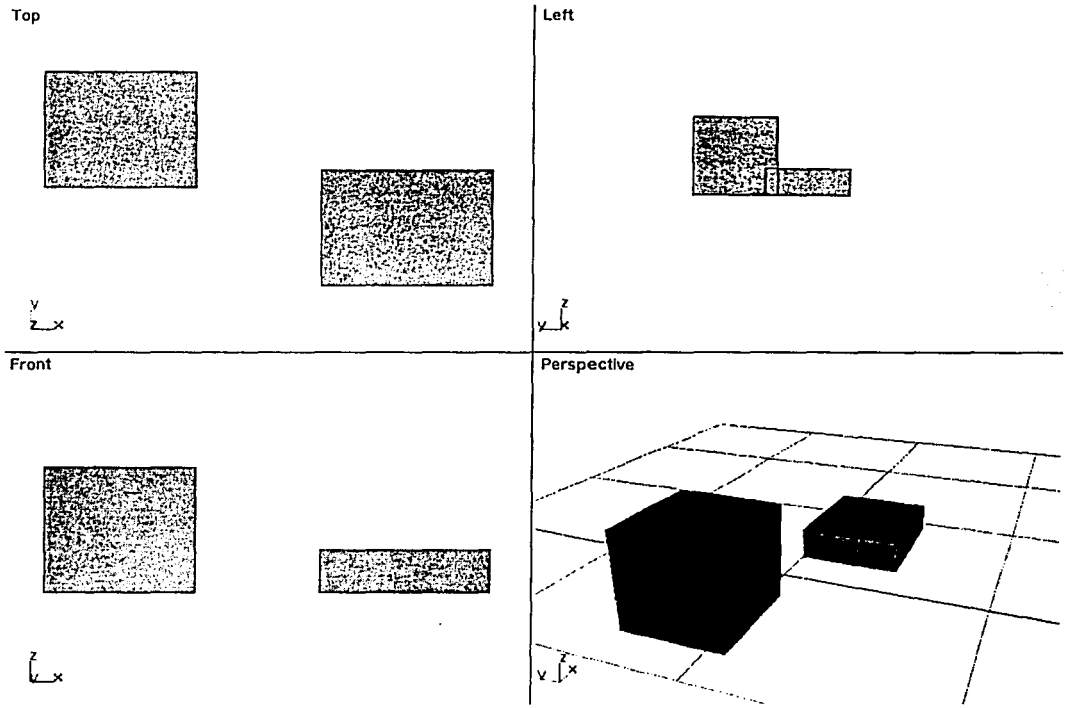


Figura III.10 Prismas y sus proyecciones.

Las proyecciones de un prisma (P) sobre un plano, como el mostrado en la figura III.10, forman figuras planas de cuatro vértices (v); además, las componentes de cada vértice en el espacio 3D son de la forma (v*x*,v*y*,v*z*). Para determinar cuál es la proyección de un vértice (v) del prisma sobre un plano paralelo, ortogonal a un eje coordenado, se hace lo siguiente (ver tabla III.1):

Plano ortogonal	*Proyección (u,w)*
X	(v*y*, v*z*)
Y	(v*x*, v*z*)
Z	(v*x*, v*y*)

Tabla III.1 Proyección de un vértice sobre los planos paralelos.

Una vez obtenidas las figuras planas que se forman al proyectar dos prismas sobre un plano, tenemos que determinar si ambas figuras se intersecan. Los datos con los que se cuenta hasta el momento son:

Vértices de la figura 1	*Vértices de la figura 2*
V11	V21
V12	V22
V13	V23
V14	V24

Tabla III.2 Datos de los vértices.

donde Vab indica vértice *b* de la figua *a* (ver tabla III.2).

Así, cada vértice Vij tiene dos componentes; los ejes pertenecientes a la proyección serán *U* y *W*. Para cada una de las figuras es necesario identificar cuáles serán las componentes Umínima, Umáxima, Wmínima y Wmáxima, como lo muestra la figura III.11.

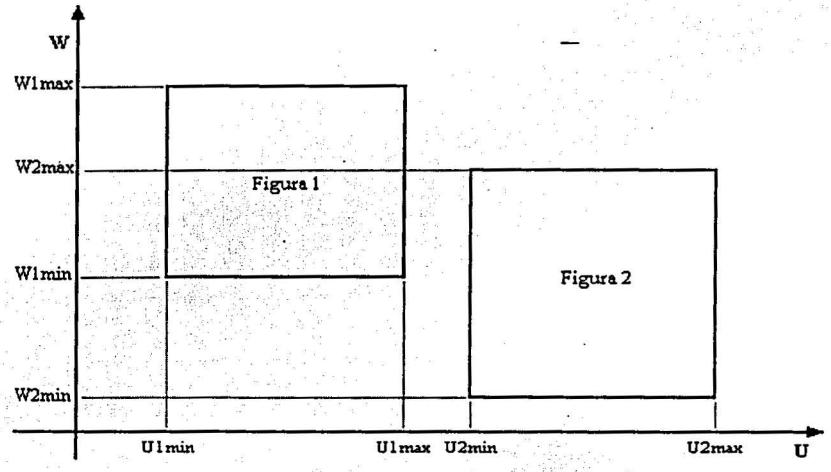


Figura III.11 Determinación de las componentes U y W.

Las figuras son proyectadas sobre líneas paralelas a los ejes U y W para obtener las componentes máximas y mínimas de cada eje (ver tabla III.3).

Figura 1	*Figura 2*
U1min	U2min
U1max	U2max
W1min	W2min
W1max	W2max

Tabla III.3 Componentes U y W de cada figura.

Utilizando los datos de la proyección de las figuras sobre los ejes U y W podemos obtener los casos mostrados en la figura III.12; donde X es un eje arbitrario.

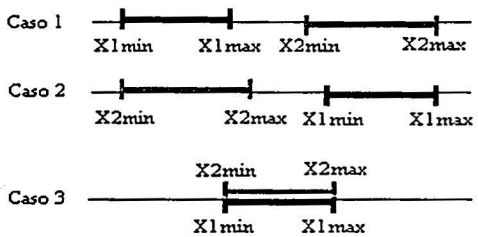


Figura III.12 Casos de proyección de dos figuras sobre una recta.

De la figura III.11 podemos observar, que si se detecta intersección en las proyecciones U y W se considera que las figuras se encuentran intersecadas.

La siguiente condición refleja cuando una figura se interseca con otra haciendo uso de la proyección de sus vértices sobre los ejes.

```
If  ( ( U1min ≤ U2min  &&  U1max ≤ U2max  &&  U2min ≤ U1max )  | |
      ( U2min ≤ U1min  &&  U2max ≤ U1max  &&  U1min ≤ U2max ) )  &&
    ( ( W1min ≤ W2min  &&  W1max ≤ W2max  &&  W2min ≤ W1max )  | |
      ( W2min ≤ W1min  &&  W2max ≤ W1max  &&  W1min ≤ W2max ) )
       Existe intersección
Else
       No hay intersección
End if
```

De manera práctica es necesario encontrar intersección en sólo dos proyecciones para garantizar que un cuerpo se interseca con otro.

Como puede observarse, el proceso de detección de colisiones ha sido planteado de tal forma que es muy simple determinar si dos prismas están colisionando, de manera que este proceso puede repetirse durante cada paso de la simulación con muy baja penalización en el tiempo de ejecución.

Dentro de Roc2, la obtención del estado de un sensor de colisión se realiza mediante una verificación de colisiones entre el cuerpo del sensor y una lista de elementos denominados obstáculos. Para cada elemento de la lista, es decir, para cada obstáculo, es necesario obtener un prisma que delimite su cuerpo de manera que se ajuste lo mejor posible; lo mismo ocurre con el

sensor: es necesario obtener un prisma que lo delimite. La obtención del prisma que envuelve un objeto se reduce a obtener los puntos que tienen las componentes con coordenadas más positivas y negativas para cada uno de los ejes. Una vez obtenido el prisma que envuelve al obstáculo y el prisma que envuelve al sensor, se aplica el algoritmo de detección de intersección antes descrito.

Existen desventajas en el algoritmo utilizado debido a las consideraciones anteriores; una de ellas es el hecho de que un cuerpo puede tener asociado un prisma que puede encerrarlo en su totalidad pero no hay garantía de que dicho prisma deje espacios vacíos al encerrar al cuerpo; la figura III.13 muestra un prisma que encierra a un cuerpo dejando espacios vacíos.

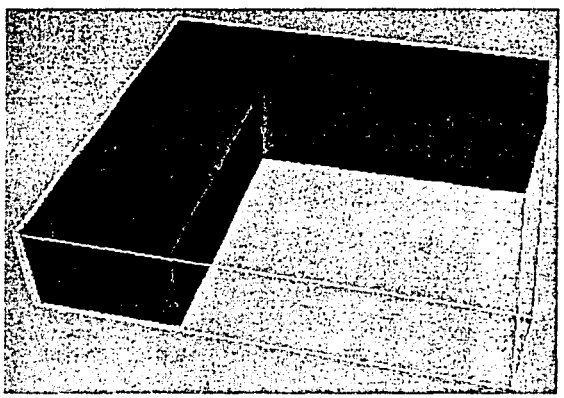


Figura III.13 Prisma con espacios vacíos.

Un prisma que deja espacios vacíos al envolver a un cuerpo, ocasiona que se detecten colisiones con lo que se denomina espacios fantasma. Un espacio fantasma es precisamente la zona vacía que queda al encerrar a un cuerpo; al realizar el cálculo de intersección de cuerpos usando el prisma que los limita, los espacios fantasma son considerados dentro del algoritmo lo cual no debería suceder ya que son zonas vacías. Una forma de solucionar el problema de los espacios fantasma consiste en dividir al cuerpo en cuerpos más pequeños, de tal forma que los espacios fantasma se reduzcan significativamente o desaparezcan.

Un robot puede o no colisionar con obstáculos dentro del mundo 3D sin ser percibido por sus sensores de contacto; los sensores de contacto pueden determinar cuándo se está realizando una colisión, pero debido a su disposición en el robot no son infalibles, por lo que existen rutinas dentro del sistema que evitan que el robot atraviese obstáculos de manera indeseable.

3.4.2 SENSORES REFLECTIVOS

Estos sensores son capaces de detectar un espectro de luz no perceptible al ojo humano denominado infrarrojo, cuya longitud de onda oscila aproximadamente entre 10^{-6} y 10^{-3} metros.

El principio de funcionamiento de un sensor reflectivo es emitir un haz de luz infrarroja sobre alguna superficie y luego capturar el rayo reflejado; dependiendo de la superficie, sólo un porcentaje del rayo emitido es reflejado y recibido; sobre superficies blancas el porcentaje de rayo reflejado es muy cercano a 100% y en superficies obscuras, como el color negro, el porcentaje es muy cercano a 0%.

La acción de emitir un rayo para determinar que tan reflectiva es la superficie sobre la que incide, implica poder obtener las características luminosas de la superficie donde se encuentra el punto de incidencia del haz de luz del sensor; de esta forma los sensores de los robots determinan, sobre una pista[11], el valor promedio de iluminación del punto donde se incide.

Crear un sensor reflectivo en un mundo 3D se reduce a determinar el valor promedio de los píxeles[12] adyacentes al punto de incidencia de la luz emitida por el sensor. En nuestro caso particular, el sensor se encuentra posicionado en la parte más baja del robot y orientado para poder recibir lecturas del suelo donde se encuentra; la figura III.14 muestra un ejemplo de sensor reflectivo. Se han tomado ciertas consideraciones que permiten simplificar el proceso de lectura del sensor.

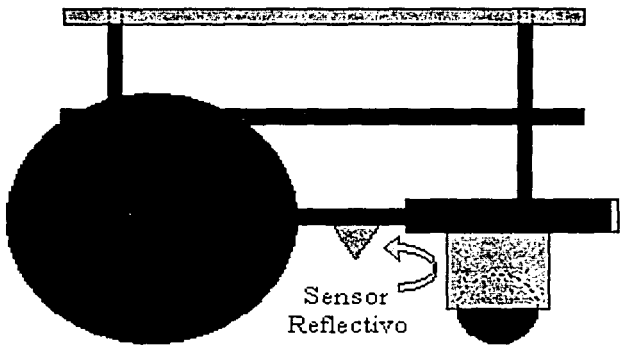


Figura III.14 Posición de un sensor reflectivo.

Para modelar los sensores de infrarrojo se tomó la siguiente consideración:

- La lectura del sensor de infrarrojo sólo se realizará sobre pistas predefinidas

[11] *Pista*. Para mayor información acerca de las pistas, consulte el apartado 3.4.2.1 en el capítulo III.

[12] *Píxel*. Un píxel (abreviatura del término inglés "picture element") es uno de los miles de minúsculos puntos que aparecen en la cuadrícula de una pantalla o de una hoja impresa. Estos puntos tienen cada uno su color para mostrar imágenes en la pantalla de la computadora, y representan los elementos más pequeños que pueden manipularse para generar gráficos. Como no son infinitamente pequeños, los píxeles sólo se aproximan al color real de un objeto. Por esta razón, las líneas de los gráficos generadas por una computadora (denominadas mapas de bits) suelen tener un aspecto dentado si se miran de cerca.

3.4.2.1 Pistas

Las pistas son mapas de bits[13] elaborados por el usuario; su representación dentro del mundo 3D es un plano al nivel del suelo que muestra la imagen de la pista. La figura III.15 muestra un robot móvil y una pista.

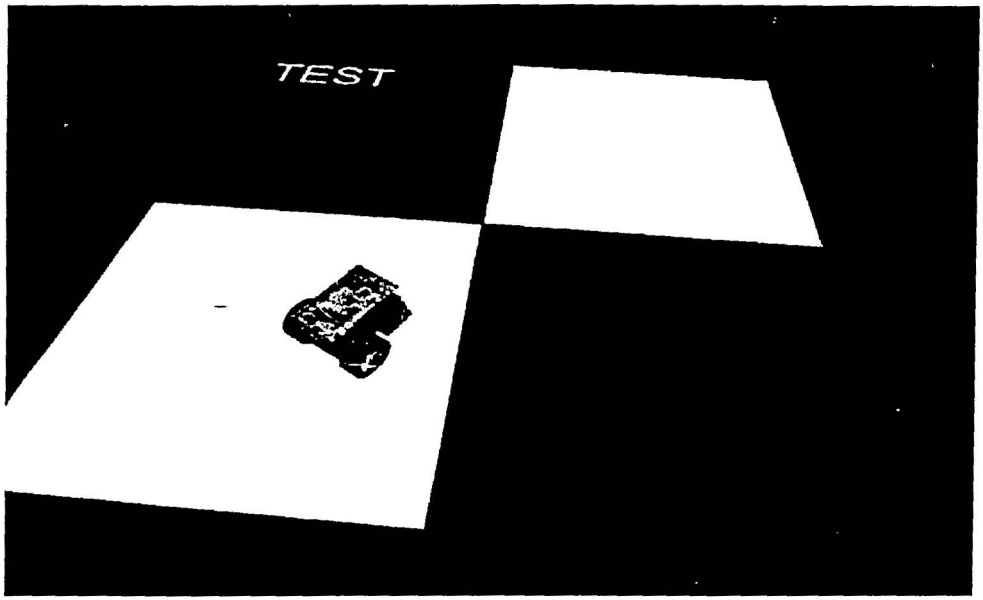


Figura III.15 Ejemplo de pista para lectura del sensor reflectivo.

Dentro de Roc2 cada pista es un elemento que pertenece a la escena gráfica ya que necesita una representación visual; pero también necesita una estructura lógica que permita realizar operaciones de obtención de datos, es decir, un conjunto de información y métodos para obtener dicha información.

Estructura lógica

- Nombre. Es el nombre del archivo que contiene el mapa de bits que forma a la pista.
- Dimensiones de la imagen. Tamaño en píxeles del mapa de bits (imagen) asociado a la pista.

[13] *Mapa de bits*. En informática, son gráficos por computadora almacenados y mantenidos como colecciones de bits que describen las características de los píxeles individuales en la pantalla.

ese tipo puede arrojar datos erróneos debido a que se realiza una conversión a escala de grises. El tamaño de la imagen no está restringido, pero se recomienda que no sea demasiado grande ya que afecta al rendimiento del sistema.

Obtención de matriz de datos. Dentro del archivo de la imagen se encuentran la información de los colores y su posición relativa; esta información es precisamente la que interesa en el proceso de lectura de sensores. Debido al formato que se usa para almacenar la imagen, acceder a los colores no es un proceso inmediato, por lo que se propone el uso de una matriz donde se almacenará sólo la información necesaria para saber qué tono se encuentra en dicho punto. La matriz obtenida con los datos de colores y sus posiciones se almacena en un archivo que tiene la siguiente forma (ver figura III.17):

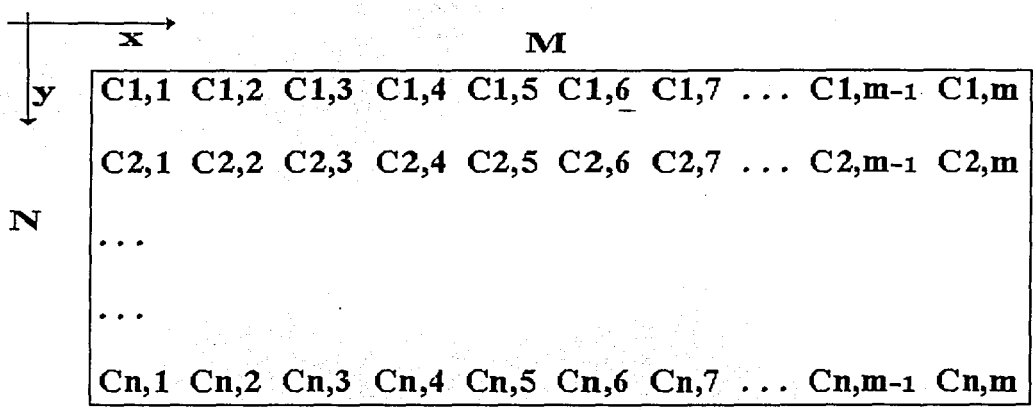


Figura III.17 Formato del archivo con la matriz de datos.

Una vez que los tonos de gris han sido almacenados dentro del archivo, es necesario contar con una función de mapeo que nos permita obtener la posición de un píxel en específico. Debido a que un archivo se maneja como un conjunto de datos secuenciales, es necesario calcular a qué distancia se encuentra a partir del inicio del archivo el dato deseado.

Tomando en cuenta que la imagen tiene dimensiones de M x N píxeles; obtener la posición del píxel (x,y) se determina a través de:

$$pos = M * y + x$$

Creación del modelo gráfico. Dentro del mundo virtual es necesario mostrar la pista que se ha añadido a la simulación. En este punto se crea un polígono con las dimensiones y en la posición que el usuario especifica para la pista; al polígono se le añade la imagen como textura para que tenga la apariencia deseada. La dimensión de la pista y su posición es almacenada dentro del sistema.

- Dimensiones de la pista. Dentro del mundo 3D la pista puede tener, o no, el mismo tamaño que el mapa de bits; por ello, es necesario almacenar el tamaño de la pista independientemente del tamaño del mapa de bits.

- Posición. Posición de la pista dentro del mundo 3D.

- Dimensiones del sensor. Es el tamaño del sensor. Cuando se realiza una lectura de sensor, corresponde al área, sobre el mapa de bits, que se considera para calcular el porcentaje de luz reflejada por la superficie (pista).

- Funciones para manipulación de datos. Considerando como datos los valores de los píxeles de la imagen asociada a la pista, debemos de poder obtener el valor de los píxeles en un punto dado y también poder especificar cuantos píxeles hay que considerar dentro de la lectura; para esto existen dos funciones:

    - Redefinir las dimensiones del sensor
    - Lectura de sensor

3.4.2.2 Creación de una pista

El proceso de creación de una pista consta de 4 pasos como se muestra en la figura III.16. Este proceso se realiza antes de la visualización de la pista en el mundo virtual, por lo que se consideran tareas en tiempo real.

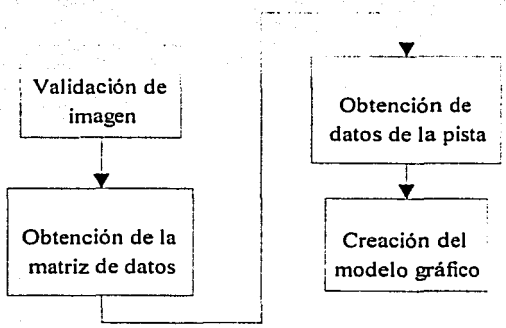


Figura III.16 Proceso de creación de una pista.

Validación de imagen. El mapa de bits (imagen) que va formar la pista debe ser analizado para verificar que cumpla con las consideraciones especificadas; es decir, la imagen debe estar dada en formato TGA[14] sin compresión y ocho bits de color. Es posible que la imagen no contenga únicamente tonos de gris, pero de antemano se indica que una lectura de sensor con una imagen de

[14] *TGA*. Es uno de los formatos para guardar mapas de bits que se utilizan de forma habitual en los programas de dibujo.

Lectura del sensor

El proceso de lectura del sensor se divide en tres partes:

- Determinar la posición del sensor sobre la pista
- Obtener los píxeles, en tonos de gris, correspondientes a la posición del sensor
- Realizar un promedio ponderado de los tonos de gris obtenidos en el punto deseado

Para determinar la posición del sensor, relativa a la pista, necesitamos lo siguiente:

- posición del sensor (SenPos)
- posición de la pista (TrackPos)
- las dimensiones de la pista dentro del mundo virtual (TrackLong y TrackWidth)

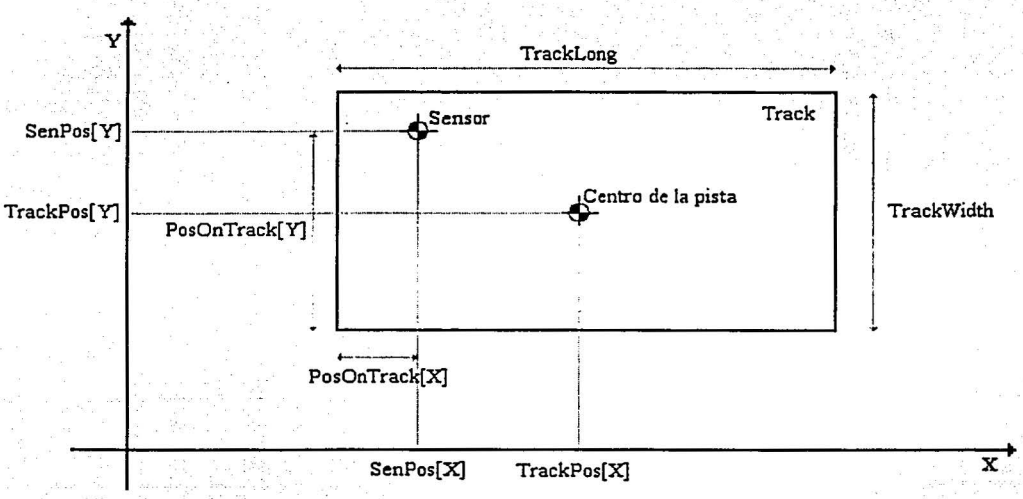


Figura III.18 Obtención de la posición relativa a la pista.

El siguiente discriminante nos permite saber si el sensor se encuentra dentro de la pista o no:

(SenPos[X] ≤ (TrackPos[X]+TrackLong/2)) && (SenPos[X] ≥ (TrackPos[X]-TrackLong/2)) && (SenPos[Z] ≤ (TrackPos[Z]+TrackWidth/2)) && (SenPos[Z] ≥ (TrackPos[Z]-TrackWidth/2))

Para determinar la posición del sensor sobre la pista (PosOnTrack) se utiliza la siguiente ecuación de mapeo:

```
PosOnTrack[X]=-SenPos[X]+ TrackPos[X]+TrackLong/2;
PosOnTrack[Y]= SenPos[Z] - TrackPos[Z]+TrackWidth/2;
```

Una vez obtenida la posición del sensor relativa a la pista, es necesario obtener los píxeles adyacentes para posteriormente ponderar la lectura.

Para calcular la cantidad de píxeles adyacentes es necesario tomar en cuenta las dimensiones del sensor y las dimensiones de la pista; las dimensiones del sensor afectan al área a considerar y las dimensiones de la pista nos permiten calcular cuantos píxeles hay por unidad en el mundo virtual.

La siguiente ecuación determina cuántos píxeles se deben tomar:

```
PixelsX= sensorWidth* imaPixelsWidth/TrackWidth;
PixelsY= sensorLong  * imaPixelsLong/TrackLong;
```

Para ponderar la lectura de los píxeles obtenidos se consideró un promedio de todas las lecturas tomadas. El valor obtenido es el resultado de la lectura del sensor, pero se adiciona un poco de ruido para simular el funcionamiento de un sensor real.

3.4.3 SENSORES DE LUZ INFRARROJA

Los sensores de infrarrojo requieren de un emisor y de un receptor para trabajar apropiadamente. El emisor se encarga de generar los rayos infrarrojos, mientras que el receptor se encarga de captar la señal de infrarrojo y evaluar la intensidad del haz recibido.

A continuación se describen las características y las restricciones consideradas para el modelado de los sensores infrarrojos dentro de la simulación (ver figura III.19).

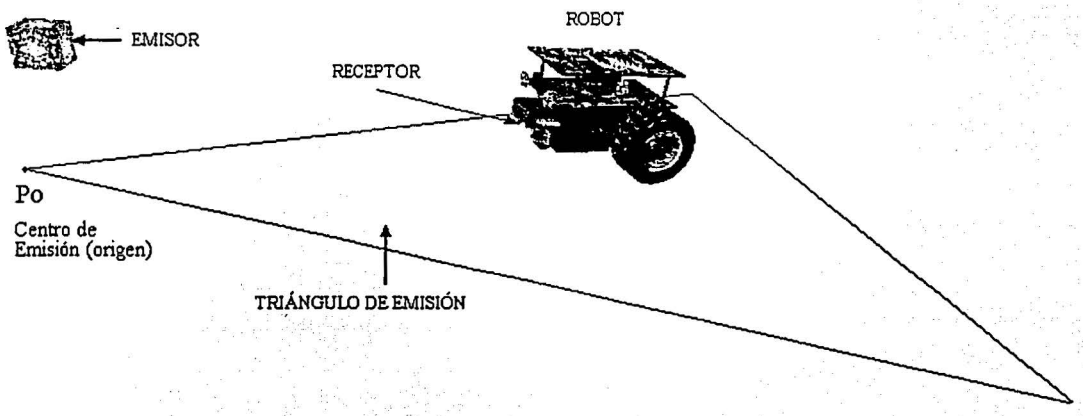


Figura III.19 Sensor de infrarrojo.

3.4.3.1 Modelado del emisor

- Los rayos infrarrojos provenientes del emisor limitan una zona de emisión dada por los límites de un triángulo isósceles. El triángulo de emisión tiene un ángulo de abertura θ, una altura h y P_o es el centro emisión (ver figura III.20).

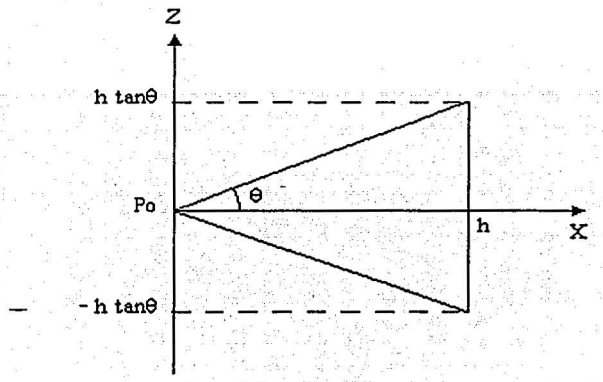


Figura III.20 Zona de emisión del sensor de infrarrojo.

- El área de emisión puede rotarse α radianes sobre el plano XZ, así como trasladarse x_1 unidades a lo largo del eje X y z_1 unidades a lo largo del eje Z. Los valores válidos para el ángulo de rotación α son de 0 a 2π radianes y se miden en sentido contrario a las manecillas del reloj (ver figura III.21).

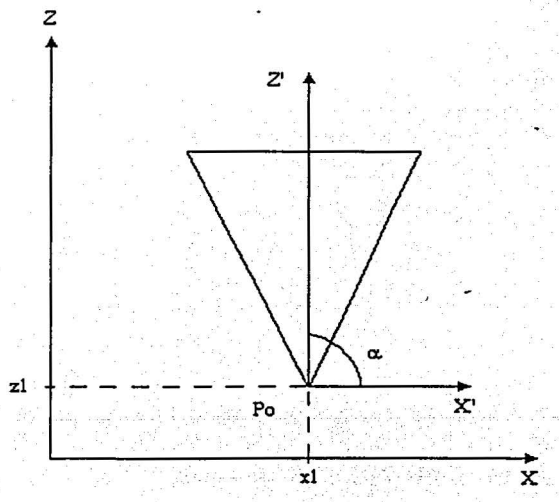


Figura III.21 Rotación y traslación del triángulo de emisión.

- La intensidad del haz de luz varía de acuerdo a la distancia entre el receptor y el centro del emisor (el punto P_o). Por ejemplo, los puntos dentro del área, próximos a P_o, tendrán un valor de intensidad mayor que los puntos cercanos a la base del triángulo. El cálculo del valor de la intensidad se realiza de la siguiente manera (véase la figura III.22):

$$I = \frac{M - d}{M}$$

donde:

$$M = \frac{h}{\cos\theta} = \text{Distancia máxima hasta el centro de emisión}$$

$$d = |\overline{RPo}| = |\overline{Po} - \overline{R}| = \sqrt{(Pox - Rx)^2 + (Poy - Ry)^2 + (Poz - Rz)^2}$$

d = Distancia entre el centro de emisión y un punto R dentro de la zona de emisión

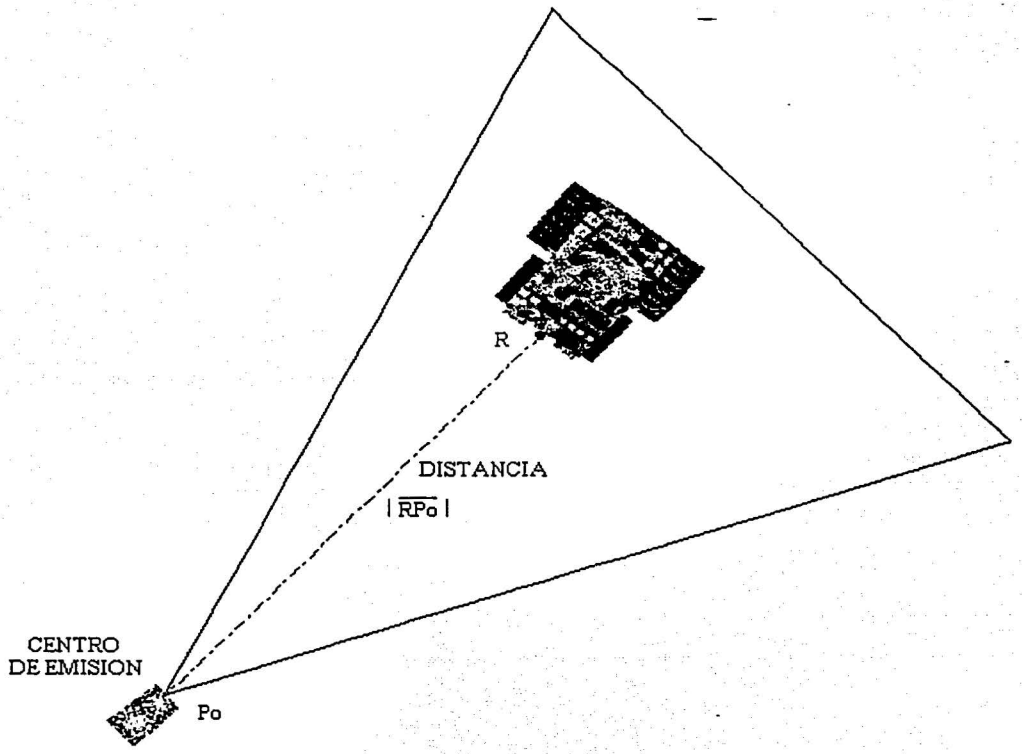


Figura III.22 Notación para el cálculo de la intensidad.

3.4.3.2 Modelado del receptor

- El receptor se representa por medio de un cubo montado sobre el robot móvil (ver figura III.23). Para describir al receptor es necesario conocer su posición y su orientación dentro de la escena gráfica; de esta manera es posible determinar si el receptor se encuentra en el triángulo de emisión, y si existe el ángulo de visión apropiado entre el emisor y el receptor para que éste último capte la señal de infrarrojo.

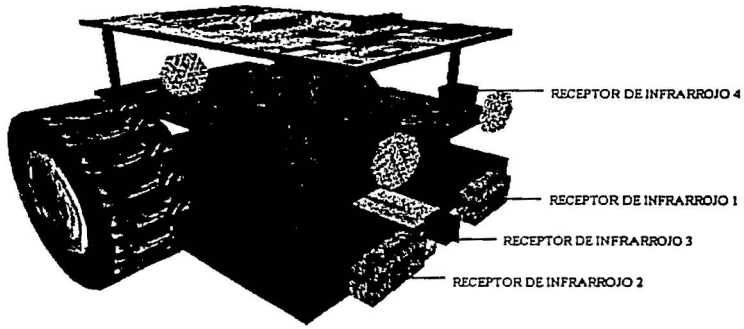


Figura III.23 Ubicación de los receptores de infrarrojo sobre el robot.

Cálculo del ángulo de visión

El ángulo de visión entre el emisor y receptor se obtiene calculando el ángulo entre los vectores de dirección de ambos elementos. El vector de dirección del emisor se obtiene a partir del ángulo α del emisor, mientras que el vector de dirección del receptor se obtiene a partir de la orientación del robot móvil. En la figura III.24 se muestra el conjunto de ángulos de visión válidos.

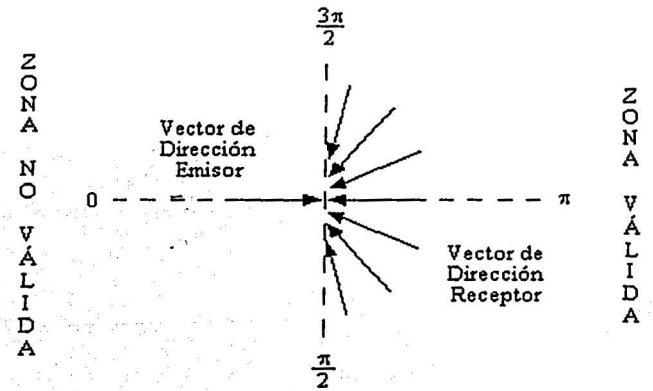


Figura III.24 Ángulos de visión válidos.

Con base en el producto punto entre ambos vectores de dirección se puede conocer el coseno del ángulo entre vectores.

$$\text{Producto\_Punto} = (\cos(\text{OrientRobot})\cos(\text{OrientEmisor})) + (\sin(\text{OrientRobot})\sin(\text{OrientEmisor}))$$

Los ángulos válidos de visión entre emisor y receptor, figura III.24, se establecieron en el rango de $\pi/2$ a $3\pi/2$ radianes, de manera que si el producto punto es menor a cero y mayor o igual a -1, indicará que el ángulo es adecuado para la recepción de rayos infrarrojos. Si se obtiene un valor diferente a los establecidos anteriormente, indicará que el receptor no es capaz de percibir los rayos infrarrojos del emisor.

Adicionalmente, se necesita verificar que el receptor se encuentre dentro del triángulo de emisión, ya que de no ser así, no tendrá sentido tener un ángulo de visión adecuado para la recepción.

Cómo determinar si un punto se encuentra dentro del área de emisión

El API de WTK incorpora una función para la prueba de intersecciones entre un polígono y un rayo (línea). Empleando esta función seremos capaces de determinar si un punto (la posición del receptor) se encuentra dentro del triángulo de emisión o no.

Existen varios métodos para determinar si un rayo se interseca con un polígono. En particular, se presenta el método de Möller y Trumbore (1997) para el cálculo de intersecciones entre un rayo y un triángulo.

Sea el triángulo A definido por tres vértices, tres puntos no colineales. Entonces, la ecuación del triángulo puede definirse utilizando estos tres puntos (ver figura III.25).

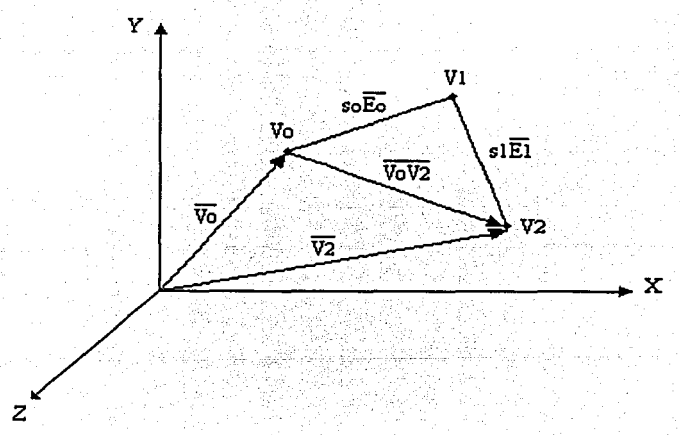


Figura III.25 Definición del triángulo.

Un plano es el conjunto de puntos P(x,y,z) tales que el vector de posición de cualquiera de ellos se puede expresar:

$$\overline{P}(x, y, z) = \overline{V_o} + s_o \overline{E_o} + s_1 \overline{E_1}$$

donde: $\overline{E_o} = \overline{V_o V_1} = \overline{V_1} - \overline{V_o}$ y $\overline{E_1} = \overline{V_1 V_2} = \overline{V_2} - \overline{V_1}$

A esta ecuación se le conoce como la ecuación vectorial del plano que se define conociendo un punto fijo V_o y dos vectores de dirección.

Existe otra forma de definir un plano conocida como forma normal; para esto, es necesario conocer un punto fijo V_o y un vector normal al plano. La normal $\overline{N}$ al plano del triángulo es un vector perpendicular al plano, dado por,

$$\overline{N} = \overline{E_o} \times \overline{E_1}$$

entonces: $\overline{V_o V_2} = \overline{V_2} - \overline{V_o}$ y $\overline{N} \bullet \overline{V_o V_2} = 0$

en general: $\overline{V_o P} = \overline{P} - \overline{V_o}$ y $\overline{N} \bullet \overline{V_o P} = 0$

Por lo tanto, la ecuación de plano en su forma normal es,

$$\overline{N} \bullet (\overline{P} - \overline{V_o}) = 0$$

Por otra parte, definir la ecuación del rayo (línea) requiere conocer un punto fijo L sobre el rayo y un vector de dirección $\overline{D}$ (ver figura III.26).

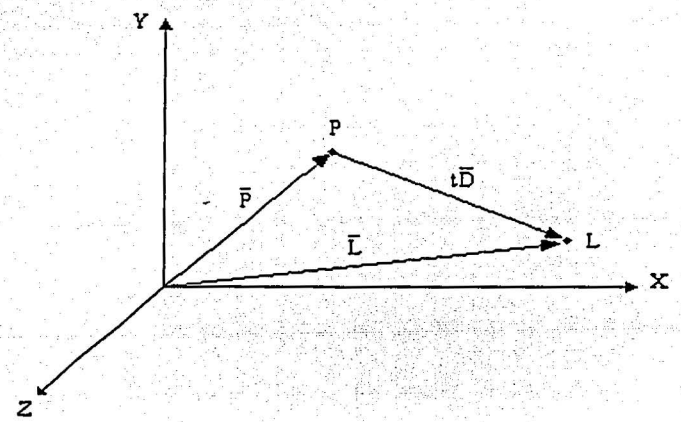


Figura III.26 Definición del rayo.

Un rayo es el conjunto de puntos L(x,y,z) tales que el vector de posición de cualquiera de ellos se puede expresar como:

$$\overline{L}(x, y, z) = \overline{P} + t\,\overline{D}$$

Una vez que se conocen las ecuaciones que describen un plano y un rayo, el primer problema es determinar si el rayo se interseca con el plano. Si $\overline{D}$ no es perpendicular a la normal del plano N, entonces el rayo se debe intersecar con el plano.

Para encontrar el punto de intersección entre el rayo y el plano, sustituimos la ecuación del rayo en la ecuación normal del plano:

$$\overline{N} \bullet (\overline{P} + t\,\overline{D} - \overline{V_o}) = 0$$

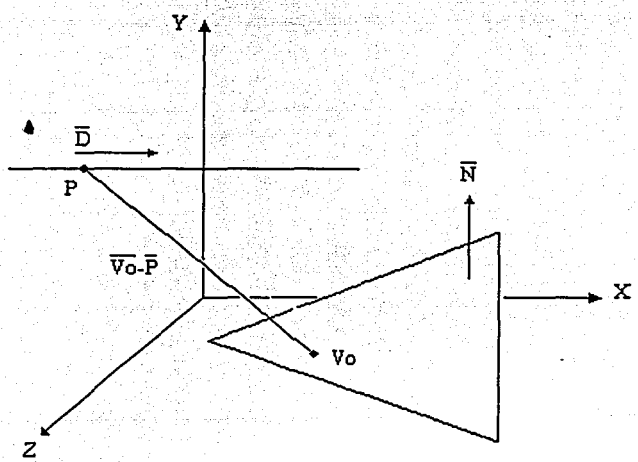
y despejamos t:

$$(\overline{N} \bullet \overline{P}) + t(\overline{N} \bullet \overline{D}) - (\overline{N} \bullet \overline{V_o}) = 0$$

$$t = T = \frac{(\overline{N} \bullet \overline{V_o}) - (\overline{N} \bullet \overline{P})}{(\overline{N} \bullet \overline{D})} = \frac{\overline{N} \bullet (\overline{V_o} - \overline{P})}{\overline{N} \bullet \overline{D}}$$

La ecuación anterior arroja tres casos de estudio.

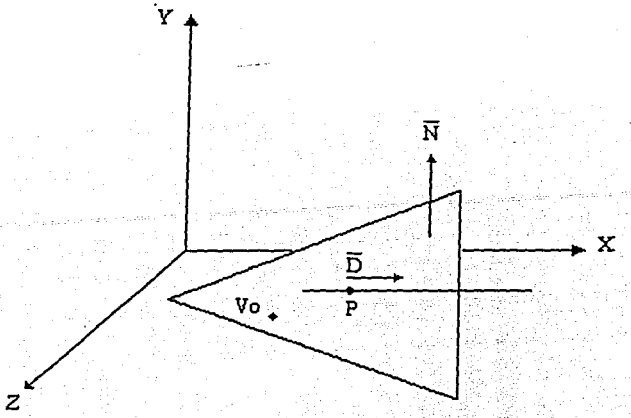
Caso 1: El rayo es paralelo al plano y no está contenido en él (figura III.27).



Condiciones:
$\overline{N} \bullet \overline{D} = 0$
$\overline{N} \bullet (\overline{V_o} - \overline{P}) \neq 0$
No existe intersección
El parámetro T no está definido

Figura III.27 Caso 1: El rayo es paralelo al plano y no está contenido en él.

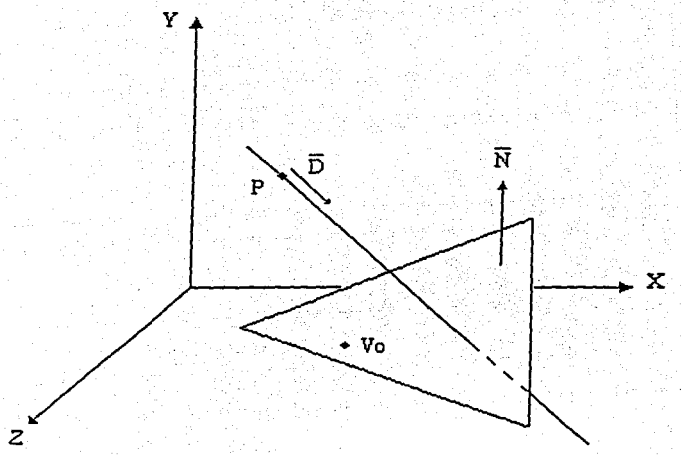
Caso 2: El rayo es paralelo al plano y está contenido en él (figura III.28).



Condiciones:
$\bar{N} \bullet \bar{D} = 0$
$\bar{N} \bullet (\bar{V_o} - \bar{P}) = 0$
Sí existe intersección
$T = 0$

Figura III.28 Caso 2: El rayo es paralelo al plano y está contenido en él.

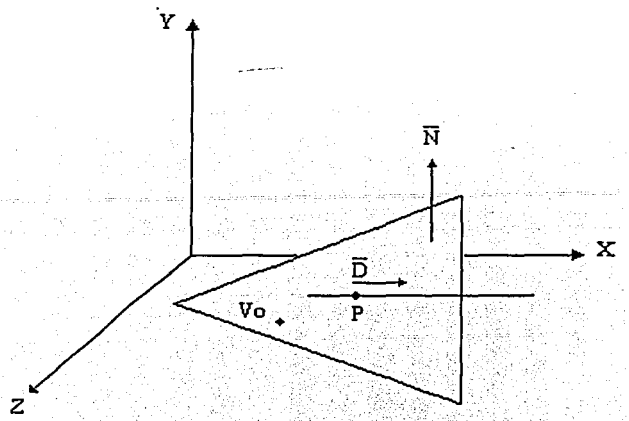
Caso 3: El rayo no es paralelo al plano, $\bar{D}$ no es perpendicular a $\bar{N}$ (figura III.29).



Condiciones :
$\bar{N} \bullet \bar{D} \neq 0$
$\bar{N} \bullet (\bar{V_o} - \bar{P}) \neq 0$
Sí existe intersección
El punto de intersección se obtiene evaluando
$\bar{L}(x, y, z) = \bar{P} + T\,\bar{D}$

Figura III.29 Caso 3: El rayo no es paralelo al plano.

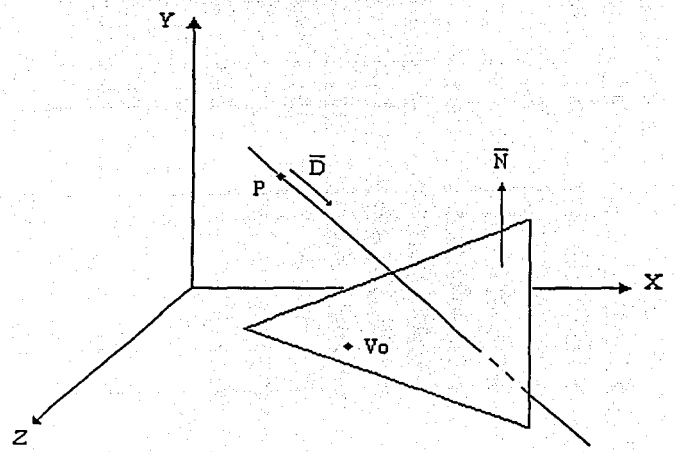
Caso 2: El rayo es paralelo al plano y está contenido en él (figura III.28).



Condiciones:
$\bar{N} \bullet \bar{D} = 0$
$\bar{N} \bullet (\bar{V_o} - \bar{P}) = 0$
Sí existe intersección
$T = 0$

Figura III.28 Caso 2: El rayo es paralelo al plano y está contenido en él.

Caso 3: El rayo no es paralelo al plano, $\bar{D}$ no es perpendicular a $\bar{N}$ (figura III.29).



Condiciones :
$\bar{N} \bullet \bar{D} \neq 0$
$\bar{N} \bullet (\bar{V_o} - \bar{P}) \neq 0$
Sí existe intersección
El punto de intersección se obtiene evaluando
$\bar{L}(x,y,z) = \bar{P} + T\,\bar{D}$

Figura III.29 Caso 3: El rayo no es paralelo al plano.

Para un punto válido de intersección, el segundo problema es determinar si el punto está dentro del triángulo. Para esto, escribimos la ecuación vectorial del plano de la siguiente manera:

$$\overline{L}(T) = \overline{V_0} + s_0 \overline{E_0} + s_1 \overline{E_1}$$

Los coeficientes s_0 y s_1 se calculan resolviendo el siguiente sistema lineal de ecuaciones.

$$\begin{bmatrix} \overline{E_0} \bullet E_0 & \overline{E_0} \bullet \overline{E_1} \\ \overline{E_0} \bullet \overline{E_1} & \overline{E_1} \bullet \overline{E_1} \end{bmatrix} \begin{bmatrix} s_0 \\ s_1 \end{bmatrix} = \begin{bmatrix} \overline{E_0} \bullet \overline{Q} \\ \overline{E_1} \bullet \overline{Q} \end{bmatrix}$$

donde: $\overline{Q} = \overline{L}(T) - \overline{V_0} = \overline{P} + T\overline{D} - \overline{V_0}$

Si $s_0 \geq 0$, $s_1 \geq 0$ y $s_0 + s_1 \leq 1$, entonces el punto de intersección está dentro del triángulo.

Por último, se presenta el algoritmo utilizado en la simulación para implantar los sensores infrarrojos. Esta función recibe como parámetro el número de sensor de infrarrojo del que se desea conocer el estado, y regresa un valor de punto flotante entre 0 y 1 que representa la intensidad del rayo infrarrojo en un punto dentro del triángulo de emisión.

```
float ReadInfraredSensor ( int numSensor ) {
    Obtener la posición del receptor de infrarrojo numSensor en la escena gráfica
    Construir un rayo utilizando la posición del receptor y el vector de dirección D = (0,1,0),
    un vector que apunta hacia abajo verticalmente

    FOR ( Cada emisor dentro de la escena gráfica ) {
        Calcular el ángulo de visión entre el emisor y el receptor

        IF ( Ángulo de visión válido ) {
            Calcular si el rayo se interseca con el triángulo de emisión

            IF ( El rayo se interseca con el polígono que define al triángulo de emisión ) {
                Obtener la posición del centro de emisión (el punto Po del emisor)
                Calcular la distancia del receptor (punto de intersección entre el rayo
                y el triángulo de emisión) hacia el centro de emisión
                Calcular la intensidad del rayo infrarrojo en el punto de recepción
                y regresar este valor de intensidad
            }
        }
    }

    Regresar cero como valor de intensidad .
}
```

3.4.4 SENSORES DE ULTRASONIDO

La mayoría de los sensores de ultrasonido se basan en la emisión de un pulso de ultrasonido cuyo lóbulo, o campo de acción, es de forma cónica. Midiendo el tiempo que transcurre entre la emisión del sonido y la percepción del eco se puede establecer la distancia a la que se encuentra el obstáculo que ha producido la reflexión de la onda sonora, mediante la fórmula:

$$d = \frac{V t}{2}$$

donde V es la velocidad del sonido en el aire y t es el tiempo transcurrido entre la emisión y recepción del pulso. A este sensor también se le conoce como SONAR, acrónimo del inglés "sound navigation and ranging" (navegación y recorrido mediante sonido).

A continuación se describen las características y las restricciones consideradas para el modelado de los sonares dentro de la simulación.

Modelado del Sonar

- Dentro de la simulación, el sonar tiene forma cilíndrica y está montado sobre el cuerpo de los robots móviles (ver figura III.30).

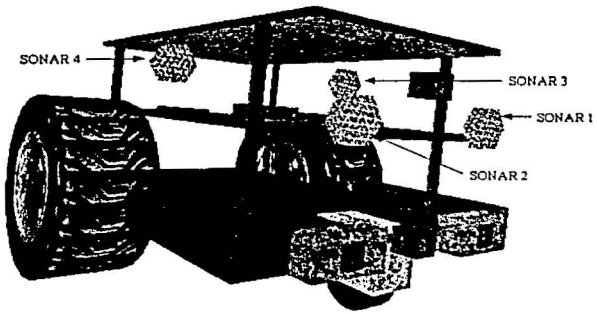


Figura III.30 Ubicación de los sonares en el robot.

- El campo de acción del sonar se limita a un rayo. El rayo se define mediante un punto fijo y una dirección, el punto fijo es la posición del sonar en la escena y la dirección es igual a la orientación que presenta el robot móvil en ese instante. De esta manera, el rayo siempre apunta hacia el frente del robot.

- Para calcular qué elemento está enfrente del robot, se prueban intersecciones entre el rayo y todos los polígonos de los posibles obstáculos con los que el robot puede colisionar. Los obstáculos con los que se podría colisionar se manejan dentro de una lista de obstáculos.

- La lectura entregada por el sonar representa la distancia que hay de la posición del sonar al polígono del obstáculo más próximo al robot.

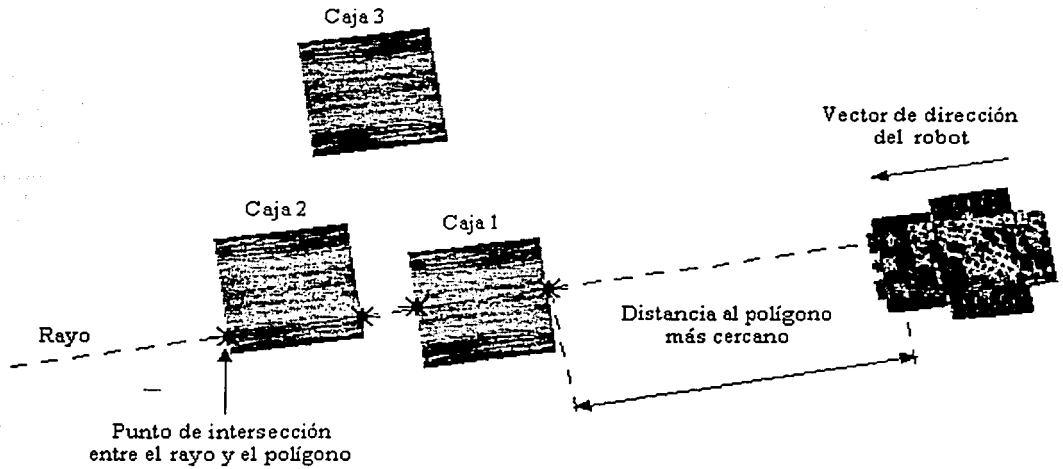


Figura III.31 Características del sonar.

A continuación se presenta el algoritmo utilizado en la simulación para implantar los sonares. Esta función recibe como parámetro el número de sonar del que se desea conocer el estado, y regresa un valor de punto flotante que representa la distancia del sonar a alguno de los obstáculos. Si no se encuentra obstáculo alguno, la función regresa un valor de cero.

```
float ReadSonarSensor( int numSensor ) {
    _Distancia_ = 0
    Obtener la posición del sonar numSensor en la escena gráfica
    Construir un rayo utilizando la posición del sonar y un vector de dirección dado por la
    orientación del robot: D = ( cos (OrientRobot), 0, sen (OrientRobot) )

    FOR ( Cada obstáculo contenido en la lista de obstáculos ) {
        FOR ( Cada polígono que compone al obstáculo ) {
            Probar si el rayo se interseca con el polígono
            IF (El rayo se interseca con el polígono ) {
                Calcular la distancia al polígono. Este valor es la distancia que existe de
                la posición del sonar al punto de intersección del rayo con el polígono
                Guardar el valor de la distancia mínima, es decir, la distancia
                al polígono más cercano al robot
            }
        }
    }
    Regresar el valor de _Distancia_
}
```

3.5 FUNCIONAMIENTO DEL ROC2

3.5.1 INICIALIZACIÓN DE LA SIMULACIÓN

En esta etapa la interfaz 3D es preparada para entrar al ciclo principal de simulación perteneciente al WTK. La preparación consiste en preprocesar la escena gráfica actual, geometrías, luz y añadir a la simulación las funciones que nos permiten llevar un control del programa; esto pertenece a las tareas que no se realizan en tiempo real, por lo que el tiempo que tarda en ejecutar este punto no es poco y se incrementa dependiendo de la complejidad de la escena.

3.5.2 EJECUCIÓN DE TAREAS DEL SISTEMA

Las tareas de sistema son una parte muy importante del ciclo de simulación. Por medio de ellas es posible asignar comportamientos especiales e individuales a los objetos que componen una escena gráfica. Algunos ejemplos del tipo de comportamientos que pueden definirse utilizando tareas son los siguientes:

- Movimientos de objetos y cámaras
- Cambios de apariencia en los objetos
- Prueba de intersecciones entre objetos (detección de colisiones)
- Activación de otros comportamientos

Todas las tareas dentro del Roc2 tienen asignada una prioridad que especifica el orden en el que la tarea se ejecutará dentro del ciclo de simulación. Esta prioridad se determina mediante un número, donde las tareas con números menores se ejecutarán antes que las tareas con números mayores. La prioridad de una tarea es un concepto que debe tenerse en mente, ya que existen tareas que requieren ejecutarse antes que otras para que estas últimas puedan operar correctamente.

Las tareas dentro del ciclo de simulación también tienen asignado un estado de activación, el cual indica si la tarea es ejecutada o no durante la simulación. Una tarea puede activarse o desactivarse en cualquier momento de la simulación.

A continuación se describen las tareas utilizadas dentro del sistema Roc2 (ver tabla III.4).

Tipos de tareas empleados:

- Detección de colisiones. Prueban intersecciones entre objetos.
- Captura de comandos. Registran entradas (comandos / órdenes) provenientes del exterior.
- Selección de objetos. Permiten seleccionar ciertos objetos de la escena utilizando los dispositivos periféricos.
- Movimiento. Calculan las posiciones y orientaciones para los objetos.
- Ajuste de cámaras. Ajustan las perspectivas (puntos de vista) desde donde el usuario observa la escena.

TAREA [PRIORIDAD]	*TIPO*	*DESCRIPCIÓN*
CollitionTask [1]	Detección de colisiones	Prueba intersecciones entre un objeto de tipo bomba[15] y un objeto de la lista obstáculos.
ReadCollitionSensor [1]	Detección de colisiones	Prueba intersecciones entre un objeto del tipo MiniRobot y un objeto de la lista obstáculos.
UseSNESPAD [1]	Captura de comandos	Lee continuamente el puerto paralelo en espera de un comando procedente del SNESPad (control de juegos).
CheckWhenIs ReachedDestiny [2]	Movimiento	Determina si un objeto del tipo MiniRobot ha alcanzado su posición destino.
RotateRobot [3]	Movimiento	Calcula las rotaciones (orientación) para un objeto del tipo MiniRobot.
MoveRobot [4]	Movimiento	Calcula las traslaciones (posiciones) para un objeto del tipo MiniRobot.
MoveBackWheels [5]	Movimiento	Animación. Calcula las rotaciones para las ruedas del objeto MiniRobot. Simula que las ruedas giran mientras el objeto avanza.
AjustNearView [6]	Ajuste de cámaras	Ajusta la posición y orientación de una cámara (punto de vista) colocada dentro del objeto MiniRobot.
AjustFarView [6]	Ajuste de cámaras	Ajusta la posición y orientación de una cámara lejana (punto de vista). Esta cámara sigue el movimiento de un objeto MiniRobot desde una posición fija.
SelectRobotUsing Mouse [7]	Selección de objetos	Permite seleccionar un objeto MiniRobot desde cualquiera de las perspectivas utilizando el mouse.
ReadWebCommand [7]	Captura de comandos	Verifica si existe un dato válido dentro del buffer[16] de datos recibidos por Internet. La captura de este comando se realiza a través de un socket[17].
Animation [10]	Movimiento	Cambia las texturas sobre el techo de un cuarto simulando la presencia y movimiento de las nubes.
BombAnimation [10]	Movimiento	Animación. Calcula las posiciones y orientaciones de los pies del objeto Bomba dando la impresión que éste camina.
MoveBomb [10]	Movimiento	Mueve al objeto Bomba a lo largo de una trayectoria definida. Calcula las posiciones para un objeto del tipo Bomba.
Explosion [10]	Movimiento	Animación. Intercambia un conjunto de texturas sobre un arreglo de polígonos simulando una explosión.
MoveFruits [12]	Movimiento	Calcula las posiciones para los frutos del objeto Árbol simulando que éstos caen.

[15] *Objeto de tipo Bomba.* Consulte el capítulo V para mayor información acerca de los obstáculos de tipo bomba.

[16] *Buffer.* En informática, es un depósito de datos intermedio, es decir, una parte reservada de la memoria en la que los datos son mantenidos temporalmente hasta tener una oportunidad de completar su transferencia hacia o desde un dispositivo de almacenamiento u otra ubicación en la memoria.

[17] *Socket.* Un socket es el punto final de un enlace de comunicación de dos vías entre dos programas que se ejecutan a través de la red.

ExeScript [15]	Captura de comandos	Lee un archivo de comandos para el objeto MiniRobot y los ejecuta.

Tabla III.4 Descripción de las tareas utilizadas en Roc2.

Y el diagrama de flujo que muestra el orden de ejecución de las tareas se presenta a continuación (figura III.32).

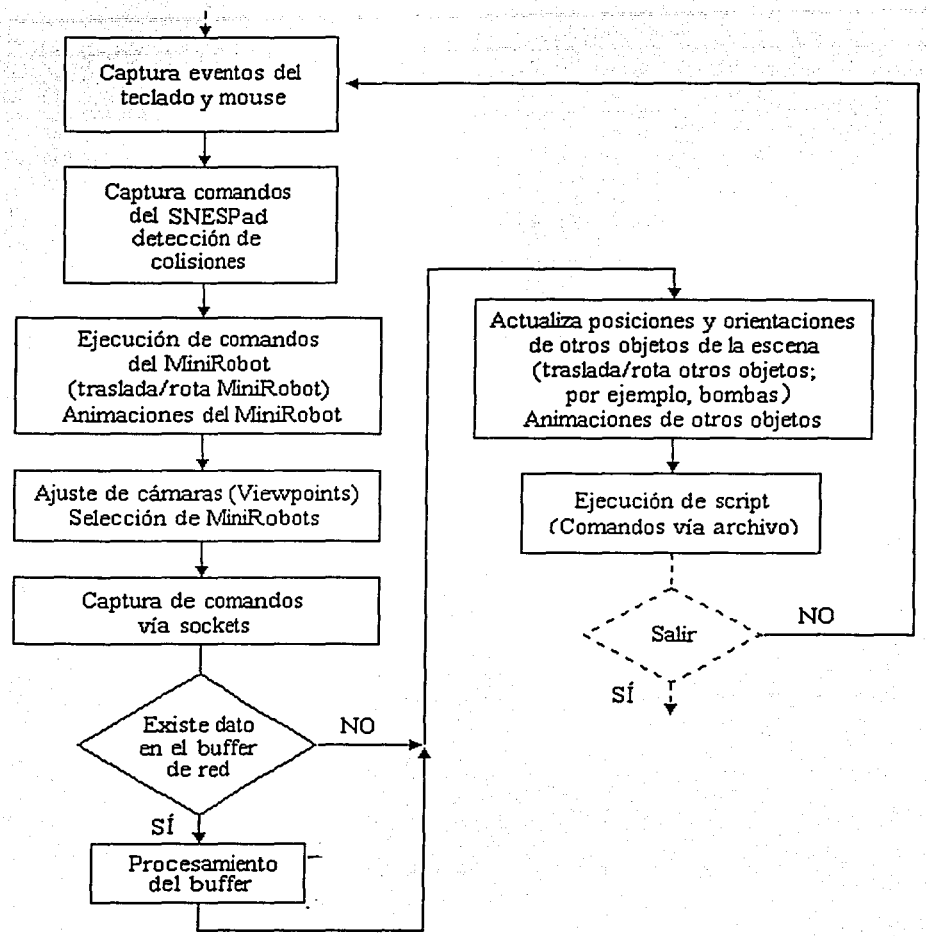


Figura III.32 Orden de ejecución de las tareas dentro del ciclo de simulación.

A continuación se describen con más detalle las tareas que pertenecen al ciclo de simulación:

Captura de eventos del teclado y mouse

En esta tarea, el Roc2 lee el buffer de teclado y el estado del mouse. Algunas de las teclas tienen asignados eventos que son inicializados al presionarlas; para controlar un robot móvil las flechas del teclado tienen asignados comandos básicos como son: movimiento hacia adelante, hacia atrás, a la izquierda y a la derecha, por lo que es posible trasladar un robot por toda la escena usando sólo el teclado. El mouse es utilizado para cambiar la posición y orientación del punto de vista.

Captura de comandos del joypad

Una de las formas para controlar los robots del Roc2 es utilizando un control de SNES; para obtener el estado del joypad se utiliza un método de polling[18] que se describe en el apéndice B. La cruz direccional tiene asignada para cada una de sus posiciones un comando básico: movimiento adelante, atrás, derecha e izquierda. Los botones pueden o no tener asignados otros comandos a manera de macros.

Detección de colisiones

El proceso de detección de colisiones es uno de los más importantes dentro del sistema, ya que evita que nuestro robot invada zonas que están siendo ocupadas por otros objetos; obteniendo así, un comportamiento más real.

Existen diversos métodos de detección de colisiones algunos de ellos están basados en el supuesto de conocer la posición de todos los elementos en el mundo y otros en el desconocimiento de los mismos. En nuestro caso se implemento un algoritmo utilizando un conocimiento predeterminado de los obstáculos que existen en el mundo.

Al crear un objeto dentro del mundo 3D se debe especificar si dicho objeto se considera un obstáculo o no. Debido a la forma en la que se diseñó el Roc2, es posible determinar el estado de los elementos haciendo uso de las siguientes reglas para determinar si un elemento es sólido:

- El elemento se encuentra en el suelo o muy próximo a él
- El elemento es visible
- El elemento no se encuentra inmerso en otro elemento

Una vez que se determina si un obstáculo es sólido se añade a una lista de objetos que se consideran obstáculos. El proceso anteriormente descrito se realiza en tiempo no real, pero es posible añadir y remover elementos durante la simulación.

Cada robot está construido con distintos elementos geométricos, por lo que verificar colisiones con cada uno de sus elementos puede ser un proceso demasiado lento. Sólo es necesario probar colisiones con algunos elementos del robot, los cuales se seleccionan garantizando que no es posible

[18] *Polling*. Se refiere al procedimiento de verificar, a intervalos regulares, a algún periférico y proporcionarle el servicio que necesita. Mediante este esquema, el periférico es considerado pasivo. Poll significa sondear.

que un robot atraviese a otro robot sin detectar una colisión; dichos elementos seleccionados se llaman elementos de contacto. En el siguiente diagrama se muestran los elementos de contacto del robot (ver figura III.33).

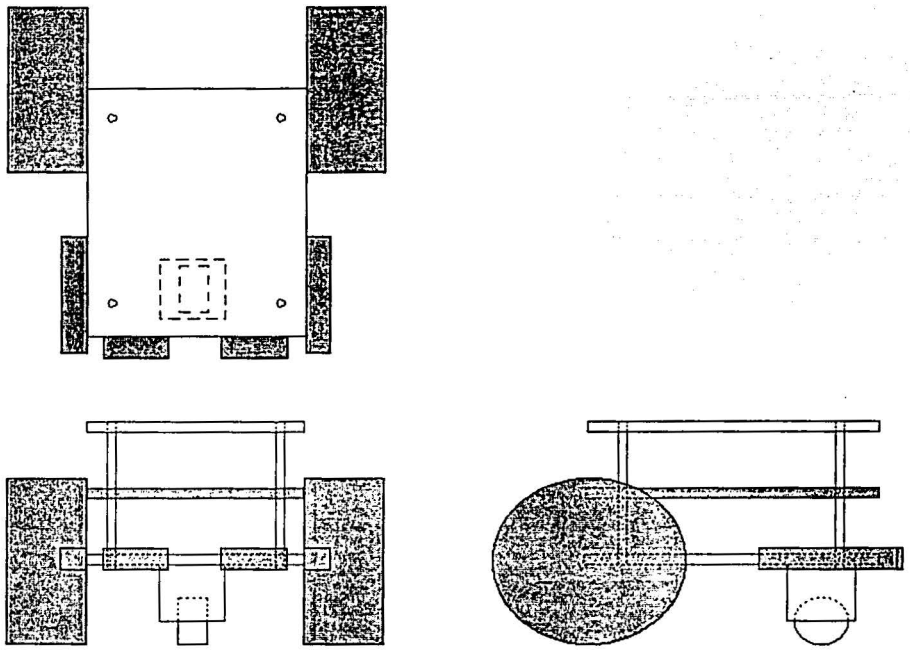


Figura III.33 Elementos de contacto del robot; son los que están sombreados.

De la figura podemos observar, que muchos elementos geométricos han sido omitidos, y sólo están presentes los que se consideran los mínimos indispensables. Es posible añadir más elementos de contacto lo cual mejoraría la calidad de la simulación pero hay que tomar en cuenta que el rendimiento final del sistema se vería afectado si la lista fuera demasiado grande.

Los elementos de contacto de cada robot se añaden a la lista de obstáculos, ya que se vuelven obstáculos para otros robots, por lo que, finalmente, la lista de obstáculos está formada por las geometrías del mundo 3D y por los elementos de contacto de los robots. Cada elemento de la lista no solo contiene la información de cada elemento geométrico sino también a qué conjunto pertenecen. Los conjuntos son identificadores que indican si el elemento pertenece al mundo o a un robot en específico.

Cada uno de los elementos del robot y de los obstáculos tiene asociado un prisma que lo contiene en forma exacta; llamaremos a dicho prisma cuerpo de contacto. También a este prisma se le conoce como Bounding Box (caja de frontera).

El problema de determinar cuándo un robot se encuentra colisionando con algún obstáculo se reduce a verificar si los elementos de contacto se encuentran colisionando con los elementos de la lista, omitiendo a los que tienen el mismo identificador que él. De no omitir los elementos que tienen el mismo identificador que el propio robot, ocasionaría que el robot siempre pensara que se encuentra colisionando con un elemento, el cual es él mismo. Para verificar la colisión de los elementos geométricos se hace uso de sus cuerpos de contacto; debido a las características de los cuerpos de contacto, es posible utilizar el mismo algoritmo descrito en la sección de sensores de colisión.

Ejecución de comandos del minirobot y animaciones

El ciclo de simulación (figura II.1, del capítulo WTK) tiene como característica realizar cálculos en cada una de sus fases, afectando con esto a las geometrías existentes. Como último paso del ciclo de simulación se muestran los elementos con las transformaciones acumuladas.

Realizar un movimiento, en un solo paso, de un punto A a un punto B dentro del ciclo de simulación, tiene el efecto de aparecer el elemento en el punto B, ya que los movimientos dieron como resultado dicho punto. Es decir, no es posible ver cómo se traslada el elemento del punto A al punto B. Tenemos el mismo caso cuando realizamos rotaciones. Si se desea observar cómo se traslada el elemento poco a poco, debe hacerse un paso a la vez dentro del ciclo de simulación; éste es el caso del movimiento de los robots quienes ejecutan sus movimientos paso a paso para que sea posible notarlo.

Los movimientos de los robots son ocasionados por comandos, en este fase del ciclo de sistema, se modifican las posiciones y rotaciones de los robots para realizarlos.

Las animaciones consisten en realizar un movimiento de rotación de las llantas traseras del robot, el cual debe ser un movimiento paso a paso para poder notarlo.

Ajuste de cámaras y selección de minirobots

El proceso de modificar la posición de las cámaras dentro del sistema no es automático, debe realizarse de forma manual. En este punto del ciclo de sistema las posiciones y orientaciones de las cámaras son actualizadas. Es importante hacer notar que primero deben realizarse los movimientos de los robots y después se deben ajustar las cámaras, ya que de lo contrario mostrarían una perspectiva incorrecta.

La selección de minirobot consiste en obtener el estado del mouse sobre las ventanas de sistema, de tal forma que sea posible determinar si se está tocando (seleccionando) un minirobot. Si en realidad se está seleccionando un minirobot entonces el robot activo dentro del sistema cambia al seleccionado y las cámaras ahora muestran su posición.

Captura de comandos vía sockets

Dentro del Roc2 es posible recibir mensajes a través de Internet usando sockets de tipo UDP[19] para Windows. Una de las características de estos sockets es que cada vez que se lee el buffer del socket la aplicación detiene su ejecución hasta que se recibe algún dato. Para evitar que la aplicación se detenga, el procedimiento de lectura del buffer del socket se realiza como un nuevo proceso. Cada vez que el procedimiento que ejecuta el socket encuentra un dato válido enciende dos banderas: una para indicar que hay un nuevo dato en el buffer y otra para indicar que el dato existente no ha sido tomado. En esta parte del ciclo de sistema se verifica el estado de las banderas, si se determina que hay un dato en el buffer del socket entonces se toma el buffer y se procesa.

Actualiza posiciones y orientaciones de otros objetos en la escena

Existen otros elementos en la escena gráfica que son necesarios de animar dentro del ciclo de sistema como son las bombas, cocos y el techo de una escena gráfica. Si bien pueden o no existir estos elementos dentro de la escena, en esta fase del ciclo de sistema se realizan los movimientos paso a paso que determinan su comportamiento.

Ejecución de script

Uno de los comandos de los robots móviles consiste en ejecutar un script. Un script es un archivo de texto que contiene comandos y se ejecutan de manera secuencial, de tal forma que en cuanto un comando termina se ejecuta el siguiente hasta el final del archivo. En este punto del ciclo de sistema cada robot verifica si ha terminado de ejecutar una instrucción indicada por un script; si ya la ha terminado entonces lee el siguiente comando y lo comienza a ejecutar. Al recibir la indicación de fin de script el robot mismo determina que no es necesario volver a entrar a esta parte del ciclo de sistema por lo que remueve este paso de la simulación.

[19] *Socket UDP* (User Datagram Protocol/Protocolo de datagramas de usuario). Es un tipo de socket que está basado en el envío y/o recepción de paquetes, que son sin conexión, no secuenciados, y no ofrecen control de errores.

CAPÍTULO IV
PUNTOS DE VISTA (CÁMARAS)

4.1 VENTANAS Y PUNTOS DE VISTA

Una ventana en WorldToolKit es un objeto que corresponde a una región en la pantalla, esto es, el lugar donde será desplegada la escena gráfica. WTK provee un conjunto de métodos para el manejo de ventanas, estos métodos incluyen la creación y destrucción, cambios en el tamaño, posición sobre la pantalla y color de fondo de la ventana, y definen la manera en cómo la escena gráfica será proyectada y dibujada.

Por otra parte, un punto de vista define la posición y orientación desde la cual todas las geometrías asociadas con la simulación son renderizadas[20] y proyectadas en la ventana. Cada ventana en WTK debe tener asociado un punto de vista, el cual definirá la posición y orientación de un observador en la escena gráfica.

Cada vez que un mundo virtual es creado, automáticamente se crea un punto de vista para él. Sin embargo, para muchas aplicaciones, un punto de vista es insuficiente, por lo tanto, WTK permite la creación de diferentes puntos de vista adicionales para una misma ventana o para ventanas diferentes.

Además de la posición y orientación, un punto de vista está caracterizado por otros parámetros tales como el "aspect ratio", el paralaje y la convergencia, los cuales se explican a continuación.

El aspect ratio es un factor de escala vertical que se aplica a la imagen de la pantalla, con la finalidad de corregir la distorsión de objetos esféricos y cuadrados cuando son dibujados. El paralaje es la distancia entre los puntos de vista del ojo derecho y el ojo izquierdo en la simulación, cuando se trata de imágenes estereoscópicas. Y la convergencia es un desplazamiento horizontal, medido en píxeles, que se aplica a las imágenes de ambos ojos. Este desplazamiento es sustraído de la imagen para el ojo izquierdo y agregado a la imagen para el ojo derecho. Los valores de todos estos parámetros, incluyendo la posición y la orientación, pueden modificarse por medio de varios métodos que WTK incorpora para el manejo de puntos de vista.

4.1.1 REPRESENTACIÓN DE ORIENTACIONES EN WTK

Las orientaciones, incluyendo las de los puntos de vista, son almacenadas en forma de cuaternión (quaternion). Un cuaternión es un conjunto de cuatro valores que forman un sistema algebraico en el cual la multiplicación no es una operación conmutativa.

Un cuaternión puede representarse de la siguiente manera:

$$q = w + x\,i + y\,j + z\,k$$

donde: w, x, y, z son números reales.

[20] *Renderizador* (Renderer). Es el componente de un sistema de gráficas por computadora que tiene como responsabilidad dibujar los polígonos de los modelos de una escena gráfica usando información de estado tales como luces (iluminación), mapas de texturas, materiales, entre otros.

Álgebra de cuaterniones[21]

Sean los cuaterniones $q_0 = w_0 + x_0 i + y_0 j + z_0 k$ y $q_1 = w_1 + x_1 i + y_1 j + z_1 k$. La suma y sustracción de cuaterniones está definida por la siguiente expresión:

$$q_0 \pm q_1 = (w_0 + x_0 i + y_0 j + z_0 k) \pm (w_1 + x_1 i + y_1 j + z_1 k)$$
$$= (w_0 \pm w_1) + (x_0 \pm x_1) i + (y_0 \pm y_1) j + (z_0 \pm z_1) k$$

La multiplicación entre los elementos primitivos i, j y k está dada por: $i^2 = j^2 = k^2 = -1$, $ij = -ji = k$, $jk = -kj = i$, y $ki = -ik = j$. Entonces, la multiplicación entre cuaterniones se define de la siguiente manera:

$$q_0 q_1 = (w_0 + x_0 i + y_0 j + z_0 k)(w_1 + x_1 i + y_1 j + z_1 k)$$
$$= (w_0 w_1 - x_0 x_1 - y_0 y_1 - z_0 z_1) + (w_0 x_1 + x_0 w_1 + y_0 z_1 - z_0 y_1) i +$$
$$(w_0 y_1 - x_0 z_1 + y_0 w_1 + z_0 x_1) j + (w_0 z_1 + x_0 y_1 - y_0 x_1 + z_0 w_1) k$$

Note que la multiplicación no es conmutativa; esto es, el producto entre $q_0 q_1$ y $q_1 q_0$ no necesariamente es igual.

El conjugado de un cuaternión se define:

$$q^* = (w + x i + y j + z k)^* = w - x i - y j - z k$$

La norma de un cuaternión está dada por:

$$N(q) = N(w + x i + y j + z k) = w^2 + x^2 + y^2 + z^2$$

Entonces, un cuaternión unitario es aquel cuya norma es igual a uno, N(q)=1. Un cuaternión unitario es de suma importancia debido a su gran utilidad para representar de manera compacta las rotaciones.

Un cuaternión unitario puede representarse como sigue:

$$q = \cos\left(\frac{\theta}{2}\right) + \overline{U} \operatorname{sen}\left(\frac{\theta}{2}\right)$$

donde: $\overline{U} = x i + y j + z k$ y el vector (x, y, z) es de longitud unitaria.

El cuaternión unitario anterior representa la rotación de un vector 3D cualquiera sobre el eje 3D U un ángulo θ.

[21] *Cuaterniones*. Para mayor información acerca del álgebra de cuaterniones consulte la siguiente fuente: 3D game engine design: A practical approach to real-time computer graphics; autor, David Eberly.

Si no se está familiarizado con la notación de los cuaterniones, WTK proporciona métodos para la conversión de cuaterniones a matrices de rotación, de cuaterniones a ángulos de Euler, de cuaterniones a parejas eje-ángulo, y viceversa.

Por ultimo, falta mencionar que los operadores de rotación en WTK operan de derecha a izquierda. Esto es, cuando se multiplica un vector por una matriz, el vector es un vector renglón y la matriz está a su derecha (la matriz opera de derecha a izquierda). Los cuaterniones también siguen ésta convención.

4.2 PUNTOS DE VISTA EN ROC2

Roc2 cuenta con tres puntos de vista asignados a tres ventanas, uno por ventana. Cuando se haga referencia a estos puntos de vista se utilizará indistintamente el nombre de cámara o de punto de vista.

El primer punto de vista permite navegar libremente en la escena gráfica mediante el uso del mouse. El segundo, es dibujado con base en la posición y orientación del robot actual, de manera que el usuario observa la escena desde la perspectiva del robot. Finalmente, el tercer punto de vista coloca al usuario en un punto fijo y desde esa posición sigue los movimientos del robot actual. Es decir, se trata de una cámara que sigue el movimiento del robot desde una posición fija. Debido a que la cámara del robot y la cámara fija están continuamente cambiando de posición y orientación, es necesario implantar métodos adicionales que actualicen estas propiedades. Estos métodos se describirán a continuación.

4.2.1 CÁMARA LIBRE

El primer punto de vista está asignado a la ventana llamada "FREE Camera" o cámara libre. Este punto de vista responde a los eventos del mouse proporcionando control de seis grados de libertad para navegar la escena. En la figura IV.1 se ilustra la orientación de los ejes coordenados del sistema, vistos desde la pantalla de la computadora, y en la tabla IV.1 se listan los efectos de los eventos del mouse sobre el punto de vista.

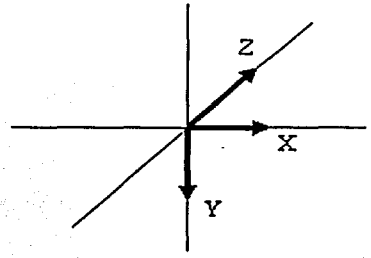


Figura IV.1 Orientación de los ejes coordenados del sistema.

Evento del mouse	*Efecto sobre el punto de vista*
Ningún botón es presionado y el cursor está centrado en la ventana.	Ninguno
El botón izquierdo es presionado y el cursor se localiza arriba del centro de la ventana.	El punto de vista se mueve hacia adelante.
El botón izquierdo es presionado y el cursor se localiza abajo del centro de la ventana.	El punto de vista se mueve hacia atrás.
El botón izquierdo es presionado y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista rota hacia la izquierda sobre el eje Y.
El botón izquierdo es presionado y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista rota hacia la derecha sobre el eje Y.
El botón derecho es presionado y el cursor se localiza arriba del centro de la ventana.	El punto de vista se mueve hacia arriba.
El botón derecho es presionado y el cursor se localiza abajo del centro de la ventana.	El punto de vista se mueve hacia abajo.
El botón derecho es presionado y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista se desplaza hacia la izquierda.
El botón derecho es presionado y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista se desplaza hacia la derecha.
Los botones izquierdo y derecho son presionados y el cursor se localiza arriba del centro de la ventana.	El punto de vista rota hacia arriba sobre el eje X.
Los botones izquierdo y derecho son presionados y el cursor se localiza abajo del centro de la ventana.	El punto de vista rota hacia abajo sobre el eje X.
Los botones izquierdo y derecho son presionados y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista rota hacia la izquierda sobre el eje Z.
Los botones izquierdo y derecho son presionados y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista rota hacia la derecha sobre el eje Z.

Tabla IV.2 Efectos de los eventos del mouse sobre la cámara libre.

4.2.2 CÁMARA DEL ROBOT

Esta cámara se localiza en el frente del robot permitiendo observar lo que el robot está viendo. Cada vez que un robot rota y se desplaza, la posición y orientación del punto de vista cambian, debido a ello, se crea una tarea que forma parte de la simulación que se encarga de ajustar los parámetros del punto de vista cada vez que la posición y orientación del robot son modificados. La figura IV.2 ilustra esta idea.

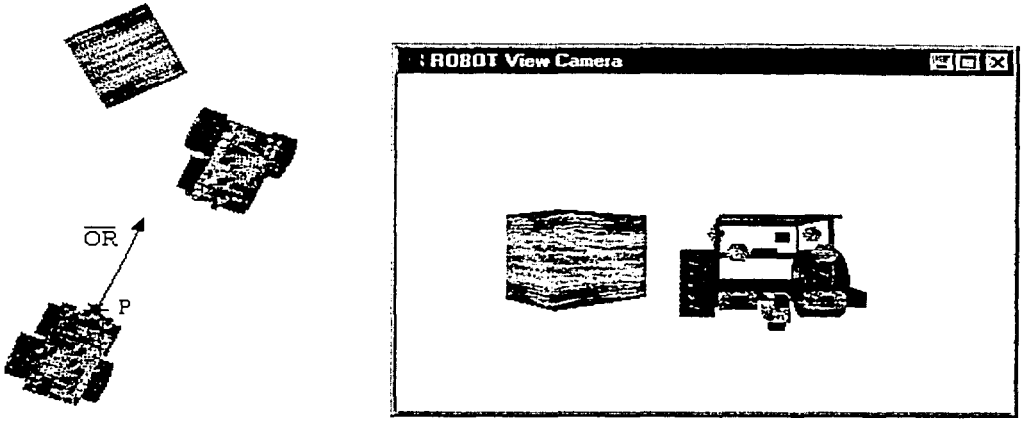


Figura IV.2 La posición y orientación del robot definen
la posición y orientación del punto de vista.

El pseudocódigo utilizado para la implantación de esta tarea se muestra a continuación.

```
void ajustNearView ( PuntoDeVista punto_de_vista ) {
        Obtener la posición actual del robot (un punto 3D)
        Obtener el ángulo de rotación actual del robot sobre el eje Y
        Construir un cuaternión que represente la orientación del robot sobre el eje Y
        Actualizar la posición del punto_de_vista en base a la posición del robot
        Actualizar la orientación del punto_de_vista con el valor del cuaternión
}
```

El cuaternión que representa la rotación del robot en el plano XZ, es decir, sobre el eje Y, es el siguiente:

$$q = \cos\left(\frac{\text{AnguloRobot}}{2}\right) + \bar{U}\ \text{sen}\left(\frac{\text{AnguloRobot}}{2}\right)$$

donde: $\bar{U} = -j$

AnguloRobot es el ángulo de rotación actual del robot sobre el eje Y, medido en radianes

4.2.3 CÁMARA LEJANA (VISTA FIJA)

La posición de este punto de vista es establecida por el usuario, y desde esa posición la cámara va siguiendo el movimiento del robot seleccionado. La orientación de este punto de vista está dada

por un vector de dirección unitario que apunta desde la posición de la cámara hacia la posición del robot (ver figura IV.3).

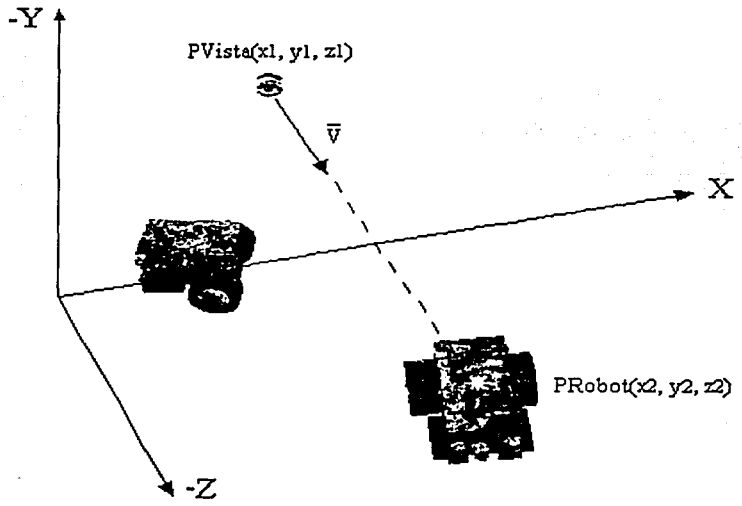


Figura IV.3 El vector de dirección $\overline{V}$ establece la orientación de la cámara fija.

El ajuste para esta cámara se realiza a través de una tarea, de manera similar al ajuste efectuado para la cámara del robot. El pseudocódigo utilizado para la implantación de esta tarea se muestra a continuación.

```
void ajustFarView (PuntoDeVista punto_de_vista ) {
        Obtener la posición actual del robot (un punto 3D, PRobot)
        Construir el vector de dirección unitario V
        Construir el cuaternión qfarView utilizando la orientación del vector de dirección V
        Actualizar la orientación del punto de vista con el valor del cuaternión calculado
}
```

El vector $\overline{V}$ se calcula,

$$\overline{V} = \overline{PVista\ PRobot} = \frac{\overline{PRobot} - \overline{PVista}}{|\overline{PRobot} - \overline{PVista}|}$$

Y el cuaternión $q_{farView}$ se obtiene a partir del vector de dirección $\overline{V}$ (figura IV.4),

$$q_{farView} = qx\ qy$$

donde:

$$qx = \cos(\theta x) + \overline{U_1}\,\mathrm{sen}(\theta x) \qquad \text{con } \overline{U_1} = i \qquad \text{y } \theta x = \arcsen yd$$

$$qy = \cos(\theta y) + \overline{U_2}\,\mathrm{sen}(\theta y) \qquad \text{con } \overline{U_2} = -j \qquad \text{y } \theta y = \arctan\left(\frac{zd}{xd}\right);\ xd > 0$$

El cuaternión qx representa la rotación de la cámara sobre el eje X y el cuaternión qy representa la rotación de la cámara sobre el eje Y.

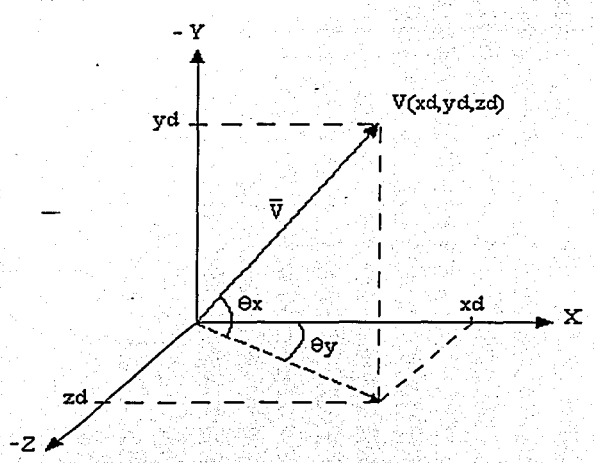


Figura IV.4 Conversión del vector de dirección a cuaternión.

4.3 PICKING

El término "picking" se refiere al proceso de selección de un objeto 3D a partir de su proyección 2D en pantalla, utilizando el puntero del mouse. La idea es construir un rayo cuyo origen es un punto de la escena proyectado en pantalla (el punto que se eligió con el mouse) y cuya dirección es hacia la escena gráfica. El rayo es transformado de su posición 2D a una posición 3D dentro de la escena gráfica y a partir de ese punto con la dirección establecida se comienzan a buscar intersecciones de los objetos y el rayo.

La operación de picking soporta varios efectos especiales, y no sólo se limita a selección de objetos 3D en pantalla. Por ejemplo, en videojuegos, el picking puede determinar si un proyectil o rayo láser disparado por el arma del jugador golpea un blanco determinado. Otros usos para el picking incluyen la determinación de la altura de los objetos sobre un terreno, permite establecer la visibilidad de objetos desde un punto de vista dado, evita colisiones con los obstáculos mientras un objeto intenta seguir una trayectoria deseada, y permite establecer estimaciones de distancia entre objetos y obstáculos.

Efectuar el picking en una escena gráfica jerárquica implica recorrer el árbol de jerarquía hasta encontrar un nodo geométrico. Los polígonos del nodo geométrico, generalmente triángulos, son probados uno a uno para ver si el rayo se interseca con ellos. Si ocurre una o varias intersecciones, éstas son reportadas en una lista y un procesamiento adicional determina cuál de todos los polígonos es el más cercano al punto de origen del rayo.

Una prueba exhaustiva de intersección entre el rayo y los polígonos puede ser muy cara en términos de tiempo de cómputo, especialmente si el rayo no se interseca con algún polígono del nodo geométrico. Para evitar esto, la estructura jerárquica de la escena puede ser explotada. La operación de picking en un nodo es propagada a los hijos del nodo sólo si el rayo se interseca con un cuerpo de frontera (bounding volume) asociado con el nodo original. Una prueba de intersección entre el rayo y el cuerpo de frontera resultará mucho más barata en términos computacionales. Si el rayo no se interseca con el cuerpo, sólo una pequeña cantidad de tiempo se requerirá para determinarlo, y no se desperdiciará demasiado tiempo en buscar dentro de ese cuerpo de frontera.

El pseudocódigo para el proceso es,

```
void EjecutaPicking ( Nodo nodo, Rayo rayo, PickResultados resultados ) {
    IF ( rayo se interseca con el cuerpo de frontera del nodo ) {
        IF ( nodo es un nodo geométrico ) {
            FOR ( Cada polígono de perteneciente a nodo ) {
                IF ( rayo se interseca con un polígono )
                    Agregar a resultados la información sobre la intersección
            }
        } ELSE {
            FOR ( Cada hijo de nodo )
                EjecutaPicking ( nodoHijo, rayo, resultados );
        }
    }
}
```

4.3.1 SELECCIÓN DE OBJETOS EN ROC2

En Roc2 el usuario es capaz de seleccionar los robots móviles desde cualquiera de las ventanas utilizando el puntero del mouse. El trabajo es realizado por medio de una tarea llamada SelectRobotUsingMouse que se ejecuta durante la simulación.

El pseudocódigo para esta tarea es el siguiente:

```
void SelectRobotUsingMouse ( ) {
    Obtener información sobre el mouse (qué botones están presionados,
    cuál es su posición en la ventana, etc.)
    IF ( El botón izquierdo del mouse fue presionado sobre ventana ) {
        Buscar el polígono más cercano al punto seleccionado sobre la pantalla (picking)
        IF ( Se encontró algún polígono ) {
```

```
                    Encontrar a qué objeto 3D pertenece el polígono
                    IF ( El polígono pertenece a un robot móvil ) {
                        Habilitar el despliegue del bounding box del robot
                        Actualizar la información acerca del robot seleccionado
                        Cambiar los parámetros de los puntos de vista (cámaras)
                        con base en el robot seleccionado
                    }
                }
            }
        }
```

WorldToolKit ofrece varios métodos para poder efectuar varias de las subtareas establecidas en el pseudocódigo anterior. Por ejemplo, las funciones WTsensor_getmiscdata( ) y WTsensor_getrawdata( ) proporcionan información acerca de los botones presionados en el mouse y sobre la posición del mouse en la pantalla. La función WTscreen_pickpoly( ) regresa un apuntador del polígono más cercano al punto de selección sobre la pantalla, y además informa del punto de intersección entre el rayo y el polígono si es que existe. Además, la función WTnode_boundingbox() habilita o deshabilita el despliegue del bounding box de un objeto.

Finalmente, la actualización de los puntos de vista se realiza por medio de las funciones de ajuste de cámaras, descritas en la sección de puntos de vista, en base a la posición y orientación del robot seleccionado.

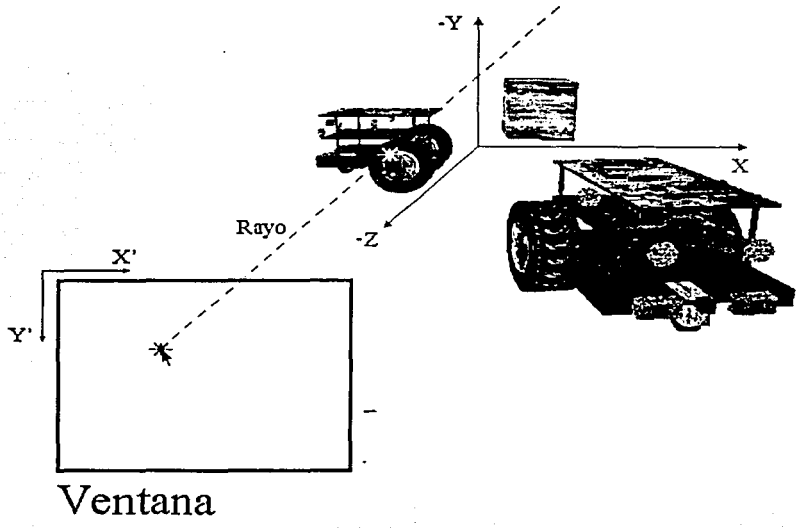


Figura IV.5 Selección de objetos.

CAPÍTULO V
OBSTÁCULOS

5.1 INTRODUCCIÓN

Todo robot o cualquier otro objeto dinámico puede ser obstruido tanto por objetos estáticos como por objetos dinámicos.[22] Algunos objetos estáticos utilizados en esta aplicación son los muros, los muebles, las cajas, entre otros; y entre los objetos dinámicos se tienen a los robots (vehículos) y otros objetos móviles. Para abreviar, se empleará el nombre 'obstáculo' cuando se haga referencia a estos objetos.

Cuando un obstáculo es construido y agregado a la escena gráfica, también es agregado a una lista de obstáculos. De esta manera, si se desea conocer con qué objeto u objetos ha chocado un objeto X en movimiento, sólo basta probar la existencia de colisión entre el objeto X y cada uno de los elementos de la lista de obstáculos.

A continuación se describe la construcción de algunos obstáculos. Se omite la descripción del objeto dinámico robot, el cual fue tratado con detalle en el capítulo III, Interfaz 3D.

5.2 OBSTÁCULO DE TIPO ÁRBOL

Este obstáculo es un objeto del tipo estático ya que no se mueve durante la ejecución de la simulación. Su nombre describe perfectamente la apariencia de este obstáculo: un árbol. La figura V.1 muestra su apariencia dentro del mundo virtual.

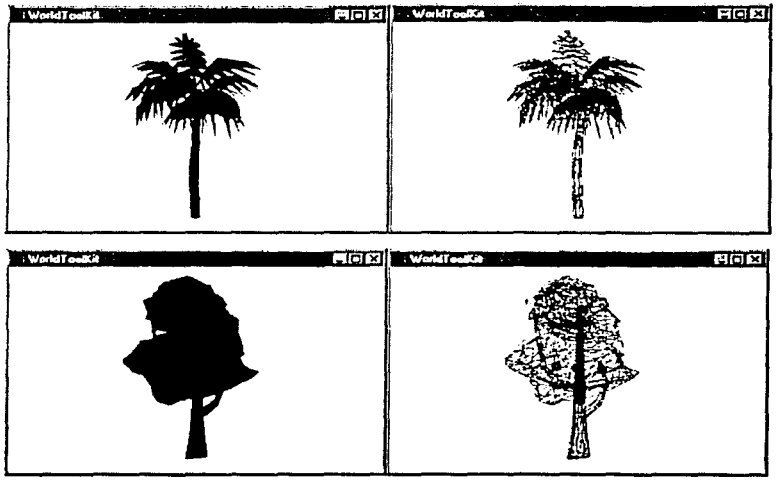


Figura V.1 Obstáculos del tipo árbol.
Modelo en colores sólidos (izquierda), modelo de alambres (derecha).

[22] Cuando se hable de un objeto dinámico se hará referencia a cualquier modelo geométrico que cambia de posición y de orientación durante la ejecución de la simulación. Y cuando se hable de objeto estático se hará referencia a un modelo geométrico que no cambia de posición, en otras palabras, que no se mueve.

5.2.1 ÁRBOL JERÁRQUICO PARA LA ENTIDAD ÁRBOL

Las siguientes figuras muestran los árboles jerárquicos para la entidad árbol.

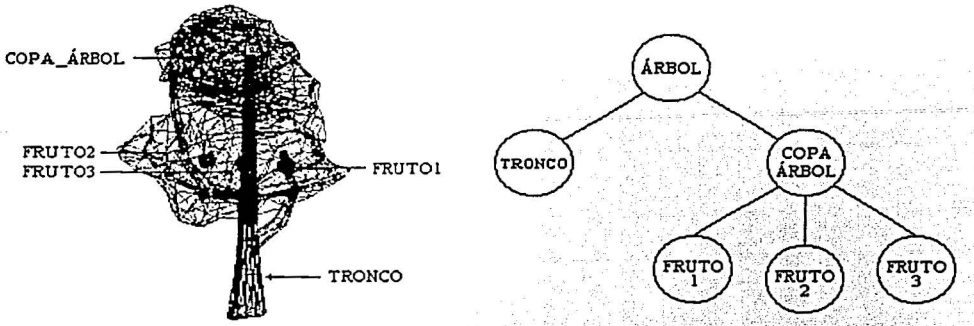


Figura V.2 Modelo geométrico[23] y árbol jerárquico para la entidad árbol.

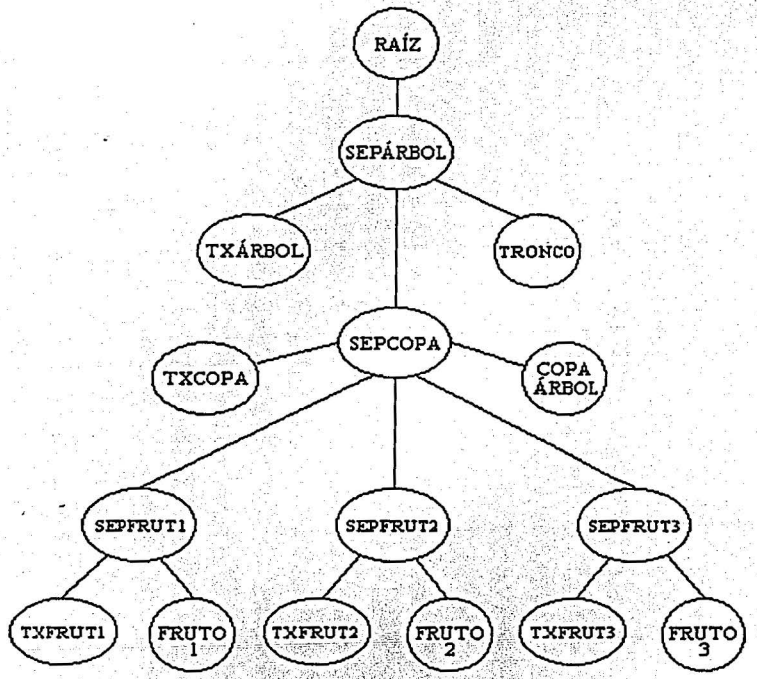


Figura V.3 Árbol jerárquico en WTK para la entidad árbol.

[23] Para mayor referencia, consulte la sección 2.3.1: modelado jerárquico.

Una escena gráfica en WTK está compuesta por una serie de elementos básicos denominados nodos. Los nodos son los bloques de construcción básicos utilizados para formar la escena gráfica. Puede pensarse en un nodo como un elemento que mantiene información sobre la jerarquía de la escena. Esta información puede ser información acerca de los elementos geométricos, las fuentes de iluminación, o acerca de las posiciones de los elementos en la escena. Los nodos también permiten agrupar otros nodos para formar geometrías más complejas y poder manejarlas como una sola entidad. Por ejemplo, revisemos el árbol jerárquico de la figura V.3.

Los nodos 'TRONCO', 'FRUTO1', 'FRUTO2', 'FRUTO3' y 'COPA ÁRBOL' son nodos del tipo geométrico que despliegan un conjunto de polígonos junto con un material asociado. Por ejemplo, el nodo 'FRUTO1' contiene todos los polígonos necesarios para representar la figura de una manzana y otros atributos como el color de cada polígono. El resto de los nodos geométricos contienen información similar para representar otras figuras.

Los nodos etiquetados con el prefijo 'TX' son nodos de transformación, y éstos almacenan información acerca de la posición y orientación de los nodos geométricos. Por último, los nodos etiquetados con el prefijo 'SEP' son nodos separadores, los cuales mantienen encapsulados los efectos de un nodo de transformación a una sola parte del árbol jerárquico. Por ejemplo, suponga que se aplica una transformación al nodo 'TXFRUT1', dicha transformación se propagará sólo hacia el nodo 'FRUTO1' y se evitarán sus efectos en el resto de los nodos. Recuerde que los nodos separadores impiden que las transformaciones salgan (se propaguen) hacia nodos superiores.

Ahora suponga que la transformación se aplica al nodo 'TXCOPA', los efectos de tal transformación se verán reflejados en los nodos 'COPA ÁRBOL', 'FRUTO1', 'FRUTO2' y 'FRUTO3', pero se evitará su propagación hacia el nodo superior 'TRONCO'. Si se aplica una transformación al nodo 'TXÁRBOL', ésta sólo afectará a los nodos que estén por debajo del nodo separador 'SEPÁRBOL' y se evitará su propagación hacia elementos del resto de la escena gráfica.

5.2.2 ANIMACIÓN

Animar es, literalmente, traer a la vida. Frecuentemente las personas piensan en la animación como sinónimo de movimiento, pero también abarca todos los cambios que tienen un efecto visual. Esto incluye variaciones de posición durante el transcurso de tiempo (movimiento dinámico), cambios de forma, color, tamaño, estructura y textura de un objeto, así como cambios en la iluminación, y en la posición, orientación y enfoque de la cámara.

Modelos de animación

Existen diversas técnicas para animar un personaje; sin embargo, las más utilizadas son las siguientes: animación por keyframes, cinemática directa, cinemática inversa y captura de movimiento.

La animación por keyframes requiere que el animador construya un personaje en varias posturas; cada postura se denomina keyframe. Cada keyframe cambia la posición y la orientación locales de los nodos de la jerarquía. Cuando la animación es muy compleja, no resulta práctico crear

todas las posturas de un personaje, pues el animador pasaría demasiado tiempo creando los keyframes y además, se necesitaría una cantidad muy grande de espacio para almacenarlos. En este caso, sólo se construyen algunos keyframes base y las posturas intermedias entre un keyframe y otro son interpoladas.

La cinemática es el estudio del movimiento sin considerar la masa del objeto o las fuerzas que actúan sobre éste. La técnica de cinemática directa especifica qué transformaciones aplicar a los objetos, y con base en ello, son calculadas las nuevas posiciones y orientaciones del objeto.

Por el contrario, la cinemática inversa especifica la posición y orientación deseada del objeto, y a partir de la posición y orientación actuales se calculan las transformaciones intermedias que dan lugar a la postura deseada. Observe que la técnica de cinemática inversa podría encontrar una o varias soluciones al problema, es responsabilidad del animador colocar algunas restricciones para evitar respuestas absurdas.

El siguiente ejemplo intenta explicar la diferencia entre cinemática directa y cinemática inversa. La descripción cinemática de una escena podría decir: Un cubo está en el origen en el tiempo t=0. Instantes después comienza a moverse con velocidad constante en la dirección dada por el vector (1, 1, 5). En el caso de la cinemática inversa, el problema sería: ¿Cuál debe ser la velocidad (constante) de un cubo para alcanzar la posición (12, 12, 60) en 5 segundos?

Por último, la técnica de captura de movimiento consiste en grabar las posturas de un personaje real para después transformarlas en keyframes. Este método provee movimientos muy reales; sin embargo, requiere de equipo costoso para captura de movimiento.

5.2.3 ANIMACIÓN DE LA ENTIDAD ÁRBOL

La finalidad de animar al objeto árbol fue agregar un detalle visual a la escena gráfica. Esta animación consiste en dejar caer los frutos del árbol cuando algún objeto colisiona con éste.

El proceso se da de la siguiente manera: un objeto móvil detecta que ha chocado con algún objeto de la lista de obstáculos. Si el objeto con el chocó es un objeto del tipo árbol, entonces se activa una tarea que anima los frutos del árbol, dando la impresión que caen debido al impacto.

Antes de continuar, recordemos que una tarea en WTK es un procedimiento que asigna comportamientos a objetos individuales. Estos comportamientos pueden especificar el movimiento de un objeto, cambios en su apariencia, pueden probar intersecciones con otros objetos, activar otros comportamientos, etc.

En particular, la tarea encargada de la animación de los frutos sólo especifica cómo deben moverse los frutos para alcanzar la base del árbol, es decir, proporciona las posiciones de los frutos para cada cuadro de la animación hasta llegar a la base del árbol (véase figura V.4).

Figura V.4 Animación de los frutos del árbol.

Si nuevamente se choca contra el árbol, un segundo fruto caerá. Si por tercera ocasión se choca contra el árbol, el tercer y último fruto caerá. Y la tarea encargada de la animación de los frutos será eliminada de la simulación pues no tiene sentido que ocupe memoria si no hay frutos que puedan caer.

Figura V.5 Fin de la animación del árbol.

Como habrá notado, para animar la caída de cada fruto se aplicó la técnica de cinemática directa porque fue el método que mejor se adaptó al tipo de animación requerido.

5.3 Obstáculo de tipo bomba

Este obstáculo es del tipo dinámico pues se encuentra en movimiento durante la ejecución de la aplicación. Su apariencia dentro de la simulación es la siguiente (figura V.6).

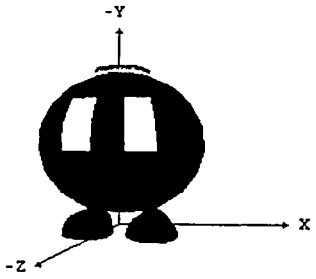


Figura V.6 Obstáculo de tipo bomba.

5.3.1 ÁRBOL JERÁRQUICO PARA LA ENTIDAD BOMBA

La figura que a continuación se presenta, muestra los elementos geométricos que componen a la entidad bomba y su árbol jerárquico básico (figura V.7).

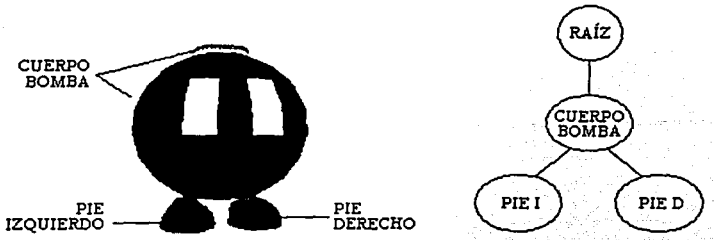


Figura V.7 Modelo geométrico y árbol jerárquico para la entidad bomba.

La disposición de los nodos para el árbol jerárquico de WTK es el siguiente (figura V.8).

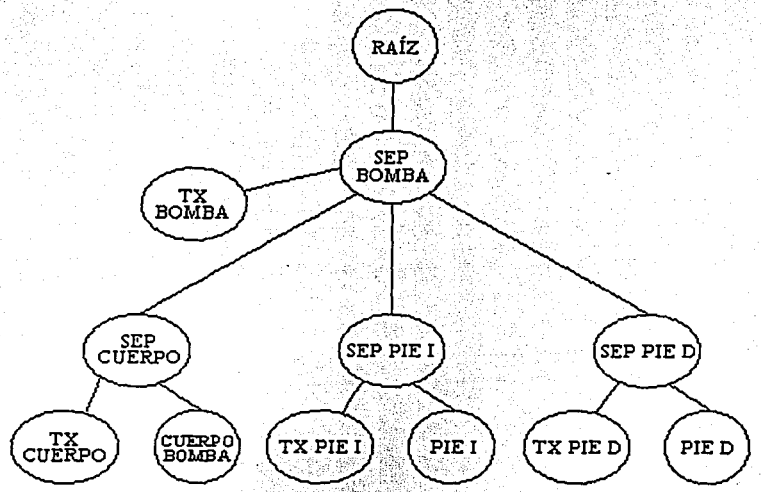


Figura V.8 Árbol jerárquico en WTK para la entidad bomba.

Los nodos 'Cuerpo bomba', 'Pie I' y 'Pie D' son nodos geométricos, estos nodos contienen los polígonos y los materiales que forman a cada geometría. Por ejemplo, el nodo 'Pie I' contiene los polígonos necesarios para construir la figura de un zapato, así como otros atributos tales como el color de cada polígono. El resto de los nodos geométricos contienen la información suficiente para construir la figura asociada.

Los nodos etiquetados con el prefijo 'TX' son nodos de transformación que mantienen información acerca de la posición y orientación de los nodos geométricos. Los nodos etiquetados con el prefijo 'SEP' son nodos separadores que encapsulan los efectos de los nodos de transformación a una sola parte del árbol jerárquico. Por ejemplo, aplicar una transformación al nodo 'TX PIE I' sólo afectará al nodo geométrico 'Pie I', mientras que aplicar una transformación al nodo 'TX BOMBA' afectará a toda la entidad bomba sin afectar al resto de los elementos de la escena gráfica.

5.3.2 TAREAS ASOCIADAS A LA ENTIDAD BOMBA

Existen varias tareas asociadas al comportamiento de la entidad bomba. La primera de ellas se encarga de la detección de colisiones con el resto de los objetos de la lista de obstáculos; la segunda es una animación de los zapatos de la bomba que da la apariencia de que la bomba camina; y la última tarea mueve a la entidad bomba a lo largo de una trayectoria bien definida.

5.3.2.1 Detección de colisiones

Esta tarea se dedica a probar colisiones entre el objeto bomba y el resto de los objetos de la lista de obstáculos. Su funcionamiento es el siguiente:

Primero se calculan los límites espaciales del objeto bomba, es decir, se obtiene un prisma que represente la máxima extensión espacial del objeto bomba en su posición y orientación actuales, a este prisma se le denomina "bounding box" (caja envolvente o caja de frontera). Para cada objeto de la lista de obstáculos también se calcula su caja envolvente y se prueban intersecciones entre la caja envolvente del objeto bomba y la caja envolvente de cada obstáculo, según el algoritmo descrito en el capítulo III. Recuerde que en la lista de obstáculos también se encuentra el objeto bomba, por ello, al revisar intersecciones se debe evitar probar colisión consigo mismo.

El objeto bomba es un obstáculo del tipo dinámico, es decir, está en movimiento. La trayectoria que describe su movimiento está dada por una serie de ecuaciones matemáticas; gracias a estas ecuaciones es posible conocer la posición del objeto todo el tiempo. Estas ecuaciones se tratarán posteriormente.

Si ocurre una colisión, el objeto bomba regresa al punto inmediato anterior, esto es, la posición del objeto antes de detectar la colisión. Este punto es fácil de encontrar ya que se conocen las ecuaciones que describen su movimiento. A continuación, se revisa si el obstáculo con el que colisionó la bomba es un obstáculo del tipo árbol. Si es así, la tarea de animación del árbol es activada, y de esta manera comienzan a caer los frutos del árbol debido al impacto.

Independientemente del tipo de obstáculo con el que se colisionó, el objeto bomba da media vuelta y comienza a recorrer la misma trayectoria desde la posición actual pero en sentido opuesto. Finalmente, la tarea verifica el estado de colisión de la bomba, es decir, prueba si la bomba ha chocado demasiadas veces. De ser cierto, se activa una tarea de animación que produce la explosión del objeto bomba, se deshabilita el objeto bomba de la escena gráfica (con el fin de que no sea dibujado en escena), se saca el objeto bomba de la lista de obstáculos para evitar que otras tareas

detecten colisiones con un objeto no existente y se desactiva la tarea de colisiones de la bomba para evitar su ejecución. Si el estado de colisión de la bomba es aceptable, la tarea de detección de colisiones continúa ejecutándose hasta que la bomba explote.

El pseudocódigo de esta tarea se muestra a continuación.

```
void collitionTask( BombObstacle Bomba ) {
    Calcular la caja envolvente (bounding box) para el objeto bomba

    FOR ( Cada objeto en la lista de obstáculos ) {
        IF ( El obstáculo actual no es un objeto bomba ) {
            Calcular la caja envolvente para el obstáculo actual
            IF ( La caja envolvente del objeto bomba se interseca con la caja
            envolvente del obstáculo actual ) {
                Obtener la posición del objeto bomba un instante antes de que se
                detectara la colisión y mover al objeto bomba hacia esa posición

                IF ( El obstáculo actual es del tipo árbol ) {
                    Activar animación del árbol (caída de los frutos)
                }
                Rotar bomba π radianes sobre el eje Y
                Invertir el sentido de recorrido sobre la trayectoria del objeto bomba

                IF ( Bomba ha chocado demasiado ) {
                    Activar animación de explosión (la bomba explota)
                    Deshabilitar el objeto bomba de la escena gráfica. También se
                    deshabilitan todas las tareas asignadas al objeto, incluyendo
                    la de detección de colisiones del objeto bomba
                    Borrar al objeto bomba de la lista de obstáculos
                }
            }
        }
    }
}
```

Como habrá observado, esta tarea no sólo prueba colisiones entre objetos, también activa o desactiva otras tareas dentro de la simulación como son la animación de los frutos del objeto árbol y la animación de la explosión de la bomba.

5.3.2.2 Cálculo de posiciones y orientaciones

Esta tarea se encarga de calcular la posición y orientación del objeto bomba a lo largo de una trayectoria bien definida. Dicha trayectoria está dada por la ecuación de Lissajous en su forma paramétrica.

$$x = a \text{ sen } (w_1 t)$$
$$z = b \cos (w_2 t)$$

Donde t es un parámetro que varía de 0 a 2π radianes, y a, b, w_1 y w_2 son constantes. Al evaluar estas ecuaciones para distintos valores de t obtendremos los puntos que describen la trayectoria; dichos puntos representan la posición de la bomba sobre la trayectoria. Por otra parte, modificar los valores de las constantes a, b, w_1 y w_2, producirá diferentes tipos de trayectorias como los que se muestran en la figura V.9.

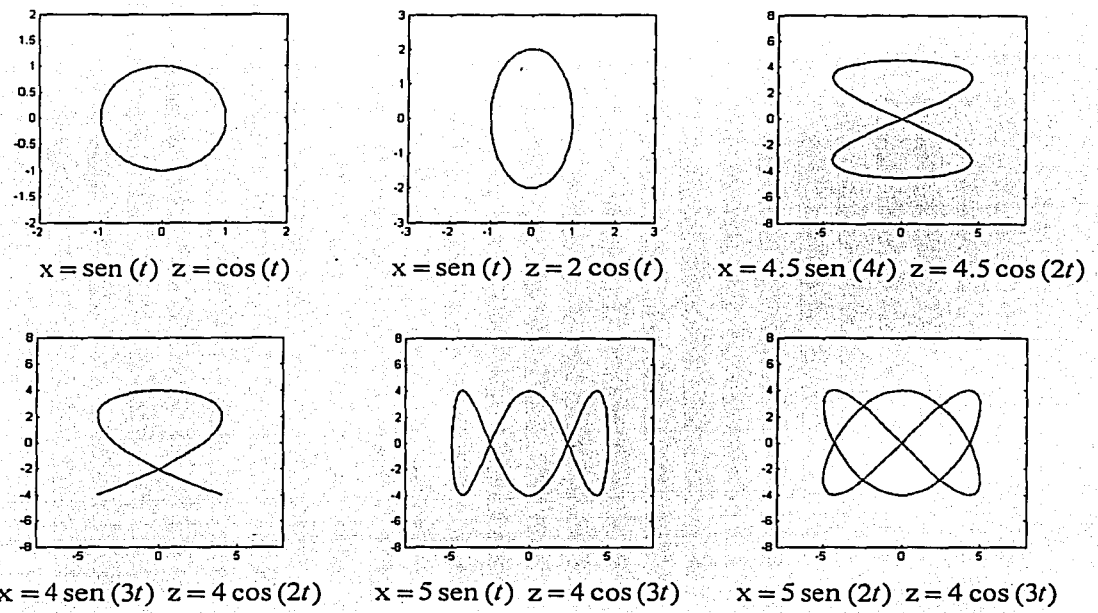


Figura V.9 Las ecuaciones de Lissajous permiten describir trayectorias con diversas formas.

Como se mencionó anteriormente, la construcción de la trayectoria de la bomba se obtiene evaluando las ecuaciones de Lissajous para valores de t entre 0 y 2π radianes. El espaciamiento entre dos valores consecutivos de t determinará la precisión con la que la trayectoria será construida. Es decir, si el espaciamiento entre dos valores consecutivos de t es muy grande, la reconstrucción de la trayectoria será poco precisa y el movimiento de la bomba se vería irreal. Por el contrario, si el espaciamiento es exageradamente pequeño, la reconstrucción de la trayectoria será muy precisa pero el desplazamiento de la bomba se verá muy lento. Así que la búsqueda de un valor para el espaciamiento de t no es cosa que se deba menospreciar.

Adicionalmente, las curvas pueden presentar una rotación entre 0 y 2π radianes alrededor del eje Y, y un desplazamiento de x_o unidades sobre el eje X y z_o unidades sobre Z.

La rotación adicional sobre el eje Y estará dada por el siguiente cuaternión unitario:

$$Q_{\text{rotación adicional}} = \cos\left(\frac{\text{ánguloAdicional}}{2}\right) + \bar{U}\,\text{sen}\left(\frac{\text{ánguloAdicional}}{2}\right)$$

donde: $\bar{U} = (0, -1, 0)$

Mientras que el desplazamiento adicional estará dado por el vector:

$$\overline{P_{\text{desplazamiento adicional}}} = (x_o, 0, z_o)$$

Si la rotación y/o el desplazamiento adicionales son diferentes de cero, la posición y orientación de la bomba se verán afectadas por estas transformaciones extras. Por lo tanto, el vector de posición P de la bomba deberá rotarse con el valor de $Q_{\text{rotación adicional}}$, y desplazarse x_o y z_o unidades sobre los ejes X y Z, respectivamente.

Entonces, el nuevo vector de posición para la bomba, $\bar{P}_{\text{final}}$, se calcula de la siguiente manera:

$$\overline{P_{\text{final}}} = (Q_{\text{rotación adicional}}\ \bar{P}\ Q_{\text{rotación adicional}}^{*}) + \overline{P_{\text{desplazamiento adicional}}}$$

donde: $Q_{\text{rotación adicional}}^{*}$ es el conjugado de $Q_{\text{rotación adicional}}$
El producto $Q_{\text{rotación adicional}}\ \bar{P}\ Q_{\text{rotación adicional}}^{*}$ es el vector $\bar{P}$ rotado por el cuaternión $Q_{\text{rotación adicional}}$.

Recuerde que un vector de posición en el espacio también puede representarse utilizando un cuaternión, por ejemplo, el siguiente cuaternión representa al vector $\bar{P} = (x, y, z)$.

$$q_P = 0 + x_P i + y_P j + z_P k$$

De esta manera, ya es posible efectuar el producto $Q_{\text{rotación adicional}}\ \bar{P}\ Q_{\text{rotación adicional}}^{*}$.[24]

Respecto a la orientación de la bomba, ésta se obtiene calculando el vector tangente a un punto sobre la trayectoria (este punto es la posición actual de la bomba). El vector tangente en el instante *t* está dado por,

$$\overline{V_{\text{tangente}}} = \left(\frac{d\,x(t)}{dt}, 0, \frac{d\,z(t)}{dt}\right)$$

[24] Para mayor información acerca de la representación de rotaciones usando cuaterniones, consulte la siguiente fuente: 3D game engine design: A practical approach to real-time computer graphics; autor, David Eberly.

donde:

$$\frac{d\,x(t)}{dt} = a\cos(w1\,t)$$

$$\frac{d\,z(t)}{dt} = -b\,\mathrm{sen}(w2\,t)$$

Si la rotación adicional de la curva es diferente de cero, el vector tangente también se verá afectado y deberá ser rotado con el valor de $Q_{\text{rotación adicional}}$. Así, el nuevo vector tangente se obtiene multiplicando:

$$\overline{V_{\text{tangente final}}} = Q_{\text{rotación adicional}}\ \overline{V_{\text{tangente}}}\ Q_{\text{rotación adicional}}^{*}$$

El pseudocódigo de esta tarea se muestra a continuación.

```
void calculateBombPositionOrientation( float T ) {
    IF ( No existe un desplazamiento ni una rotación adicionales ) {
        // Obtener la posición de la bomba
        Calcular la posición de la bomba en el instante T, utilizando
        las ecuaciones de Lissajous
        // Obtener la orientación de la bomba
        Calcular el vector tangente en el instante T, utilizando la primera derivada
        de las ecuaciones de Lissajous y el sentido de recorrido de la trayectoria
        Construir el cuaternión Qvector utilizando la orientación del vector tangente
    } ELSE {
        // Obtener la posición de la bomba
        Calcular la posición de la bomba en el instante T, utilizando
        las ecuaciones de Lissajous
        Aplicar la rotación adicional al vector de posición anterior
        Sumar al vector de posición rotado el desplazamiento adicional
        // Calcular la orientación de la bomba
        Calcular el vector tangente en el instante T, utilizando la primera derivada
        de las ecuaciones de Lissajous y el sentido de recorrido de la trayectoria
        Aplicar la rotación adicional al vector tangente
        Construir el cuaternión Qvector utilizando la orientación del vector tangente rotado
    }

    Trasladar el objeto bomba a la posición calculada
    Orientar al objeto bomba con la rotación dada por el cuaternión Qvector
}
```

5.3.2.3 Animación de los zapatos de la bomba

Esta tarea se dedica a rotar los zapatos y el cuerpo de la bomba, de tal manera que parezca que el objeto en realidad está caminando. Para construir esta animación se utilizó la técnica de

keyframes, y por lo tanto, fue necesario construir cada postura de la bomba para definir la orientación que los pies debían tener con respecto al cuerpo.

Los keyframes son los siguientes. Tenga en cuenta que las rotaciones descritas son acumulativas, es decir, la nueva rotación se suma a las rotaciones anteriores (véanse las figuras V.10 y V.11).

Keyframes para el movimiento del pie derecho

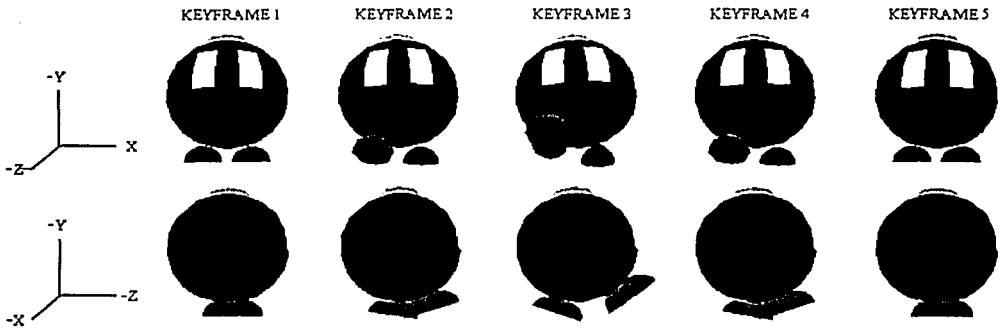


Figura V.10 Keyframes para el movimiento del pie derecho de la bomba.

- **Keyframe 1**. Postura neutral. Ni los pies ni el cuerpo de la bomba presentan rotaciones.
- **Keyframe 2**. Gira el pie derecho hacia adelante y el pie izquierdo hacia atrás. El pie derecho rota θ radianes sobre el eje X en sentido positivo y el pie izquierdo ϕ radianes sobre X en sentido negativo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido positivo.
- **Keyframe 3**. Gira el pie derecho hacia adelante y el pie izquierdo hacia atrás. El pie derecho rota θ radianes sobre el eje X en sentido positivo y el pie izquierdo ϕ radianes sobre X en sentido negativo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido positivo.
- **Keyframe 4**. Gira el pie derecho hacia atrás y el pie izquierdo hacia adelante. El pie derecho rota θ radianes sobre el eje X en sentido negativo y el pie izquierdo ϕ radianes sobre X en sentido positivo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido negativo.
- **Keyframe 5**. Gira el pie derecho hacia atrás y el pie izquierdo hacia adelante. El pie derecho rota θ radianes sobre el eje X en sentido negativo y el pie izquierdo ϕ radianes sobre X en sentido positivo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido negativo. Se regresa a la postura neutral.

Keyframes para el movimiento del pie izquierdo

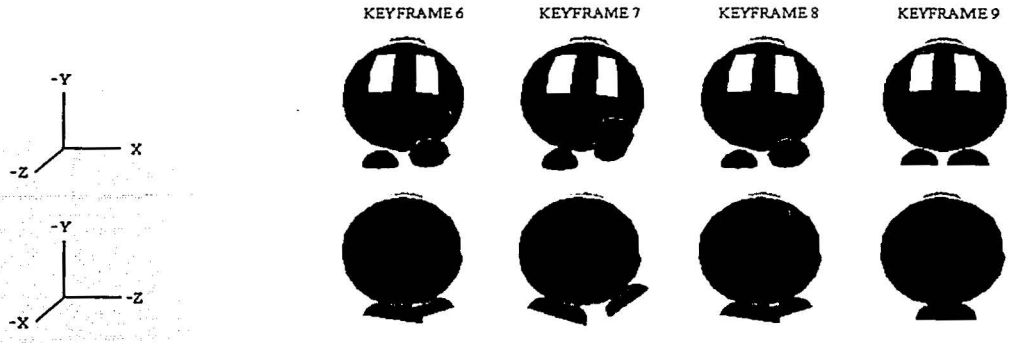


Figura V.11 Keyframes para el movimiento del pie izquierdo de la bomba.

- **Keyframe 6.** Gira el pie izquierdo hacia adelante y el pie derecho hacia atrás. El pie izquierdo rota θ radianes sobre el eje X en sentido positivo y el pie derecho ϕ radianes sobre X en sentido negativo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido negativo.
- **Keyframe 7.** Gira el pie izquierdo hacia adelante y el pie derecho hacia atrás El pie izquierdo rota θ radianes sobre el eje X en sentido positivo y el pie derecho ϕ radianes sobre X en sentido negativo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido negativo.
- **Keyframe 8.** Gira el pie izquierdo hacia atrás y el pie derecho hacia adelante. El pie izquierdo rota θ radianes sobre el eje X en sentido negativo y el pie derecho ϕ radianes sobre X en sentido positivo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido positivo.
- **Keyframe 9.** Gira el pie izquierdo hacia atrás y el pie derecho hacia adelante. El pie izquierdo rota θ radianes sobre el eje X en sentido negativo y el pie derecho ϕ radianes sobre X en sentido positivo. El cuerpo de la bomba rota α radianes sobre el eje Z en sentido positivo. Se regresa a la postura neutral.

Como puede observar, los keyframes establecen las rotaciones que los pies y el cuerpo deben tener para obtener la postura deseada. En conclusión, la tarea encargada de esta animación aplica cada una de estas rotaciones sobre la porción indicada del cuerpo de la bomba.

En particular, para nuestra simulación, los ángulos de rotación utilizados son los siguientes:

$\theta = 20° \approx 0.349$ radianes
$\phi = 10° \approx 0.1745$ radianes
$\alpha = 2° \approx 0.0349$ radianes

Las rotaciones correspondientes al pie derecho se aplican al nodo de transformación 'TX PIE D', las del pie izquierdo al nodo 'TX PIE I', y las del cuerpo al nodo de transformación 'TX CUERPO'. En contraste, las transformaciones obtenidas en la tarea 'calculateBombPositionOrientation' son aplicadas al nodo de transformación 'TX BOMBA', ya que aplicar cualquier transformación sobre este nodo afecta a todo el objeto bomba.

5.3.2.4 Animación de la explosión

Existe una tarea que es habilitada cuando una bomba ha colisionado demasiadas veces; esta tarea es la animación de una explosión. La tarea de la explosión utiliza un arreglo de tres polígonos con la disposición mostrada en la figura V.12. Sobre cada uno de estos polígonos se mapea una serie de imágenes, las cuales desplegadas en una secuencia continua simulan una explosión.

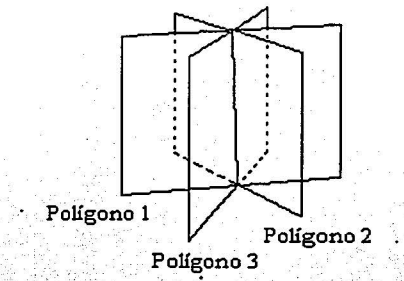


Figura V.12 Disposición de los polígonos en la animación de la explosión.

El árbol jerárquico para este modelo geométrico se presenta en la figura V.13.

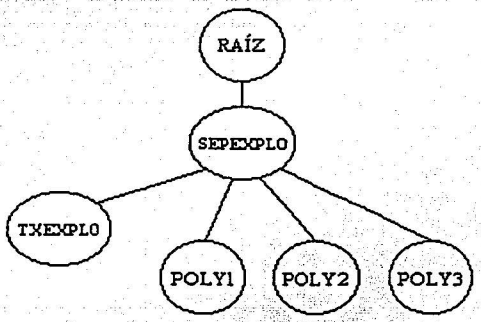


Figura V.13 Árbol jerárquico para el arreglo de polígonos.

Observe que este árbol jerárquico contiene un nodo de transformación que nos permite desplazar el arreglo de polígonos a la posición en donde se encuentra el objeto que explotará. Recuerde que el nodo separador 'SEPEXPLO' impide que las transformaciones aplicadas a 'TXEXPLO' se propaguen al resto de los elementos geométricos de la escena.

La función que ejecuta la tarea de explosión es muy sencilla. Primero, se mueve el arreglo de polígonos al punto en donde se localiza la bomba que va a explotar. Enseguida, se comienzan a desplegar las 17 imágenes que componen la animación de la explosión sobre cada uno de los polígonos. Una vez que las todas imágenes han sido desplegadas sobre cada polígono, se desactiva la tarea de la explosión.

CAPÍTULO VI
ROBEL

6.1 INTRODUCCIÓN

A través de la interfaz 3D es posible asignar instrucciones a un robot móvil, que son especificadas por el usuario. También es posible hacer que un robot ejecute un script (guión) con base en dichas instrucciones.

Los scripts cuentan con un vocabulario, un conjunto de reglas gramaticales para indicar los movimientos básicos de un robot, y reglas especiales para indicar las instrucciones. Debido a estas características, denominamos a esto un lenguaje de programación para controlar robots móviles, y su nombre es Robel.

Robel es un lenguaje de programación orientado al control de un robot móvil. Desde un principio se planeó como una herramienta para llevar a cabo el control de forma rápida y sencilla haciendo uso de instrucciones.

Sus características principales son:

- Instrucciones para el control de movimiento
- Instrucciones para la lectura del estado lógico del robot
- Instrucciones para comunicación vía Internet
- Control de flujo de programa haciendo uso de ciclos while, if y saltos a etiquetas
- Soporta los siguientes tipos de variables: enteros y flotantes
- Independencia de la plataforma
- Posibilidad de tener recursividad
- Llamadas a funciones creadas por el usuario (otros scripts)

Para que un programa escrito en Robel se pueda ejecutar sobre distintas plataformas, es necesario hacer uso de un programa que traduzca cada instrucción de Robel al lenguaje propio de cada sistema, es decir, se debe contar con la Máquina Virtual de Robel (MVR).

6.2 MÁQUINA VIRTUAL DE ROBEL

Las instrucciones deben poderse ejecutar de manera transparente, independientemente del diseño del robot. En general, los robots tienen distinta arquitectura y forma de operar, por lo que es necesario contar con un programa cuya tarea sea interpretar las instrucciones y ejecutarlas de manera particular a la arquitectura, es decir, se debe contar con una máquina virtual.

El modelo de la MVR se describe haciendo uso del diagrama de bloques de la figura VI.1. Como puede apreciarse, la máquina virtual consta de estados muy sencillos que básicamente se pueden dividir en dos: cargar un script en memoria y ejecución de un script.

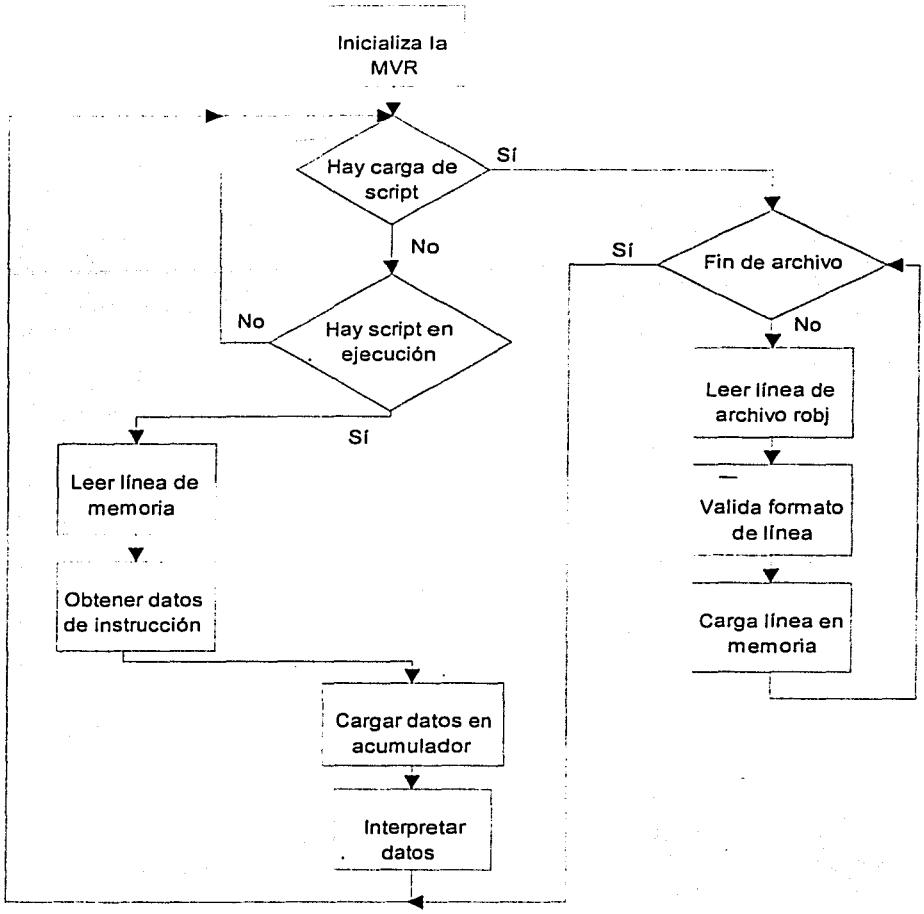


Figura VI.1 Diagrama de bloques de la MVR.

1. Carga de script en memoria. Antes de ejecutar un script es necesario cargarlo en memoria, para lo cual se debe de abrir el archivo objeto y hacer uso de un programa cargador. El programa cargador lee línea a línea el contenido del archivo objeto, valida el contenido contra errores en su construcción o alteraciones indeseables y después almacena el contenido de la línea directamente a memoria para su posterior ejecución.

2. Ejecución de script. El proceso de ejecución de script consta de distintos pasos que lo llevan a interpretar correctamente cada una de las líneas del programa previamente cargadas. Los pasos son los siguientes:

a) Leer línea de memoria

b) Obtención de datos de instrucción. Los datos contenidos dentro de la instrucción, es decir, números enteros, flotantes o uso de variables, son extraídos de la memoria para ser utilizados posteriormente.

c) Cargar datos en acumulador. Los datos extraídos son guardados en un acumulador con el fin de compartir la información durante la interpretación y ejecución de la instrucción.

d) Interpretar datos. Cada línea de programa de Robel es interpretada, clasificándola en tres tipos para su posterior manejo. Los tipos de línea son los siguientes:

1. Sistema. Instrucciones que modifican el flujo del programa.
2. Operación. Operaciones algebraicas haciendo uso de variables y/o números.
3. Instrucción. Instrucciones propias de Robel.

De esta forma, Robel es un lenguaje que para poder ser ejecutado es necesario interpretarlo; cada archivo fuente de Robel debe ser compilado para generar un archivo intermedio, archivo con extensión robj, el cual puede ser cargado e interpretado por la MVR. Actualmente se han creado 3 máquinas virtuales de Robel para ejecutar instrucciones sobre diferentes tipos de robots móviles.

6.2.1 ACUMULADOR

El acumulador forma parte de la MVR; su propósito es guardar datos para el manejo interno de la ejecución de instrucciones, así como también, almacenar el resultado de la ejecución de algunas instrucciones. Es decir, el acumulador es una estructura de datos utilizada para compartir información a través de cada una de las etapas de ejecución de una instrucción. La estructura del acumulador se muestra en el cuadro VI.1.

Registro	*Tipo*	*Nombre*	*Mnemónico en Robel*
F1	float	registro flotante 1	RegF1
F2	float	registro flotante 2	RegF2
F3	float	registro flotante 3	RegF3
I1	integer	registro entero 1	RegI1
I2	integer	registro entero 2	RegI2
S	string	registro string[25]	RegS
Stack[26]	pila de flotantes	pila	RegSp

Cuadro VI.1 Registros de la máquina virtual.

[25] *String*. Para este caso, su traducción al español es cadena.
[26] *Stack*. Su traducción al español es pila.

Para el manejo del acumulador, se crearon un conjunto de funciones que nos permiten obtener el valor de los registros y de la pila. Debido a que el acumulador es usado por el sistema de forma interna, no se tienen funciones para modificar su contenido, excepto rutinas push (introducir un dato a la pila) y pop (sacar un dato de la pila) para el manejo de la pila.

El uso del acumulador proporciona una manera sencilla de acceder a la información recabada por el proceso de análisis de la instrucción y durante cualquier momento del proceso de ejecución.

6.3 CONJUNTO DE INSTRUCCIONES DE ROBEL

Haciendo uso de Robel y de la interfaz 3D, el usuario puede controlar robots virtuales y reales de forma sencilla y rápida, por lo que se propone el uso de instrucciones de alto nivel para interactuar con los robots.

Considerando que las instrucciones deben poderse ejecutar sobre distintas plataformas[27], para mantener la mayor compatibilidad posible, entonces se hace necesario un conjunto de instrucciones básico, confiable e independiente de la arquitectura del robot.

El conjunto de instrucciones propuesto permite controlar el movimiento del robot, obtener el estado lógico en el que se encuentra, ejecutar scripts basados en las instrucciones, control del flujo del script, así como también, entablar, de ser posible, comunicación con una computadora usando Internet.

Las instrucciones de Robel pueden ser ejecutadas por el sistema a través de dos formas distintas: un script o introduciéndolas en línea durante la ejecución de la interfaz 3D. Cuando se desea utilizar un script, éste es compilado por el sistema y luego es cargado en memoria; para las entradas en línea, éstas sólo se interpretan. Para ambos casos las instrucciones deben tener el formato indicado en el cuadro VI.2. Los parámetros entre corchetes son opcionales.

Formato para instrucción
[<nombre>] instrucción [parámetros] @n

Cuadro VI.2 Formato de instrucción para Robel.

El formato de instrucción de Robel se divide en 4 partes que se describen a continuación:

- <nombre>: este parámetro es utilizado para indicar el nombre del robot que ejecutará la instrucción. Algunas instrucciones son independientes de la existencia de un robot y en esos casos está por demás
- instrucción: instrucción a ser ejecutada

[27] Entendiendo por plataforma diferentes configuraciones de robots.

- parámetros: parámetros asociados a la instrucción

- @n: código de control; para llevar un control de las instrucciones asignadas, ya sea en scripts o vía Internet, es necesario añadir un código de control denotado por @ y seguido por un identificador numérico (n) preferiblemente progresivo. Esto es con la finalidad de conocer qué instrucción se está ejecutando y en el caso de ser un comando remoto, enviar el código de la instrucción al host (anfitrión, en este contexto se refiere a la máquina que envió el mensaje) como mensaje de fin de ejecución

El cuadro VI.3 muestra el conjunto completo de instrucciones pertenecientes a Robel.

Robel también acepta comentarios; para indicar que una línea es un comentario ésta debe de contener al inicio un asterisco. Por ejemplo: *Este es un comentario.

Instrucciones	
mv	shs
mvto	ic
script	sc
loads	putbot
opsk	while
ksk	wend
left	if
right	endif
forward	ascript
backward	tap

Cuadro VI.3 Conjunto de instrucciones de Robel.

6.3.1 INSTRUCCIONES PARA EL CONTROL DE MOVIMIENTO

La parte más importante del conjunto de instrucciones es el control del movimiento del robot.

Los movimientos más sencillos son instrucciones de avanzar, retroceder, girar a la derecha y girar a la izquierda; combinando estos movimientos simples es posible crear un nuevo tipo de movimiento más complicado haciendo uso de coordenadas polares. Utilizando este último tipo de movimiento y suponiendo que conocemos la posición actual del robot, es posible realizar un movimiento que traslade a un robot desde el sitio donde se encuentra hasta un punto en específico. El cuadro VI.4 muestra el conjunto de instrucciones de movimiento.

Para realizar cualquier movimiento, es necesario establecer qué unidades se utilizarán; se seleccionó al decímetro como unidad fundamental para distancias, y radianes para los giros.

La razón por la cual se seleccionaron los decímetros como unidad fundamental tiene que ver con la compatibilidad que se buscaba con otros robots de diferentes tamaños y configuraciones, ya que algunos de ellos no pueden realizar movimientos muy cortos. La razón por la cual se seleccionaron radianes como unidad fundamental de giro es debido a su fácil manejo en los métodos computacionales.

Nivel	*Instrucciones*	*Parámetros*
básico	forward	Ninguno
básico	backward	Ninguno
básico	left	Ninguno
básico	right	Ninguno
medio	mv d g t	(d) distancia, (g) ángulo de giro y (t) tiempo
alto	mvto x y t	Posición en (x ,y) y (t) tiempo

Cuadro VI.4 Conjunto de instrucciones de movimiento.
El tiempo t es opcional para todas las instrucciones.

Instrucciones básicas

Siendo las instrucciones más sencillas, sólo es posible realizar movimientos de avance, retroceso y giros. La distancia a avanzar se ha fijado a una unidad fundamental, es decir, a un decímetro. El ángulo de giro en una instrucción left o right también es fijo y es de 0.1745 radianes (aproximadamente 10°).

Modo de empleo:
left
right
forward
backward

Instrucciones de nivel medio

Otra forma de controlar robots móviles es hacer que éstos sigan líneas rectas.

Básicamente, para seguir una línea recta, el robot necesita una forma de calcular hacia donde se encuentra orientada la siguiente línea recta. Los movimientos del robot están basados en giros y desplazamientos, por lo que conviene emplear coordenadas polares.

La forma más sencilla de calcular la orientación de la recta siguiente es realizando transformaciones geométricas, es decir, calcular la rotación necesaria para que el ángulo de la recta siguiente tenga la orientación necesaria para llegar al punto deseado.

La figura VI.2 muestra el movimiento de un robot haciendo uso de líneas rectas.

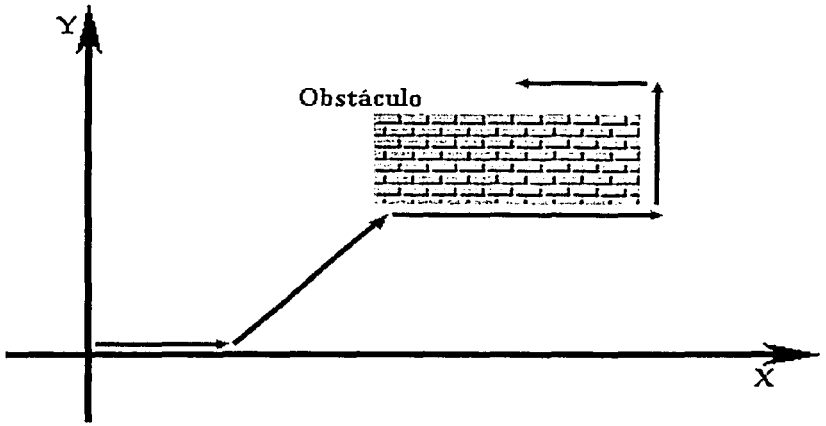


Figura VI.2 Movimiento del robot usando líneas rectas.

La posición del robot en el tiempo i (t_i) es descrita por (x_i, y_i, θ_i), donde θ_i representa el ángulo del robot con respecto al eje X. El sistema de coordenadas x', y' representa los nuevos ejes coordenados después de la rotación y desplazamiento del robot.

La figura VI.3 muestra el sistema de coordenadas del cual se obtienen las siguientes ecuaciones.

$$x_i = \overline{OA} = x_{i-1} + r\cos(\theta + \theta') = x_{i-1} + r\cos(\theta)\cos(\theta') - r\sin(\theta)\sin(\theta) \quad (1)$$

$$y_i = \overline{AP} = y_{i-1} + r\sin(\theta + \theta') = y_{i-1} + r\sin(\theta)\cos(\theta') + r\cos(\theta)\sin(\theta') \quad (2)$$

$$x_i' = \overline{O'A'} = r\cos(\theta') \quad (3)$$

$$y_i' = \overline{A'P} = r\sin(\theta') \quad (4)$$

$$x_i = x_i'\cos(\theta) - y_i'\sin(\theta) + x_{i-1} \quad (5)$$

$$y_i = x_i'\sin(\theta) + y_i'\cos(\theta) + y_{i-1} \quad (6)$$

$$\theta_i = \theta_i' + \theta_{i-1} \quad (7)$$

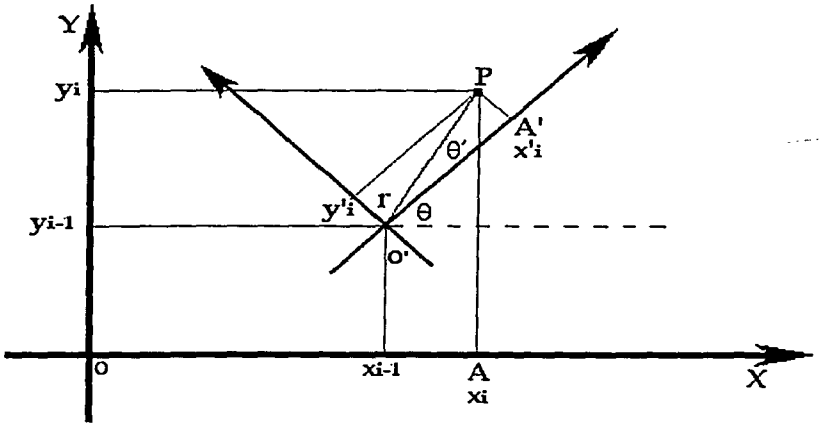


Figura VI.3 Sistema coordenado.

Donde x_i' y y_i' representan el desplazamiento del robot y θ' el ángulo de rotación en los nuevos ejes.

Si $y_i'=0$, el robot rota θ_i grados y avanza una distancia $x_i = d_i$, por lo que las nuevas ecuaciones de posición del robot son:

$$x_i = d_i \cos(\theta_i) + x_{i-1} \quad (8)$$

$$y_i = d_i \sin(\theta_i) + y_{i-1} \quad (9)$$

La forma de implantar el movimiento de un robot móvil dentro de la interfaz 3D es calculando los puntos sobre los cuales se desplazará el objeto dentro de una recta. El cálculo consta de un conjunto de ecuaciones que deben ser resueltas en cada uno de los pasos del proceso de simulación, por lo que es crítico el que sea rápida de resolver.

Un esquema que puede ser utilizado para calcular cada paso es el uso de incrementos constantes, por lo que el siguiente paso se calcularía realizando una suma; ver ecuación 10.

$$x_{i+1} = x_i + paso \quad (10)$$

El esquema anterior tiene ciertos problemas al utilizarse; al fijar el tamaño del paso a cierta cantidad no hay garantía que la suma de todos los pasos efectuados sea el punto final deseado. Una mejor aproximación es dividir la distancia a avanzar en un número de pasos definido, con esto garantizamos que se alcance el punto final; ver ecuación 11.

paso = distancia / Número_de_pasos

$$x_{i+1} = x_i + paso \quad (11)$$

Usar el esquema anterior ocasionaría que el movimiento del robot fuera demasiado lento en distancias cortas y demasiado rápido en distancias largas, ya que no importando la distancia a recorrer, el número de pasos es el mismo.

Otro tipo de problema originado por usar un número de pasos fijo se debe a que la determinación de las colisiones está basada en la posición actual del robot, un paso demasiado grande podría originar que el robot "atravesara" objetos y paredes.

La figura VI.4 muestra los problemas de usar un esquema de número de pasos fijos.

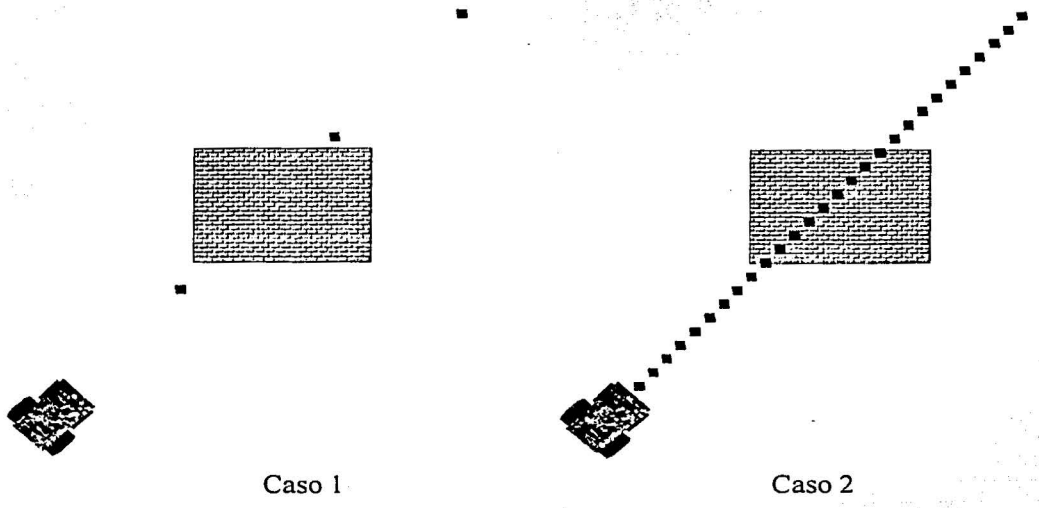


Figura VI.4 Problemas de usar un esquema de pasos fijos.

En la figura VI.4 tenemos que el caso 1 muestra la posibilidad de atravesar objetos de forma indeseable. El caso 2 presenta un muestreo con mayor densidad, esto ocasiona mayor cantidad de tiempo para finalizar el recorrido.

El esquema implantado en el sistema utiliza lo mejor de los dos anteriores; se utiliza un número de pasos dependiente de la distancia a recorrer y además dependiente del tiempo en el que se desea hacer.

De esta manera, se puede crear una función de dos variables que determine el número de pasos a realizar. La figura VI.5 muestra la superficie que se forma al graficar la función y la figura VI.6 muestra un caso del uso de este esquema.

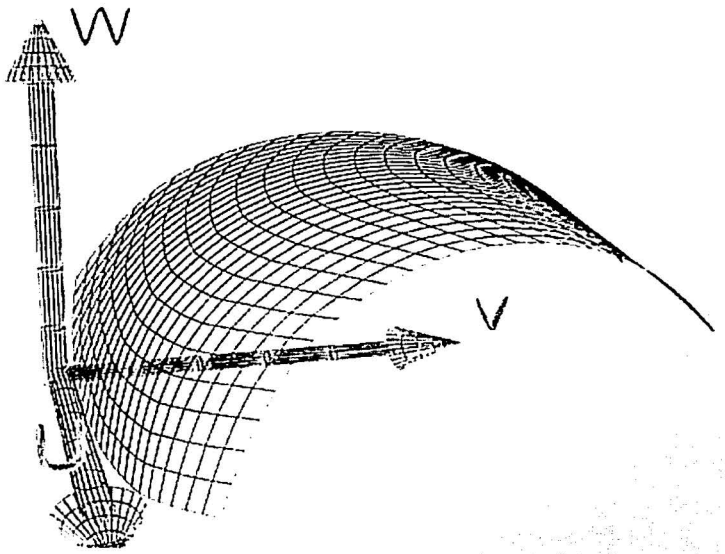


Figura VI.5 Superficie utilizada para el cálculo de los incrementos.

En la figura anterior, los ejes U y V son los ejes de la distancia y el tiempo respectivamente, y el eje W nos indica el número de pasos que son necesarios para recorrer la distancia requerida.

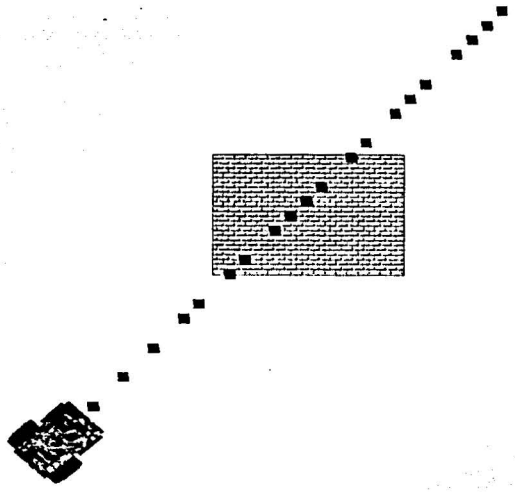


Figura VI.6 Cálculo de pasos usando una superficie paramétrica.

Instrucciones de nivel alto

Haciendo uso de la instrucción de nivel medio y suponiendo que conocemos la posición actual del robot, es posible que el robot se dirija a un punto específico del plano en el que se encuentra.

El cálculo de la posición siguiente está dado por las siguientes ecuaciones y haciendo uso de la figura VI.7.

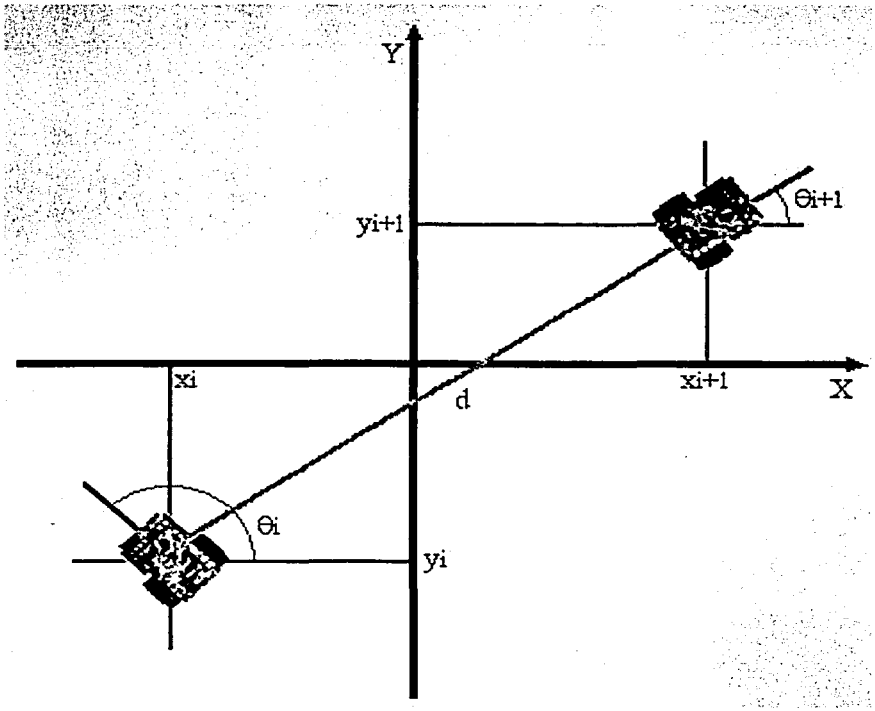


Figura VI.7 Cambios de posición haciendo uso de la instrucción mvto.

$$d = \sqrt{(x_{i+1} - x_i)^2 + (y_{i+1} - y_i)^2} \qquad (12)$$

$$x = x_{i+1} - x_i \qquad (13)$$

$$y = y_{i+1} - y_i \qquad (14)$$

$$\text{Si } x \cong 0 \text{ o } x=0 \qquad \begin{cases} y < 0 & \theta_{i+1} = 3\pi/2 \\ y >= 0 & \theta_{i+1} = \pi/2 \end{cases} \qquad (15)$$

$$\begin{cases} x > 0; y >= 0 & \theta_{i+1} = \tan^{-1}(y/x) \\ x < 0; & \theta_{i+1} = \tan^{-1}(y/x) + \pi \\ x > 0; y <= 0 & \theta_{i+1} = \tan^{-1}(y/x) + 2\pi \end{cases} \quad (16)$$

$$x_{i+1} = x_i + d\cos(\theta_{i+1}) \quad (17)$$

$$y_{i+1} = y_i + d\sin(\theta_{i+1}) \quad (18)$$

donde:

x_i, y_i es la posición actual del robot
x_{i+1}, y_{i+1} es la posición siguiente del robot
d es la distancia a recorrer
θ_i es el ángulo actual con el que se encuentra orientado el robot
θ_{i+1} es el ángulo en el que se encontrará orientado el robot en la posición siguiente

6.3.2 INSTRUCCIONES DE ESTADO LÓGICO DEL ROBOT

Este tipo de instrucciones son utilizadas para inicializar y conocer la posición actual del robot, así como también, obtener lecturas de los sensores con los que cuenta. El conjunto de instrucciones de estado lógico se muestra en el cuadro VI.5.

Instrucciones	*Parámetros*
shs sensor num_sensor	(sensor) nombre del tipo de sensor (num_sensor) identificador del sensor
ic x y θ	Posición (x, y) y orientación θ
sc	Ninguno

Cuadro VI.5 Conjunto de instrucciones para el estado lógico del robot.

Instrucción shs. Esta instrucción es utilizada para leer el estado lógico de un sensor. Los parámetros que recibe la instrucción shs son precisamente el tipo de sensor que se desea leer y el identificador del sensor seleccionado. Los robots pueden tener distintos tipos de sensores y más de un sensor de cada tipo; debido a esto, a cada sensor se le asigna un identificador numérico progresivo; es decir, si se cuenta con 4 sensores de contacto tendremos que el nombre del sensor es contact y sus identificadores son 1, 2, 3 y 4; así que para leer el sensor de contacto número 3 la sintaxis sería: shs contact 3. Algunos ejemplos de sensores que son posibles de utilizar son: sensores de contacto (contact), infrarrojo (infrared), reflectivos (reflective) y sonar (sonar).

La instrucción shs no regresa ningún valor, el estado del sensor es almacenado en el registro A, el cual pertenece a la máquina virtual (MVR).

Instrucción ic. En cualquier momento el usuario es capaz de redefinir la localización y la orientación del robot móvil que controla. Dentro de la estructura interna de cada robot está contenido el sistema coordenado con el cual trabaja; al inicializar un robot, dicho sistema se alínea con el sistema coordenado absoluto que pertenece a la interfaz 3D. De esta forma es posible redefinir el sistema coordenado propio de cada robot.

Instrucción sc. Esta instrucción regresa la posición y orientación del robot con respecto a su sistema coordenado.

En el manejo interno de la interfaz 3D sólo se puede hacer uso del sistema coordenado absoluto. Para realizar los cálculos asociados a los movimientos del robot, haciendo uso de su propio sistema coordenado, es necesario obtener funciones de mapeo entre ambos dominios. En la figura VI.8 se muestra un esquema del uso de coordenadas relativas y absolutas.

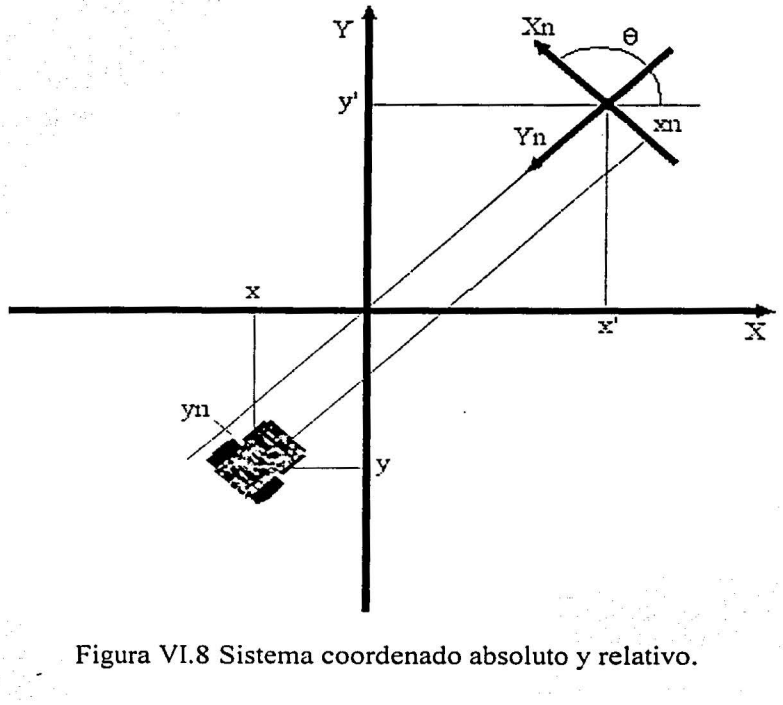


Figura VI.8 Sistema coordenado absoluto y relativo.

Donde:

X, Y son los ejes absolutos del sistema
Xn, Yn son los ejes relativos del sistema (propios al robot)
(x , y) es la posición del robot relativa a los ejes coordenados absolutos
(xn, yn) es la posición del robot relativa a sus propios ejes
x',y' es la posición del origen de los ejes relativos con respecto a los ejes absolutos
θ es la orientación de los ejes coordenados relativos con respecto a los ejes absolutos

El mapeo de coordenadas absolutas a relativas se calcula utilizando la ecuación (19). Como puede observarse, básicamente el mapeo entre el dominio absoluto y relativo se realiza alineando ambos sistemas coordenados.

$$\begin{bmatrix} xn \\ yn \\ 1 \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x' \\ 0 & 1 & -y' \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \tag{19}$$

$$\begin{aligned} xn &= x\cos\theta - y\sin\theta - x'\cos\theta + y'\sin\theta \\ yn &= x\sin\theta + y\cos\theta - x'\sin\theta - y'\cos\theta \end{aligned} \tag{20}$$

La ecuación (20) nos proporciona una función de mapeo del sistema de coordenadas absoluto al sistema de coordenadas relativo.

El mapeo de coordenadas relativas a absolutas se calcula a partir del mapeo anterior (ec.19).

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -x' \\ 0 & 1 & -y' \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}^{-1} \begin{bmatrix} xn \\ yn \\ 1 \end{bmatrix} \tag{21}$$

$$\begin{aligned} x &= xn\cos\theta + yn\sin\theta + x' \\ y &= -xn\sin\theta + yn\cos\theta + y' \end{aligned} \tag{22}$$

6.3.3 INSTRUCCIONES PARA COMUNICACIÓN REMOTA

Este tipo de instrucciones son utilizadas para entablar comunicación vía Internet utilizando la interfaz 3D; es por ello que son independientes del robot. El conjunto de instrucciones de comunicación se muestra en el cuadro VI.6.

Instrucciones	*Parámetros*
opsk w/r host port	(r/w) lectura/escritura (host) nombre del host (port) número de puerto
ksk w/r	(r/w) lectura/escritura

Cuadro VI.6 Conjunto de instrucciones para comunicación.

Instrucción opsk. Esta instrucción inicializa la comunicación vía Internet. Los parámetros que recibe permiten configurar qué tipo de comunicación se necesita, ya sea enviar (w) o recibir mensajes (r), así como también, especifica el nombre del host y puerto de comunicaciones.

Instrucción ksk. Esta instrucción cierra las comunicaciones.

La comunicación vía Internet que permite la interfaz 3D no es interactiva; su propósito es de manejo interno del sistema. Es decir, no es posible enviar mensajes por parte del usuario haciendo uso de instrucciones, tampoco es posible recibirlos; su función es permitir la entrada de mensajes de forma remota. Dichos mensajes son analizados por una parte del código y posteriormente enviados al manejador adecuado. Dentro de los manejadores que tenemos se encuentran el manejador de instrucciones (MVR); otros manejadores pueden ser implementados en versiones posteriores del sistema.

6.3.4 INSTRUCCIONES PARA MANEJO DE SCRIPTS

Estas instrucciones permiten cargar y ejecutar scripts con instrucciones de Robel. El conjunto de instrucciones para manejo de scripts se muestra en el cuadro VI.7.

Instrucciones	*Parámetros*
loads script_name	(script_name) nombre del script a cargar en el sistema
script script_name	(script_name) nombre del script a ejecutar
ascript	Ninguno

Cuadro VI.7 Conjunto de instrucciones de para manejo de scripts.

Instrucción loads. Esta instrucción es utilizada para cargar en memoria un script con instrucciones de Robel. El parámetro que recibe es el nombre del script. Como se ha descrito, antes de cargar un script en un robot es necesario compilarlo primero; en caso de cargar un script usando la interfaz 3D, la instrucción loads llama por sí misma al compilador de Robel y en caso de que no se hayan presentado errores se cargará el script en memoria. Para el caso de cargar un script dentro de un robot real, es necesario compilarlo y cargar el archivo objeto.

Instrucción script. Esta instrucción es utilizada para mandar a ejecutar un script que ha sido previamente cargado en memoria haciendo uso de la instrucción loads. El parámetro que recibe la instrucción es el nombre del script a ejecutar. Mientras el script es ejecutado el robot no atiende a ninguna otra instrucción salvo las que pertenecen al script.

Instrucción ascript. Esta instrucción es utilizada para abortar el script en ejecución. No recibe parámetros.

6.3.5 INSTRUCCIONES MISCELÁNEAS

Instrucción putbot. Esta instrucción permite colocar un robot en la posición y orientación indicada por los parámetros que recibe. Para mayor referencia véase el apartado C.6.1: Lista de instrucciones de Robel.

Instrucción hp. Esta instrucción despliega en pantalla una breve sinopsis del uso de las instrucciones (ayuda).

Instrucción tap. Esta instrucción tiene como propósito capturar la imagen mostrada en la perspectiva del robot. Usando esta instrucción, puede capturarse una imagen cuando ocurra algún evento como una colisión, y posteriormente utilizar procesamiento digital de imágenes para determinar qué acciones realizar.

6.3.6 INSTRUCCIONES DE SISTEMA

Las instrucciones de sistema permiten modificar el flujo de ejecución de un script. Como en otros lenguajes de programación, Robel es capaz de realizar ciclos, condicionales y saltos incondicionales; cada uno tiene su propia sintaxis como se describe a continuación.

6.3.6.1 Ciclos

Los ciclos en Robel pueden realizarse haciendo uso de la instrucción while. La sintaxis de la instrucción while es la siguiente:

```
while a $ b
        * línea 1
        * ...
        * línea n
wend
```

Donde a, b son un par de datos numéricos y $ es un operador de relación. Los operadores de relación que pertenecen a Robel están en el cuadro VI.8.

Operador	*Significado*
<	menor que
>	mayor que
=	igual
!	diferente

Cuadro VI.8 Operadores de relación de Robel.

Mientras la condición indicada por a $ b sea verdadera, se ejecutarán los comandos que se encuentren entre el indicador while y wend; cuando la condición es falsa, la siguiente instrucción a ejecutar será la que se encuentre en la línea siguiente al indicador wend. Debido a la forma de operar, es posible tener ciclos anidados en Robel.

6.3.6.2 Condicionales

Los condicionales en Robel pueden realizarse haciendo uso de la instrucción if. La sintaxis de la instrucción if es la siguiente:

```
if a $ b
        * línea 1
        * ...
        * línea n
endif
```

Donde a, b son un par de datos numéricos y $ es un operador de relación. Los operadores de relación que pertenecen a Robel están en el cuadro VI.8.

Si la condición indicada por a $ b es verdadera, entonces se ejecutan los comandos que se encuentren entre el indicador if y endif; cuando la condición es falsa, la siguiente instrucción a ejecutar será la que se encuentre en la línea siguiente al indicador endif. Debido a la forma de operar, es posible tener condicionales anidadas en Robel.

6.3.6.3 Saltos incondicionales

Los saltos incondicionales pueden realizarse haciendo uso de la instrucción goto. La sintaxis de la instrucción goto es la siguiente:

```
        goto label1
        ...
        ...
        ...
label1: ...
```

donde label1 es una etiqueta indicada dentro del propio script. Los saltos pueden hacerse tanto hacia atrás como hacia adelante sin importar el número de líneas.

6.3.6.4 Declaración de variables

Robel permite declarar variables que pueden ser utilizadas como parámetros en instrucciones o para otros propósitos determinados por el usuario. Los tipos de variables soportados son enteros y reales de punto flotante. La sintaxis de la declaración de variables es la siguiente:

```
int a,b,c,d
float e,f,g,h
...
int  x
float y
...
...
```

La declaración de variables hace uso de las palabras reservadas int y float para declarar variables de tipo entero y reales de punto flotante respectivamente. Como puede observarse, la declaración de variables puede realizarse en cualquier parte del archivo script.

6.3.6.5 Uso de operaciones aritméticas

Robel es capaz de realizar operaciones aritméticas sobre líneas exclusivas para dicho propósito, es decir, sólo es posible realizar operaciones aritméticas en líneas donde sólo hay una operación qué realizar. Por ejemplo:

```
a <- a + b - 1.3
```

Dado que los scripts de Robel pueden ejecutarse sobre distintas plataformas, las operaciones aritméticas posibles de realizar son limitadas. El cuadro VI.9 muestra los operadores aritméticos que son soportados por Robel.

Operador	*Significado*
<-	asignación
+	adición
-	sustracción
*	multiplicación
/	división

Cuadro VI.9 Operadores aritméticos.

6.3.6.6 Constantes

Dentro de un script es posible crear constantes que pueden ser utilizadas en cualquier parte dentro de un archivo script. La sintaxis para el uso de constantes es la siguiente:

```
#define MI_CONSTANTE 300
#define APUNTA       mvto
```

Como puede observarse el uso de constantes es similar al utilizado en lenguaje C. Dichas constantes pueden ser declaradas en cualquier parte del archivo script. Haciendo uso de esta directiva es posible redefinir el nombre de un comando.

A continuación se muestra un ejemplo de script:

```
                *Ejemplo de script de Robel
                #DEFINE SENSOR_DEFAULT 0
                *Este es un comentario... gracias.

                int status

                status <- 0                          @0
leer:           shs reflective SENSOR_DEFAULT        @1
                status <- RegF1                      @2
                if status < 30                       @3
                      forward                        @4
                      goto leer                      @5
                endif                                @6
                backward                             @7
                goto leer                            @8
```

Una de las ventajas que tiene el uso de instrucciones de alto nivel es que un robot que sea capaz de interpretar dichas instrucciones es mucho más fácil de controlar y empezar a utilizar, que otro robot que solamente cuenta con instrucciones dependientes y específicas para él mismo, haciendo que el tiempo de desarrollo del control del robot aumente significativamente.

El uso de instrucciones de alto nivel permiten que un robot realice operaciones que de otra forma serían complicadas de realizar. Además, la posibilidad de ejecutar el mismo código en diferentes tipos de robots, tanto virtuales como reales, nos da una forma de evaluar el correcto desempeño del sistema.

CAPÍTULO VII
PRUEBAS AL SISTEMA Y CONCLUSIONES

7.1 PRUEBAS AL SISTEMA

Las pruebas de sistema consistieron en verificar el correcto funcionamiento de Roc2. Para ello, se probó el conjunto de instrucciones de Robel sobre los robots virtuales, tanto de manera local como de manera remota. Las pruebas locales (pruebas efectuadas en la misma máquina donde está ejecutándose Roc2) consistieron en probar todas las interfaces que nos permiten interactuar con Roc2, como son teclado, mouse y joypad. Además, se escribieron algunos scripts (guiones) para que los robots del sistema los ejecutaran, y se verificaba que en verdad se estuvieran realizando los comandos adecuados.

En cuanto a las pruebas remotas (en máquinas lejanas que mantienen comunicación con Roc2 vía Internet), también se verificó el correcto funcionamiento de los comandos básicos de Robel.

Los comandos que Roc2 recibía remotamente consistían primordialmente de comandos de control de movimiento; estos comandos a su vez eran generados por un sistema experto encargado de la planificación de rutas, de manera que el sistema experto calculaba la mejor ruta que el robot debía seguir para llegar de un punto a otro dentro de un mundo virtual, y enviaba, vía Internet, los comandos de movimiento necesarios para que el robot alcanzara el destino propuesto. El mundo virtual que se empleó en el planificador y en Roc2 fue el Laboratorio de Interfaces Inteligentes.

Gracias a la portabilidad del lenguaje, los comandos de movimiento generados por el planificador pudieron exportarse a un robot verdadero, y se pudieron probar los mismos algoritmos que se ejecutaron satisfactoriamente en los robot virtuales.

Se realizaron pruebas en un robot Real World Interface (RWI) comercial que cuenta con sistema operativo Linux, procesador Pentium, sensores de sonar, infrarrojos y de contacto, además de estar conectado a Internet inalámbricomente. Véase figura VII.1.

También se realizaron pruebas en un minirobot que fue diseñado en el Laboratorio de Interfaces Inteligentes basado en el microcontrolador MC68HC11, que cuenta con sensor de sonar, sensores infrarrojos y sensores de contacto. Véase figura VII.2. El ambiente de prueba fue el Laboratorio de Interfaces Inteligentes.

De esta manera se cumplió con el objetivo de Roc2: probar algoritmos de control de movimiento en robots móviles dentro de ambientes virtuales, antes de probar dichos algoritmos en robots reales dentro de ambientes verdaderos. El objetivo de controlar un robot real se alcanzó al utilizar el lenguaje desarrollado, pues se pudo emplear sin ningún problema en un robot verdadero.

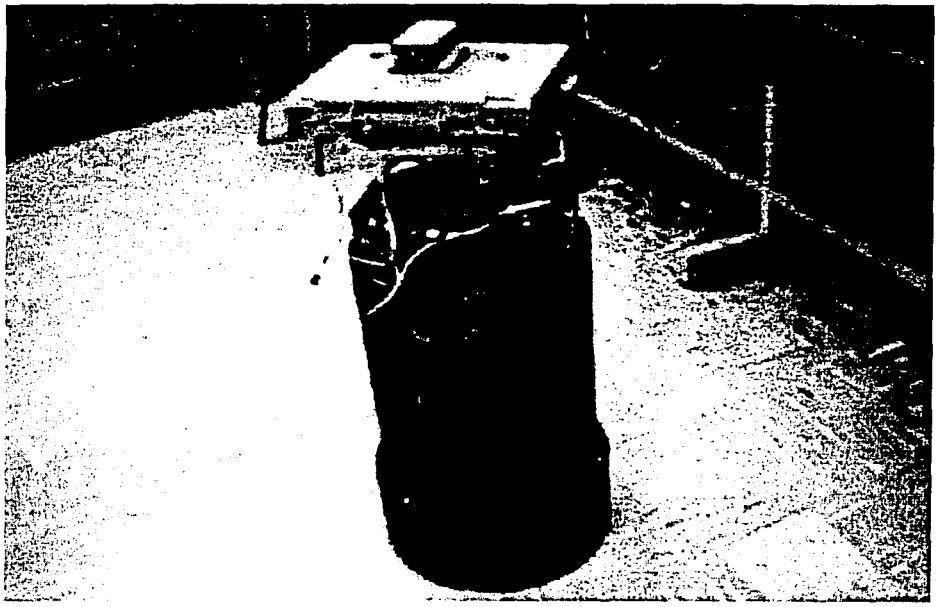


Figura VII.1 Robot RWI.

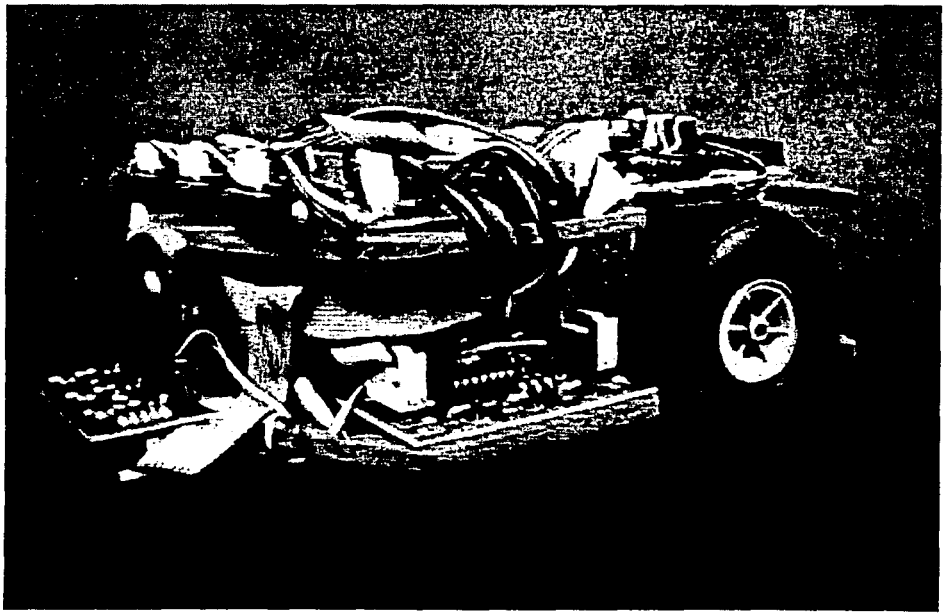


Figura VII.2 Robot basado en MC68HC11.

A continuación se presentan los scripts utilizados en la pruebas y se describe brevemente cada uno de ellos.

Script 1. Script generado por el sistema experto. Consiste en recorrer la siguiente trayectoria dentro de un ambiente virtual.

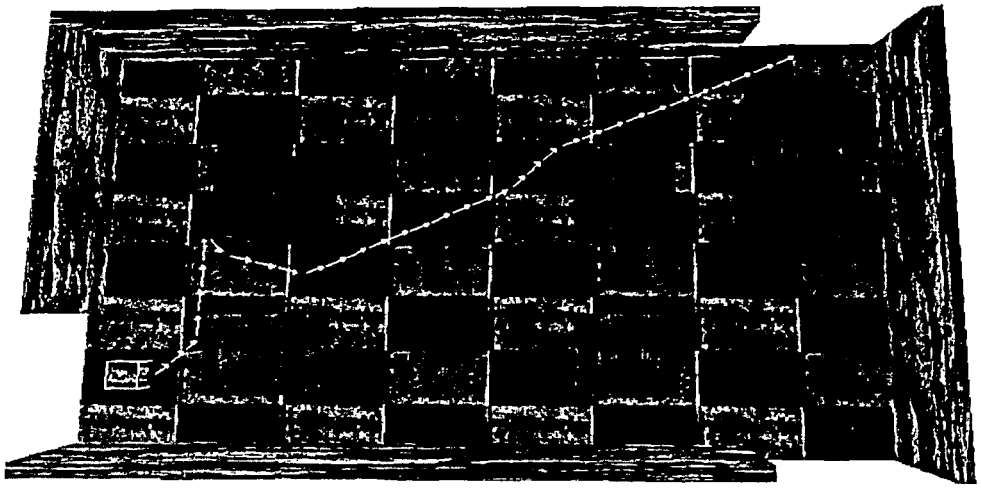


Figura VII.3 Trayectoria del robot definida por el script 1.

```
*Script 1
putbot 24.0 3.0 0.0   @0
mvto 24.881  3.087 1 @1
mvto 23.377  5.683 1 @2
mvto 21.873  8.279 1 @3
mvto 21.870  8.284 1 @4
mvto 21.867  8.289 1 @5
mvto 21.864  8.294 1 @6
mvto 21.861  8.299 1 @7
mvto 21.857  8.304 1 @8
mvto 21.854  8.309 1 @9
mvto 21.851  8.314 1 @10
mvto 21.848  8.319 1 @11
mvto 21.845  8.324 1 @12
mvto 21.841  8.329 1 @13
mvto 21.838  8.335 1 @14
mvto 21.835  8.340 1 @15
mvto 21.832  8.345 1 @16
mvto 18.842  8.091 1 @17
mvto 15.845  7.974 1 @18
mvto 16.199 10.953 1 @19
mvto 16.716 13.908 1 @20
mvto 16.999 16.895 1 @21
mvto 15.488 19.487 1 @22
mvto 13.976 22.078 1 @23
mvto 12.464 24.669 1 @24
mvto 10.974 27.273 1 @25
mvto  9.458 29.862 1 @26
mvto  6.594 30.753 1 @27
mvto  5.430 33.518 1 @28
mvto  4.267 36.284 1 @29
mvto  3.104 39.049 1 @30
mvto  1.949 41.818 1 @31
mvto  0.784 44.582 1 @32
```

Script 2. Script generado por el sistema experto. Consiste en recorrer la siguiente trayectoria dentro del ambiente virtual del Laboratorio de Interfaces Inteligentes.

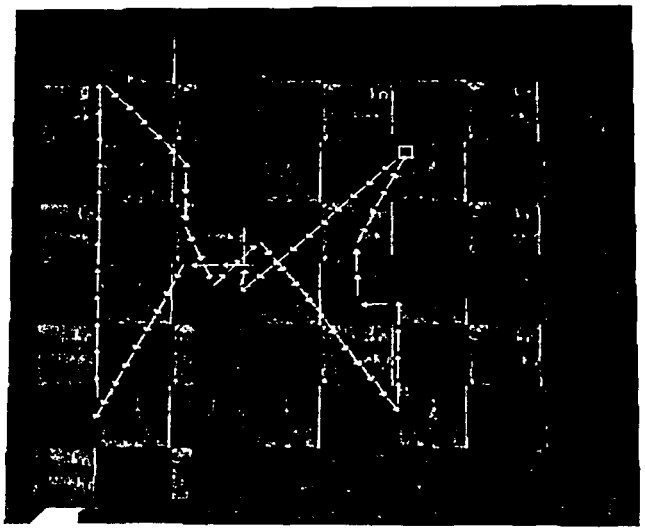


Figura VII.4 Trayectoria del robot definida por el script 2.

```
*Script 2
putbot 20.5 49.5 0.0 @0
mvto 19.397 48.489 1 @1
mvto 20.993 46.976 1 @2
mvto 22.590 45.462 1 @3
mvto 24.187 43.949 1 @4
mvto 25.783 42.435 1 @5
mvto 27.380 40.922 1 @6
mvto 28.977 39.408 1 @7
mvto 30.573 37.895 1 @8
mvto 32.170 36.381 1 @9
mvto 33.766 34.868 1 @10
mvto 35.363 33.354 1 @11
mvto 36.960 31.841 1 @12
mvto 38.582 30.354 1 @13
mvto 40.186 28.849 1 @14
mvto 40.189 28.846 1 @15
mvto 40.192 28.842 1 @16
mvto 40.194 28.839 1 @17
mvto 40.197 28.835 1 @18
mvto 40.200 28.832 1 @19
mvto 40.203 28.828 1 @20
mvto 40.205 28.825 1 @21
mvto 40.208 28.822 1 @22
mvto 40.211 28.818 1 @23
mvto 40.213 28.815 1 @24
mvto 40.216 28.811 1 @25
mvto 38.234 29.765 1 @26
mvto 36.344 30.891 1 @27
mvto 36.474 28.695 1 @28
mvto 36.672 26.504 1 @29
mvto 36.871 24.313 1 @30
mvto 37.071 22.122 1 @31
mvto 38.875 20.863 1 @32
mvto 40.836 19.865 1 @33
mvto 42.793 18.861 1 @34
mvto 44.749 17.854 1 @35
mvto 46.705 16.846 1 @36
mvto 48.661 15.839 1 @37
mvto 50.617 14.832 1 @38
mvto 52.573 13.825 1 @39
mvto 54.528 12.818 1 @40
mvto 56.484 11.810 1 @41
mvto 58.440 10.803 1 @42
*DESTINE REACHED 1
```

```
mvto 60.396 9.796 1 @43
mvto 57.801 9.931 1 @44
mvto 55.602 9.861 1 @45
mvto 53.403 9.792 1 @46
mvto 51.204 9.722 1 @47
mvto 49.005 9.653 1 @48
mvto 46.807 9.583 1 @49
mvto 44.608 9.514 1 @50
mvto 42.409 9.445 1 @51
mvto 40.210 9.375 1 @52
mvto 38.011 9.306 1 @53
mvto 35.812 9.236 1 @54
mvto 33.613 9.167 1 @55
mvto 31.414 9.097 1 @56
mvto 29.215 9.028 1 @57
mvto 27.016 8.958 1 @58
mvto 24.818 8.889 1 @59
mvto 22.619 8.820 1 @60
mvto 20.420 8.750 1 @61
mvto 18.221 8.681 1 @62
mvto 16.022 8.611 1 @63
mvto 13.823 8.542 1 @64
*DESTINE REACHED 2

mvto 11.624 8.472 1 @65
mvto 13.217 9.902 1 @66
mvto 14.888 11.333 1 @67
mvto 16.558 12.765 1 @68
mvto 18.229 14.197 1 @69
mvto 19.899 15.629 1 @70
mvto 21.569 17.060 1 @71
mvto 23.240 18.492 1 @72
mvto 24.910 19.924 1 @73
mvto 26.595 21.339 1 @74
mvto 28.120 19.754 1 @75
mvto 29.726 21.258 1 @76
mvto 31.910 20.994 1 @77
mvto 33.710 22.258 1 @78
mvto 35.863 22.713 1 @79
mvto 37.281 24.395 1 @80
mvto 38.729 26.051 1 @81
mvto 40.187 27.699 1 @82
mvto 40.189 27.702 1 @83
mvto 40.192 27.706 1 @84
mvto 40.194 27.710 1 @85
mvto 40.197 27.713 1 @86
mvto 40.199 27.717 1 @87
mvto 40.202 27.720 1 @88
mvto 40.204 27.724 1 @89
mvto 40.207 27.728 1 @90
mvto 40.210 27.731 1 @91
mvto 40.212 27.735 1 @92
mvto 40.215 27.738 1 @93
mvto 38.492 29.107 1 @94
mvto 36.883 30.608 1 @95
mvto 38.248 32.334 1 @96
mvto 39.956 33.719 1 @97
mvto 41.685 35.080 1 @98
mvto 43.391 36.470 1 @99
mvto 45.096 37.859 1 @100
mvto 46.802 39.249 1 @101
mvto 48.508 40.638 1 @102
mvto 50.213 42.028 1 @103
mvto 51.919 43.417 1 @104
mvto 53.625 44.807 1 @105
mvto 55.330 46.196 1 @106
mvto 57.036 47.586 1 @107
mvto 58.742 48.975 1 @108
*DESTINE REACHED 3

mvto 60.447 50.365 1 @109
mvto 57.800 50.000 1 @110
mvto 55.600 50.000 1 @111
mvto 53.400 50.000 1 @112
mvto 51.200 50.000 1 @113
mvto 49.000 50.000 1 @114
mvto 46.800 50.000 1 @115
mvto 44.600 50.000 1 @116
mvto 44.596 49.998 1 @117
mvto 44.593 49.995 1 @118
mvto 44.589 49.993 1 @119
mvto 44.585 49.990 1 @120
mvto 44.582 49.987 1 @121
mvto 44.579 49.984 1 @122
mvto 44.575 49.981 1 @123
mvto 44.572 49.978 1 @124
mvto 44.569 49.975 1 @125
mvto 44.566 49.972 1 @126
mvto 44.564 49.968 1 @127
mvto 44.879 47.791 1 @128
mvto 44.841 45.592 1 @129
mvto 44.847 43.392 1 @130
mvto 43.284 41.843 1 @131
mvto 41.120 42.240 1 @132
mvto 38.959 42.649 1 @133
mvto 36.795 43.050 1 @134
mvto 34.625 43.411 1 @135
mvto 32.569 44.192 1 @136
mvto 30.571 45.114 1 @137
mvto 28.574 46.037 1 @138
mvto 26.577 46.960 1 @139
mvto 24.580 47.883 1 @140
mvto 22.583 48.806 1 @141
*DESTINE REACHED 4
```

Script 3. Este script lee el estado de los sensores de contacto de un robot y con base en ello determina qué acción ejecutar. El robot utilizado es un robot básico de Roc2. Por ejemplo, si los sensores frontales (sensor 1 y 2) no están chocando contra algún objeto, entonces el robot avanza hacia adelante; si el sensor frontal 1 o el sensor lateral derecho (sensor 3) están chocando, entonces el robot retrocede una pequeña distancia y gira hacia su izquierda; y si el sensor frontal 2 o el sensor lateral izquierdo (sensor 4) están chocando, el robot retrocederá cierta distancia y girará hacia la derecha. Este script continuará ejecutándose hasta que el usuario lo detenga.

```
*CONTACT TEST
float r1, r2, r3, r4

Ini:
    *READ CONTACT SENSORS
    shs contact 1          @0
    r1 <- RegF1            @1
    shs contact 2          @2
    r2 <- RegF1            @3
    shs contact 3          @4
    r3 <- RegF1            @5
    shs contact 4          @6
    r4 <- RegF1            @7

    if r3 = 1.0            @8
      goto LeftTurn        @9
    endif                  @10
    if r4 = 1.0            @11
      goto RightTurn       @12
    endif                  @13
    if r1 = 1.0            @14
      goto LeftTurn        @15
    endif                  @16
    if r2 = 1.0            @17
      goto RightTurn       @18
    endif                  @19
    if r1 = 0.0            @20
      if r2 = 0.0          @21
          forward          @22
      endif                @23
    endif                  @24
    goto Ini               @25

LeftTurn:
    backward               @26
    left                   @27
    left                   @28
    goto Ini               @29
RightTurn:
    backward               @30
    right                  @31
    right                  @32
    goto Ini               @33
```

7.2 CONCLUSIONES

El Robot Command Center (Roc2) consiste de una interfaz que hace uso de gráficos 3D para poder interactuar con robots virtuales y cuenta con un lenguaje de programación, Robel, que puede ser ejecutado en robots reales y virtuales; tomando en cuenta los puntos anteriores, es posible decir que los objetivos del proyecto han sido cumplidos.

Al contar el Roc2 con gráficas 3D como medio de visualización de resultados, se dispone de una buena representación de robots reales, ya que éstos se desenvuelven en ambientes tridimensionales.

Usar un simulador para probar algoritmos de robots móviles permite reducir el tiempo de desarrollo e implementación de algoritmos, es decir, permite probar algoritmos de forma intensiva. En caso de encontrar un error es posible determinar las causas, ya que se cuenta con un entorno controlado, por lo que es posible repetir las circunstancias del error.

La comunicación vía Internet del Roc2 permite utilizar sistemas de reconocimiento de voz, planeadores de rutas, sistemas expertos, sistemas de reconocimiento de patrones, entre otros, por lo que puede utilizarse en otros proyectos.

Usar un lenguaje de programación de alto nivel para programar robots móviles ha sido una forma sencilla y rápida para desarrollar programas complicados. Aún no existe un lenguaje ideal para la programación de robots; sin embargo, son muchos los lenguajes que han sido creados; algunas de las causas principales de la amplia gama de lenguajes son:

- Portabilidad. La mayoría de los lenguajes han sido desarrollados tomando como base un robot en específico; esto anula la posibilidad de usar el lenguaje en robots diferentes. En nuestro caso, Robel necesita una máquina virtual para que sus instrucciones puedan ser interpretadas y ejecutadas. Un robot que desee tener compatibilidad con Robel necesita contar con su propia máquina virtual diseñada para funcionar específicamente con el modelo de robot. Desarrollar una máquina virtual para un diseño de robot en especifico puede limitarse a modificar las rutinas de comunicación y movimiento del robot real; sin embargo, esto no garantiza que todo el conjunto de instrucciones pueda ser ejecutado.

- Aplicaciones. En muchos casos, los lenguajes se dirigen hacia aplicaciones diferentes; su utilización está limitada al conjunto para el que fue diseñado, es decir, es incapaz de realizar tareas fuera de su contexto. Robel está dirigido al control de movimiento de un robot, pero debido a la posibilidad de crear subrutinas y utilizar sensores, es posible crear comportamientos más complicados.

Algunos otros proyectos en los que se utiliza actualmente Robel son:

- Control de un dispositivo autónomo de captura de imágenes para vistas de 360°

- Control de un robot móvil que corrige sus movimientos mediante una red neuronal

- Uso de algoritmos genéticos para planeación de rutas
- Sistema experto que planea trayectorias de movimiento de un robot

Existen mejoras que pueden realizarse al proyecto tanto en la parte de la interfaz 3D como en Robel. La siguiente es una lista de elementos que deben de tomarse en cuenta para mejorar el sistema:

- Mejorar el desempeño final del sistema para que pueda ser utilizado en máquinas con menor cantidad de memoria y menor velocidad de procesador
- Añadir nuevos modelos de robots
- Mejorar el formato de definición de entornos virtuales (mapas)
- Mejorar la información de estado del sistema que se proporciona al usuario
- Añadir interacción usando voz
- Expandir el número de instrucciones de Robel que pueden ser utilizadas
- Reducir el tamaño de la máquina virtual de Robel
- Añadir macros dentro de los archivos de Robel.

APÉNDICE A
BIBLIOTECAS DE ENLACE DINÁMICO

A.1 DEFINICIÓN

Una biblioteca de enlace dinámico (Dynamic-Link Library o DLL) es un conjunto de rutinas que pueden ser llamadas desde procedimientos, y son cargadas y ligadas a la aplicación en tiempo real. Es decir, es un archivo que contiene una o más funciones que son compiladas, ligadas y almacenadas de forma separada al proceso que las usa, el sistema operativo carga la función requerida del DLL dentro del espacio de memoria requerido durante el tiempo de ejecución.

Dentro de un DLL existen funciones internas y funciones externas o de exportación. Las funciones exportadas pueden ser llamadas por otros módulos. Las funciones internas sólo pueden ser llamadas donde son definidas, es decir, dentro del módulo del DLL.

A.2 VENTAJAS

La ventaja de utilizar bibliotecas de enlace dinámico consiste en que las aplicaciones pueden ser modularizadas de forma sencilla y por lo mismo es más sencillo realizar actualizaciones. Usar DLL permite que varias aplicaciones utilicen la misma subrutina compartiendo código, pero cada una utilizando distintos datos.

A.3 CÓMO CREAR UN DLL UTILIZANDO VISUAL C++

El siguiente ejemplo muestra cómo crear una biblioteca DLL haciendo uso de Microsoft Visual C++ 6 .0 .

Pasos a seguir:

- Crear un nuevo proyecto de creación de DLL
- Añadir archivos fuente
- Crear archivo make.def
- Compilar el proyecto

Crear un proyecto de creación de DLL

Dentro del Microsoft Visual C++ 6.0 hay que dirigirse al apartado *File/New* y dentro de la pestaña de *Projects* seleccionar la opción *Win32 Dynamic-Link Library* como lo muestra la figura A.1.

El proyecto debe tener un nombre que en nuestro caso es: *"DLLexample"*. Cuando se pregunte qué tipo de proyecto se va a crear es necesario seleccionar que sea un proyecto vacío.

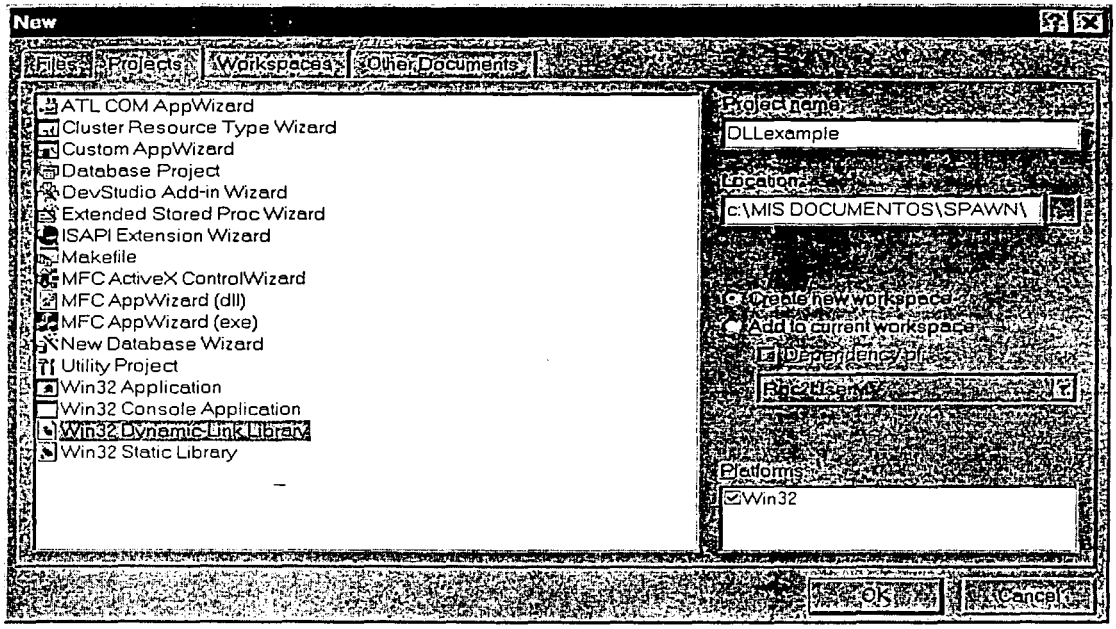


Figura A.1 Selección de proyecto de creación de un DLL.

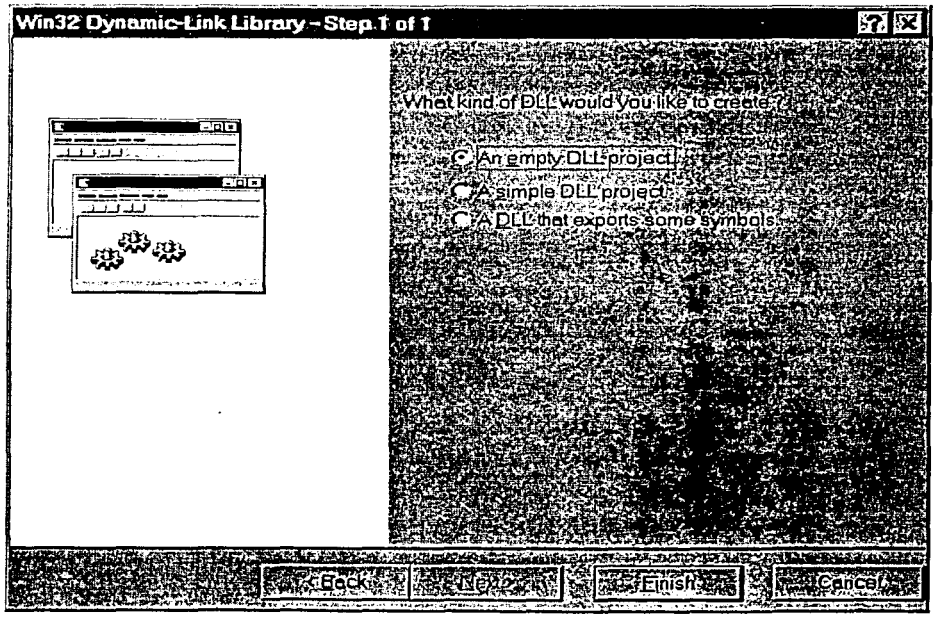


Figura A.2 Selección de un proyecto DLL vacío.

Añadir archivos fuente

Crear un DLL no es muy diferente de crear una biblioteca convencional; debemos contar con al menos un archivo de código fuente. En nuestro caso crearemos los archivos:

- dll_source.cpp
- dll_header.h

El contenido de cada archivo es mostrado a continuación.

Contenido de los archivos dll_source.cpp y dll_source.h

```
/*** INICIO DE ARCHIVO dll_source.cpp ***/
/*************************
      Archivo dll_source.cpp
*************************/

float get_Square(float value){
/*****************************************
Esta función regresa el cuadrado de value
y es de uso interno del DLL
*****************************************/
      return value*value;
}

float WINAPI evaluate_Function(float x, float y){
/*************************************************
Esta función regresa un flotante que es el valor
de la función evaluada en la coordenada (x,y).
La declaración WINAPI determina que esta función
es de exportación.
*************************************************/
      float aux;
      aux = get_Square(x*y)+2*y-x/3+2.0f;
      return aux;
}
/*** FIN DE ARCHIVO dll_source.cpp ***/


/*** INICIO DE ARCHIVO dll_source.h ***/
/************************
      Archivo dll_source.h
************************/

float get_Square(float value);
float WINAPI evaluate_Function(float x, float y);
/*** FIN DE ARCHIVO dll_source.h ***/
```

Una vez creados ambos archivos es necesario añadirlos al proyecto como lo muestra la figura A.3.

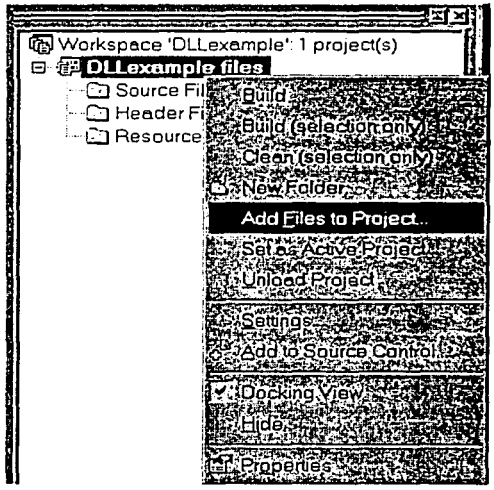


Figura A.3 Añadir archivos fuente y encabezados.

Creación del archivo make.def

El archivo make.def tiene como función determinar cuales serán las funciones que serán exportadas por el DLL. Una vez creado es necesario añadirlo al proyecto de la misma forma en que se añadieron los archivos dll_source.dll y dll_source.h. El siguiente es un ejemplo de formato del archivo make.def:

```
LIBRARY Dllexample
DESCRIPTION 'Ejemplo de archivo make.def'

EXPORTS
        _evaluate_Function@16
```

Compilar el proyecto

Por último, compile el proyecto.

APÉNDICE B

SNES PAD

B.1 EL PUERTO PARALELO

El puerto paralelo es un puerto estándar que sirve como interfaz entre un dispositivo externo y el CPU; por lo general, el dispositivo externo más utilizado es la impresora. Esta interfaz se encarga de gobernar una serie de señales de entrada/salida, que el CPU emplea para conocer el estado del dispositivo externo.

El hardware del puerto paralelo consiste de 8 líneas de salida para datos y 9 líneas de entrada o salida para control. Las 8 líneas de datos contienen la información que el CPU envía hacia el dispositivo externo, mientras que las líneas de control sirven para sincronizar el CPU con el dispositivo externo y para conocer el estado del dispositivo.

Todas las señales de entrada/salida están conectadas a un conector hembra de 25 pines con niveles lógicos TTL. A continuación se enlistan estas líneas.

Número de Pin	*Nombre de la Línea*	*Tipo*	*Descripción*
1	STROBE/	Entrada/Salida	Comunica al dispositivo externo que los ocho bits de datos están disponibles para ser leídos. Conmuta a nivel lógico bajo cuando los datos están preparados.
2	DATA 0	Salida	Bit 0 de datos.
3	DATA 1	Salida	Bit 1 de datos.
4	DATA 2	Salida	Bit 2 de datos.
5	DATA 3	Salida	Bit 3 de datos.
6	DATA 4	Salida	Bit 4 de datos.
7	DATA 5	Salida	Bit 5 de datos.
8	DATA 6	Salida	Bit 6 de datos.
9	DATA 7	Salida	Bit 7 de datos.
10	ACK/	Entrada	Informa a la CPU que los datos han sido recibidos correctamente.
11	BUSY	Entrada	Línea de ocupado. Ejemplo: la impresora pone esta línea a nivel lógico alto si el buffer de memoria está lleno. El ordenador dejará de enviar más datos.
12	PE	Entrada	Línea de Paper-Out/Paper-End. El papel se ha acabado en la impresora.
13	SLCT	Entrada	Indica al ordenador que se dispone de una impresora.
14	AUTO FD/	Entrada/Salida	Línea de Auto-Linefeed.
15	ERROR/	Entrada	Dice al ordenador que se ha producido un error. El CPU deja de enviar más datos.
16	INIT/	Entrada/Salida	señal de reset.
17	SLCT IN/	Entrada/Salida	Línea de Select-Printer. Selecciona la impresora.
18-25	GROUND		Tierra (0 volts).

Tabla B.1 Descripción de los pines del puerto paralelo.

El sistema operativo MS-DOS soporta tres puertos paralelos denominados LPT1, LPT2 y LPT3. Para acceder vía software a las líneas de entrada/salida de cada puerto paralelo, se cuenta con tres juegos de direcciones dedicadas de memoria en el mapa de memoria de entradas/salidas de la computadora. Estas direcciones dedicadas son las siguientes.

Puerto paralelo	*Dirección de datos*	*Dirección de status*	*Dirección de control*
LPT1	378H	379H	37AH
LPT2	278H	279H	27AH
LPT3	3BCH	3BDH	3BEH

Tabla B.2 Direcciones de memoria asociadas a cada puerto paralelo.

Cada dirección de memoria está formada por 8 bits. Su función y la disposición de los pines en cada localidad de memoria es la siguiente.

- *Dirección de datos.* En esta dirección se escriben los datos que serán enviados al dispositivo externo. Este registro es de salida (escritura).
- *Dirección de status.* A través de esta dirección el CPU puede conocer el estado del dispositivo externo. Este registro es de entrada (sólo lectura).
- *Dirección de control.* En esta dirección el CPU escribe las señales que controlan al dispositivo externo.

Dirección	*Bit*	*Función*	*Pin*
Datos	D0	Dato 0	2
	D1	Dato 1	3
	D2	Dato 2	4
	D3	Dato 3	5
	D4	Dato 4	6
	D5	Dato 5	7
	D6	Dato 6	8
	D7	Dato 7	9
Status	D0	No conectado	---
	D1	No conectado	---
	D2	IRQ (not)	---
	D3	ERROR/	15
	D4	SLCT/	13
	D5	PE	12
	D6	ACK/	10
	D7	BUSY/	11
Control	D0	STROBE	1
	D1	AUTO FD/	14
	D2	INIT/	16

	D3	SLCT IN/	17
	D4	Habilitación IRQ vía la línea ACK/	---
	D5	Habilitación bidireccional	---
	D6	No conectado	---
	D7	No conectado	---

Tabla B.3 Disposición de los pines del puerto paralelo en la memoria.

B.2 EL CONTROL DE SNES

El siguiente esquema muestra la disposición de los pines del conector hembra del SNES. También se incorpora una tabla con la descripción de cada uno de estos pines.

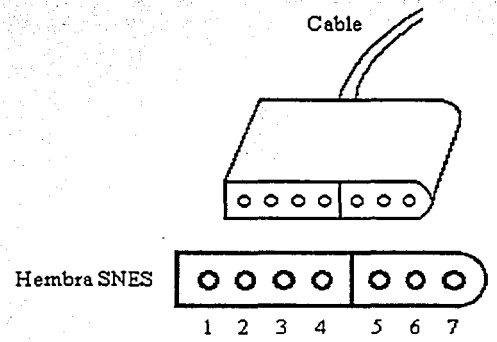


Figura B.1 Conector del control de SNES.

Pin	*Descripción*	*Color del alambre en el cable*
1	+ 5 volts	blanco
2	reloj	amarillo
3	data latch	– naranja
4	dato serial	rojo
5	?	sin alambrar
6	?	sin alambrar
7	0 volts	café

Tabla B.4 Descripción de los pines del control de SNES.

B.2.1 PROTOCOLO DE COMUNICACIÓN ENTRE EL CONTROL DEL SNES Y EL CPU DEL SNES

Cada 16.67 milisegundos, es decir, cada 60 Hz, el CPU del SNES envía un pulso positivo con duración de 12 microsegundos al pin 3 del conector del SNES (al pin 'data latch'). Este pulso funciona como una señal de reset que prepara a todos los circuitos integrados, dentro del control de SNES, para la lectura de los botones del control. Seis microsegundos después de la caída del pulso en el pin 3, el CPU envía un tren de pulsos al pin 2. Este tren de pulsos está formado por 16 pulsos con duración de 12 microsegundos cada uno. El ciclo de trabajo de un pulso es del 50%, es decir, cada pulso dura el 50% del tiempo (6 microsegundos) en alto y el otro 50% en bajo (formando así la señal de reloj).

Durante el tiempo en el que está presente el tren de pulsos, el control de SNES enviará el estado de cada uno de los botones. Cada dato leído es enviado serialmente al pin 4 durante el flanco de subida del reloj, y el CPU del SNES se encarga de leer este dato durante el flanco de bajada del reloj.

Cada uno de los botones del control tiene asignado un identificador que corresponde a un pulso en el tren de pulsos, de manera que durante ese pulso de reloj el estado del botón será reportado. A continuación se lista el identificador para cada botón del control.

Identificador / Ciclo de reloj	*Botón reportado*
1	B
2	Y
3	Select
4	Start
5	Arriba
6	Abajo
7	Izquierda
8	Derecha
9	A
10	X
11	L
12	R
13	Ninguno (se mantiene en alto)
14	Ninguno (se mantiene en alto)
15	Ninguno (se mantiene en alto)
16	Ninguno (se mantiene en alto)

Tabla B.5 Identificador para los botones del control del SNES.

No olvide que varios botones pueden estar presionados o sin presionar en cualquier momento. Un valor lógico alto en la línea de datos seriales indicará que el botón no ha sido presionado.

Al terminar la secuencia de los 16 ciclos de reloj, la línea de datos seriales es colocada a nivel lógico bajo hasta el siguiente pulso en la línea 'data latch'. Y la línea 'reloj' (pin 2) permanecerá en un nivel lógico alto hasta finalizar el pulso en la línea 'data latch' (momento en el cual comienza a generarse el tren de 16 pulsos).

Enseguida se muestra un diagrama de tiempos para aclarar las ideas antes expuestas. Observe que los botones B, Select, Start e Izquierda están presionados.

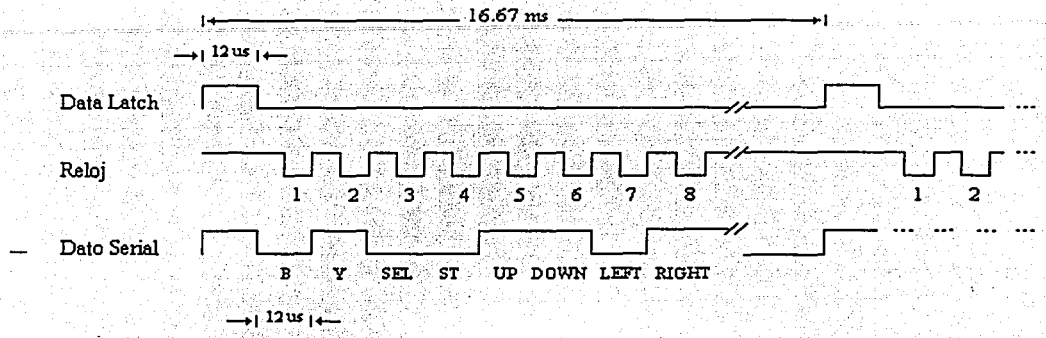


Figura B.2 Diagrama de tiempos para la comunicación entre el control de SNES y el CPU del SNES.

B.3 CONECTANDO EL CONTROL DE SNES A LA PC

B.3.1 MONTAJE

El siguiente esquema ilustra la manera de conectar el control del SNES al puerto paralelo de la computadora.

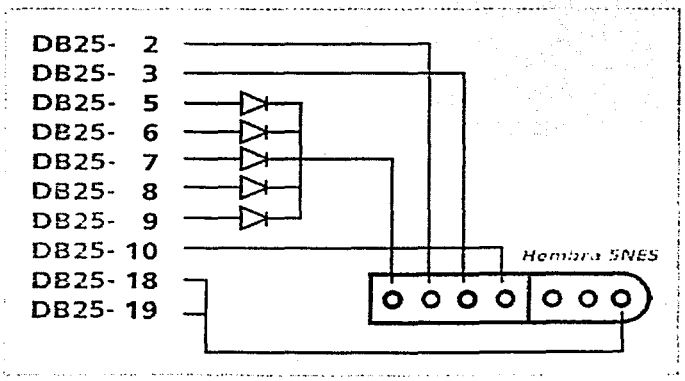


Figura B.3 Conexión del control de SNES a la computadora.

La línea 2 del puerto paralelo (dato 0) se conecta al pin 'reloj' del control de SNES. A través de esta línea del puerto se genera la señal de reloj que requiere el control para operar. La línea 3 del puerto paralelo (dato 1) se conecta al pin 'data latch' del control. A través de ésta línea se envía la señal de reset que necesita el control antes de leer el estado de los botones.

Las líneas 5, 6, 7, 8 y 9 del puerto paralelo (dato 3 – dato 7) son conectadas al pin de alimentación del control de SNES. Se coloca un diodo 1N4148 en cada pin del puerto para conseguir el voltaje y corriente necesarios que el control requiere para funcionar; además, estos diodos evitan que alguna corriente regrese hacia el puerto paralelo y lo dañe.

La línea 10 del puerto paralelo (ACK/) se conecta al pin 'dato serial' del control de SNES. A través de esta línea se recibirán serialmente los valores leídos del control, es decir, el estado de los botones.

Finalmente, se conecta la tierra del puerto paralelo con la tierra del control de SNES.

La visión final del montaje es la siguiente.[28]

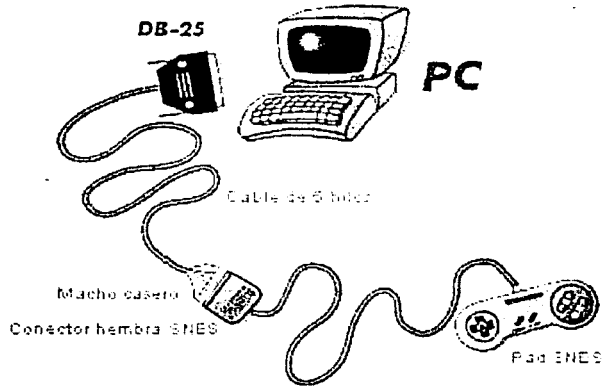


Figura B.4 Montaje final.

B.3.2 El SOFTWARE DE CONTROL

Una vez conectado el control de SNES al puerto paralelo de la PC, es necesario escribir un programa que lea el estado de los botones en el control.

El siguiente método, escrito en lenguaje C, se encarga de la lectura del control.

[28] Agradecemos la información proporcionada por emulatronia.com para conectar el control de SNES a la computadora.

```
void mySNES::scanPort(void) {

//Se genera la señal de Reset (Data Latch)
_outp(_DATA_, 0xFF);            waitMoment_2(_Time_);
                                waitMoment_2(_Time_);

//1er. Ciclo de reloj: Se lee el estado del botón B
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _B = 0;
else                            _B = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//2do. Ciclo de reloj: Se lee el estado del botón Y
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Y = 0;
else                            _Y = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//3er. Ciclo de reloj: Se lee el estado del botón SELECT
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Select = 0;
else                            _Select = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//4o. Ciclo de reloj: Se lee el estado del botón START
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Start = 0;
else                            _Start = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//5o. Ciclo de reloj: Se lee el estado del botón ARRIBA
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Up = 0;
else                            _Up = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//6o. Ciclo de reloj: Se lee el estado del botón ABAJO
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Down = 0;
else                            _Down = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//7o. Ciclo de reloj: Se lee el estado del botón IZQUIERDA
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Left = 0;
else                            _Left = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//8o. Ciclo de reloj: Se lee el estado del botón DERECHA
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Right = 0;
else                            _Right = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);
```

```
void mySNES::scanPort(void) {

//Se genera la señal de Reset (Data Latch)
_outp(_DATA_, 0xFF);            waitMoment_2(_Time_);
                                waitMoment_2(_Time_);

//1er. Ciclo de reloj: Se lee el estado del botón B
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _B = 0;
else                            _B = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//2do. Ciclo de reloj: Se lee el estado del botón Y
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Y = 0;
else                            _Y = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//3er. Ciclo de reloj: Se lee el estado del botón SELECT
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Select = 0;
else                            _Select = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//4o. Ciclo de reloj: Se lee el estado del botón START
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Start = 0;
else                            _Start = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//5o. Ciclo de reloj: Se lee el estado del botón ARRIBA
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Up = 0;
else                            _Up = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//6o. Ciclo de reloj: Se lee el estado del botón ABAJO
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Down = 0;
else                            _Down = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//7o. Ciclo de reloj: Se lee el estado del botón IZQUIERDA
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Left = 0;
else                            _Left = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//8o. Ciclo de reloj: Se lee el estado del botón DERECHA
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _Right = 0;
else                            _Right = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);
```

```
//9o. Ciclo de reloj: Se lee el estado del botón A
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _A = 0;
else                            _A = 1;press
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//10o. Ciclo de reloj: Se lee el estado del botón X
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _X = 0;
else                            _X = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//11o. Ciclo de reloj: Se lee el estado del botón L
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _L = 0;
else                            _L = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//12o. Ciclo de reloj: Se lee el estado del botón R
_outp(_DATA_, 0xFD);            waitMoment_2(_Time_);
if( _inp(_STATUS_) & 0x40)      _R = 0;
else                            _R = 1;
_outp(_DATA_, 0xFC);            waitMoment_2(_Time_);

//13o. Ciclo de reloj: Opcional
_outp(_DATA_, 0xFD);    waitMoment_2(_Time_);
_outp(_DATA_, 0xFC);    waitMoment_2(_Time_);

//14o. Ciclo de reloj: Opcional
_outp(_DATA_, 0xFD);    waitMoment_2(_Time_);
_outp(_DATA_, 0xFC);    waitMoment_2(_Time_);

//15o. Ciclo de reloj: Opcional
_outp(_DATA_, 0xFD);    waitMoment_2(_Time_);
_outp(_DATA_, 0xFC);    waitMoment_2(_Time_);

//16o. Ciclo de reloj: Opcional
_outp(_DATA_, 0xFD);    waitMoment_2(_Time_);
_outp(_DATA_, 0xFC);    waitMoment_2(_Time_);

_outp(_DATA_, 0xFD);
waitMoment_2(_Time_);

}
```

Ahora analicemos el método.

Primero se genera la señal de reset para el pin 'data latch'; enseguida, se construye la secuencia de ciclos de reloj. El primer ciclo de reloj comienza en nivel lógico alto y mantiene su valor durante un tiempo _Time_. A continuación se lee el valor o estado del botón, mientras ocurre la transición de alto a bajo en el ciclo de reloj. Según el valor leído, sabremos si el botón está

presionado o no. En este caso, el valor que envía el control de SNES al puerto paralelo, se captura en el pin 10 del puerto, es decir, en la línea de ACK/. Un nivel lógico alto presente en la línea ACK/ indicará que el botón no fue presionado, mientras que un nivel lógico bajo presente en la misma línea indicará que el botón fue presionado. Una vez realizada la lectura del botón, se genera la segunda mitad del ciclo de reloj; ésta segunda mitad mantendrá un nivel lógico bajo durante un tiempo _Time_. Se procede de la misma manera para leer el resto de los botones.

Los ciclos 13, 14, 15 y 16 también son generados como se indicó anteriormente, solo que durante estos ciclos no se realiza lectura del puerto paralelo. El programa termina colocando a nivel lógico alto la señal de reloj del control para cumplir con el protocolo de comunicación.

Como habrá notado, no se ha mencionado la duración de tiempo de los ciclos de reloj, ya que esta duración puede adecuarse según nuestras necesidades, es decir, qué tan rápido se desean tomar muestras del control. No es necesario que dicha duración sea la establecida en el protocolo, ya que ese protocolo es de comunicación entre el control de SNES y el CPU del SNES. En nuestro caso, el protocolo solo es utilizado para muestrear los botones del control en el orden apropiado. Algo similar ocurre con los ciclos 13, 14, 15 y 16 del reloj, los cuales pueden ignorarse, puesto que las funciones para las que fueron reservados no son utilizadas en nuestra aplicación.

Notas Adicionales sobre Nomenclatura:

Considere al puerto LPT1 como el puerto paralelo donde se conectó el control de SNES: la constante _DATA_, igual a 0x0378, es la dirección de datos para el puerto LPT1; la constante _STATUS_, igual a 0x0379, es la dirección de status; y la constante _CONTROL_, igual a 0x037A, es la dirección de control. Por otra parte, las variables _B, _Y, _Select, _Start, _Up, _Down, _Left, _Right, _A, _X, _L y _R, sirven para almacenar el estado de los botones del control.

La función _inp (puerto) lee un byte de datos del puerto de salida especificado. La función _outp (dato, puerto) escribe un byte de datos al puerto de salida especificado. El argumento puerto es un entero sin signo en el rango de 0 a 65535; y el argumento dato es un entero en el rango de 0 a 255. Ambas funciones están incorporadas en el lenguaje C.

Finalmente, la función waitMoment_2(tiempo) es una función definida por el usuario que ejecuta un retardo de tiempo; esta función se utiliza para generar la señal de reloj del control de SNES. La función recibe como argumento el tiempo de retardo deseado.

```
void mySNES::waitMoment_2(long wait) {
      while(wait--);
}
```

APÉNDICE C
MANUAL DE USUARIO DEL ROC2

C.1 REQUERIMIENTOS MÍNIMOS

- Procesador compatible con Intel a 400Mhz
- 64 MB de memoria RAM
- 20 MB de disco duro
- 2 MB de memoria de video
- Windows 98, NT o 2000
- DirectX 6.0 o superior
- Resolución de 800x600
- 16bits de color

C.2 INSTALACIÓN

- Ejecute el archivo Roc2setup.exe
- Siga las instrucciones que aparecen en pantalla

Durante el proceso de instalación se generó el archivo CODE.WSC el cual debe de ser enviado por correo electrónico a la dirección que el propio ROC2 le haya indicado. A vuelta de correo se le enviará su archivo llave (KEY.WSC) que deberá copiar al directorio de instalación para permitir el funcionamiento del sistema.

C.3 ENTORNO DE SISTEMA

El entorno de sistema del Roc2 se muestra en la figura C.1.

En la parte superior de la ventana del Roc2 se despliega el nombre de la aplicación, la versión y la fecha de elaboración del programa ejecutable (número 1 de la figura C.1).

El menú de sistema (número 2 de la figura C.1) permite al usuario interactuar con el ambiente virtual; haciendo uso del menú es posible realizar diferentes operaciones como más adelante se indicará.

El entorno de trabajo está formado por cuatro ventanas principales; tres de las ventanas que forman al entorno tienen como función mostrar distintas perspectivas de la simulación y la cuarta ventana es la llamada ventana de comandos.

Las tres perspectivas que se muestran durante la simulación son las siguientes:

- Vista libre. Esta perspectiva puede ser modificada por el usuario en todo momento, por lo que su posición y orientación no son actualizadas automáticamente (número 3 de la figura C.1).

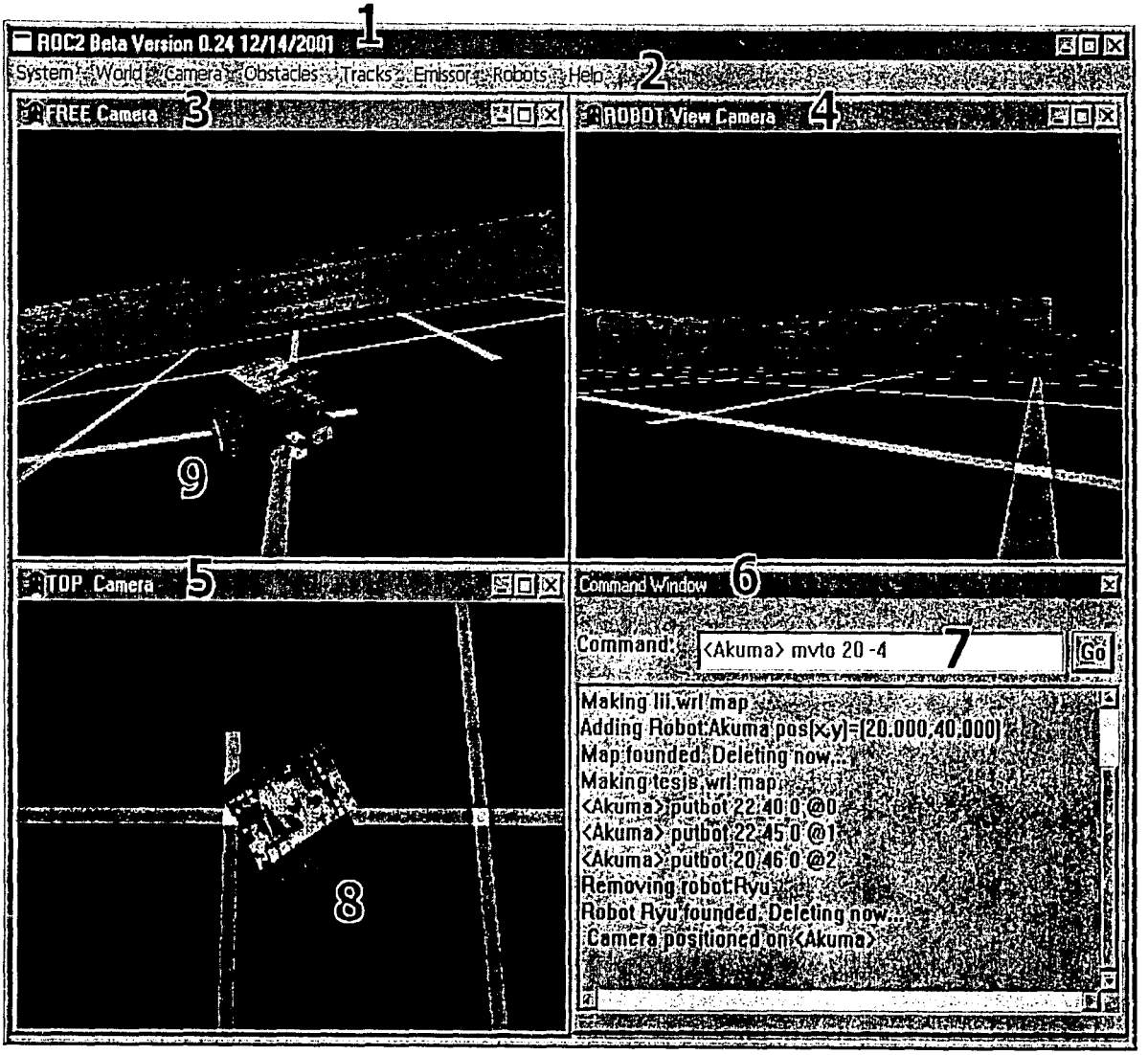


Figura C.1 Entorno de trabajo del Roc2.

- Vista del robot. Esta perspectiva nos muestra automáticamente lo que el robot está viendo en todo momento, por lo que no puede ser modificada por el usuario (número 4 de la figura C.1).

- Vista superior lejana. Esta perspectiva muestra en todo momento al robot desde una cámara fija que se encuentra en la parte superior del ambiente virtual. El usuario tiene control sobre la cámara únicamente desde el menú de sistema (número 5 de la figura C.1).

En la figura C.1, las ventanas de vista libre, vista del robot y vista superior están asociadas al robot indicado con los números 8 y 9.

La ventana de comandos (número 6 de la figura C.1) nos permite interactuar con los robots del Roc2 utilizando instrucciones de Robel. Las instrucciones son indicadas dentro del campo de texto (número 7 de la figura C.1) que se encuentra en la ventana de comandos.

Durante la ejecución, algunas instrucciones muestran datos propios de la instrucción y posteriormente su resultado; además la propia interfaz muestra algunos mensajes de sistema. Todos los mensajes son mostrados en la ventana de estado (figura C.2), o bien, en la parte inferior de la ventana de comandos.

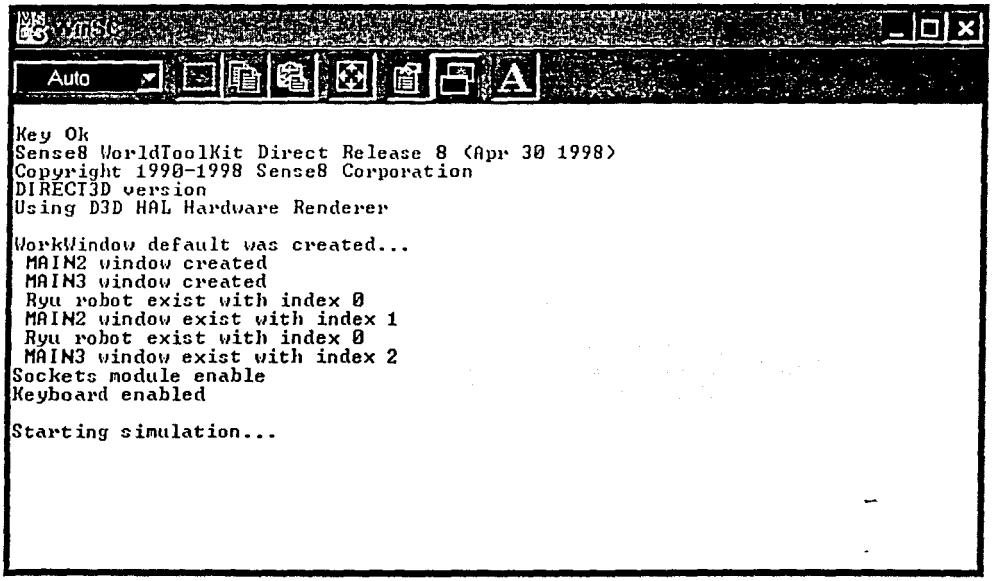


Figura C.2 Ventana de estado.

C.4 INTERACCIÓN CON EL SISTEMA

Existen diversas formas para interactuar con los robots móviles del Roc2. Las interfaces permiten controlar el sistema de forma local o remota. Las formas de interactuar con el Roc2 son las siguientes:

- Teclado y mouse. Interacción local. Estos dos dispositivos permiten controlar el Roc2 con toda su funcionalidad. Usando el teclado es posible asignar instrucciones de Robel a los robots, así como también, manipular los menús. Usando el mouse es posible cambiar la perspectiva de la cámara libre y también seleccionar un robot para interactuar con él.

  La presente sección utilizará la convención de ejes mostrada en la figura C.3.

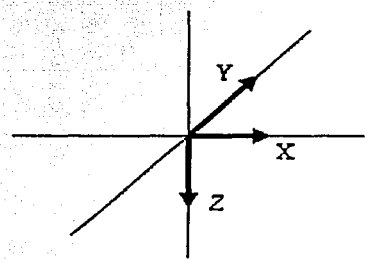


Figura C.3 Orientación de los ejes coordenados.

A continuación se listan los efectos de los eventos del mouse sobre el punto de vista (tabla C.1).

Evento del mouse	*Efecto sobre el punto de vista*
Ningún botón es presionado y el cursor está centrado en la ventana.	Ninguno
El botón izquierdo es presionado y el cursor se localiza arriba del centro de la ventana.	El punto de vista se mueve hacia adelante.
El botón izquierdo es presionado y el cursor se localiza abajo del centro de la ventana.	El punto de vista se mueve hacia atrás.
El botón izquierdo es presionado y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista rota hacia la izquierda sobre el eje Z.
El botón izquierdo es presionado y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista rota hacia la derecha sobre el eje Z.

El botón derecho es presionado y el cursor se localiza arriba del centro de la ventana.	El punto de vista se mueve hacia arriba.
El botón derecho es presionado y el cursor se localiza abajo del centro de la ventana.	El punto de vista se mueve hacia abajo.
El botón derecho es presionado y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista se desplaza hacia la izquierda.
El botón derecho es presionado y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista se desplaza hacia la derecha.
Los botones izquierdo y derecho son presionados y el cursor se localiza arriba del centro de la ventana.	El punto de vista rota hacia arriba sobre el eje X.
Los botones izquierdo y derecho son presionados y el cursor se localiza abajo del centro de la ventana.	El punto de vista rota hacia abajo sobre el eje X.
Los botones izquierdo y derecho son presionados y el cursor se localiza a la izquierda del centro de la ventana.	El punto de vista rota hacia la izquierda sobre el eje Y.
Los botones izquierdo y derecho son presionados y el cursor se localiza a la derecha del centro de la ventana.	El punto de vista rota hacia la derecha sobre el eje Y.

Tabla C.1 Efectos de los eventos del mouse sobre la cámara libre.

A continuación se listan los efectos del teclado sobre cualquiera de las perspectivas del sistema (ver tabla C.2).

Tecla	*Efecto sobre el robot que se encuentra seleccionado actualmente*
flecha arriba	avanza 1 unidad fundamental[29]
flecha abajo	retrocede 1 unidad fundamental
flecha izquierda	rota a la izquierda 10°
flecha derecha	rota a la derecha 10°

Tabla C.2 Efectos de teclado sobre las perspectivas de sistema.

- Internet. Interacción remota. Este método de interacción permite controlar al Roc2 a distancia por medio de Internet. Existen limitaciones para este método ya que no se cuenta con un control total del sistema; se limita a asignar instrucciones de Robel a los robots

[29] Una unidad fundamental equivale a 10 cm

existentes. Para hacer uso de este método, véase las instrucciones de Robel correspondientes a comunicación.

- SNES PAD. Interacción local. Este método de interacción permite controlar el sistema de forma limitada; sólo se permite controlar un solo robot asignándole comandos de movimiento básico. Los comandos de movimiento están asociados a la cruz direccional propia del pad como lo muestra la siguiente tabla.

Botón del SNES PAD	*Efecto sobre el robot que se encuentra asociado*
flecha arriba	avanza 1 unidad fundamental
flecha abajo	retrocede 1 unidad fundamental
flecha izquierda	rota a la izquierda 10°
flecha derecha	rota a la derecha 10°

Tabla C.3 Efectos del SNES PAD sobre un robot asociado.

C.5 MENÚS DEL ROC2

C.5.1 MENÚ SYSTEM

Este menú permite configurar el comportamiento que tiene el sistema. Dentro de este menú encontramos:

- Sensors
- Movement
- Exit

Sensors (sensores). Dentro del Roc2 es posible seleccionar el tipo de sensores que utilizarán los robots para la simulación. Existen dos opciones: Roc2, que son los sensores propios del sistema y User que son sensores simulados por el usuario. Es decir, Roc2 permite que el usuario cree su propio método de lectura de sensores.

Para hacer uso de los sensores de usuario es necesario contar con una biblioteca de enlace dinámico (DLL) que debe ser creada por el usuario mismo. El archivo DLL debe tener el nombre Roc2UserSensor.dll y debe encontrarse en el directorio de instalación del Roc2.

Dentro del archivo DLL se debe tener un método de exportación llamado *shs*, el cual tiene como función regresar el estado del sensor que se le solicita; sus especificaciones son las siguientes:

```
float shs(float *pq, char *name, int number);
```

donde:

pq	es un arreglo de números reales de punto flotante que contiene la posición y orientación del robot al momento de solicitar la lectura del sensor. El formato de pq es el siguiente:
pq[0]	posición en X del robot
pq[1]	posición en Y del robot
pq[2]	ángulo, en radianes, que tiene el robot con respecto al sistema
Name	nombre del tipo del sensor para el que se solicita su lectura. Como mínimo debe poderse obtener lecturas de los siguientes tipos de sensores: contact, reflective, infrared, sonar.
Number	identificador del sensor del que se requiere su lectura.

El valor que regresa shs debe ser la lectura del sensor.

Movement (movimiento). Dentro del Roc2 es posible seleccionar el tipo de movimientos que utilizarán los robots dentro de la simulación. Existen dos opciones: Roc2, que son los movimientos propios del sistema y User que son los movimientos calculados por el usuario. Es decir, Roc2 permite que el usuario modifique las rutinas de movimiento del robot.

Para hacer uso de los movimientos de usuario es necesario contar con una biblioteca de enlace dinámico que debe de ser creada por el usuario mismo. El archivo DLL debe tener el nombre Roc2UserMV.dll y debe de encontrarse en el directorio de instalación del Roc2.

Dentro del archivo DLL se deben tener dos funciones de exportación: getMaxSteps y getBotNextPosition.

getMaxSteps. Indica al sistema en cuántos pasos se realizará el próximo movimiento. El encabezado de la función es el siguiente:

```
int getMaxSteps(float *p1, float *p2)
```

donde:

p1	contiene la posición actual del robot en el formato: (p1[0], p1[1]) = (X, Y)
p2	contiene la posición final (deseada) del robot en el formato: (p2[0], p2[1]) = (X, Y)

La función debe de regresar el número de pasos que tardará el sistema para trasladar al robot desde el punto p1 al punto p2.

getBotNextPosition. Solicita la posición del robot para el n-ésimo paso de su movimiento. El encabezado de la función es el siguiente:

```
void getBotNextPosition(robotX r, int step, float *p)
```

donde:

r es una estructura de tipo robotX.

La estructura robotX contiene información del estado actual del robot y su forma es la siguiente:

```
typedef struct robot_description{
        char Name[SizeChar];
        coordinates Position;
        sensorX  *Sonar;
        sensorX  *Contact;
        sensorX  *Infrared;
        sensorX  *Reflective;
        actuator *MotorDc;
        int NumSonars;
        int NumContacts;
        int NumInfrareds;
        int NumReflectives;
        int NumMotorsDc;
} robotX;
```

donde:

Name es el nombre del robot

Position es una estructura de tipo *coordinates* que contiene la posición y orientación del robot.

Sonar es un arreglo de sensores donde cada elemento contiene la información del estado de cada sonar; el número de elementos del arreglo está dado por *NumSonars*.

Contact es un arreglo de sensores donde cada elemento contiene la información del estado de cada sensor de contacto; el número de elementos del arreglo está dado por *NumContacts*.

Infrared es un arreglo de sensores donde cada elemento contiene la información del estado de cada sensor infrarrojo; el número de elementos del arreglo está dado por *NumInfrareds*.

Reflective es un arreglo de sensores donde cada elemento contiene la información del estado de cada sensor reflectivo; el número de elementos del arreglo está dado por *NumReflectives*.

MotorDc es un arreglo de estructuras de tipo *actuator*, donde cada elemento contiene información acerca de la posición actual de los motores; el número de motores está dado por *NumMotorsDc*.

```
typedef struct coordinate{
        double x;
        double y;
        double Angle;
}coordinates;
```

donde:

x, y	determinan la posición actual del robot dentro del sistema
angle	determina la orientación del robot dentro del sistema

```
typedef struct sensorsX{
        int   Port;
        int   Mask;
        float Value;
        float PositionX;
        float PositionY;
        float PositionZ;
} sensorX;
```

donde:

Port, Mask	no son utilizados en esta versión. Contienen NULL
Value	contiene el estado del sensor
PositionX	posición relativa del sensor con respecto al robot (en X)
PositionY	posición relativa del sensor con respecto al robot (en Y)
PositionZ	posición relativa del sensor con respecto al robot (en Z)

```
typedef struct actuators{
        int Port;
        int Mask;
        float PositionX;
        float PositionY;
        float PositionZ;
        float value;
} actuator;
```

donde:

Port, Mask	no son utilizados en esta versión; contienen NULL
Value	contiene el estado del paso del motor
PositionX	junto con *PositionY* y *PositionZ* establecen la posición relativa del motor con respecto al robot.

Exit. Finaliza la ejecución de la aplicación.

C.5.2 MENÚ WORLD

Este menú permite configurar algunas características relativas al ambiente virtual donde se desarrolla la simulación. Dentro de este menú encontramos:

- Load
- Wired Frame
- Solid Colors

- Texturized

Load (carga). Este elemento del menú permite cargar un ambiente virtual. Llamaremos mapa a dicho ambiente virtual. Al seleccionar esta opción la siguiente ventana de diálogo aparecerá (ver figura C.4):

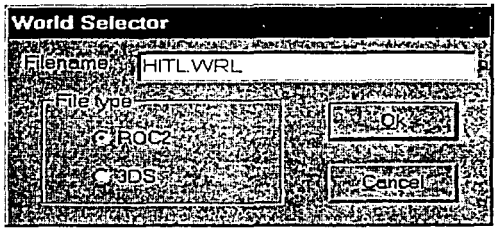


Figura C.4 Cuadro de diálogo de selección de mapa.

Es necesario indicar el nombre del archivo del mapa sin extensión, y también el tipo; una vez indicados se debe presionar el botón *Ok*. Para cancelar la operación de carga de mapa se debe de presionar el botón *Cancel*.

Existen dos tipos de mapas que pueden ser cargados dentro del Roc2:

- Roc2
- 3Ds

Véase el apartado de mapas para más detalles de ambos formatos, en la página 179.

Los archivos de mapas se encuentran en el subdirectorio *maps* del directorio de instalación del Roc2. Los mapas en formato 3Ds deben de ser guardados en el subdirectorio *3Ds* del directorio *maps*.

Los mapas en formato Roc2 deben de ser guardados en el subdirectorio *Roc2* del directorio *maps* y deben tener la extensión *.wrl*. Las texturas asociadas a los mapas deben guardarse en el subdirectorio *textures* del directorio de instalación.

Sólo es posible tener un mapa a la vez, de manera que si desea cargar otro mapa, el anterior será descargado automáticamente.

Wired Frame (modelo de alambres). Este elemento del menú permite modificar el render del ambiente virtual mostrando todos los elementos en modelo de alambres. Véase figura C.5.

Solid colors (colores sólidos). Este elemento del menú permite modificar el render del ambiente virtual mostrando todos los elementos como polígonos rellenos. Véase figura C.6.

Texturized (texturizado). Este elemento del menú permite modificar el render del ambiente virtual mostrando todos los elementos con textura e iluminación. Véase figura C.7.

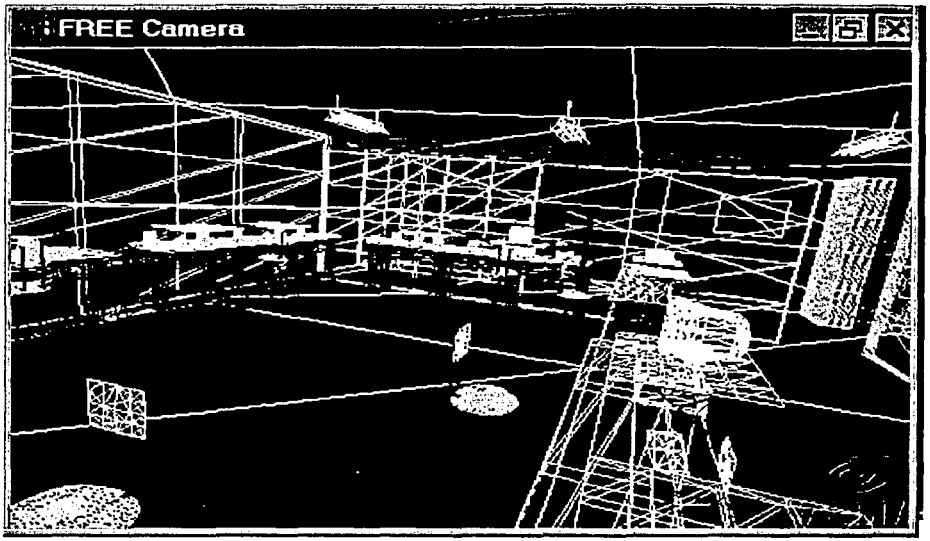


Figura C.5 Modelo de alambres del mundo virtual.

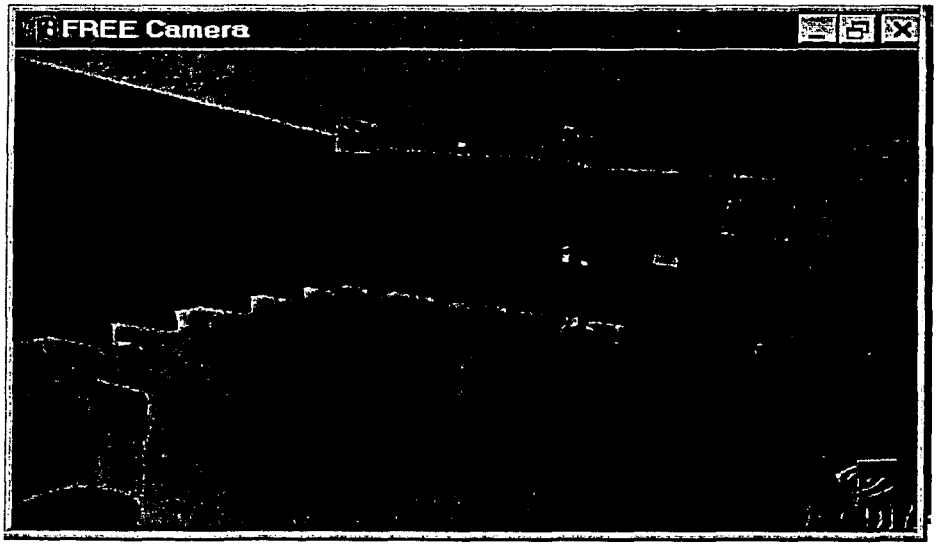


Figura C.6 Modelo en colores sólidos del mundo virtual.

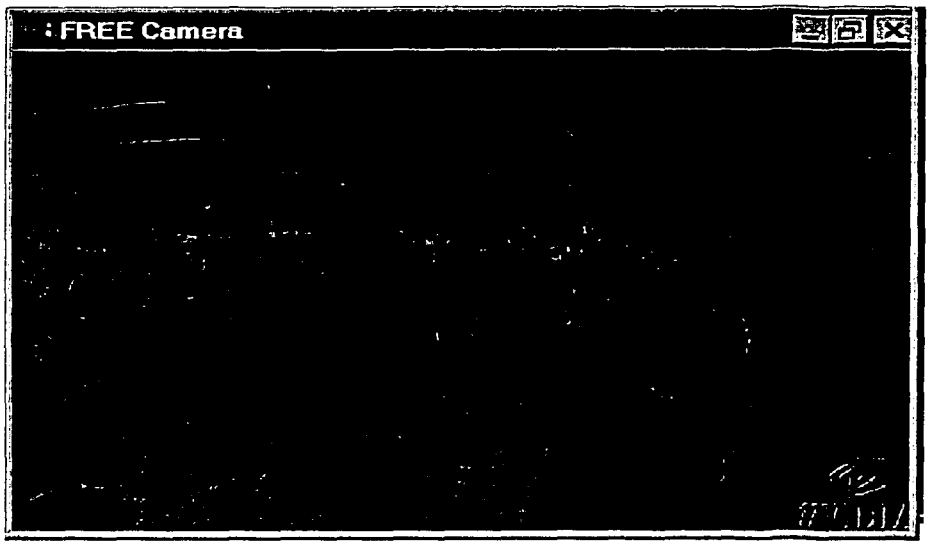


Figura C.7 Modelo con texturas del mundo virtual.

C.5.3 MENÚ CAMERA

Este menú permite configurar la posición de la cámara fija dentro del ambiente virtual. Dentro de este menú encontramos:

- Set Top Camera (fijar cámara superior)

Set Top Camera. Este elemento del menú permite modificar la posición de la cámara asociada a la perspectiva lejana de la simulación. Al seleccionar esta opción aparecerá el siguiente cuadro de diálogo (ver figura C.8):

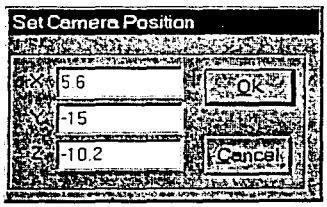


Figura C.8 Cuadro de diálogo de cámara superior.

Es necesario indicar las componentes X, Y, Z de la posición de la cámara y luego presionar el botón *Ok.* Para cancelar la modificación de la posición de la cámara se presiona el botón *Cancel.*

C.5.4 MENÚ OBSTACLES

Usando este menú es posible añadir y remover distintos tipos de obstáculos dentro del Roc2. Existen dos tipos de obstáculos: dinámicos y estáticos. Dentro de los estáticos encontramos árboles y cajas; en el caso de los dinámicos, tenemos bombas que explotan después de 3 colisiones.

- Add (añadir)
  - Dinamic (dinámico)
  - Tree (árbol)
  - Box (caja)
- Remove (remover)

Dinamic. Este elemento del menú permite cargar un obstáculo dinámico del tipo bomba. Al seleccionar esta opción, aparecerá el siguiente cuadro de diálogo (ver figura C.9):

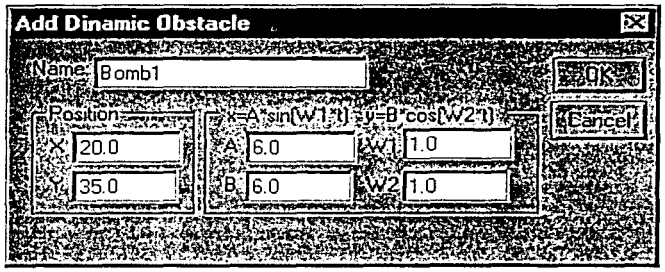


Figura C.9 Cuadro de diálogo para insertar obstáculos dinámicos.

En el campo *Name* (nombre) se especifica un nombre que identifica al obstáculo dentro del Roc2. También es necesario indicar las componentes X, Y de la posición de la bomba, además de los parámetros a, b, w1 y w2 que permiten modificar la trayectoria que sigue la bomba según las siguientes ecuaciones (véase el apartado 5.3.2.2 para mayor referencia):

$$x = a \operatorname{sen}(w_1 t)$$
$$y = b \cos(w_2 t)$$

Tree. Este elemento del menú permite cargar un obstáculo estático del tipo árbol. Al seleccionar esta opción, aparecerá el siguiente cuadro de diálogo (ver figura C.10):

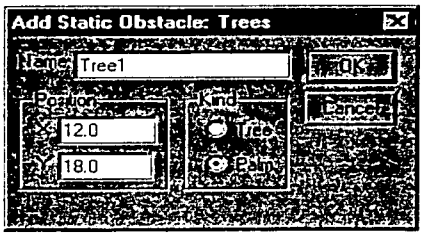


Figura C.10 Cuadro de diálogo para insertar obstáculos estáticos del tipo árbol.

En el campo *Name* (nombre) se especifica el nombre que identifica al obstáculo dentro del Roc2. La posición de obstáculo dentro del ambiente virtual se especifica por medio de las componentes X y Y. Por último, el campo Kind permite elegir algún tipo de árbol en particular (Tree -árbol- o Palm -palma-).

Box. Este elemento del menú permite cargar un obstáculo estático del tipo caja. Al seleccionar esta opción, aparecerá el siguiente cuadro de diálogo (ver figura C.11):

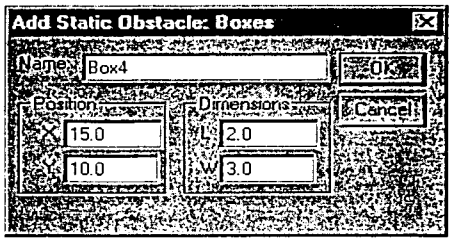


Figura C.11 Cuadro de diálogo para insertar obstáculos estáticos del tipo Caja.

En el campo *Name* (nombre) se especifica el nombre que identifica al obstáculo dentro del Roc2. Es necesario indicar las componentes X, Y de la posición de la caja, así como también sus dimensiones.

Remove. Este elemento del menú permite remover un obstáculo que se encuentra cargado actualmente. Al seleccionar esta opción la siguiente ventana de diálogo aparecerá (ver figura C.12):

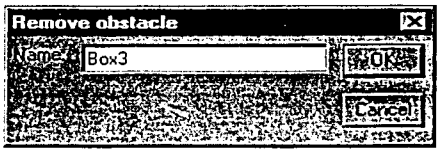


Figura C.12 Cuadro de diálogo para remover obstáculos.

En el campo *Name* (nombre) se indica el nombre del obstáculo que se desea remover. Una vez que se ha indicado el nombre del obstáculo es necesario presionar el botón *OK*. Para cancelar la acción de remover un obstáculo se debe presionar el botón *Cancel*.

C.5.5 MENÚ TRACKS

Este menú permite añadir y remover pistas dentro del ambiente virtual. Las pistas son usadas en conjunto con los sensores reflectivos. Dentro de este menú encontramos:

- Load (cargar)
- Remove (remover)

Load. Este elemento del menú permite cargar una pista dentro del ambiente virtual. Al seleccionar esta opción, aparecerá el siguiente cuadro de diálogo (ver figura C.13):

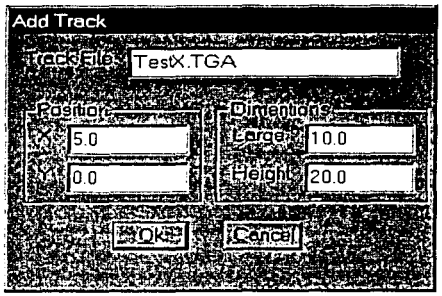


Figura C.13 Cuadro de diálogo para añadir pistas.

Al crear una pista es necesario contar con un archivo de imagen en formato TGA de 8 bits sin compresión. El archivo debe estar guardado en el subdirectorio *tracks* que se encuentra en el directorio de instalación.

En el campo *Track File* (archivo de pista) se debe de indicar, con extensión, el nombre del archivo de imagen que se utilizará para crear la pista.

En los campos de posición, *X* y *Y*, se indica la posición donde se colocará la pista dentro del ambiente virtual con respecto al origen del sistema de coordenadas absoluto. En los campos de dimensión (*Large* y *Height*) se indica el tamaño que tendrá la pista dentro del ambiente virtual.

Una vez llenos todos los campos del cuadro de diálogo es necesario presionar el botón Ok. Para cancelar la carga de una pista se debe presionar el botón Cancel. Sólo es posible cargar una pista a la vez, por lo que si se desea añadir otra pista primero debe removerse la actual.

Remove. Este elemento del menú permite remover la pista que se encuentra cargada actualmente.

C.5.6 MENÚ EMISSOR

Este menú permite hacer operaciones con los dispositivos emisores, los cuales pueden ser cargados dentro del ambiente virtual. Dentro de este menú encontramos la opción:

- Infrared (infrarrojo)

Infrared. Este elemento está asociado a emisores de infrarrojo y consiste de dos puntos:

- Add (añadir)
- Remove (remover)

Add. Este elemento del menú permite añadir a la simulación un emisor de infrarrojo. Al seleccionar esta opción, aparecerá el siguiente cuadro de diálogo (ver figura C.14):

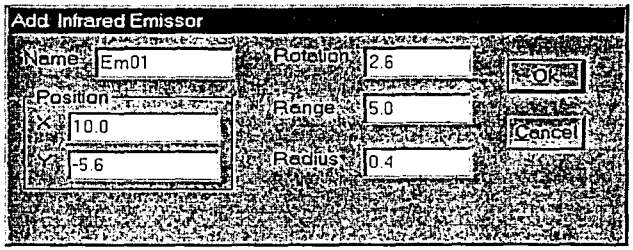


Figura C.14 Cuadro de diálogo para añadir emisores de infrarrojo.

En el campo *Name* (nombre) se especifica un nombre que identifica al emisor de infrarrojo dentro del sistema. Por otra parte, los campos de posición X y Y indican la posición donde se colocará el emisor dentro del ambiente virtual. Recuerde que la posición es con respecto al origen del sistema de coordenadas absoluto.

El campo *rotation* (rotación) establece la orientación del rayo infrarrojo dentro de la simulación. Los valores válidos para este campo están en el rango de 0 a 2π radianes.

El campo *range* (rango) determina el alcance del rayo infrarrojo a partir del centro del emisor.

El campo *radius* (radio) determina la amplitud del cono, en radianes, que se forma con el rayo emisor. Su valor debe mantenerse en el rango de (0,1). Para valores fuera de rango se tomará el valor máximo o mínimo más próximo, según sea el caso.

Una vez llenos todos los campos del cuadro de diálogo, es necesario presionar el botón *Ok*. Para cancelar la acción de añadir un emisor de infrarrojo se debe presionar el botón *Cancel*. Es posible tener hasta 4 emisores infrarrojos dentro de la simulación.

Remove. Este elemento del menú permite remover un emisor que se encuentra cargado actualmente. Al seleccionar esta opción, el siguiente cuadro de diálogo aparecerá (ver figura C.15):

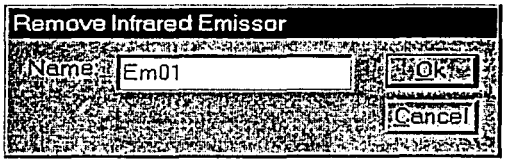


Figura C.15 Cuadro de diálogo para remover un emisor.

En el campo *Name* (nombre) se indica el nombre del emisor que se desea remover del mundo virtual. Una vez que se ha indicado el nombre de un emisor es necesario presionar el botón *Ok*. Para cancelar la acción de remover un emisor se debe presionar el botón *Cancel*.

C.5.7 MENÚ ROBOTS

En este menú se encuentran las operaciones que permiten manipular a los robots móviles de la simulación. Dentro de este menú encontramos las siguientes opciones:

- Add (añadir)
- Remove (remover)

Add. Este elemento del menú permite añadir a la simulación un robot. Al seleccionar esta opción, el siguiente cuadro de diálogo aparecerá (ver figura C.16):

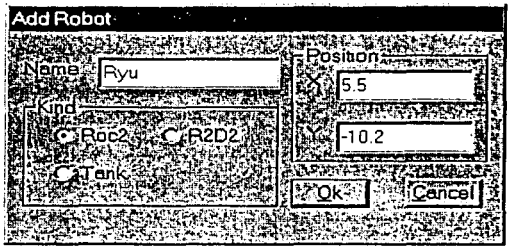


Figura C.16 Cuadro de diálogo para añadir un robot.

En el campo *Name (nombre)* se debe indicar el nombre que tendrá el robot dentro de la simulación; dicho nombre servirá como identificador del robot.

El campo *kind* (clase) permite los siguientes valores:

- Roc2
- R2D2
- Tank (tanque)

Cada uno de los valores corresponde a un modelo de robot distinto, si bien cuentan con la misma cantidad de sensores, cada uno tiene diferente forma y dimensiones. La figura C.17 muestra los 3 modelos de robot existentes.

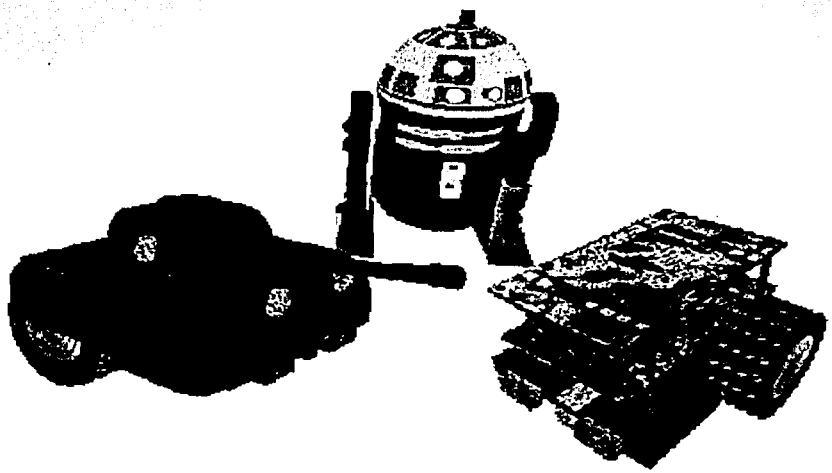


Figura C.17 Robots del Roc2. De derecha a izquierda: Roc2, R2D2 y Tank.

En el campo *position (posición) X* y *Y* se indica la posición donde se colocará el robot dentro del ambiente virtual. Recuerde que la posición es con respecto al sistema de coordenadas absoluto.

Una vez llenos todos los campos del cuadro de diálogo es necesario presionar el botón *Ok*. Para cancelar la acción de añadir un robot se debe presionar el botón *Cancel*. El máximo número de robots que pueden ser cargados en el sistema es de 5.

Remove. Este elemento del menú permite remover un robot que se encuentra cargado actualmente. Al seleccionar esta opción, el siguiente cuadro de diálogo aparecerá (ver figura C.18):

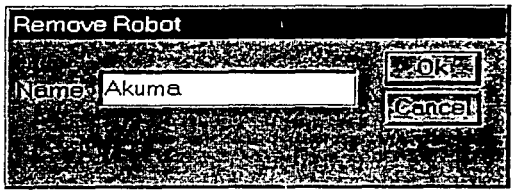


Figura C.18 Cuadro de diálogo para remover un robot.

En el campo *Name (nombre)* se indica el nombre del robot que se desea remover del mundo virtual. Una vez que se ha indicado el nombre de un robot es necesario presionar el botón *Ok*. Para cancelar la acción de remover un robot se debe presionar el botón *Cancel*.

Al menos un robot debe de existir dentro del Roc2, por lo que es imposible remover de la simulación a todos los robots.

C.5.8 MENÚ HELP

Help (ayuda). En este menú se encuentra la información de soporte del sistema.

C.6 ROBEL

Robel es un lenguaje de programación orientado al control de un robot móvil. Desde un principio se planeó como una herramienta para llevar a cabo el control de forma rápida y sencilla haciendo uso de instrucciones. Sus características principales son:

- Instrucciones para el control de movimiento
- Instrucciones para la lectura del estado lógico del robot
- Instrucciones para comunicación vía Internet
- Control de flujo del programa haciendo uso de ciclos while, if y saltos a etiquetas
- Soporta dos tipos de variables: enteros y flotantes
- Independencia de la plataforma
- Posibilidad de tener recursividad
- Llamadas a funciones creadas por el usuario (otros scripts)

Para que un programa escrito en Robel se pueda ejecutar sobre distintas plataformas, es necesario hacer uso de un programa que traduzca cada instrucción de Robel al lenguaje propio de cada sistema; es decir, se debe contar con la Máquina Virtual de Robel (MVR). Cada uno de los robots del Roc2 contiene su propia máquina virtual.

El acumulador forma parte de la MVR y su propósito es guardar datos para el manejo interno de la ejecución de instrucciones, así como también para almacenar el resultado de la ejecución de

algunas instrucciones. Es decir, el acumulador es una estructura de datos utilizada para compartir información por medio de cada una de las etapas de ejecución de las instrucciones. La estructura del acumulador se muestra en el cuadro C.1.

Registro	*Tipo*	*Nombre*	*Mnemónico en Robel*
F1	float	registro flotante 1	RegF1
F2	float	registro flotante 2	RegF2
F3	float	registro flotante 3	RegF3
I1	integer	registro entero 1	RegI1
I2	integer	registro entero 2	RegI2
S	string	registro string	RegS
Stack	pila de flotantes	pila	RegSp

Cuadro C.1 Registros de la máquina virtual.

Sólo puede hacerse uso de los registros de la máquina virtual dentro un script y no es posible modificar su contenido. Para hacer referencia a un registro sólo hay que usar el mnemónico asociado.

Las instrucciones de Robel pueden ser ejecutadas por el sistema mediante dos formas distintas: un script o introduciéndolas en línea usando la ventana de comandos. Si desea utilizar un script, éste es compilado y luego cargado en el sistema; en el caso de la entrada en línea, solamente se interpreta. Para ambos casos las instrucciones deben tener el formato indicado en el cuadro C.2. Los parámetros entre corchetes son opcionales.

Formato para instrucción
[<nombre>] instrucción [parámetros] @n

Cuadro C.2 Formato de instrucción para Robel.

El formato de instrucción de Robel se divide en cuatro partes, las cuales se describen a continuación:

- <nombre>: este parámetro es utilizado para indicar el nombre del robot que ejecutará la instrucción. Algunas instrucciones son independientes de la existencia de un robot y en esos casos el parámetro no es necesario.
- instrucción: instrucción a ser ejecutada.
- parámetros. parámetros asociados a la instrucción.
- @n: código de control. Para llevar un control de las instrucciones asignadas, ya sea en scripts o vía Internet, es necesario añadir un código de control denotado por @ y seguido

por un identificador numérico (n) preferiblemente progresivo. Esto tiene como finalidad conocer qué instrucción se está ejecutando, y en el caso de ser un comando remoto, enviar el código de la instrucción al host (anfitrión) como mensaje de fin de ejecución. Si el comando es introducido a través de la ventana de comandos, no es necesario añadir el código de control ya que éste es asignado automáticamente.

La siguiente tabla muestra el conjunto de instrucciones que son soportadas por el Roc2. Véase Cuadro C.3.

Instrucción	*Categoría*	*Sintaxis*	*Ejemplo*	*Descripción*
left	movimiento	left	left	Movimiento rotatorio a la izquierda de 10°
right	movimiento	right	right	Movimiento rotatorio a la derecha de 10°
forward	movimiento	forward	forward	Movimiento de traslación de 1 unidad fundamental hacia adelante
backward	movimiento	backward	backward	Movimiento de translación de 1 unidad fundamental hacia atrás
mv	movimiento	mv distancia ángulo [tiempo]	mv 10 3.4	Movimiento de rotación dado por *ángulo*, seguido de una traslación dada por *distancia*
mvto	movimiento	mvto posX posY [tiempo]	mvto –5.3 2.7 10	Traslación hacia la posición dada por (posX, posY)
loads	script	loads nombre_script	loads test.txt	Carga el script *nombre_script* en la máquina virtual
script	script	script nombre_script	script test.txt	Ejecuta el script *nombre_script* que debe estar cargado en la máquina virtual
ascript	script	ascript	ascript	Aborta la ejecución de un script
opsk	comunicaciones	opsk r/w hostname puerto	opsk w localhost 2000	Establece comunicaciones de entrada (r) o salida (w)
ksk	comunicaciones	ksk r/w	ksk w	Cierra comunicación de entrada (r) o salida (w)
ic	estado	ic posX posY ángulo	ic –5 5 3.1	Establece la posición actual del robot (cambia el origen del sistema de coordenadas relativo al robot)
sc	estado	sc	sc	Muestra la posición y orientación del robot
shs	estado	shs nombre_sensor número	shs reflective 1	Obtiene una lectura del sensor *número*, cuyo tipo es *nombre_sensor*
tap	estado	tap	tap	El robot captura una imagen de lo que está observando en ese instante

Cuadro C.3 Lista de instrucciones de Robel.

C.7 MAPAS

Los ambientes virtuales dentro del Roc2 se cargan haciendo uso de mapas. Un mapa es un archivo de texto que describe los elementos que contendrá el entorno virtual, también indica la posición de los elementos y otras características. Llamaremos a este tipo de mapas formato *mroc2*.

Algunas de las ventajas que se tienen al usar mapas mroc2 son:

- Buen desempeño del sistema durante la simulación
- No hay necesidad de un modelador 3D para crear un entorno virtual
- Las modificaciones al entorno son precisas

Usar un archivo mroc2 tiene sus desventajas. La principal desventaja reside en la precisión de los datos que se utilizan para crear el ambiente virtual; dichos datos son calculados por el usuario y la modificación de las dimensiones del elemento, su posición u orientación, significan en la mayoría de los casos, modificar todos los datos que forman al elemento.

Además del uso de mapas mroc2, el Roc2 es capaz de utilizar archivos en formato 3Ds (3Dstudio) para poder crear un entorno virtual.

Algunas de las ventajas que se tienen al usar mapas en formato 3Ds son:

- Creación de ambientes virtuales complejos y con gran detalle, de forma sencilla
- Uso de una herramienta 3D para crear el entorno virtual

Usar un archivo 3Ds para crear un ambiente virtual también tiene desventajas. La desventaja principal se refleja en el rendimiento del sistema; un archivo de 3Ds generalmente está formado por cientos o miles de polígonos, los cuales disminuyen el rendimiento final del sistema significativamente. El archivo de 3Ds también tiene un sistema de coordenadas distinto al del Roc2 por lo que crear un ambiente preciso puede ser complicado.

C.7.1 FORMATO DE MAPAS MROC2

Un archivo mroc2, está formado por directivas. Las reglas para indicar las directivas son las siguientes:

- Debe de estar contenida entre paréntesis
- Se debe indicar a qué mundo virtual pertenece
- Sólo es posible definir una directiva por línea
- Debido a la implementación, cada directiva tiene que ser de 255 caracteres como máximo.
- Si los datos de la directiva son erróneos o incompletos, se ignorará la directiva

Un archivo mroc2 puede contener comentarios, para ello sólo hay que colocar un punto y coma (;) al inicio de la línea.

Existen 3 directivas que todo archivo mroc2 debe tener. Las directivas son las siguientes:

- limit_area (límite del contorno)
- dimensions (dimensiones)
- middle-point (punto medio)

limit-area. Indica la posición de cada uno de los vértices que forman el límite o contorno del ambiente virtual. Los vértices se deben especificar siguiendo el sentido de las manecillas del reloj. Además de los límites del ambiente, es necesario indicar el nombre del ambiente. El siguiente es un ejemplo de esta directiva:

```
(limit_area   LII  0.00  0.00  0.00 77.00 74.50 77.00 74.50  0.00)
```

En este caso, el nombre del ambiente es `LII` y sus dimensiones están dadas por los vértices que le siguen. Estos vértices forman un polígono cuadrado cuyos vértices son:

```
v1 = ( 0.00, 0.00 )
v2 = ( 0.00, 77.00 )
v3 = ( 74.50, 77.00 )
v4 = ( 74.50, 0.00 )
```

dimensions. Indica el largo y el ancho del ambiente virtual. El siguiente es un ejemplo de esta directiva:

```
(dimensions LII 74.50 77.00)
```

En este ejemplo se definieron las dimensiones para el ambiente virtual `LII`; los valores de largo y ancho aparecen después del nombre del ambiente.

middle-point. Contiene el punto medio del ambiente virtual. El siguiente es un ejemplo de esta directiva:

```
(middle-point LII 5. 3.)
```

En este ejemplo se definió el punto medio para el ambiente virtual `LII`.

C.7.2 CREACIÓN DE ELEMENTOS GEOMÉTRICOS

Para crear objetos dentro del ambiente virtual se utiliza la directiva *polygon (polígono).* El formato de esta directiva permite especificar los vértices que forman parte de la base de un prisma recto. El formato de la directiva *polygon* se describe a continuación:

```
(polygon tipo ambiente cuarto vertex1 vertex2....)
```

donde:

tipo	indica el tipo de elemento geométrico que se está creando. Usando este parámetro es posible indicarle al sistema que el elemento a crear se encuentra en la clasificación indicada;
cuarto	indica a qué entorno virtual pertenece el elemento;
nombre	indica el nombre del elemento dentro del mundo virtual;
vertex	especifica los vértices del polígono que forman al objeto, ordenados siguiendo el sentido de las manecillas del reloj.

Utilizando las propiedades tipo y nombre podremos crear una organización lógica para representar nuestro ambiente virtual. Es posible crear cuartos diferentes con el mismo nombre del elemento; por ejemplo, podríamos especificar un elemento silla1 para la cocina y también un elemento silla1 para la sala.

```
(polygon cocina   LII silla1  1.50 22.90  1.50 30.50 32.00 30.50 32.00 22.90)
(polygon sala     LII silla1  1.50 31.50  1.50 61.40  8.90 61.40  8.90 31.50)
```

Por último, también es posible añadir puntos de referencia dentro de los mapas del Roc2. El formato de la directiva para los puntos de referencia es el siguiente:

```
(reference-point cuarto posX posY)
```

donde:

cuarto	indica el nombre del cuarto al que pertenece el punto de referencia;
posX, posY	indica la posición del punto de referencia dentro del cuarto;

Finalmente, es importante mencionar que los archivos de descripción utilizados en Roc2 también son utilizados en un sistema experto CLIPS, para tener una representación simbólica del medio ambiente en donde se navega.

El siguiente es un ejemplo de mapa en formato mroc2, que corresponde al espacio existente del Laboratorio de Interfaces Inteligentes:

```
;*****************************************************************
;* File:    hitl.txt                                             *
;* Purpose: Definition of the forbidden and allowed areas in the Robot's  *
;*          world. These areas are derivated from the objects in the      *
;*          world.                                                        *
;*****************************************************************

( limit_area      LII 0.00 0.00 0.00 77.00 74.50 77.00 74.50 0.00)
( dimensions      LII 74.50 77.00)
( middle-point    LII 5. 3.)
( reference-point LII 10.0 10.0)
( reference-point LII 60.0 10.0)
( reference-point LII 20.0 50.0)
```

```
( reference-point LII 60.0 50.0)

( polygon object-wall  LII mesa_1   1.50  22.90 1.50  30.50 32.00 30.50 32.00 22.90 )
( polygon corners      LII mesa_1   35.00 32.50 35.00 20.90                          )
( polygon object-wall  LII mesa_2   1.50  31.50 1.50  61.40 8.90  61.40 8.90  31.50 )
( polygon object-wall  LII mesa_3   6.40  68.20 6.40  75.50 36.40 75.50 36.40 68.20 )
( polygon object-wall  LII mesa_4   42.00 68.20 42.00 75.50 73.00 75.50 73.00 68.20 )
( polygon object-wall  LII mesa_5   65.40 31.20 65.40 61.60 73.00 61.60 73.00 31.20 )
( polygon object-wall  LII mesa_6   42.70 22.90 42.70 30.50 73.00 30.50 73.00 22.90 )
( polygon corners      LII mesa_6   40.70 20.90 40.70 32.50                          )
( polygon object-wall  LII mesa_7   36.50 45.50 36.50 74.30 42.50 74.30 42.50 45.50 )
( polygon corners      LII mesa_7   34.50 43.50 44.50 43.50                          )
( polygon object-wall  LII est_1    6.40  67.50 1.50  67.50 1.50  75.00 6.40  75.00 )
( polygon object-wall  LII est_2    65.40 66.30 73.00 66.30 73.00 61.60 65.40 61.60 )
( polygon wall         LII pared1   0.00  0.00  0.00  1.50  74.50 1.50  74.50 0.00  )
( polygon wall         LII pared2   0.00  0.00  0.00  77.00 1.50  77.00 1.50  0.00  )
( polygon wall         LII pared3   0.00  75.50 0.00  77.00 74.50 77.00 74.50 75.50 )
( polygon wall         LII pared4   74.50 9.00  73.00 9.00  73.00 77.00 74.50 77.00 )
```

El ambiente virtual creado usando este mapa se muestra a continuación. Figura C.19.

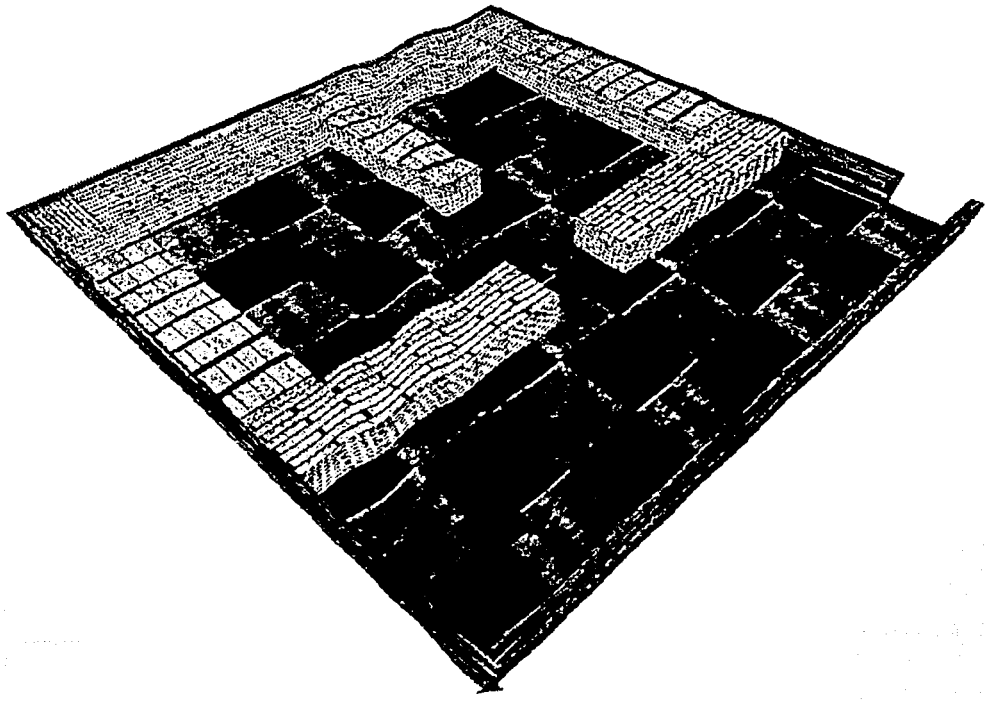


Figura C.19 Cuarto formado por el archivo hitl.txt

BIBLIOGRAFÍA

[1] Cox, I.J.; Wilfong G.T.
Autonomus robot vehicles
Edit. Springer-Verlag. 1a. Edición. USA, 1990.

Empleo de la referencia:

1.2 Robots móviles

[2] MSDN ibrary Visual Studio 6.0
Visual C++ 6.0. Microsoft Corporation. USA, 1998.

Empleo de la referencia:

2.9 Creación de una aplicación de WTK
3.4 Sensores
4.1.1 Representación de orientaciones en WTK
4.2 Puntos de vista en Roc2
4.3.1 Selección de objetos en Roc2
A Bibliotecas de enlace dinámico

[3] World Tool Kit Reference manual Release 8
Manual de Referencia.
Engineering Animation Inc. 1a. Edición. USA, 1999.

Empleo de la referencia:

Cap.II World Tool Kit
3.2 Robot command center
3.3 Creación de robots
3.4 Sensores
3.5 Funcionamiento del Roc2
4.1 Ventanas y puntos de vista
4.2 Puntos de vista en Roc2
4.3.1 Selección de objetos en Roc2
5.2.1 Árbol jerárquico para la entidad árbol
5.3.1 Árbol jerárquico para la entidad bomba
5.3.2 Tareas asociadas a la entidad bomba

[4] Foley, James; van Dam, Andries; Feiner,Steven; Hughes, John.
Computer Graphics: Principles and practice, Second edition in C
Edit. Addison-Wesley. 2a. Edición. USA, 1990.

Empleo de la referencia:

3.1 Introducción
3.3 Creación de robots
3.4.1 Sensores de contacto
3.4.3 Sensores de luz infrarroja
4.3 Picking
5.2.2 Animación
5.2.3 Animación de la entidad árbol
5.3.2.3 Animación de los zapatos de la bomba

[5] Hearn, Donald; Baker, Pauline.
Gráficas por computadora
Edit. Prentice Hall Hispanoamericana. 1a. Edición. México. 1995.

Empleo de la referencia:

3.1 Introducción

[6] Eberly, David.
3D Game engine design: A practical approach to real-time computer graphics
Edit. Morgan Kaufmann Publishers. 1a. Edición. USA, 2000.

Empleo de la referencia:

3.4.1 Sensores de contacto
3.4.3 Sensores de luz infrarroja
3.4.4 Sensores de ultrasonido
4.1.1 Representación de orientaciones en WTK
4.2 Puntos de vista en Roc2
4.3 Picking
5.2.2 Animación
5.3.2.2 Cálculo de posiciones y orientaciones

[7] Watt, Alan; Policarpo, Fabio.
3D Games. Volume 1: Real-time rendering and software technology
Edit. Addison-Wesley. 1a. Edición. USA, 2000.

Empleo de la referencia:

3.4.4 Sensores de ultrasonido

[8] Aho, Alfred; Sethi, Ravi; Ullman, Jeffrey.
Compilers: Principles, techniques, and tools
Edit. Addison-Wesley. 1a. Edición. USA, 1985.

Empleo de la referencia:

1.4 Compiladores

[9] Monzón Rodríguez, Patricia; Reyna Rosey, Felipe.
Tesis: Diseño e implementación de un operador automático que reporte anomalías del sistema por medio de síntesis de voz en los equipos Sparc, Sun de la D.G.B.
UNAM. México, 1999.

Empleo de la referencia:

A.1 Definición
A.2 Ventajas

[10] Parberry, Ian.
Introduction to computer game programming with DirectX 8.0
Editorial Wordware Publishing Inc. 1a. Edición. USA, 2001.

Empleo de la referencia:

3.1 Introducción

[11] Deitel, Harvey M.; Deitel, Paul J.
Cómo programar en JAVA
Editorial Pearson Education. 1a. Edición. México, 1998.

Empleo de la referencia:

6.2 Máquina virtual de Robel

[12] Deitel, Harvey M.; Deitel, Paul J.
C++ Cómo programar
Editorial Pearson Education. 2a. Edición. México, 1999.

Empleo de la referencia:

3.4.2 Sensores reflectivos
A.3 Cómo crear un DLL utilizando visual C++

[13] González, Mileidys.
Estudio sobre los lenguajes de programación para la robótica
Fuente: http://www.monografias.com

Empleo de la referencia:

1.5 Lenguajes de programación

[14] Especificaciones del control de SNES y cómo montarlo a la computadora
http://www.emulatronia.com

Empleo de la referencia:

B.2 El control de SNES
B.3.1 Montaje

[15] Formatos de archivos: JPG, BMP, TGA.
http://www.wotsit.org

Empleo de la referencia:

3.4.2 Sensores reflectivos

[16] Descripción del puerto paralelo y programación del mismo
http://www.ctv.es/pckits/home.html

Empleo de la referencia:

B.1 El puerto paralelo

[17] Virgilio Gómez Negrete
El puerto paralelo de la computadora
http://www.modelo.edu.mx/univ/virtech/circuito/paralelo.htm

Empleo de la referencia:

B.1 El puerto paralelo

[18] Jim Christy
Control de SNES: descripción, pines que forman al conector y protocolos de comunicación
http://an.hitchcock.org/repairfaq/repair/f_snes.html

Empleo de la referencia:

B.2 El control de SNES
B.2.1 Protocolo de comunicación entre el control del SNES y el CPU del SNES
B.3 Conectando el control de SNES a la computadora
B.3.2 El software de control

[19] Máquinas virtuales
http://www.webopedia.com

Empleo de la referencia:

6.2 Máquina virtual de Robel