ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

Teleoperacion del Robot "Robonova-I" mediante Wiimote

AUTOR: Francisco José Esparza San Nicolás
DIRECTORA: Mª Francisca Rosique Contreras
CODIRECTOR: Diego Alonso Cáceres

Septiembre / 2011

Autor	Francisco José Esparza San Nicolás
E-mail del Autor	fran.esparza@hotmail.com
Directora	María Francisca Rosique Contreras
E-mail del Director	paqui.rosique@upct.es
Codirector	Diego Alonso Cáceres
Título del PFC	Teleoperacion del robot “Robonova-I” mediante Wiimote
Descriptores	

Resumen

Este Proyecto Fin de Carrera (PFC) presenta el estudio realizado sobre la utilización de nuevos dispositivos como alternativas de interfaz de control a bajo coste para sistemas teleoperados. El caso de estudio seleccionado se centra en el desarrollo de una aplicación que permita la comunicación entre el mando de la consola Wii (Wiimote) y el robot teleoperado Robonova-I.

Titulación	I.T. Telecomunicaciones. Especialidad Telemática
Intensificación	
Departamento	Departamento de Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Septiembre– 2011

INDICE

ANEXOS

INDICE DE FIGURAS

C A P I T U L O **1**

Introducción

Este capítulo sirve como presentación del Proyecto Fin de Carrera (PFC) realizado, así como de las motivaciones que nos llevo a realizarlo y el objeto de nuestro estudio.

Este capítulo se divide en tres secciones, la primera sección presenta las motivaciones por las que decidimos realizar este Proyecto Fin de Carrera (PFC), la segunda sección presenta los objetivos marcados en este Proyecto, la última sección, es una estructura de nuestro Proyecto Fin de Carrera.

1.1 Motivación

El desarrollo de este Proyecto Fin de Carrera (PFC) se debe a la búsqueda de nuevos dispositivos como alternativas de una interfaz de control a bajo coste para sistemas teleoperados.

El caso de estudio seleccionado se centra en el desarrollo de una aplicación que permita la comunicación entre el mando de la consola Wii [1] y el Robot teleoperado Robonova-I [2].

La videoconsola Wii de Nintendo aparte de sostenerse en la pulsación de botones para el control de sus aplicaciones, está basada también en los movimientos del usuario, lo que permite una nueva forma de interactuar con el sistema.

El mando responsable de detectar los movimientos del usuario tiene por nombre Wiimote y gracias a su conectividad Bluetooth, es posible utilizarlo como una interfaz de bajo coste hombre-maquina

Por otra parte, se ha tratado de realizar este Proyecto Fin de Carrera (PFC), con el Robot teleoperado Robonova-I, por tratarse de un Robot de uso educativo.

Esta búsqueda de una interfaz de bajo coste, no solo es aplicable al desarrollo de una aplicación utilizando el Robot Robonova-I, también se podría aplicar a otros Robots y a otros sistemas teleoperados.

1.2. Objetivos

En cuanto a los objetivos que se han fijado para este Proyecto Fin de Carrera (PFC), destacan los siguientes:

- El estudio y la clasificación de los robots teleoperados
- El estudio de interfaces Hombre-Maquina
- El estudio, manejo y control del Robot Humanoide Robonova -1
- El estudio de la API Wiiusej para Wiimote basada en Java
- La creación de una aplicación de teleoperacion mediante Wiimote

1.3 Estructura del PFC

Este documento se ha dividido en 6 capítulos, junto con 7 anexos.

En el primer capítulo a excepción de este apartado se dará una breve introducción al PFC explicando la motivación y los objetivos fijados para la realización de este PFC.

En el capítulo 2 se realizará un estudio del estado del arte de los sistemas teleoperados centrándose en el capítulo 3 en el robot teleoperado ROBONOVA, en el capítulo 4 se realizara un estudio del mando Wiimote como interfaz de control.

En el capítulo 5 se desarrollara una aplicación capaz de manejar el Robot Robonova-I mediante el mando Wiimote haciendo uso de las librerías estudiadas. El capitulo 6 sirve a modo de Conclusión.

El Anexo 1 se corresponde a las librerías utilizadas para la realización de este PFC, así como el uso de algunas aplicaciones utilizando como interfaz el mando Wiimote. El Anexo 2 y 3, se corresponde a la documentación relacionada con las librerías necesarias para el desarrollo de nuestra aplicación. El Anexo 4, es un breve manual para la Conexión Bluetooth PC del mando Wiimote, El Anexo 5, se corresponde la norma UNE-EN 62115 referente a juguetes eléctricos. Los Anexos 6 y 7, se corresponden a la aplicación desarrollada en nuestro PFC

C A P I T U L O **2**

Teleoperación

Este capítulo presenta los conceptos generales que definen a los sistemas teleoperados, También presenta las líneas generales de los principales sistemas teleoperados existentes actualmente en el mercad, centrándonos en los Robots teleoperados.

El capítulo se organiza en seis secciones. La primera sección tratara sobre el concepto de teleoperacion. La segunda sección presenta una breve descripción de los componentes presentes en un sistema teleoperado. En La tercera sección se presentan las características más comunes de los sistemas teleoperados. La cuarta sección hará referencia a los robots, detallando sus tipos y centrándonos en los robots teleoperados. La quinta sección explica el concepto de Interfaz, y sirve como introducción para el uso de nuestra Interfaz: El mando Wiimote. La última sección, es una recopilación de diferentes sectores en los cuales se han aplicado conceptos de teleoperacion y el uso de los robots teleoperados

2.1 Introducción a la teleoperación

Desde la antigüedad, el hombre ha desarrollado diferentes herramientas para poder aumentar su capacidad de manipulación. Este desarrollo ha desembocado en lo que hoy en día se conoce como sistema de teleoperacion [3], el cual no deja de ser un conjunto de tecnologías enfocadas a la operación o gobierno a distancia de un dispositivo o robot por un ser humano

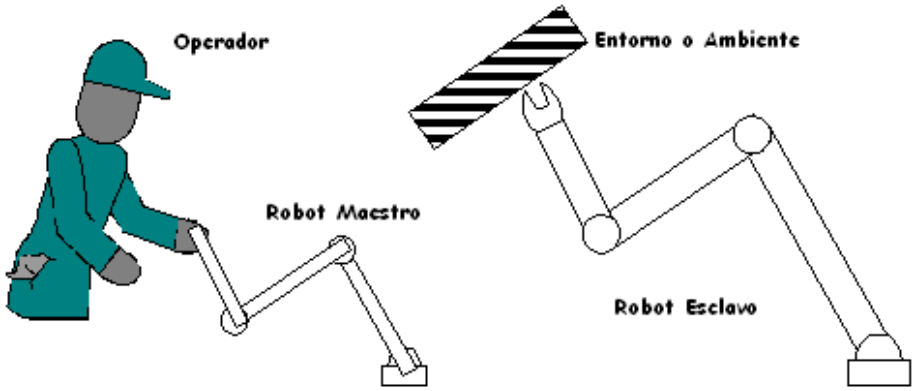


Figura 2.1: Esquema de un sistema de teleoperación

Existen muchas circunstancias en las cuales no es conveniente emplear personas para la realización de algunas labores debido al alto riesgo al que ellos se exponen; por esta razón se han desarrollado diversas herramientas o equipos que permiten reemplazar al hombre al realizar estas operaciones a distancia. Dentro de estos equipos se encuentran los móviles teleoperados también conocidos como robots a pesar de no ser autónomos en sí mismos.

2.2 Componentes de un sistema teleoperado

Un sistema teleoperado se compone principalmente de:

• *Operador o teleoperador:* es un ser humano que realiza a distancia el control de la operación. Su acción puede ir desde un control continuo hasta una intervención intermitente, con la que únicamente se ocupa de monitorizar y de indicar objetivos y planes cada cierto tiempo.

• *Dispositivo teleoperado:* podrá ser un manipulador, un robot, un vehículo o dispositivo similar. Es la maquina que trabaja en la zona remota y que está siendo controlada por el operador. En nuestro PFC utilizaremos el robot Robonova I como dispositivo teleoperado

• *Interfaz:* conjunto de dispositivos que permiten la interacción del operador con el sistema de teleoperación. Se considera al manipulador maestro como parte del interfaz, así como a los monitores de vídeo, o cualquier otro dispositivo que permita al operador mandar información al sistema y recibir información del mismo. En nuestro PFC, utilizaremos el Wiimote como Interfaz

• *Control y canales de comunicación:* conjunto de dispositivos que modulan, transmiten y adaptan el conjunto de señales que se transmiten entre la zona remota y la local. Generalmente se contará con uno o varias unidades de procesamiento.

• *Sensores:* conjunto de dispositivos que recogen la información, tanto de la zona local como de la zona remota, para ser utilizada por el interfaz y el control.

2.3 Características de los sistemas teleoperados

Las diferentes arquitecturas de teleoperación existentes proporcionan distintos grados de telepresencia que para el operador son fácilmente comparables y evaluables (enseguida se da cuenta de cuál es el algoritmo que le permite trabajar mejor). Sin embargo, en el desarrollo de sistemas teleoperados surge la necesidad de valorar de modo objetivo su funcionamiento.

Estabilidad: El concepto de estabilidad es bastante común y más que una característica es una condición indispensable para un sistema teleoperado: sería inaceptable que durante el manejo del sistema por parte del operador alguno de los robots empezara a sacudirse de manera descontrolada. El ruido eléctrico y los retrasos en las comunicaciones son causas típicas de inestabilidad. En cuanto al primero, es algo inherente a cualquier sistema de control real, pero los retrasos sólo tienen lugar cuando los robots están suficientemente alejados (por ejemplo cuando se precisa que el robot esclavo trabaje a una cierta profundidad bajo el mar), cosa que no ocurre en el sistema sobre el que se ha desarrollado el presente proyecto.

Transparencia: El concepto de transparencia por su parte, es mucho más específico del mundo de la teleoperación. Para que las labores realizadas mediante el esclavo sean precisas no es suficiente con un control de la posición del esclavo, sino que además es necesario que el operador sea capaz de percibir las fuerzas que aparecen sobre el robot remoto durante la teleoperación. En este sentido, se suele hablar de sistema teleoperado con reflexión de fuerza, ya que se pretende que las fuerzas que actúan sobre el esclavo aparezcan reflejadas de alguna manera sobre el maestro, para que así el operador sea capaz de sentirlas. Así, la transparencia infinita sería una característica de un sistema teleoperado que fuera capaz de hacer sentir al operador exactamente las mismas fuerzas que sentiría si manipulara directamente el entorno. En otras palabras, en un sistema teleoperado de transparencia infinita la impedancia[1] transmitida (la que siente el operador) coincide exactamente con la impedancia del entorno remoto. Por lo tanto, la transparencia es una medida de hasta qué punto la impedancia del entorno con el que interacciona el esclavo es alterada al ser sentida por el operador desde el maestro y a través del algoritmo de teleoperación.

Basándose en lo dicho hasta ahora, lo ideal sería desarrollar un sistema de teleoperación con reflexión de fuerza que fuera capaz de garantizar la estabilidad en la interacción con cualquier entorno, al tiempo que proporcionara una transparencia infinita y un seguimiento perfecto. Sin embargo, en la práctica ambos objetivos son incompatibles, ya que una mejora en la transparencia se consigue mediante ajustes en el sistema de control que perjudican la estabilidad del mismo.

Tipos de locomoción: Los robots móviles pueden desplazarse por medio de diversos sistemas de locomoción tales como ruedas, orugas, patas o una mezcla de los anteriores bajo diferentes configuraciones como:

Ackerman: Similar al sistema de dirección de cualquier automóvil de cuatro ruedas. Normalmente los vehículos robóticos desarrollados con esta configuración resultan de la modificación o adaptación de vehículos convencionales.

Triciclo clásico: Está compuesto de tres ruedas, la delantera sirve para proveer tracción y dirección, el eje trasero tiene acopladas dos ruedas paralelas que se mueven libremente. Tiene problemas de estabilidad en terrenos difíciles debido a que tiene pocos puntos de apoyo.

Pistas de deslizamiento: Son vehículos tipo oruga en los que tanto la tracción como el direccionamiento se consigue mediante bandas de tracción o pistas de deslizamiento este tiene desventajas como una reducción de velocidad en comparación al uso de ruedas y un gasto mayor de energía ya que en la rotación existe mucho rozamiento entre el suelo y las pistas de deslizamiento una solución a este problema es buscar que el polígono de sustentación sea lo más pequeño posible procurando no perder estabilidad.

Locomoción mediante patas: Tienen la ventaja de permitir locomoción en terrenos difíciles evitar obstáculos y omnidireccionalidad, pero su desventaja es requerir un consumo de energía mayor que con ruedas. Además que el problema de planificación y control es más complejo que un robot de ruedas u orugas.

También existen otros tipos de configuraciones como por ejemplo el uso ruedas omnidireccionales o la mezcla de patas con ruedas. Estas configuraciones tienen la intención de explorar diferentes clases de terrenos y proveer mayor funcionalidad.

La elección de una configuración específica depende de características del proyecto a realizar tales como el tipo de terreno, la velocidad y el nivel de obstáculos que se encuentren.

Sensores: Los robots deben estar provistos de diversos tipos de sensores los cuales cumplen tareas como la detección de proximidad de obstáculos y medición de posición, velocidad, aceleración e inclinación. Para la detección de obstáculos se usan principalmente foto celdas, fotodiodos, sensores de ultrasonido, sensores infrarrojos y cámaras.

Cuando se requiere la medición de posición y velocidad son usados los encoders (codificadores) los cuales están compuestos de un par fotodiodo y fotocelda que

permiten registrar posición y velocidad de partes móviles a los cuales se le acopla un disco.

La realimentación de fuerzas es una técnica útil para el control de manipuladores y robots móviles, ya que la información proveniente de los sensores puede ser convertida en fuerzas aplicadas en los dispositivos de manipulación de la interfaz de teleoperación, es decir el usuario puede sentir las colisiones que se presenten en el manejo del robot. Para esto existen sensores de fuerza y presión de tipo piezoeléctrico, resistores de fuerza-sensado o medidores de fuerza.

Visión: Dentro de los sensores que pueden colocarse en un robot móvil se encuentran las cámaras de video, las cuales además de permitir el control a distancia (visión remota) también sirven como fuente de información para la toma de decisiones automáticas, esto se realiza por medio de procesamiento de imágenes. Las operaciones principales que realiza un sistema de procesamiento de imágenes para proveer mecanismos de visión a un robot móvil son: el promediado, la segmentación de bordes, el análisis de regiones y la detección de formas.

En sistemas más sofisticados se hace uso sistemas de visión estereoscópica ubicados en el robot y cascos de realidad virtual para el operador, en los cuales graficas por computador se combinan con las imágenes provenientes de las cámaras el resultado es visualizada en un monitor o en el casco, lo cual se conoce como realidad aumentada.

Inteligencia Artificial: Dentro del sistema de procesamiento de datos pueden existir algoritmos que permitan simplificar el trabajo del operador, estos pueden ser algoritmos de planeación de trayectorias con el objeto de que el robot realice tareas simples como ir de una posición a otra evitando obstáculos utilizando la información de los sensores, y la de posición que puede ser obtenida mediante el uso de brújulas electrónicas, triangulación con sensores infrarrojos o un GPS.

Existen métodos para hacer la evasión de obstáculos en tiempo real como por ejemplo el campo de fuerza virtual y el histograma de vector de campo, otra ventaja del uso de la IA es dotar al robot de mayor autonomía, ya que es posible que el usuario no tenga el tiempo suficiente, para evitar los obstáculos que se presenten en el camino ya sea a causa de demora en las comunicaciones o por el hecho de estar realizando otras al tiempo.

2.4 Robots

Las características anteriormente vistas de los sistemas teleoperados, se pueden extrapolar a los robots [4], los cuales tienen una importancia muy grande dentro de la teleoperacion.

En los sistemas de teleoperación de robots la intervención del operador humano muchas veces es imprescindible, especialmente en entornos no estructurados y dinámicos en los cuales los problemas de percepción y planificación automática son muy complejos.

En muchos casos, el operador está físicamente separado del robot, existiendo un sistema de telecomunicaciones entre los dispositivos que utiliza directamente el operador y el sistema de control local del robot.

Los robots, atendiendo a sus características, se pueden clasificar de distintas formas, ya sea según su tipo de arquitectura, o su potencia de software.

En nuestro PFC, nos centraremos concretamente en los robots teleoperados.

2.4.1 Clasificación según su Arquitectura

Androides o Humanoides: Androide o Humanoide es la denominación que se le da a un robot antropomorfo que, además de imitar la apariencia humana, imita algunos aspectos de su conducta de manera autónoma

Móviles: Los robots móviles están provistos de patas, ruedas u orugas que los capacitan para desplazarse de acuerdo su programación. Elaboran la información que reciben a través de sus propios sistemas de sensores y se emplean en determinado tipo de instalaciones industriales, sobre todo para el transporte de mercancías en cadenas de producción y almacenes. También se utilizan robots de este tipo para la investigación en lugares de difícil acceso o muy distantes, como es el caso de la exploración espacial y las investigaciones o rescates submarinos.

Zoomórficos: Robots caracterizados principalmente por sus sistema de locomoción que imita a diversos seres vivos. Los androides también podrían considerarse robots zoomórficos.

Médicos: Los robots médicos son, fundamentalmente, prótesis para disminuidos físicos que se adaptan al cuerpo y están dotados de potentes sistemas de mando. Con ellos se logra igualar con precisión los movimientos y funciones de los órganos o extremidades que suplen.

Industriales: Los robots industriales son artilugios mecánicos y electrónicos destinados a realizar de forma automática determinados procesos de fabricación o manipulación. Son en la actualidad los más frecuentes. Japón y Estados Unidos lideran la fabricación y consumo de robots industriales siendo Japón el número uno.

Teleoperados: Los robots teleoperados se controlan remotamente por un operador humano. Cuando pueden ser considerados robots se les llama "telerobots". Cualquiera que sea su clase, los robots teleoperados son generalmente muy sofisticados y extremadamente útiles en entornos peligrosos tales como residuos químicos y desactivación de bombas.

Híbridos: Estos robots corresponden a aquellos de difícil clasificación cuya estructura resulta de una combinación de las expuestas anteriormente

2.4.2 Clasificación según la Potencia del Software.

Por Nivel Generacional:

1.- **Robots Play-back**, los cuales regeneran una secuencia de instrucciones grabadas, como un robot utilizado en recubrimiento por espray o soldadura por arco. Estos robots comúnmente tienen un control de lazo abierto.

2.- Robots controlados por sensores, éstos tienen un control en lazo cerrado de movimientos manipulados, y toman decisiones basados en datos obtenidos por sensores.

3.- Robots controlados por visión, donde los robots pueden manipular un objeto al utilizar información desde un sistema de visión.

4.- Robots controlados adaptablemente, donde los robots pueden automáticamente reprogramar sus acciones sobre la base de los datos obtenidos por los sensores.

5.- Robots con inteligencia artificial, donde los robots utilizan las técnicas de inteligencia artificial para hacer sus propias decisiones y resolver problemas.

Figura 2.2: Robot Generacional

Por Nivel de Inteligencia:

La Asociación de Robots Japonesa (JIRA) ha clasificado a los robots dentro de seis clases sobre la base de su **nivel de inteligencia**:

1.- Dispositivos de manejo manual, controlados por una persona.

2.- Robots de secuencia arreglada.

3.- Robots de secuencia variable, donde un operador puede modificar la secuencia fácilmente.

4.- Robots regeneradores, donde el operador humano conduce el robot a través de la tarea.

5.- Robots de control numérico, donde el operador alimenta la programación del movimiento, hasta que se enseñe manualmente la tarea.

6.- Robots inteligentes, los cuales pueden entender e interactuar con cambios en el medio ambiente.

Figura 2.3: Robot Inteligente

Por Nivel de Control:

1.- Nivel de inteligencia artificial, donde el programa aceptará un comando como "levantar el producto" y descomponerlo dentro de una secuencia de comandos de bajo nivel basados en un modelo estratégico de las tareas.

2.- Nivel de modo de control, donde los movimientos del sistema son modelados, para lo que se incluye la interacción dinámica entre los diferentes mecanismos, trayectorias planeadas, y los puntos de asignación seleccionados.

3.- Niveles de servosistemas, donde los actuadores controlan los parámetros de los mecanismos con el uso de una retroalimentación interna de los datos obtenidos por los sensores, y la ruta es modificada sobre la base de los datos que se obtienen de sensores externos. Todas las detecciones de fallas y mecanismos de corrección son implementados en este nivel

Figura 2.4: Robot por Control

Por Nivel del Lenguaje de Programación:

Los sistemas de programación de robots se ubican dentro de tres clases:

1.- Sistemas guiados, en el cual el usuario conduce el robot a través de los movimientos a ser realizados.

2.- Sistemas de programación de nivel-robot, en los cuales el usuario escribe un programa de computadora al especificar el movimiento.

3.- Sistemas de programación de nivel-tarea, en el cual el usuario especifica la operación por sus acciones sobre los objetos que el robot manipula.

Figura 2.5: Placa base MR-C3024 Robonova-I

2.4.3 Clasificación de los Robots Teleoperados

Los robots teleoperados son definidos por la NASA como: Dispositivos robóticos con brazos manipuladores y sensores con cierto grado de movilidad, controlados remotamente por un operador humano de manera directa o a través de un ordenador.

Dada su gran utilidad, se han empleado en diversos campos. Este tipo de manejo supone una ventaja desde el punto de vista de la protección y seguridad del usuario, ya que en caso de realizar trabajos en ambientes inseguros o inestables o con sustancias potencialmente peligrosas, como químicos o explosivos, no se arriesga su integridad física.

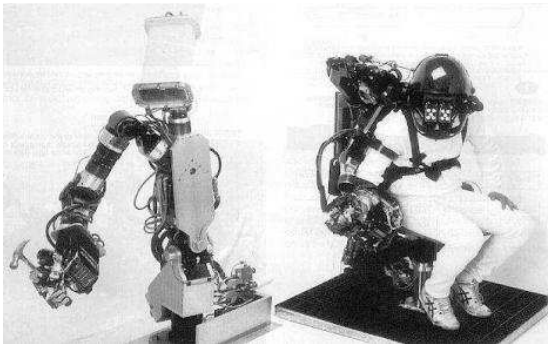


Figura 2.6: Robot teleoperado

En el desarrollo de robots teleoperados se involucra la electrónica, las comunicaciones, el control, la inteligencia artificial (IA) y la visión por computador.

El uso de IA se puede apreciar en las decisiones que debe tomar el robot por ejemplo: evitar obstáculos al ir de un sitio a otro, eligiendo el camino más corto o cuando se le enseña a reaccionar frente a ciertos estímulos y responde acertadamente a estímulos nuevos como en el caso de las redes neuronales.

La visión por computador es utilizada cuando las tareas del robot involucran el procesamiento de imágenes provenientes de cámaras de video que pueden estar ubicada en el mismo.

Tanto la IA como la visión por computador pueden simplificar significativamente el trabajo del operador.

Podemos clasificarlos en:

ROBOTS PARA USO MILITAR: El desarrollo de estos proyectos comúnmente es financiado por el departamento de defensa de los países desarrollados. Entre ellos cabe destacar:

Chrysor. Es un vehículo terrestre no tripulado, estable y robusto ideal para misiones de reconocimiento y vigilancia, incluso en terrenos difíciles. Como sistema de reconocimiento, transporte o servicio activo, este vehículo contribuye a mantener la capacidad de acción, incluso en situaciones de conflicto dinámico que pueden poner la vida en peligro.

Cougar 20-H. Se trata de un robot ligero manejado por control remoto que puede detectar la respiración humana y buscar a través de paredes de concreto con su conjunto de sensores de radio de ultra alta frecuencia. Consta de varias cámaras incorporadas lo que permite la exploración de una zona peligrosa sin que los humanos corran riesgos. El Cougar 20-H se mueve sobre rieles y puede rodar hasta un edificio**,** extender su brazo y comenzar a escanear a través de la pared

Atlas. Este robot ha sido diseñado para caminar por terrenos irregulares al estilo de un humano, arrastrándose o poniéndose de lado toda vez que sea necesario por riesgo o porque el terreno así lo requiera. Atlas también puede mantenerse en posición vertical, erguido como una persona, cuando es empujado y actualmente puede viajar a 5,15 kilómetros por hora. Atlas es la versión mejorada de otro robot humanoide llamado Petman**,** que había sido diseñado para testear trajes contra armas químicas sin necesidad de que los soldados corrieran algún tipo de riesgo.

Figura 2.7: Robot Militar Chrysor

ROBOTS PARA USO POLICIAL: Los robots teleoperados que se emplean para la manipulación y desactivación de bombas están provistos, en su mayoría, de manipuladores y de una o más cámaras de alta definición, Por tal razón deben tener un alto grado de precisión en el control de manipuladores, también estar dotados de mecanismos de locomoción que permitan afrontar diferentes tipos de terrenos. Entre ellos cabe destacar:

Wolverine: Es un modelo ya en uso fabricado por la empresa Remotec, equipado con el software SMART, el cual puede ser pre-programado para tomar muchas decisiones sin consultar con su operador, en especial sobre "cómo hacer" una cosa, dejando para el policía "qué hacer" después. También pueden usarse para la limpieza de sustancias químicas, para tareas de seguridad (patrullando perímetros y respondiendo a ataques), para actuar en accidentes nucleares, para trabajos de ingeniería de combate (derribando barreras o cortando cable de espino), en exploración espacial, asistencia sanitaria, etc.

Proytecsa Miura: Es un modelo de robot utilizado por los TEDAX de la Guardia Civil, está fabricado en España, pesa 620 Kg. El Miura mide de largo 3,5 metros (con el brazo extendido), de ancho 67 Cm y de alto 1,35 metros. Todo se maneja con un joystick y un PC portátil, que ofrece las imágenes de las 5 cámaras visibles e infrarrojas del robot, por vía inalámbrica, incluso hasta 200 metros de distancia.

Figura 2.8: Robot Policial Proytecsa Miura

ROBOTS PARA INVESTIGACIÓN: Este tipo de robot constituye un resultado de investigaciones realizadas en universidades y centros de investigación y desarrollo. Los robots teleoperados de pequeña escala se desarrollan con propósitos científicos. Podemos destacar:

Scorbot ER-400: Es un Vehículo autónomo inteligente idóneo para investigación, educación y aplicaciones en industria ligera. Integra varios sistemas de última generación: de navegación, detección de peldaños, de seguimiento de usuarios de un lugar a otro e incluso de retraso en el arranque. La integración de sensores de proximidad multidireccionales previenen el choque repetido con obstáculos y paredes y su sistema único de exploración le permite controlar muy eficientemente un área dada.

Motoman HP3: Es un robot compacto y de alta velocidad que requiere un mínimo espacio para su instalación. Su característica principal es un brazo de 701 mm (27.6\") que ofrece un amplio espacio de trabajo. Por este motivo es fantástico para trabajar en espacios de almacenaje en celdas. Su revolucionario controlador NX100 presenta características como programación bajo entorno Windows mediante pantalla táctil a color, alta velocidad de procesado, gran memoria (60,000 pasos, 10,000 instrucciones), y una robusta arquitectura de PC.

Robonova-I: Robonova-I es un robot de tipo humanoide capaz de saltar, bailar, caminar y hacer volteretas. Este tipo de robot, será el utilizado para realizar nuestro PFC. En el siguiente capítulo, se detallan todas las características de este robot con más detalle, haciendo uso del mando Wiimote como interfaz de enlace.

Figura 2.9: Robot Humanoide Robonova-I

2.5 Interfaces

Las interfaces cobran un papel muy importante en la teleoperacion ya que son los dispositivos de contacto entre un usuario y una máquina. El diseño de las interfaces incide activamente en el proceso de comunicación interactiva; así, es necesario profundizar en los aspectos cognitivos del ser humano, para definir modelos de interfaces adaptadas a las necesidades individuales y sociales, y al mismo tiempo, preparar a la sociedad para la adaptación a las nuevas interfaces que formarán parte de su entorno.

Podemos encontrar distintos tipos de interfaces como:

Interfaz de línea de comando. Requiere que el usuario introduzca la instrucción o comando por medio del teclado. El usuario teclea o escribe los comandos, carácter a carácter ante un **indicador,** usando la sintaxis y nomenclatura correctas y luego oprime Enter para ejecutarlo.

Interfaz controlada por menús. Esta interfaz proporciona menús para seleccionar opciones del programa, así el usuario no tiene que memorizar comandos. En lugar de esto los comandos son seleccionados del menú presentado en pantalla.

Interfaz gráfica del usuario (GUI - Graphical User Interfaz). En este tipo de interfaz, los usuarios controlan el sistema señalando y haciendo clic en gráficos o iconos de la pantalla que representan las características del programa. Se basa en el hecho de que la gente reconoce con más rapidez y facilidad las representaciones gráficas que las palabras o frases que lee. Se le asocia generalmente a otras características, como el uso de una interfaz de ratón activo con menús de despliegue descendente, cajas de diálogo, cajas de verificación, botones de radio y elementos semejantes.

Podemos distinguir tres tipos:

A) Una interfaz de hardware, a nivel de los dispositivos utilizados para ingresar, procesar y entregar los datos: teclado, ratón y pantalla visualiza dora.

B) Una interfaz de software, destinada a entregar información acerca de los procesos y herramientas de control, a través de lo que el usuario observa habitualmente en la pantalla.

C) Una interfaz de Software-Hardware, que establece un puente entre la máquina y las personas, permite a la máquina entender la instrucción y a el hombre entender el código binario traducido a información legible.

En Nuestro PFC, haremos uso del mando Wiimote, como interfaz Hombre-Máquina

2.6 Aplicaciones

Los robots teleoperados pueden encontrarse en múltiples sitios tales como:

- Industria Nuclear (mantenimiento de reactores)
- Industria Química (manejo a distancia de sustancias peligrosas o tóxicas)
- Industria Militar (detección, manipulación y desmantelamiento de cargas explosivas)
- Industria espacial (exploraciones realizadas en la luna/Marte, transbordadores espaciales)
- Industria Minera (excavaciones, manejo de cargas explosivas en minas y túneles)
- Sector de seguridad, mantenimiento y rescate (inspección de sistemas de alcantarillado y tuberías, reconocimiento de zonas de desastres)
- Telecirugía

C A P I T U L O **3**

Robonova I

En este capítulo se explica en detalle la herramienta principal en la que se basa el desarrollo del presente proyecto, El robot humanoide Robonova - I

El capítulo se organiza en cinco secciones. La primera sección es una introducción al Robonova-I. La segunda sección es una breve descripción sobre el contenido que nos vamos a encontrar al adquirir un kit Robonova I. La tercera y cuarta sección presentan las funcionalidades tanto Hardware como Software del Robonova I. Finalmente, la última sección explicará brevemente los movimientos que puede desarrollar el Robonova Ì.

3.1 Introducción al Robonova I

El robot Robonova-I es un robot teleoperado, de tipo humanoide capaz de saltar, bailar, caminar y hacer volteretas. Este robot, es utilizado tanto en el ámbito de la investigación, como en el ámbito de ocio debido a su coste (alrededor de los 595 €) y su facilidad de uso.

Este tipo de robot, será el utilizado para realizar nuestro PFC, ya que su placa de control permite múltiples ampliaciones, y su mecánica es modular, la cual se puede adaptar a las futuras necesidades**.**

El robonova-I, es una plataforma de desarrollo abierta, y su mecánica y software, están perfectamente adaptados a las necesidades de este PFC.

Figura 3.1: Kit Robonova-I

3.2 Contenido del Robonova I

El kit del Robonova-I contiene:

- Pletina de Control MR C-3024
- 16 servos especiales HiTEC Robot HSR-8498HB
- Todas las piezas de aluminio y plástico
- Todas las piezas pequeñas necesarias
- Batería recargable 6 Voltios/1000 mAh NiMH

- Cargador rápido para 230 Voltios
- Paquete de Software en CD-ROM
- Cable de conexión RS-232
- Mando a distancia Remocon y sensor de Infrarrojos

3.3 Hardware de Robonova I

Se trata de un producto de la empresa *HiTec Robotics*. Por una parte, permite una utilización muy simple (entre otras cosas, dispone de programación Catch and Play: el robot puede moverse a mano hasta cada una de las posiciones deseadas y generar automáticamente un programa que lleve al robot secuencialmente a cada una de esas posiciones) si la finalidad es únicamente el entretenimiento; pero, para usuarios expertos, las posibilidades son casi ilimitadas, tanto en programación como en posibilidades de incorporación de sensores. Por este motivo, se ha convertido en uno de los robots humanoides más populares en entornos docentes.

Robonova I es un robot humanoide comercial. Con la adquisición del robot se proporciona todo lo necesario para ponerlo en funcionamiento. En el CD suministrado viene el entorno de desarrollo RoboBasic, que con una sintaxis parecida a Basic es posible crear programas fácilmente.

Este humanoide mecánico y totalmente articulado, de 12" de altura, está formado por pletinas de aluminio anodizado en color oro que hacen de exoesqueleto ligero y resistente, le dan darle la rigidez necesaria y le confieren una imagen de alta calidad y aspecto imponente. El resto del cuerpo lo forman piezas de plástico rígido que protege el circuito y asegura que el robot es suficientemente robusto para el uso diario. Así se consigue un esqueleto ligero y robusto que posibilita unos grados de libertad de movimientos y una potencia única en su clase.

El Robonova I está compuesto por:

Servos

El esqueleto de Robonova I está formado por los 16 servos digitales. Los 16 servos digitales HSR 8498HB incluyen características especiales como "Motion Feedback" o lo que es lo mismo la posibilidad de leer externamente la posición real del servo, lo que permite que se pueda colocar el robot manualmente en cualquier posición y luego leer y guardar la posición leyendo los valores de los 16 servos desde el propio controlador.

El servo HSR 8498HB de 7,4 kg empleado en el robot Robonova I destaca por sus características avanzadas como son el interfaz multiprotocolo HMI (HiTec Multiprotocolo Interface) que incluye funciones programables así como la posibilidad de leer desde el controlador la posición, la tensión y el consumo actual de corriente, lo que permite crear sistemas robóticos avanzados e inteligentes, capaces de reaccionar al entorno. Este servo también se puede utilizar con los controladores de servos normales

y los receptores de radio control por lo que sus aplicaciones son infinitas. El servo incluye un completo juego de fijaciones y platos de control que le permiten funcionar de varios modos tal y como ocurre en el caso del robot Robonova.

Sus Características técnicas son:

Velocidad para 60º: 0,2 s a 6V / 0,18 s a 7,4V.

Fuerza: 7,4 Kg/cm a 6V/ 9 Kg/cm a 7,4V.

Dimensiones: 40 x 20 x 47 mm. Peso 55 gramos.

Engranajes: Karbonite.

Cojinetes: 2.

Existen gran cantidad de accesorios y repuestos para este servo, incluyendo las plaquetas metálicas del Robonova que permiten crear estructuras articuladas muy fácilmente como brazos, articulaciones...

Figura 3.2: Servos Robonova

Mando

Consta de mando a distancia por infrarrojos y sensor de infrarrojos para el manejo del robot robonova1. Con este mando se pueden ejecutar distintas órdenes e instrucciones en el Robonova de una forma muy sencilla gracias a la aplicación RoboRemocon que incluye el software que viene con el robot.

Figura 3.3: Mando Robonova-I

Cable de conexión serie

El Cable de conexión serie Robonova S300464 es de 1,8 metros de longitud y se emplea para programar el robot. El cable tiene un jack de 3,5 mm estéreo en un extremo y un canon hembra de 9 pines en el otro.

Figura 3.4: Cable de Conexión Serie

Microcontrolador

El robot Robonova 1 está controlado por el microcontrolador Atmel ATMega 128 que está integrada en la placa MR C-3024, que está integrada entre otras cosas con 40 puertos de entrada y salida digitales, puerto serie, bus I2C y 8 entradas analógicas. Con este elevado número de puertos se pueden controlar dispositivos de todas clases como servos, sensores de distancia, giróscopos, displays LCD, sensores de infrarrojos, etc. Algunos de estos ejemplos se han añadido en la sensorización. Además la placa cuenta con un altavoz para generar tonos de diferentes frecuencias y un conector para un led que se puede gobernar a voluntad. Otros componentes de la placa incluyen más de 64 Kbyte de memoria para los programas, que permiten que una vez que se han descargado, el robot sea completamente autónomo.

Sus características principales son:

Alto rendimiento:

- Microcontrolador AVRR 8-bit de bajo consumo

Arquitectura:

- 130 Instrucciones
- 32 x 8 Registros de Propósito General

Alta durabilidad de Memoria No Volátil:

- 8K Bytes Flash p
- 512 Bytes EEPROM
- 1K Byte Internal SRAM
- Ciclos de Borrado: 10,000 Flash/100,000 EEPROM

Periféricos

- Dos Temporizador/Contador de 8-bit con pre escala separada y modo de comparación
- Un Temporizador/Contador de 16-bit con pre escala separada y modo de comparación y captura
- Contador de Tiempo real con Oscilador separado
- Tres canales PWM
- 8-canales ADC
- Interfaz serie de dos cables orientado a byte
- USART Programable serie
- Interfaz serie SPI Maestro/Esclavo
- Watchdog programable
- Comparador analógico

Entrada/Salida

- 23 líneas de entrada/salida programables

Voltajes de operación

- 2.7 - 5.5V (ATmega8L)
- 4.5 - 5.5V (ATmega8)

• Velocidad de procesador

- 0 - 8 MHz (ATmega8L)
- 0 - 16 MHz (ATmega8)

• Consumo de energía a 4 MHz, 3V, 25°C

- Activo: 3.6 mA
- Reposo: 0.5 μA

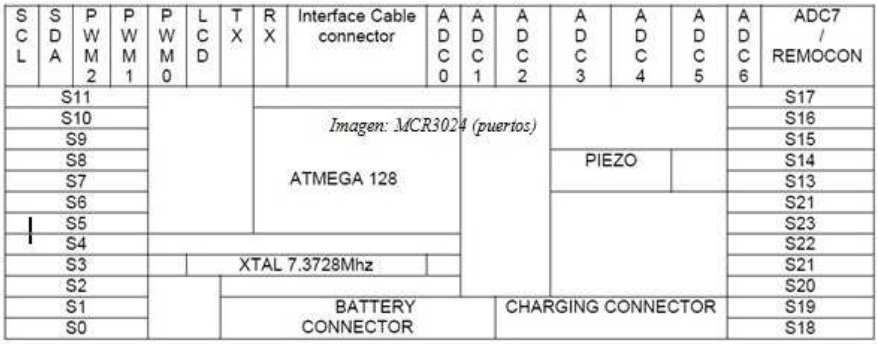


Figura 3.5: MCR3024 (puertos)

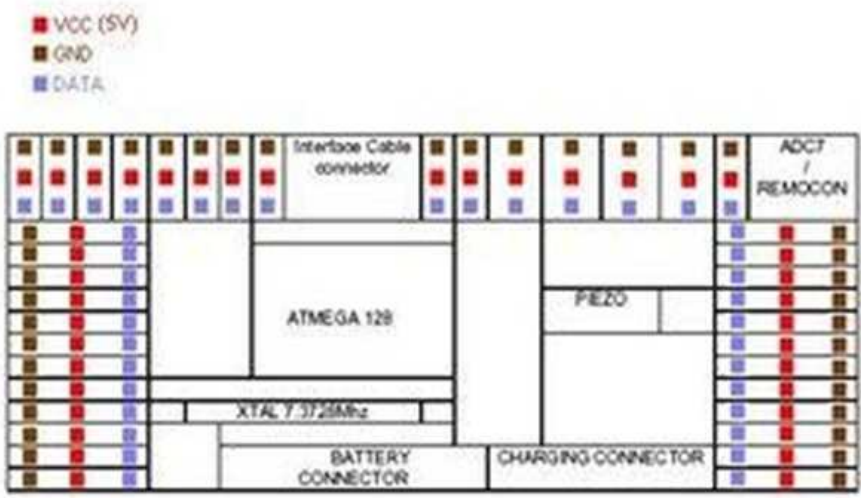


Figura 3.6: MCR3024 (patillaje)

Baterías

El Robonova-I se alimenta con una batería recargable de 5 celdas NiMH, que permite una hora aproximada de funcionamiento. Una batería de níquel-hidruro metálico (NiMH) es un tipo de batería recargable que utiliza un ánodo de oxidroxido de níquel (NiOOH), como la batería de níquel cadmio, pero su cátodo es de una aleación de hidruro metálico. Esto permite eliminar el costoso (y medioambientalmente peligroso) cadmio a la vez que se beneficia de una mayor capacidad de carga (entre dos y tres veces la de una pila de NiCd del mismo tamaño y peso) y un menor efecto memoria. Son, por tanto, ideales para los servos por sus prestaciones a la hora de esfuerzos puntuales, ya que las baterías basadas en litio son peores en dicho aspecto.

El cargador de corriente incluido, de tipo rápido, dispone de un circuito de protección para evitar problemas en la carga de la batería.

Figura 3.7: Batería Robonova

3.4 Software del Robonova I

El software es el soporte lógico de un ordenador o microcontrolador, y es, junto al hardware, la base de la robótica actual. Esto es así porque sin hardware solo tendríamos piezas inanimadas y sin software el robot estaría muy limitado en sus capacidades.

El software del Robonova-I es de fácil complejidad, es una de las razones por la cual se decidió realizar este PFC utilizando este robot y su software.

3.4.1 Lenguaje de programación

La programación del microcontrolador de la placa MR C-3024 está basada en el lenguaje de programación RoboBasic, basado a su vez en el conocido lenguaje BASIC. Este último fue desarrollado en la década de 1960 por Thomas Eugene Kurtz y John George Kemeny, e inicialmente era una herramienta de enseñanza, aunque en los años 80 se comenzó a utilizar en ordenadores personales. Con el paso de los años se crearon versiones y dialectos basados en BASIC, algunos con diferencias notables.

Algunos ejemplos son Turbo Basic y el más moderno Visual Basic de Microsoft. El caso que nos ocupa, RoboBasic, es parecido ya que a los comandos originales añade otros específicos para el control del robot. Estos comandos combinados entre sí determinaran la función del programa y su flujo de ejecución. Se pueden clasificar de la siguiente manera:

- Declaración, definición
- Control de flujo
- Entradas y salidas de señales digitales
- Manipulación de memoria
- Pantalla de LCD

- Operandos
- Control de servos
- Asignación de parámetros de grupos de servos
- Control del sonido
- Comunicaciones externas
- Procesado de señales analógicas
- Sucesos
- Otros
- Intención

3.4.2 Herramientas de programación

Para facilitar y complementar la tarea de programación del robot, contamos con diversos software para probar los programas en el Robonova. Son una pieza fundamental en la programación, y sin ellos no podríamos programar el Robonova desde un ordenador personal.

RoboBasic v2.72

La herramienta imprescindible para la programación del microprocesador. Con el podemos escribir programas en lenguaje RoboBasic para posteriormente compilarlos, enlazarlos y descargarlos en la memoria del robot, para lo cual se comunica vía puerto serie con el robot. Estos archivos tienen extensión .bas, aunque el programa también es capaz de manipular los archivos de extensión .rsf. También tiene diversas utilidades tales como:

- Depuración del código con puntos de ruptura (breakpoints)
- Obtención de información del microcontrolador
- Configuración de la comunicación serie con el robot
- Control de los servos en tiempo real
- Configuración del punto cero de los servos
- Resteo y paro de la ejecución
- Borrado de la memoria de programa
- Captura de la posición de los servos, lo que nos permite mover las articulaciones del robot a nuestro antojo y capturar esa posición

Una parte de nuestra aplicación, utilizara el lenguaje RoboBasic, el cual se detalla más extensamente en el Anexo A.

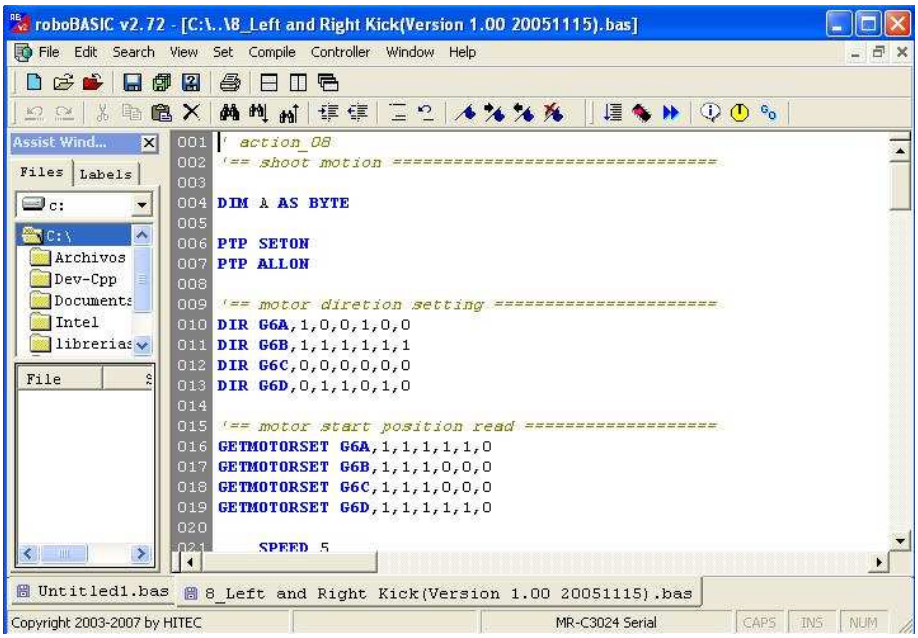


Figura 3.8: RoboBasic

RoboSCRIPT v2.72

Los archivos manejados por este programa tienen la extensión .rsf. Nos permite crear rutinas que más tarde podrán ser incluidas en el programa usando tanto RoboBasic como RoboRemocon, asignándoles en este último un botón del mando a distancia.

RoboRemocon facilita la creación de rutinas fundamentalmente de movimiento, ya que integra botones y barras para que el programador no necesite escribir ninguna línea de código.

Los botones tienen funciones como DELAY, para esperar un determinado tiempo, SPEED, para cambiar la velocidad de movimiento de los servos, GOTO, para cambiar el flujo de programa o MOVE, que insertara una posición a la que deberán moverse los servos.

Dicha posición la configuramos con las barras deslizantes en la parte inferior de la pantalla.

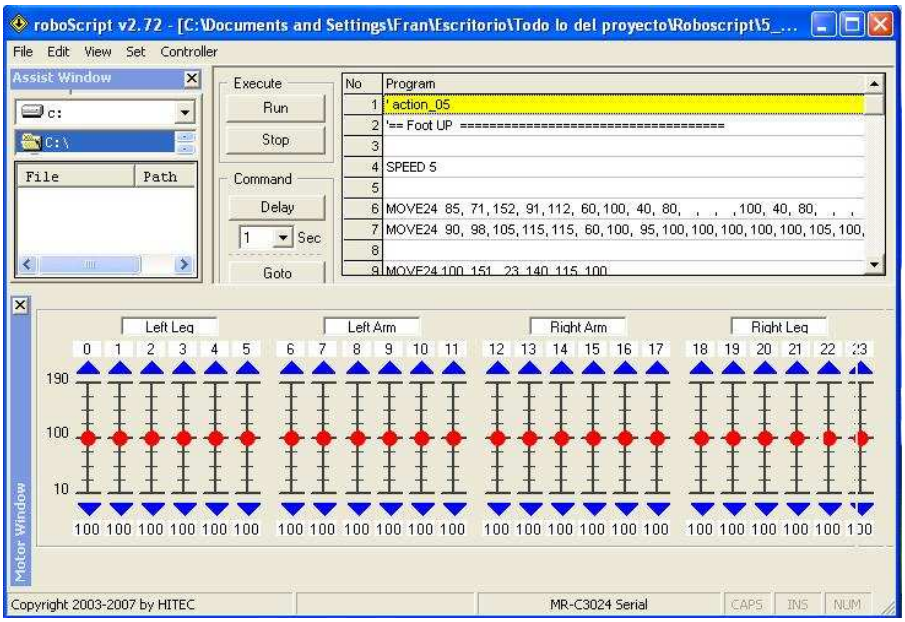


Figura 3.9: RoboScript

RoboRemocon v2.72

Esta herramienta facilita la programación del mando a distancia por infrarrojos con el que viene equipado Robonova. Mediante una sencilla interfaz grafica, podemos asignar a cada botón del mando una rutina diseñada previamente en el RoboSCRIPT, abriendo el archivo .rsf. Posteriormente se puede poner a prueba la programación pulsando tanto los botones del mando a distancia real como los del virtual, mostrado en una imagen inferior.

El RoboRemocon se puede identificar con un ID entre 1 y 4. Esto permite el control de hasta 4 robots Robonova-I diferentes, de manera simultánea y sin interferencias.

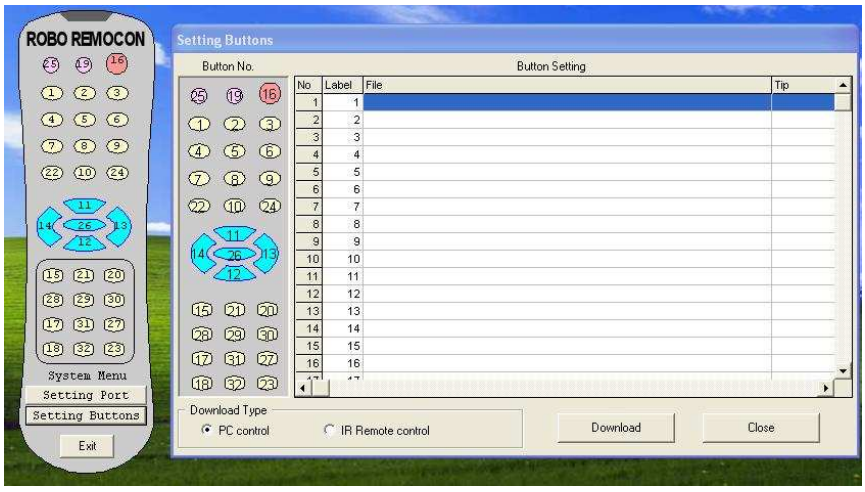


Figura 3.10: Remocon

3.4.3 Software de simulación

En microrrobotica la simulación en ordenador es una herramienta muy útil, en algunos casos fundamentales. Gracias a ella podemos pre visualizar las acciones del robot, y observar cómo se comportaría en la realidad. De esta manera se pueden corregir errores antes de probar el programa en el robot real. También nos dan información relevante del robot, y nos permiten crear diversos entornos y situaciones para hacer pruebas.

En el caso del Robonova, la herramienta que nos permite simular su comportamiento es el software SimROBOT 0. Con el podemos simular tridimensionalmente ciertos movimientos del robot y editar su reproducción. También nos permite, via puerto serie, programar el microcontrolador con la secuencia de movimientos, y comprobar que el robot se comporta igual que en la simulación.

Otra utilidad es la de importar los archivos de rutina de extensión .rsf, generados por el software RoboScript, de forma que también se puedan simular movimientos creados por el usuario.

El software está compuesto por 4 ventanas:

• La primera, denominada Time Line, tiene las líneas temporales que en las que se representan los movimientos programados. Consta de dos barras de tiempo, y en una de ellas se incluyen los keyframes que marcan el comienzo y final de cada instrucción de movimiento.

• La segunda, llamada 3DView, es la visualización en 3 dimensiones del robot, que tendrá una pose inicial que podremos cambiar a nuestro antojo, y cuando comience la simulación se moverá acorde a los movimientos programados. Podemos tener más de una ventana de este tipo para tener una visión más amplia de las acciones del robot. En esta ventana también se incluyen varias utilidades, siendo la más practica la posibilidad de ver el punto de gravedad del robot proyectado sobre el suelo, lo que nos permitirá prever perdidas de equilibrio y caídas, cosa bastante frecuente en este tipo de robots.

• La tercera ventana es la de Opciones, en la cual se añaden los movimiento a realizar, se controla la pose del robot servo por servo, se configuran las comunicaciones por puerto serie, y se cambian ajustes como la detección de colisiones o la activación de seguridad del robot para que los giros de los servos no hagan chocar partes de su cuerpo

• La cuarta y última ventana es la de Video, que integra un reproductor de video, en el cual se puede reproducir una grabación del robot, y sincronizar sus movimientos con los de la simulación, para comprobar una correcta ejecución de los mismos.

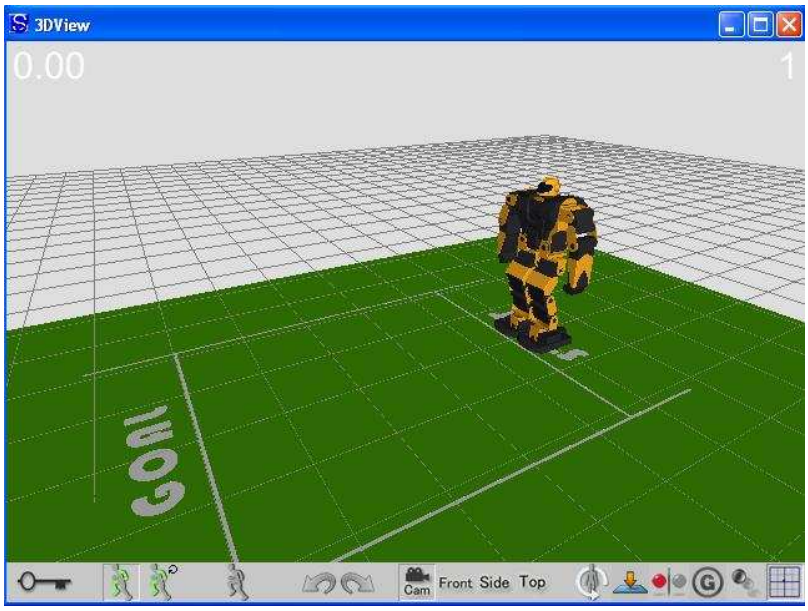


Figura 3.11: Software de simulación (Vista 3D)

3.5 Movimientos del Robonova I

Como base de partida para elaborar movimientos se utilizó el programa demostración que venía incluido con el robot, gracias al cual se obtuvieron los movimientos básicos como pueden ser caminar hacia delante, caminar hacia atrás y girar en ambos sentidos.

Para definir ciertos movimientos desde el principio. Se puede utilizar la herramienta proporcionada por RoboBasic denominada "Servo Motor Real-Time Control", la cual permite la monitorización en tiempo real de la posición de cada servo e insertar en el código una línea que efectúe dicho movimiento. Cabe destacar que cada servo tiene un recorrido de 10 a 190 traduciéndose esto en una amplitud de movimiento de unos 180º pudiendo por tanto relacionar el 10 con la posición de 0º y del mismo modo el 190 con 180º.

Otra funcionalidad destacable de robonova es que permite agrupar los servos de 6 en 6 generando 4 grupos.

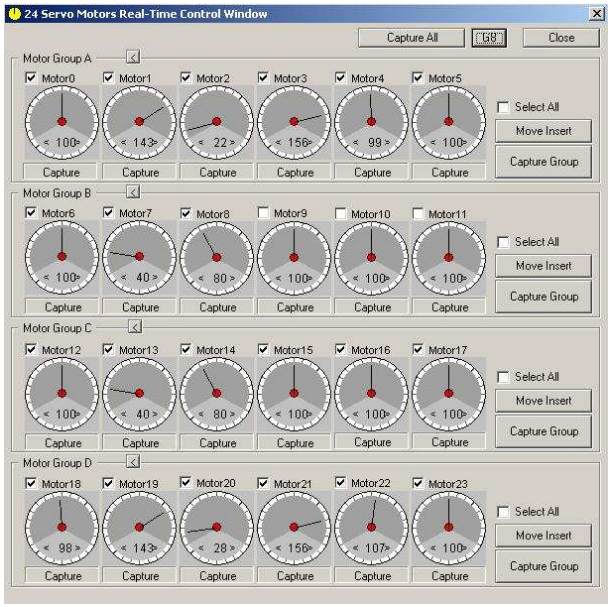


Figura 3.12: Servo Motor Real-Time Control.

Los grupos son:

- G6A que controla la pierna izquierda.
- G6B que controla el brazo izquierdo.
- G6C que controla el brazo derecho.
- G6D que controla la pierna derecha.

La asignación de cada servo dentro de cada grupo es la siguiente:

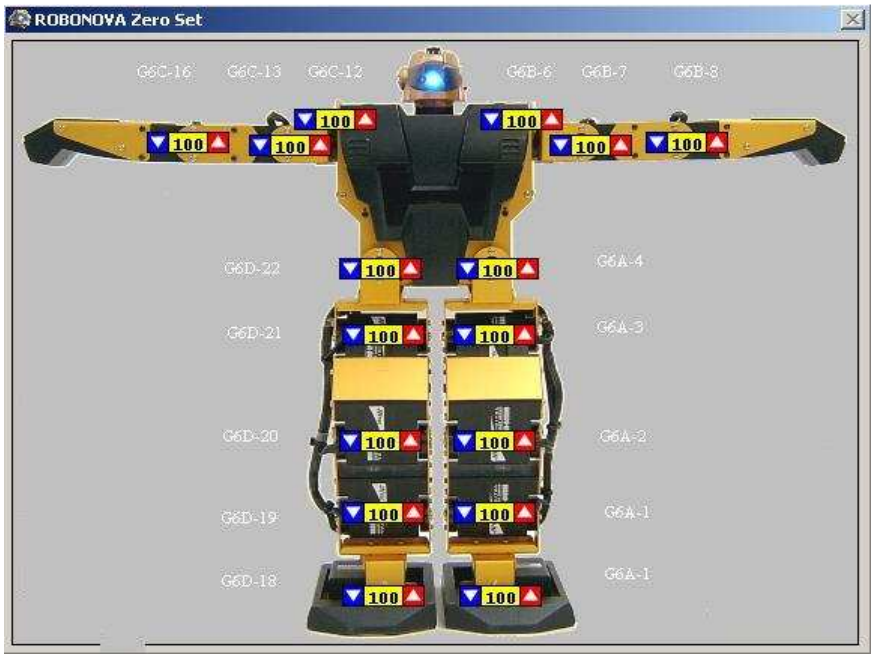


Figura 3.13: Asignación de servos.

Existen otros tipos de asignación, utilizando los 24 servos simultáneamente o por grupos de 8. No obstante, la mencionada, por su sencillez, ha sido la más utilizada.

A continuación se muestra un ejemplo de utilización de los servos del robot.

```
SPEED 15            'Selecciona la velocidad de movimiento.
MOTOR G24           'Inicia los 24 servos.

MOVE G6A,100,  82, 115, 124, 100, 100
MOVE G6B,110,  36,  66, 100, 100, 100
MOVE G6C,110,  37,  65, 102, 100, 100
MOVE G6D, 98,  93, 116, 118, 103, 100

WAIT                'Espera a que el movimiento de los servos haya  concluido.
```

CAPITULO 4

Wiimote

En este capítulo se explica en detalle una de las herramientas en la que se basa el desarrollo del presente proyecto, El mando de la consola Wii: Wiimote

El capítulo se organiza en seis secciones. La primera sección es una introducción sobre el mando de la consola de Wii. La segunda sección presenta una breve descripción del diseño del Wiimote. La tercera y cuarta sección presentan las funcionalidades así como las características del mando. La quinta sección hará referencia a los accesorios que se pueden equipar al mando Wiimote acompañado de una breve descripción detallando las características más significativas de dichos accesorios. Finalmente, la última sección explicará brevemente algunas de las formas en las que podemos conectar nuestro mando Wiimote al PC mediante una conexión bluetooth

4.1 Introducción

El **Wii Remote** o **Wiimote** [5] es el mando principal de la consola Wii de Nintendo. Sus características más destacables son la capacidad de detección de movimiento en el espacio y la habilidad de apuntar hacia objetos en la pantalla.

Servirá como interfaz Hombre – Máquina en nuestro PFC

Figura 4.1: Wiimote

4.2 Diseño de Wiimote

El diseño del Wiimote no se basa en los tradicionales mandos para los videojuegos. A diferencia de ellos, es similar a un control remoto de televisión creado para ser utilizado en una sola mano y de la manera más intuitiva posible.

En su cara frontal, el Wiimote presenta los botones "A", "1", "2", "+", "-", "HOME" "POWER" y la cruz de direcciones. En la parte anterior sólo presenta el botón "B", en un formato similar a un gatillo.

Adicionalmente, en su parte frontal incluye un altavoz y cuatro luces numeradas que indican el número de jugador al que corresponde tal mando y ver el tiempo de vida de la batería. En la parte inferior viene unida una correa de seguridad que se amarra a la muñeca para evitar soltar y dañar el control de forma accidental.

4.3 Funcionalidad de Wiimote

- **Barra de sensores con leds Infrarrojos y detección de movimientos**

El Wiimote tiene la capacidad de detectar la aceleración a lo largo de tres ejes mediante la utilización de un acelerómetro ADXL330. El Wiimote también cuenta con un sensor óptico PixArt, lo que le permite determinar el lugar al que el Wiimote está apuntando.

A diferencia de un mando que detecta la luz de una pantalla de televisión, el Wiimote detecta la luz de la Barra sensor de la consola (número de modelo RVL-014), lo que permite el uso coherente, independientemente del tipo o tamaño de la televisión. Esta barra mide aproximadamente 20 cm de longitud y cuenta con diez LED infrarrojos, con cinco LED dispuestos en cada extremo de la barra. En cada grupo de cinco LED, el LED más lejano fuera del centro apunta ligeramente lejos del centro, el LED más cercano al centro apunta ligeramente hacia el centro, mientras que los tres LED entre ellos están apuntando directamente hacia adelante y agrupados. El cable de la barra sensor mide 353 cm de longitud. La barra puede ser colocada por encima o por debajo de la televisión, y debe centrarse. Si está colocada por encima, el sensor debe estar alineado con la parte delantera de la televisión, y si coloca en la parte inferior, debe alinearse con la parte delantera de la superficie de la televisión en la que se coloca. No es necesario señalar directamente a la barra sensor, pero apuntar significativamente fuera de la barra de posición perturbará la capacidad de detección debido al limitado ángulo de visión del Wiimote.

El uso de la Barra de Sensores permite al Wiimote ser utilizado como un dispositivo de señalamiento preciso de hasta 5 metros de distancia de la barra. El sensor de imagen del Wiimote se utiliza para localizar los puntos de luz de la Barra con respecto al campo de visión del Wiimote. La luz emitida desde cada extremo de la Barra de Sensores se centra en el sensor de imagen que ve la luz brillante como dos puntos separados por una distancia de "mi" en el sensor de imagen. La segunda distancia "m" entre los dos grupos de emisores de luz de la barra sensor es una distancia fija. A partir de estas dos distancias y mi m, el procesador de la consola Wii calcula la distancia entre el Wiimote y la barra de sensores utilizando la triangulación. Además, la rotación del Wiimote con respecto al suelo también puede ser calculada a partir del ángulo relativo de los dos puntos de luz en el sensor de imagen. Los juegos pueden ser programados para detectar si el sensor de imagen está cubierto, lo que fue demostrado en un mini juego de *Smooth Moves*, donde si el jugador no descubre el sensor, la botella de champán que el mando a distancia representa no se abre.

La barra Sensor es necesaria cuando el Wiimote está controlando movimientos arriba-abajo, izquierda-derecha de un cursor en la pantalla del televisor para apuntar a las opciones de menú u objetos como los enemigos en un juego. Debido a que la Barra de sensores también permite calcular la distancia entre el Wiimote y la sensibilidad de la Barra, el Wiimote también puede controlar el movimiento adelante-retroceso hacia un objeto en un juego en 3 dimensiones. El movimiento rápido hacia delante o hacia atrás, como los puñetazos en un juego de boxeo, está controlado por los sensores de aceleración. Usando estos sensores de aceleración (que actúan como sensores de inclinación), el Wiimote también puede controlar la rotación de un cursor u otros objetos.

El uso de un sensor infrarrojo para detectar posición puede causar algunos problemas cuando otras fuentes de infrarrojos se encuentran alrededor, como bombillas incandescentes o velas. Esto puede ser fácilmente mitigado por el uso de luces fluorescentes alrededor de la Wii, ya que emiten poca o ninguna luz infrarroja. Usuarios innovadores han utilizado otras fuentes de luz IR como sustitutos Sensor Bar, como un par de linternas y un par de velas. Tales sustitutos de la Barra de Sensores ilustran el hecho de que un par de luces estáticas proporcionan una calibración continua de la dirección que el Wiimote está apuntando y su ubicación física en relación con las fuentes luminosas. No hay manera de calibrar la posición del cursor en relación con la que el usuario está señalando con el controlador sin las dos fuentes estables de referencia de la luz proporcionada por la Barra de Sensores o sustitutos.

Los leds pueden verse a través de algunas cámaras y otros dispositivos con un mayor espectro visible que el ojo humano.

La posición y seguimiento del movimiento del Wii Remote permite al jugador imitar las acciones reales de juego, como blandir una espada o una pistola con objetivo, en lugar de simplemente pulsando los botones. Uno de los primeros videos de marketing mostró actores mimetizando acciones como la pesca, cocina, tocar la batería, dirigiendo un cuarteto de cuerda, disparando un arma de fuego, luchando con espada, y la realización de cirugía dental.

- **Memoria**

El Wiimote contiene un chip de 16 KB EEPROM donde una sección de 6 kilobytes puede ser libremente leída y escrita por el host. Parte de esta memoria está disponible para almacenar hasta 10 avatares Mii, que pueden ser transportados para su uso en otra consola Wii. Al menos 4000 bytes están disponibles y no utilizados antes de los datos Mii.

- **Alimentación**

El Wiimote utiliza dos baterías AA como fuente de energía, que pueden alimentar el Wii Remote durante 60 horas usando sólo la función de acelerómetro y 25 horas utilizando acelerómetro y puntero. Todavía no se ha lanzado un cargador para el Wiimote, pero terceros fabricantes del mercado han desarrollado sus soluciones.

4.4 Características de Wiimote

- Botones (POWER, A, -, HOME, +, 1, 2 y B como gatillo en la parte inferior)
- Cruz direccional
- 4 leds: indican el número de jugador o el nivel de batería.
- 2 pilas AA: con una duración de entre 30 y 60 horas según el uso o no de la función de puntero.

- Acelerómetro *ADXL330* de 3 ejes: para detección de movimientos del usuario.
- Sensor óptico de infrarrojos: para detección de movimientos del usuario.
- Puerto de expansión: para la conexión de otros elementos como el controlador Nunchuck
- Conexión Bluetooth: para la comunicación entre consola y el propio Wiimote.
- Altavoz
- Motor de vibración
- Chip de memoria EEPROM de 16 KB: destinado a guardar información del juego en ejecución y hasta 10 perfiles de usuario (denominados *Mii*)

4.5 Accesorios de Wiimote

El Wiimote tiene un puerto de expansión para añadir periféricos en su parte inferior.

Nunchuk

El Nunchuk es una expansión para el mando inalámbrico de Wii. Su nombre proviene del Nunchaku, arma de artes marciales, ya que al conectarlo al Wiimote da una apariencia similar a un Nunchaku. Se entrega uno de serie con cada Wii y puede adquirirse por separado

Tiene una longitud de 11 cm y una forma ovalada que se adapta a la mano. Dispone, como éste, de un sensor de movimiento y añade tres nuevas funciones al mando de Wii :

Botón C: pequeño, redondo, situado en el frontal, pensado para ser accionado con el índice

Botón Z: 4 veces mayor y situado debajo, de forma cuadrada, pensado para ser accionado por el corazón

Stick analógico de control situado en la empuñadura y pensado para ser accionado por el pulgar.

Figura 4.2: Nunchuk

Wii Classic Controller

El **Wii Classic Controller** es un mando de control de Wii parecido al de GameCube (con la excepción de que no tiene grips para sujetarlo).

El cuerpo Wii Classic Controller mide 6.57 centímetros (2.59 pulgadas) de altura y 13,57 centímetros (5,34 pulgadas) de ancho. El cuerpo del mando contiene ranuras en la parte inferior, abierto a través de un botón en la parte superior del controlador, la función de las ranuras nunca se aclaró oficialmente.

Figura 4.3: Wii Classic Controller

Wii Classic Controller Pro

A principios de 2009, Nintendo anunció el Classic Controller Pro, que funciona igual que el Classic Controller original, con la excepción de los botones traseros, que ahora son botones en forma de gatillo dispuestos verticalmente en lugar de horizontalmente. Los cambios físicos incluyen los botones ZL y ZR, que son botones completos traseros, la adición de agarres por debajo del regulador para una estabilidad adicional

Figura 4.4: Wii Classic Controller Pro

Wii Zapper

Accesorio usado en algunos juegos de pistolas.

Figura 4.5: Wii Zapper

Wii Wheel

En este accesorio tendremos un modo avanzado de control para videojuegos de carreras, entre otros. Es una carcasa de plástico donde se aloja el Wiimote, pero no añade características adicionales.

Figura 4.6: Wii Wheel

Wii Guitar

Es un accesorio con forma de guitarra en el que se inserta el Wiimote que sirve para el juego Guitar Hero para Wii.

Figura 4.7: Wii Guitar

Wii Balance Board

La Wii Balance Board es un accesorio para la consola Wii de Nintendo que consiste en una tabla capaz de calcular la presión ejercida sobre ella. Es de color blanco como todos los accesorios de la Wii y puede soportar hasta 150 kg. Mide de largo 30 cm, y de ancho 48 cm.

Figura 4.8: Wii Balance Board

Wii MotionPlus

Wii Motion Plus es un accesorio que incorpora tres sensores de movimiento correspondientes a los ejes vertical, longitudinal y lateral, añadiendo precisión extra al Wiimote. Tiene incorporado un puerto de expansión para poder añadir otro accesorio. Algunos títulos de Wii requieren la presencia obligatoria de este accesorio, que salió a la venta en junio de 2009. Requiere una funda protectora de tamaño mayor.

Figura 4.9: Wii Motion Plus

Wii Vitality Sensor

Es un nuevo accesorio anunciado en el E3 del 2009 que puede medir la presión y detectar si se siente miedo, alegría, etc. Hasta la fecha no se han dado más datos sobre este accesorio.

Figura 4.10: Wii Vitality Sensor

Wii Remote Plus

Conocido también como Wiimote Plus. Combina el Wii Remote y el Wii MotionPlus en un sólo control, sin alargar el tamaño normal del Wiimote.

Figura 4.11: Wiimote Plus

UDraw Game Tablet

Es una tableta de juego que se utiliza en los juegos UDraw Studio, Dood's Big Adventure y Pictionary para Wii.

Figura 4.12: UDraw Game Tablet

4.6 Conectividad de Wiimote

Dado que el Wiimote utiliza Bluetooth para la transmisión de datos con la videoconsola Wii, se puede aprovechar dicha tecnología para la comunicación con él mediante un PC, teléfono móvil o cualquier dispositivo que disponga de la misma.

Gracias a la posibilidad de conectar mediante Bluetooth el Wiimote y un PC, han surgido diversas librerías y aplicaciones para su utilización como dispositivo de entrada sustituyendo por ejemplo un ratón, teclado, o joystick.

Podemos conectar el Wiimote al PC utilizando el gestor de Windows XP o utilizando un gestor externo de conexiones Bluetooth como el BlueSoleil.

Para más información acerca de la conexión del Wiimote al PC consulte el anexo D

C A P I T U L O **5**

Aplicación para controlar Robonova I

En este capítulo se abordara la creación de una aplicación capaz de controlar el Robot Humanoide Robonova I mediante nuestra interfaz de bajo coste: el mando de control Wiimote.

El capítulo se organiza en cinco secciones. La primera sección es una breve introducción del porque del desarrollo de esta aplicación. La segunda sección explica brevemente cada uno de los recursos utilizados para la realización de este PFC. La tercera sección es una breve descripción de la aplicación. La cuarta sección es la parte de la aplicación encargada de la comunicación entre Wiimote y el PC. La quinta sección es la parte de la aplicación encargada del control y manejo del robot Humanoide Robonova I.

5.1. Introducción a la aplicación

El desarrollo de esta aplicación, se debe al estudio y a la búsqueda de nuevos dispositivos para ser usados como interfaces Hombre-Máquina de bajo coste para sistemas teleoperados.

Esta aplicación, se centra exclusivamente en la teleoperacion entre el mando Wiimote y el Robot Robonova-I, para demostrar que el mando Wiimote, es una interfaz Hombre-Máquina realmente valida, útil y de bajo coste para poder teleoperar y manejar diferentes sistemas teleoperados o robots.

5.2. Recursos utilizados

Para la realización de esta aplicación, dentro del marco de nuestro proyecto fin de carrera (PFC) se requiere la utilización de una serie de recursos tanto software como hardware que se detallan a continuación:

Software:

· **Java:** Lenguaje de Programación orientado en objetos. Nuestra aplicación estará basada en este tipo de Lenguaje de Programación

· **RoboBASIC:** Lenguaje de Programación utilizado por el Robonova I

· **Eclipse:** Software Utilizado para el desarrollo de nuestra aplicación en lenguaje Java

· **Windows XP:** Se utilizará este S.O. como base para el desarrollo de esta aplicación para tratar de maximizar la compatibilidad con las librerías de *Wiimote* y de RoboBASIC

Hardware:

· **PC Portátil**: Un PC de gama media-alta que será suficiente para el desarrollo y pruebas de la aplicación.

· **Mando Wiimote:** Será nuestra interfaz Hombre-Máquina, encargada de capturar los movimientos del usuario.

· **Dongle Bluetooth:** Adaptador Bluetooth para la comunicación PC-Wiimote.

· **Adaptador Puerto Serie:** Un adaptador USB – Puerto Serie si nuestro PC Portátil no dispone de un Puerto Serie

· **Cable de Conexión Robonova:** Cable de Conexión serie incluido en el kit Robonova I para realizar la comunicación PC-Robonova

· **Robonova-I:** Robot Humanoide encargado de realizar los movimientos indicados por el usuario

5.3 Descripción de la Aplicación.

Esta Aplicación, consistirá en la comunicación y control de un robot Humanoide Robonova I mediante el mando Wiimote que hará de interfaz.

La comunicación existente entre los distintos dispositivos se detalla a continuación:

Wiimote – PC: Esta comunicación será mediante una conexión Bluetooth

PC - Robonova I: Esta comunicación será mediante un cable puerto serie

COMUNICACIÓN VIA BLUETOOTH		COMUNICACIÓN VIA PUERTO SERIE
		
ENLAZAR MANDO Y PC MEDIANTE BLUETOOTH	APLICACIÓN PARTE 1 DESARROLLADA EN JAVA	APLICACIÓN PARTE 2 DESARROLLADA EN ROBOBASIC

Figura 5.1: Esquema Aplicación Desarrollada

El Mando Wiimote se enlazara al ordenador mediante bluetooth, será el encargado de capturar los movimientos realizados por el usuario.

El primer apartado de la aplicación, será el encargado de recoger la información proveniente del mando de Wiimote, así como el posterior envió de información al robot Robonova I.

El segundo apartado de la aplicación, será el encargado de recibir la información enviada por el usuario, y gestionar los movimientos del robot Robonova I

5.4 Aplicación Parte 1.

Nuestra aplicación esta realizada en lenguaje JAVA utilizando el patrón observador. Será la encargada de detectar si el mando Wiimote esta activado, ha realizado algún movimiento sensorial o se ha pulsado un botón.

Posteriormente, esta aplicación transmitirá esta información via Puerto Serie. Esto es realizado mediante un Manejador de eventos.

Utilizaremos la librería Wiiusej [6] , la cual esta detallada en el Anexo A.

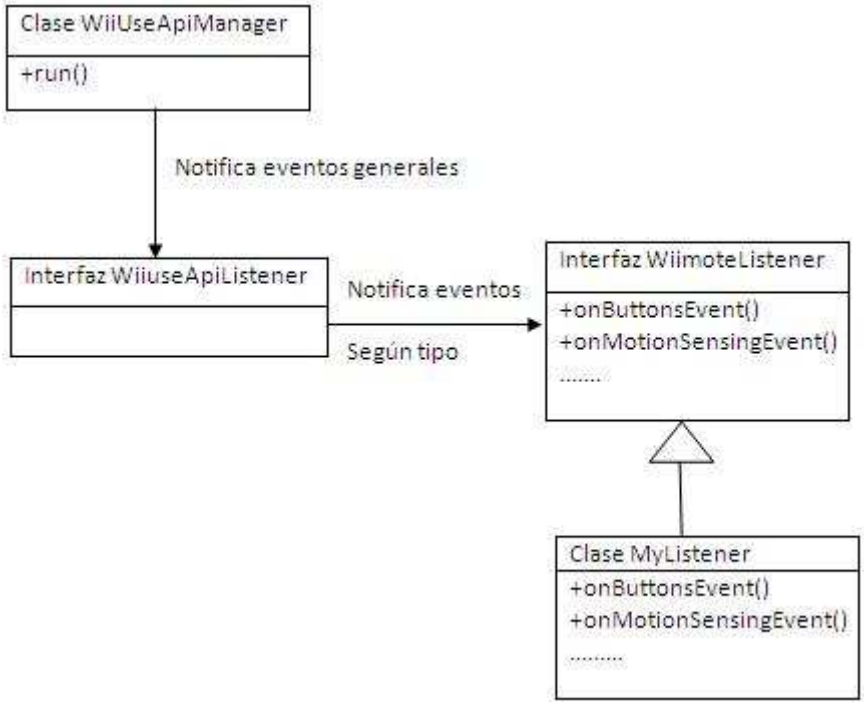


Figura 5.2: Diagrama UML de la Aplicación Desarrollada

La clase **WiiUseApiManager**, Se encarga de lanzar el "polling" de eventos, en su método run().

WiiUseApiListener: Se encarga de escuchar los eventos que recoge WiiUseApiManager. La clase que implemente esta interfaz se encargará de llamar a los métodos adecuados dependiendo del tipo de evento escuchado.

WiimoteListener: La clase principal del proyecto tiene que implementar esta interfaz, que es la que define las acciones a llevar a cabo en la aplicación concreta cuando se producen los eventos recogidos por WiiUseApiListener.

MyListener: Clase principal del proyecto

Nuestra aplicación, en primer lugar, intentara detectar el mando Wiimote, para poder capturar sus movimientos y pulsaciones. Esta parte de la aplicación se detalla a continuación:

```
Wiimote[] wiimotes = WiiUseApiManager.getWiimotes(1, true);
//Detecta los mandos que haya

if (wiimotes.length==0){

	System.out.println(" ****MANDO WIIMOTE NO DETECTADO**** ");
	WiiUseApiManager.definitiveShutdown();
	System.exit(1);
}

else {

	System.out.println(" *****MANDO WIIMOTE 1 DETECTADO**** ");

	Wiimote wiimote = wiimotes[0];
	wiimote.activateMotionSensing(); // Activar sensor de movimiento
	wiimote.activateIRTRacking(); // Activar sensor por infrarrojos
	wiimote.setLeds(true, false, false, false);    // Activar leds
	wiimote.addWiiMoteEventListeners(new MyListener());
}
```

Una vez que nuestra aplicación ha detectado el mando Wiimote y para poder detectar si se ha realizado un movimiento o se ha pulsado algún botón, hacemos uso de 2 métodos definidos en la interfaz WiimoteListener. Estos métodos capturan la acción realizada por el usuario.

Una vez capturada la acción realizada por el usuario, se procede a asignar un número concreto a cada acción, para posteriormente enviar esta información a través del puerto serie.

Método 1: onButtonsEvent : Detecta la pulsación de botones.

```
// Método onButtonsEvent:
// Detecta los botones pulsados del mando Wiimote

public void onButtonsEvent(WiimoteButtonsEvent arg0) {


	if (arg0.isButtonBHeld()){    // Detecta si botón B esta pulsado

		boton=true;}

	else{

		boton=false;
	}


	if (arg0.isButtonAPressed()){
```

```
            System.out.println("He pulsado el Boton A");
            char[] datos={7};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonOnePressed()){

            System.out.println("He pulsado el Boton 1");
            char[] datos={5};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }


      if (arg0.isButtonTwoPressed()){

            System.out.println("He pulsado el Boton 2");
            char[] datos={6};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }



      if (arg0.isButtonMinusPressed()){

            System.out.println("He pulsado el Boton -");
            char[] datos={};
            enviarDatos(datos);
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }


      if (arg0.isButtonPlusPressed()){

            System.out.println("He pulsado el Boton +");
            char[] datos={};
            enviarDatos(datos);
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonHomePressed()){

            System.out.println("He pulsado el Boton Home");
            char[] datos={};
            enviarDatos(datos);
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonUpPressed()){
```

```
            System.out.println("He pulsado el Boton Izquierda");
            char[] datos={4};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonDownPressed()){

            System.out.println("He pulsado el Boton Derecha");
            char[] datos={3};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }


      if (arg0.isButtonLeftPressed()){

            System.out.println("He pulsado el Boton Detras");
            char[] datos={2};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }



      if (arg0.isButtonRightPressed()){

            System.out.println("He pulsado el Boton Delante");
            char[] datos={1};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }


}     // Fin Método onButtonsEvent
```

Método 2: onMotionSensingEvent: Detecta el movimiento del Mando Wiimote.

```
// Método onMotionSensingEvent:
// Detecta el Movimiento del mando Wiimote


public void onMotionSensingEvent(MotionSensingEvent arg0) {

      Orientation orientation =arg0.getOrientation();
      vertical=(int)orientation.getPitch();
      horizontal=(int)orientation.getRoll();

// Si inclino el mando y tengo pulsado el botón B:

      if((vertical<=-55 && boton==true)){

            System.out.println("Estoy girando a la derecha");
            char[] datos={3};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
      }


      if((vertical>60 && boton==true)){

            System.out.println("Estoy girando a la izquierda");
            char[] datos={4};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
      }


      if((horizontal<=-50 && boton==true)){

            System.out.println("Estoy inclinando el Mando hacia
atras");
            char[] datos={2};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
      }


      if(horizontal>50 && boton==true){

            System.out.println("Estoy inclinando el Mando hacia
delante");

            char[] datos={1};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
```

```
	}
}// Fin Método onMotionSensingEvent
```

El envío de la información, se realiza mediante el Puerto Serie, para ello, hemos de configurar el Puerto de comunicación y la velocidad de transmisión, necesarios para que el envió se realice de forma satisfactoria.

Haremos uso de la librería Giovynet [7], la cual queda detallada en el Anexo A.

```
// Método enviar datos

public void enviarDatos(char[] data){

	try{

		listar = puerto.getFreeSerialPort();
		Parameters parameters = new Parameters();
		parameters.setPort("COM5");
		parameters.setBaudRate(Baud._4800);
		Com com = new Com(parameters);
		com.sendArrayChar(data);
		com.close();

	}

	catch(Exception e){}
}
```

Para una información más detallada acerca de la Aplicación desarrollada, consulte Anexo F

5.5 Aplicación Parte 2.

Esta parte de la Aplicación, está desarrollada en lenguaje RoboBasic y es la encargada de leer los valores recibidos por el puerto serie, y ejecutar una rutina en función del valor recibido, para así poder ejercer un control absoluto del robot.

Esta aplicación, debe cargarse previamente en el Robot Robonova-I. Mediante el comando Run All.

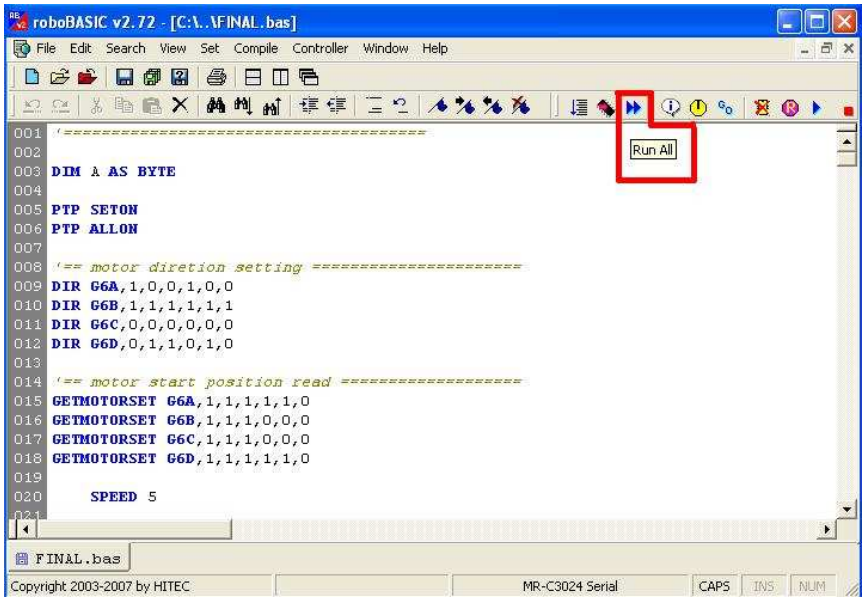


Figura 5.3: Comando Run All de RoboBasic

Esta parte de la Aplicación lee el puerto serie, y dependiendo del valor recibido, ejecuta una rutina u otra, la cual contiene toda la información acerca de los movimientos del Robonova-I.

MAIN:

```
ERX 4800, A, PROGRAMA  ' LEE EL PUERTO SERIE
```

PROGRAMA:

```
IF A = 6 THEN

  GOSUB RUTINAN1            ' CAMINA HACIA DELANTE
  DELAY 1000
  GOSUB POS_INICIAL
  A=100
```

```
ELSEIF A= 24 THEN

  GOSUB RUTINAN2          ' CAMINA HACIA ATRAS
  DELAY 1000
  GOSUB POS_INICIAL
  A=100

ELSEIF A =30 THEN

  GOSUB RUTINAN3          ' GIRA A LA DERECHA
  DELAY 1000
  GOSUB POS_INICIAL
  A=100

ELSEIF A =96 THEN
  GOSUB RUTINAN4           ' GIRA A LA IZQUIERDA
  DELAY 1000
  GOSUB POS_INICIAL
  A=100

ELSEIF A =102 THEN '102

  GOSUB RUTINAN5          ' VUELO
  DELAY 1000
  GOSUB POS_INICIAL
  A=100

ELSEIF A =120 THEN

  GOSUB RUTINAN6          ' PUNCH
  DELAY 1000
  SPEED 10
  GOSUB POS_INICIAL
  A=100

ELSEIF A =126 THEN        ' REVERENCIA
  GOSUB RUTINAN7
  DELAY 1000
  GOSUB POS_INICIAL
  A=100

ELSEIF A=100 THEN

 GOTO MAIN
```

```
  ENDIF

  GOTO MAIN

SALIR:
  END
```

Nuestra Aplicación utiliza siete rutinas diferentes, numeradas RUTINAN1, hasta RUTINAN7.

Estas rutinas, son las encargadas de realizar los movimientos del robot, para ver estas rutinas con más detalle consulte el Anexo G

CAPITULO 6

Conclusiones

Este Proyecto Fin de Carrera ha cumplido su objetivo general: el estudio realizado sobre la utilización de nuevos dispositivos como alternativas de interfaz de control a bajo coste para sistemas teleoperados y la creación de una aplicación capaz de controlar el robot Robonova I mediante el mando Wiimote.

Si bien es cierto la necesidad de un cable Puerto Serie para la comunicación eficaz de esta aplicación. Podría ser objeto de estudio, una futura aplicación, capaz de controlar el Robonova I de manera inalámbrica, instalándole un modulo Bluetooth al Robot Humanoide Robonova I

El Futuro de muchos sectores, tanto en el ámbito industrial como en el personal, pasa por el control teleoperado de Robots, y este proyecto sirve como iniciación al mundo de la Teleoperacion, así como de adquisición de conceptos básicos para futuros desarrollos teleoperados.

El robot Humanoide acapara gran parte de la robótica actual, y en el futuro serán capaces de realizar diversidad de tareas incapaces de realizarse en este momento. Esta investigación profesional, se une al interés cultural y emocional que nos llevara a construir maquinas que parezcan y trabajen como humanos.

REFERENCIAS

[1] Wikipedia. *Wii*.
http://en.wikipedia.org/wiki/Wii

[2] Robonova
http://www.superrobotica.com/Robonova.htm

[3] Teleoperacion
http://www.slideshare.net/newtonrules/telecontrol-haptico

[4] Robots
http://www.miportal.edu.sv/sitios/operacionred2008/OR08034417/clasificacion.html

[5] Wikipedia. *Wii Remote.*
http://en.wikipedia.org/wiki/Wii_Remote

[6] Google Code. *Wiiusej*.
http://code.google.com/p/wiiusej/

[7] Giovynet
http://java.giovynet.com/Giovynet/

[8] Manual Robonova

[9] Abstrakraft. *CWiid.*
http://abstrakraft.org/cwiid/wiki/libcwiid

[10] WiiYourself!.
http://wiiyourself.gl.tter.org/

[11] WiiLi. *WiiremoteJ*.
http://www.wiili.org/WiiremoteJ

[12] SourceForge. *motej – A Wiimote Library for Java.*
http://motej.sourceforge.net/index.html

[13] Carl Kenner. *GlovePIE*.
http://carl.kenner.googlepages.com/glovepie

[14] Winsoftmagic. Win Remote Pc
http://www.winsoftmagic.com/winremotepc.html

[15] Sourceforge. Darwiin Remote
http://darwiin-remote.sourceforge.net/

ANEXOS

A N E X O 

Librerías Utilizadas y otras Aplicaciones

1. Descripción de la Librería Wiiusej

Wiiusej es una API Java para el acceso al periférico Wiimote de la consola Wii de Nintendo. Se basa en una API C llamada WiiUse que accede directamente al bluetooth stack de nuestro sistema. La diferencia de Wiiusej con otras APIs Java reside en que no se basa en el paquete javax.bluetooth (implementación del estándar JSR-82), con lo cual resulta ser un mecanismo más eficiente para acceder al mando al estar implementada sobre un lenguaje de bajo nivel como es C.

A continuación se detallan las Interfaces y clases más significativas de esta API:

1.1 Interfaz WiimoteListener

Esta es la interfaz a implementar para escuchar los eventos que ocurren con el Wiimote.

Los métodos más significativos se detallan a continuación:

onButtonsEvent

void onButtonsEvent (WiimoteButtonsEvent e) : Método que es llamado cuando se produce un evento de botón.

Parámetros: e – El ButtonEvent con los últimos datos acerca de los botones del Wiimote.

onIrEvent

Void onIrEvent (IREvent e): Método que es llamado cuando se produce un evento de infrarrojos.

Parámetros: e – El IREvent con los puntos de vista IR

onMotionSensingEvent

Void onMotionSensingEvent (MotionSensingEvent e): Método que es llamado cuando se produce un evento de detección de movimiento.

Parámetros: e – El sensor de movimiento con la orientación y la aceleración.

onExpansionEvent

Void onExpansionEvent (ExpansionEvent e): Método que es llamado cuando se produce un evento de expansión.

Parámetros: e – El caso de la expansión que se produjo.

OnStatusEvent

void OnStatusEvent (StatusEvent e): Método que es llamado cuando se produce un evento de estado. Un evento de estado se produce cuando preguntamos si “un controlador de expansión ha sido conectado” o “un controlador de expansión ha sido desconectado” Aquí es donde se pueden obtener los diferentes valores de la configuración de los parámetros del Wiimote.

Parámetros: e – El evento de estado.

onDisconnectionEvent

void onDisconnectionEvent (DisconnectionEvent e): Método que es llamado cuando se produce un evento de desconexión Un evento de desconexión ocurre cuando el Wiimote ha sido apagado o la conexión se interrumpe

Parámetros: e – El caso de la desconexión.

onNunchukInsertedEvent

void onNunchukInsertedEvent (NunchukInsertedEvent e): Método que es llamado cuando se produce la conexión de un Nunchuk.

Parámetros: e – El evento de la conexión del Nunchuk

onNunchukRemovedEvent

void onNunchukRemovedEvent (NunchukRemovedEvent e): Método que es llamado cuando se produce la desconexión de un Nunchuk.

Parámetros: e – El evento de la desconexión del Nunchuk

onGuitarHeroInsertedEvent

void onGuitarHeroInsertedEvent (GuitarHeroInsertedEvent e): Método que es llamado cuando se produce la conexión del Guitar Hero.

Parámetros: e – El evento de la conexión del Guitar Hero

onGuitarHeroRemovedEvent

void onGuitarHeroRemovedEvent (GuitarHeroRemovedEvent e): Método que es llamado cuando se produce la desconexión del Guitar Hero.

Parámetros: e – El evento de la desconexión del Guitar Hero

onClassicControllerInsertedEvent

void onClassicControllerInsertedEvent (ClassicControllerInsertedEvent e): Método que es llamado cuando se produce la conexión del Classic Controller

Parámetros: e – El evento de la conexión del Classic Controller

onClassicControllerRemovedEvent

void onClassicControllerRemovedEvent (ClassicControllerRemovedEvent e): Método que es llamado cuando se produce la desconexión del Classic Controller

Parámetros: e – El evento de la desconexión del Classic Controller

1.2 Clase WiiuseApi

Clase para manipular la API Wiiusej. Entre sus métodos, destacan:

Continuous

`Void activateContinuous(int id)`: Hace que el Wiimote genere un evento cada vez

`void deactivateContinuous(int id)`:Desactiva el método anterior

IRTracking

`void activateIRTracking(int id)`: Activa el puerto IR de seguimiento en el Wiimote pasándole un identificador

`void deactivateIRTracking(int id)`: Desactiva el puerto IR de seguimiento en el Wiimote pasándole un identificador

MotionSensing

`void activateMotionSensing(int id)`: Activa el sensor de movimiento del Wiimote pasándole un identificador

`void deactivateMotionSensing(int id)`:Desactiva el sensor de movimiento del Wiimote pasándole un identificador

Rumble

`void activateRumble(int id)`: Activa la vibración sobre el Wiimote pasándole un identificador

`void deactivateRumble(int id)`:Desactiva la vibración sobre el Wiimote pasándole un identificador

Smoothing

`void activateSmoothing(int id)`: Hace que los acelerómetros den resultados más suaves.

`void deactivateSmoothing(int id)`: Desactiva el método anterior

setLeds

`void setLeds(int id, boolean led1, boolean led2, boolean led3, boolean led4)`: Establece el estado de los leds del Wiimote.

closeConnection

`void closeConnection(int id)`:Cierra la conexión del Wiimote pasándole un identificador

1.3 Interfaz WiiUseApiListener

Esta es la interfaz a implementar para escuchar los eventos de la WiiUse API.

1.4 Clase WiiUseApiManager

Clase que maneja el uso de la Librería API Wiiusej

1.5 Crear una aplicación con Wiiusej

1. Copiar los archivos de la librería WiiUse (.dll si estamos en Windows o .so si estamos en Linux) en la carpeta principal de nuestro proyecto Java.

2. Crear una clase (MyListener) que implemente el interfaz WiimoteListener).

3. En la clase main:

Wiimote[] wiimotes = WiiUseApiManager.getWiimotes(X, true);

Esto detecta los mandos que haya. Se encarga de lanzar la hebra de WiiUseApiManager. Wiimote es una clase que implementa el interfaz WiiUseApiListener, así que lo que devuelve es un array de listeners del WiiUseApi.

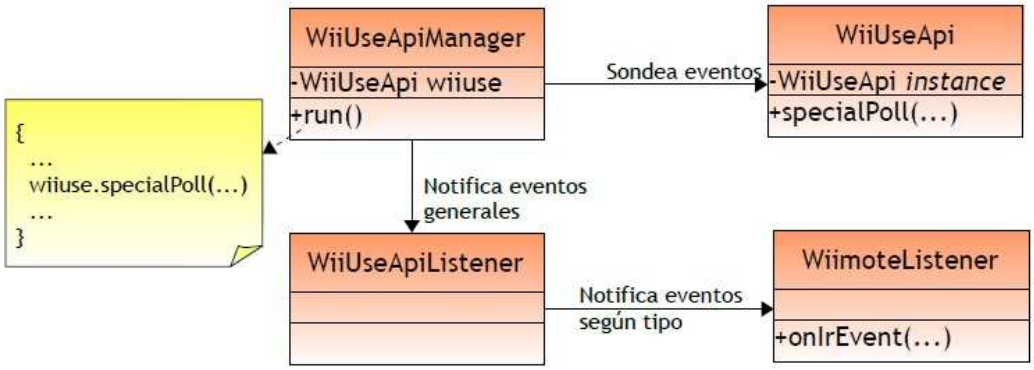
El parámetro booleano “true” hace que se produzca un sonido la primera vez que se detectan los Wiimotes.

4. En la clase main:

Wiimote[0].addWiiMoteEventListeners(new MyListener());

Registra el primer Wiimote con nuestro listener implementado en el primer paso. Cuando el mando lance un evento, sucederá lo que hayamos implementado en los métodos de esta clase.

A continuación se muestra un ejemplo grafico del uso de la librería Wiiusej



Ejemplo grafico de la librería Wiiusej

1.6 Pruebas con Wiiusej

Para probar la API, ésta incluye clases que implementan una interfaz gráfica. Esta GUI se lanza desde la clase Main que viene con la API, o ejecutando **wiiusej.jar**

Interfaz Grafica de la librería Wiiusej

2. Descripción de la librería Giovynet

Giovynet es una solución de código abierto Java que se utiliza para controlar los puertos serie RS-232 en conexión.

Giovynet trabaja bajo los sistemas operativos Linux o Windows, que consta de tres archivos:

El archivo". jar" identifica el sistema operativo;

Los otros dos archivos, contienen la lógica para manipular el puerto serie de acuerdo con el sistema operativo si es Linux, utiliza las funciones de archivo ". os ", pero si es Windows, utiliza las funciones de archivo ". dll".

A continuación se muestra la definición de las clases así como sus métodos más utilizados:

2.1 Clase SerialPort

Clase que sirve para el manejo de la comunicación del puerto serie. Su método más representativo es:

`getFreeSerialPort`

`getFreeSerialPort()`: Obtiene los puertos serie libres

2.2 Clase Parameters

Clase que establece los parámetros para la comunicación por puerto serie. Sus métodos más utilizados son:

setPort

setPort (java.lang.String puerto): Establece el nombre del puerto.

setBaudRate

setBaudRate (transmisión en baudios): Establece la tasa de baudios

2.3 Clase Com

Esta Clase representa un puerto serie por defecto. Su constructor seria:

public **Com**(Parameters parameters)

Sus métodos más habituales son:

sendString

`Void sendString(java.lang.String data)`: Envia una cadena de caracteres; Solo valido en el S.O. Windows

Void sendSingleData

`Void sendSingleData(char data)` : Envía un carácter del tipo Char.

Void `sendSingleData(int data)`: Envía un sólo dato, como representación de entero o la representación hexadecimal con prefijo de Ox.

receiveSingleString

receiveSingleString (): Lee un carácter y lo devuelve como tipo String.

receiveSingleChar

receiveSingleChar (): Lee un carácter y lo devuelve como tipo char.

Close

close() : Cierra el puerto serie.

3. Descripción Lenguaje RoboBasic

RoboBASIC está basado en el lenguaje de programación BASIC y está diseñado específicamente para el control de los controladores de la serie MR-C para gestión de Robots.

Es un lenguaje educativo que mejora el lenguaje de programación BASIC para permitir el control de Robots.

Es compatible con MS Windows 98, ME, 2000 y XP

3.1 Sumario de Comandos

RoboBASIC [8] es un exclusivo lenguaje de programación diseñado para controlar robots. Los comandos necesarios para controlar un robot han sido añadidos al lenguaje BASIC.

(2) Este símbolo indica que el comando solo se puede ejecutar en controladores de la serie MRC2000.

(3) Este símbolo indica que el comando solo puede ejecutar en controladores de la serie MR-C3000.

Comandos usados para declaraciones/definiciones

DIM Declara una variable
AS Asigna una variable durante su declaración
CONST Declara una constante
BYTE Se define una variable como tipo byte en su declaración
INTEGER Se define una variable como tipo integer (entero) en su declaración

Comandos para el control de flujo

IF Comienzo de una sentencia condicional
THEN Se ejecuta la siguiente sentencia (o sentencias) cuando la condición es cierta
ELSE Se ejecuta la siguiente sentencia (o sentencias) cuando la condición es falsa.
ELSEIF Comienzo de otra sentencia condicional.
ENDIF Fin de una sentencia condicional
FOR Comienza un bucle.
TO Asigna el rango de iteraciones de un bucle. NEXT Sentencia de cierre de ejecución de bucle
GOTO Divide el flujo del programa.
GOSUB Invoca a una sub-rutina
RETURN Sale (retorna) de una sub-rutina
END Finaliza la ejecución del programa. STOP Detiene la ejecución del programa .
RUN Ejecuta un programa (continua).
WAIT Espera hasta que se complete el programa.
DELAY Hace una pausa en el programa durante un tiempo.
(2) BREAK Detiene la ejecución del programa y pasa a modo de depuración.

Comandos de entrada y salida de señales digitales

IN Lee las señales de un puerto de entrada.
OUT Envía una señal a un puerto de salida
BYTEIN Lee una señal byte de un puerto de entrada
BYTEOUT Envía una señal byte a un puerto de salida.
(2) INKEY Tecla (clave) proveniente de un puerto de entrada.
STATE Estado del puerto de salida
PULSE Envía una señal (de pulso) al puerto de salida.
TOGGLE Invierte el estado del puerto de salida.
(3) KEYIN Toma la entrada del teclado analógico (de entrada).

Comandos de memoria

PEEK Lee datos del controlador de la RAM.
POKE Escribe datos en el controlador de la RAM.
ROMPEEK Lee datos del controlador externo EEPROM RAM.
ROMPOKE Escribe datos en el controlador EEPROM RAM.

Comandos para el LCD

LCDINIT Inicializa el módulo LCD.
CLS Borra todos los caracteres del módulo LCD.
LOCATE Define la posición de caracteres en el módulo LCD.
PRINT Muestra caracteres en el módulo LCD.
FORMAT Define el formato a mostrar en el módulo LCD
CSON Hace que aparezca el cursor en el módulo LCD.
CSOFF Oculta el cursor del módulo LCD.
CONT Define el contraste de los caracteres en el módulo LCD.
DEC Envía un valor decimal al módulo LCD.
HEX Envía un valor hexadecimal al módulo LCD.
(3) BIN Envía un valor binario al módulo LCD

Operadores lógicos

AND Usa la operación lógica AND.
OR Usa la operación lógica OR.
MOD Calcula el módulo de una operación aritmética.
XOR Usa la operación lógica XOR.
(3) NOT Invierte todos los bits

Comandos para el control del motor

ZERO Define el punto 0 (neutro) del servo.
MOTOR Activa el puerto de salida de un servo.
MOTOROFF Desactiva el puerto de salida de un servo.
MOVE Maneja varios servos al mismo tiempo.
SPEED Define la velocidad del servo.
(2) ACCEL Define la aceleración del servo.
DIR Define la dirección (sentido) del motor del servo.
PTP Activa/desactiva varias operaciones del control del motor.
SERVO Controla el servo.
PWM Define la amplitud del pulso de un motor DC.
(2) FASTSERVOManeja el servo a la velocidad máxima.
(3) HIGHSPEED Activa/desactiva la velocidad máxima del servo.
(3) MOVEPOS Mueve el grupo de servos definidos con POS.
(3) POS Selecciona una posición especifica del robot.
(3) FPWM Cambia la amplitud y la frecuencia del pulso
(3) MOVE24 Maneja los 24 servos a la vez.
(3) INIT Define la posición inicial
(3) MOTORIN Lee la posición actual del servo (valor).
(3) AIMOTOR Configuración de uso del motor AI.
(3) AIMOTOROFF Cancela el motor AI.
(3) AIMOTORIN Lee la posición actual del motor AI (valor).
(3) SETON Configuración para usar la función “setup ”
(3) SETOFF Cancela la función “set up”.
(3) ALLON Función de configuración para todos los servos.
(3) ALLOFF Cancela la función de config. de todos los servos.
(3) GETMOTORSET Lee los valores de un servo y mantiene la posición.

Parámetros para asignación de grupos de motores

(3) G6A Asigna los servos #0~#5 al grupo A.
(3) G6B Asigna los servos #6~#11 al grupo B.
(3) G6C Asigna los servos #12~#17 al grupo C.
(3) G6D Asigna los servos #18~#23 al grupo D.
(3) G6E Asigna los servos #24~#29 al grupo E.
(3) G8B Asigna los servos #8~#15 al grupo B.
(3) G8C Asigna los servos #16~#23 al grupo C.
(3) G8D Asigna los servos #24~#31 al grupo D.
(3) G12 Asigna los servos #0~#11.
(3) G16 Asigna los servos #0~#15.
(3) G24 Asigna los servos #0~#23.
(3) G32 Asigna los servos #0~#31.

Comandos para el control de sonidos

(2) BEEP Emite una alerta con el PIEZO.
(2) SOUND Reproduce una frecuencia con el PIEZO.
(2) PLAY Reproduce un sonido con el PIEZO.
(3) MUSIC Reproduce música con el PIEZO.
(3) TEMPO Define el ritmo de un sonido.

Comandos para comunicaciones con el exterior

(2) RX Recibe una señal RS-232 por el puerto RX.
(2) TX Envía una señal RS-232 por el puerto TX.
(2) MINIIN Recibe una señal minibus por el puerto de comunicaciones mini.
(2) MINIOUT Envía una señal minibus por el puerto de comunicaciones mini.
(3) ERX Recibe una señal RS-232 por el puerto RX.
(3) ETX Envía una señal RS-232 por el puerto TX.

Comandos para el proceso de señales analógicas

(3) AD Lee señales analógicas por el puerto AD.
(3) REMOCON Lee el valor de una tecla por el control remoto infrarrojos.
(3) SONAR Lee la distancia desde el puerto del sensor de ultrasonidos
RCIN Lee el valor de entrada del controlador remoto RC.
(3) GYRODIR Define la dirección del giróscopo.
(3) GYROSET Asigna un giróscopo a un servo.
(3) GYROSENSE Define la sensibilidad (ganancia) de un giróscopo.

Procesado de comandos

ON...GOTO Salto dependiendo del valor de una variable.

Otros comandos

RND Crea un número aleatorio.
REMARK Comentario

Comandos de intención

$DEVICE Configura el controlador para ser manejado por el programa en ejecución.
(3) $LIMIT Limita el recorrido de un servo.

4. Información sobre otras Librerías

A día de hoy existen diversas librerías y drivers para poder utilizar el *Wiimote* con distintos lenguajes y sistemas operativos, así como para manejar el puerto serie del ordenador.

4.1 Librerías para utilizar el Wiimote:

WiimoteLib (C)

Librería desarrolada por Brian Peek Para facilitar la gestión del control del mando desde C#.

WiimoteLib es una librería de dominio público desarrollada en C# que permite obtener todos los datos enviados por el Wiimote de una forma sencilla.

CWiid (C)

CWiid [9] como cuyo nombre indica, está realizada en lenguaje C y orientada a sistemas *Linux*. Actualmente está en una fase de desarrollo muy temprano con una API muy reducida y más complicada de usar que otras. Soporta funciones muy básicas del *Wiimote*.

WiiYourself! (C++)

WiiYourself! [10] es una librería basada en la creada por *Brian Peek* orientada a C++ y extendida en funcionalidad. Además de las funciones que aporta la librería original, añade algunas novedades como:

Estimación de la orientación: Mediante los métodos añadidos es posibledetectar el ángulo del *Wiimote* de una manera más sencilla.

Mayor soporte de gestores Bluetooth: Es capaz de utilizar cualquier programa gestor de conexiones Bluetooth.

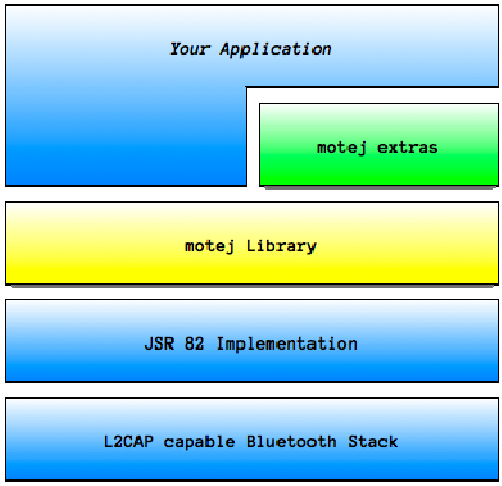
WiiremoteJ (Java)

Wiiremotej [11] es muy similar a motej, también requiere una implementación del estándar JSR82 y el listado de funcionalidades es idéntico pero añadiendo algunas funciones como soporte para extensiones del mando y del nuevo dispositivo wii Balance Board

Motej

Motej [12] es una librería Java para la comunicación con *Wiimote*, bajo licencia ASL 2.0.
Está basada en dos paquetes: librería y extras. La librería ofrece acceso básico al *Wiimote*. Los extras por su parte extienden la funcionalidad básica ofrecida por la librería añadiendo funciones más complejas para facilitar el trabajo al programador.

Como se puede observar, la librería depende de una implementación del estándar de Java *JSR82*, por lo que se requiere una API adicional para el acceso al dispositivo. El módulo de extras accede a las funciones básicas de la librería para extender la funcionalidad de la misma.



Libreria Motej

motej ofrece las siguientes funcionalidades, tanto para comprobar el estado del como para ejecutar acciones sobre el mismo:

Descubrimiento y conexión a dispositivos
Cámara de infrarrojos.
Acelerómetro.
Botones.
Activación/desactivación de la función de vibración.
Encendido/apagado de
Acceso a la memoria EEPROM del
Acceso a los registros del
Acceso al estado del
Acceso a los datos de calibración.

4.2 Librerías para utilizar el Puerto Serie:

RXTX (Java): Librería desarrollada en Java para Windows, inicialmente era compatible con la librería javacomm.

Javacomm (Java): Librería desarrollada inicialmente en Windows. La versión 3.0 es incompatible para Windows y está desarrollada para ser utiliza en un sistema operativo Linux Destaca porque contiene un control de errores efectivo

Lnxcomm (C): Esta librería está completamente desarrollada en Lenguaje C. Esta diseñada para brindar un apoyo a los programadores que estén relacionados con el diseño y construcción de hardware. LnxComm nos permite crear una conexion con el puerto serie.

Otra ventaja de esta librería es que nos permite crear programas que podrán ser compilados en sistemas operativos Linux y Windows brindando así mayor portabilidad a los programas desarrollados

5. Aplicaciones mediante Wiimote

5.1 Glovepie

GlovePIE [13] (Glove Programmable Input Emulator) fue inicialmente creado para la emulación de teclados, ratones y joysticks utilizando guantes de realidad virtual (o VR) modelo P5. Se trata de un intérprete de *scripts* (con extensión *.PIE*) con un lenguaje específico parecido a *Basic*, para la asignación de acciones del guante a pulsaciones del teclado o movimientos de un supuesto joystick o ratón. Actualmente este programa no sólo sirve para la emulación de guantes VR P5 si no que es compatible con una gran cantidad de dispositivos entre los que se incluyen otros modelos de guantes VR, micrófonos, joysticks o gamepads, dispositivos de juego que funcionen por puerto paralelo, máquinas de remo, dispositivos MIDI, y lo que es más interesante, compatibilidad total con *Wiimote*.

Este intérprete tiene dos cualidades que le han hecho cultivar un gran éxito en el mundo del *Wiimote* en PC. Por una parte realizar scripts para él es muy sencillo a la vez que completo. Cualquier persona que haya programado en cualquier lenguaje no tardará en adaptarse, e incluso los no programadores encuentran bastante fácil su utilización.

Los comentarios se ponen con // delante, así no lse interpreta como una acción.

La mayor parte de las líneas son simples asignaciones del tipo:

```
Mouse.RightButton = Wiimote.A     //Botón A del Wiimote equivale a click derecho
Mouse.LeftButton = Wiimote.B      //Botón B del Wiimote equivale a click izquierdo
```

Aunque también permite la creación de estructuras de control básicas como la clásica IF/ELSE y el uso de variables.

La otra ventaja es que no requiere realizar modificaciones en las aplicaciones o juegos que se deseen utilizar, ya que simplemente *mapea* acciones del *Wiimote* a acciones de teclado, ratón o joystick, por lo tanto es compatible con prácticamente todos los programas y juegos del mercado.

Por último, el éxito que ha cosechado ha hecho que existan cientos de *scripts* en Internet disponibles para su descarga y utilización gratuita, por lo que utilizar el *Wiimote* como dispositivo de juego para PC es muy sencillo.

A continuación se detallan varios Scripts de ejemplos utilizando Glovepie

// Script para un juego de carreras

```
//Controles de dirección

Left = Wiimote.Up              // Control para girar a la Izquierda
Right = Wiimote.Down           // Control para girar a la derecha

//Controles de aceleración

Up = Wiimote.One               // Control de Aceleracion
Down = Wiimote.Two                   // Control de Frenada
```

Con este script conseguiremos que, girando el Wiimote dejando la cruceta a la izquierda, al apretar el boton 1 aceleraremos y con el boton 2 frenaremos, y pulsando arriba o abajo en el Wiimote (que girado sería izquierda y derecha) giraremos el coche.

// Script para controlar el Reproductor Media Player Classic

```
// Controles de archivo

Ctrl+o = Wiimote.One                 // Abrir Archivo
Ctrl+l = Wiimote.Two                 // Cargar Subtitulos

// Controles de reproducción

Space = Wiimote.A              // Play-Pause
Period = Wiimote.B             // Stop
Ctrl+right = Wiimote.Right     // Avanzar 5 segundos
Ctrl+Left = Wiimote.Left       // Retroceder 5 segundos
Add = Wiimote.up               // Retraso de audio +10 ms
Subtract = Wiimote.Down        // Retraso de audio -10 ms

// Controles del reproductor
```

```
Ctrl+Enter = Wiimote.Home          // Pantalla Completa
Alt+X = Wiimote.B + Wiimote.A      // Salir del Media Player
Up = Wiimote.Plus                  // Subir Volumen
Down = Wiimote.Minus               // Bajar Volumen
```

Este script hace uso de todos los botones del Wiimote e incluso de una funcion usando 2 botones a la vez.

5.2 Win Remote PC

Win Remote PC [14] Control es un software gratuito para plataformas windows (para uso no comercial) utilizado para administrar y manejar una PC en forma remota, es decir para poder controlar una computadora como si estuvieses en tu propia máquina, ya sea desde internet o una red local.

Como la mayoría de los programas de este estilo, el software incluye dos programas: un cliente y un servidor. El primero es el que permite acceder a la máquina que se quiere controlar o administrar, y el segundo es el programa que debe estar corriendo o instalado en la misma para que el cliente pueda acceder.

Entre las características principales de Win Remote PC Control podemos:

- Enviar combinaciones de teclas a la pc, por ejemplo las teclas Ctrl+Alt+Supr para poder acceder rápidamente al administrador de tareas del sistema por ejemplo.

- Se pueden administrar múltiples escritorios mediante pestañas, para que sea más cómodo acceder a las distintas conexiones establecidas

- Utilizar para el acceso, la seguridad integrada de Windows o la del propio programa

- Tomar el control del mouse, del teclado y del escritorio

- Incluye su propio controlador de vídeo para una rápida visualización que proporciona casi el control en tiempo real con mayor rendimiento y velocidad

- Ejecutar a pantalla completa

- Envio de datos en forma encriptada utilizando los algoritmos BlowFish, TEA y otros.

- Poder indicar el modo de color entre los siguientes valores: 16 y 24 Bits para óptima visualización, de 4 , 8 y 12 bits en color, y también en blanco y negro

- Estadísticas de las conexiones : Ver las activas, las velocidades, los bytes enviados y recibidos y otros datos

- Realizar transferencias de archivos entre las máquinas conectadas

- Arquitectura de plugins, para poder extender las funciones y opciones del programa.

- Soporte para clipboard.

5.3 Darwiin Remote

La Aplicación DarwiinRemote [15] **nos permite controlar el ordenador de Apple Macintosh (Mac) con el** Wiimote, Posee varios modos de control:

Modo básico: Nos permite enlazar el mando con el Mac.

Modo avanzado: N**os permite monitorizar el acelerómetro** del mando y controlar el puntero del Mac con el mismo.

Con Darwiin Remote podemos usar **el Wiimote para controlar FrontRow en su totalidad.**

Front Row es una aplicación para los ordenadores de Apple Macintosh, que actúa como máscara para QuickTime, DVD Player y para las librerías de iTunes e iPhoto que permite a los usuarios navegar por los contenidos multimedia del ordenador

Darwiin Remote

A N E X O 

Documentación API WiiuseJ

Packages	
1	**Wiiusej**
2	**wiiusej.test**
3	**wiiusej.utils**
4	**wiiusej.values**
5	**wiiusej.wiiusejevents**
6	**wiiusej.wiiusejevents.physicalevents**
7	**wiiusej.wiiusejevents.utils**
8	**wiiusej.wiiusejevents.wiiuseapievents**

1- Package wiiusej

Class Summary	
Wiimote	Class that represents a wiimote.
WiiUseApi	Singleton used to manipulate WiiUse Api.
WiiUseApiManager	Class that manages the use of Wiiuse API.

2- Package wiiusej.test

Class Summary	
ClassicControllerGuiTest	This frame is used to display events from a classic controller.
CloseGuiTestCleanly	This class is used to close wiiusej cleanly.
GuitarHero3GuiTest	This frame is used to display events from a Guitar Hero 3.
Main	Main Class to launch WiiuseJ GUI Test.
NunchukGuiTest	This frame is used to display events from a nunchuk.
Tests	This class used to test WiiuseJ in text mode.
WiiuseJGuiTest	Gui class to test WiiuseJ.

3- Package wiiusej.utils

Class Summary	
AccelerationExpansionEventPanel	Panel to display Acceleration in a MotionSensingEvent from an expansion.
AccelerationPanel	This panel is used to watch raw acceleration values from a MotionSensingEvent.
AccelerationWiimoteEventPanel	Panel to display Acceleration in a MotionSensingEvent from a wiimote.
ButtonsEventPanel	This panel is used to see what buttons are pressed on the wiimote.
ClassicControllerButtonsEventPanel	This panel is used to display what happens on the classic controller.
GForceExpansionEventPanel	Panel to display GForce in a MotionSensingEvent from an expansion.
GForcePanel	This panel is used to watch gravity force values from a MotionSensingEvent.
GForceWiimoteEventPanel	Panel to display GForce in a MotionSensingEvent from a wiimote.
GuitarHero3ButtonsEventPanel	This panel is used to display what happens on the buttons of the Guitar Hero 3 controller.
GuitarHeroJoystickEventPanel	Panel to display Guitar Hero 3 controller joystick events.
IRPanel	This panel is used to see what the IR camera of the wiimote sees.
JoystickEventPanel	Panel to display joystick events.
NunchukJoystickEventPanel	Panel to display nunchuk joystick events.
OrientationExpansionEventPanel	Panel to display Orientation in a MotionSensingEvent from an expansion.
OrientationPanel	This panel is used to watch orientation values from a MotionSensingEvent.
OrientationWiimoteEventPanel	Panel to display Orientation in a MotionSensingEvent from a wiimote.

4- Package wiiusej.values

Class Summary	
GForce	Represents gravity force on each axis.
IRSource	Class used for IR sources.
Orientation	Class that represents the orientation of the wiimote.
RawAcceleration	Represents raw acceleration on each axis.

5- Package wiiusej.wiiusejevents

Class Summary	
GenericEvent	Abstract mother class representing an event with a wiimote id.

6- Package wiiusej.wiiusejevents.physicalevents

Class Summary	
ButtonsEvent	Class which represents a buttons event.
ClassicControllerButtonsEvent	Class which represents a buttons event from a Classic controller.
ClassicControllerEvent	This class represents the values from the classic controller and its events.
ExpansionEvent	Mother Class of all expansions event.
GuitarHeroButtonsEvent	Class which represents a buttons event from a Guitar Hero controller.
GuitarHeroEvent	This class represents the values from the GuitarHero and its events.
IREvent	Class which represents an IR event.
JoystickEvent	Class that stores values on a joystick Event.
MotionSensingEvent	Class which represents a motion sensing event.
NunchukButtonsEvent	Class which represents a buttons event from a Nunchuk.
NunchukEvent	This class represents the values from the joystick and its events.
WiimoteButtonsEvent	Class which represents a buttons event for a generic event.

7- Package wiiusej.wiiusejevents.utils

Interface Summary	
WiimoteListener	This is the interface to implement to listen to events from wiimotes.
WiiUseApiListener	This is the interface to implement to listen to events from the wiiuse API.

Class Summary	
EventsGatherer	This class is used to gather events during a call to the Wiiuse API.

8- Package wiiusej.wiiusejevents.wiiuseapievents

Class Summary	
ClassicControllerInsertedEvent	Event that represents the connection of a classic controller to a wiimote.
ClassicControllerRemovedEvent	Event that represents the disconnection of a classic controller from a wiimote.
DisconnectionEvent	Class representing a disconnection event.
GuitarHeroInsertedEvent	Event that represents the connection of a Guitar hero controller to a wiimote.
GuitarHeroRemovedEvent	Event that represents the disconnection of a guitar hero controller from a wiimote.
NunchukInsertedEvent	Event that represents the connection of a nunchuk to a wiimote.
NunchukRemovedEvent	Event that represents the disconnection of a nunchuk from a wiimote.
StatusEvent	Class used to represent a status event.
WiimoteEvent	Class that is a bean to be filled by the wiiuse API on an event that occurs on a wiimote.
WiiUseApiEvent	This class describes the structure of an event from the WiiUse API event.

A N E X O **C**

Manual Giovynet

How do you know which serial ports are free?

First instantiate an object of type giovynet.nativelink.SerialPort, then getFreeSerialPort() method is used to get a String list of free ports:

```
SerialPort serialPort = new SerialPort();
        List<String> portsFree = serialPort.getFreeSerialPort();
        for (String free : portsFree) {
                System.out.println(free);
        }
```

How to configure the serial port?

First create the object of type giovynet.serial.Parameters, which presents by "default" settings as follows:

port = COM1
baudrate = 9600
ByteSize = 8
StopBits = 1
parity = N (no)

To work with the previous configuration, it only necessary to instantiate an object of giovynet.serial.Com type, using as parameter in the Constructor, the giovynet.serial.Parameters instance. Thus opens the serial port "COM1" and ready to work.

```
Parameters parameter = new Parameters ();
Com = new Com Com (parameter);
```

To change the settings for "default" use the "set" methods of the object "parameter" before instantiating the class giovynet.serial.Com. For example, to configure the COM2 port at a speed of 460 800 bps, following these instructions:

```
Parameters param = new Parameters ();
param.setPort ("COM2");
param.setBaudRate (Baud._460800);
Com com2 = new Com (param);
```

How to send data?

To perform this task, use giovynet.serial.Com class; this has three methods presented below:

sendArrayChar(char [] data): void.data):

This method is used to send elements of an "array of type char." The time interval that each element is sent is determined by the method setMinDelayWrite (int milliseconds) from giovynet.serial.Paramgeters class. By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, the following code is used to send the elements in an array char stored, every 100 milliseconds:

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(100);
        Com com1 = new Com(param);
        char[] data = {'1','A','2','B'};
        com1.sendArrayChar(data);
        com1.close();
}
```

To run of this code, will send after 100 milliseconds the *char '1 '*, then another 100 milliseconds will send the *char' A '*, then another 100 milliseconds will send the *char'2'*, and so on in turn until the final char data stored in the array is sent.

sendSingleData(overloaded): void.

This method is used to send an element of type "char" or of type "String" or "Hex". The time interval that each element is sent is determined by the method setMinDelayWrite (int milliseconds) from giovynet.serial.Parameters class. By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, in the following code it shows how to send data *'a', 41, "S"*, at intervals of 100 milliseconds

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(100);
        Com com1 = new Com(param);
        com1.sendSingleData('a');
        com1.sendSingleData(41);
        com1.sendSingleData("S");
        com1.sendSingleData(0xff);
        com1.close();
}
```

sendString(String data): void.

This method is used to send elements of a "String". The time interval that each element is sent is determined by the method setMinDelayWrite (int milliseconds) from giovynet.serial.Parameters class. By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, in the following code it shows how to send the elements of the string "hello" every 50 milliseconds:

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);
        com1.sendString("hello");
        com1.close();
}
```

The implementation of the above code, will send after 50 milliseconds the element *"h"*, followed by another 50 milliseconds, will send the item *"e"* after another 50 milliseconds, will send the element *"l"*, and so on until the word "hello."

How to receive data?

To accomplish this task it is uses the class giovynet.serial Driver.Com, which has five methods presented below:

receiveSingleChar(): char.

This method is used to receive a element "char", after the time set by the method setMinDelayWrite (int milliseconds) of the giovynet.serial.Parameters class. By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds.

Not recommended for receiving control characters, such as <ACK>, <NACK>, <LF>, ... etc.
For example the following codes receive 10 elements "char" every 50 milliseconds, and print it the console:

```
public static void main(String[] args){

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);

        char data;
        for(int x=0; x<10; x++){
                data=com1.receiveSingleChar();
                System.out.println(data);
        }
        com1.close();
}
```

receiveSingleDataInt(): int.

This method is used to receive element "Int", after the time set by the method setMinDelayWrite (int milliseconds) of the giovynet.serial.Parameters class, By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds.

This method is recommended to receive control characters such as <ACK>, <NACK>,<LF>, ... etc. For example, the following code receives 10 elements "Int", every 50 milliseconds, then prints to the console:

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);

        int data;
        for(int x=0; x<10; x++){
                data=com1.receiveSingleDataInt();
                System.out.println(data);
        }
        com1.close();
}
```

Only Windows. receiveSingleString(): String

This method is used for receiving a element "String", then the time set by the method setMinDelayWrite(int milliseconds) of the giovynet.serial.Parameters class. By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds.

For example, the following code reads data after 50 milliseconds, and displays it the console:

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);

        String data;
        data=com1.receiveSingleString();
        System.out.println(data);
        com1.close();
}
```

receiveToString(int amount): String.

This method is use to receiving several single elements of type "String", and return them like group "String". The receipt of each element is made after the time set by the method setMinDelayWrite(int milliseconds) of the giovynet.serial.Parameters class. By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, the following code is used for receiving 10 elements type "String" every 50 milliseconds, these are stored in a variable and then we will print the variable in console.

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(50);
        Com com1 = new Com(param);

        String data;
        data=com1.receiveToString(10);
        System.out.println(data);
        com1.close();
}
```

receiveToStringBuilder(untilAmount int, StringBuilder, StringBuilder):

This method is used to receive several elements of type "String" and store them in an object of type StringBuilder". The number of elements to receive and the "StringBuilder" object are passed as parameter.

The receiving of each element is made after the time set by the method setMinDelayWrite(int milliseconds) of the giovynet.serial.Parameters class. By "default" the time set for Windows is 0 milliseconds, and Linux is 10 milliseconds. For example, the following code reads 20 elements "String" every 30 milliseconds and these stored in a StringBuilder object.

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        param.setMinDelayWrite(30);
        Com com1 = new Com(param);

        StringBuilder stringBuilder = new StringBuilder;
        com1.receiveToStringBuilder(20,stringBuilder);
        com1.close();
}
```

How to send control characters?

To send control characters are recommended to use the decimal or hexadecimal representation with the method sendSingleData () from giovynet.serial.Com class.

For example the following code sends the symbol ACK (Acknowledge):

```
public static void main (String [] args) throws Exception (
        Parameters param = new Parameters ();
        param.setPort ("COM1");
        param. setBaudRate (Baud._9600)
        Com1 = new Com Com (param);

        int ack = 6 / / The integer representation of ACK in ASCII code is 6.
        data = com1.sendSingleData (ack);
        )
        com1.close ();
)
```

How to receive control characters?

For to receive control characters, these should be obtained in decimal or hexadecimal representation, using the method receiveSingleDataInt () from giovynet.serial.Com class

For example, the following code, it prints on the screen "OK" when it receives the ACK symbol.

```
public static void main(String[] args)throws Exception{

        Parameters param = new Parameters();
        param.setPort("COM1");
        param.setBaudRate(Baud._9600);
        Com com1 = new Com(param);
        int ack=06; //The integer representation of ACK in ASCII code is 06.
        int data=0;

        while(true){
                data=com1.receiveSingleDataInt();
                if (data==ack){
                        System.out.println("OK");
                        break;
                }
        }
        com1.close();
}
```

ONLY WINDOWS. Set ActionListenerReadPort (event listener port for reading).

In Windows operating systems you can add an "event listener" to a class instance giovynet.serial.Com, for reading a port to with addActionListenerReadPor (ActionListenerReadPort actionListenerReadPort) method, the actionListenerReadPort parameter must be an object that implements the interface ActionListenerReadPort. This interface has two methods to be implemented.

The first is tryActionPerformed (Buffer buffer), this is executed if the data is received correctly, in this case, the data is stored in an object of type giovynet.serial.Buffer, through which these can be manipulated as objects of type Character Array, Integer Array, Integer List or StringBuffer, according to need. The second method is catchActionPerformed (Exception e) is executed if a problem occurs reading from the port.

For example, the following code sets up a ActionListenerReadPort at the port "COM1" for print to screen all incoming data (in integer representation), if any error occurs in receiving it prints the stack dump to the console and it close the port.

```
public class ListenerRead {
        private Com com1;

        public static void main(String[] args)throws {
                Parameters param = new Parameters();
                param.setPort("COM1");
                param.setBaudRate(Baud._9600);
                com1 = new Com(param);
                com1.addActionListenerReadPort(new ActionListenerReadPort()
                {

                public void tryActionPerformed(Buffer buffer) {
                        appendLineTextArea("\n<< ");
                        for (int i = 0; i < buffer.getBufferInIntegerList().size();
                        i++) {
                        System.out.print(buffer.getBufferInIntegerList().get(i));
                        }
                        buffer.bufferClear();
                }

                public void catchActionPerformed(Exception e) {
                com1.close();
                }

                });

                /** Other Routines Thread Main
                ...
                ...
                ...
                **/
        }
}
```

Setup "thread" to receive data independently of the "main thread”

In Java there are two ways to implement a "thread", one way is to inherit the class "Thread" and the other way is to implement the interface "Runnable" in both ways is necessary to implement the method "run ()" with the tasks you want to execute, which are independent to the main task or main thread. "

For example, the following code implements the interface "Runnable" to create a separate task to print to screen all characters received on port "COM2", if an error occurs then it prints the stack dump at the screen and it closes the port.

```
public class ReceiveThread implements Runnable {

        private Com com2;

        public static void main(String[] args)throws Exception{

                Parameters param = new Parameters();
                param.setBaudRate(Baud._460800);
                param.setPort("COM2");
                com2 = new Com(param);
                Thread threadReadPort = new Thread(ReceiveThread.this);
                threadReadPort.start();
                /** Other Routines Thread Main
                ...
                ...
                ...
                **/
        }

        /** Routines threadReadPort, independent from Thread Main **/

        public void run() {
                try {
                        int data = 0;
                        while (true) {
                        Thread.sleep(250);
                        data = com2.receiveSingleDataInt();
                        if (data !=
                        do {
                        System.out.println(data);
                        Thread.sleep(50);
                        data = com2.receiveSingleDataInt();
                        } while (data != 0);
                        }
                        }
                } catch (Exception e) {
        com2.close();
        }
        }
}
```

A N E X O **D**

Conexión Wiimote - PC

Conexión de *Wiimote* utilizando el gestor de *Windows XP*

El proceso de conexión del *Wiimote* con el gestor de conexiones *bluetooth* de *Windows XP* es muy sencillo:

1. Pulsar con el botón derecho del ratón sobre el icono del logo *bluetooth* de la barra de tareas y pinchar en "*Agregar dispositivo bluetooth"*.

2. En la ventana que aparecerá, seleccionar la casilla "*Mi dispositivo está configurado y listo para ser usado"*.

3. El programa comenzará a buscar dispositivos, mientras lo hace, mantener pulsados los botones *1* y *2* del *Wiimote.*

4. Seleccionar el *Wiimote* que aparece listado como "*Nintendo RVL-CNT-01"* y pulsamos en "*Siguiente*" mientras se mantienen pulsados los botones *1* y *2* del *Wiimote*.

5. En la siguiente ventana, seleccionar la opción "*No usar ninguna clave de paso"* mientras seguimos manteniendo pulsados los botones *1* y *2*.

Tras realizar todos los pasos anteriores, el *Wiimote* estará conectado y listo para utilizar las aplicaciones compatibles con este método.

Conexión de *Wiimote* utilizando *BlueSoleil*

Para utilizar *BlueSoleil* se deben seguir los siguientes pasos:

1. Instalar la aplicación.
2. Emparejar el *Wiimote* con *BlueSoleil.*

Instalar la aplicación

Instalar *BlueSoleil* no es una tarea más compleja que instalar cualquier otra aplicación destinada a usuarios sin muchos conocimientos.

Emparejar el *Wiimote* con *BlueSoleil*

Una vez instalado correctamente *BlueSoleil*, se debe emparejar el *Wiimote* con el mismo. Para ello pinchamos con el botón derecho del ratón en el icono azul que representa el logotipo de *bluetooth* de la barra de tareas y elegimos "*Pantalla de Vista Clásica*".

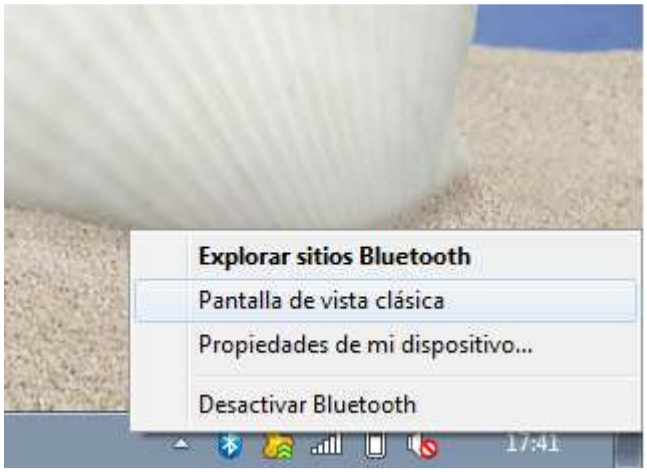


Imagen: Apertura de BlueSoleil

Aparecerá la siguiente ventana:

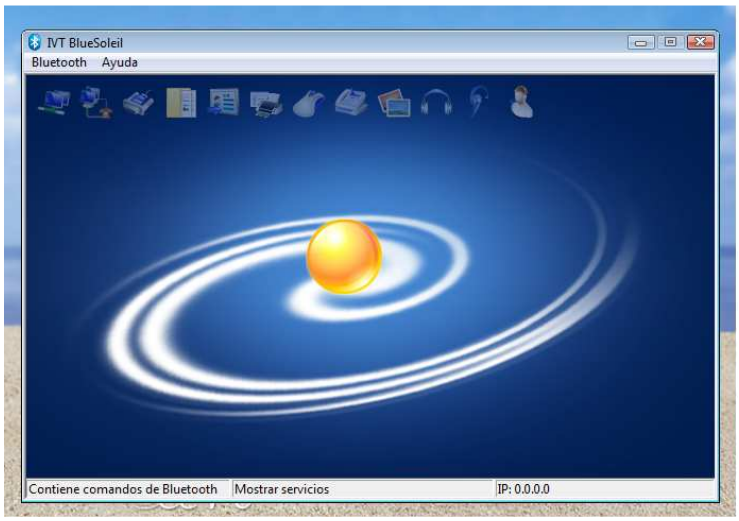


Imagen del programa BlueSoleil

Para emparejar el *Wiimote* con el programa, se deben seguir los siguientes pasos:

1. **Mantener pulsados simultáneamente los botones *1* y *2* del *Wiimote*.**

2. **Hacer doble *click* sobre la bola amarilla de *BlueSoleil*:** Tras ello aparecerá el *Wiimote* representado como un *joystick.*

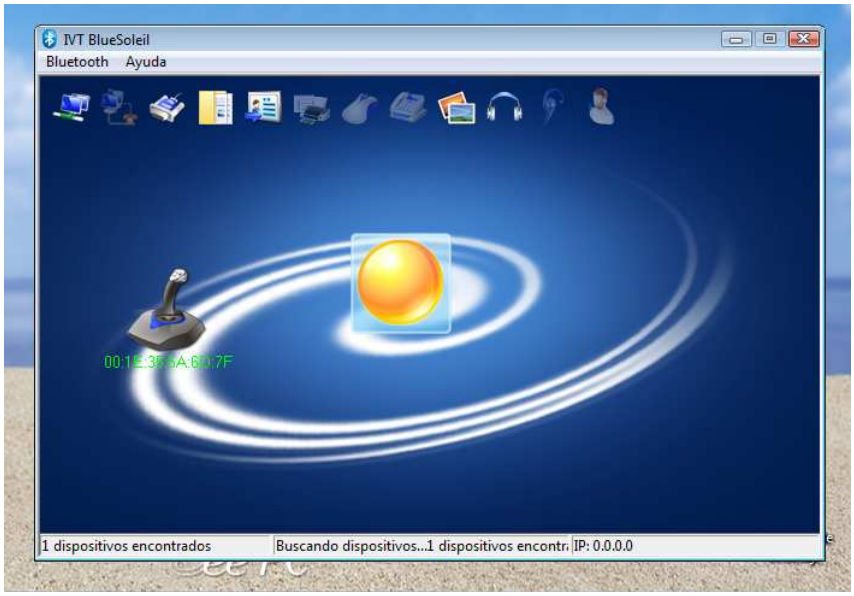


Imagen: Paso 2

3. **Hacer doble *click* sobre el *Wiimote*:** aparecerán iluminados arriba los servicios que proporciona el dispositivo. En nuestro caso se iluminará el ratón, que indica que el *Wiimote* puede usarse como dispositivo de entrada.

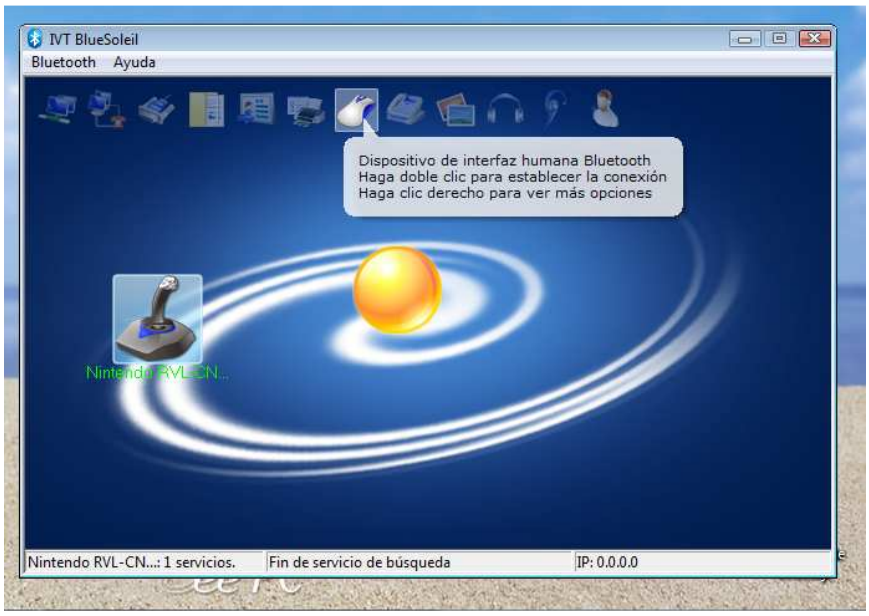


Imagen: Paso 3

4. **Hacer doble *click* sobre el icono del ratón:** A partir de ese momento, el *Wiimote* quedará emparejado con nuestro PC, listo para ser usado por nuestras aplicaciones.

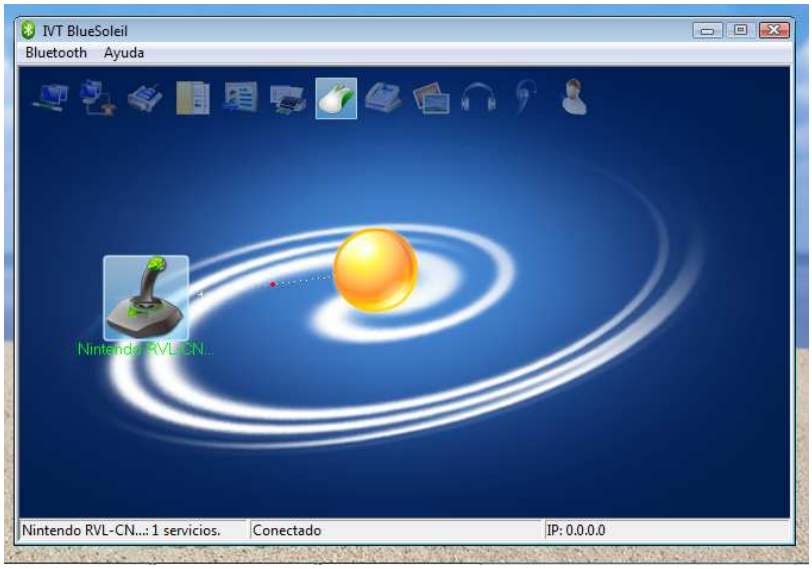


Imagen: Paso 4

A N E X O 

Norma UNE-EN 62115

El Robonova-1 es un robot humanoide de pequeño tamaño, alimentado por una batería de bajo voltaje, cuyo uso está destinado principalmente a aficionados a la robótica, aunque por sus características puede tener una función educacional e interactuar con niños, por tanto debe cumplir los requisitos de la norma UNE-EN 62115 Juguetes eléctricos. Seguridad.

Objeto y campo de aplicación

Esta norma trata de la seguridad de los juguetes que tienen al menos una función que depende de la electricidad.

Los juguetes que usan electricidad para funciones secundarias están dentro del campo de aplicación de esta norma.

Los transformadores de juguetes y los cargadores de baterías no se consideran juguetes, incluso si se suministran con ellos.

Si está previsto que el niño juegue también con el envase este se considera como parte del juguete.

Requisitos de la norma

- Los juguetes o su embalaje deben llevar las indicaciones siguientes:

  -El nombre, la marca comercial o la marca de identificación del fabricante o de su vendedor responsable

  -El modelo o referencia del tipo

  - Cuando el juguete está marcado, estas marcas deben ir colocadas sobre la parte principal.

Imagen: Nombre del Modelo y Fabricante

- Los juguetes a batería con baterías reemplazables deben estar marcados con:

- La tensión nominal de la batería, dentro o sobre el hueco para la batería

- El símbolo de corriente continua, si el juguete tiene un compartimento de baterías

- Deben suministrarse instrucciones que den información relativa a la limpieza y mantenimiento cuando ello es necesario para una utilización segura del juguete.

- Deben indicar que los transformadores o cargadores de batería utilizados con el juguete han de ser examinados regularmente para detectar deterioros en el cable, clavija, envolvente y otras partes, y que en caso de tales deterioros, no tienen que ser usados hasta que el daño haya sido reparado.

- Los juguetes deben acompañarse de instrucciones para el montaje si:

  Están previstos para ser montados por un niño

  Estas instrucciones son necesarias para la utilización segura del juguete

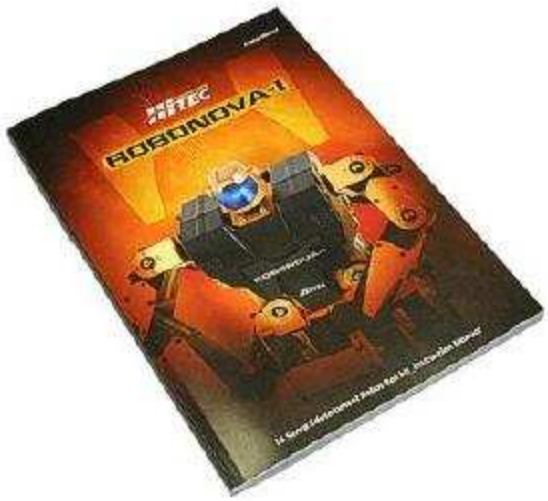


Imagen: Manual Robonova

- Los juguetes no deben alcanzar temperaturas excesivas durante el uso.

- Deben estar construidos de forma tal que sean evitados en la medida de lo posible los riesgos de incendio, daños mecánicos que afectan a la seguridad u otros peligros, como resultado de un uso negligente o de un fallo de un componente.

- El aislamiento eléctrico del juguete a la temperatura de funcionamiento debe ser adecuado.

- Los juguetes deben ser resistentes a la humedad.

- El aislamiento eléctrico del juguete a la temperatura ambiente debe ser el adecuado.

- Los juguetes deben ser juguetes a batería, juguetes con transformador o juguetes con doble alimentación. Su tensión de alimentación no debe exceder de 24 V.

- La tensión de servicio entre dos partes cualesquiera del juguete no debe sobrepasar 24 V cuando el juguete es alimentado a la tensión asignada.

Imagen: Batería del Robonova de 6 V

- Las baterías recargables colocadas en los juguetes no deben dejar gotear su contenido, cualquiera que sea la posición del juguete. El electrolito no debe ser accesible incluso si se tiene que usar una herramienta para retirar la tapa o una parte análoga.

- Los juguetes no deben ser alimentados por baterías conectadas en paralelo, a menos que una mezcla de baterías nuevas y usadas o la inserción invertida de las baterías no impida la conformidad con esta norma.

- Las partes no amovibles que protegen del contacto con las partes móviles o de las superficies calientes, o que impiden el acceso a los lugares donde una explosión o un incendio podrían originarse, deben ser fijadas de forma segura y deben soportar las solicitaciones mecánicas susceptibles de producirse durante el uso.

- No debe ser posible recargar las baterías cuando están en el juguete, excepto si:

-Para los juguetes donde la masa es inferior a 5 Kg, no es posible:

Retirar la batería sin romper el juguete

Cargar otras baterías

-Para otros juguetes:

La batería está fijada en el juguete

Se suministran los medios de conexión que aseguran una correcta polaridad durante la operación de carga

No es posible que el juguete funcione durante la operación de carga

- Los juguetes no deben incorporar motores serie que tengan una potencia superior a 20 W.

- Los juguetes no deben contener amianto.

- Los pasos de cables deben ser lisos y sin aristas vivas.

- Los cables y conductores deben estar protegidos de forma tal que no entren en contacto con asperezas, aletas de refrigeración u otras aristas susceptibles de dañar el aislamiento.

- Los orificios en las paredes metálicas para el paso de cables y conductores deben tener superficies lisas, estar bien redondeados o estar provistos de pasatapas.

- Cualquier contacto entre los cables y conductores y las partes móviles debe ser eficazmente impedido.

Imagen: Cableado del Robonova

- Los componentes deben cumplir con los requisitos de seguridad de las normas IEC relevantes correspondientes, en tanto que sean razonablemente aplicables.

- Los juguetes no deben tener:

  -Disyuntores térmicos que puedan ser rearmados por soldadura

  -Interruptores de mercurio

- Las fijaciones en las que un fallo pueda comprometer el cumplimiento de esta norma y las conexiones eléctricas deben superar las solicitaciones mecánicas que surjan durante el juego.

- Los tornillos utilizados con estos fines no deben ser de metal blando o sujeto a deformaciones, como el zinc o el aluminio. Si son de material aislante deben tener un diámetro nominal de al menos 3mm y no deben ser utilizados para ninguna conexión eléctrica.

- Los tornillos usados para conexiones eléctricas deben roscarse en metal.

- Las conexiones eléctricas que conducen una corriente superior a 0,5 A deben estar construidos de forma que la presión de contacto no se transmita por medio de materiales aislantes que sean susceptibles de contraerse o distorsionarse a menos que una suficiente elasticidad de las partes metálicas compense cualquier posible contracción o distorsión del material aislante.

- Las distancias en el aire y líneas de fuga del aislamiento funcional no deben ser menores de 0,5mm.

- Las partes en material no metálico que encierran partes eléctricas y partes en material aislante que soportan partes eléctricas, deben ser resistentes a la ignición y a la propagación del fuego.

- Este requisito no se aplica a ornamentos decorativos, mandos y otras partes no susceptibles de incendiarse o propagar llamas originadas en el interior del juguete.

- Los juguetes no deben ser tóxicos ni presentar peligros similares.

A N E X O 

Aplicación PARTE 1

```
import wiiusej.*;
import wiiusej.values.Orientation;
import wiiusej.wiiusejevents.utils.*;
import wiiusej.wiiusejevents.physicalevents.*;
import wiiusej.wiiusejevents.wiiuseapievents.*;

import giovynet.nativelink.SerialPort;
import giovynet.serial.Baud;
import giovynet.serial.Com;
import giovynet.serial.Parameters;
import java.util.List;



// Clase MyListener

public class MyListener implements WiimoteListener {

	public static int vertical=0;
	public static int horizontal=0;
	public static boolean boton;

	SerialPort puerto = new SerialPort();
	public static Com com;
	List<String> listar;



// Metodo enviar datos

public void enviarDatos(char[] data){

	try{

		listar = puerto.getFreeSerialPort();
		Parameters parameters = new Parameters();
		parameters.setPort("COM5");
		parameters.setBaudRate(Baud._4800);
		Com com = new Com(parameters);
		com.sendArrayChar(data);
		com.close();

	}

	catch(Exception e){}
}
```

```
// Metodo onButtonsEvent:
// Detecta los botones pulsados del mando Wiimote

public void onButtonsEvent(WiimoteButtonsEvent arg0) {


      if (arg0.isButtonBHeld()){    // Detecta si boton B esta pulsado

            boton=true;}

      else{

            boton=false;
      }



      if (arg0.isButtonAPressed()){

            System.out.println("He pulsado el Boton A");
            char[] datos={7};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonOnePressed()){

            System.out.println("He pulsado el Boton 1");
            char[] datos={5};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }


      if (arg0.isButtonTwoPressed()){

            System.out.println("He pulsado el Boton 2");
            char[] datos={6};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }



      if (arg0.isButtonMinusPressed()){

            System.out.println("He pulsado el Boton -");
            char[] datos={};
            enviarDatos(datos);
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }
```

```
      if (arg0.isButtonPlusPressed()){

            System.out.println("He pulsado el Boton +");
            char[] datos={};
            enviarDatos(datos);
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonHomePressed()){

            System.out.println("He pulsado el Boton Home");
            char[] datos={};
            enviarDatos(datos);
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonUpPressed()){

            System.out.println("He pulsado el Boton Izquierda");
            char[] datos={4};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

      if (arg0.isButtonDownPressed()){

            System.out.println("He pulsado el Boton Derecha");
            char[] datos={3};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }


      if (arg0.isButtonLeftPressed()){

            System.out.println("He pulsado el Boton Detras");
            char[] datos={2};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }


      if (arg0.isButtonRightPressed()){

            System.out.println("He pulsado el Boton Delante");
            char[] datos={1};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(1000);}
            catch (InterruptedException e){ }
      }

}     // Fin Metodo onButtonsEvent
```

```
// Metodo onMotionSensingEvent:
// Detecta el Movimiento del mando Wiimote


public void onMotionSensingEvent(MotionSensingEvent arg0) {

      Orientation orientation =arg0.getOrientation();
      vertical=(int)orientation.getPitch();
      horizontal=(int)orientation.getRoll();

// Si inclino el mando y tengo pulsado el botón B:

      if((vertical<=-55 && boton==true)){

            System.out.println("Estoy girando a la derecha");
            char[] datos={3};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
      }


      if((vertical>60 && boton==true)){

            System.out.println("Estoy girando a la izquierda");
            char[] datos={4};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
      }


      if((horizontal<=-50 && boton==true)){

            System.out.println("Estoy inclinando el Mando hacia
atras");
            char[] datos={2};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
      }


      if(horizontal>50 && boton==true){

            System.out.println("Estoy inclinando el Mando hacia
delante");

            char[] datos={1};       // Asignación de un Nº a una orden
            enviarDatos(datos);     // Envío de Información por PUERTO
SERIE
            try { Thread.sleep(2000);}
            catch (InterruptedException e){ }
      }

}// Fin Metodo onMotionSensingEvent
```

```
public void onIrEvent(IREvent arg0) {}


public void onExpansionEvent(ExpansionEvent arg0) { }

public void onStatusEvent(StatusEvent arg0) { }

public void onDisconnectionEvent(DisconnectionEvent arg0) { }

public void
onClassicControllerInsertedEvent(ClassicControllerInsertedEvent arg0)
{ }

public void
onClassicControllerRemovedEvent(ClassicControllerRemovedEvent arg0){ }

public void onGuitarHeroInsertedEvent(GuitarHeroInsertedEvent arg0){ }

public void onGuitarHeroRemovedEvent(GuitarHeroRemovedEvent arg0) { }


public void onNunchukInsertedEvent(NunchukInsertedEvent arg0) {

      System.out.println("He conectado el mando Nunchuk al Wiimote");

}


public void onNunchukRemovedEvent(NunchukRemovedEvent arg0) {

      System.out.println("He desconectado el mando Nunchuk del
Wiimote");

}
```

```
// ****************** METODO MAIN ******************

public static void main(String[] args) throws Exception{

            // ****** WIIMOTE ******


Wiimote[] wiimotes = WiiUseApiManager.getWiimotes(1, true);
//Detecta los mandos que haya

if (wiimotes.length==0){

      System.out.println(" ****MANDO WIIMOTE NO DETECTADO**** ");
      WiiUseApiManager.definitiveShutdown();
      System.exit(1);
}

else {

      System.out.println(" *****MANDO WIIMOTE 1 DETECTADO**** ");

      Wiimote wiimote = wiimotes[0];
      wiimote.activateMotionSensing(); // Activar sensor de movimiento
      wiimote.activateIRTRacking(); // Activar sensor por infrarrojos
      wiimote.setLeds(true, false, false, false);    // Activar leds
      wiimote.addWiiMoteEventListeners(new MyListener());
}



}// Fin Metodo Main

} // FInal de la clase MyListener
```

A N E X O 

Aplicación PARTE 2

```
'====================================

DIM A AS BYTE
PTP SETON
PTP ALLON

'== motor diretion setting ======================
DIR G6A,1,0,0,1,0,0
DIR G6B,1,1,1,1,1,1
DIR G6C,0,0,0,0,0,0
DIR G6D,0,1,1,0,1,0

'== motor start position read ===================

GETMOTORSET G6A,1,1,1,1,1,0
GETMOTORSET G6B,1,1,1,0,0,0
GETMOTORSET G6C,1,1,1,0,0,0
GETMOTORSET G6D,1,1,1,1,1,0
        SPEED 5

'== motor power on  ============================

MOTOR G24
GOSUB POS_INICIAL
```

```
'=============================================

MAIN:

  ERX 4800, A, PROGRAMA  ' LEE EL PUERTO SERIE

PROGRAMA:

  IF A = 6 THEN

    GOSUB RUTINAN1                  '  CAMINA HACIA DELANTE
    DELAY 1000
    GOSUB POS_INICIAL
    A=100

  ELSEIF A= 24 THEN

    GOSUB RUTINAN2                  ' CAMINA HACIA ATRAS
    DELAY 1000
    GOSUB POS_INICIAL
    A=100

  ELSEIF A =30 THEN

    GOSUB RUTINAN3                  ' GIRA A LA DERECHA
    DELAY 1000
    GOSUB POS_INICIAL
    A=100

  ELSEIF A =96 THEN
    GOSUB RUTINAN4                   ' GIRA A LA IZQUIERDA
    DELAY 1000
    GOSUB POS_INICIAL
    A=100



  ELSEIF A =102 THEN '102

    GOSUB RUTINAN5                  ' VUELO
    DELAY 1000
    GOSUB POS_INICIAL
    A=100

  ELSEIF A =120 THEN
```

```
    GOSUB RUTINAN6                  ' PUNCH
    DELAY 1000
    SPEED 10
    GOSUB POS_INICIAL
    A=100

  ELSEIF A =126 THEN                ' REVERENCIA
    GOSUB RUTINAN7
    DELAY 1000
    GOSUB POS_INICIAL
    A=100

  ELSEIF A=100 THEN

   GOTO MAIN

  ENDIF

  GOTO MAIN

SALIR:
  END
```

```
'============================================================
'RUTINAS DEL ROBONOVA-1
'============================================================

RUTINAN1:       ' CAMINAR HACIA DELANTE

        SPEED 5
        MOVE24  85,  71, 152,  91, 112,  60, 100,  40,  80,   ,   ,   , 100,  40,  80,   ,   ,   , 112,
        76, 145,  93,  92,  60,

        SPEED 14
        MOVE24  90, 107, 105, 105, 114,  60,  90,  40,  80,   ,   ,   , 100,  40,  80,   ,   ,   , 114,
        76, 145,  93,  90,  60,
        MOVE24  90,  56, 143, 122, 114,  60,  80,  40,  80,   ,   ,   , 105,  40,  80,   ,   ,   , 113,
        80, 145,  90,  90,  60,
        MOVE24  90,  46, 163, 112, 114,  60,  80,  40,  80,   ,   ,   , 105,  40,  80,   ,   ,   , 112,
        80, 145,  90,  90,  60,

        SPEED 10
        MOVE24 100,  66, 141, 113, 100, 100,  90,  40,  80,   ,   ,   , 100,  40,  80,   ,   ,   , 100,
        83, 156,  80, 100, 100,
        MOVE24 113,  78, 142, 105,  90,  60, 100,  40,  80,   ,   ,   , 100,  40,  80,   ,   ,   ,  90,
        102, 136,  85, 114,  60,

        SPEED 14
        MOVE24 113,  76, 145,  93,  90,  60, 100,  40,  80,   ,   ,   ,  90,  40,  80,   ,   ,   ,  90,
        107, 105, 105, 114,  60,
        MOVE24 113,  80, 145,  90,  90,  60, 105,  40,  80,   ,   ,   ,  80,  40,  80,   ,   ,   ,  90,  56,
        143, 122, 114,  60,
        MOVE24 112,  80, 145,  90,  90,  60, 105,  40,  80,   ,   ,   ,  80,  40,  80,   ,   ,   ,  90,  46,
        163, 112, 114,  60,

        SPEED 10
        MOVE24 100,  83, 156,  80, 100, 100, 100,  40,  80,   ,   ,   ,  90,  40,  80,   ,   ,   , 100,
        66, 141, 113, 100, 100,
        MOVE24  90, 102, 136,  85, 114,  60, 100,  40,  80,   ,   ,   , 100,  40,  80,   ,   ,   , 113,
        78, 142, 105,  90,  60,

        SPEED 14
        MOVE24  90, 107, 105, 105, 114,  60,  90,  40,  80,   ,   ,   , 100,  40,  80,   ,   ,   , 113,
        76, 145,  93,  90,  60,
        SPEED 5
        MOVE24  85,  71, 152,  91, 112,  60, 100,  40,  80,   ,   ,   , 100,  40,  80,   ,   ,   , 112,
        76, 145,  93,  92,  60,
        RETURN
```

```
'================================================
'================================================
'================================================

RUTINAN2:       ' CAMINAR HACIA ATRAS

        SPEED 5
        GOSUB backward_walk1

        SPEED 13
        GOSUB backward_walk2

        SPEED 7
        GOSUB backward_walk3
        GOSUB backward_walk4
        GOSUB backward_walk5

        SPEED 13
        GOSUB backward_walk6

        SPEED 7
        GOSUB backward_walk7
        GOSUB backward_walk8
        GOSUB backward_walk9

        SPEED 13
        GOSUB backward_walk2

        SPEED 5
        GOSUB backward_walk1

        RETURN

backward_walk1:
        MOVE G6A, 85,  71, 152,  91, 112, 60
        MOVE G6D,112,  76, 145,  93,  92, 60
        MOVE G6B,100,  40,  80, , , ,
        MOVE G6C,100,  40,  80, , , ,
        WAIT
        RETURN

backward_walk2:
        MOVE G6A, 90, 107, 105, 105, 114, 60
        MOVE G6D,113,  78, 145,  93,  90, 60
        MOVE G6B, 90,  40,  80, , , ,
```

```
        MOVE G6C,100,  40,  80, , , ,
        WAIT
        RETURN

backward_walk9:
        MOVE G6A, 90,  56, 143, 122, 114,  60
        MOVE G6D,113,  80, 145,  90,  90,  60
        MOVE G6B, 80,  40,  80, , , ,
        MOVE G6C,105,  40,  80, , , ,
        WAIT
        RETURN

backward_walk8:
        MOVE G6A,100,  62, 146, 108, 100, 100
        MOVE G6D,100,  88, 140,  86, 100, 100
        MOVE G6B, 90,  40,  80, , , ,
        MOVE G6C,100,  40,  80, , , ,
        WAIT
        RETURN

backward_walk7:
        MOVE G6A,113,  76, 142, 105,  90, 60
        MOVE G6D,90,  96, 136,  85, 114, 60
        MOVE G6B,100,  40,  80, , , ,
        MOVE G6C,100,  40,  80, , , ,
        WAIT
        RETURN

backward_walk6:
        MOVE G6D,90, 107, 105, 105, 114, 60
        MOVE G6A,113,  78, 145,  93, 90, 60
        MOVE G6C,90,  40,  80, , , ,
        MOVE G6B,100,  40,  80, , , ,
        WAIT
        RETURN

backward_walk5:
        MOVE G6D,90,  56, 143, 122, 114,  60
        MOVE G6A,113,  80, 145,  90,  90,  60
        MOVE G6C,80,  40,  80, , , ,
        MOVE G6B,105,  40,  80, , , ,
        WAIT
        RETURN

backward_walk4:
```

```
        MOVE G6D,100,  62, 146, 108, 100, 100
        MOVE G6A,100,  88, 140,  86, 100, 100
        MOVE G6C,90,  40,  80, , ,,
        MOVE G6B,100,  40,  80, , , ,
        WAIT
        RETURN

backward_walk3:
        MOVE G6D,113,  76, 142, 105,  90, 60
        MOVE G6A, 90,  96, 136,  85, 114, 60
        MOVE G6C,100,  40,  80, , , ,
        MOVE G6B,100,  40,  80, , , ,
        WAIT
        RETURN

'================================================
'================================================
'================================================

RUTINAN3: ' GIRO A LA DERECHA

        SPEED 6

        MOVE G6A,  85,  71, 152,  91, 112,  60
        MOVE G6D, 112,  76, 145,  93,  92,  60
        MOVE G6B, 100,  40,  80,   ,   ,   ,
        MOVE G6C, 100,  40,  80,   ,   ,   ,
        WAIT

        SPEED 9

        MOVE G6D, 113,  75, 145,  97,  93,  60
        MOVE G6A,  90,  50, 157, 115, 112,  60
        MOVE G6C, 105,  40,  70,   ,   ,   ,
        MOVE G6B,  90,  40,  70,   ,   ,   ,
        WAIT

        MOVE G6D, 108,  78, 145,  98,  93,  60
        MOVE G6A,  95,  43, 169, 110, 110,  60
        MOVE G6C, 105,  40,  70,   ,   ,   ,
        MOVE G6B,  80,  40,  70,   ,   ,   ,
        WAIT

        RETURN
```

```
'================================================
'================================================
'================================================


RUTINAN4:       ' GIRO A LA IZQUIERDA

        SPEED 6

        MOVE G6D,  85,  71, 152,  91, 112,  60
        MOVE G6A, 112,  76, 145,  93,  92,  60
        MOVE G6C, 100,  40,  80,   ,   ,
        MOVE G6B, 100,  40,  80,   ,   ,
        WAIT

        SPEED 9

        MOVE G6A, 113,  75, 145,  97,  93,  60
        MOVE G6D,  90,  50, 157, 115, 112,  60
        MOVE G6B, 105,  40,  70,   ,   ,
        MOVE G6C,  90,  40,  70,   ,   ,
        WAIT

        MOVE G6A, 108,  78, 145,  98,  93,  60
        MOVE G6D,  95,  43, 169, 110, 110,  60
        MOVE G6B, 105,  40,  70,   ,   ,
        MOVE G6C,  80,  40,  70,   ,   ,
        WAIT

        RETURN
```

```
'=============================================
'=============================================
'=============================================

RUTINAN5:       ' VUELO

        DIM i AS BYTE
        SPEED 5

        MOVE G6A, 85,  71, 152,  91, 112, 60
        MOVE G6D,112,  76, 145,  93,  92, 60
        MOVE G6B,100,  40,  80, , , ,
        MOVE G6C,100,  40,  80, , , ,
        WAIT

        MOVE G6A, 90,  98, 105,  115, 115, 60
        MOVE G6D,116,  74, 145,  98,  93, 60
        MOVE G6B,100,  150,  150, 100, 100, 100
        MOVE G6C,100,  150,  150, 100, 100, 100
        WAIT

        MOVE G6A, 90, 121,  36, 105, 115,  60
        MOVE G6D,116,  60, 146, 138,  93,  60
        MOVE G6B,100,  150,  150, 100, 100, 100
        MOVE G6C,100,  150,  150, 100, 100, 100
        WAIT

        MOVE G6A, 90,  98, 105,  64, 115,  60
        MOVE G6D,116,  50, 160, 160,  93,  60
        MOVE G6B,145, 110, 110, 100, 100, 100
        MOVE G6C,145, 110, 110, 100, 100, 100
        WAIT

        FOR i = 10 TO 15
                SPEED i
                MOVE G6B,145,  80,  80, 100, 100, 100
                MOVE G6C,145,  80,  80, 100, 100, 100
                WAIT

                MOVE G6B,145,  120,  120, 100, 100, 100
                MOVE G6C,145,  120,  120, 100, 100, 100
                WAIT
        NEXT i

        DELAY 1000
```

```
SPEED 6

MOVE G6A, 90,  98, 105,  64, 115,  60
MOVE G6D,116,  50, 160, 160,  93,  60
MOVE G6B,100, 160, 180, 100, 100, 100
MOVE G6C,100, 160, 180, 100, 100, 100
WAIT


MOVE G6A, 90, 121,  36, 105, 115,  60
MOVE G6D,116,  60, 146, 138,  93,  60
MOVE G6B,100,  150,  150, 100, 100, 100
MOVE G6C,100,  150,  150, 100, 100, 100
WAIT
SPEED 4

MOVE G6A, 90,  98, 105,  115, 115, 60
MOVE G6D,116,  74, 145,  98,  93, 60
WAIT

MOVE G6A, 85,  71, 152,  91, 112, 60
MOVE G6D,112,  76, 145,  93,  92, 60
MOVE G6B,100,  40,  80, , , ,
MOVE G6C,100,  40,  80, , , ,
WAIT
RETURN
```

```
'================================================
'================================================
'================================================

RUTINAN6: ' PUNCH

        SPEED 15
        MOVE G6A, 92, 100, 110, 100, 107, 100
        MOVE G6D, 92, 100, 110, 100, 107, 100
        MOVE G6B,190, 150,  10, 100, 100, 100
        MOVE G6C,190, 150,  10, 100, 100, 100
        WAIT
        SPEED 15
        HIGHSPEED SETON

        MOVE G6B,190,  10, 75, 100, 100, 100
        MOVE G6C,190, 140,  10, 100, 100, 100
        WAIT
        DELAY 500
        MOVE G6B,190, 140,  10, 100, 100, 100
        MOVE G6C,190,  10, 75, 100, 100, 100
        WAIT
        DELAY 500

        MOVE G6A, 92, 100, 113, 100, 107, 100
        MOVE G6D, 92, 100, 113, 100, 107, 100
        MOVE G6B,190, 150,  10, 100, 100, 100
        MOVE G6C,190, 150,  10, 100, 100, 100
        WAIT

        HIGHSPEED SETOFF

        MOVE G6A,100, 115,  90, 110, 100, 100
        MOVE G6D,100, 115,  90, 110, 100, 100
        MOVE G6B,100,  80,  60, 100, 100, 100
        MOVE G6C,100,  80,  60, 100, 100, 100
        WAIT

        RETURN
```

```
'================================================
'================================================
'================================================

RUTINAN7:       ' REVERENCIA

        MOVE G6A, 100,  58, 135, 160, 100, 100
        MOVE G6D, 100,  58, 135, 160, 100, 100
        MOVE G6B, 100,  30,  80,  ,  ,  ,
        MOVE G6C, 100,  30,  80,  ,  ,  ,
        WAIT

        RETURN
'================================================
'================================================
'================================================

POS_INICIAL:    'POSICION INICIAL DEL ROBOT

  MOVE G6A,100,  76, 145,  93, 100, 100
  MOVE G6D,100,  76, 145,  93, 100, 100
  MOVE G6B,100,  30,  80, 100, 100, 100
  MOVE G6C,100,  30,  80, 100, 100, 100
  WAIT
  RETURN
'================================================
'================================================
'================================================
```