ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA DE TELECOMUNICACIÓN

UNIVERSIDAD POLITÉCNICA DE CARTAGENA

Proyecto Fin de Carrera

TELEOPERACIÓN DE ROBOT ANTROPMÓRFICO DOCENTE TEACHMOVER MEDIANTE WIIMOTE

AUTOR: Miguel Blázquez Viudas
DIRECTORES: Mª Francisca Rosique Contreras
Diego Alonso Cáceres

Junio 2012

Autor	Miguel Blázquez Viudas
E-mail del Autor	miguelblazquez16@hotmail.com
Director(es)	Mª Francisca Rosique Contreras, Diego Alonso Cáceres
Título del PFC	TELEOPERACIÓN DE ROBOT ANTROPMÓRFICO DOCENTE TEACHMOVER MEDIANTE WIIMOTE
Descriptores	Wiimote, Robot manipulador, Microbot TeachMover, Teleoperación, Interfaz humano-máquina

Resúmen

La videoconsola Wii de Nintendo, desde su lanzamiento en 2006 revoluciono la jugabilidad actual de las consolas. La característica más distintiva de dicha consola es su mando inalámbrico, el Wiimote, el cual utiliza una combinación de acelerómetros e infrarrojos de detección para detectar su posición tridimensional en el espacio. Este diseño permite a los usuarios controlar el juego mediante gestos físicos, así como presionar tradicionales botones. Gracias a su conectividad por Bluetooth, es posible utilizarlo también como dispositivo de entrada para aplicaciones de PC.

Este Proyecto Fin de Carrera (PFC) tiene por objeto el estudio del Wiimote como interfaz humano-maquina de bajo costes y el desarrollo de nuevas aplicaciones que aproveche el movimiento como forma de interacción. El caso de estudio seleccionado se centra en el desarrollo de una aplicación de control que permita manejar de forma remota el Microbot TeachMover mediante el dispositivo Wiimote.

Dicha aplicación hacen uso de los siguientes elementos:

- Wiimote
- Comunicación Bluetooth
- Microbot TeachMover
- Comunicación puerto USB

Dentro de la aplicación desarrollada conviene distinguir dos partes. Por una lado relacionada con la recepción de coordenadas de los movimientos del usuario utilizando el Wiimote como dispositivo de seguimiento y por otro lado la comunicación y envío de órdenes al robot.

Titulación	Ingeniero Técnico en Telecomunicaciones, esp. Telemática
Departamento	Tecnologías de la Información y las Comunicaciones
Fecha de Presentación	Junio - 2012

Índice de Contenidos

Índice de Anexos

Índice de Figuras

1. Introducción

2. Estado de la Técnica

3. Tecnología y Aplicaciones utilizadas

4. Caso de Estudio

5. Conclusión y trabajos futuros

Capitulo 1

Introducción

Este capítulo sirve como presentación del Proyecto Fin de Carrera (PFC) realizado, exponiendo en primer lugar la motivación que ha llevado el desarrollo de este proyecto, los objetivos perseguidos para su realización y por último se describe la estructura del documento de una manera general.

1.1 Motivación

Los robots desde sus inicios han estado presentes en trabajos relacionados con el sector industrial, principalmente en la industria automovilística, debido a su capacidad de realizar trabajos simples pero repetitivos con rapidez y uniformidad superior al operador humano. Inicialmente los robots ejecutaban las tareas que le eran programadas con un escaso margen de adaptación del entorno, pero a medida que se desarrollaron nuevos sensores, técnicas de control, aumento de la capacidad de cálculo, etc. Han aumentado su grado de adaptación, ayudando a expandir el campo de la aplicación de los robots más allá de las fábricas.

Debido a la expansión de la robótica surge la necesidad de buscar nuevas maneras de operar aprovechar las ventajas de los robots y del operador humano, surgiendo la teleoperación, se centra en realización de tareas de forma remota mediante un operador humano. La teleoperación permite la creación de nuevos dispositivos de control permitiendo que se simplifique la interacción entre el hombre y la maquina, de manera que el aprendizaje para los operadores sea lo más sencillas posibles.

Existen diversas circunstancias por las cuales no es conveniente el uso de personas para la realización de algunas laboreas debido al alto riesgo al que se exponen, por esta razón se han desarrollado diversas herramientas o aplicaciones que permiten al hombre realizar estas operaciones a distancia.

Esto es lo que se pretende con el Wiimote (interfaz de control de la consola Wii), crear una aplicación para controlar con el mismo un robot manipulado. La consola Wii de Nintendo desde su lanzamiento en 2006 ha tenido un éxito mundial siendo distribuida en todos los hogares de todo el mundo y por tanto su controlador Wiimote. Dicho mando permite el desarrollo de aplicaciones para PC que aprovechen las características que nos ofrece.

La gran ventaja que nos ofrece el controlador Wiimote frente a otros dispositivos, es su capacidad de detección de movimientos tridimensionales, permitiendo además la utilización de botones tradicionales. Gracias a su producción a gran escala dicho dispositivo se puede obtener a precios reducidos permitiendo su adquisición sin necesidad de obtener la consola Wii.

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 Microbot TeachMover. Gracias a la conectividad por Bluetooth que proporciona el Wiimote permite la capacidad del desarrollo de aplicaciones interactivas.

El Microbot TeachMover es un robot de uso educativo, diseñado para proporcionar una inusual combinación de destreza y bajo costo. Además contiene un software que utiliza toda la potencia de Java incluyendo todas las estructuras de control y métodos de programación permitiendo una mayor libertad.

1.2 Objetivos

Las innovaciones tecnológicas y sus aplicaciones representan un papel importante en la sociedad actual, permitiendo aplicar conocimientos para satisfacer necesidades de manera eficaz.

El Objetivo de este Proyecto Fin de Carrera (PFC) es crear una aplicación innovadora y actual aprovechando las distintas características que nos proporciona los dispositivos de bajo coste. Utilizando como dispositivo de bajo coste el mando de la videoconsola Wii de Nintendo, conocido comercialmente con el nombre de Wiimote. Dado que los desarrollos para Wiimote son escasos en número y muy recientes, se opta por la creación de una nueva aplicación para dicho dispositivo. Este proyecto se centrará mayoritariamente en el desarrollo de una aplicación de control necesaria para que el dispositivo Wiimote interactúe de manera remota con el robot "Microbot TeachMover".

Para ello se han fijado los siguientes objetivos:

- Estudio de los desarrollos actuales
- Estudio de interfaces Humano-Maquina
- Estudio de la API WiiUseJ, de desarrollo para Wiimote
- Estudio, manejo y control del Microbot TeachMover
- Identificación de aplicaciones a desarrollar
- Creación de una aplicación de teleoperación mediante Wiimote

Teniendo como objetivos el uso de aplicaciones de bajo coste, nos lleva a utilizar siempre aplicaciones gratuitas disponibles en internet. Por tanto, dicho proyecto hace uso de la aplicación de desarrollo Java, gracias a la utilización de Eclipse como entorno de desarrollo integrado de código abierto multiplataforma.

1.3 Estructura del documento

El presente documento se estructura en cinco capítulos. En el primer capítulo se da una breve introducción sobre el proyecto, explicando la motivación y los objetivos propuestos. En el segundo capítulo se analiza el estado de la Técnica, en relación con el trabajo que se pretende realizar (Teleoperación del Microbot TeachMover mediante el Wiimote). En el tercer capítulo se describe la tecnología y aplicaciones necesarias para su desarrollo. En el cuarto capítulo se explica de forma detallada todo el proceso necesario que hace posible el control del Microbot TeachMover mediante el Wiimote y por ultimo en el quinto capítulo se analizan las conclusiones de este trabajo y las líneas futuras que se deriva de ello.

Capitulo 2

Estado de la Técnica

Este capítulo pretende introducir al lector en el contexto en que se desarrolla este Proyecto Fin de Carrera. Concretamente, se realiza una presentación sobre el concepto de la teleoperación y los sistemas robóticos teleoperados, se describen los principales componentes que lo forman y, por último, las principales características de las interfaces humano-maquina así como sus tipos.

2.1 Introducción a la teleoperación

El hombre, desde hace mucho tiempo ha venido utilizando distintas herramientas para poder mejorar su capacidad de manipulación, desde la utilización de palos para hacer caer la fruta de un árbol hasta la utilización de brazos robóticos para la manipulación de sustancias peligrosas.

Estos desarrollos tecnológicos desembocaron finalmente en lo que se conoce como sistemas de teleoperación maestro-esclavo [Figura 2.1], en los que un manipulador denominado esclavo reproduce fielmente los movimientos de un dispositivo o manipulador maestro, controlado a su vez manualmente por un operador humano. Se puede decir que es entonces cuando la teleoperación cobra importancia como tecnología.

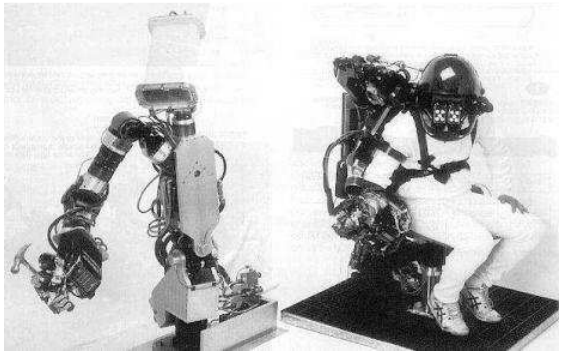

Figura 2.1: Sistema teleoperado

Desde que surgen estos sistemas de teleoperación hasta nuestros días podemos darnos cuenta que ha existido un gran avance: cada vez el área de acción de la teleoperación es mayor, las aplicaciones en la actualidad pueden ir desde la diversión y el entretenimiento hasta el rescate de personas en peligro.

En los sistemas de teleoperación de robots la intervención del operador humano 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 utilizan directamente el operador y el sistema de control local del robot.

Los robots teleoperados son definidos por la NASA [1] como “dispositivos robóticos con brazos manipulados y sensores con cierto grado de movilidad, controlados remotamente por un operador humano de manera directa o a través de un ordenador”. La intervención del operador puede producirse de muchas formas diferentes, desde la teleoperación directa activando un proceso, hasta la simple especificación de movimientos, o incluso de tareas, que se realizan de manera automática en el entorno remoto.

2.2 Componentes de un sistema teleoperado

Una manera sencilla de describir un sistema teleoperado es mediante un operador que envía comandos a un robot remoto, el cual ejecutará las órdenes correspondientes y enviará información de su estado y del entorno en el cual se encuentre.

Para facilitar la identificación de todos los componentes de un sistema teleoperado, se va a hacer una división del sistema de acuerdo a su localización. En la figura 2.2, se puede observar que un sistema teleoperado básicamente puede ser dividido en 3 zonas, Zona Local, Zona Remota y Canal de Comunicación.

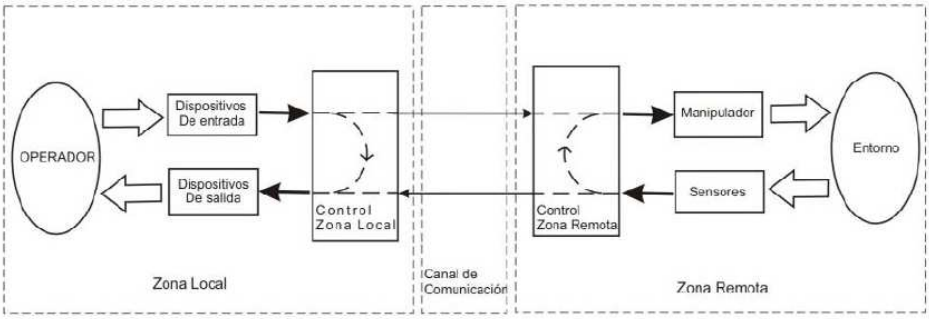

Figura 2.2: Elementos de un sistema teleoperado

2.2.1 Zona Local

En dicha zona se ubica el operador humano, que es el encargado de controlar la ejecución de la tarea remota. El mismo deberá contar con dispositivos de actuación para generar los comandos que serán enviados al robot en la zona remota. Para que el operador pueda tener conciencia del trabajo que se está realizando, se necesita dotar a la interfaz del sistema de dispositivos de realimentación con los que el operador puede tener información de la ejecución de la tarea.

- Operador: Es la persona encargada de realizar la tarea, el elemento que cierra el bucle de control. Su intervención pude ser diversa, dependiendo de la aplicación y puede ir desde su control absoluto, pasando por un control compartido hasta el control supervisado.

- Dispositivo de actuación: Son los dispositivos encargados de capturar los comandos generados por el operador, para ser trasmitidos al manipulador remoto. Existen una gran variedad de dispositivos capaces de realizar esta función entre los más comunes podemos nombrar: teclados, ratones, joysticks, etc.

- Dispositivos de realimentación: Son los dispositivos encargados de presentar al operador cualquier tipo de información relacionada con el desarrollo de la tarea que está ejecutando. Uno de los dispositivos de salida indispensables en cualquier

operación es el monitor, son capaces de presentar al operador información de tipo texto, gráfico o imágenes.

- Control Zona Loca: Cuando se quiere mejorar la calidad de la operación, presentado ayudas al operador como displays predictivos o simulaciones, mejorando el desempeño en la ejecución de la tarea.

2.2.2 Zona Remota

Es el entorno en el que se encuentra ubicado el robot esclavo, y el lugar donde se encuentra localizado el entorno remoto de trabajo.

- Control Zona Remota: Es el encargado de que el manipulador esclavo ejecute los comandos enviados por la Zona Local.

- Robot o manipulador: Es el dispositivo a través del cual se lleva a cabo la tarea.

- Sensores: Los sensores son los encargados de adquirir el estado tanto de la Zona Local como de la Zona Remota.

- Entorno: Es el lugar o aérea circundante al esclavo en el cual se lleva a cabo la tarea, llamada también entorno de trabajo.

2.2.3 Canal de Comunicaciones

Es el medio a través del cual se trasmiten los comandos enviados desde la Zona Local a la Zona Remota y la información de realimentación desde la Zona Remota a la Zona Local.

2.3 Robots teleoperados

Los robots teleoprerados pueden clasificarse de varias maneras, una de las formas más conocidas es la determinada por su arquitectura [2]. Cabe decir que pese a que la clasificación anterior es la más conocida, existen otros no menos importantes donde se tiene más en cuenta la potencia del software en el controlador.

- Poliarticulados: Bajo este grupo los robots son muy diversos respecto a forma y configuración, cuya característica común es la de ser básicamente sedentarios y estar estructurados para mover sus elementos terminales en un determinado espacio de trabajo según uno o más sistemas de coordenadas y con un número limitado de grados de libertad. En este grupo se encuentran los manipuladores y algunos robots industriales, y se emplean cuando es preciso abarcar una zona de trabajo relativamente amplia o alargada.

- Móviles: Cuentan con gran capacidad de desplazamiento, basados en carros o plataformas y dotados de un sistema locomotor de tipo rodante. Siguen su camino por telemando o guiándose por la información recibida de su entorno a través de sus sensores. Este tipo de robots se utilizan para la investigación en lugares de difícil acceso o distantes.

- Androides: Estos intentan reproducir total o parcialmente la forma y el comportamiento cinemático del ser humano. Actualmente los androides son todavía dispositivos muy poco evolucionados y sin utilidad práctica, y destinados, fundamentalmente, al estudio y experimentación.

- Zoomórficos: Los robots zoomórficos, que considerados en sentido no restrictivo podrían incluir también a los androides, constituyen una clase caracterizada principalmente por sus sistemas de locomoción que imitan a los diversos seres vivos.

- Híbridos: Estos robots corresponden a aquellos de difícil clasificación cuya estructura resultante, se sitúa en una combinación con algunas de las expuestas anteriormente.

El Microbot TeachMover es el robot utilizado en el proyecto, dentro de la clasificación realizada podemos incluirlo en los robots poliarticulados, ya que permite la realización de determinados movimientos respecto a un espacio de trabajo determinado según una serie de coordenadas y un número limitado de grados de libertad.

2.4 Sistemas de teleoperación reales

Un sistema teleoperado [Figura 2.3] se compone principalmente de una estación de teleoperación, un sistema de comunicación y un esclavo, el esclavo puede ser un manipulador o un robot móvil equipado con un manipulador ubicado en un entorno remoto.

A continuación se describe algunos de los sistemas de teleoperación mas sofisticados:

Robot Andros Wolverine [3]: Fabricado por la empresa Remotec, especializado en operaciones de uso militar o policial. El cual puede ser pre-programado para tomar decisiones sin necesidad de consultar al operador. Se utiliza para operaciones de limpieza de sustancias químicas, tareas de seguridad, asistencia sanitaria, etc.

Figura 2.3: Sistema teleoperado

Robot Chrysor [Figura 2.4]: Es un vehículo estable y robusto ideal para tareas de reconocimiento no tripulados y misiones de vigilancia, incluso en terrenos difíciles. Por otra parte Chrysor es capaz de atravesar caminos pre-especificados de forma autónoma, alternativamente puede ser controlado por radio desde un puesto de mando [4].

Figura 2.4: Robot Chrysor

2.5 Interfaz Humano-Maquina (HMI)

La interfaz de un sistema teleoperado es el medio con el que el usuario puede realizar una comunicación con una máquina, un equipo o un computador. Dichas interfaces incluyen elementos como menús, ventanas, teclado, ratón y en general todos aquellos canales por los cuales se permite la comunicación entre el ser humano y la computadora. Debe ser fácil de manejar, robusta, completa y sobre todo facilitar al operador la realización de tareas remotas, cobran un papel importante en la teleoperación ya que es el medio de contacto entre el usuario y la maquina. Se identifican 3 puntos que debe cumplir una interfaz:

1. Establecer todas las conexiones necesarias entre el operador y la zona remota de trabajo.

2. Facilitar la ejecución de tareas, permitiendo al operador enviar comandos referentes al trabajo a realizar.

3. Suministrar al operador toda la información necesaria del entorno de trabajo, con el fin de que alcance el mayor grado posible de trasparencia.

2.5.1 Tipos de interfaces de usuario

Según la forma de interactuar del usuario: Nos encontramos con varios tipos de interfaces de usuario.

- Interfaces alfanuméricas (intérpretes de comandos) que solo presentan texto.

- Interfaces gráficas de usuario, las que permiten comunicarse con el ordenador de una forma rápida e intuitiva representando gráficamente los elementos de control y medida.

- Interfaces táctiles, representan gráficamente un "panel de control" en una pantalla sensible que permite interactuar con el dedo de forma similar a si se accionara un control físico.

Según su construcción: Pueden ser hardware o software.

- Interfaces hardware: Se trata de un conjunto de controles o dispositivos que permiten al usuario el intercambio de datos con la máquina, ya se introduciéndolos (pulsadores, botones, teclas, …) o leyéndolos (pantallas, medidores, marcadores, …).

- Interfaces software: Son programas que permiten expresar los deseos del usuario a un ordenador o visualizar una respuesta.

Dentro de las interfaces de usuario según su construcción podemos decir que el Wiimote se encuentra identificado como "Interfaz hardware", ya que es un dispositivo que mediante la pulsación de botones y movimientos trasmite datos a una maquina.

En definitiva existen infinidad de interfaces que permiten la interactuación entre humanos y máquinas, aunque el camino no ha terminado y la investigación continúa en la mejora de nuevos dispositivos de interacción que poco a poco dejen para el museo los actualmente más extendidos interfaces. Entre las más prometedoras interfaces podemos destacar:

- Herramientas de reconocimiento de voz.

- Interfaces Cerebro Computador (BrainComputer Interface, BCI).

- Sensores fisiológicos o biométricos: reconocimiento de características faciales, huellas dactilares.

- Sensores de movimiento: acelerómetros y giróscopos.

Un *acelerómetro* [Figura 2.5] es un dispositivo para medir la aceleración y su propio movimiento. Los acelerómetros pueden detectar y medir vibración, inclinación y velocidad, además, podemos encontrarlos en podómetros (para medir los pasos de

una persona), sismógrafos, lavadoras, cámaras fotográficas (para evitar las fotos "movidas") y hasta en controles de videojuegos.

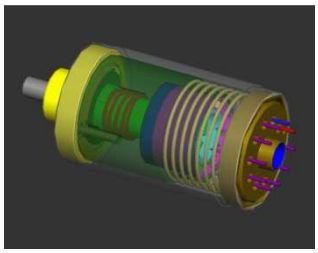

Figura 2.5: Acelerómetro

Un *giróscopo* [Figura 2.6] es un dispositivo mecánico formado esencialmente por un cuerpo con simetría de rotación que gira alrededor de su eje de simetría. Cuando se somete el giróscopo a un momento de fuerza que tiende a cambiar la orientación del eje de rotación su comportamiento es aparentemente paradójico ya que el eje de rotación, en lugar de cambiar de dirección como lo haría un cuerpo que no girase, cambia de orientación en una dirección perpendicular a la dirección intuitiva.

Figura 2.6: Giróscopo

Capitulo 3

Tecnología y Aplicaciones utilizadas

En este capítulo se explica con detalle las tecnologías utilizadas, como pude ser el Wiimote y Microbot TeachMover, además de explicar las aplicaciones que han sido necesarias para el desarrollo de este proyecto.

3.1 Tecnología utilizada

3.1.1 Controlador Wiimote

Introducción

Wiimote [5] es el mando principal de la consola Wii de Nintendo, sus características van más allá de la simple pulsación de botones para el control de videojuegos, permitiendo la capacidad de detección de movimiento en el espacio y la habilidad de apuntar a objetos en pantalla. El mando Wiimote [Figura 3.1] tiene un papel fundamental en el desarrollo de este proyecto, ya que dispone de distintos sensores para la detección de movimientos.

Figura 3.1: Wiimote

Diseño del Wiimote

El diseño del Wiimote no está basado en los tradicionales mandos para los videojuegos, su forma es similar a la de un mando de televisión o reproductor de DVD midiendo exactamente 14,6x3,5x3,1 cm. Este tamaño permite adaptarse de forma ergonómica a la mano, alejándolo de las formas rectangulares de los mandos tradicionales. Esta forma permite que el dedo pulgar descanse sobre el botón “A” sin perder el fácil acceso a la cruceta y dejado de forma intuitiva que el dedo índice apoye sobre el botón “B” a modo de gatillo.

Dicho diseño permite que el control del mando puede ser tomado de dos formas: vertical (con una mano) y horizontal (con dos manos). Esta última está diseñada para usarse en juegos de conducción y vuelo, además de ser usado para algunos juegos de la Consola Virtual. El diseño del Wiimote es simétrico, por lo que puede tomarse tanto con la mano derecha como con la izquierda.

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. En la parte inferior se encuentra el puerto de expansión del mando y viene unida una correa de seguridad que se amarra a la muñeca para evitar soltar y dañar el controlador de forma accidental.

Se encuentra disponible en seis colores: blanco, negro, azul, rosa, rojo y dorado.

Funcionalidad del Wiimote

El Wiimote tiene la capacidad de detectar la aceleración a lo largo de tres ejes mediante la utilización de un acelerómetro ADXL330. Cuenta también 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, 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. La barra lleva un sobrado cable de 3 m de largo que se conecta a un puerto propietario de la consola, encargado de alimentar y poner en funcionamiento las luces de los extremos. La barra puede ser colocada por encima o por debajo de la televisión, y debe centrarse.

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, este sistema de posicionamiento de puntos recibe el nombre de “Sistema de posicionamiento multiobjeto”, o también llamado MOTS (Multi-Object Tracking System). Se trata de un sistema de procesamiento de imagen asistido por ordenador, integrado en el hardware, lo que permite localizar la posición de los puntos brillantes recibidos por la cámara.

El chip de procesado de la posición de los puntos puede procesar hasta 4 puntos diferentes simultáneamente, y enviar la posición de todos ellos. Por tanto, podríamos posicionar 4 punteros infrarrojos diferentes si se deseara. Si hubiera más de 4 punteros en la superficie, escogería los 4 puntos más brillantes por defecto.

El mando dispone de un tiempo de muestreo de 100 ms. Este será el tiempo que se tarda en refrescar la información de los puntos.

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 Wiimote 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.

Esta capacidad de detección de movimientos [Figura 3.2] es gracias al acelerómetro ADXL330 integrado que contiene el Wiimote, capaz de medir tanto la aceleración como la dirección del movimiento comprobando los cambios en los electrones de su interior. Para entenderlo mejor, imaginemos una pila de dos minúsculas placas dentro del sensor. Una se mantiene fija, pero la otra se mueve. Los electrones que rodean las placas se mueven con este movimiento, y midiendo la capacitancia, el sensor es capaz de capturar los datos requeridos. El dispositivo

puede así enviar datos de movimientos relativos en todos los ejes de coordenadas, aparte de los datos de aceleración, detectando (dentro de sus limitaciones) giros, movimientos en el aire, inclinaciones... Estos datos se envían al chip Broadcom, encargado de la emisión de datos a la consola mediante Bluetooth.

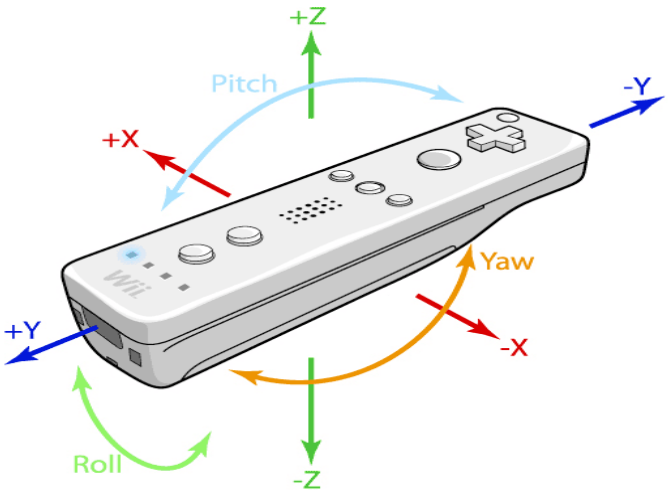

Figura 3.2: Capacidad de movimientos

Componentes Wiimote

El mando está compuesto de diversos componentes, a continuación comentaremos los más importantes [6]:

Acelerómetro

El mando dispone de un acelerómetro integrado que proporciona los datos de movimiento, giro e inclinación. Este tipo de dispositivos se conocen como MEMS (sistema micro electro-mecánico).

El chip MEMS utilizado en el Wiimote proviene de Analog Devices y es exactamente el modelo ADXL330 [Figura 3.3], con un tamaño de 4x4x1,45 mm. Este chip está diseñado específicamente para tener un consumo muy bajo, alargando la vida de las pilas. Su rango de detección es de +-3,6 gravedades y tiene una resolución de 300 mV/g.

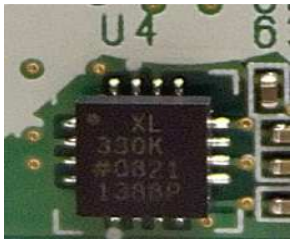

Figura 3.3: Chip MEMS

Emisor / Receptor Bluetooth

Para comunicarse con la consola, el Wiimote dispone de un chip [Figura 3.4] emisor y receptor de datos por Bluetooth, proporcionado por la empresa especializada en comunicaciones Broadcom Technologies.

Este chip, modelo BCM2042 ofrece una velocidad de transferencia de 2.1 Mbits/s. Broadcom ha incluido mejoras especiales que permiten una latencia muy baja entre consola y mando, intentando que la respuesta sea prácticamente la de un mando cableado clásico, además dicho componente tiene un consumo relativamente bajo.

Figura 3.4: Chip Bluetooth

Gracias a este protocolo se ha establecido una comunicación entre el Wiimote y el PC.

Sensor infrarrojo

El Wiimote incluye en su parte superior una cámara monocroma [Figura 3.5] con una resolución de 128x96, con un procesado de imagen hardware ya incorporado. En la parte delantera del mando encontramos un plástico de color negro, se trata de un filtro que solo deja pasar rayos infrarrojos, con lo que esta cámara únicamente captará puntos en los que haya un emisor infrarrojo. Los datos de posición de estos puntos se envían por Bluetooth al dispositivo que esté conectado el mando.

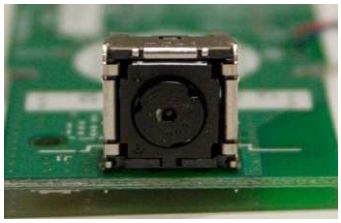

Figura 3.5: Sensor infrarrojo

El procesador MOST de la marca PixArt incorporado utiliza un análisis de subpixeles, los cuales multiplican automáticamente por 8, llegando a obtener una resolución de 1024x768 para los puntos posicionados.

Memoria Wiimote

El Wiimote contiene un chip de 16 KB EEPROM [Figura 3.6] 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. Esta función es también utilizada en juegos permitiendo al usuario guardar los datos de configuración del controlador para Wiimote.

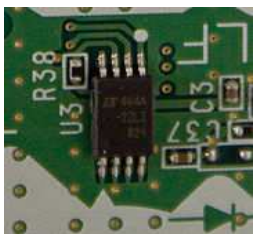

Figura 3.6: Memoria Wiimote

Alimentación Wiimote

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. Un condensador de 3300 µF proporciona una fuente temporal de energía para movimientos rápidos del Wiimote, cuando la conexión a las pilas pudiera ser interrumpida temporalmente.

Configuración del Wiimote

El funcionamiento del controlador Wiimote es el siguiente:

- En el frente del wiimote encontramos el "puntero", que nos permite apuntar con precisión a la pantalla.

- En la esquina superior izquierda del mando encontramos el botón POWER, que apaga y enciende la consola a distancia.

- La cruceta de control se situará en la zona superior del Wiimote. Permite la ejecución de movimientos en determinados juegos.

- El primer botón de acción que encontramos es A, el que más usaremos, junto a B, que está en la parte trasera del Wiimote.

- Después vemos tres pequeños botones alineados horizontalmente: +, HOME y -. + y - suelen hacer la función de SELECT y START, o aumentar y reducir el tamaño de ciertas cosas. HOME te lleva a un menú que te permite volver rápidamente al menú de Wii.

- Debajo encontramos un pequeño altavoz para oír efectos de sonido.

- En la parte inferior vemos los botones 1 y 2 alineados verticalmente. Son los botones de acción secundarios y permiten realizar acciones de poca importancia.

- Debajo vemos cuatro LEDs. El encendido nos indicará el jugador que somos (1, 2, 3 o 4). Debajo el logo Wii.

- En la parte trasera vemos una entrada para extensiones del Wiimote, como el Nunchuk o el Classic controller. También vemos la cinta para ajustar a la muñeca.

- Dentro de la tapa, en la parte trasera, además del espacio para pilas, encontramos el botón de sincronización.

Configuración del mando Wiimote acorde a las necesidades del proyecto:

- Botón POWER: No tiene asignado ninguna funcionalidad.
- Cruz de direcciones: Controla el movimiento de la pinza, permitiendo ejecutar multiplex movimientos (arriba, abajo, derecha e izquierda).
- Botón A: No tiene asignado ninguna funcionalidad.
- Botones + y -: Permite la apertura y cierre de la pinza.
- Botón HOME: Permite al usuario volver a la posición de inicio del robot.
- Botones 1 y 2: También llamados botones de seguridad, ya que obliga al usuario la necesidad de tener dichos botones presionados para la ejecución de movimientos.
- Leds: Indican al usuario la conectividad correcta con la maquina.
- Botón B o gatillo: Este botón obliga al usuario tenerlo pulsado cuando quiera hacer uso de la cruz de direcciones. También es considerado como botón de seguridad.

Accesorios del Wiimote

El Wiimote tiene un puerto de expansión para añadir periféricos en su parte inferior. Los distintos periféricos disponibles son: Nunchuk [Figura 3.7], Wii Classic Controller, Wii Classic Controller Pro, Wii Zapper, Wii Wheel, Wii Guitar [Figura 3.8], Wii Balance Board [Figura 3.9], Wii MotionPlus, Wii Vitality Sensor, Wii Remote Plus y UDraw Game Tablet.

A continuación se muestra algunos de los periféricos:

Figura 3.7

Figura 3.8

Figura 3.9

3.1.2 Microbot TeachMover

Introducción

El TeachMover [Figura 3.10] es un brazo resistente utilizado para la enseñanza de los fundamentos de la robótica, especialmente diseñado para simular operaciones de robots industriales en aulas o laboratorios. Dicho robot es el utilizado en el proyecto, el cual recibe y ejecuta las órdenes enviadas por la computadora.

El brazo TeachMover contiene un microprocesador que controla seis brazo articulado mecánico diseñado para proporcionar una inusual combinación de destreza y bajo costo. El TeachMover se puede utilizar en cualquiera de los dos modos:

- Modo interfaz serie: El brazo TeachMover puede ser controlado por un ordenador central o un terminal a través de uno de sus puertos serial. Interfaz utilizada en el proyecto.

- Modo TeachControl [Figura 3.11]: Permite el control del brazo TeachMover mediante la mano humana, se utiliza para enseñar, editar y ejecutar una variedad de programas de manipulación.

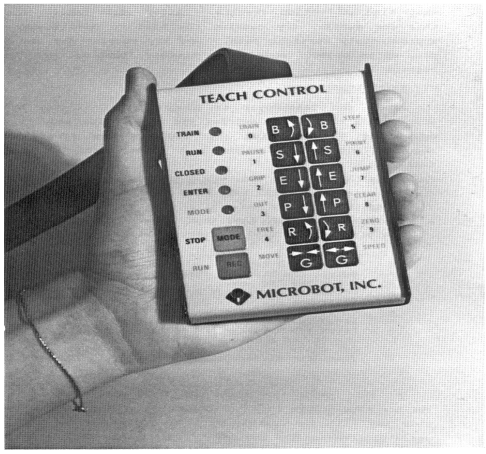

Figura 3.11: TeachControl

El robot se puede utilizar para una variedad de propósitos:

- Educativos
- Disfrute
- Automatización industrial
- Experimentación

Descripción técnica

En la siguiente figura se puede apreciar un diagrama del ARMDROID con todas sus partes detalladas:

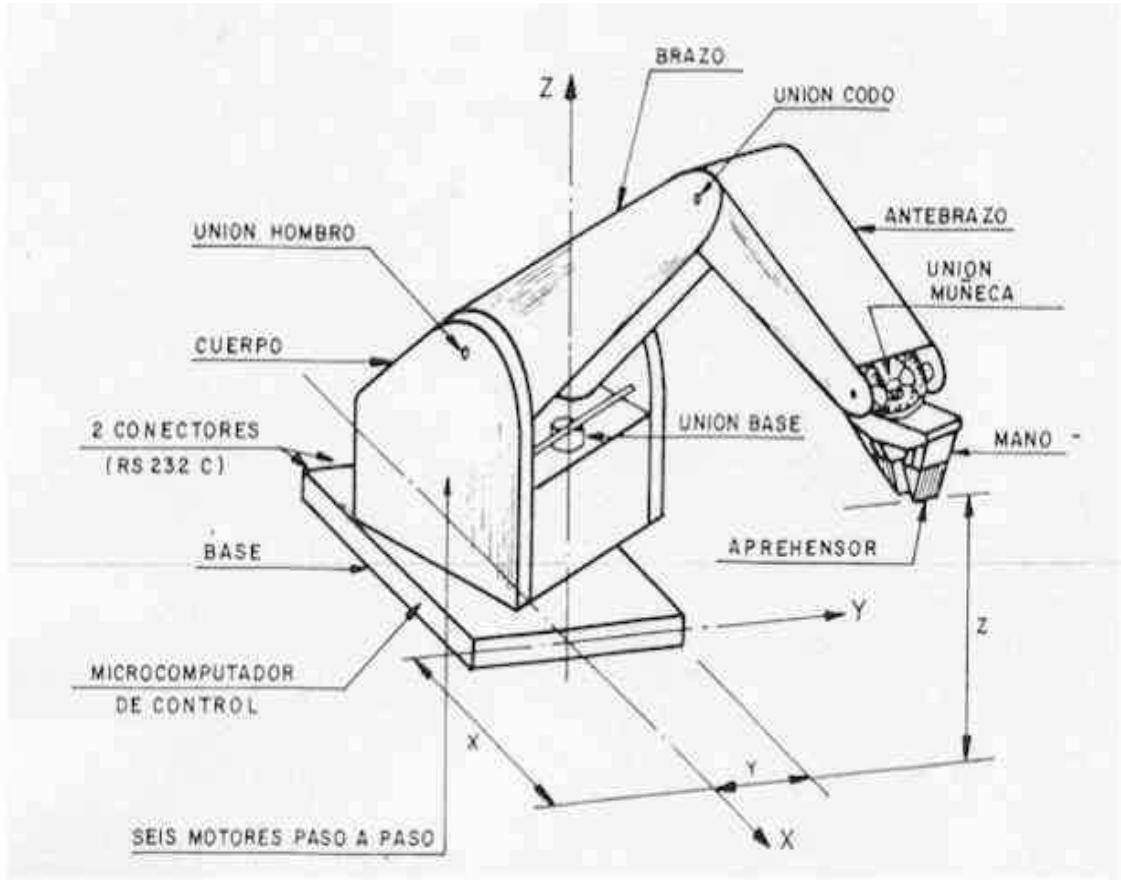

Figura 3.10: Microbot TeachMover

Como se puede apreciar este modelo es un brazo muy completo, posee cuatro ejes de movimiento: Base, Hombro, Codo y Muñeca.

El ángulo de giro de la base posee un movimiento de derecha a izquierda y viceversa con un ángulo de giro generalmente limitado por los cables que conectan el cuerpo del brazo con la base de apoyo. Alcanzando ángulos de giro muy cercanos a los 360°.

Este modelo utiliza 6 motores paso a paso. Uno es utilizado para el movimiento lateral de la base, un segundo y tercer motor para dar movimiento al brazo y antebrazo, un cuarto y quinto motor para accionar la muñeca hacia arriba y abajo y darle giro, y el sexto y último motor para controlar la apertura y cierre del aprehensor (pinza) de la mano.

La mayoría de los brazos robóticos tienen al menos algunos de los motores de accionamiento montado en los miembros de extensión (brazo, antebrazo o mano). Por desgracia esto se suma al peso de los miembros, provocando que la unidad de los miembros de extensión tenga la necesidad de ser más grande y más caro de lo que sería necesario.

Para evitar este problema los seis motores del Microbot TeachMover están montados en la base del robot. Este minimiza el peso de los miembros de extensión y mantiene los requerimientos del motor de carga de trabajo lo más bajo posible. Para reducir el número de piezas móviles, los seis engranajes de accionamiento están montados en el mismo eje.

Los motores utilizados tienen 4 bobinas, cada uno accionado por un transistor de potencia. La unidad es digital, con los transistores ya sea encendido o apagado para obtener el deseado patrón de corrientes, un campo magnético giratorio se obtiene en el interior del motor, haciendo que el motor gire en pequeños incrementos o pasos.

La conexión mecánica entre los motores y los ejes de cada extremidad se realiza por medio de delgados cables de acero, engranajes y poleas.

Un sistema único se emplea para manipular los elementos de brazo. Este diseño de cable es una adaptación y el refinamiento del tendón "Technolgy "utilizado en las aeronaves, impresoras de alta velocidad, y otros tipos de equipos. Cada cable es enrolla alrededor del eje del engranaje de accionamiento. Esto sirve no sólo para proporcionar un recipiente de recogida para el cable, pero también da a los propios engranajes de reducción de tasas para cada una de las seis unidades de disco.

El brazo TeachMover tiene una capacidad de elevación de una libra cuando está completamente extendida, y una resolución (movimiento de robot) de 0.011 pulgadas. El final de la mano puede colocarse en cualquier lugar dentro de una esfera parcial de un radio de 17,5 centímetros. La velocidad máxima es de 2 a 7 pulgadas por segundo, dependiendo de la carga de rendimiento (peso del objeto que se maneja).

En general, la base, el cuerpo, y todos los miembros de extensión son huecas de chapa, pero fuerte. Todos los miembros están conectados entre sí por medio de ejes, que se pasa a través casquillos montados sobre los miembros.

Memoria

El Microbot TeachMover contiene una tarjeta interna que alberga todos los componentes electrónicos, incluidos los del microprocesador 6502A. En términos técnicos, es un microprocesador de 8 bit y 2MHz, que se usa para coordinar todos los movimientos del robot y manejar todas la entradas y salidas de datos.

TeachMover firmware (incorporada en el software) está contenido en otro chip de 4K bytes de memoria de sólo lectura (ROM), este firmware interpreta los comandos que se le da al brazo.

La tarjeta también incluye un chip que contienen de 1 KB de memoria de acceso aleatorio (RAM), esto es suficiente memoria para que pueda almacenar un programa de movimiento del brazo de hasta 53 pasos. Es posible un segundo conjunto de RAMs, con lo que amplía su capacidad del programa para 126 pasos.

Puertos seriales

En la parte trasera de la base, se encuentra los puertos de interfaz serie que conectan el TeachMover a un ordenador. Dichas interfaces permiten configurar la velocidad de transmisión de datos, velocidades estándar que están disponibles de 110 a 9600 baudios. En el proyecto la velocidad de trasmisión utilizada es 9600 baudios (velocidad por defecto).

Entradas y salidas de usuario

La tarjeta del Microbot TeachMover, también contiene un equipo auxiliar en paralelo de entrada/salida. Esto le permite interconectar el TeachMover a un equipo externo con un cable de cinta plana de 16 conductores. Cinco salida TTL compatible con el usuario, los bits se pueden configurar (a 1) o cleard (a 0) bajo control de un programa para convertir otro equipo o desactivar un determinado movimiento del brazo. Siete TTL compatibles de entrada de usuario, se puede utilizar para controlar una secuencia de armado cuando una condición dada externa se cumple.

3.1.3 Otras tecnologías utilizadas

En este apartado explicaremos de forma breve otros dispositivos utilizados en el proyecto de carácter menos relevante.

Ordenador: Maquina encargada de procesar la información obtenida del Wiimote y enviarla al Microbot TeachMover.

Adaptador USB-RS232: Ya que los portátiles actuales carecen de puerto serie, fue necesario el uso de un adaptador. Dicho adaptador convierte una señal procedente de un puerto USB a una de tipo serie, este dispositivo viene con un CD de instalación con sus drivers, que tuvieron que ser instalados en el portátil.

Cable serie DB25 macho-DB9 hembra: Cable necesario para poder realizar la conectividad entre el ordenador y el Microbot TeachMover.

Dispositivo Bluetooth: Wiimote se comunica con el PC mediante una conexión Bluetooth. Dicho dispositivo lleva integrado un chip controlador Bluetooth Broadcom 2042, diseñado para aparatos que utilicen el estándar HID (Human Interface Device), como teclados o ratones. Cuando se encuentra en modo discovery (Service Discovery Protocol, SDP) el Wiimote envía información al receptor al ser descubierto. Lo más relevante es:

- Name: Nintendo RVL-CNT-01
- Vendor ID: 0x057e
- Product ID: 0x0306

Esta información se envía al receptor con una frecuencia máxima de 100 informes por segundo. La cantidad de veces que podemos leer los acelerómetros, será algo menor ya que se envían datos de control, de los botones, infrarrojos, etc.

3.2 Aplicaciones utilizadas

3.2.1 Entorno de desarrollo Eclipse

Para la codificación del programa realizado es este proyecto se ha utilizado Eclipse como entorno de desarrollo integrado de código abierto multiplataforma. Eclipse mayoritariamente se utiliza para desarrollar lo que se conoce como "Aplicaciones de Cliente Enriquecido", opuesto a las aplicaciones "Cliente-liviano" basadas en navegadores. Es una potente y completa plataforma de programación, desarrollo y compilación de elementos tan variados como sitios web, programas en C++ o aplicaciones Java (utilizado en el proyecto). En Eclipse se encuentra todas las herramientas y funciones necesarias para el proyecto, recogidas además en una atractiva interfaz que lo hace fácil y agradable de usar.

Introducción

El *Eclipse Project* [7] fue creado originalmente por *IBM* en el año *2001*. Estaba financiado por un consorcio de vendedores de *software*. La *Eclipse Foundation* se creó en *2004* como una organización independiente sin ánimo de lucro para actuar como administrador de la comunidad *Eclipse*. En un principio, permitía a los vendedores el acceso a una comunidad en torno a *Eclipse* neutral, transparente y abierta para establecerse. Hoy día, la comunidad *Eclipse* está formada por organizaciones individuales y organizaciones compuestas por agrupaciones de vendedores/fabricantes de *software*.

La fundación *Eclipse* se financia con las aportaciones anuales de sus miembros y se encuentra gobernada por un conjunto de directivos representantes de las *fuerzas vivas* actuales del software y que conforman la denominada *Board of Direction*. Desarrolladores y consumidores importantes tienen cabida en ella, así como representantes de los comités *Open Source*.

Eclipse utiliza la licencia *EPL* (*Eclipse Public License*). La licencia *EPL* permite a las empresas incluir *Eclipse* en sus productos comerciales, aunque también exige contribuir con la comunidad a aquellos que creen trabajos derivados del código licenciado bajo licencia *EPL*. La naturaleza *commercial-friendly* de la licencia *EPL* puede verse en multitud de casos en que diversas compañías han desarrollado y comercializado productos basados en *Eclipse.*

Arquitectura

La base para *Eclipse* es la *Plataforma de cliente enriquecido* (del inglés *Rich Client Platform* o *RCP*). Los siguientes componentes constituyen la plataforma de cliente enriquecido:

- Plataforma principal. Inicio de *Eclipse*, ejecución de *plugins*.

- *OSGi*. Una plataforma para *bundling* estándar.

- El *Standard Widget Toolkit* (*SWT*). Un *widget toolkit* portable.

- *JFace*. Manejo de archivos, manejo de texto, editores de texto.

- El *Workbench* de *Eclipse*. Vistas, editores, perspectivas, asistentes.

Los *widgets* de *Eclipse* están implementados como se acaba de mencionar por una herramienta de *widget* para *Java* llamada *SWT*; a diferencia de la mayoría de las aplicaciones *Java*, que usan las opciones estándar *Abstract Window Toolkit* (*AWT*) o *Swing*. La interfaz de usuario de *Eclipse* también tiene una capa *GUI* intermedia llamada *JFace*, la cual simplifica la construcción de aplicaciones basada en *SWT*.

Eclipse utiliza *plug-ins* para proporcionar toda la funcionalidad, a diferencia de otras aplicaciones donde la funcionalidad suele estar *hard coded*. El sistema *runtime* de eclipse está basado en *Equinox*, un estándar *OSGi*.

El mecanismo para implementar esta funcionalidad basada en *plug-ins* es un *framework* ligero de composición de software (basado en *Component-based software engineering* o *CBSE*, también conocido como *component-based development* o *CBD*). Gracias a este mecanismo, *Eclipse* ha sido extendido para usar otros lenguajes de programación como *C* y *Python*. El *plug-in framework* permite a *Eclipse* funcionar con lenguajes como *LaTeX*, aplicaciones de red como *Telnet* o sistemas de bases de datos. Por tanto, la arquitectura de *plug-in* posibilita escribir cualquier extensión deseada para al entorno.

Con la excepción de un pequeño *kernel run-time*, todo en *Eclipse* es un *plug-in*. Esto significa que cada *plug-in* desarrollado se integra con *Eclipse* y con los demás *plug-ins* de manera similar. *Eclipse* proporciona *plug-ins* para una amplia variedad de aplicaciones, como por ejemplo un *plug-in* para soportar el secuenciamiento de diagramas *UML*, un *plug-in* para explorar la base de datos, y otros muchos. Estos *plug-ins* dan soporte a los *plug-in* de usuario.

El *SDK* de *Eclipse* contiene también las *Herramientas de Desarrollo Java de Eclipse* (*Eclipse Java Development* Tools o *JDT*), ofreciendo un *IDE* con un compilador de *Java* y un modelo completo de los archivos fuente de *Java*. Esto permite técnicas avanzadas de refactorización y análisis de código. El *IDE* también hace uso de un espacio de trabajo, en este caso un grupo de *metadata* en un espacio para archivos plano, permitiendo modificaciones externas a los archivos en tanto se refresque el espacio de trabajo correspondiente.

En cuanto a las aplicaciones clientes, eclipse provee al programador con *frameworks* muy ricos para el desarrollo de aplicaciones gráficas, definición y manipulación de modelos de software, aplicaciones web, etc. Por ejemplo, *GEF* (*Graphic Editing Framework* - Framework para la edición gráfica) es un *plug-in* de Eclipse para el desarrollo de editores visuales que pueden ir desde procesadores de texto *wysiwyg* hasta editores de diagramas *UML*, interfaces gráficas para el usuario (*GUI*), etc. Dado que los editores realizados con *GEF* se encuentran “dentro” de Eclipse, además de poder ser usados conjuntamente con otros *plugins*, hacen uso de su interfaz gráfica personalizable y profesional.

3.2.2 Librerías utilizadas

3.2.2.1 Librería WiiUse

Tras la investigación de algunas aplicaciones y bibliotecas para controlar las entradas y salidas del Wiimote, se opto por la librería WiiUse [8], ya que permite la interacción con el Wiimote en un lenguaje de programación familiar, como puede ser C o Java (lenguaje usado en el proyecto). Entre las APIs disponibles se encuentra WiiUseJ, permitiendo en nuestro proyecto la interacción con el mando, dicha API es de destacar entre las otras por su facilidad de uso.

Una diferencia importante de esta librería frente a otras es que no necesita del complemento de terceras librerías de soporte como Avetana o BlueCove, ya que WiiUseJ es una API que trabaja directamente sobre la biblioteca WiiUse, la cual está disponible tanto para Linux como para Windows.

Funcionamiento

Para que WiiUseJ [9] acceda a la librería WiiUse, es necesario copiar los archivos de esta última librería (.dll si estamos en Windows o .so si estamos en Linux) en la carpeta principal de nuestro proyecto Java. En este proyecto ya que trabajamos bajo el sistema operativo Windows utilizamos la librería .dll.

A continuación se detallan algunos de los métodos de la API WiiUseJ con mayor relevancia en el proyecto.

onButtonsEvent

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

onMotionSensingEvent

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

Para más detalle sobre esta API, encontraremos información en el Anexo I.

3.2.2.2 Librería Robot

Para el manejo del Microbot TeachMover se ha utilizado la librería Robot.jar, la cual contiene los métodos que permite realizar los diversos movimientos al robot mediante un lenguaje de programación bien conocido, como es Java. Dicha librería ha tenido que ser modificada durante el proceso de programación, persiguiendo el objetivo de obtener un mayor realismo en los movimientos.

Alguno de los métodos más importantes de la librería robot:

MOVE

move(Location loc): Permite mover el robot a una posición y orientación especifica.

DRAW

Draw(double distanceX, double distanceY, double distanceZ): Permite mover el robot a una posición determinada, mediante las coordenas x,y,z.

En el Anexo II encontraremos de forma ampliada todos los métodos disponibles de la librería Robot.

3.2.2.3 Librería Giovynet

Para establecer la comunicación mediante Java, se provee de una librería para el manejo de comunicaciones a través de puertos como el serial y paralelo (la librería comm), sin embargo, el uso e implementación de esta librería está enfocado para sistemas Linux y Solaris, y su configuración en Windows es un tanto complicada. Además ya no existe soporte actualizado para esta librería por lo que fue necesario optar por una mejor opción.

Giovynet nos da la posibilidad de manejar fácilmente el puerto serie, y además es compatible con Microsoft Windows, dicha librería provee de métodos para enviar y recibir datos de los puertos serial y paralelo.

Podemos encontrar información más detallada sobre Giovynet en el Anexo III.

3.2.3 TeachVal II

Es un software que se ejecuta en su propio entorno, donde la edición, compilación y la ejecución se realiza bajo una interfaz gráfica de usuario intuitiva. Este software utiliza toda la potencia de Java incluyendo todas las escrituras de control y métodos de programación que permite una mayor libertad, permitiendo a los usuarios escribir programas en Java para el control de los robots. TeachVAL II fue utilizado en el proyecto en modo de prueba, ya que es el software oficial del Microbot TeachMover.

Capitulo 4

Caso de Estudio

En este capítulo se aborda con detalle la creación de una aplicación capaz de interactuar de manera remota entre el Wiimote y el Microbot TeachMover. Explicando de forma detallada el contenido de la aplicación y los pasos necesarios que hacen posible el control del robot mediante una interfaz de bajo coste. En la primera sección del capítulo presente, se realiza una introducción explicando a rasgos generales como se ha desarrollado dicha aplicación y a continuación, se explica de forma más detallada el contenido y los diversos procesos que han sido necesarios para su creación.

4.1 Introducción

En el ámbito de la investigación tecnológica, la elección de una hipótesis es generalmente considerada como el punto de partida, que el caso de estudio intentará confirmar o invalidar. El caso de estudio es ante todo un medio para estudiar detalladamente un ejemplo en el que se presenta una historia positiva sobre los beneficios que proporciona un producto.

Hacer uno o varios casos de estudio supone por tanto partir de uno o varios ejemplos reales, con el fin de obtener un conocimiento profundo del tema estudiado, en la medida de lo posible.

Este Proyecto Fin de Carrera tiene por objetivo la creación de una aplicación innovadora teleoperada, por tanto el caso de estudio seleccionado se centra en el desarrollo de una aplicación de control que permita manejar de forma remota el Microbot TeachMover mediante el mando Wiimote.

Para ello fue necesario establecer dos tipos de comunicación, la primera es la encargada del manejo de eventos generados por el operador al mover y accionar los botones del Wiimote y la segunda comunicación, nos permitirá transformar los eventos en órdenes legibles por el Microbot que serán enviadas a través de la comunicación puerto serie establecida.

Para el correcto funcionamiento de la aplicación desarrollada es necesario que se haya establecido previamente la conexión Bluetooth entre el Wiimote y el PC, de lo contrario cuando arranque el programa saldrá un mensaje de error notificando que no se encuentra ningún mando conectado a la aplicación. Por el contrario, si la conexión ha sido exitosa se enciende el segundo led del mando.

En la siguiente figura [Figura 4.1] se puede observar de forma esquematizada los distintos componentes que forman el sistema y el tipo de comunicación establecido entre ellos.

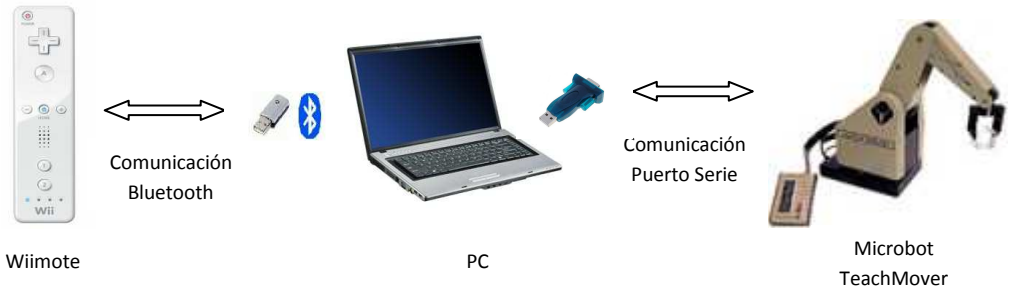

Figura 4.1: Esquema de la aplicación desarrollada

4.2 Aplicación desarrollada: 1ª parte.

En este apartado se desarrolla el proceso de conectividad entre el Wiimote y el PC, además de explicar el proceso de intercambio de datos entre ambos dispositivos.

4.2.1 Conexión del Wiimote al PC.

Sigue un proceso de conexión estándar a cualquier dispositivo HID (Human Interface Device) dentro de un SO Windows.

Para poder realizar la conexión, los pasos a seguir son los siguientes:

1) *Encender Wiimote*. Será necesario poner el Wiimote en modo discovery presionando una sola vez al mismo tiempo los botones 1 y 2, o presionando el botón rojo que hay debajo de la tapa de las pilas (botón de sincronización) del Wiimote. Al ser pulsados, los leds azules que hay en la parte inferior del mando se ponen a parpadear.

2) *Obtener identificador del dispositivo*. Para ello, se debe abrir la carpeta donde se encuentra los dispositivos detectados por Bluetooth. Para acceder a esta carpeta generalmente se dispone de un acceso directo pulsando clic derecho en el icono , que se encuentra en la barra de herramientas, en la parte inferior derecha de la pantalla.

3) *Agregar dispositivo.* Una vez hecho clic, aparecerá una ventana donde se seleccionara "**Mostrar dispositivos Bluetooth**", una vez abierta esta ventana, se hará clic en "Agregar un dispositivo". A continuación aparecerá una lista de los dispositivos Bluetooth detectados, procediendo a hacer doble clic en el icono que recibe el nombre "Nintendo RVL-CNT-01". Este dispositivo no necesita ninguna opción de autenticación o encriptación del estándar Bluetooth.

En el caso de que no diese ningún error, el PC estaría disponible para establecer la comunicación con el mando. Por el contrario en caso de error [Figura 4.2], saldría un mensaje notificándolo, teniendo que volver a realizar el proceso desde el inicio. Es posible que ocurra este tipo de error en la conexión, si tarda mucho tiempo entre la pulsación de los botones del mando y hacer clic en los iconos, ya que solo dispone de 20 segundos desde que se pulsan los botones del mando para realizar la conexión. Si fuera así, compruebe que el mando tiene todos los leds azules apagados y pulse de nuevo los botones 1 y 2, volviendo a realizar clic sobre el icono asociado al Wiimote y este se conectara al PC.

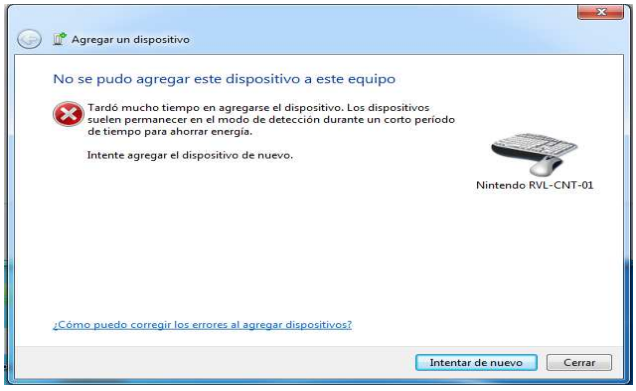

Figura 4.2: Error al agregar dispositivo

4.2.2 Apertura del programa Eclipse.

Para la realización de la aplicación desarrollada se ha seleccionado el lenguaje de programación Java, dada su versatilidad, facilidad de programación y disponibilidad de herramientas y librerías, siendo Eclipse el entorno de programación utilizado.

En la web de Eclipse existen diversos paquetes de descargas, por lo que en primer lugar hay que descargar uno de ellos y descomprimirlo. A partir de ese momento Eclipse ya es plenamente funcional, ya que no precisa instalación. Durante el proceso de carga de dicho programa aparece una ventana, donde el usuario tendrá que elegir la carpeta de trabajo donde se encuentra la aplicación, una vez elegida la carpeta de trabajo terminara el proceso de carga de Eclipse. Una vez finalizado este proceso, el usuario se encuentra a disposición de la aplicación software.

4.2.3 Descripción de la aplicación.

Esta primera parte de la aplicación es la encargada, como se ha dicho anteriormente, de realizar la interacción con el Wiimote y gestionar los eventos producidos por dicho dispositivo. Este proceso se ha realizado siguiendo el patrón observador, incorporando un manejador de eventos y haciendo uso de la librería WiiUseJ.

A continuación se describe la implementación de la aplicación llevada a cabo, que como se ve en la figura 4.3 puede dividirse en varias partes.

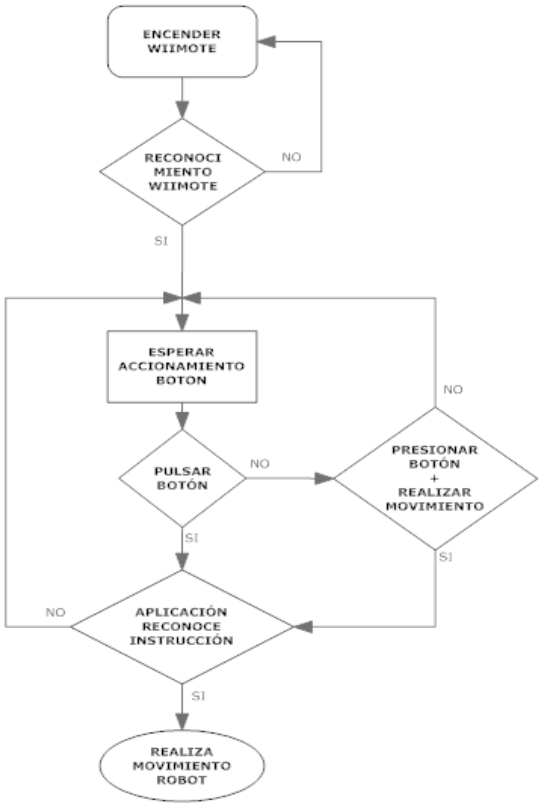

Figura 4.3: Diagrama de flujo

1) *Establecer conexión Wiimote*. En primer lugar, la aplicación desarrollada intenta detectar el mando Wiimote antes de proceder a la escucha de movimientos y/o pulsaciones de botones. El código desarrollado ha contemplado la posibilidad de incorporar multiplex conexiones Wiimote recogidas en un array. Para el caso de utilizar un único mando (como ocurre en este PFC) tan solo es necesario recuperar el Wiimote de la posición 0 del array. En este apartado también se configura el mando con las características deseadas (vibración, leds,…).

```
System.out.println("Buscando Wiimote");

while(control==2){
	try{
		//Devuelve una array con el Wiimote conectado
		Wiimote[] wiimotes = WiiUseApiManager.getWiimotes(1, true);
		wiimote = wiimotes[0];

		wiimote.setLeds(false, true, false, false);

		//Generador de eventos Wiimote angulo<0 grados
		wiimote.setOrientationThreshold(0);

		//Añado el escuchador de eventos
		wiimote.addWiiMoteEventListeners(this);

		//Activo motor de sensibilidad del Wiimote
		wiimote.activateMotionSensing();

		p=robot.getHome();

		System.out.println("Mando Wiimote encontrado");
		control=0;

	}catch(Exception e){
		System.out.println("Error,Wiimote no ha sido encontrado");
		control=2;
	}
}
```

2) *Escucha de eventos*. El flujo principal del programa es un bucle continuo que recoge los eventos producidos por el Wiimote y realiza las acciones predefinidas. Estos eventos son gestionados por la API WiiUseJ que ofrece la interfaz WiimoteListener encargada de definir las acciones a llevar a cabo en la aplicación cuando se producen los eventos recogidos por WiiUseApiListener. Define el método onButtonsEvent(WiimoteButtonsEvent arg0) y onMotionSensingEvent(MotionSensingEvent arg0).

Diagrama UML del programa.

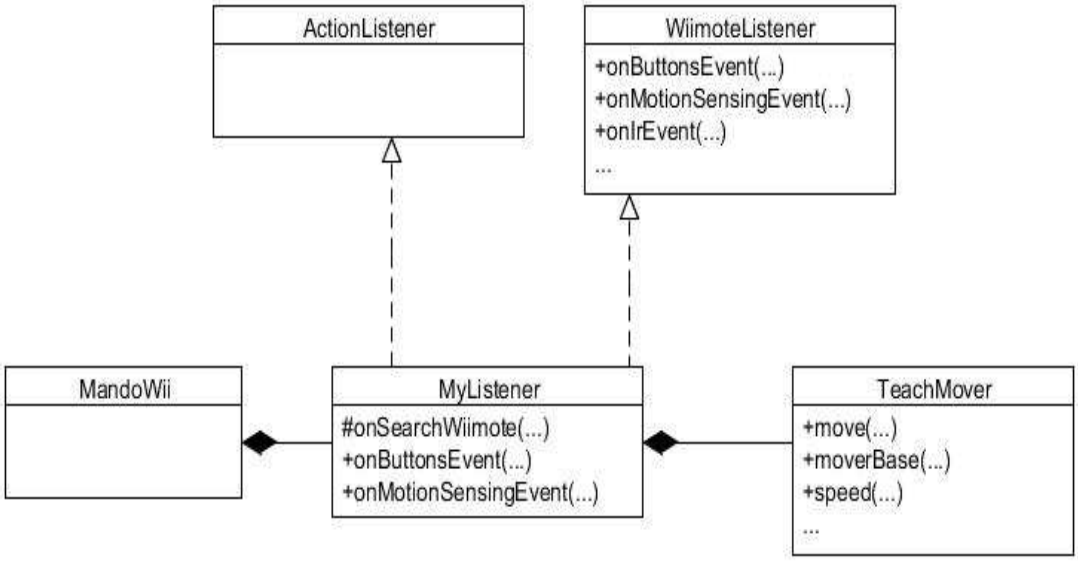

Figura 4.4: Diagrama UML

Los eventos escuchados pueden ser de dos tipos, por el accionamiento de botones o/y por la ejecución de un movimiento. Teniendo en cuenta que la seguridad en el ámbito de la teleoperación tiene gran importancia, se opto por la necesidad de introducir botones de seguridad. Estos botones implican al usuario la necesidad de presionarlos a la vez que se ejerce movimientos con el Wiimote.

En la siguiente tabla se muestra que pulsación de botones son necesarias realizar en el Wiimote para ejecutar una acción determinada en el robot.

ACCIÓN	PULSACIÓN DE BOTONES
MOVER PINZA HACIA ARRIBA	B + CRUZ DE DIRECCIONES HACIA ARRIBA
MOVER PINZA HACIA ABAJO	B + CRUZ DE DIRECCIONES HACIA ABAJO
ROTAR PINZA HACIA LA IZQUIERDA	B + CRUZ DE DIRECCIONES HACIA LA IZQUIERDA
ROTAR PINZA HACIA LA DERECHA	B + CRUZ DE DIRECCIONES HACIA LA DERECHA
POSICIÓN INICIAL DEL ROBOT	HOME
ABRIR PINZA	+
CERRAR PINZA	-

Eventos por accionamiento de botón:

```
//Detecta si boton B esta pulsado
if(arg0.isButtonBHeld()){
	boton=true;
}else{
	boton=false;
}

//Movimientos Pinza
if(arg0.isButtonUpJustPressed() & (boton==true)){
	robot.moverPinzas(15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonUpHeld() & (boton==true)){
	robot.moverPinzas(15);
}

if(arg0.isButtonDownJustPressed() & (boton==true)){
	robot.moverPinzas(-15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonDownHeld() & (boton==true)){
	robot.moverPinzas(-15);
}

if (arg0.isButtonLeftJustPressed() & (boton==true)){
	robot.moverPinzasA(-15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonLeftHeld() & (boton==true)){
	robot.moverPinzasA(-15);
}

if (arg0.isButtonRightJustPressed() & (boton==true)){
	robot.moverPinzasA(15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonRightHeld() & (boton==true)){
	robot.moverPinzasA(15);
}

//Posicion inicial Robot
if (arg0.isButtonHomeJustReleased()){
	robot.move(p);
	System.out.println("HOME");
}

//Abrir Pinza
```

```
if(arg0.isButtonPlusJustPressed()){
	System.out.println("ABRIR PINZA");
	robot.pinza(0.1);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonPlusHeld()){
	System.out.println("ABRIR PINZA");
	robot.pinza(0.1);
}

//Cerrar Pinza
if (arg0.isButtonMinusJustPressed()){
	System.out.println("CERRAR PINZA");
	robot.pinza(-0.1);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonMinusHeld()){
	System.out.println("CERRAR PINZA");
	robot.pinza(-0.1);
}

//Botones de seguridad
if(arg0.isButtonOneHeld()){
	seguridad_1=1;
}
if(arg0.isButtonTwoHeld()){
	seguridad_2=2;
}
```

En la siguiente tabla se muestra que combinación de movimientos y pulsaciones de botón son necesarias realizar en el Wiimote para ejecutar una acción determinada en el robot.

ACCIÓN	PULSACIÓN DE BOTONES
MOVER BRAZO HACIA ABAJO	2 + IMAGEN 2 HACIA ADELANTE
MOVER BRAZO HACIA ARRIBA	2 + IMAGEN 2 HACIA ATRÁS
MOVER BASE HACIA LA IZQUIERDA	2 + IMAGEN 1 HACIA LA IZQUIERDA
MOVER BASE HACIA LA DERECHA	2 + IMAGEN 1 HACIA LA DERECHA
MOVER ANTEBRAZO HACIA ABAJO	1 + IMAGEN 2 HACIA ADELANTE
MOVER ANTEBRAZO HACIA ARRIBA	1 + IMAGEN 2 HACIA ATRÁS

Imagen 1

Imagen 2

Eventos por la ejecución de movimientos:

```
//Posibles movimientos Wiimote
 if(seguridad_2==2){
      //Punto muerto Wiimote
      if((roll_media>20) & (pitch_media<-50)){
            robot.moverBase(60);
            System.out.println("MOVIMIENTO DERECHA BASE");
            seguridad_2=3;
      //Punto muerto Wiimote
      }else if((roll_media>20) & (pitch_media>50)){
            robot.moverBase(-60);
            System.out.println("MOVIMIENTO IZQUIERDA BASE");
            seguridad_2=3;
      }else if((roll_media>20) & (roll_media<40)){
            robot.moverBrazo(10);
            System.out.println("MOVIMIENTO ABAJO BRAZO");
            seguridad_2=3;
      }else if((roll_media>40) & (roll_media<65)){
            robot.moverBrazo(30);
            System.out.println("MOVIMIENTO ABAJO BRAZO");
            seguridad_2=3;
      }else if((roll_media>65) & (roll_media<90)){
            robot.moverBrazo(60);
            System.out.println("MOVIMIENTO ABAJO BRAZO");
            seguridad_2=3;
      }else if((roll_media<-20) & (roll_media>-40)){
            robot.moverBrazo(-10);
            System.out.println("MOVIMIENTO ARRIBA BRAZO");
            seguridad_2=3;
      }else if((roll_media<-40) & (roll_media>-65)){
            robot.moverBrazo(-30);
            System.out.println("MOVIMIENTO ARRIBA BRAZO");
            seguridad_2=3;
      }else if((roll_media<-65) & (roll_media>-90)){
            robot.moverBrazo(-60);
            System.out.println("MOVIMIENTO ARRIBA BRAZO");
            seguridad_2=3;
      }else if((pitch_media>20) & (pitch_media<40)){
            robot.moverBase(-10);
            System.out.println("MOVIMIENTO IZQUIERDA BASE");
            seguridad_2=3;
      }else if((pitch_media>40) & (pitch_media<65)){
            robot.moverBase(-30);
            System.out.println("MOVIMIENTO IZQUIERDA BASE");
            seguridad_2=3;
      }else if((pitch_media>65) & (pitch_media<90)){
            robot.moverBase(-60);
            System.out.println("MOVIMIENTO IZQUIERDA BASE");
            seguridad_2=3;
      }else if((pitch_media<-20) & (pitch_media>-40)){
            robot.moverBase(10);
            System.out.println("MOVIMIENTO DERECHA BASE");
            seguridad_2=3;
      }else if((pitch_media<-40) & (pitch_media>-65)){
            robot.moverBase(30);
            System.out.println("MOVIMIENTO DERECHA BASE");
            seguridad_2=3;
      }else if((pitch_media<-65) & (pitch_media>-90)){
            robot.moverBase(60);
            System.out.println("MOVIMIENTO DERECHA BASE");
            seguridad_2=3;
```

```
	}
}

if(seguridad_1==1){
	if((roll_media>20) & (roll_media<40)){
		robot.moverAnteBrazo(10);
		System.out.println("MOVIMIENTO ABAJO ANTEBRAZO");
		seguridad_1=3;
	}else if((roll_media>40) & (roll_media<65)){
		robot.moverAnteBrazo(30);
		System.out.println("MOVIMIENTO ABAJO ANTEBRAZO");
		seguridad_1=3;
	}else if((roll_media>65) & (roll_media<90)){
		robot.moverAnteBrazo(60);
		System.out.println("MOVIMIENTO ABAJO ANTEBRAZO");
		seguridad_1=3;
	}else if((roll_media<-20) & (roll_media>-40)){
		robot.moverAnteBrazo(-10);
		System.out.println("MOVIMIENTO ARRIBA ANTEBRAZO");
		seguridad_1=3;
	}else if((roll_media<-40) & (roll_media>-65)){
		robot.moverAnteBrazo(-30);
		System.out.println("MOVIMIENTO ARRIBA ANTEBRAZO");
		seguridad_1=3;
	}else if((roll_media<-65) & (roll_media>-90)){
		robot.moverAnteBrazo(-60);
		System.out.println("MOVIMIENTO ARRIBA ANTEBRAZO");
		seguridad_1=3;
	}
}
```

En el código anterior se observa que además de capturar los distintos eventos de movimiento que han sido necesarios para el desarrollo del proyecto, se ha capturado también, distintos intervalos de inclinación del mando. Estos intervalos nos han permitido asociar la inclinación del mando a una velocidad de movimiento determinada, proporcionando distintas velocidades en los movimientos del Microbot TeachMover. Dicho código también contempla la captura de dos eventos de movimiento llamados en el código fuente “Punto muerto Wiimote”, estas capturas fueron necesarias ya que cuando realizaba ciertos movimientos con el Wiimote, el Microbot TeachMover se movía de forma antinatural.

Para evitar sobresaltos en la recepción de eventos por la ejecución de movimientos Wiimote, se opto por introducir un filtro de datos. Este filtro nos proporciona una atenuación de los eventos recibidos, evitando en determinados momentos, movimientos inesperados por parte del Microbot TeachMover.

Filtro de datos:

```
if(cont%2==0){
	roll_1=arg0.getOrientation().getRoll();
	pitch_1=arg0.getOrientation().getPitch();
	roll_media=(roll_1*0.8)+(roll_2*0.2);
	pitch_media=(pitch_1*0.8)+(pitch_2*0.2);
}else{
	roll_2=arg0.getOrientation().getRoll();
	pitch_2=arg0.getOrientation().getPitch();
	roll_media=(roll_2*0.8)+(roll_1*0.2);
	pitch_media=(pitch_2*0.8)+(pitch_1*0.2);
}
```

4.3 Aplicación desarrollada: 2ª parte.

En esta parte de la aplicación, se desarrollara la comunicación necesaria entre el Microbot TeachMover y el PC, encargado de escuchar los eventos provenientes del Wiimote y ejecutar una rutina de instrucciones en función del evento escuchado.

4.3.1 Funcionamiento de la librería Robot.

Una vez capturada la acción realizada por el usuario, se procede al envió de información. Como se ha comentado en el tema anterior para el manejo del Microbot TeachMover se ha utilizado la librería Robot.jar, la cual no nos permitía realiza ciertos movimientos con el robot. Por lo que se opto por la creación de nuevos métodos, persiguiendo el objetivo de obtener nuevas instrucciones que nos proporcionaran un mayor realismo en los movimientos.

Métodos de la librería Robot usadas en el proyecto

move(p): Permite mover el robot a la posición inicial de arranque, pulsado el botón HOME del Wiimote.

Para más información sobre esta librería, diríjase al Anexo II.

Métodos específicos creados en el PFC

moverPinzas(int valor): Permite mover la pinza hacia arriba/abajo del robot, dependiendo del valor introducido (positivo o negativo). Este movimiento se genera cuando el usuario pulsa en la cruz de direcciones adelante/atrás, a la misma vez que tiene pulsado el botón B del mando.

Código fuente del método "moverPinzas". Dicho método está compuesto por seis variables de tipo entero, representando cada variable cada uno de los seis motores que contiene el Microbot TeachMover. Para poder realizar el movimiento que nos permite mover la pinza hacia arriba del robot es necesario incrementar el valor del motor RightWrist, por el contrario si queremos bajar la pinza, necesitamos decrementar el valor del motor LeftWrist. Una vez realizado el cambio de variables dependiendo del movimiento que se quiere realizar, se procede a la ejecución del movimiento mediante el método "move", que espera recibir una variable de tipo "Locatión" la cual contiene los valores de los distintos motores.

```
public void moverPinzas(int valor)
{
   int i = currentLocation.getBase() ;
   int j = currentLocation.getShoulder() ;
   int k = currentLocation.getElbow() ;
   int l = currentLocation.getRightWrist()+valor;
   int i1 = currentLocation.getLeftWrist()-valor ;
   int j1 = currentLocation.getGripper() ;
   Location l3=new Location(i, j, k, l, i1, j1, "l3");
   move(l3);
}
```

moverPinzasA(int valor): Permite rotar la pinza hacia la izquierda/derecha del robot, dependiendo del valor introducido (negativo o positivo). Este movimiento se genera cuando el usuario pulsa en la cruz de direcciones izquierda/derecha, a la misma vez que tiene pulsado el botón B del mando.

pinza(int valor): Permite abrir/cerrar la pinza del robot, dependiendo del valor introducido (positivo o negativo). Este movimiento se genera cuando el usuario pulsa el botón plus o minus del mando.

moverBase(int valor): Permite mover la base hacia la izquierda/derecha del robot, dependiendo del valor introducido (negativo o positivo). Este movimiento se genera cuando el usuario gira el mando hacia la izquierda/derecha (Figura 4.5), a la misma vez que tiene pulsado el botón 2 del mando.

Figura 4.5: Movimiento Wiimote

moverBrazo(int valor): Permite mover el brazo hacia arriba/abajo del robot, dependiendo del valor introducido (positivo o negativo). Este movimiento se genera cuando el usuario rota el mando hacia adelante/atrás (Figura 4.6), a la misma vez que tiene pulsado el botón 2 del mando.

Figura 4.6: Movimiento Wiimote

moverAnteBrazo(int valor): Permite mover el antebrazo hacia arriba/abajo del robot, dependiendo del valor introducido (positivo o negativo). Este movimiento se genera cuando el usuario rota el mando hacia adelante/atrás (Figura 4.7), a la misma vez que tiene pulsado el botón 1 del mando.

Figura 4.7: Movimiento Wiimote

Código fuente de los métodos creados en el Proyecto Fin de Carrera.

```
public void moverBase(int m)
{
  int i = currentLocation.getBase() + m;
  int j = currentLocation.getShoulder() ;
  int k = currentLocation.getElbow() ;
  int l = currentLocation.getRightWrist();
  int i1 = currentLocation.getLeftWrist() ;
  int j1 = currentLocation.getGripper() ;
  Location lo=new Location(i, j, k, l, i1, j1, "lo");
  move(lo);
}

 public void moverBrazo(int m)
 {
  int i = currentLocation.getBase();
  int j = currentLocation.getShoulder()+m ;
  int k = currentLocation.getElbow() ;
  int l = currentLocation.getRightWrist();
  int i1 = currentLocation.getLeftWrist() ;
  int j1 = currentLocation.getGripper() ;
  Location l1=new Location(i, j, k, l, i1, j1, "l1");
  move(l1);
 }

 public void moverAnteBrazo(int m)
 {
   int i = currentLocation.getBase();
   int j = currentLocation.getShoulder() ;
   int k = currentLocation.getElbow()+m ;
   int l = currentLocation.getRightWrist();
   int i1 = currentLocation.getLeftWrist() ;
   int j1 = currentLocation.getGripper()+m ;
   Location l2=new Location(i, j, k, l, i1, j1, "l2");
   move(l2);
 }

 public void moverPinzas(int m)
 {
   int i = currentLocation.getBase() ;
   int j = currentLocation.getShoulder() ;
   int k = currentLocation.getElbow() ;
   int l = currentLocation.getRightWrist()+m;
   int i1 = currentLocation.getLeftWrist()-m ;
   int j1 = currentLocation.getGripper() ;
   Location l3=new Location(i, j, k, l, i1, j1, "l3");
   move(l3);
 }
 public void moverPinzasA(int m)
 {
   int i = currentLocation.getBase() ;
   int j = currentLocation.getShoulder() ;
   int k = currentLocation.getElbow() ;
   int l = currentLocation.getRightWrist()+m ;
   int i1 = currentLocation.getLeftWrist()+m ;
   int j1 = currentLocation.getGripper() ;
   Location l4=new Location(i, j, k, l, i1, j1, "l4");
   move(l4);
 }
```

```
public void pinza(double d)
{
  d *= 371D;
  int i = (int)d - currentOpening;
  if(allowExecute)
  {
     if(comm.step(speed, 0, 0, 0, 0, 0, i) != 1)
     {
        allowExecute = false;
     } else
     {
        RobotComm _tmp = comm;
        RobotComm _tmp1 = comm;
      }
   }
}
```

4.3.2 Funcionamiento de Giovynet.

Para poder enviar la información mediante el puerto serie de forma satisfactoria y además compatible con Windows, se utiliza la librería Giovynet. Gracias a las librerías importadas dentro de la clase TeachMover se ha podido establecer la comunicación mediante el puerto serie.

Librerías importadas:

import giovynet.serial.Baud

import giovynet.serial.Com

import giovynet.serial.Parameters

import giovynet.nativelink.SerialPort

Para más información sobre Giovynet, diríjase al Anexo III.

Capitulo 5

Conclusiones y trabajos futuros

Este capítulo tiene como propósito argumentar los objetivos conseguidos, además de las distintas líneas de investigación abierta tras la realización del proyecto.

5.1 Conclusiones

Los sistemas teleoperados desde sus inicios han tenido un papel importante en la industria, obteniendo grandes avances y abarcando cada vez más el área de acción. Esta evolución ha tenido un resurgir en los últimos años debido principalmente a la gran cantidad de aplicaciones no tradicionales que han surgido. En la actualidad las aplicaciones teleoperadas pueden ir desde la diversión y el entrenamiento hasta el rescate de personas en peligro.

Gracias a estos avances en el campo de la teleoperación he podido desarrollar una nueva forma de interacción entre un operador y un robot manipulado. Todo ello ha sido posible a la gran expansión y experimentación que está teniendo el Wiimote, obteniendo una diversidad de documentación y aplicaciones ya existentes que me han servido de gran ayuda para el desarrollo de este proyecto. La gran expansión de este dispositivo lo convierte en un producto muy asequible para el desarrollo de aplicaciones.

En este Proyecto Fin de Carrera se ha logrado cumplir todos los objetivos que inicialmente estaban expuestos.

Los objetivos alcanzados son:

- Estudio de los desarrollos actuales
- Estudio de interfaces Humano-Maquina
- Estudio de la API WiiUseJ, de desarrollo para Wiimote
- Estudio, manejo y control del Microbot TeachMover
- Identificación de aplicaciones a desarrollar
- Creación de una aplicación de teleoperación mediante Wiimote

En el ámbito personal, este proyecto me ha ayudado a enfrentarme a nuevos entornos desconocido por mí hasta el momento, como es la investigación tecnológica. Ayudándome a mejorar como ingeniero, proporcionándome nuevos conocimientos.

5.2 Trabajos futuros

La teleoperación promete ser un campo con grandes perspectivas a futuro en cuanto a nuevas aplicaciones, ya que se trata de un tema de actualidad y de gran interés, especialmente en lo que a la investigación y desarrollo se refiere.

Como trabajos futuros se puede desarrollar nuevas aplicaciones, que mejore la aplicación creada hasta el momento. Una de las posibles ampliaciones podría ser la instalación de una barra sensora USB, que nos permitiese un mayor número de reconocimiento de gestos. Está barra sensora ampliaría la capacidad de movimientos del Microbot TeachMover proporcionando un mayor realismo en los gestos realizado por los usuarios.

BIBLIOGRAFIA

Bogado Torres, Juan Manuel. (2007). *Control bilateral de robots teleoperados por convergencia de estado.*

Bonache Samaniego, Ricardo. (2009). *Wiimote Hack.*

Alcor Martín, Enrique. (2008). *Estudio y desarrollo de interfaces hombre-máquina basados en sensores inalámbricos.*

Manual Microbot TeachMover. (2002). http://www.ctrl-c.liu.se/~magnus/ROBOT.HTMLX

REFERENCIAS

[1] http://cfievalladolid2.net/tecno/cyr_01/robotica/teleoperado.htm

[2] http://cfievalladolid2.net/tecno/cyr_01/robotica/intro.htm#tipos_robot

[3]http://www.is.northropgrumman.com/by_solution/remote_platforms/product/wolverine/index.html

[4] http://www.army-guide.com/eng/product4044.html

[5] http://es.wikipedia.org/wiki/Wiimote

[6] http://www.taringa.net/posts/info/1843549/Como-funciona-el-Wiimote.html

[7] http://www.ecured.cu/index.php/Eclipse,_entorno_de_desarrollo_integrado

[8] http://www.wiiuse.net/

[9] http://code.google.com/p/wiiusej/

ANEXOS

Anexo I

Librería WiiUseJ

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 detalla 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.

Anexo II

Librería Robot

Below is a listing all available Library Robot commands.

APPRO	APPROS
CLOSE	CLOSEI
DELAY	DEPART
DEPARTS	DRAW
GRASP	HERE
MOVE	MOVES
MOVET	MOVEST
OPEN	OPENI
READY	SHIFT
SIGNAL	SPEED
TYPE	WAIT

APPRO

Purpose: To move the robot to the position specified with an offset along the grippers Z (up and down) axis to the distance given.

Syntax: appro(Location location, double distance);

location is a variable of type Location that is instantiated to a reachable point for the gripper.

distance is a double value that still allows the robot to reach the point.

Example: appro(pick, 2);

This will move the gripper to a point 2 inches above a point named 'pick'.

APPROS

Purpose: Same as APPRO but arm is moved along a straight line path.

Syntax: appros(Location location, double distance);

location is a variable of type Location that is instantiated to a reachable point for the gripper.

distance is a double value that still allows the robot to reach the point.

Example: appros(pick, 2);

This will move the gripper to a point 2 inches above a point named 'pick' along a straight line path.

CLOSE

Purpose: Changes the gripper opening to the specified size before the next move. If the given size is bigger then what the gripper is currently at then the command will be equivalent to an open command.

Syntax: close(double opening);

opening is a double value of the requested opening size.

Example: close(1.5);

This will close the gripper to obtain a 1.5 inches opening. If the opening is currently smaller then 1.5 inches then the gripper will be opened to the requested size. Passing no argument will close the gripper completely.

CLOSEI

Purpose: Same as CLOSE but closes the gripper to the requested opening immediately.

Syntax: closei(double opening);

opening is a double value of the requested opening size.

Example: closei(1.5);

This will close the gripper to obtain a 1.5 inches opening. If the opening is currently smaller then 1.5 inches then the gripper will be opened to the requested size. Passing no argument will close the gripper completely.

DELAY

Purpose: Causes the program to stop executing for a specified number of seconds

Syntax: delay(double time);

time is a double value of the requested delay.

Example: delay(5);

This will cause program execution to stop for 5 seconds. The default value of delay is 2 seconds and is achieved by passing no parameter.

DEPART

Purpose: Moves the gripper the given distance along the gripper Z axis (up and down).

Syntax: depart(double distance);

distance is a double value of the requested departure.

Example: depart(2);

This will move the gripper a distance of 2 inches upwards from the current location. If a negative value is passed then the gripper will move downwards.

DEPARTS

Purpose: Moves the gripper the given distance along the gripper Z axis (up and down) along a straight line path.

Syntax: departs(double distance);

distance is a double value of the requested departure.

Example: departs(2);

This will move the gripper a distance of 2 inches upwards from the current location using a straight line path. If a negative value is passed then the gripper will move downwards.

DRAW

Purpose: To move the gripper an incremental amount.

Syntax: draw(double distanceX, double distanceY, double distanceZ);

distanceX is a double value of the requested distance in the X axis.
distanceY is a double value of the requested distance in the Y axis.
distanceZ is a double value of the requested distance in the Z axis.

Example: draw(2, -3 4);

This will move the gripper a distance of 2 inches along the positive X axis, 3 inches along the negative Y axis and 4 inches along the positive Z axis. Any of the values can be 0 if you do not wish to change that particular location.

GRASP

Purpose: To close the gripper on an object and return the size of the object in the gripper.

Syntax: grasp();

Example: grasp();

This will close the gripper on an object if one exists. It will then return the size of the object in between the gripper.

HERE

Purpose: To set the value of a Location variable to the current location of the robot.

Syntax: here(String name);

name is the string name that you wish to give to this new location

Example: Location center = here("center");

This will create a new Location object with the name of 'center' and its coordinates will the current coordinated of the robot.

MOVE

Purpose: To move the robot to the position and orientation specified.

Syntax: move(Location loc);

loc is the name of a previously defined Location object.

Example: move(center);

This will move the robot to the location described by the Location object named 'center'.

MOVES

Purpose: To move the robot to the position and orientation specified. Very similar to MOVE but this command moves the robot along a straight line path

Syntax: moves(Location loc);

loc is the name of a previously defined Location object.

Example: moves(center);

This will move the robot to the location described by the Location object named 'center' along a straight line path from the current location.

MOVET

Purpose: To move the robot to the position and orientation specified, by first setting the gripper opening to the requested size.

Syntax: movet(Location loc, double opening);

loc is the name of a previously defined Location object.

opening is a double value of the requested gripper opening.

Example: movet(center, 1.5);

This will move the robot to the location described by the object named 'center' but first the gripper opening will be set to 1.5 inches.

MOVEST

Purpose: To move the robot to the position and orientation specified along a straight line path, by first setting the gripper to the requested size.

Syntax: movest(Location loc, double opening);

loc is the name of a previously defined Location object.

opening is a double value of the requested gripper opening.

Example: movest(center, 1.5);

This will move the robot to the location described by the object named 'center' along a straight line path but first the gripper opening will be set to 1.5 inches.

OPEN

Purpose: To change the gripper opening immediately prior to the next move statement to the desired size.

Syntax: open(double opening);

opening is a double value of the requested gripper opening.

Example: open(2.25);

This will set the gripper opening to a width of 2.25 inches. If the current opening is smaller then 2.25 inches then the opening will get wider, if it is wider the gripper will close to the desired size.

OPENI

Purpose: To change the gripper opening immediately to the desired size.

Syntax: openi(double opening);

opening is a double value of the requested gripper opening.

Example: openi(2.25);

This will set the gripper opening to a width of 2.25 inches. If the current opening is smaller then 2.25 inches then the opening will get wider, if it is wider the gripper will lose to the desired size.

READY

Purpose: To move the robot arm to the initial (home) position.

Syntax: ready();

Example: ready();

This will move the robot arm to the initial position that is generally called the HOME position.

SHIFT

Purpose: To modify the X, Y and Z coordinated of a Location variable.

Syntax: shift(Location loc, double x, double y, double z);

loc is the Location object that you wish to shift.

x,y,z are double values corresponding to the desired shift of each of the coordinates.

Example: shift(center, 2, 0, -1);

This will shift the coordinates of the location named ‘center’ 2 inches in the positive X direction, 0 inches in the Y direction and 1 inch in the negative Z direction.

SIGNAL

Purpose: To turn the robot output ports ON or OFF.

Syntax: signal(integer whatPort[]);

whatPort can be a single integer or an array of integers between -5 and 5

Example: signal(-3);

This sets the 3rd output port to go low (become 0). If the output port is already low then there will be no change.

SPEED

Purpose: Changes the speed of execution of the proceeding robot commands by the given speed. This speed is maintained until the program finishes executing or until another SPEED command changes the speed.

Syntax: speed(int speed);

speed is an integer value between 0 and 240 corresponding to the desired speed.

Example: speed(Microbot.MEDIUM);

This will set the speed of execution to the MEDIUM speed set in the class Microbot. This speed corresponds to a value of 190.

NOTE: If the given speed is greater then 240 the default of 240 will be set. Also if the given speed is less then 0 then 0 will be set as the speed.

TYPE

Purpose: To display a message on the screen.

Syntax: type(String text);

text is a String that you wish to appear on the screen.

Example: type("Start of program");

This will print "Start of program" (without the quotes) to the system output window.

WAIT

Purpose: To suspend program execution until a desired input status is achieved.

Syntax: wait(integer whatPort[]);

whatPort can be a single integer or an array of integers between -7 and 7

Example: int array[] = {-3, 2, 7};

wait(array);

This will suspend program execution until input port 3 becomes low and inputs 2 and 7 become high.

Anexo III

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 !=0){
                                do {
                                        System.out.println(data);
                                        Thread.sleep(50);
                                        data = com2.receiveSingleDataInt();
                                } while (data != 0);
                        }
                }
        } catch (Exception e) {
                com2.close();
        }
}
```

Anexo IV

Código fuente

En este anexo se incluye el código fuente que ha sido necesario desarrollar para la creación de la aplicación.

```
public class MandoWii{

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

		MyListener me= new MyListener();
		me.onSearchWiimote();


	}

}

-----------------------------------------------------------------

import java.awt.event.*;

import wiiusej.*;
import wiiusej.wiiusejevents.utils.*;
import wiiusej.wiiusejevents.physicalevents.*;
import wiiusej.wiiusejevents.wiiuseapievents.*;

import Robot.*;


public class MyListener implements ActionListener, WiimoteListener{

	//Variables de instancia
	private Wiimote wiimote;
	private int seguridad_1,seguridad_2;
	private int control;
	private double roll_1,roll_2;
	private double pitch_1,pitch_2;
	private double cont;
	private double roll_media;
	private double pitch_media;
	private boolean boton;

	private TeachMover robot =new TeachMover("COM3");
	Location p;
```

```
	//Metodo Constructor
	public MyListener() throws Exception{
		wiimote=null;
		seguridad_1=0;
		seguridad_2=0;
		roll_1=0;
		roll_2=0;
		pitch_1=0;
		pitch_2=0;
		cont=0;
		roll_media=0;
		pitch_media=0;
		control=2;
	}

	protected void onSearchWiimote(){
		System.out.println("Buscando Wiimote");

		while(control==2){
			try{
				//Devuelve una array con el Wiimote conectado
				Wiimote[] wiimotes =
WiiUseApiManager.getWiimotes(1, true);
				wiimote = wiimotes[0];

				wiimote.setLeds(false, true, false, false);

				//Generador de eventos Wiimote angulo<0 grados
				wiimote.setOrientationThreshold(0);

				//Añado el escuchador de eventos
				wiimote.addWiiMoteEventListeners(this);

				//Activo motor de sensibilidad del Wiimote
				wiimote.activateMotionSensing();

				p=robot.getHome();

				System.out.println("Mando Wiimote encontrado");
				control=0;

			}catch(Exception e){
				System.out.println("Error,Wiimote no ha sido
encontrado");
				control=2;
			}
		}
	}


	public void onButtonsEvent(WiimoteButtonsEvent arg0){

		//Detecta si boton B esta pulsado
		if(arg0.isButtonBHeld()){
			boton=true;
		}else{
			boton=false;
		}
```

```
//Movimientos Pinza
if (arg0.isButtonUpJustPressed() & (boton==true)){
	robot.moverPinzas(15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonUpHeld() & (boton==true)){
	robot.moverPinzas(15);
}

if (arg0.isButtonDownJustPressed() & (boton==true)){
	robot.moverPinzas(-15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonDownHeld() & (boton==true)){
	robot.moverPinzas(-15);
}

if (arg0.isButtonLeftJustPressed() & (boton==true)){
	robot.moverPinzasA(-15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonLeftHeld() & (boton==true)){
	robot.moverPinzasA(-15);
}

if (arg0.isButtonRightJustPressed() & (boton==true)){
	robot.moverPinzasA(15);
	try {
		Thread.sleep(600);
	} catch (InterruptedException e) {
		e.printStackTrace();
	}
}else if(arg0.isButtonRightHeld() & (boton==true)){
	robot.moverPinzasA(15);
}

//Posicion inicial Robot
if (arg0.isButtonHomeJustReleased()){
	robot.move(p);
	System.out.println("HOME");
}
```

```
		//Abrir Pinza
		if (arg0.isButtonPlusJustPressed()){
			System.out.println("ABRIR PINZA");
			robot.pinza(0.1);
			try {
				Thread.sleep(600);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}

		}else if(arg0.isButtonPlusHeld()){
			System.out.println("ABRIR PINZA");
			robot.pinza(0.1);
		}

		//Cerrar Pinza
		if (arg0.isButtonMinusJustPressed()){
			System.out.println("CERRAR PINZA");
			robot.pinza(-0.1);
			try {
				Thread.sleep(600);
			} catch (InterruptedException e) {
				e.printStackTrace();
			}
		}else if(arg0.isButtonMinusHeld()){
			System.out.println("CERRAR PINZA");
			robot.pinza(-0.1);
		}

		//Botones de seguridad
		if(arg0.isButtonOneHeld()){
			seguridad_1=1;
		}
		if(arg0.isButtonTwoHeld()){
			seguridad_2=2;
		}
	}

	public void onMotionSensingEvent(MotionSensingEvent arg0){

		if(cont%2==0){
			roll_1=arg0.getOrientation().getRoll();
			pitch_1=arg0.getOrientation().getPitch();
			roll_media=(roll_1*0.8)+(roll_2*0.2);
			pitch_media=(pitch_1*0.8)+(pitch_2*0.2);
		}else{
			roll_2=arg0.getOrientation().getRoll();
			pitch_2=arg0.getOrientation().getPitch();
			roll_media=(roll_2*0.8)+(roll_1*0.2);
			pitch_media=(pitch_2*0.8)+(pitch_1*0.2);
		}
		cont=cont+1;
```

```
//Posibles movimientos Wiimote
if(seguridad_2==2){
	//Punto muerto Wiimote
	if((roll_media>20) & (pitch_media<-50)){
		robot.moverBase(60);
		System.out.println("MOVIMIENTO DERECHA BASE");
		seguridad_2=3;
	//Punto muerto Wiimote
	}else if((roll_media>20) & (pitch_media>50)){
		robot.moverBase(-60);
		System.out.println("MOVIMIENTO IZQUIERDA BASE");
		seguridad_2=3;
	}else if((roll_media>20) & (roll_media<40)){
		robot.moverBrazo(10);
		System.out.println("MOVIMIENTO ABAJO BRAZO");
		seguridad_2=3;
	}else if((roll_media>40) & (roll_media<65)){
		robot.moverBrazo(30);
		System.out.println("MOVIMIENTO ABAJO BRAZO");
		seguridad_2=3;
	}else if((roll_media>65) & (roll_media<90)){
		robot.moverBrazo(60);
		System.out.println("MOVIMIENTO ABAJO BRAZO");
		seguridad_2=3;
	}else if((roll_media<-20) & (roll_media>-40)){
		robot.moverBrazo(-10);
		System.out.println("MOVIMIENTO ARRIBA BRAZO");
		seguridad_2=3;
	}else if((roll_media<-40) & (roll_media>-65)){
		robot.moverBrazo(-30);
		System.out.println("MOVIMIENTO ARRIBA BRAZO");
		seguridad_2=3;
	}else if((roll_media<-65) & (roll_media>-90)){
		robot.moverBrazo(-60);
		System.out.println("MOVIMIENTO ARRIBA BRAZO");
		seguridad_2=3;
	}else if((pitch_media>20) & (pitch_media<40)){
		robot.moverBase(-10);
		System.out.println("MOVIMIENTO IZQUIERDA BASE");
		seguridad_2=3;
	}else if((pitch_media>40) & (pitch_media<65)){
		robot.moverBase(-30);
		System.out.println("MOVIMIENTO IZQUIERDA BASE");
		seguridad_2=3;
	}else if((pitch_media>65) & (pitch_media<90)){
		robot.moverBase(-60);
		System.out.println("MOVIMIENTO IZQUIERDA BASE");
		seguridad_2=3;
	}else if((pitch_media<-20) & (pitch_media>-40)){
		robot.moverBase(10);
		System.out.println("MOVIMIENTO DERECHA BASE");
		seguridad_2=3;
	}else if((pitch_media<-40) & (pitch_media>-65)){
		robot.moverBase(30);
		System.out.println("MOVIMIENTO DERECHA BASE");
		seguridad_2=3;
	}else if((pitch_media<-65) & (pitch_media>-90)){
		robot.moverBase(60);
		System.out.println("MOVIMIENTO DERECHA BASE");
		seguridad_2=3;}
```

```
       if(seguridad_1==1){
            if((roll_media>20) & (roll_media<40)){
                  robot.moverAnteBrazo(10);
                  System.out.println("MOVIMIENTO ABAJO ANTEBRAZO");
                  seguridad_1=3;
            }else if((roll_media>40) & (roll_media<65)){
                  robot.moverAnteBrazo(30);
                  System.out.println("MOVIMIENTO ABAJO ANTEBRAZO");
                  seguridad_1=3;
            }else if((roll_media>65) & (roll_media<90)){
                  robot.moverAnteBrazo(60);
                  System.out.println("MOVIMIENTO ABAJO ANTEBRAZO");
                  seguridad_1=3;
            }else if((roll_media<-20) & (roll_media>-40)){
                  robot.moverAnteBrazo(-10);
                  System.out.println("MOVIMIENTO ARRIBA ANTEBRAZO");
                  seguridad_1=3;
            }else if((roll_media<-40) & (roll_media>-65)){
                  robot.moverAnteBrazo(-30);
                  System.out.println("MOVIMIENTO ARRIBA ANTEBRAZO");
                  seguridad_1=3;
            }else if((roll_media<-65) & (roll_media>-90)){
                  robot.moverAnteBrazo(-60);
                  System.out.println("MOVIMIENTO ARRIBA ANTEBRAZO");
                  seguridad_1=3;
            }
       }
    }

    public void
onClassicControllerInsertedEvent(ClassicControllerInsertedEvent e){
            System.out.println("Classic Controller Extension Inserted
on Wiimote #" + e.getWiimoteId());
      }

      public void
onClassicControllerRemovedEvent(ClassicControllerRemovedEvent e){
            System.out.println("Classic Controller Extension Removed on
Wiimote #" + e.getWiimoteId());
      }

    public void onIrEvent(IREvent e){
      System.out.println("Movimiento x,y,z" + e.getWiimoteId());
    }

    public void onExpansionEvent(ExpansionEvent arg0){
      System.out.println("Expansion Event on Wiimote #" +
arg0.getWiimoteId());
    }

    public void onStatusEvent(StatusEvent arg0){
      System.out.println("Status Event on Wiimote #" +
arg0.getWiimoteId());
    }

    public void onDisconnectionEvent(DisconnectionEvent arg0){
      System.out.println("Disconnection Event on Wiimote #" +
arg0.getWiimoteId());
    }
```

```
    public void onNunchukInsertedEvent(NunchukInsertedEvent arg0){
      System.out.println("Nunchuk Inserted Event on Wiimote #" +
arg0.getWiimoteId());
    }

    public void onNunchukRemovedEvent(NunchukRemovedEvent arg0){
      System.out.println("Nunchuk Removed Event on Wiimote #" +
arg0.getWiimoteId());
    }

      public void onGuitarHeroInsertedEvent(GuitarHeroInsertedEvent
arg0){
            System.out.println("Guitar Hero Inserted Event on Wiimote
#" + arg0.getWiimoteId());
      }

      public void onGuitarHeroRemovedEvent(GuitarHeroRemovedEvent
arg0){
            System.out.println("Guitar Hero Removed Event on Wiimote #"
+ arg0.getWiimoteId());
      }

      public void actionPerformed(ActionEvent e){
            // TODO Auto-generated method stub
      }


}
```