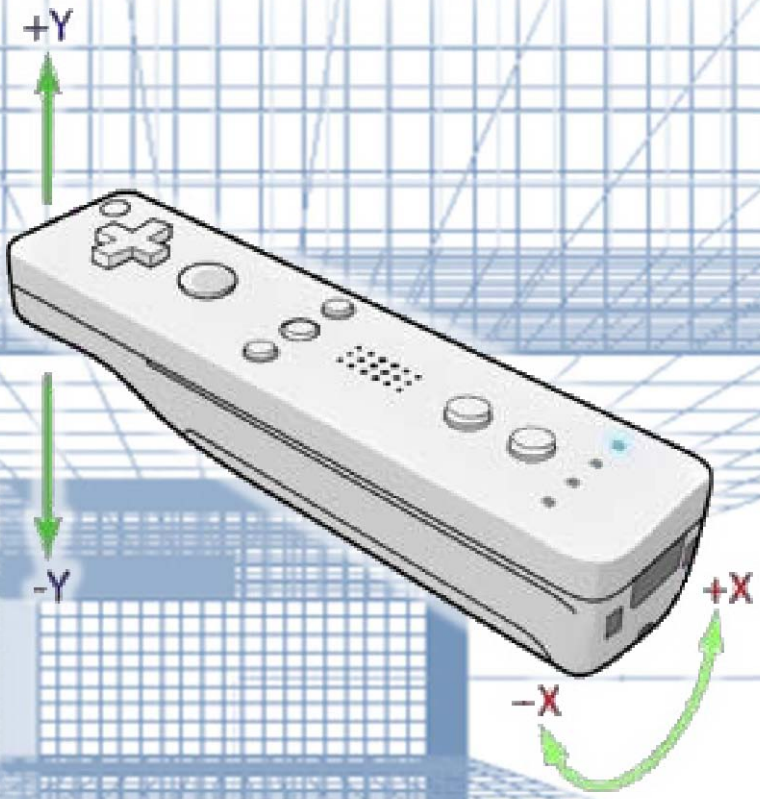
PROGRAMACIÓN DE WIIMOTE COMO INTERFAZ PARA NAVEGACIÓN EN AMBIENTES VIRTUALES



TESIS PARA OBTENER EL TÍTULO DE INGENIERO EN COMPUTACIÓN

ALUMNO:
ANDRÉS CHÁVEZ RAMOS

DIRECTOR DE TESIS:
MI. TINTOR PÉREZ RODRIGO GUILLERMO

SINODALES:
DR. ÁLVAREZ LÓPEZ MANUEL
DR. ESCALANTE RAMÍREZ BORIS
ING. VALENCIA CASTRO LUIS SERGIO
DRA. VÁZQUEZ VARGAS ANA MARÍA

2011

AGRADECIMIENTOS

Esta tesis la dedico a todas las personas que me han apoyado en cada ámbito de mi vida para lograr este objetivo.

Primero, a mi madre que sin sus consejos, exigencia y todo lo que me dio en la vida para tener siempre la oportunidad de lograr lo que me proponga, no sería el hombre que soy ahora. A mi novia que me inspiró en cada momento para seguir adelante, dándome la motivación y felicidad para lograr mis metas cada día… ¡TE AMO! A mi hermana que siempre apoyó mis decisiones, me dio sus consejos y su cariño para lograr ser una gran persona. Y por supuesto a mis amigos que compartieron conmigo muchos consejos y experiencias inolvidables a través de esta etapa de mi vida en la que fortalecimos amistades y aprendimos muchas cosas.

Además, a mi maestro y director de tesis Rodrigo Tintor que me dio la oportunidad de trabajar con él y me dio la idea de realizar este trabajo tan agradable y excepcional. A mis sinodales y a mis demás maestros que en el transcurso de mis estudios en la Facultad de Ingeniería me dieron las herramientas para lograr este proyecto, me compartieron sus conocimientos para darme un vistazo de lo que se puede hacer más adelante y me proporcionaron el gusto por la ingeniería gracias a sus consejos y buenas técnicas didácticas.

A todos… gracias.

Programación de Wiimote como interfaz para navegación en ambientes virtuales

Contenido

I. **Objetivo**

Desarrollar una aplicación para Windows capaz de configurar las funciones de un Wiimote, a través de una interfaz gráfica de fácil comprensión; con el propósito de interactuar a distancia con la computadora y facilitar la navegación en aplicaciones con ambientes virtuales.

Además, utilizar el lenguaje C++ a través de WinAPI para crear una aplicación compatible con diversos sistemas operativos Windows, para operar en conjunto con el Wiimote, aprovechando la tecnología inalámbrica Bluetooth y simular las funciones de un ratón y un teclado a distancia.

II. Definición del problema

Hoy en día los ambientes virtuales forman parte de muchas actividades, ya sean académicas, culturales, profesionales o incluso de entretenimiento, y la manipulación de estos ambientes generalmente implica el uso de dispositivos periféricos como un ratón y un teclado en conjunto. Lamentablemente, la movilidad de estos dispositivos se ve limitada por cables de corta distancia y superficies de apoyo que se requieren para su operación adecuada, por lo que en muchas ocasiones estas tareas se consideran muy complicadas, caras o difíciles de implementar, dando por consecuencia que las aplicaciones de ambientes virtuales se descarten de entre diversas opciones, provocando que el usuario prefiera otras formas gráficas distintas a los ambientes virtuales para generar y compartir una fuente de conocimiento que podría ser muy importante y trascendental si se descubriera a través de las tecnologías que involucran la realidad virtual.

Es importante destacar que en muchos sitios, como en las aulas y auditorios donde se llevan a cabo desde clases hasta congresos, se requiere de la manipulación de las computadoras y, cada día más, de aplicaciones con ambientes virtuales, siendo de gran importancia e indispensable contar con la tecnología necesaria para llevar a cabo dichas tareas de manera eficiente y a gran distancia para permitir que tanto alumnos como profesores y profesionales en distintas ramas sean capaces de utilizar sus habilidades libremente y así aprovechar todo su potencial.

Por otro lado, los ratones y los teclados inalámbricos podrían verse como una solución a este problema, sin embargo suelen ser de alto costo y no mejorarían considerablemente la movilidad de los mismos y por ende del usuario, tan sólo incrementarían la distancia a la que podrían funcionar. Por lo tanto, una solución factible para esta problemática sería un dispositivo que cuente con conectividad inalámbrica y que simule las funciones de varios dispositivos periféricos como el ratón y el teclado, además de que no requiera de algún factor físico que limite su posición y movilidad.

Es por eso que, tomando en cuenta la problemática planteada y las características innovadoras del Wiimote, en esta tesis se propone el uso de este dispositivo como alternativa para facilitar la navegación virtual, entre otras funciones.

III. Resumen

En esta tesis se desarrolló la aplicación WiimoteAPI para sistemas operativos Windows, capaz de configurar las funciones de un Wiimote, como son los botones, el acelerómetro y la extensión Nunchuck (tanto sus botones como su joystick). La WiimoteAPI se implementó con el lenguaje de programación C++ a través de WinAPI.

Se aprovechó la tecnología inalámbrica Bluetooth para comunicar el Wiimote con la computadora y se logró simular las funciones de un ratón y un teclado, logrando sustituirlos en el uso de aplicaciones de realidad virtual, así como en otras que requieren la intervención de estos periféricos simultáneamente.

Como ejemplo práctico de la operación del Wiimote y la WiimoteAPI se utilizó la aplicación VMD, comprobando así que la funcionalidad de estos dos componentes permiten asistir y facilitar el aprendizaje de cualquier usuario, tanto de estudiantes en su desarrollo como futuros profesionales en áreas que requieren la navegación virtual, así como de expertos al utilizar aplicaciones como VMD para la manipulación de objetos en ambientes virtuales, o incluso otras.

IV. Antecedentes

2.1 El Wiimote como dispositivo de interfaz humana.

El control remoto de la consola Wii de Nintendo, conocido como **Wiimote**, funciona como el dispositivo de entrada principal de dicha consola. El Wiimote es un dispositivo inalámbrico que utiliza la tecnología Bluetooth para comunicarse con otros dispositivos, utilizando un sistema Broadcom BCM2042, por lo que es posible utilizarlo, a distancia, como dispositivo de entrada de un ordenador; consta de once botones de control factibles para programar y realizar funciones del teclado, un acelerómetro capaz de simular el movimiento del ratón, un sensor de luz infrarroja y un puerto de expansión que permite utilizar otros dispositivos en conjunto con el Wiimote. Cabe destacar que el botón "POWER" del Wiimote únicamente funciona para desconectarlo de cualquier equipo al que está sincronizado y apagarlo.

Uno de los dispositivos a conectar al Wiimote es el Nunchuck, un control de mando o "joystick" que consta de una palanca capaz de moverse 360°, dos botones y un acelerómetro. A continuación se muestran las dos caras del Wiimote y el Nunchuck.

El Wiimote utiliza el protocolo estándar Bluetooth HID (Human Interface Device – Dispositivo de Interfaz Humana) para comunicarse con el dispositivo anfitrión; sin embargo, el Wiimote no utiliza los tipos de datos estándar ni el descriptor HID, el cual escribe los paquetes de datos; por contrario, sólo describe el tamaño de su formato de reporte, dejando el contenido restante indefinido, lo cual permite omitir el uso de controladores.

La comunicación entre el Wiimote y otros dispositivos se da a través de un conjunto de operaciones, las cuales se transmiten por medio de los reportes de entrada y de salida del protocolo HID, transmitiendo así los paquetes de datos distintos que contienen información de los diversos sistemas integrados en el Wiimote, ya sean botones, dispositivos conectados al puerto de expansión, acelerómetros, etc.

2.2 Conexión del Wiimote por Bluetooth.

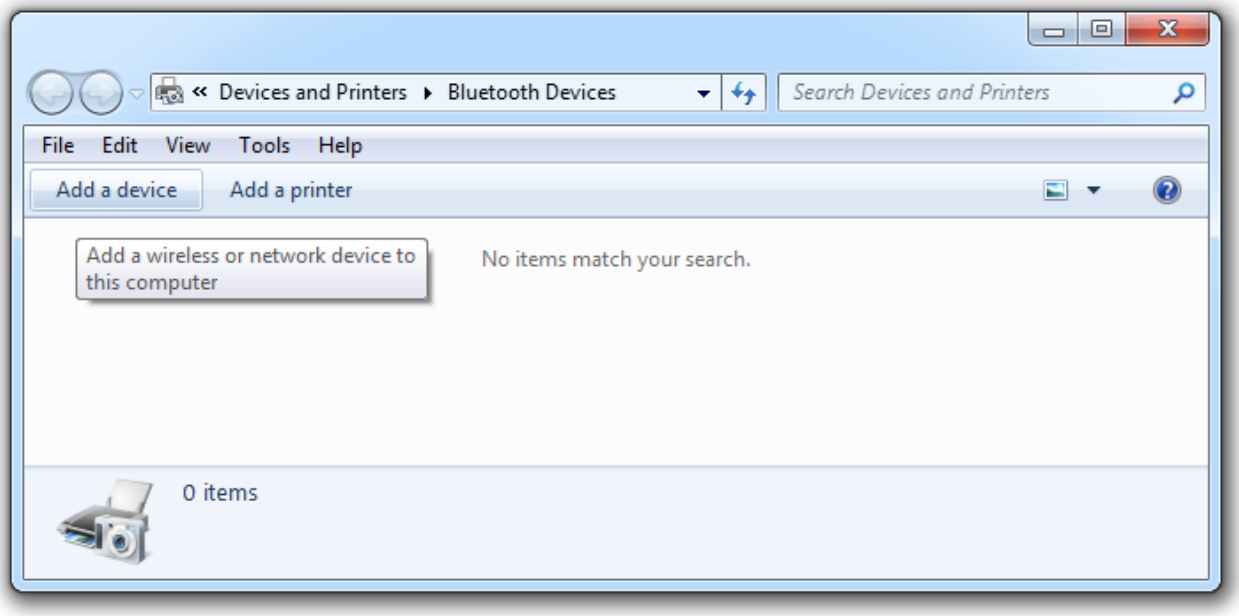
El Wiimote es invocado por medio del SDP (Protocolo de descubrimiento de servicios) de Bluetooth, un protocolo encargado de identificar los dispositivos inalámbricos con el propósito de establecer la comunicación entre estos y el ordenador. El Wiimote reporta la siguiente información a través de este protocolo:

Name	**Nintendo RVL-CNT-01**
Vendor ID	0x057e
Product ID	0x0306
Major Device Class	1280
Minor Device Class	4
Service Class	0
(Summary of all Class Values)	0x002504

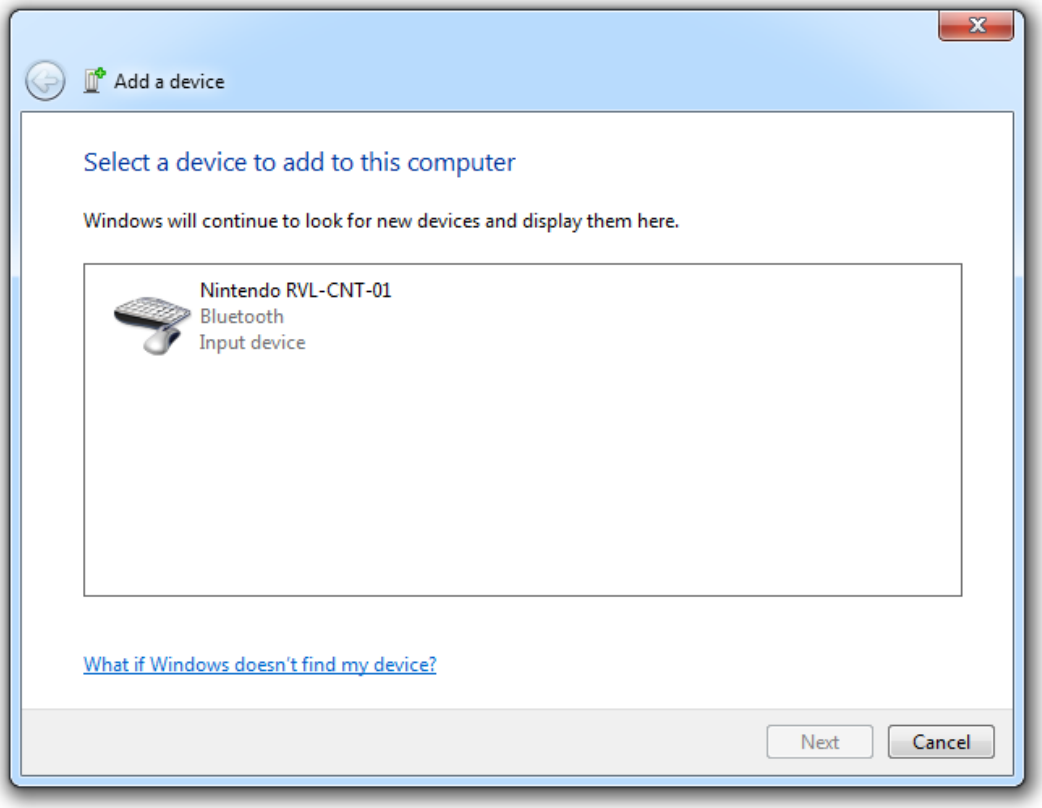
El Wiimote no requiere de autenticación o de algún tipo de cifrado del estándar de Bluetooth para comunicarse. Para reconocerlo desde un ordenador y agregarlo como HID se activa el modo de detección presionando los botones 1 y 2 al mismo tiempo, o presionando el botón de sincronización localizado bajo la cubierta de las baterías. Este estado del Wiimote dura solamente 20 segundos, a menos que se mantengan presionados los botones 1 y 2 para forzar y mantener un estado de detección permanente hasta soltar los botones, a diferencia del botón de sincronización que sólo permite 20 segundos de modo de detección aunque se mantenga presionado. Mientras el Wiimote se encuentra en modo de detección, sus LEDs encienden intermitentemente y el número de LEDs que encienden indica la cantidad proporcional de carga de las baterías.[14]

A continuación se muestra el proceso de conexión del Wiimote a una computadora:

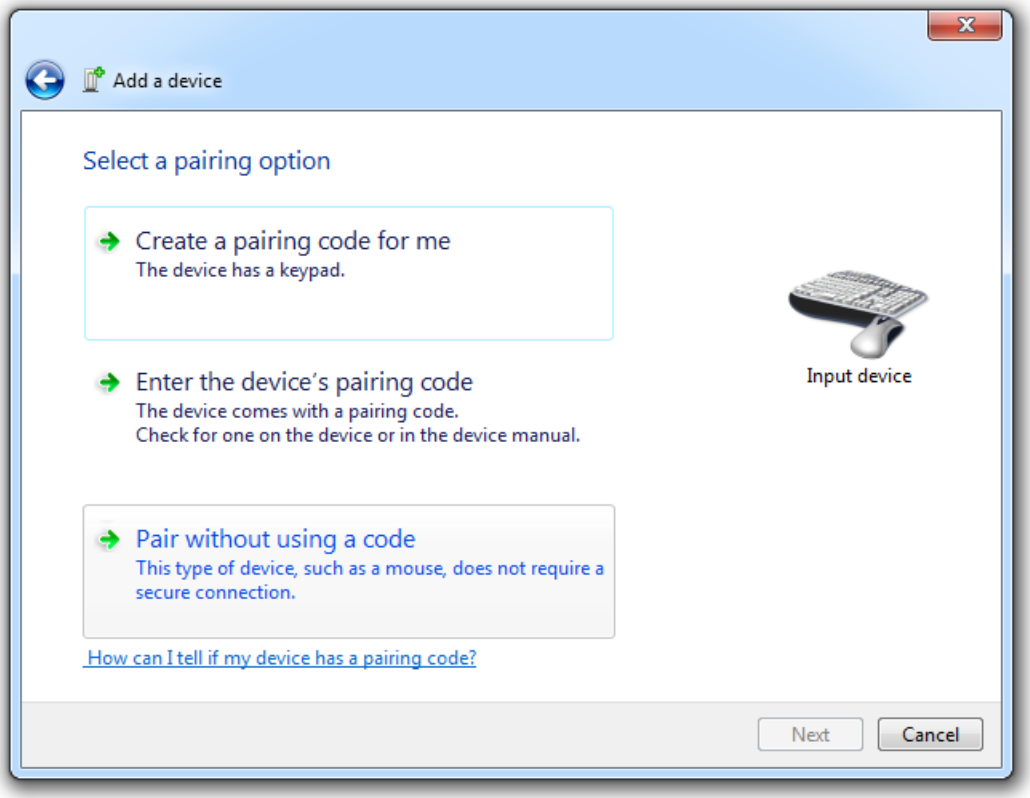
1. Abrir el panel de dispositivos Bluetooth:



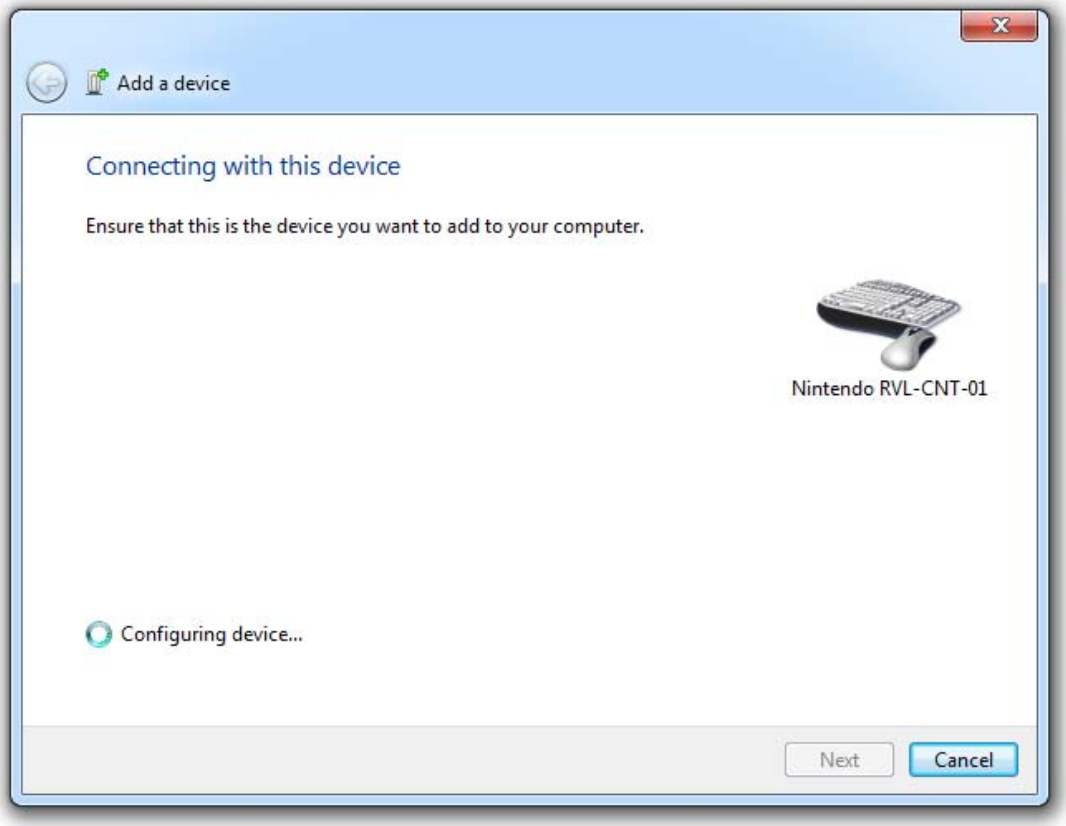
2. Hacer click en “agregar dispositivo” y colocar el Wiimote en modo de detección para visualizar el Wiimote como dispositivo a agregar:



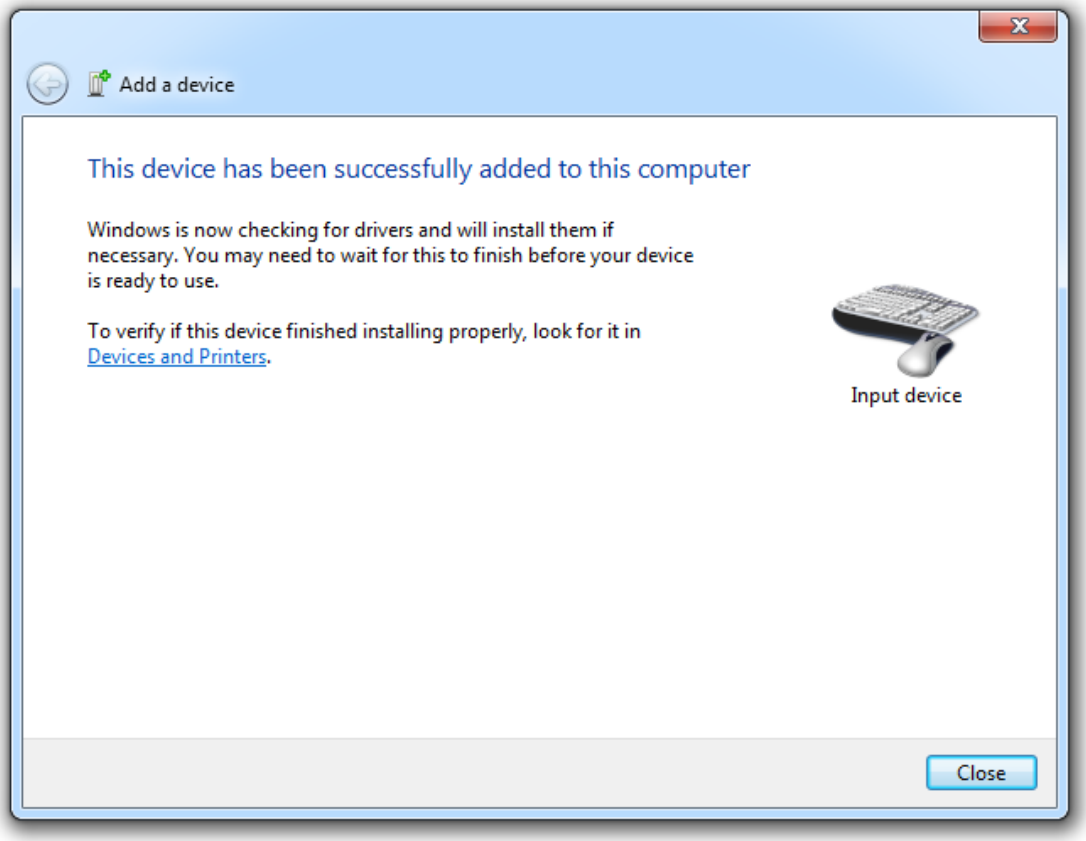
3. Hacer doble click sobre el ícono “Nintendo RVL-CNT-01”, o hacer un click sobre él y hacer click en “siguiente”; a continuación se mostrarán las opciones de conexión para el Wiimote:



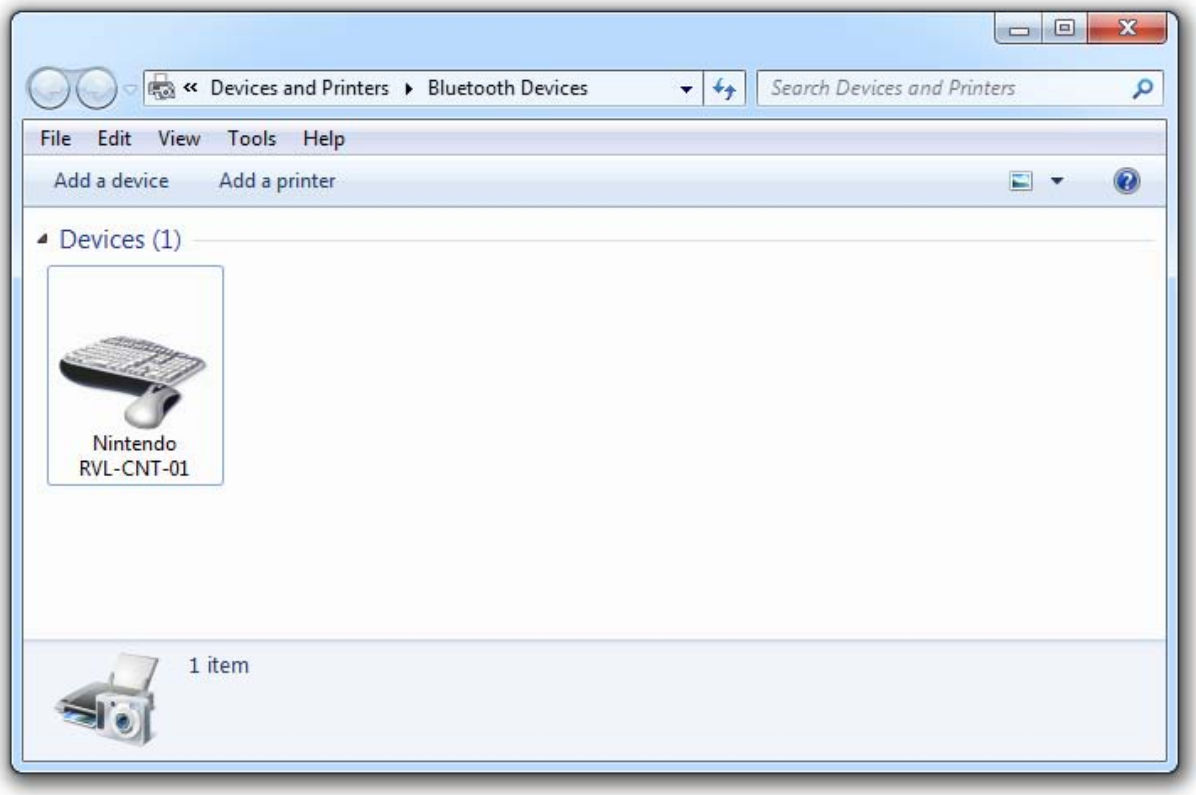
4. Dar click en “Aparear sin código” y esperar a que el equipo configure la conexión con el Wiimote:



5. Una vez que se han sincronizado el Wiimote y la PC con éxito, hacer click en cerrar:



6. Verificar que el Wiimote se agregó correctamente localizándolo en el panel de dispositivos Bluetooth:



7. Una vez que se terminó de utilizar el Wiimote se debe eliminar de los dispositivos Bluetooth ya que es incapaz de recuperar la sincronización después de que la PC o el mismo Wiimote han sido apagados, por lo que este procedimiento se debe realizar cada vez que el equipo o el Wiimote es encendido.

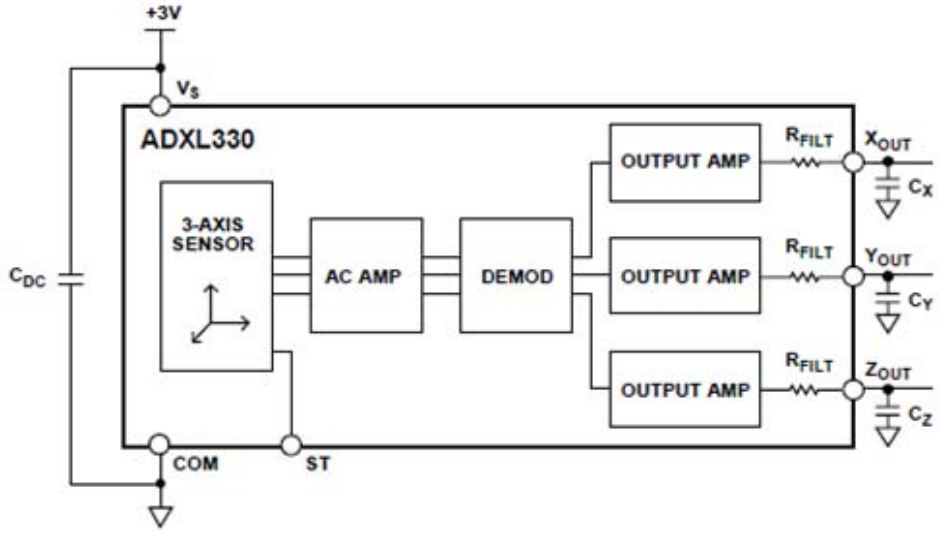
2.3 Acelerómetro ADXL330 en el Wiimote.

Un acelerómetro es un tipo de sensor de movimiento, diseñado para realizar la transducción de un fenómeno físico en una señal fácil de medir. En este caso el fenómeno físico es una fuerza originada por la aceleración a la que es sometido el sensor. Generalmente esta fuerza se convierte a una salida de voltaje linealmente escalada y con cierta sensibilidad.[5]

El Wiimote cuenta con un acelerómetro ADXL330 de baja potencia, fabricado por Analog Devices, Inc., una de las productoras de sensores inerciales más grandes del mercado, con su proceso bien establecido iMEMS (1993), el cual integró eficazmente el sensor mecánico y el procesamiento de señales en un mismo circuito.[7]

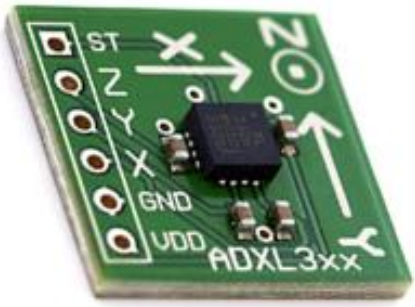
El ADXL330 ganó el concurso al mejor diseño para el **Wiimote** del Nintendo Wii, estableciendo a los MEMS (Microelectromechanical Systems - Sistemas Microelectromecánicos) como la llave de la innovación en una nueva generación de consolas de videojuegos. Además, este dispositivo MEMS surgió de una mejora del ADXL203, un acelerómetro de dos ejes, añadiendo la detección de movimiento en el eje Z y manteniendo la detección en los ejes X y Y sin cambios esenciales.[9]

Las aplicaciones más frecuentes del ADXL330 se encuentran en consolas y controles de videojuegos como el Wiimote, dispositivos móviles, dispositivos deportivos y de salud, protección de discos duros y estabilización de imagen. Consta de un sensor de 3 ejes de detección con salidas de voltaje analógico proporcionales a la aceleración registrada, en un solo circuito integrado monolítico. Este sensor mide la aceleración en un rango de ±3 g y con una sensibilidad del 10%; puede utilizarse en varias aplicaciones que requieran medir la aceleración estática de la gravedad de acuerdo a cierta inclinación a la que es sometido el sensor, así como la aceleración dinámica que resulta de movimientos, impactos y vibraciones.[5] A continuación se muestra el diagrama de bloques funcional del acelerómetro ADXL330:



El usuario puede seleccionar el ancho de banda de las señales del acelerómetro usando los pines de salida X_{OUT}, Y_{OUT} y Z_{OUT}, correspondientes a los capacitores C_X, C_Y y C_Z; y los anchos de banda se pueden configurar para satisfacer la aplicación deseada, con un rango de 0.5 Hz a 1600 Hz para los ejes X y Y, y un rango de 0.5 Hz a 550 Hz para el eje Z. Una vez que las señales salen del sensor, se digitalizan en enteros sin signo de 8 bits.

A continuación se muestra una fotografía del circuito ADXL330, con los ejes indicados a modo de mostrar el sentido positivo de la aceleración en cada uno:



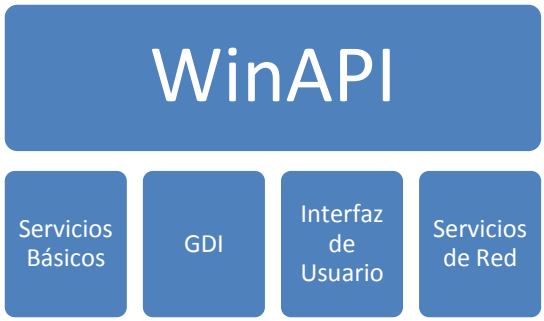
El acelerómetro contiene un sensor implementado para de medir la aceleración en una arquitectura de un ciclo abierto (sin retroalimentación). Este sensor está construido sobre una oblea de silicio; utiliza resortes de polisilicio que suspenden una estructura sobre la superficie de la oblea creando resistencia ante fuerzas de aceleración. La deflexión de la estructura debida a la aceleración se mide utilizando un capacitor diferencial que consiste de placas sostenidas de la misma estructura y otras placas fijas e independientes a la estructura. Las placas fijas se controlan por señales cuadradas 180° fuera de fase.

Cuando existe cierta aceleración la estructura se desplaza y el capacitor diferencial pierde balance resultando en tres salidas del sensor cuya amplitud es proporcional a las componentes ortogonales de la aceleración. Después se utilizan técnicas de demodulación sensible a la fase para determinar las magnitudes y direcciones de las componentes de la aceleración. Cada salida del demodulador se amplifica conecta al exterior del chip con una resistencia de 32 kΩ. Entonces el usuario puede establecer el ancho de banda de una señal añadiendo un capacitor. Si se filtran las salidas se puede mejorar la resolución de la medición y se puede prevenir el efecto de “aliasing”, evitando así la distorsión de las señales de salida.

2.4 API de Windows.

Windows es el sistema operativo más usado en la industria de las computadoras, cubriendo un poco más del 80% del mercado.[2] Su API, también conocida como **WinAPI,** es un conjunto de *códigos fuente de interfaz* que se utiliza para crear aplicaciones de Windows, con ayuda del SDK (Software Development Kit - Kit de Desarrollo de Software) de Windows, el cual contiene ficheros cabecera, bibliotecas, documentación y otras herramientas. WinAPI está diseñada para los lenguajes de programación C y C++ y es el modo más directo para crear aplicaciones de Windows.

WinAPI consta de cuatro componentes básicos:

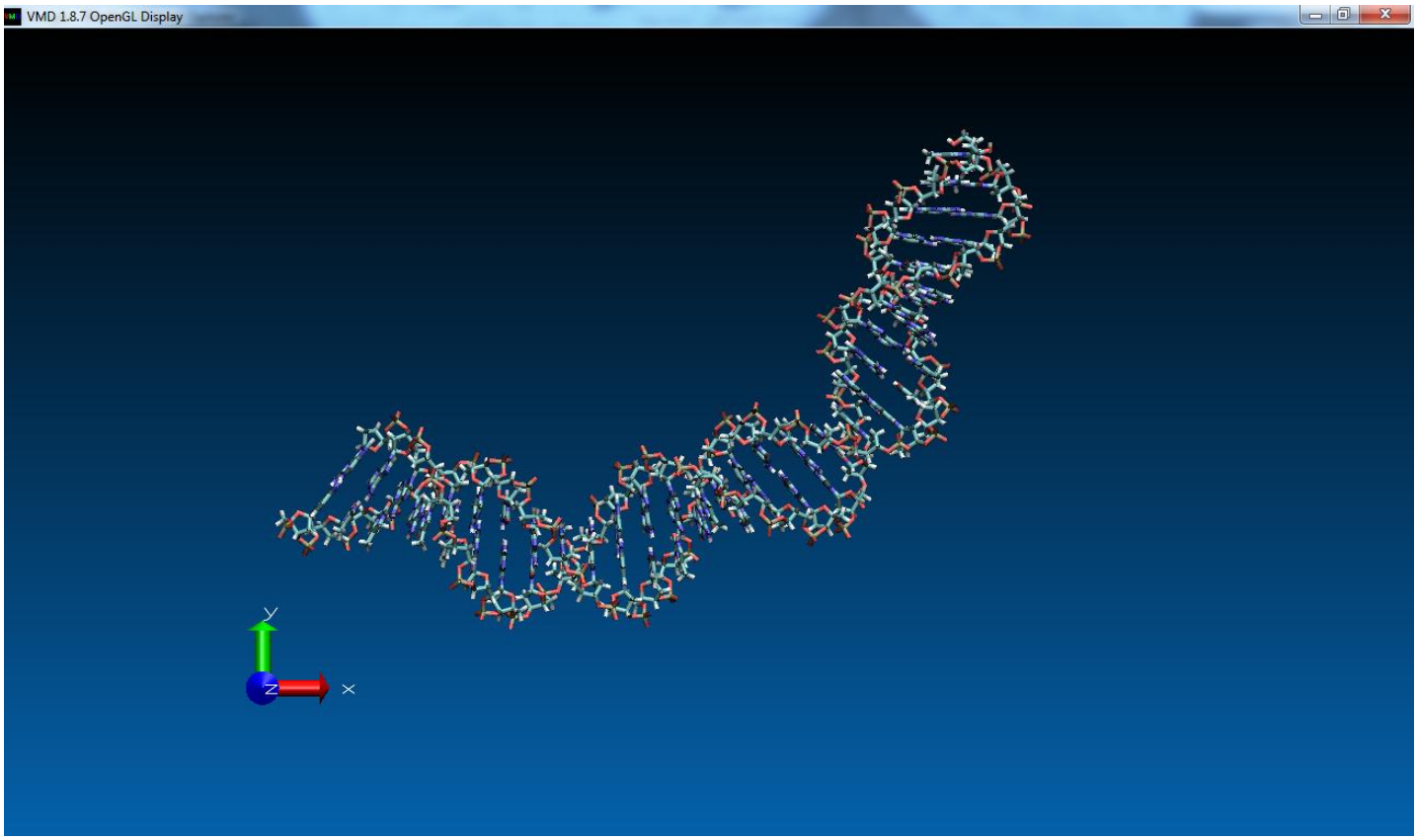


Los **Servicios Básicos** proporcionan acceso a los recursos fundamentales de Windows, entre estos se incluyen los archivos de sistema, dispositivos, procesos, hilos, registros o manejo de errores. El **GDI** (Graphics Device Interface – Interfaz de Dispositivo Gráfico) es una interfaz que permite trabajar con gráficos y permite la interacción entres dispositivos gráficos como, el monitor, la impresora o incluso archivos. La **Interfaz de Usuario** proporciona funcionalidad para crear ventanas y distintos tipos de controles como botones o menús. Los **Servicios de Red** proporcionan acceso a las capacidades de redes de Windows.

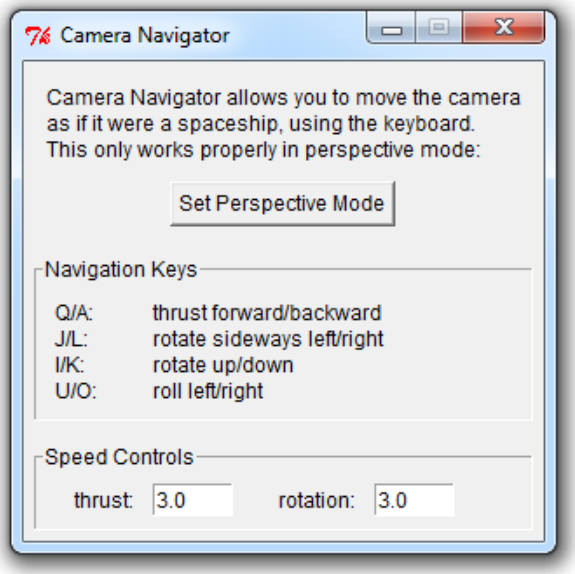
Los lenguajes de programación C y C++ pueden usar WinAPI de forma directa, es decir, las bibliotecas de esta API fueron creadas en C y C++, por lo que los demás lenguajes de programación llaman a WinAPI de forma indirecta.

2.5 Visual Molecular Dynamics

La aplicación VMD (Visual Molecular Dynamics – Dinámica Molecular Visual) es un ejemplo de ambiente virtual que permitirá ejemplificar el objetivo de esta tesis. Permite navegar a través de modelos moleculares virtuales, así como manipularlos y modificar sus propiedades visuales, entre otras tareas.[13] Por lo que cumple con las características más importantes e indispensables dentro de un sistema de navegación virtual. VMD cuenta con varias ventanas de trabajo. A continuación se muestra la ventana de navegación:

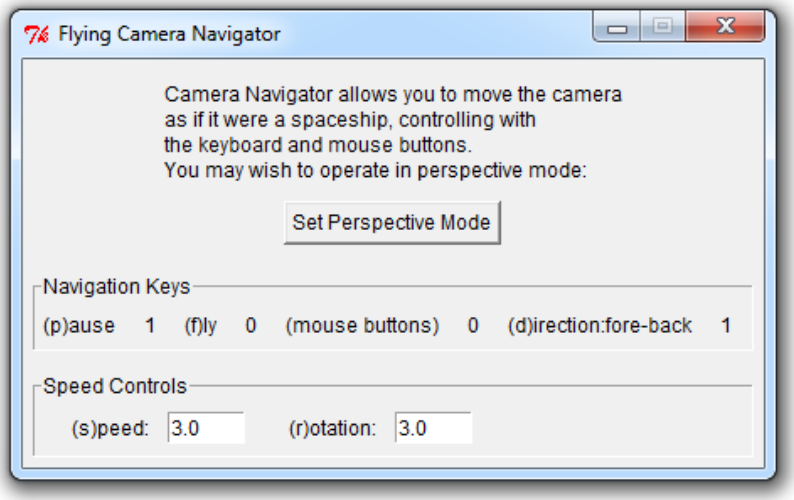


La siguiente ventana de VMD muestran las opciones de teclas a utilizar para navegar en este ambiente virtual:



Esta aplicación cuenta con dos modos de navegación virtual. El primero se realiza con las teclas mencionadas en la ventana anterior, es decir, el teclado permite manipular la cámara desde donde se ve el objeto en la ventana de navegación. El segundo se realiza con el ratón, ya que es posible rotar el modelo molecular al mantener presionado el botón izquierdo del ratón y moverlo en el sentido deseado de rotación.

Sólo tres teclas intervienen en la manipulación del modelo virtual para configurar el modo del ratón, de manipulación del modelo a navegación, así como pausar el movimiento de navegación o cambiar de dirección:



En resumen, VMD es una aplicación que depende de dos dispositivos periféricos, el ratón y el teclado, para su completa manipulación.

V. Desarrollo

5.1. Estudio del Wiimote e interfaces de usuario.

Dentro de la navegación virtual se requieren diversos dispositivos para lograr visualizar, modificar o manipular apropiadamente objetos y ambientes virtuales; entre estos están el ratón y el teclado. Sin embargo, difícilmente se puede encontrar un único dispositivo capaz de sustituirlos o incluso mejorar sus características.

El ratón es un tipo de interfaz cuya movilidad está limitada por la necesidad de una superficie plana para operar adecuadamente y en caso de ser inalámbrico su velocidad de respuesta es proporcional al costo del dispositivo, y en el teclado se da el mismo caso y por lo tanto tiene las mismas limitaciones que el ratón.

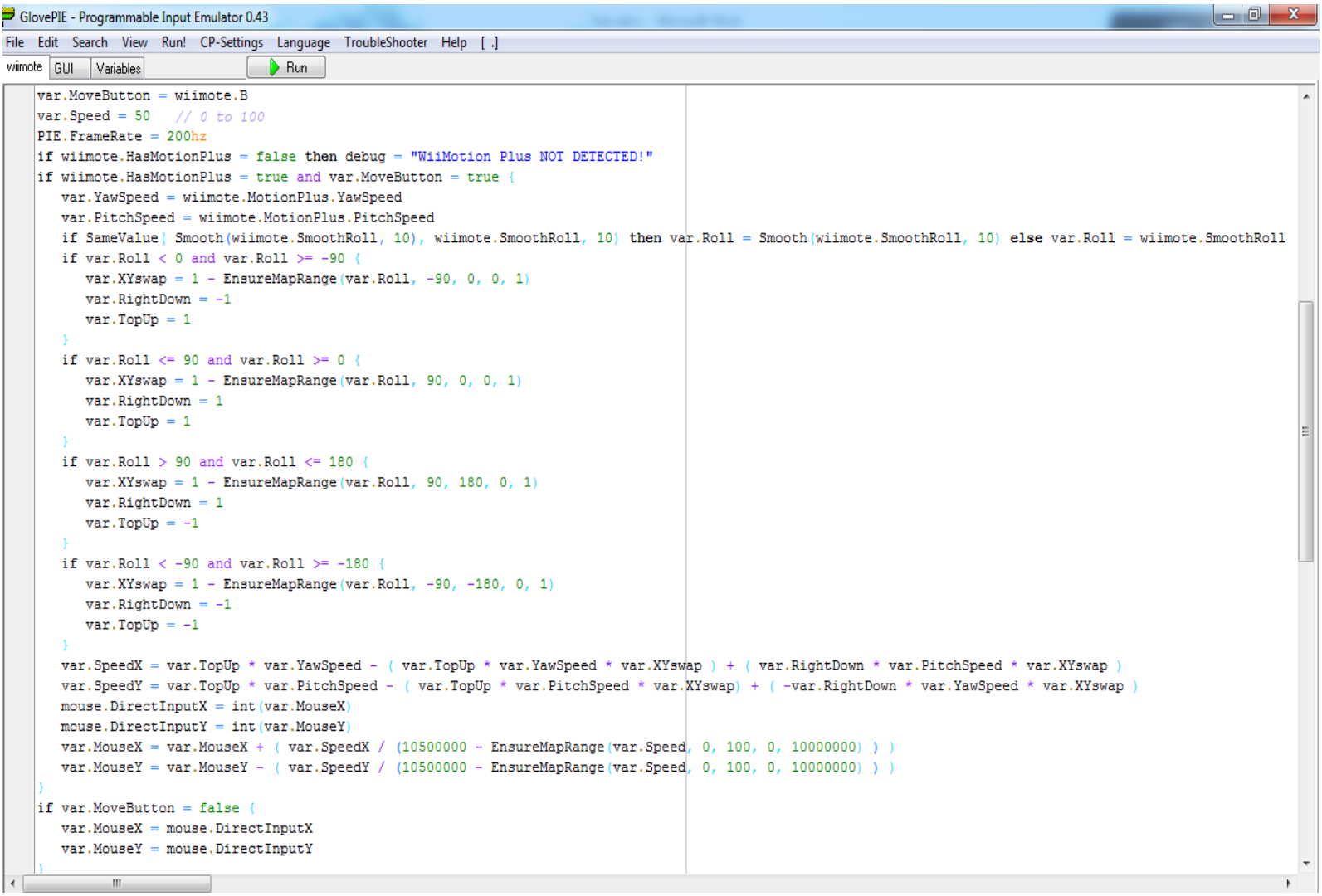
Tomando en cuenta lo anterior, en esta tesis se pretende introducir un nuevo concepto de dispositivo de navegación virtual: el Wiimote. El propósito principal de este dispositivo es permitir la navegación virtual con este sólo dispositivo y de manera inalámbrica, proporcionando así libertad máxima en el movimiento del usuario.

5.2. Análisis y diseño de interfaces gráficas para el Wiimote.

5.2.1. Análisis de la aplicación GlovePIE para programación de Wiimote.

Para conocer los avances en la programación del Wiimote primero se investigaron aplicaciones ya implementadas para funcionar con el Wiimote.

La mejor encontrada durante el desarrollo de este proyecto fue esta, una aplicación que se puede encontrar de forma gratuita en la red y está diseñada para programar dispositivos como el Wiimote con el propósito de personalizar las funciones de los mismos; sin embargo, para realizar esto, la aplicación requiere del conocimiento previo de su lenguaje de programación específico y por ende esto es un obstáculo para el usuario promedio al intentar configurar un dispositivo. A continuación se muestra una ventana de esta aplicación con el respectivo código para simular el movimiento del ratón con el Wiimote:



Además, se requiere de código similar al siguiente para asignar funciones, como los botones del ratón y del teclado, al Wiimote:

```
Key.I = Wiimote.Up
Key.J = Wiimote.Left
Key.K = Wiimote.Down
Key.L = Wiimote.Right
Key.U = Wiimote.One
Key.O = Wiimote.Two
Mouse.RightButton = Wiimote.home
Mouse.WheelUp = Wiimote.plus
Mouse.WheelDown = Wiimote.minus
mouse.LeftButton = wiimote.A

Key.Z = Nunchuk.Z
if nunchuk.C = true {
   WASD = nunchuk.Joy
 }
```

La desventaja de esta aplicación es que el usuario debe conocer el código necesario para modificar la configuración del Wiimote cada vez que requiera ajustarla a sus necesidades.

5.2.2 Análisis del Wiimote como puntero.

Existen dos dispositivos que componen al Wiimote y que permiten realizar un muestreo y procesamiento de señales capaces de indicar su posición.

El primero es el acelerómetro. Este nos permite saber el sentido de la gravedad respecto al Wiimote y por lo tanto es posible saber la posición del mismo.

El segundo es una cámara infrarroja. Esta nos permite detectar fuentes de luz infrarroja y el Wiimote nos proporciona hasta 4 coordenadas de distintas fuentes de luz. Por lo tanto, es posible realizar cálculos para detectar la orientación del Wiimote respecto a estas luces.

A continuación se muestra la vista frontal del Wiimote, donde se puede apreciar la ventana de la cámara (izquierda) y la parte interna del Wiimote donde se encuentra la misma (derecha):

Sin embargo, para esta tesis se pretende utilizar el acelerómetro ya que, a diferencia de la cámara infrarroja, este no requiere de apuntar siempre a un lugar específico. Además, las luces infrarrojas deben estar fijas arriba o debajo de la pantalla, por lo que esta configuración le resta movilidad al Wiimote y provoca incomodidad a algunos usuarios que no encuentran factible utilizar luces infrarrojas cerca de sus pantallas.

5.2.3 Análisis de requerimientos de la interfaz gráfica a diseñar.

La interfaz gráfica de la aplicación que funcionará en conjunto con el Wiimote debe constar de tres tareas básicas para asistir la navegación virtual, con la facilidad para que cualquier usuario pueda manipularla. Esas tres tareas son:

- La capacidad de facilitar la programación de los botones del Wiimote de acuerdo a las necesidades del usuario, utilizando menús que permitan visualizar las distintas opciones a seleccionar para cada botón;

- Poder ajustar la sensibilidad del acelerómetro del Wiimote para permitir un movimiento adecuado en cualquier tamaño de pantalla y para cada tipo de usuario, con la ayuda de una barra de control graduada que permita visualizar el nivel de sensibilidad; y

- La activación del Wiimote desde la misma interfaz gráfica a través de un botón, para iniciar su funcionamiento y que el usuario detecte este suceso con facilidad.

5.2.4 Diseño de la interfaz gráfica de la aplicación.

Para crear una interfaz gráfica fácil de comprender por cualquier usuario y compatible con la todos los sistemas operativos de Windows, en esta tesis se pretende utilizar WinAPI, ya que sus componentes, similares a los de una ventana promedio de Windows, permitirán al usuario sentirse familiarizado con la aplicación del Wiimote, la cual llamaremos **WiimoteAPI** en esta tesis.

El componente principal de la WiimoteAPI será el **combobox**, el cual se muestra a continuación:

Este permitirá desplegar y seleccionar la configuración deseada de una lista de opciones de funciones a asignar a cada botón del Wiimote, ya sean de ratón (click izquierdo por ejemplo) o de teclado (letras, números, entrar, escape, pausa, etc.)

También se utilizará un **botón** para iniciar la simulación de movimiento del ratón con el Wiimote, así como para inicializar las funciones definidas de los botones. Dicho botón tendrá la siguiente apariencia:

Para indicar a qué botón corresponde cada combobox se requieren etiquetas de texto, también para dar mensajes al usuario o incluso para indicar escalas o numeraciones que llegan a ser necesarias en elementos como el **slider**, el cual se muestra a continuación:

El slider permite asignar un valor a cierto parámetro; en este caso es necesario utilizar un slider para definir la sensibilidad del movimiento del acelerómetro del Wiimote para simular el control de sensibilidad de un ratón, y así ajustar adecuadamente este movimiento ya que para cada tamaño de pantalla existe una velocidad de puntero distinta.

5.3. Implementación

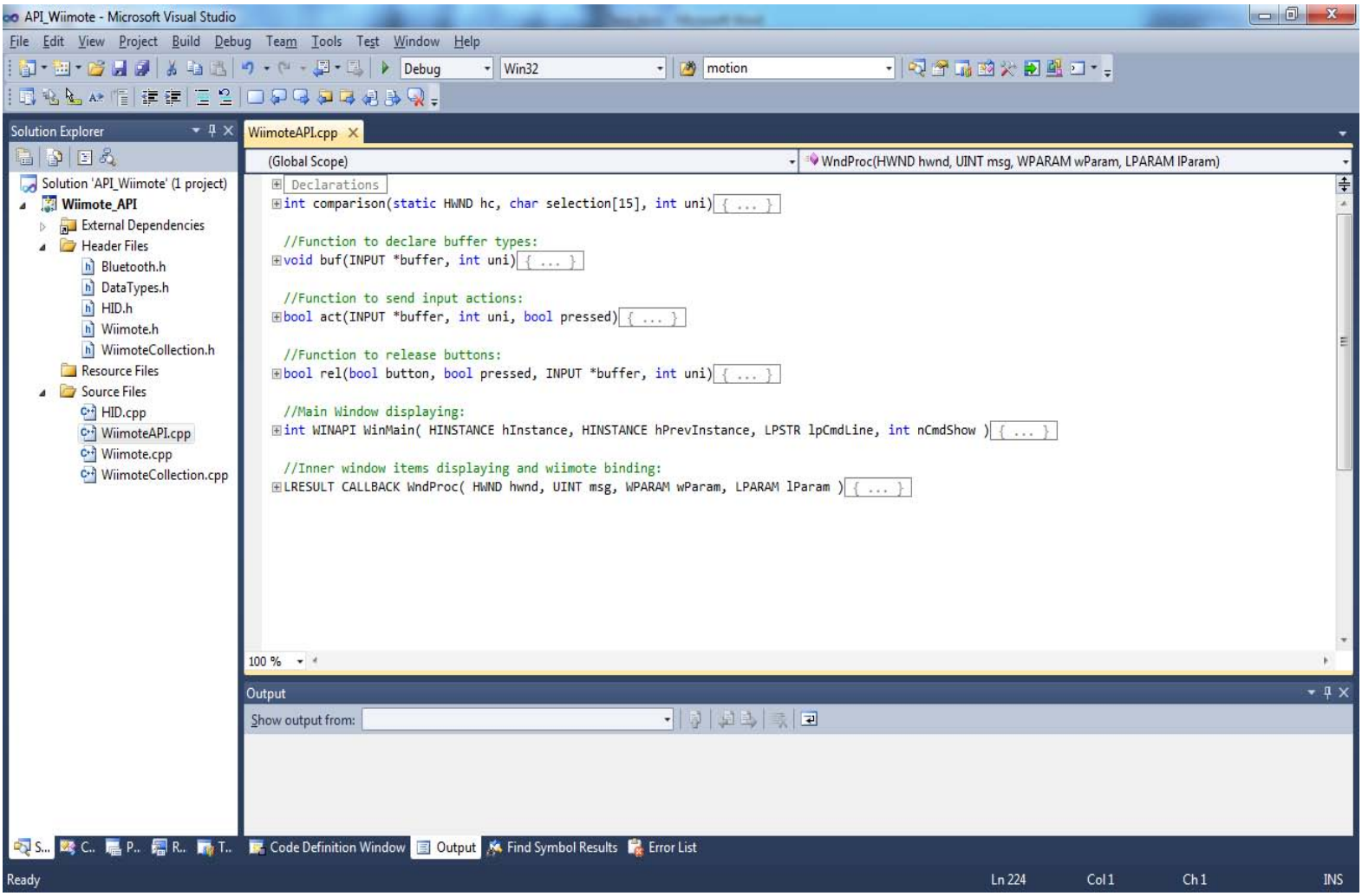
Para desarrollar este proyecto de tesis se utilizó Visual Studio 2010 y se utilizó el SDK para Windows 7 y .NET Framework versión 4, lo cual permite el uso de las diversas funciones de programación de interfaces gráficas a través de WinAPI.

5.3.1. Bibliotecas y cabeceras.

Existen varias bibliotecas y cabeceras que se utilizaron para mejorar la compatibilidad de la aplicación, para poder utilizar estructuras de datos específicas y para llamar funciones y métodos que permiten la comunicación con el Wiimote. Estas son las principales cabeceras y librerías utilizadas por el código principal de la aplicación, WiimoteAPI.cpp:

```
#pragma comment(lib, "setupapi.lib")
#include <iostream>
#include <assert.h>
#include "WiimoteCollection.h"
#include <windows.h>
```

Para tener una idea de las bibliotecas y demás códigos de ayuda en el desarrollo de esta aplicación a continuación se muestra una vista del proyecto en Visual Studio:



La cabecera principal es Wiimote.h, sin embargo el código principal WiimoteAPI.cpp se comunica con esta cabecera a través de WiimoteCollection.h, la cual se muestra a continuación:

```
#include "Wiimote.h"

// Function to convert everything to a void pointer
void* ToVoidPointer( void* Dummy, ... );

// Class for a vector with 'Wiimote' objects
// This class is for multiple wiimote connection
class WiimoteCollection:Wiimote{
public:
	WiimoteCollection();
	~WiimoteCollection();
	void FindAllWiimotes();
	bool WiimoteFound(LPSTR devicePath);
	vector<Wiimote*> wmCollection;

private:
	Wiimote* myWiimote;
};
```

Esta cabecera se encarga de detectar los Wiimotes conectados y disponibles en el equipo, para realizar la conexión con aquellos que desee el programador.

La cabecera Wiimote.h en conjunto con el código Wiimote.cpp, contienen la definición de todos los registros del Wiimote, desde la activación de los LEDs y la detección de los botones presionados, hasta la lectura de los acelerómetros. De aquí se obtuvieron las funciones utilizadas en el desarrollo de la WiimoteAPI, las cuales se muestran en la siguiente tabla:

Función	Objetivo
`WiimoteCollection()`	Define los vectores de los Wiimotes a utilizar.
`FindAllWiimotes()`	Encuentra los Wiimotes reconocidos por Windows.
`wmCollection.begin()`	Inicia el conteo de Wiimotes del sistema.
`Connect()`	Conecta el Wiimote definido por el usuario.
`mWiimoteState.ExtensionType`	Define el tipo de extensión a utilizar en el puerto de expansión del Wiimote.
`mWiimoteState.Extension`	Bandera que indica si existe una extensión conectada en el puerto de expansión del Wiimote.
`~Wiimote()`	Desconecta el Wiimote definido por el usuario.
`SetReportType(InputReport::ExtensionAccel, true)`	Define el tipo de muestreo; en este caso se activan el puerto de expansión y los acelerómetros.
`SetLEDs`	Activa los LEDs.
`mWiimoteState.ButtonState`	Conjunto de banderas que indican el estado de los botones del Wiimote.
`mWiimoteState.AccelState.RawValues`	Valores enteros que indican las posiciones de los ejes X, Y y Z del acelerómetro.
`mWiimoteState.NunchukState`	Conjunto de banderas que indican el estado de los botones C y Z del Nunchuck.
`mWiimoteState.NunchukState.RawJoystick`	Valores enteros que indican la posición, en un plano cartesiano con coordenadas X y Y, del joystick del Nunchuck.

5.3.2. Programación de la WiimoteAPI en lenguaje C++ con WinAPI.

En la programación de esta aplicación, se realizaron comentarios de código en inglés con el propósito de hacer un programa al alcance de cualquiera, internacionalmente si es necesario. El código principal consta de varias partes, la primera es la definición de bibliotecas a utilizar:

```
#pragma comment(lib, "setupapi.lib")
#include <iostream>
#include <assert.h>
#include "WiimoteCollection.h"
#include <windows.h>
```

A continuación se definen las variables globales a utilizar:

```
//Variables for WINAPI:
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
HINSTANCE g_hinst;

//Variables
	POINT xyp;	//Variable to get cursor position.
	int cx = GetSystemMetrics(SM_CXSCREEN);  //Screen width.
	int cy = GetSystemMetrics(SM_CYSCREEN);  //Screen height.
	int pace, sens;		//Mouse sensibility parameters.
	bool pressed1, pressed2, pressed3, pressed4, pressed5, pressed6, pressed7,
	pressed8, pressed9, pressed10, pressed11, pressed12, pressed13, pressed14,
	pressed15, pressed16;		//Pressed buttons flags.
	int x, y;	//Accels calibration coordinates.
	int dx, dy;	//Distance between accels coordinates and cursor position.
	int unic;	//Counter for keys.

//Texts items:
	static HWND hwndCombo, hwndCombo2, hwndCombo3, hwndCombo4, hwndCombo5, hwndCombo6,
	hwndCombo7, hwndCombo8, hwndCombo9, hwndCombo10, hwndCombo11, hwndCombo12, hwndCombo13,
	hwndCombo14, hwndCombo15, hwndCombo16;
	const CHAR *items[] = {"", "LEFT CLICK", "RIGHT CLICK", "LEFT", "UP", "RIGHT", "DOWN",
	"PAGE UP", "PAGE DOWN", "END", "HOME", "ENTER",  "A","B", "C", "D", "E", "F", "G", "H", "I",
	"J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V", "W", "X", "Y", "Z", "0",
	"1", "2", "3", "4", "5", "6", "7", "8", "9", ".", "-", "/", "BACKSPACE", "TAB", "PAUSE",
	"ESC", "SPACEBAR", "PRINT", "INSERT", "DELETE"};
	const CHAR *buttons[] =  { "UP", "DOWN", "LEFT", "RIGHT", "MINUS", "PLUS", "HOME", "ONE",
	"A", "B", "TWO", "C", "Z"};
	char choice[15];
	int B1=0x00, B2=0x00, B3=0x00, B4=0x00, B5=0x00, B6=0x00, B7=0x00, B8=0x00, B9=0x00,
	B10=0x00, B11=0x00, B12=0x00, B13=0x00, B14=0x00, B15=0x00;
	const CHAR *scale[] = {"1","2","3","4","5","6","7","8","9","10"};
	int i,j=0;
	bool nunflag;
	int itemsize=sizeof(items)/4;

//KEYBOARD AND MOUSE ACTIONS DEFINITIONS:
	char empty[15]="", leftk[15]="LEFT", up[15]="UP", rightk[15]="RIGHT", down[15]="DOWN",
	pageup[15]="PAGE UP", pagedown[15]="PAGE DOWN", endk[15]="END", home[15]="HOME",
	enter[15]="ENTER", lclick[15]="LEFT CLICK", rclick[15]="RIGHT CLICK", period[15]=".",
	minusk[15]="-", slash[15]="/", bspace[15]="BACKSPACE", tab[15]="TAB", pause[15]="PAUSE",
	esc[15]="ESC", spacebar[15]="SPACEBAR", print[15]="PRINT", insert[15]="insert",
	deletek[15]="DELETE";
```

Después vienen las declaraciones de funciones.

Esta primera función se encarga de identificar los datos ingresados por el usuario en los distintos campos tipo combobox; cada dato se convierte en su valor UNICODE, para posteriormente ser reconocido por el teclado:

```
//Function to compare defined keys on Wiimote to real keyboard values:
int comparison(static HWND hc, char selection[15], int uni) {

	GetWindowText(hc, selection, 15);
	if (strcmp (empty, selection) == 0){
		uni=0x00;
	}
	else if (strcmp (lclick, selection)== 0){
		uni=0x01;
	}
	else if (strcmp (rclick, selection)== 0){
		uni=0x02;
	}
	else if (strcmp (bspace, selection)== 0){
		uni=0x08;
	}
	else if (strcmp (tab, selection)== 0){
		uni=0x09;
	}
	else if (strcmp (enter, selection) == 0){
		uni=0x0D;
	}
	else if (strcmp (pause, selection) == 0){
		uni=0x13;
	}
	else if (strcmp (esc, selection) == 0){
		uni=0x1b;
	}
	else if (strcmp (spacebar, selection) == 0){
		uni=0x20;
	}
	else if (strcmp (pageup, selection)== 0){
		uni=0x21;
	}
	else if (strcmp (pagedown, selection)== 0){
		uni=0x22;
	}
	else if (strcmp (endk, selection)== 0){
		uni=0x23;
	}
	else if (strcmp (home, selection)== 0){
		uni=0x24;
	}
	else if (strcmp (leftk, selection)== 0){
		uni=0x25;
	}
	else if (strcmp (up, selection)== 0){
		uni=0x26;
	}
	else if (strcmp (rightk, selection)== 0){
		uni=0x27;
	}
```

```
	else if (strcmp (down, selection)== 0){
		uni=0x28;
	}
	else if (strcmp (print, selection)== 0){
		uni=0x2C;
	}

	else if (strcmp (insert, selection)== 0){
		uni=0x2D;
	}
	else if (strcmp (deletek, selection)== 0){
		uni=0x2E;
	}
	else if (strcmp (period, selection)== 0){
		uni=0x6E;
	}
	else if (strcmp (minusk, selection)== 0){
		uni=0x6D;
	}
	else if (strcmp (slash, selection)== 0){
		uni=0x6F;
	}
	else{
		uni = cin.widen(selection[0]);
	}
	return uni;
}
```

A continuación se declara una función para identificar el tipo de buffer a definir, la siguiente función se encarga de clasificar los buffers para los casos posibles de teclado o ratón:

```
//Function to declare buffer types:
void buf(INPUT *buffer, int uni){

	if(uni>0x07){
		buffer->type		= INPUT_KEYBOARD;
		buffer->ki.wVk		= uni;
		buffer->ki.wScan	= 0;
		buffer->ki.time		= 0;
		buffer->ki.dwExtraInfo	= 0;
	}

	else {
		buffer->type		= INPUT_MOUSE;
		buffer->mi.dx		= 0;
		buffer->mi.dy		= 0;
		buffer->mi.mouseData	= 0;
		buffer->mi.time		= 0;
		buffer->mi.dwExtraInfo	= 0;
	}

	return;
}
```

Posteriormente se define la función encargada de enviar el comando ya sea al ratón o al teclado:

```
//Function to send input actions:
bool act(INPUT *buffer, int uni, bool pressed){

	if(uni>0x07 && pressed==false){
		buffer->ki.dwFlags = KEYEVENTF_UNICODE;
		SendInput(1,buffer,sizeof(INPUT));
		pressed=true;
		return pressed;
	}

	else if(uni==0x02 && pressed==false){
		buffer->mi.dwFlags = MOUSEEVENTF_RIGHTDOWN;

		SendInput(1,buffer,sizeof(INPUT));
		pressed=true;
		return pressed;
	}

	else if(uni==0x01 && pressed==false){
		buffer->mi.dwFlags = MOUSEEVENTF_LEFTDOWN;

		SendInput(1,buffer,sizeof(INPUT));
		pressed=true;
		return pressed;
	}
}
```

Después está definida una función que se encarga de "liberar" los botones, ya sean de ratón o de teclado. Esta función es muy importante para realizar una simulación adecuada de las tareas de estos dispositivos periféricos:

```
//Function to release buttons:
bool rel(bool button, bool pressed, INPUT *buffer, int uni){

	if (!button && pressed){
			if(uni==0x01){
				buffer->mi.dwFlags = MOUSEEVENTF_LEFTUP;
				SendInput(1,buffer,sizeof(INPUT));
				pressed=false;
				return pressed;
			}
			else if(uni==0x02){
				buffer->mi.dwFlags = MOUSEEVENTF_RIGHTUP;
				SendInput(1,buffer,sizeof(INPUT));
				pressed=false;
				return pressed;
			}
			else if (uni>0x07){
				pressed=false;
				return pressed;
			}
	}
}
```

Para poder desplegar la ventana de la WiimoteAPI se necesita de una función **principal** que define las propiedades de la misma, como tamaño, nombre, color, etc.

```
//Main Window displaying:
int WINAPI WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance, LPSTR lpCmdLine, int nCmdShow ){

  MSG  msg;
  WNDCLASS wc = {0};
  wc.style         = CS_HREDRAW | CS_VREDRAW;
  wc.cbClsExtra    = 0;
  wc.cbWndExtra    = 0;
  wc.lpszMenuName =  0;
  wc.hIcon         = LoadIcon(NULL, IDI_APPLICATION);
  wc.lpszClassName = "Wiimote";
  wc.hInstance     = hInstance ;
  wc.hbrBackground = GetSysColorBrush(COLOR_3DFACE);
  wc.lpfnWndProc   = WndProc ;
  wc.hCursor       = LoadCursor(0,IDC_ARROW);

  RegisterClass(&wc);
  CreateWindow( wc.lpszClassName, TEXT("WIIMOTE API   -   -   -   -   -   BY ANDREW (DARKCYBORG)"),
                WS_OVERLAPPEDWINDOW | WS_VISIBLE,
                350, 150, 550, 550, 0, 0, hInstance, 0);

       while( GetMessage(&msg, NULL, 0, 0)) {
              TranslateMessage(&msg);
              DispatchMessage(&msg);
       }

       return RegisterClass(&wc);
       return (int) msg.wParam;
}
```

Finalmente, se declara la función que define cada objeto de la interfaz gráfica y que contiene el código del ciclo de muestreo de las señales de entrada del Wiimote.

La primera parte de esta función consiste en la declaración de unas cuantas variables locales, como la bandera del Nunchuck, que indica si está conectado, y algunas variables de objetos de la interfaz gráfica.

```
//Inner window items displaying and wiimote binding:
LRESULT CALLBACK WndProc( HWND hwnd, UINT msg, WPARAM wParam, LPARAM lParam )
{
  nunflag=false;

  HWND hwndsens;
  HWND hwndLeftLabel;
  HWND hwndRightLabel;
```

En la segunda parte se inicia una conexión rápida del Wiimote para verificar que el Nunchuck esté conectado al puerto de expansión del Wiimote y de ser así se guarda el valor de la bandera de conexión como verdadero:

```
switch(msg)
{
	case WM_CREATE:
	{
		//Nunchuk connection verification:
		WiimoteCollection* wmColl = new WiimoteCollection();
		wmColl->FindAllWiimotes();
		vector<Wiimote*>::const_iterator wmCounter;
		wmCounter = wmColl->wmCollection.begin();
		wmColl->wmCollection[0]->Connect();
		wmColl->wmCollection[0]->mWiimoteState.ExtensionType=Nunchuk;

		if(wmColl->wmCollection[0]->mWiimoteState.Extension){
			nunflag=true;
		}
```

La tercera parte consiste en crear las etiquetas, correspondientes a los distintos objetos tipo combobox, de cada botón del Nunchuck.

```
		if(nunflag){
			CreateWindow(TEXT("STATIC"), TEXT("- NUNCHUK BUTTONS BINDING -"),
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    230, 8, 250 25, hwnd, NULL, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[11],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 43, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[12],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 83, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[0],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 123, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[1],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 163, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[2],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 203, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);

			CreateWindow(TEXT("STATIC"), buttons[3],
		    WS_CHILD | WS_VISIBLE | SS_LEFT,
		    250, 243, 50, 30,
		    hwnd, (HMENU) 1, NULL, NULL);
```

La cuarta parte consta de la creación de los objetos tipo combobox para definir los botones del Nunchuck; en caso de no estar conectado, se crea un mensaje de aviso al usuario para conectarlo, aunque es opcional. Y se termina la conexión rápida al Wiimote la cual se realizó en fracciones de segundo:

```
                hwndCombo10 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 40, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo11 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 80, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo12 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 120, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo13 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 160, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo14 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 200, 110, 300, hwnd, NULL, g_hinst, NULL);

                hwndCombo15 = CreateWindow(TEXT("combobox"), NULL,
            WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
            300, 240, 110, 300, hwnd, NULL, g_hinst, NULL);
      }

      else{
             CreateWindow(TEXT("STATIC"), TEXT("PLEASE, CONNECT A NUNCHUK!"),
    WS_CHILD | WS_VISIBLE | SS_LEFT,
    270, 38, 130, 30, hwnd, NULL, NULL, NULL);
      }

      wmColl->wmCollection[0]->~Wiimote();
      delete wmColl;
```

Cabe destacar que esta conexión no se puede mantener activa mientras el usuario define las funciones de los botones del Wiimote, ya que el despliegue de la interfaz gráfica y el muestreo del Wiimote son dos procesos cíclicos, que no pueden trabajar en paralelo.

La quinta parte se encarga de crear las etiquetas de los botones del Wiimote para reconocer sus respectivos objetos tipo combobox:

```
                CreateWindow(TEXT("STATIC"), TEXT("- BUTTONS BINDING -"),
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    20, 8, 180, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[0],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 43, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[1],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 83, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[2],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 123, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[3],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 163, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[4],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 203, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[5],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 243, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[6],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 283, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[7],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 323, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[8],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 363, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[9],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 403, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);

                CreateWindow(TEXT("STATIC"), buttons[10],
                    WS_CHILD | WS_VISIBLE | SS_LEFT,
                    10, 443, 60, 30,
                    hwnd, (HMENU) 1, NULL, NULL);
```

En la sexta parte se crean los objetos tipo combobox de los botones del Wiimote:

```
hwndCombo = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 40, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo2 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 80, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo3 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 120, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo4 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 160, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo5 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 200, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo6 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 240, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo7 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 280, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo8 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 320, 110, 300, hwnd, NULL, g_hinst, NULL);

hwndCombo9 = CreateWindow(TEXT("combobox"), NULL,
    WS_CHILD | WS_VISIBLE | WS_VSCROLL | CBS_DROPDOWN,
    75, 360, 110, 300, hwnd, NULL, g_hinst, NULL);
```

La séptima parte de esta función se encarga de crear varios objetos: las etiquetas de los dos botones con funciones fijas, las cuales son “Activar mouse” y “Detener Wiimote”; también la etiqueta del botón de inicio de funcionamiento del Wiimote; y las etiquetas correspondientes a la barra de ajuste de sensibilidad del Wiimote.

```
  CreateWindow(TEXT("static"), TEXT("ACTIVATE MOUSE"),
WS_CHILD | WS_VISIBLE | SS_LEFT,
75, 402, 150, 25, hwnd, NULL, NULL, NULL);

  CreateWindow(TEXT("static"), TEXT("STOP WIIMOTE"),
WS_CHILD | WS_VISIBLE | SS_LEFT,
75, 442, 150, 25, hwnd, NULL, NULL, NULL);

  CreateWindow(TEXT("button"), TEXT("START WIIMOTE!"),
      WS_VISIBLE | WS_CHILD | SS_CENTER,
      260, 402, 150, 50, hwnd, (HMENU)1, NULL, NULL);

  hwndCombo16 = CreateWindowEx(
  0, TRACKBAR_CLASS, "Trackbar Control",
  WS_CHILD | WS_VISIBLE |
  TBS_AUTOTICKS | TBS_ENABLESELRANGE,
  280, 320, 100, 30,
  hwnd, (HMENU) 3, NULL, NULL);

  hwndsens = CreateWindow("STATIC", "MOUSE SENSIBILITY:",
             WS_CHILD | WS_VISIBLE,
             260, 300, 150, 25,
             hwnd, (HMENU)1, NULL, NULL);

  hwndLeftLabel = CreateWindow("STATIC", "0",
             WS_CHILD | WS_VISIBLE,
             0, 0, 15, 15,
             hwnd, (HMENU)1, NULL, NULL);

  hwndRightLabel = CreateWindow("STATIC", "10",
             WS_CHILD | WS_VISIBLE,
             0, 0, 15, 15,
             hwnd, (HMENU)2, NULL, NULL);
```

La octava sección se encarga de crear los contenidos de cada objeto tipo combobox, para que el usuario seleccione de esta lista de funciones la que desee asignar a cada botón:

```
        for ( i = 0; i < itemsize; i++ ) {
                        SendMessage(hwndCombo, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo2, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo3, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo4, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo5, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo6, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo7, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo8, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo9, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo10, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo11, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo12, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo13, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo14, CB_ADDSTRING, 0, (LPARAM) items[i]);
                        SendMessage(hwndCombo15, CB_ADDSTRING, 0, (LPARAM) items[i]);
         }

               SendMessage(hwndCombo16, TBM_SETRANGE,  TRUE,  MAKELONG(0, 10));
               SendMessage(hwndCombo16, TBM_SETPAGESIZE, 0,  1);
               SendMessage(hwndCombo16, TBM_SETTICFREQ, 1, 0);
               SendMessage(hwndCombo16, TBM_SETPOS, 5, 5);
               SendMessage(hwndCombo16, TBM_SETBUDDY, TRUE, (LPARAM) hwndLeftLabel);
               SendMessage(hwndCombo16, TBM_SETBUDDY, FALSE, (LPARAM) hwndRightLabel);


               SendMessage(hwndCombo, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo2, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo3, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo4, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo5, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo6, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo7, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo8, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo9, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo10, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo11, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo12, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo13, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo14, CB_SETCURSEL, 0, 0);
               SendMessage(hwndCombo15, CB_SETCURSEL, 0, 0);
        }
```

La novena parte inicia con la activación del botón de inicio del Wiimote. Primero se desactiva el botón para que el usuario identifique el inicio del funcionamiento del Wiimote. Posteriormente se hacen las comparaciones de las funciones para cada botón del Wiimote que asignó el usuario y así identificar de qué tipo son. Después se activa la conexión del Wiimote y se encienden dos LEDs para hacer aún más evidente que el Wiimote está activado y listo para usarse:

```
        case WM_COMMAND:
        {

               if (LOWORD(wParam) == 1) {

                      CreateWindow(TEXT("button"), TEXT("START WIIMOTE!"),
                   WS_VISIBLE | WS_CHILD | SS_CENTER,
                   260, 402, 150, 50, hwnd, (HMENU)1, NULL, NULL);

                      B1=comparison(hwndCombo,choice,B1);
                      B2=comparison(hwndCombo2,choice,B2);
                      B3=comparison(hwndCombo3,choice,B3);
                      B4=comparison(hwndCombo4,choice,B4);
                      B5=comparison(hwndCombo5,choice,B5);
                      B6=comparison(hwndCombo6,choice,B6);
                      B7=comparison(hwndCombo7,choice,B7);
                      B8=comparison(hwndCombo8,choice,B8);
                      B9=comparison(hwndCombo9,choice,B9);
                      B10=comparison(hwndCombo10,choice,B10);
                      B11=comparison(hwndCombo11,choice,B11);
                      B12=comparison(hwndCombo12,choice,B12);
                      B13=comparison(hwndCombo13,choice,B13);
                      B14=comparison(hwndCombo14,choice,B14);
                      B15=comparison(hwndCombo15,choice,B15);

               WiimoteCollection* wmColl = new WiimoteCollection();
               wmColl->FindAllWiimotes();
               vector<Wiimote*>::const_iterator wmCounter;
               wmCounter = wmColl->wmCollection.begin();

               wmColl->wmCollection[0]->Connect();
               wmColl->wmCollection[0]->SetReportType(InputReport::ExtensionAccel, true);
               wmColl->wmCollection[0]->SetLEDs(true, false, false, true);
```

Posteriormente, la décima parte consta de la creación de los buffers para cada tipo de acción correspondiente a los botones del Wiimote, cada buffer representa una señal de entrada al sistema, que puede ser de teclado o de ratón para este caso. Después se configura el puerto de expansión del Wiimote para reconocer al Nunchuck. A continuación se definen algunas variables para inicializar el puntero del ratón y se obtiene el valor de la sensibilidad definido por el usuario desde la interfaz gráfica:

```
        INPUT *buffer1       = new INPUT;
        buf(buffer1,B1);

        INPUT *buffer2       = new INPUT;
        buf(buffer2,B2);

        INPUT *buffer3       = new INPUT;
        buf(buffer3,B3);

        INPUT *buffer4       = new INPUT;
        buf(buffer4,B4);

        INPUT *buffer5       = new INPUT;
        buf(buffer5,B5);

        INPUT *buffer6       = new INPUT;
        buf(buffer6,B6);

        INPUT *buffer7       = new INPUT;
        buf(buffer7,B7);

        INPUT *buffer8       = new INPUT;
        buf(buffer8,B8);

        INPUT *buffer9       = new INPUT;
        buf(buffer9,B9);

        INPUT *buffer10      = new INPUT;
        buf(buffer10,B10);

        INPUT *buffer11      = new INPUT;
        buf(buffer11,B11);

        INPUT *buffer12      = new INPUT;
        buf(buffer12,B12);

        INPUT *buffer13      = new INPUT;
        buf(buffer13,B13);

        INPUT *buffer14      = new INPUT;
        buf(buffer14,B14);

        INPUT *buffer15      = new INPUT;
        buf(buffer15,B15);

        //Nunchuck activation:
        wmColl->wmCollection[0]->mWiimoteState.ExtensionType=Nunchuk;

        //Variables to initialize mouse movement:
        SetCursorPos(cx/2,cy/2);
        GetCursorPos(&xyp);
        x=120,y=120;

        //Mouse sensibility parameters:
        pace=SendMessage(hwndCombo16, TBM_GETPOS, 0, 0);
        sens=1;
```

La décima primera parte consiste en iniciar el ciclo de muestreo del acelerómetro del Wiimote:

```
while(!wmColl->wmCollection[0]->mWiimoteState.ButtonState.Two()) {

		Sleep(2*(10-pace));  //Sensibility assignment.
		dx=0,dy=0;

		//Mouse development:
		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.X<x-sens)){
			dx=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.X-x);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}

		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.X>x+sens)){
			dx=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.X-x);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}

		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.Y<y-sens)){
			dy=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.Y-y);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}

		if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.B() && (wmColl->wmCollection[0]->
		mWiimoteState.AccelState.RawValues.Y>y+sens)){
			dy=(wmColl->wmCollection[0]->mWiimoteState.AccelState.RawValues.Y-y);
			SetCursorPos(xyp.x+dx,xyp.y+dy);
		}
```

En la décima segunda parte se realizan las acciones si están presionados los botones del Wiimote:

```
//Functions binding:
if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Up()){

	pressed1=act(buffer1,B1,pressed1);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Down()){

	pressed2=act(buffer2,B2,pressed2);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Left()){

	pressed3=act(buffer3,B3,pressed3);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Right()){

	pressed4=act(buffer4,B4,pressed4);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Minus()){

	pressed5=act(buffer5,B5,pressed5);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Plus()){

	pressed6=act(buffer6,B6,pressed6);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Home()){

	pressed7=act(buffer7,B7,pressed7);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.One()){
	pressed8=act(buffer8,B8,pressed8);
}

if(wmColl->wmCollection[0]->mWiimoteState.ButtonState.A()){

	pressed9=act(buffer9,B9,pressed9);
}

if(wmColl->wmCollection[0]->mWiimoteState.NunchukState.C){

	pressed10=act(buffer10,B10,pressed10);
}

if(wmColl->wmCollection[0]->mWiimoteState.NunchukState.Z){

	pressed11=act(buffer11,B11,pressed11);
}
```

La décima tercera parte se encarga de realizar el mapeo del movimiento del joystick del Nunchuck para poder asignarle cuatro botones, los cuales serían arriba, abajo, izquierda y derecha. Después se liberan los botones que fueron presionados, tanto del Wiimote como del Nunchuck:

```
	//Nunchuck direction mapping:
		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y>
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y>165) ){
			pressed12=act(buffer12,B12,pressed12);
		}

		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y<
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y<105) ){
			pressed13=act(buffer13,B13,pressed13);
		}

		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X<
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X<105) ){
			pressed14=act(buffer14,B14,pressed14);
		}

		if( (wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X>
		abs(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y)) &&
		(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X>165) ){
			pressed15=act(buffer15,B15,pressed15);
		}

		//Buttons release:
		pressed1=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Up(),pressed1,buffer1,B1);
		pressed2=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Down(),pressed2,buffer2,B2);
		pressed3=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Left(),pressed3,buffer3,B3);
		pressed4=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Right(),pressed4,buffer4,B4);
		pressed5=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Minus(),pressed5,buffer5,B5);
		pressed6=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Plus(),pressed6,buffer6,B6);
		pressed7=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.Home(),pressed7,buffer7,B7);
		pressed8=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.One(),pressed8,buffer8,B8);
		pressed9=rel(wmColl->wmCollection[0]->mWiimoteState.ButtonState.A(),pressed9,buffer9,B9);
		pressed10=rel(wmColl->wmCollection[0]->mWiimoteState.NunchukState.C,pressed10,buffer10,B10);
		pressed11=rel(wmColl->wmCollection[0]->mWiimoteState.NunchukState.Z,pressed11,buffer11,B11);

		if(	(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y<165)
			&&
			(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.Y>105)
			&&
			(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X>105)
			&&
			(wmColl->wmCollection[0]->mWiimoteState.NunchukState.RawJoystick.X<165)
			){
				pressed12=rel(false,pressed12,buffer12,B12);
				pressed13=rel(false,pressed13,buffer13,B13);
				pressed14=rel(false,pressed14,buffer14,B14);
				pressed15=rel(false,pressed15,buffer15,B15);
		}
```

Y finalmente, se actualiza la posición del puntero. Una vez que el usuario presiona el botón “dos” del Wiimote, se finaliza el ciclo de muestreo y se eliminan los buffers creados para que el usuario pueda volver a definirlos si lo necesita; a continuación se desactiva el Wiimote y la WiimoteAPI vuelve a estar disponible. En caso de que el usuario de click en el botón de salir, como en cualquier otra ventana, se terminará la aplicación.

```
        GetCursorPos(&xyp);

        }

        delete buffer1;
        delete buffer2;
        delete buffer3;
        delete buffer4;
        delete buffer5;
        delete buffer6;
        delete buffer7;
        delete buffer8;
        delete buffer9;
        delete buffer10;
        delete buffer11;
        delete buffer12;
        delete buffer13;
        delete buffer14;
        delete buffer15;

        wmColl->wmCollection[0]->~Wiimote();

        delete wmColl;

                }

                break;
      }

         case WM_DESTROY:{
                PostQuitMessage(0);
          break;
         }

  }

  return DefWindowProc(hwnd, msg, wParam, lParam);
}
```

VI. Análisis y metodología empleada

La metodología empleada en el desarrollo de esta aplicación se aproxima en su mayor parte a la forma básica del **modelo de cascada**, con los siguientes procedimientos:

Análisis de requisitos. En esta etapa se analizaron y definieron las funciones características de la WiimoteAPI como una herramienta para asistir la navegación virtual. Se decidió entre las opciones de utilizar la cámara infrarroja o el acelerómetro del Wiimote como motor para simular el movimiento del puntero, tomando en cuenta las ventajas y desventajas de cada una.

Diseño. Aquí de eligió el tipo de interfaz y los elementos a utilizar dentro de la misma para proporcionar una aplicación accesible para el usuario. Asimismo, se definió el tipo de configuración que utilizaría el Wiimote tanto como dispositivo inalámbrico como simulador de teclado y ratón.

Implementación. Esta etapa se desarrolló en dos partes, primero se programó el código que permitió la conexión y el funcionamiento del Wiimote y posteriormente se programó la interfaz gráfica de la WiimoteAPI.

Pruebas. Se realizaron pruebas de funcionamiento en distintas versiones de Windows y en distintos tamaños de pantallas, ya que el movimiento y sensibilidad que refleja el acelerómetro del Wiimote depende de estos dispositivos gráficos.

Mantenimiento. Se realizaron mejoras en la sensibilidad y en la interfaz gráfica para facilitar más el uso del Wiimote como puntero y la comprensión de la WiimoteAPI.

VII. Participación profesional

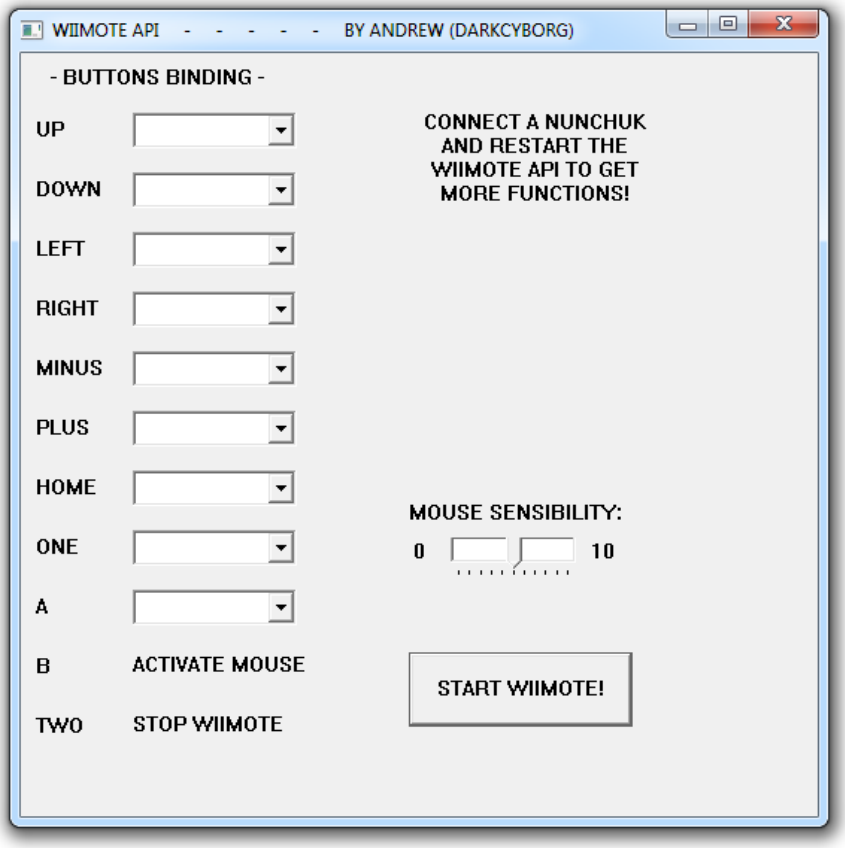
En busca de una opción para titularme, consideré la oportunidad de trabajar como tesista con el M.I. Rodrigo Tintor. Él me sugirió desarrollar alguna aplicación que permitiera mejorar la navegación virtual con ayuda del Wiimote. Sin embargo, yo no conocía plenamente su uso y sus características antes de definir claramente este proyecto y por ello investigué los avances existentes en la programación del mismo a sugerencia de Rodrigo.

Encontré muchas aplicaciones que registraban el estado de los botones, el acelerómetro y la cámara infrarroja del Wiimote, pero ninguna que permitía configurarlo o no proporcionaba código fuente. Fue entonces que decidí desarrollar esta aplicación innovadora.

Al principio consideré buscar un compañero para así dividir el trabajo de la programación, pero debido a que no hubo interesados en esta tesis, yo participé como único programador, tanto en el desarrollo gráfico como funcional de la aplicación WiimoteAPI.

VIII. Resultados y aportaciones

Como resultado de esta investigación y desarrollo se obtuvo la aplicación WiimoteAPI que consta de una interfaz gráfica como se muestra a continuación:



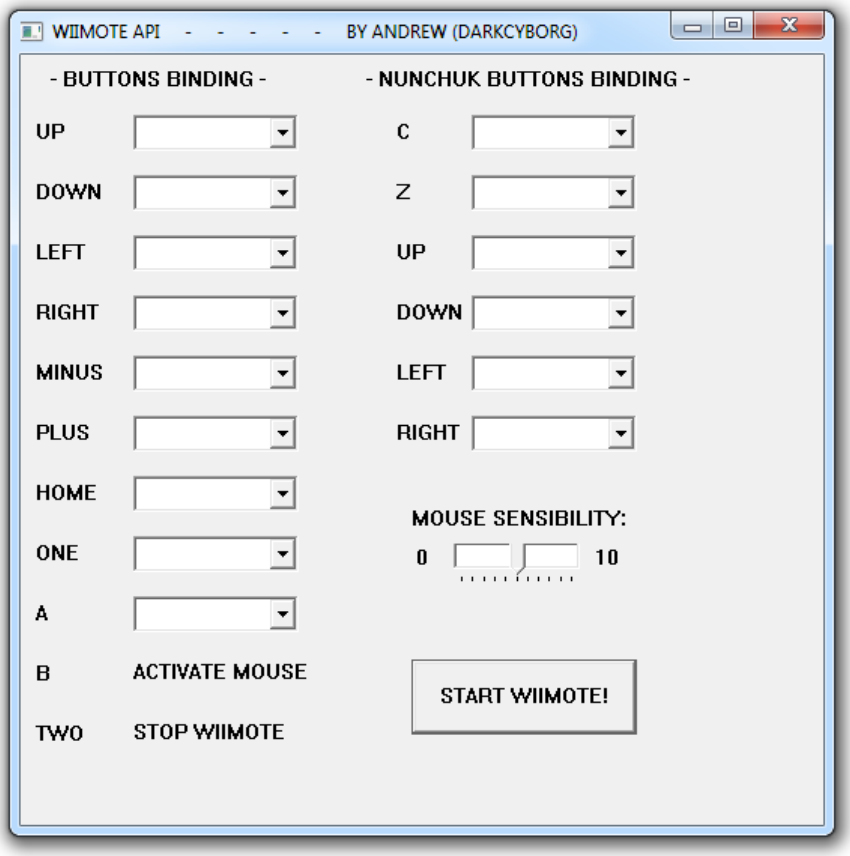
El conjunto de objetos “combobox” del lado izquierdo permite asignar una función a cada botón del Wiimote, ya sea de ratón o de teclado. Los dos últimos botones se indican solamente para que el usuario reconozca las opciones de activación del puntero y de detener el muestreo del Wiimote.

En la parte superior derecha se muestra un mensaje que sugiere al usuario conectar un Nunchuck, si es que no está presente en el puerto de expansión del Wiimote, para así incrementar la funcionalidad de la aplicación.

Posteriormente se tiene un “slider” que permite ajustar la sensibilidad del Wiimote, el cual conviene ajustar de acuerdo al tamaño de la pantalla que se utiliza. Mientras mayor sea el tamaño de la pantalla, mayor debe ser el grado de sensibilidad. Se recomienda iniciar la WiimoteAPI después de conectar algún tipo de proyector para evitar que se presente una configuración de sensibilidad incorrecta.

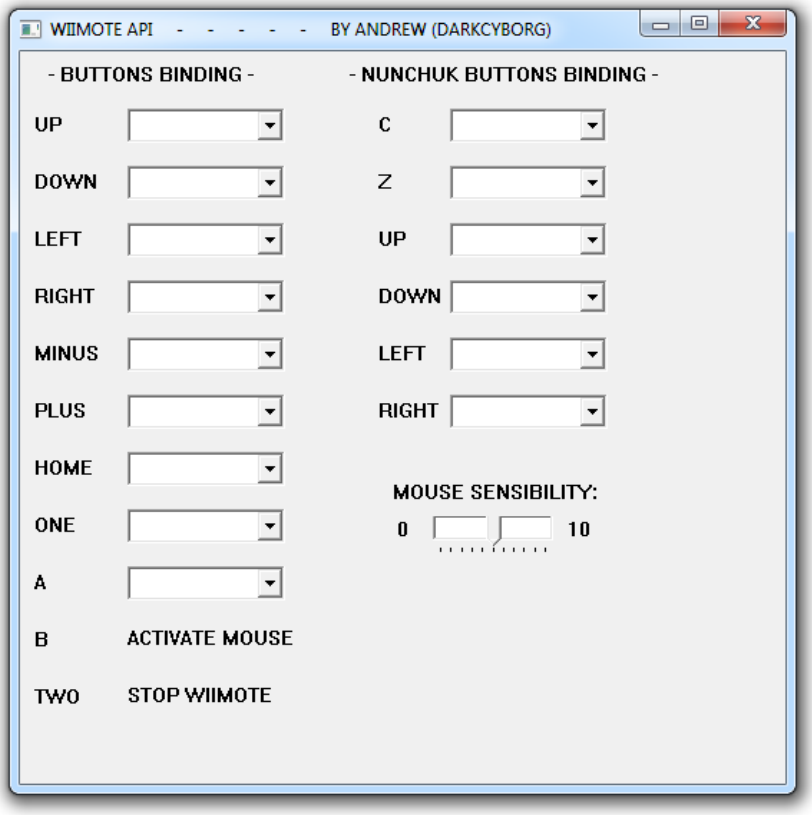
Por último se tiene el botón de inicio del Wiimote, el cual, al ser presionado, permite realizar la asignación de funciones al Wiimote e iniciar el muestreo de su acelerómetro y sus botones para reflejar las acciones definidas por el usuario en la WiimoteAPI.

En caso de contar con un Nunchuck y conectarlo antes de iniciar la aplicación, se puede visualizar un conjunto de opciones agregadas a la interfaz de la WiimoteAPI para configurar el joystick y los botones C y Z de esta extensión, como se muestra en la siguiente imagen:



Cada elemento combobox permite definir una de las siguientes funciones: click izquierdo y click derecho, movimiento de puntero, teclas correspondientes a las letras y números y teclas especiales como ENTER, ARRIBA, ABAJO, IZQUIERDA, DERECHA, AVANZAR y RETROCEDER PÁGINA, INICIO, FIN, PUNTO, GUIÓN, DIAGONAL, RETORNO, TABULADOR, PAUSA, ESCAPE, ESPACIO, IMPRIMIR PANTALLA, INSERTAR y SUPRIMIR.

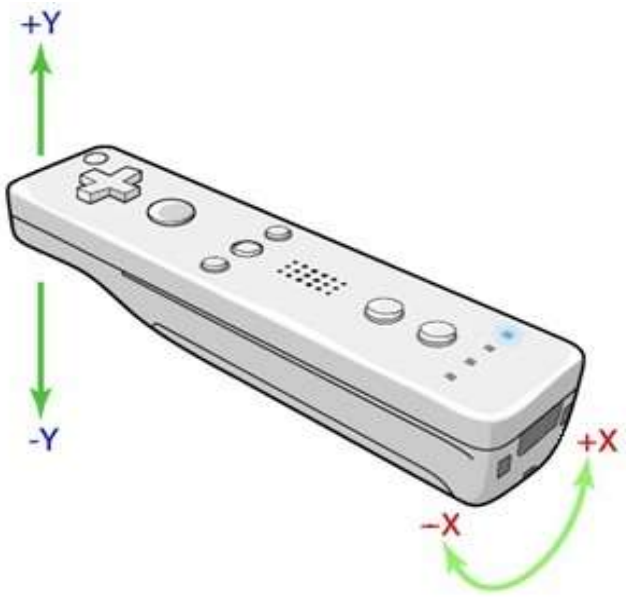
Una vez que se inicia el Wiimote dando click en el botón “START WIIMOTE”, éste se inhabilita y desaparece, además de que los LEDs 1 y 4 del Wiimote se activan para que el usuario identifique con facilidad que puede empezar a utilizar el Wiimote. La Interfaz entonces se muestra así:



Para poder reflejar el movimiento del Wiimote como puntero de ratón en pantalla, se requiere mantener presionado el botón **B**. Esta configuración fija del botón B se diseñó así con el propósito de evitar que el usuario olvide asignar la función de movimiento de puntero.

Asimismo, al tener un botón de activación es posible suspender y activar dicho movimiento cada vez que lo requiera el usuario.

El movimiento del Wiimote se debe realizar de acuerdo a las flechas de la siguiente figura:



El movimiento horizontal se realiza girando el Wiimote sobre su propio eje, mientras que el vertical se realiza subiendo y bajando la parte superior respecto a la inferior.

Finalmente, la aportación de esta tesis con el Wiimote y la WiimoteAPI es una herramienta inalámbrica y de gran compatibilidad que permite realizar tareas como las de un ratón y un teclado. Por ejemplo, manipular la computadora, desde el sistema operativo hasta los programas instalados y los distintos archivos, así como asistir en aplicaciones de navegación en ambientes virtuales en varios escenarios.

IX. Conclusiones

La Wiimote API es una aplicación innovadora que no sólo permite manipular una computadora de manera inalámbrica, también permite impulsar el trabajo multidisciplinario en actividades de enseñanza e investigación, al ser una herramienta de realidad virtual, cuyo objetivo es facilitar la visualización, simulación y manipulación de ambientes gráficos que enriquecen el conocimiento de alumnos, profesores, investigadores y profesionistas, mejorando la comprensión y análisis de temas complejos que requieren de recursos visuales.

Por ejemplo, hoy en día el quirófano de cualquier hospital requiere de diversos dispositivos que intentan evitar la intervención prolongada y demasiado invasiva de un paciente, por lo que actualmente la tendencia a utilizar equipo médico que utiliza sistemas de navegación virtual para así operar desde el exterior del cuerpo y reducir el riesgo del paciente, va creciendo con el tiempo. Si el cirujano tiene la preparación adecuada para realizar este tipo de intervenciones se lograría disminuir u incluso omitir el uso, demasiado invasivo, de los instrumentos quirúrgicos comunes y si se implementa una herramienta como el Wiimote en conjunto con la WiimoteAPI para asistir el aprendizaje de estas técnicas de cirugía en los médicos, entonces se tendrían mejores cirujanos y por ende una mejor recuperación, calidad de salud y de vida en los pacientes.

Asimismo, este proyecto de tesis también sirve como una herramienta de simulación de sistemas de navegación utilizados en distintos tipos de salas virtuales como el observatorio Ixtli, cuyo objetivo es el de aumentar la sensación de realismo, permitiendo así aprender y conservar el conocimiento adquirido en la sala de manera más eficiente y por más tiempo, y en cualquier área de investigación que cuente con estos medios visuales.

Las principales ventajas de la WiimoteAPI son: mayor facilidad de configuración a comparación de otras aplicaciones, capacidad de asignar funciones distintas del teclado y el ratón, independencia de conexión y lugar de funcionamiento gracias a su conectividad inalámbrica por Bluetooth, así como bajo precio a comparación de otros dispositivos inalámbricos.

El único límite para el Wiimote y la WiimoteAPI son aquellas aplicaciones en las que se requiere el uso del teclado como dispositivo periférico principal, como es el caso de editores de texto, aunque esta tesis no tiene como objetivo reemplazar en su totalidad a un teclado.

X. Referencias

1. Analog Devices. **Small, Low Power, 3-Axis ±3 g *i*MEMS® Accelerometer, ADXL330**. www.analog.com [En línea]. Última modificación: Septiembre-2006. <http://www.analog.com/static/imported-files/data_sheets/ADXL330.pdf> [Consulta: 13-Octubre-2010].

2. Awio Web Services LLC. **Web Browser Market Share**. W3Counter [En línea]. Última modificación: 30-Abril-2011 <http://www.w3counter.com/globalstats.php> [Consulta: 2-Mayo-2011].

3. Bodnar, Jan. **The Winapi (C Win32 API, No MFC) tutorial**. ZetCode [En línea]. Última modificación: 28-Julio-2007. <http://zetcode.com/tutorials/winapi> [Consulta: 15-Enero-2011].

4. Brian, Peek. **Managed Wiimote Library**. Wii Yourself [En línea]. Última modificación: 31-Marzo-2010. <http://wiiyourself.gl.tter.org> [Consulta: 10-Diciembre-2010].

5. Gad-el-Hak, Mohamed. **The MEMS Handbook**. University of Notre Dame, CRC PRESS 2002, p. 887.

6. Huang, Albert. **Bluetooth for Programmers**. GNU Free Documentation License [En línea]. Última modificación: 2008. < http://people.csail.mit.edu/albert/bluez-intro> [Consulta: 21-Noviembre-2010].

7. James, Dick. **Two different approaches to integrated MEMS**. Chipworks [En línea]. Última modificación: 7-Mayo-2008. <http://www.electroiq.com/index/display/semiconductors-article-display/328028/articles/solid-state-technology/chip-forensics/2008/05/two-different-approaches-to-integrated-mems.html> [Consulta: 7-Agosto-2010].

8. Kenner, Carl. **GlovePIE**. GlovePIE Home Page [En línea]. Última modificación: 21-Enero-2010. <http://glovepie.org/glovepie_download.php> [Consulta: 1-Noviembre-2010].

9. Lewis, S., et al. **Integrated Sensor and Electronics Processing for >10^8. iMEMS. Inertial Measurement Unit Components**. IEDM Technical Digest 2003, p. 949.

10. Microsoft. **Windows SDK for Windows 7 and .NET Framework 3.5 SP1**. MSDN [En línea]. Última modificación: Agosto-2009. <http://msdn.microsoft.com/en-us/windows/bb980924> [Consulta: 29-Diciembre-2010].

11. Microsoft. **Visual Studio Professional 2010**. Microsoft [En línea].
Última modificación: 2010.
<http://msdn34.e-academy.com/elms/Storefront/Home.aspx?campus=ufdi_comp>
[Consulta: 11-Julio-2010].

12. Sommerville, Ian. **Ingeniería del Software**.
Pearson - Addison Wesley, 7ª edición, 2005, p 62.

13. Theoretical and Computational Biophysics Group. **Visual Molecular Dynamics**. UIUC [En línea].
Última modificación: 14-Marzo-2011.
<http://www.ks.uiuc.edu/Research/vmd>
[Consulta: 27-Octubre-2010].

14. Wiili. **Wiimote**. GNU Free Documentation License [En línea].
Última modificación: 5-Diciembre-2006.
<http://homepage.mac.com/ianrickard/wiimote/wiili_wimote.html>
[Consulta: 10-Septiembre-2010].